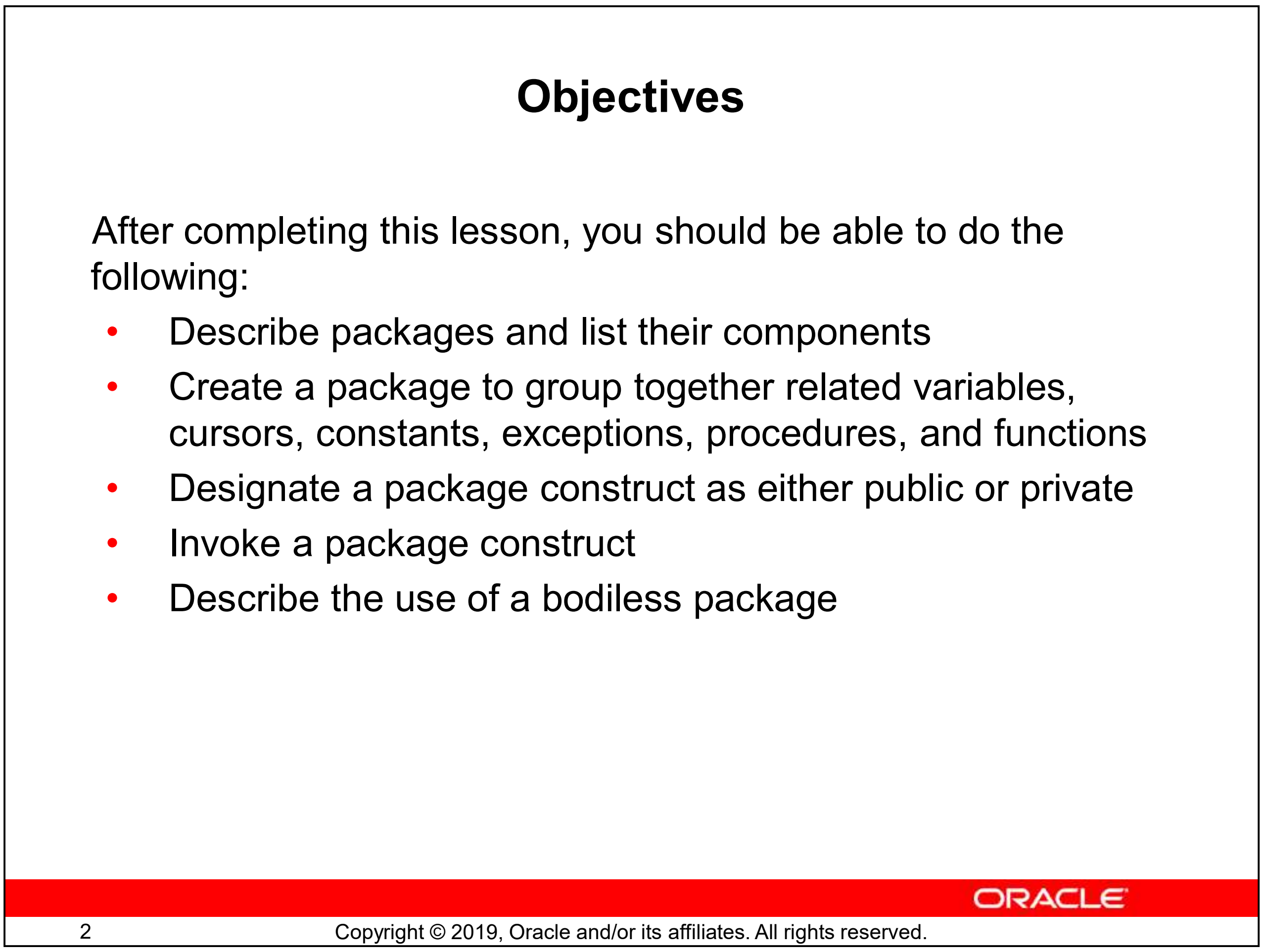
Creating Packages

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



Objectives

After completing this lesson, you should be able to do the following:

- Describe packages and list their components
- Create a package to group together related variables, cursors, constants, exceptions, procedures, and functions
- Designate a package construct as either public or private
- Invoke a package construct
- Describe the use of a bodiless package

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

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

What Are PL/SQL Packages?

- A package is a schema object that groups logically related PL/SQL types, variables, and subprograms.
- Packages usually have two parts:
  – A specification (spec)
  – A body
- The specification is the interface to the package. It declares the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package.
- The body defines the queries for the cursors and the code for the subprograms.
- Enable the Oracle server to read multiple objects into memory at once.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

PL/SQL Packages: Overview

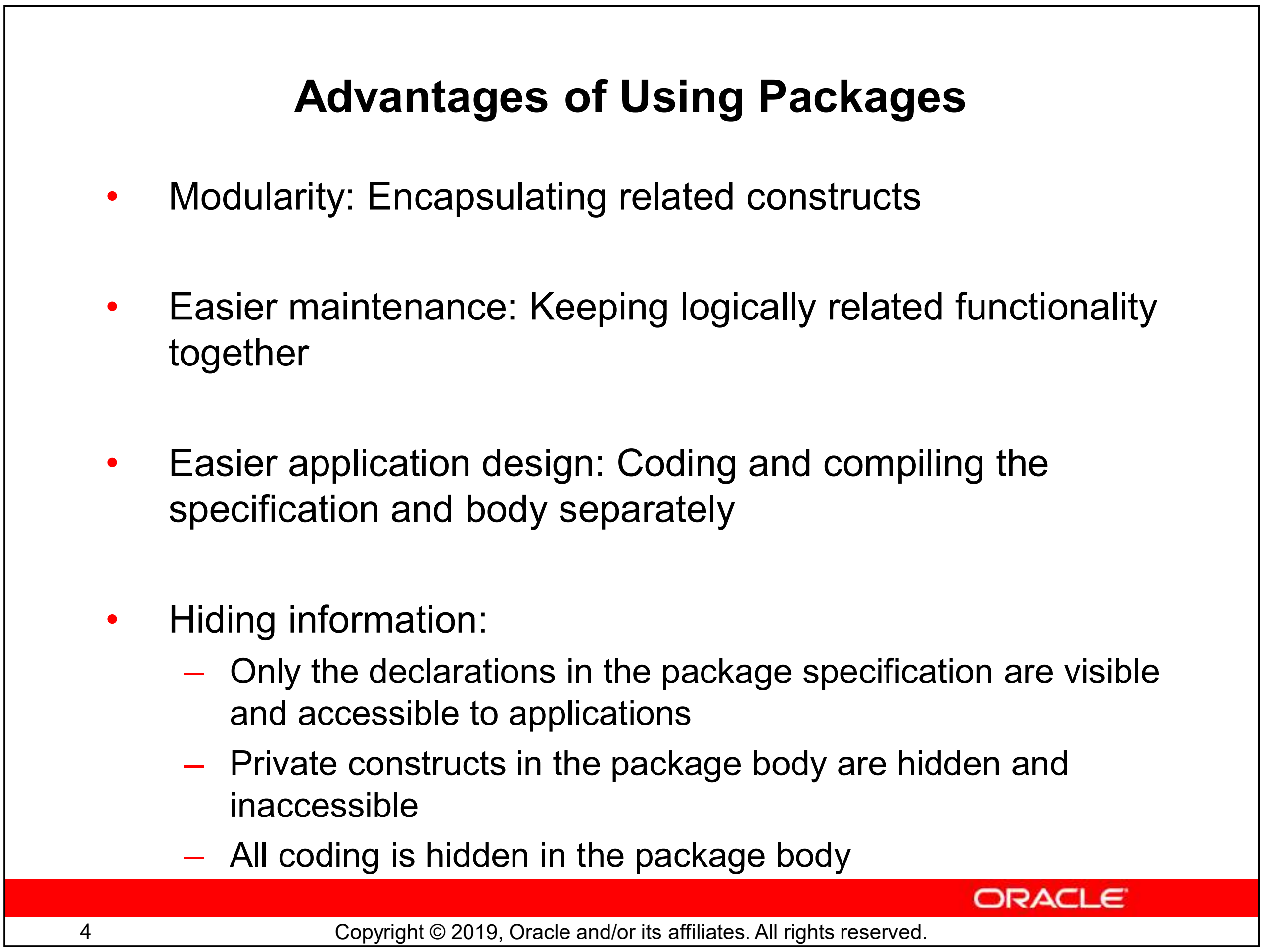
PL/SQL packages enable you to bundle related PL/SQL types, variables, data structures, exceptions, and subprograms into one container. For example, a Human Resources package can contain hiring and firing procedures, commission and bonus functions, and tax exemption variables.

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

- A specification
- A body (optional)

The package itself cannot be called, parameterized, or nested. After writing and compiling, the contents can be shared with many applications.

When a PL/SQL-packaged construct is referenced for the first time, the whole package is loaded into memory. Subsequent access to constructs in the same package does not require disk input/output (I/O).


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

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

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

- **Modularity and easier 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. Thereafter, 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.

Advantages of Using Packages

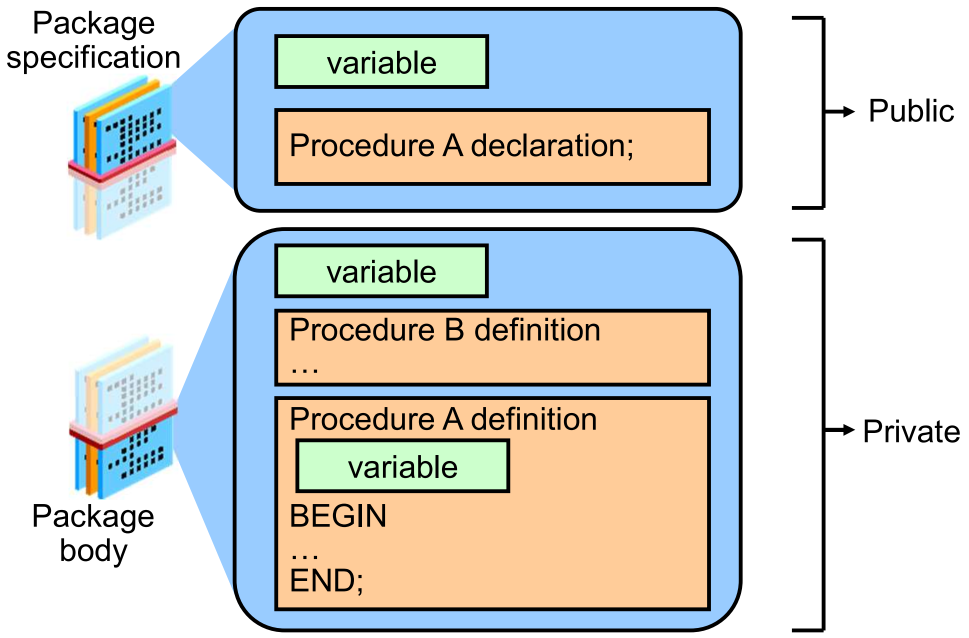
- Better performance:
  - The entire package is loaded into memory when the package is first referenced.
  - There is only one copy in memory for all users.
- Overloading: Multiple subprograms of the same name

ORACLE
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- **Added functionality:** Packaged public 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.
- **Better performance:** When you call a packaged subprogram the first time, the entire package is loaded into memory. Later calls to related subprograms in the package, therefore, require no further disk I/O. Packaged subprograms also stop cascading dependencies and thus avoid unnecessary compilation.
- **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.

Note: Dependencies are covered in detail in the lesson titled “Managing Dependencies.”

Components of a PL/SQL Package



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You create a package in two parts:

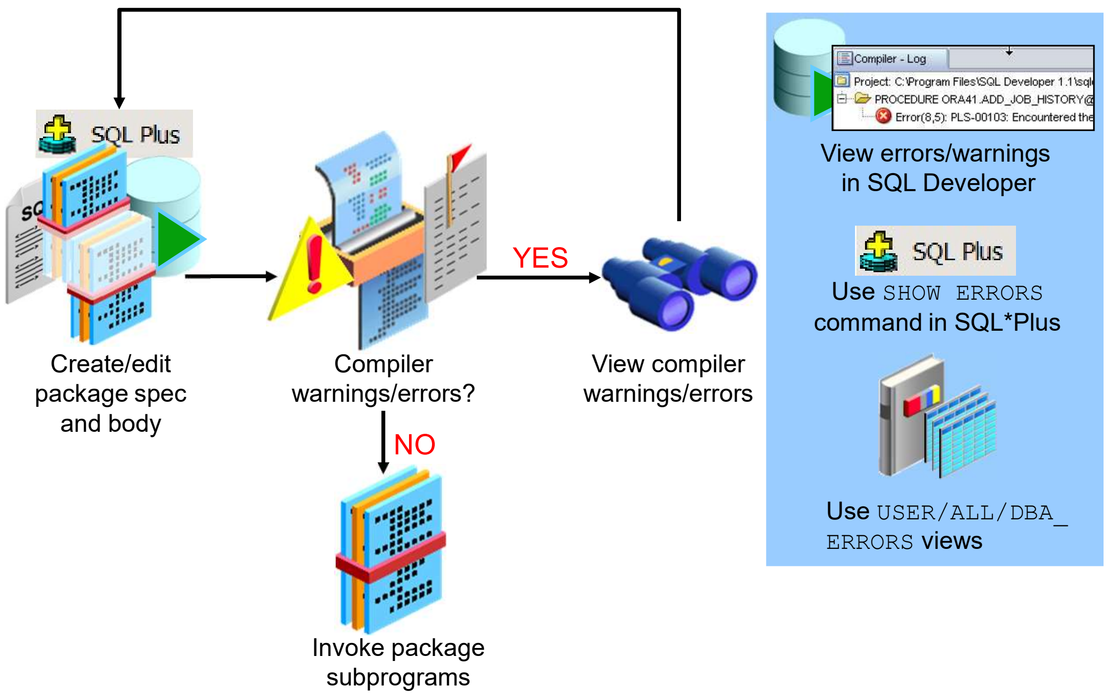
- The **package specification** is the interface to your applications. It declares the public types, variables, constants, exceptions, cursors, and subprograms available for use. The package specification may also include `PRAGMA`s, which are directives to the compiler.
- The **package body** defines its own subprograms and must fully implement subprograms declared in the specification part. The package body may also define PL/SQL constructs, such as types, variables, constants, exceptions, and cursors.

Public components are declared in the package specification. The specification defines a public application programming interface (API) for users of package features and functionality—that is, public components can be referenced from any Oracle server environment that is external to the package.

Private components are placed in the package body and can be referenced only by other constructs within the same package body. Private components can reference the public components of a package.

Note: If a package specification does not contain subprogram declarations, then there is no requirement for a package body.

Developing PL/SQL Packages: Overview


SQL Plus
Create/edit package spec and body
Compiler warnings/errors?
YES
NO
View compiler warnings/errors
Invoke package subprograms
Compiler - Log
View errors/warnings in SQL Developer
SQL Plus
Use SHOW ERRORS command in SQL*Plus
Use USER/ALL/DBA_ ERRORS views

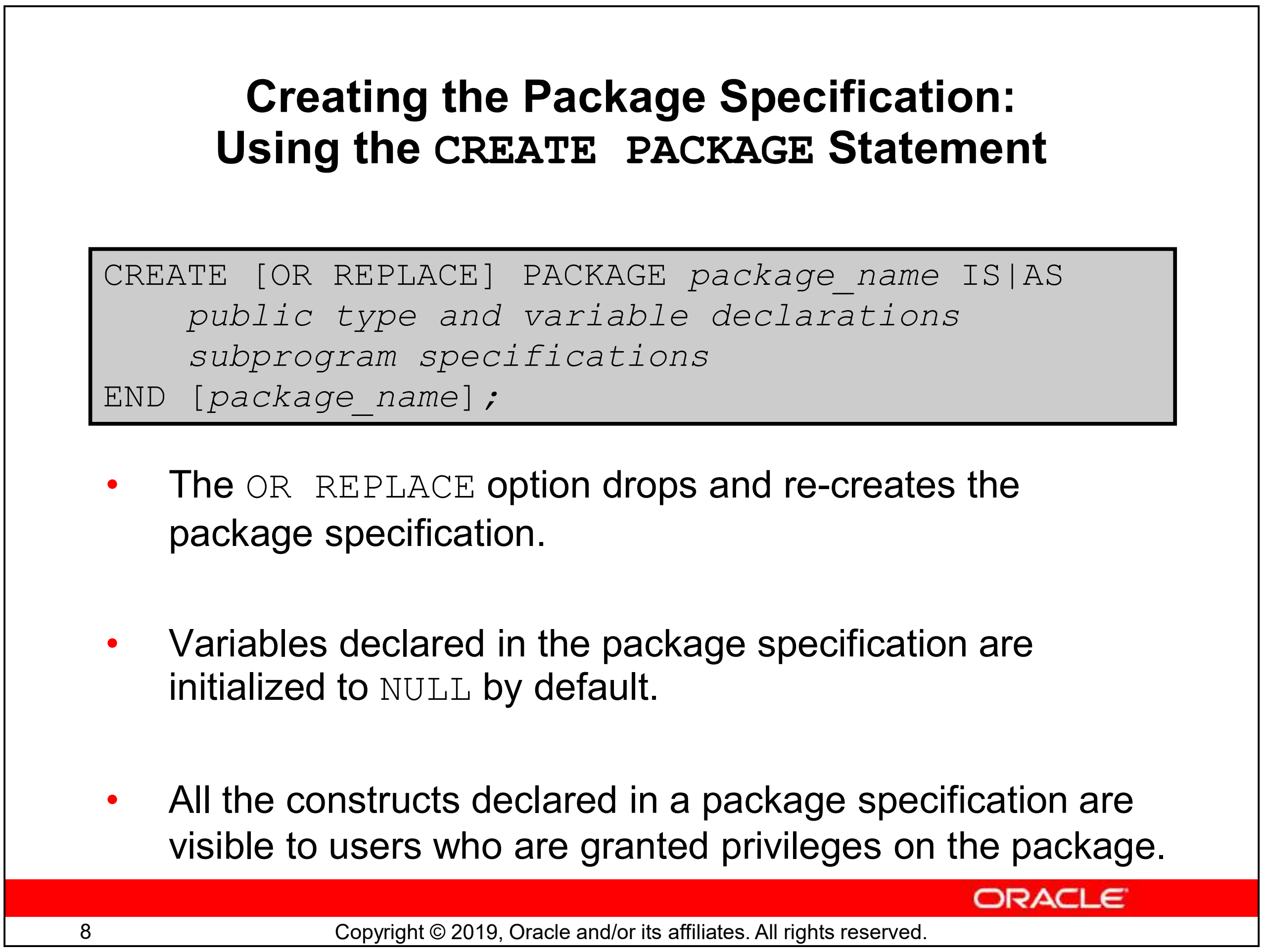
```
Project: C:\Program Files\SQL Developer 1.1\sql
PROCEDURE ORA41.ADD_JOB_HISTORY@
Error(8,5): PLS-00103: Encountered the
```

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Developing PL/SQL Packages

The graphic in the slide illustrates the basic steps involved in developing and using a package:

1. Create the procedure using SQL Developer's Object Navigator tree or the SQL Worksheet area.
2. Compile the package. The package is created in the database. The `CREATE PACKAGE` statement creates and stores source code and the compiled *m-code* in the database. To compile the package, right-click the package's name in the Object Navigator tree, and then click Compile.
3. If there are no compilation warnings or errors, you execute any public construct within the package specification from an Oracle Server environment.
4. If there are compilation warning or errors, you can view (and then correct) the warnings or errors using one of the following methods:
   - Using the SQL Developer interface (the Compiler – Log tab)
   - Using the `SHOW ERRORS` SQL*Plus command
   - Using the `USER/ALL/DBA_ERRORS` views



Creating the Package Specification: Using the `CREATE PACKAGE` Statement

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

- 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 privileges on the package.

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Creating the Package Specification

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

- Specify the `OR REPLACE` option if overwriting an existing package specification.
- Initialize a variable with a constant value or formula within the declaration, if required; otherwise, the variable is initialized implicitly to `NULL`.

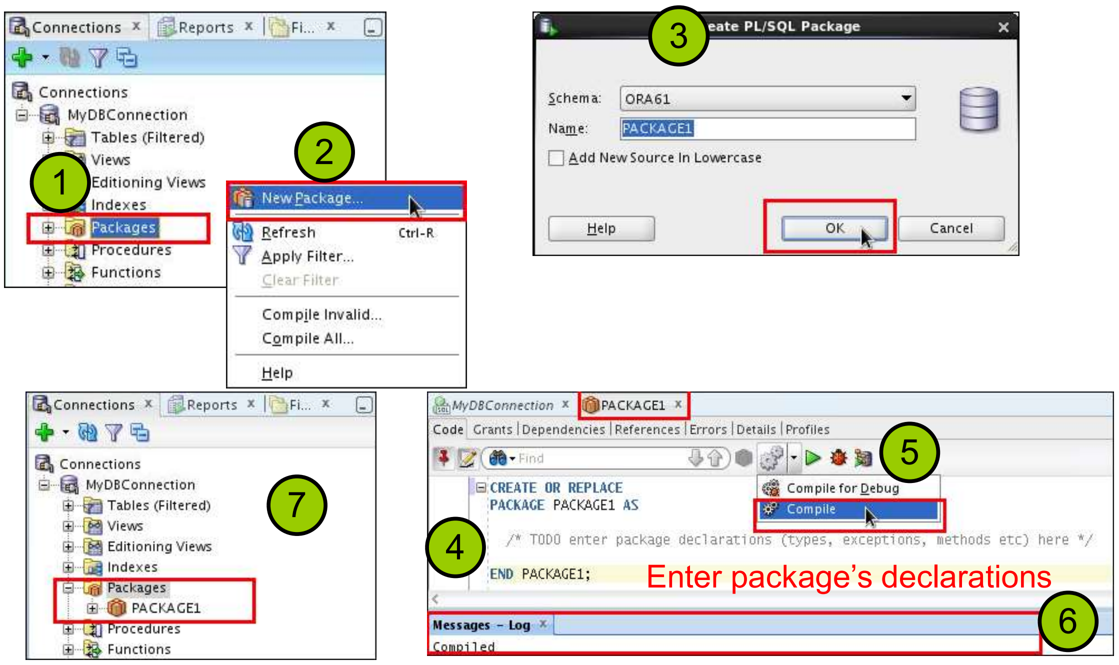
The following are definitions of items in the package syntax:

- **`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.
- **`subprogram specification`** specifies the public procedure or function declarations.

The package specification should contain procedure and function headings terminated by a semicolon, without the `IS` (or `AS`) keyword and its PL/SQL block. The implementation of a procedure or function that is declared in a package specification is done in the package body.

The Oracle database stores the specification and body of a package separately. This enables you to change the implementation of a program construct in the package body without invalidating other schema objects that call or reference the program construct.

Creating the Package Specification: Using SQL Developer



Schema: ORA61

Name: PACKAGE1

```
CREATE OR REPLACE
PACKAGE PACKAGE1 AS

  /* TODO enter package declarations (types, exceptions, methods etc) here */

END PACKAGE1;
```

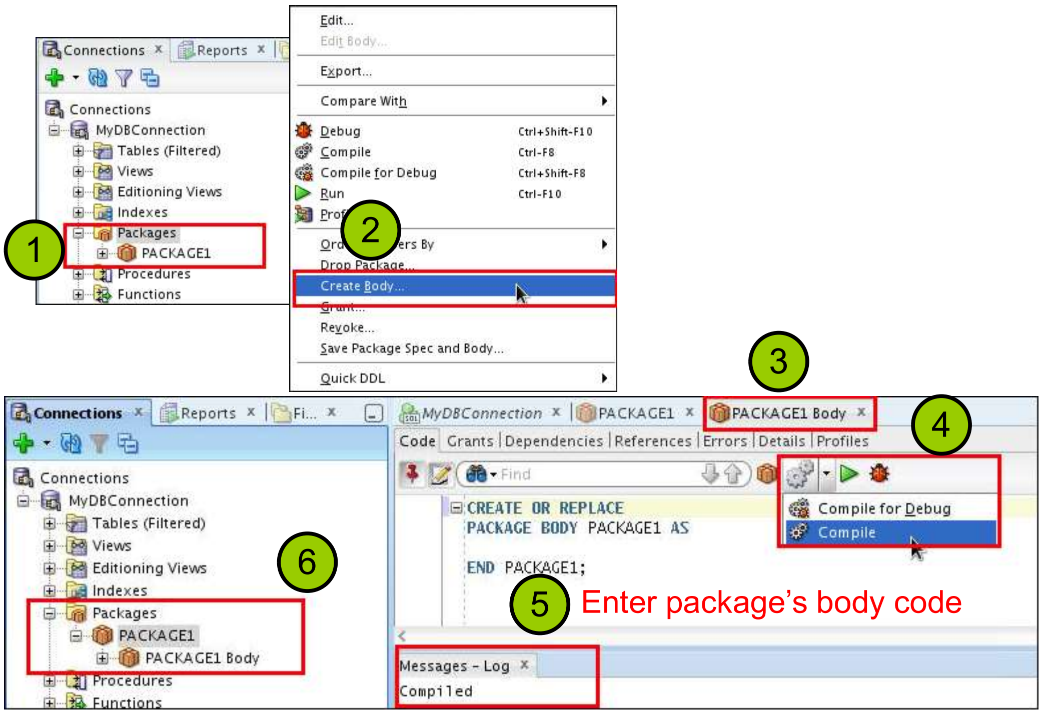
```
Compiled
```

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can use SQL Developer to create the package specification as follows:

1. Right-click the **Packages** node in the Connections navigation tree.
2. Select **New Package** from the shortcut menu.
3. In the **Create PL/SQL Package** window, select the schema name, enter the name for the new package, and then click OK. A tab for the new package is displayed along with the shell for the new package.
4. Enter the code for the new package.
5. Compile or save (using the Save icon on the main toolbar) the new package.
6. The **Messages – Log** tab displays whether or not the compilation was successful.
7. The newly created package is displayed under the **Packages** node in the Connections navigation tree.

Creating the Package Body: Using SQL Developer



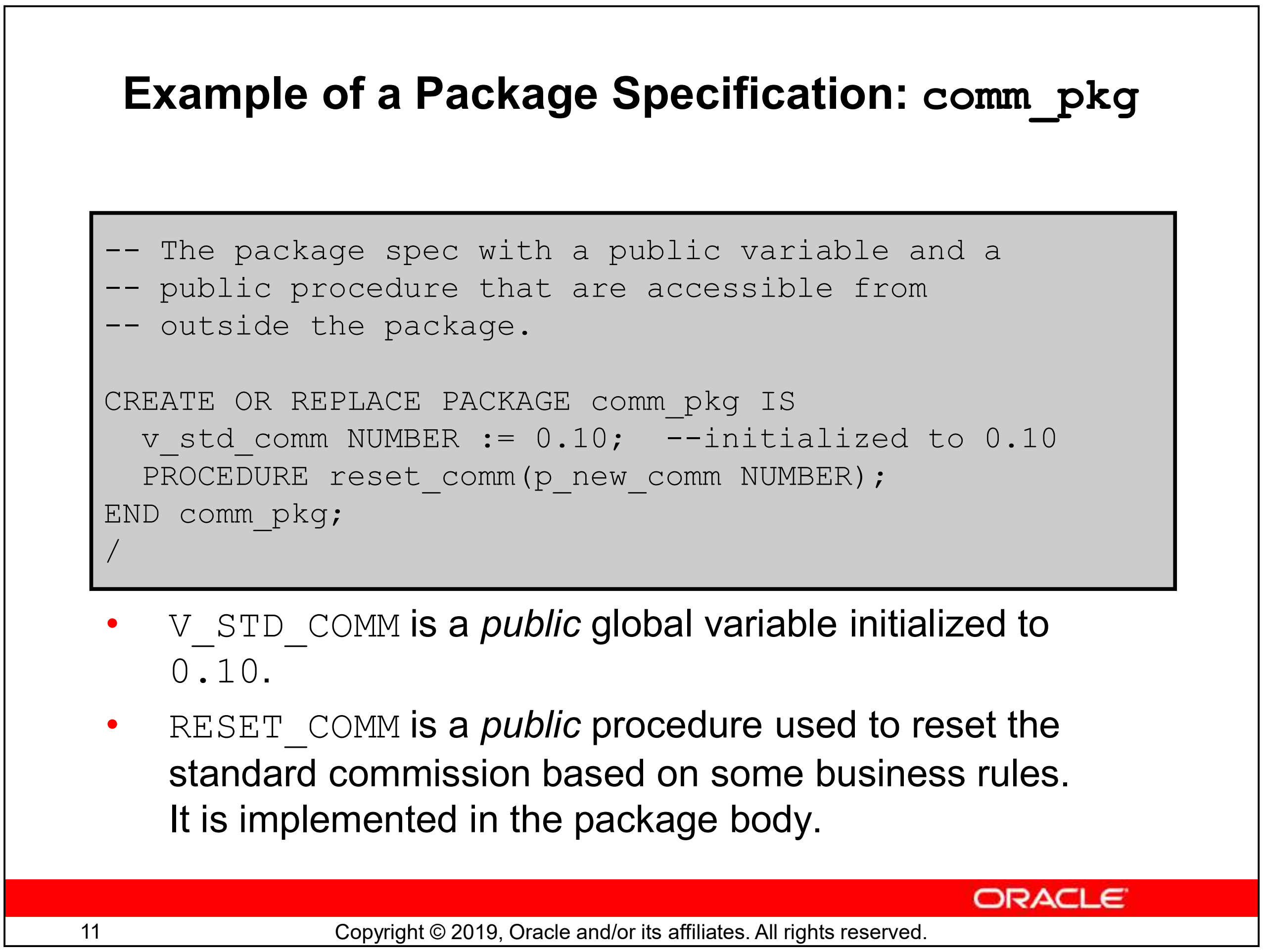
```
CREATE OR REPLACE
PACKAGE BODY PACKAGE1 AS

END PACKAGE1;
```

```
Compiled
```

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can use SQL Developer to create the package body as follows:

1. Right-click the package name for which you are creating a body in the **Packages** node in the Connections navigation tree.
2. Select **Create Body** from the shortcut menu. A tab for the new package body is displayed along with the shell for the new package body.
3. Enter the code for the new package body.
4. Compile or save the package body.
5. The **Messages – Log** tab displays whether or not the compilation was successful.
6. The newly created package body is displayed under the **Packages** node in the Connections navigation tree.



Example of a Package Specification: `comm_pkg`

```
-- The package spec with a public variable and a
-- public procedure that are accessible from
-- outside the package.

CREATE OR REPLACE PACKAGE comm_pkg IS
  v_std_comm NUMBER := 0.10;  --initialized to 0.10
  PROCEDURE reset_comm(p_new_comm NUMBER);
END comm_pkg;
/
```

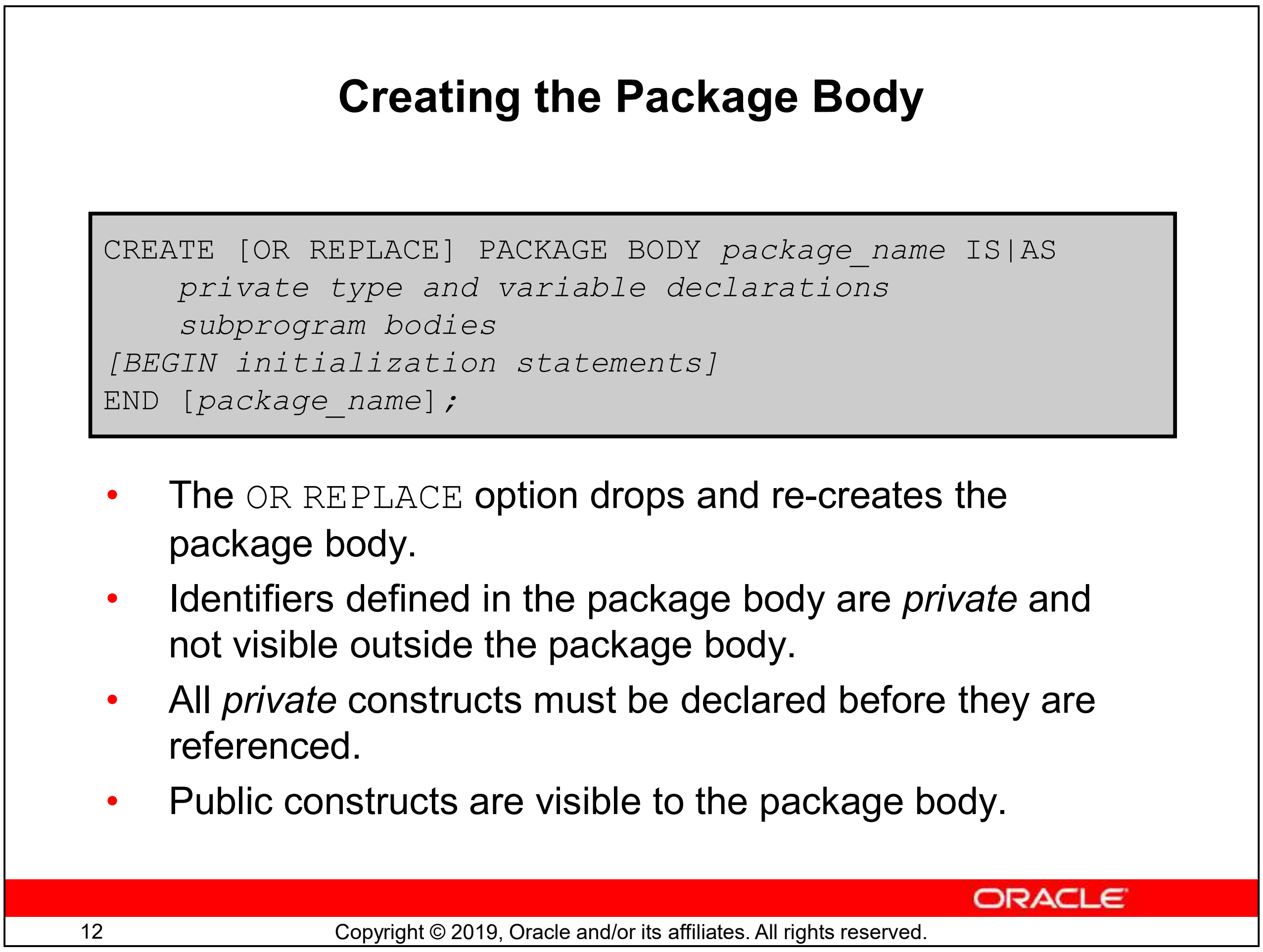
- `V_STD_COMM` is a *public* global variable initialized to `0.10`.
- `RESET_COMM` is a *public* procedure used to reset the standard commission based on some business rules. It is implemented in the package body.

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The example in the slide creates a package called `comm_pkg` used to manage business processing rules for commission calculations.

The `v_std_comm` public (global) variable is declared to hold a maximum allowable percentage commission for the user session, and it is initialized to `0.10` (that is, 10%).

The `reset_comm` public procedure is declared to accept a new commission percentage that updates the standard commission percentage if the commission validation rules are accepted. The validation rules for resetting the commission are not made public and do not appear in the package specification. The validation rules are managed by using a private function in the package body.



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];
```

- The `OR REPLACE` option drops and re-creates the package body.
- Identifiers defined in the package body are *private* and not visible outside the package body.
- All *private* constructs must be declared before they are referenced.
- Public constructs are visible to the package body.

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

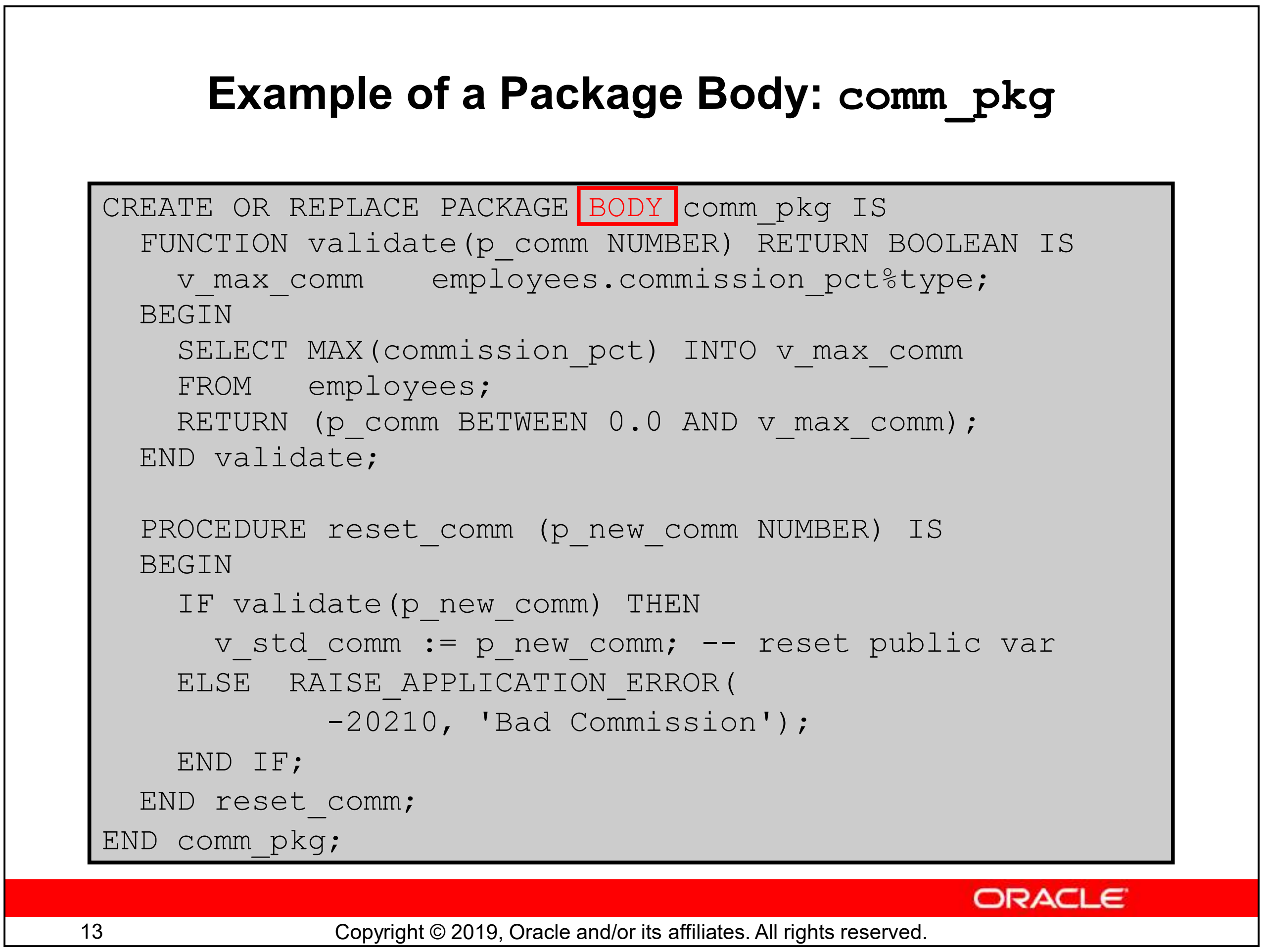
Create a package body to define and implement all public subprograms and supporting private constructs. When creating a package body, perform the following steps:

- 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. It is common to see all private variables and subprograms defined first and the public subprograms defined last in the package body.
- Complete the implementation for all procedures or functions declared in the package specification within the package body.

The following are definitions of items in the package body syntax:

- **`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.
- **`private type and variable declarations`** declares private variables, constants, cursors, exceptions, user-defined types, and subtypes.
- **`subprogram specification`** specifies the full implementation of any private and/or public procedures or functions.
- **`[BEGIN initialization statements]`** is an optional block of initialization code

that executes when the package is first referenced.



Example of a Package Body: `comm_pkg`

```
CREATE OR REPLACE PACKAGE BODY comm_pkg IS
  FUNCTION validate(p_comm NUMBER) RETURN BOOLEAN IS
    v_max_comm    employees.commission_pct%type;
  BEGIN
    SELECT MAX(commission_pct) INTO v_max_comm
    FROM   employees;
    RETURN (p_comm BETWEEN 0.0 AND v_max_comm);
  END validate;

  PROCEDURE reset_comm (p_new_comm NUMBER) IS
  BEGIN
    IF validate(p_new_comm) THEN
      v_std_comm := p_new_comm; -- reset public var
    ELSE  RAISE_APPLICATION_ERROR(
            -20210, 'Bad Commission');
    END IF;
  END reset_comm;
END comm_pkg;
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The slide shows the complete package body for `comm_pkg`, with a private function called `validate` to check for a valid commission. The validation requires that the commission be positive and less than the highest commission among existing employees. The `reset_comm` procedure invokes the private validation function before changing the standard commission in `v_std_comm`. In the example, note the following:

- The `v_std_comm` variable referenced in the `reset_comm` procedure is a public variable. Variables declared in the package specification, such as `v_std_comm`, can be directly referenced without qualification.
- The `reset_comm` procedure implements the public definition in the specification.
- In the `comm_pkg` body, the `validate` function is private and is directly referenced from the `reset_comm` procedure without qualification.

Note: The `validate` function appears before the `reset_comm` procedure because the `reset_comm` procedure references the `validate` function. It is possible to create forward declarations for subprograms in the package body if their order of appearance needs to be changed. If a package specification declares only types, constants, variables, and exceptions without any subprogram specifications, then the package body is unnecessary. However, the body can be used to initialize items declared in the package specification.

Invoking the Package Subprograms: Examples

```
-- Invoke a function within the same packages:
CREATE OR REPLACE PACKAGE BODY comm_pkg IS ...
  PROCEDURE reset_comm(p_new_comm NUMBER) IS
  BEGIN
    IF validate(p_new_comm) THEN
      v_std_comm := p_new_comm;
    ELSE ...
    END IF;
  END reset_comm;
END comm_pkg;
```

```
-- Invoke a package procedure from SQL*Plus:
EXECUTE comm_pkg.reset_comm(0.15)
```

```
-- Invoke a package procedure in a different schema:
EXECUTE scott.comm_pkg.reset_comm(0.15)
```

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Invoking Package Subprograms

After the package is stored in the database, you can invoke public or private subprograms within the same package, or public subprograms if external to the package. Fully qualify the subprogram with its package name when invoked externally from the package. Use the `package_name.subprogram` syntax.

Fully qualifying a subprogram when invoked within the same package is optional.

Example 1: Invokes the `validate` function from the `reset_comm` procedure within the same package. The package name prefix is not required; it is optional.

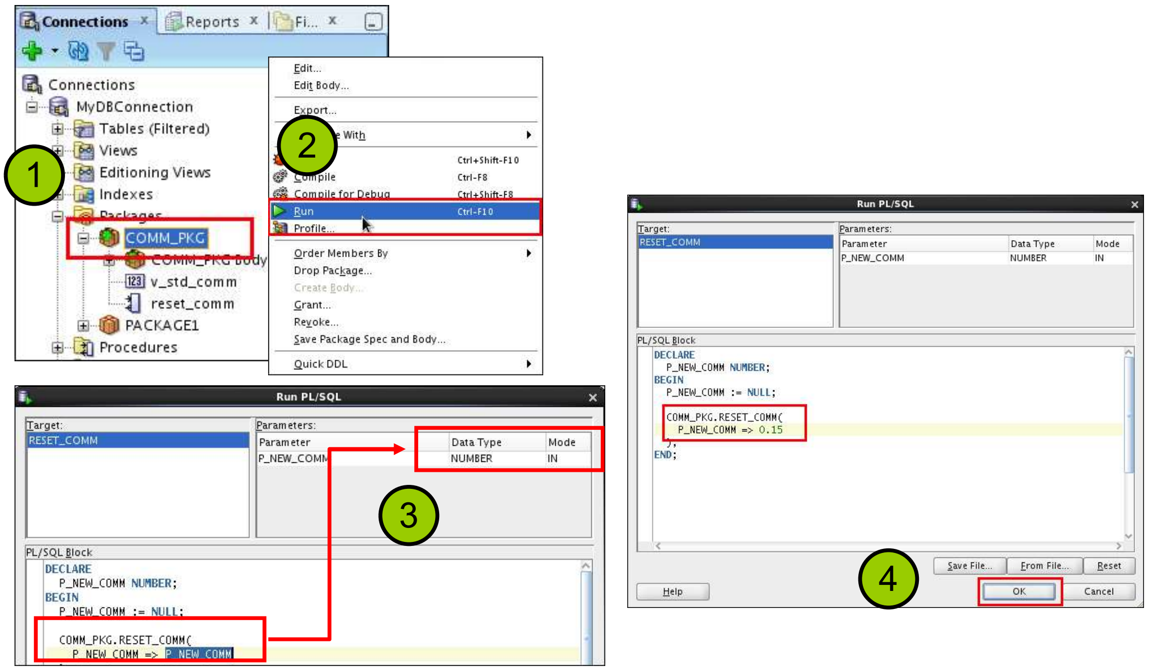
Example 2: Calls the `reset_comm` procedure from SQL*Plus (an environment external to the package) to reset the prevailing commission to 0.15 for the user session.

Example 3: Calls the `reset_comm` procedure that is owned by a schema user called `SCOTT`. Using SQL*Plus, the qualified package procedure is prefixed with the schema name. This can be simplified by using a synonym that references the `schema.package_name`.

Assume that a database link named `NY` has been created for a remote database in which the `reset_comm` package procedure is created. To invoke the remote procedure, use:

```
EXECUTE comm_pkg.reset_comm(0.15)
```

Invoking the Package Subprograms: Using SQL Developer



Run PL/SQL

Target: RESET_COMM

Parameters:

Parameter	Data Type	Mode
P_NEW_COMM	NUMBER	IN

PL/SQL Block

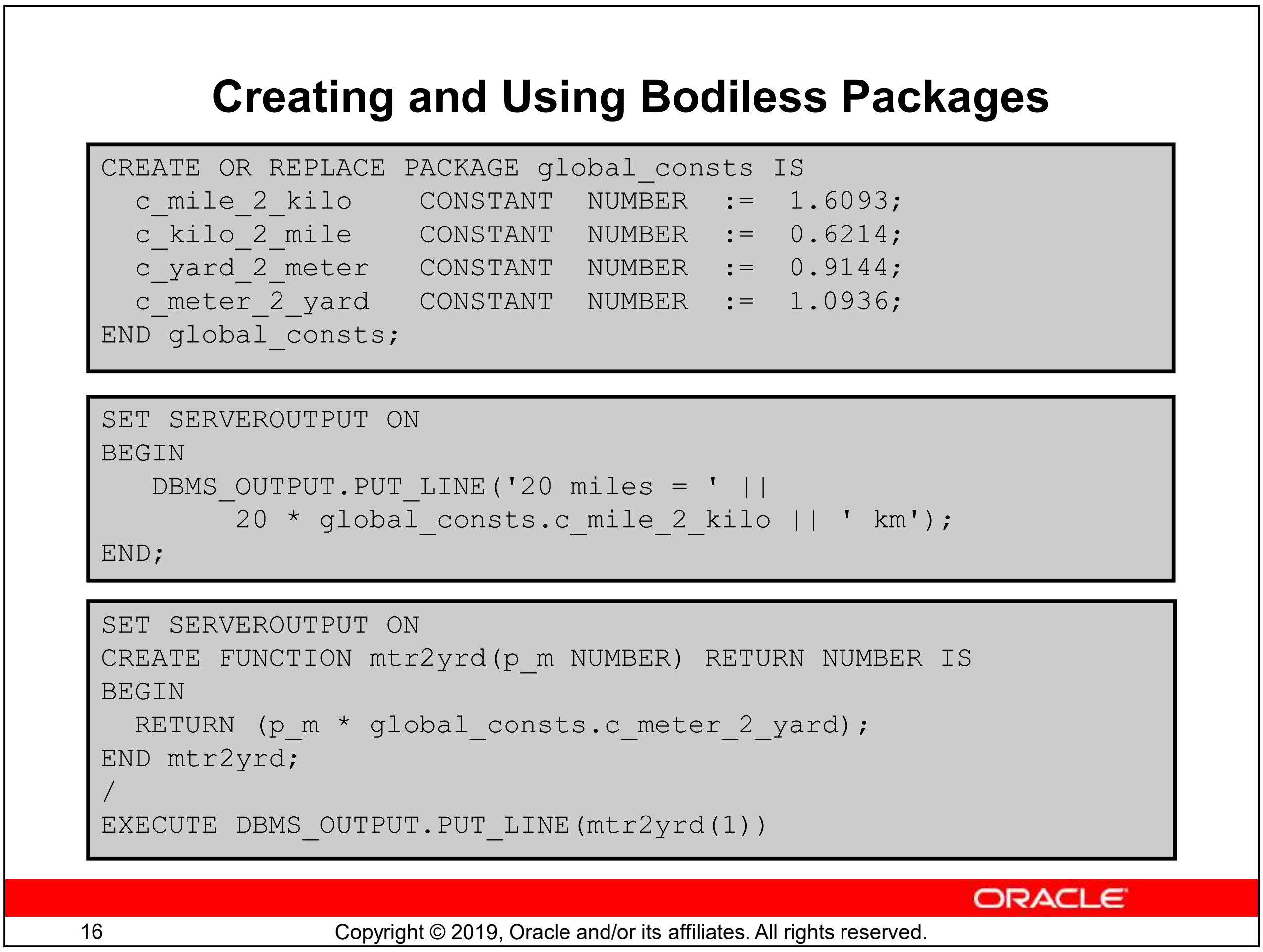
```
DECLARE
  P_NEW_COMM NUMBER;
BEGIN
  P_NEW_COMM := NULL;

  COMM_PKG.RESET_COMM(
    P_NEW_COMM => P_NEW_COMM
```

```
DECLARE
  P_NEW_COMM NUMBER;
BEGIN
  P_NEW_COMM := NULL;

  COMM_PKG.RESET_COMM(
    P_NEW_COMM => 0.15
  );
END;
```

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can use SQL Developer to invoke a package's subprogram as follows:

1. Right-click the package's name in the Packages node in the Navigation tree.
2. Select **Run** from the floating menu. The **Run PL/SQL** window is displayed. You can use the **Run PL/SQL** window to specify parameter values for running a PL/SQL function or procedure. (If you specify a package, select a function or procedure in the package.) Specify the following:
   a. **Target:** Select the name of the function or procedure to run.
   b. **Parameters:** This section lists each parameter for the specified target. The mode of each parameter can be `IN` (the value is passed in), `OUT` (the value is returned), or `IN/OUT` (the value is passed in, and the result of the function or procedure's action is stored in the parameter).
3. In the **PL/SQL Block** section, change the formal `IN` and `IN/OUT` parameter specifications in this block to actual values that you want to use for running the function or procedure. For example, to specify `0.15` as the value for an input parameter named `P_NEW_COMM`, change `P_NEW_COMM => P_NEW_COMM` to `P_NEW_COMM => 0.15`.
4. Click **OK**. SQL Developer runs the function or procedure.



Creating and Using Bodiless Packages

```
CREATE OR REPLACE PACKAGE global_consts IS
  c_mile_2_kilo    CONSTANT  NUMBER  :=  1.6093;
  c_kilo_2_mile    CONSTANT  NUMBER  :=  0.6214;
  c_yard_2_meter   CONSTANT  NUMBER  :=  0.9144;
  c_meter_2_yard   CONSTANT  NUMBER  :=  1.0936;
END global_consts;
```

```
SET SERVEROUTPUT ON
BEGIN
   DBMS_OUTPUT.PUT_LINE('20 miles = ' ||
        20 * global_consts.c_mile_2_kilo || ' km');
END;
```

```
SET SERVEROUTPUT ON
CREATE FUNCTION mtr2yrd(p_m NUMBER) RETURN NUMBER IS
BEGIN
  RETURN (p_m * global_consts.c_meter_2_yard);
END mtr2yrd;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(mtr2yrd(1))
```

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The variables and constants declared within stand-alone subprograms exist only for the duration that the subprogram executes. To provide data that exists for the duration of the user session, create a package specification containing public (global) variables and constant declarations. In this case, create a package specification without a package body, known as a *bodiless package*. As discussed earlier in this lesson, if a specification declares only types, constants, variables, and exceptions, then the package body is unnecessary.

Examples

The first code box in the slide creates a bodiless package specification with several constants to be used for conversion rates. A package body is not required to support this package specification. It is assumed that the `SET SERVEROUTPUT ON` statement was issued before executing the code examples in the slide.

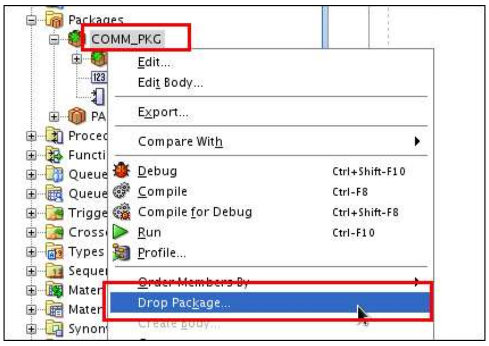
The second code box references the `c_mile_2_kilo` constant in the `global_consts` package by prefixing the package name to the identifier of the constant.

The third example creates a stand-alone function `c_mtr2yrd` to convert meters to yards, and uses the constant conversion rate `c_meter_2_yard` declared in the `global_consts` package. The function is invoked in a `DBMS_OUTPUT.PUT_LINE` parameter.

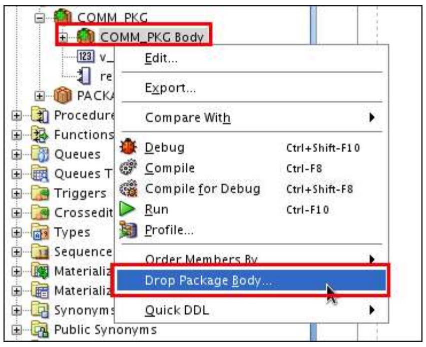
Rule to be followed: When referencing a variable, cursor, constant, or exception from outside the package, you must qualify it with the name of the package.

Removing Packages by Using SQL Developer or the SQL `DROP` Statement

Drop the package specification and body.



Drop only the package body.



```
-- Remove the package specification and body
DROP PACKAGE package_name;
```

```
-- Remove the package body only
DROP PACKAGE BODY package_name;
```

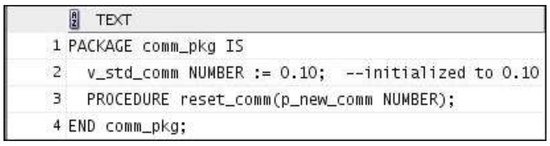
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Removing Packages

When a package is no longer required, you can use a SQL statement in SQL Developer to remove it. A package has two parts; therefore, you can remove the whole package, or you can remove only the package body and retain the package specification.

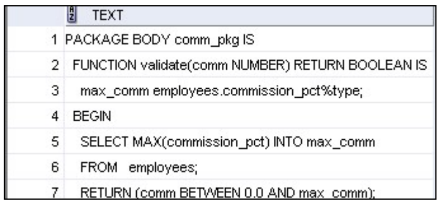
Viewing Packages by Using the Data Dictionary

```
-- View the package specification.
SELECT text
FROM   user_source
WHERE  name = 'COMM_PKG' AND type = 'PACKAGE'
ORDER BY LINE;
```



```
TEXT
PACKAGE comm_pkg IS
  v_std_comm NUMBER := 0.10;  --initialized to 0.10
  PROCEDURE reset_comm(p_new_comm NUMBER);
END comm_pkg;
```

```
-- View the package body.
SELECT text
FROM   user_source
WHERE  name = 'COMM_PKG' AND type = 'PACKAGE BODY'
ORDER BY LINE;
```



```
TEXT
PACKAGE BODY comm_pkg IS
 FUNCTION validate(comm NUMBER) RETURN BOOLEAN IS
   max_comm employees.commission_pct%type;
 BEGIN
   SELECT MAX(commission_pct) INTO max_comm
   FROM  employees;
   RETURN (comm BETWEEN 0.0 AND max_comm);
```

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Viewing Packages in the Data Dictionary

The source code for PL/SQL packages is also stored in the `USER_SOURCE` and `ALL_SOURCE` data dictionary views. The `USER_SOURCE` table is used to display PL/SQL code that you own. The `ALL_SOURCE` table is used to display PL/SQL code to which you have been granted the `EXECUTE` right by the owner of that subprogram code and provides an `OWNER` column in addition to the preceding columns.

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

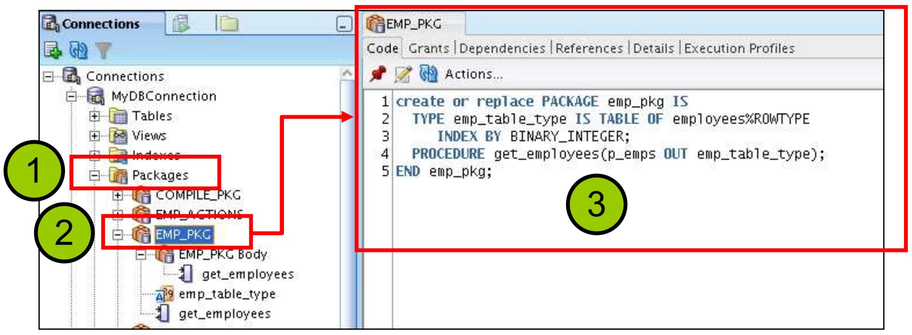
- Equal to `'PACKAGE'` to display the source code for the package specification
- Equal to `'PACKAGE BODY'` to display the source code for the package body

You can also view the package specification and body in SQL Developer using the package name in the Packages node.

Note: You cannot display the source code for Oracle PL/SQL built-in packages, or PL/SQL whose source code has been wrapped by using a `WRAP` utility or obfuscation. Obfuscating and wrapping PL/SQL source code is covered in a later lesson. Clicking the Execute Statement (F9) icon (instead of the Run Script icon) in the SQL Worksheet toolbar, sometimes displays a better formatted output in the Results tab as shown in the slide examples.

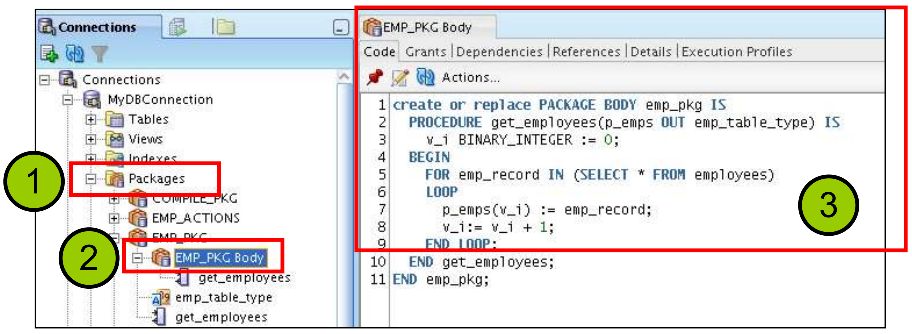
Viewing Packages by Using SQL Developer

To view the package spec, click the package name.



```
create or replace PACKAGE emp_pkg IS
  TYPE emp_table_type IS TABLE OF employees%ROWTYPE
     INDEX BY BINARY_INTEGER;
  PROCEDURE get_employees(p_emps OUT emp_table_type);
END emp_pkg;
```

To view the package body, click the package body.



```
create or replace PACKAGE BODY emp_pkg IS
  PROCEDURE get_employees(p_emps OUT emp_table_type) IS
    v_i BINARY_INTEGER := 0;
  BEGIN
    FOR emp_record IN (SELECT * FROM employees)
    LOOP
      p_emps(v_i) := emp_record;
      v_i:= v_i + 1;
    END LOOP;
  END get_employees;
END emp_pkg;
```

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To view a package's spec in SQL Developer, use the following steps:

1. Click the **Packages** node in the **Connections** tab.
2. Click the package's name.
3. The package's spec code is displayed in the **Code** tab as shown in the slide.

To view a package's body in SQL Developer, use the following steps:

1. Click the **Packages** node in the **Connections** tab.
2. Click the package’s body.
3. The package’s body code is displayed in the **Code** tab as shown in the slide.

Quiz

The package specification is the interface to your applications. It declares the public types, variables, constants, exceptions, cursors, and subprograms available for use. The package specification may also include PRAGMAs, which are directives to the compiler.

a. True

b. False

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Answer: a

Summary

In this lesson, you should have learned how to:

- Describe packages and list their components
- Create a package to group related variables, cursors, constants, exceptions, procedures, and functions
- Designate a package construct as either public or private
- Invoke a package construct
- Describe the use of a bodiless package

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You group related procedures and functions in a package. Packages improve organization, management, security, and performance.

A package consists of a package specification and a package body. You can change a package body without affecting its package specification.

Packages enable you to hide source code from users. When you invoke a package for the first time, the entire package is loaded into memory. This reduces the disk access for subsequent calls.