Release Date: August, 2016

Updates:

Database Programming with PL/SQL

10-2
Managing Package Concepts

ORACLE ACADEMY

Objectives

This lesson covers the following objectives:

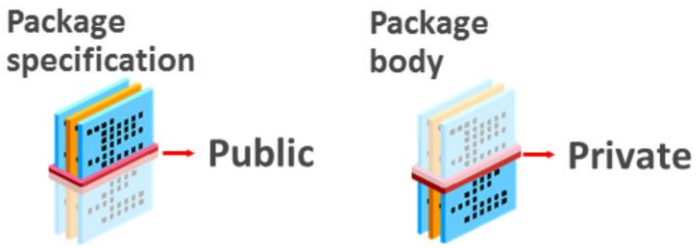
- Explain the difference between public and private package constructs
- Designate a package construct as either public or private
- Specify the appropriate syntax to drop packages
- Identify views in the Data Dictionary that manage packages
- Identify guidelines for using packages

Purpose

- 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.

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.

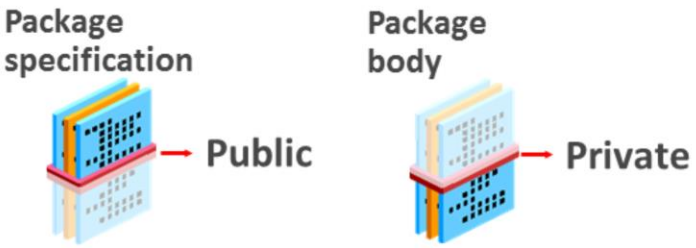


Any package component can be either public or private. This includes not only procedures and functions, but also variables, constants, cursor declarations, and exception declarations.

The package can hide the implementation of a private subprogram so that only the package (not your application) is affected if the code of that private subprogram is changed. Also, by hiding implementation details from users, you protect the integrity of the package.

Components of a PL/SQL 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.

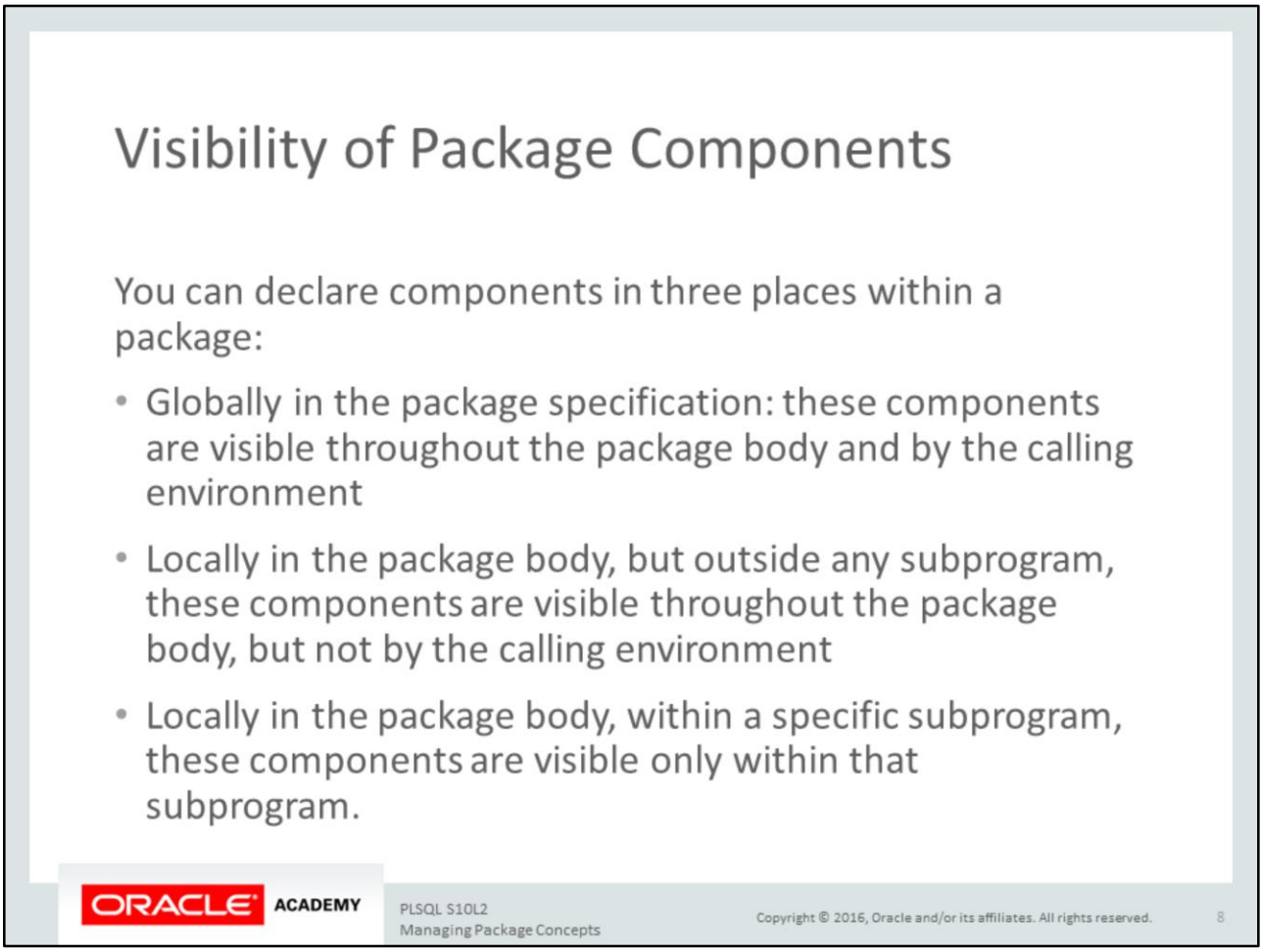


Any package component can be either public or private. This includes not only procedures and functions, but also variables, constants, cursor declarations, and exception declarations.

The package can hide the implementation of a private subprogram so that only the package (not your application) is affected if the code of that private subprogram is changed. Also, by hiding implementation details from users, you protect the integrity of the package.

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.



Remember that a component declared in the specification is (by definition) visible throughout the package. Therefore it does not make sense to talk about declaring local components within the specification. Similarly, it does not make sense to talk about global components declared in the package body.

Public code is defined in the package specification and is available to any schema that has EXECUTE authority on the package. Private code is defined in and visible only from within the package. External programs using the package cannot see the private code, they can only use it indirectly (if it is called from a public subprogram they are calling).

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?

Global/Local Compared to Public/Private

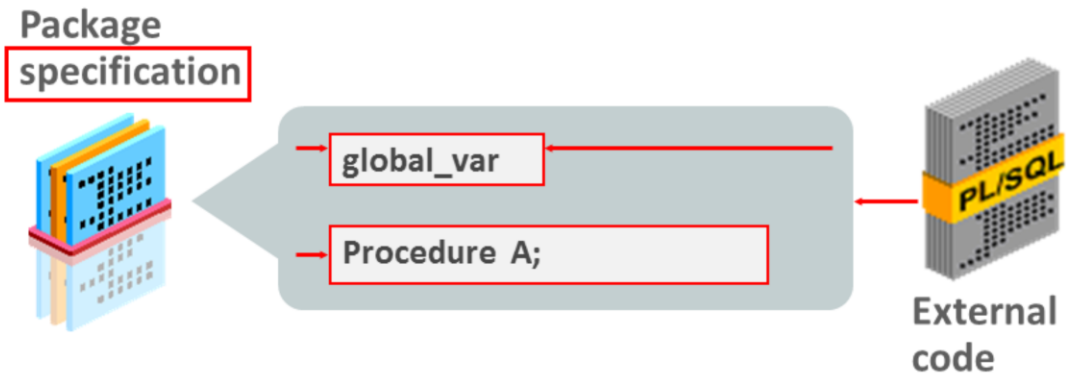
- The answer is “no difference”—they are the same thing!
- But you use public/private when describing procedures and functions, and global/local when describing other components such as variables, constants, and cursors.

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).

Visibility of Global (Public) Components



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.

Visibility of Local (Private) Components

Package body

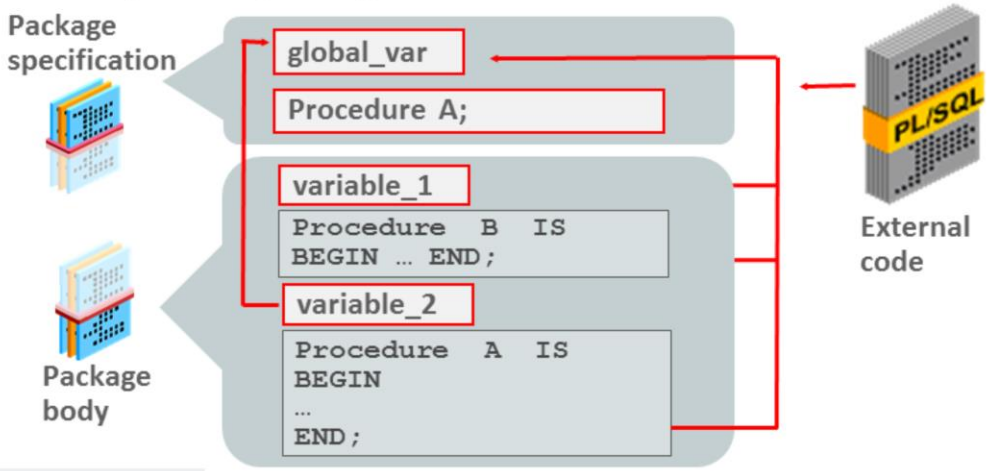
```
variable_1
Procedure  B  IS
BEGIN … END;

variable_2
Procedure  A  IS

BEGIN
…
END;
```


– Variable_1 can be used in both procedures A and B within the package body, but not outside the package

– Variable_2 can be used only in procedure A.

Visibility of Local (Private) Components

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



Why is Procedure A public?

Answer: because it is declared in the specification.

Why is Procedure B local?

Answer: because it is not declared in the specification.

What is the difference between global_var and variable_1?

Answer: they can both be referenced by any procedure in the package, but global_var can be referenced by the calling environment, while variable_1 cannot.

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.

The validation subprogram for checking whether the salary increase is within the 20% limit will be implemented by a private function declared in the package body.

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
```


Validate_raise is a private function and can be invoked only from other subprograms in the same package.

Note that the function references the global variable g_max_sal_raise.

Example of Package Body: salary_pkg:

```
...
 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;
```


Validate_raise is a function returning a Boolean which is tested directly in the IF statement.

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**
	`Subprogram;`
	You can fully qualify a subprogram within the same package, but this is optional.
	`package_name.subprogram;`

External to the package	**Fully qualify the (public) subprogram with its package name** `package_name.subprogram;`

Invoking Package Subprograms

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;
```


ANSWERS:

Invocations #1 and #4 are valid, the others are not.

Invocations #2 is invalid because the procedure-name has not been prefixed with the package-name.

Invocations #3 is invalid because validate_raise is a private function and cannot be referenced from outside the package.

Invocations #5 is invalid because v_old_salary is a local variable which cannot be referenced from outside the update_sal procedure.

Regarding the validity of invocation #4, remember that max_sal_raise is a global constant declared in the specification and is therefore visible from outside the package.

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.

The package must be in your own schema or you must have the DROP ANY PROCEDURE system privilege.

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;
```


The source code for PL/SQL packages is also stored in the data dictionary tables, just as for stand-alone procedures and functions. The source code is viewable in the data dictionary when you execute a SELECT statement on the USER_SOURCE and ALL_SOURCE tables.

When querying the package, use a condition in which the TYPE column is:

Equal to 'PACKAGE' to display the source code for the package specification

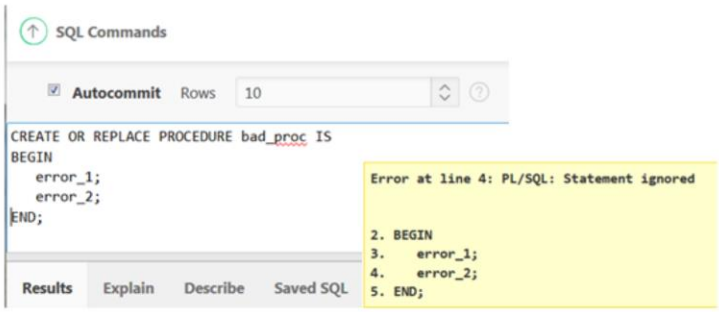
Equal to 'PACKAGE BODY' to display the source code for the package body

Use ORDER BY line to display the lines of source code in the correct sequence.

Note: The values of the NAME and TYPE columns must be uppercase.

Using USER_ERRORS

- When a PL/SQL subprogram fails to compile, Application Express displays the error number and message text for the FIRST error.



- You can query USER_ERRORS to see all errors.

USER_ERRORS contains the most recent compiler errors for all subprograms in your schema.

Using USER_ERRORS

- To see all the errors (not just the first one), you query the USER_ERRORS dictionary table:

```
CREATE OR REPLACE PROCEDURE bad_proc
IS BEGIN
  error_1;  -- this is an error
  error_2;  -- this is another error
END;
```

```
SELECT line, text, position  -- where and error message
  FROM USER_ERRORS
  WHERE name = 'BAD_PROC' AND type = 'PROCEDURE'
  ORDER BY sequence;
```

- The output of this code is on the next slide.

Whenever we try to compile a subprogram, the PL/SQL compiler automatically logs errors in USER_ERRORS, as well as the source code in USER_SOURCE.

Using USER_ERRORS

- The code on the previous slide produces this output:

LINE	TEXT	POSITION
3	PLS-00201: identifier 'ERROR_1' must be declared	4
3	PL/SQL: Statement ignored	4
4	PLS-00201: identifier 'ERROR_2' must be declared	4
4	PL/SQL: Statement ignored	4

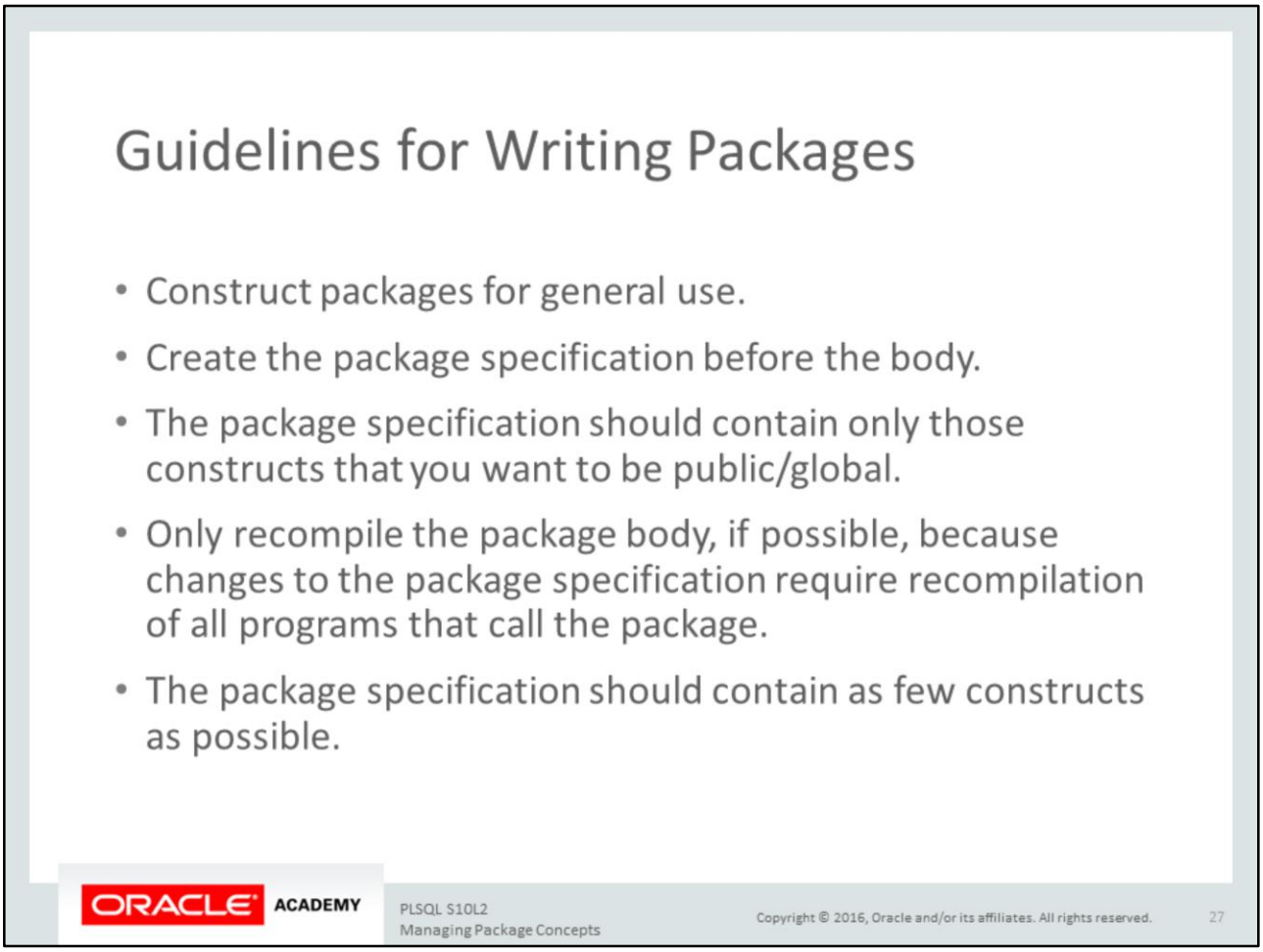
- `USER_ERRORS` does not show the source code.
- We can `JOIN` our query to `USER_SOURCE` to see the source code as well. The next slide shows how.

Adding USER_SOURCE

Join USER_SOURCE and USER_ERRORS to see a more complete picture of the compile errors.

```
SELECT e.line, e.position, s.text AS SOURCE, e.text AS ERROR
  FROM USER_ERRORS e, USER_SOURCE s
  WHERE e.name = s.name AND e.type = s.type
    AND e.line = s.line
    AND e.name = 'BAD_PROC' and e.type = 'PROCEDURE'
  ORDER BY e.sequence;
```

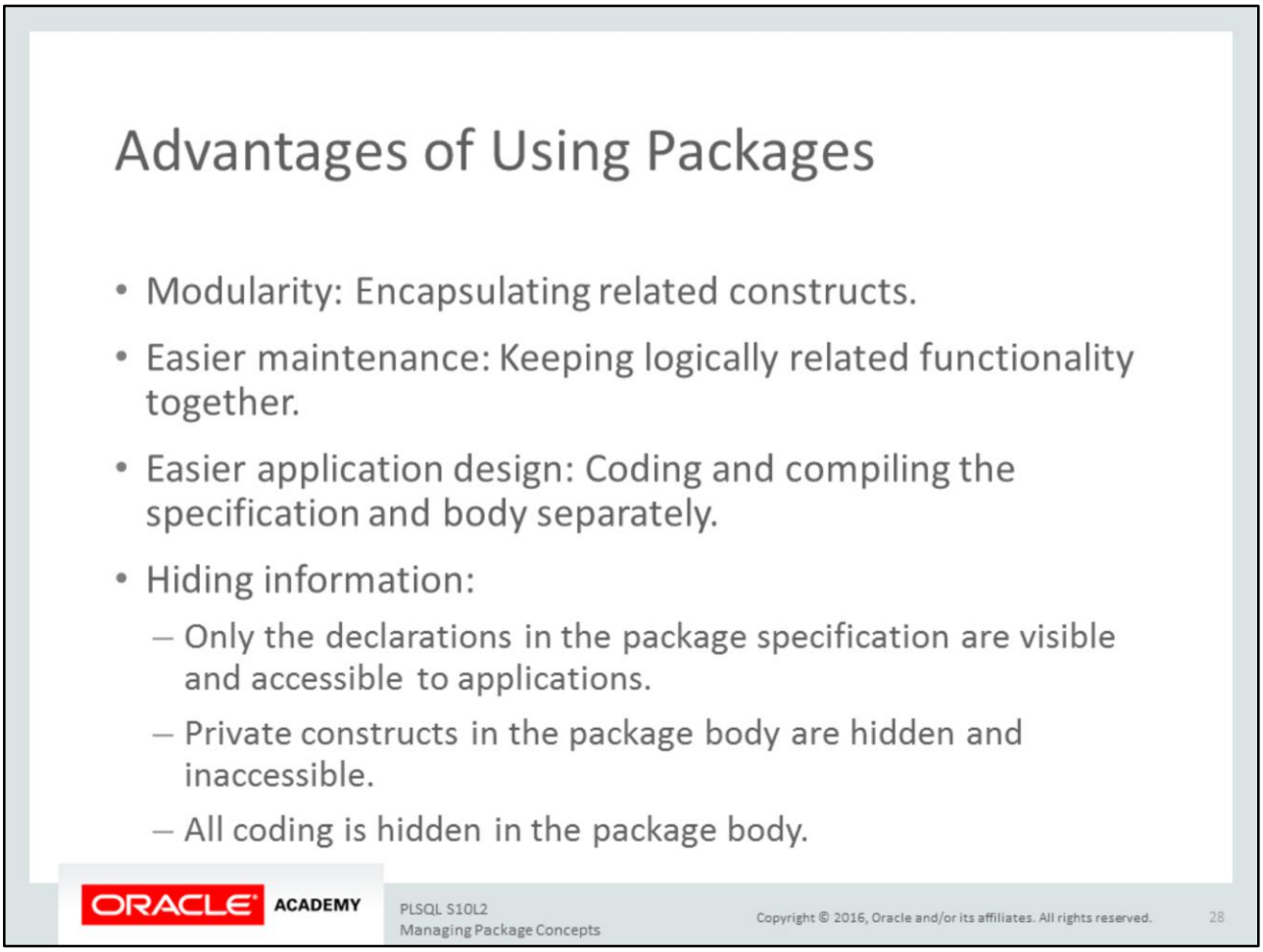
LINE	POSITION	SOURCE	ERROR
3	4	error_1;	PLS-00201: identifier 'ERROR_1' must be declared
3	4	error_1;	PL/SQL: Statement ignored
4	4	error_2;	PLS-00201: identifier 'ERROR_2' must be declared
4	4	error_2;	PL/SQL: Statement ignored



Keep your packages as general as possible so they can be reused in future applications. Avoid writing packages that duplicate features provided by the Oracle server. Well written and well constructed packages can increase the efficiency of the system.

The package specification should reflect the design of your application, so it should be created before creating the package body. The package specification should contain only those constructs that must be visible to the users of the package. This prevents other developers from misusing the package by basing code on irrelevant details.

Changes to the package body do not require recompilation of dependent constructs, whereas changes to the package specification require recompilation of every stored subprogram that references the package. To reduce the need for recompiling when code is changed, place as few constructs as possible in a package specification.



Packages provide an alternative to creating procedures and functions as stand-alone schema objects, and they offer several benefits.

Modularity and ease of maintenance: You encapsulate logically related programming structures in a named module. Each package is easy to understand, and the interface between packages is simple, clear, and well defined.

Easier application design: All you need initially is the interface information in the package specification. You can code and compile a specification without its body. Then stored subprograms that reference the package can compile as well. You need not define the package body fully until you are ready to complete the application.

Hiding information: You decide which constructs are public (visible and accessible) and which are private (hidden and inaccessible). Declarations in the package specification are visible and accessible to applications. The package body hides the definition of the private constructs, so that only the package is affected (not your application or any calling programs) if the definition changes. This enables you to change the implementation without having to recompile the calling programs. Also, by hiding implementation details from users, you protect the integrity of the package.

Most real-life applications will have literally hundreds of individual subprograms. If they are all implemented as standalone programs, they will be very hard to manage, document, and maintain. New developers joining your team will easily be overwhelmed. Packages help you manage the number of stored programs, they make your data dictionary and code more manageable.

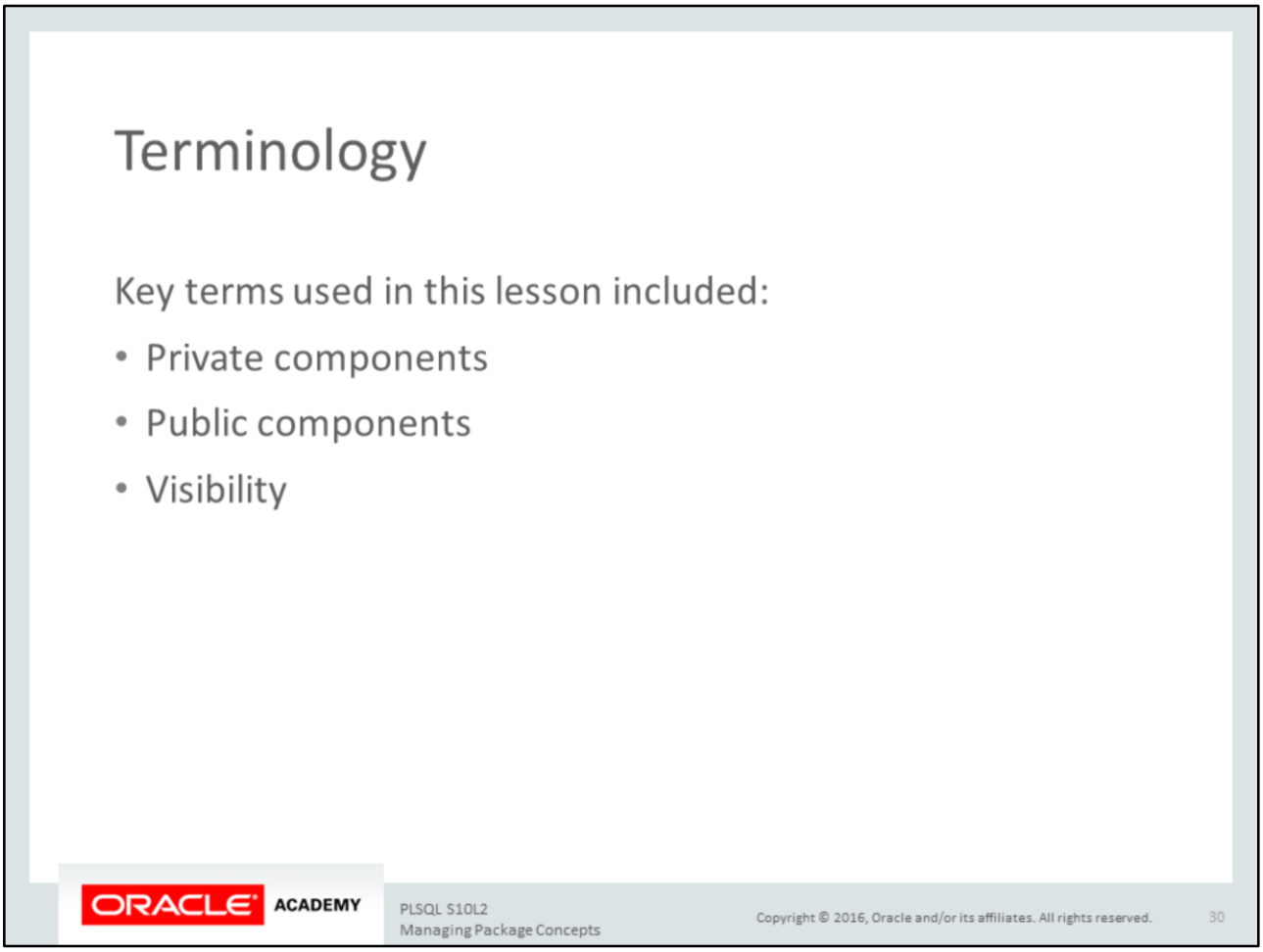
Advantages of Using Packages

- 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.

Added functionality: Packaged global variables and cursors persist for the duration of a session. Thus, they can be shared by all subprograms that execute in the environment. They also enable you to maintain data across transactions without having to store it in the database. Private constructs also persist for the duration of the session but can be accessed only within the package. More details in Lesson 4.

Better performance: When you call a packaged subprogram the first time, the entire package is loaded into memory. Thus, later calls to related subprograms in the package require no further disk I/O. Packaged subprograms also stop cascading dependencies and thus avoid unnecessary compilation. Dependencies are explained in Section 12.

Overloading: With packages, you can overload procedures and functions, which means you can create multiple subprograms with the same name in the same package, each taking parameters of different number or data type. Overloading is explained in the next lesson.



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.

Public components – are declared in the package specification and can be invoked from any calling environment, provided the user has been granted EXECUTE privilege on the package.

Visibility – Describes if a component can be seen, that is, referenced and used by other components or objects.

Summary

In this lesson, you should have learned how 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


ORACLE
ACADEMY