Creating Packages

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

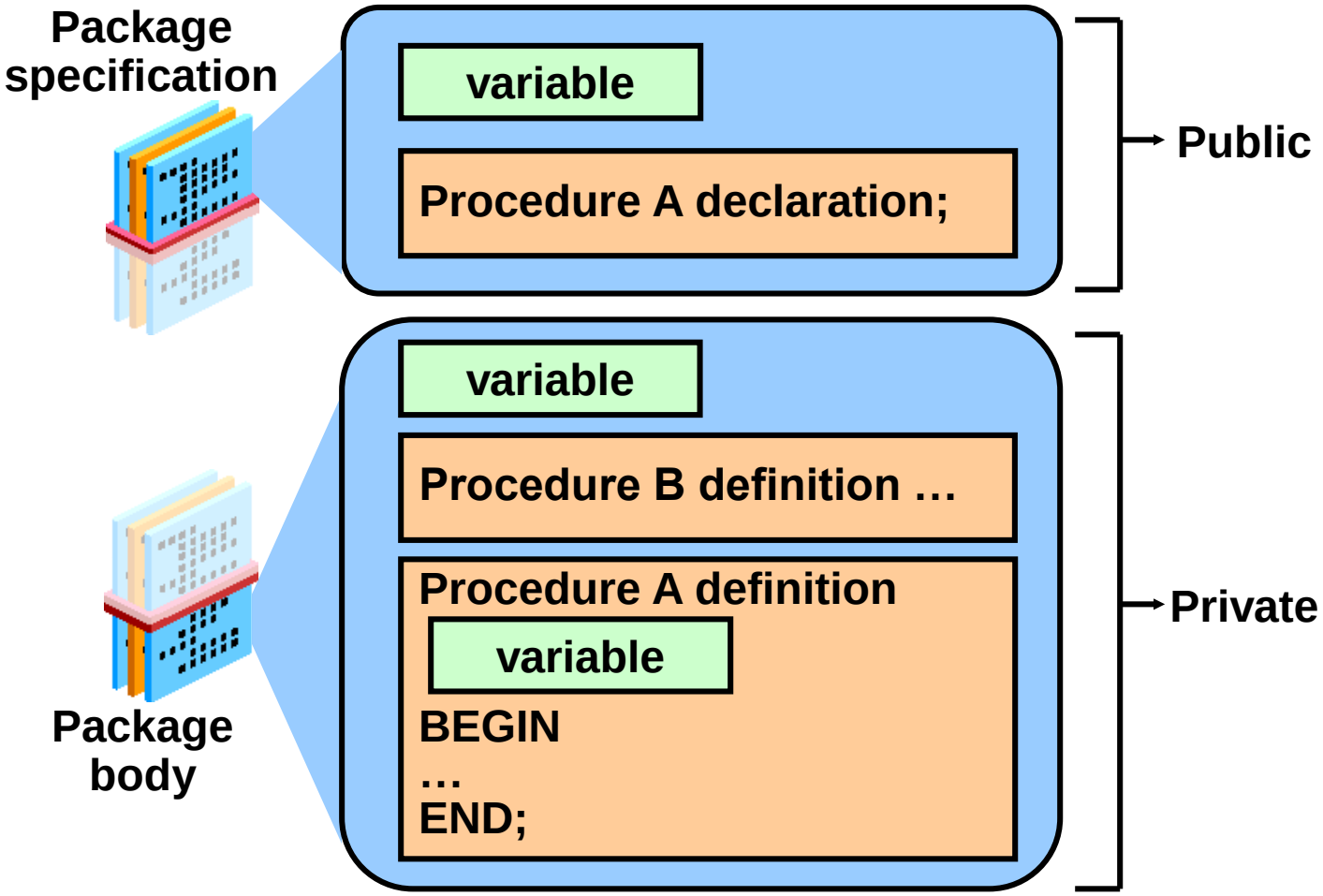
PL/SQL Packages: Overview

PL/SQL packages:

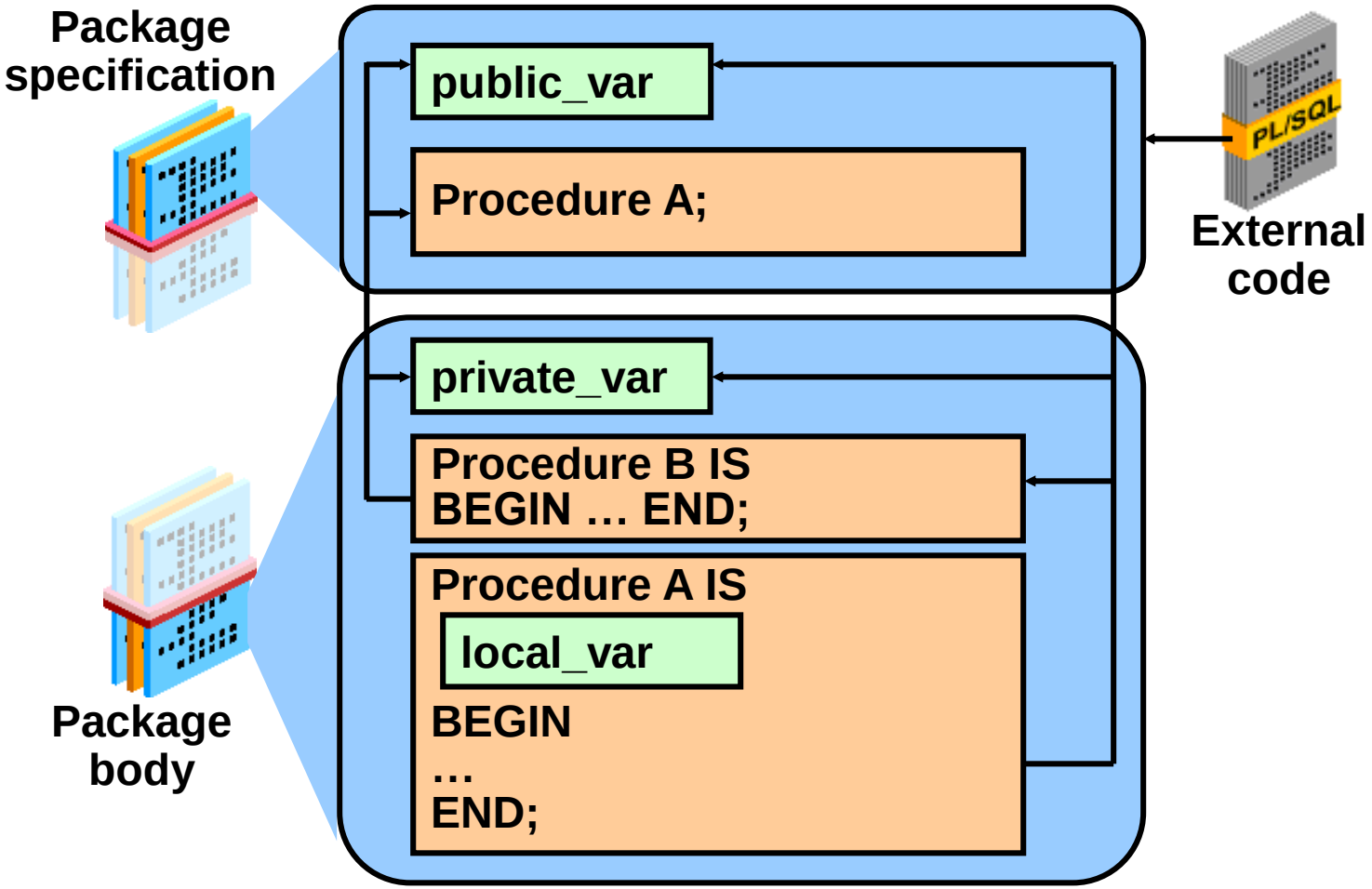
- **Group logically related components:**
  - **PL/SQL types**
  - **Variables, data structures, and exceptions**
  - **Subprograms: Procedures and functions**
- **Consist of two parts:**
  - **A specification**
  - **A body**

- **Enable the Oracle server to read multiple objects into memory at once**

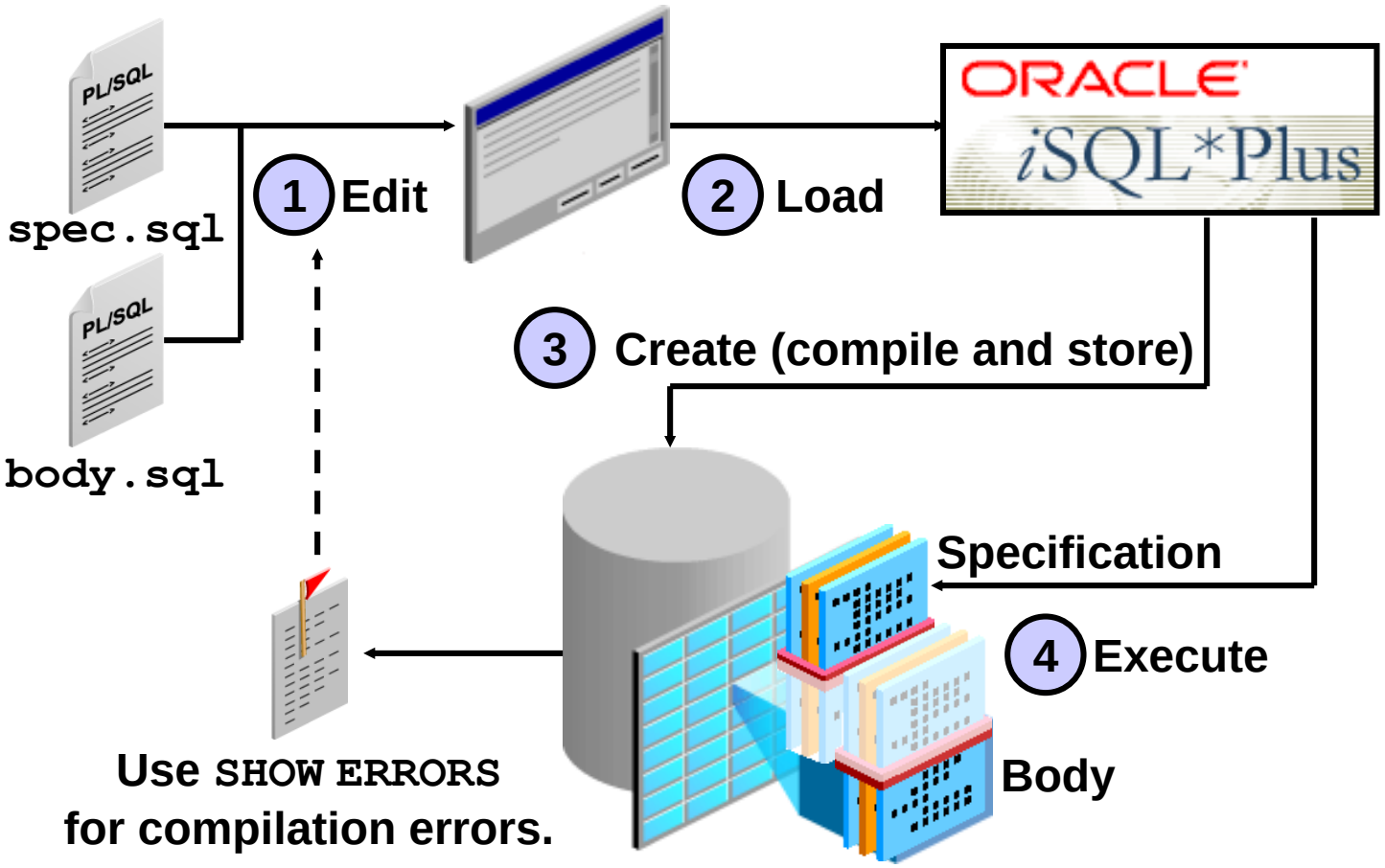
Components of a PL/SQL Package



Visibility of Package Components



Developing PL/SQL Packages



Creating the Package Specification

Syntax:

```
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.

Example of Package Specification: comm_pkg

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

- **STD_COMM is a 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.**

Creating the Package Body

Syntax:

```
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.

Example of Package Body: comm_pkg

```
CREATE OR REPLACE 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);
  END validate;
  PROCEDURE reset_comm (new_comm NUMBER) IS BEGIN
    IF validate(new_comm) THEN
      std_comm := new_comm; -- reset public var
    ELSE  RAISE_APPLICATION_ERROR(
            -20210, 'Bad Commission');
    END IF;
  END reset_comm;
END comm_pkg;
```

Invoking Package Subprograms

- Invoke a function within the same package:

```
CREATE OR REPLACE PACKAGE BODY comm_pkg IS ...
  PROCEDURE reset_comm(new_comm NUMBER) IS
  BEGIN
    IF validate(new_comm) THEN
      std_comm := new_comm;
    ELSE ...
    END IF;
  END reset_comm;
END comm_pkg;
```

- Invoke a package procedure from *i*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)
```

Creating and Using Bodiless Packages

```
CREATE OR REPLACE PACKAGE global_consts IS
  mile_2_kilo    CONSTANT  NUMBER  :=  1.6093;
  kilo_2_mile    CONSTANT  NUMBER  :=  0.6214;
  yard_2_meter   CONSTANT  NUMBER  :=  0.9144;
  meter_2_yard   CONSTANT  NUMBER  :=  1.0936;
END global_consts;
```

```
BEGIN  DBMS_OUTPUT.PUT_LINE('20 miles = ' ||
        20 * global_consts.mile_2_kilo || ' km');
END;
```

```
CREATE FUNCTION mtr2yrd(m NUMBER) RETURN NUMBER IS
BEGIN
  RETURN (m * global_consts.meter_2_yard);
END mtr2yrd;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(mtr2yrd(1))
```

Removing Packages

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

```
DROP PACKAGE package_name;
```

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

```
DROP PACKAGE BODY package_name;
```

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 = 'COMM_PKG' AND type = 'PACKAGE';
```

- To view the package body, use:

```
SELECT text
FROM   user_source
WHERE  name = 'COMM_PKG' AND type = 'PACKAGE BODY';
```

Guidelines for Writing Packages

- Construct packages for general use.
- Define the package specification before the body.
- The package specification should contain only those constructs that you want to be public.
- Place items in the declaration part of the package body when you must maintain them throughout a session or across transactions.
- Changes to the package specification require recompilation of each referencing subprogram.
- The package specification should contain as few constructs as possible.

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.

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 of the same name**

Summary

In this lesson, you should have learned how to:

- Improve code organization, management, security, and performance by using packages
- Create and remove package specifications and bodies
- Group related procedures and functions together in a package
- Encapsulate the code in a package body
- Define and use components in bodiless packages
- Change a package body without affecting a package specification

Summary

Command	Task
`CREATE [OR REPLACE] PACKAGE`	Create (or modify) an existing package specification.
`CREATE [OR REPLACE] PACKAGE BODY`	Create (or modify) an existing package body.
`DROP PACKAGE`	Remove both the package specification and package body. Remove only the package body.
`DROP PACKAGE BODY`	