3

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

Lesson Aim

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

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

ORACLE

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.

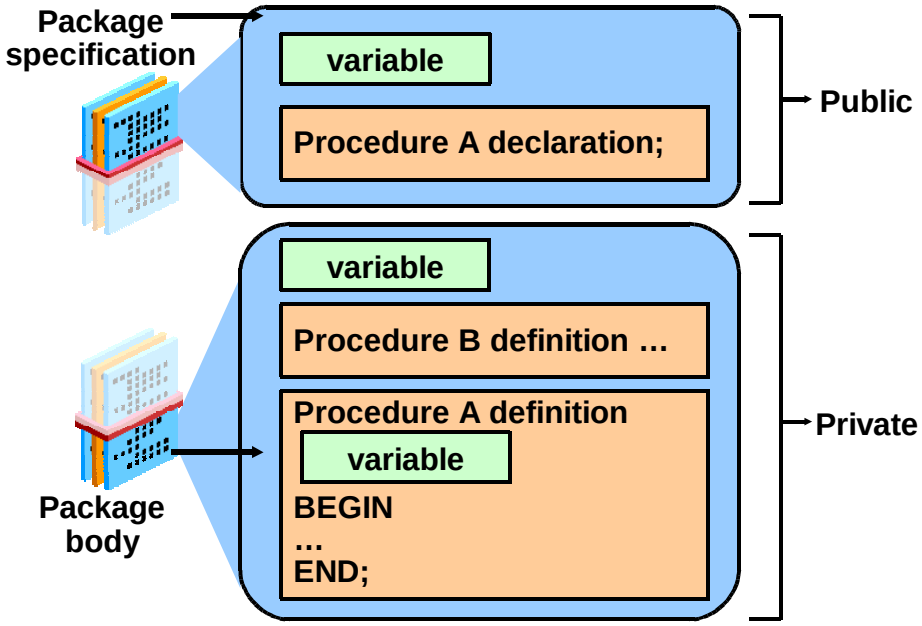
A package usually consists of two parts stored separately in the database:

- 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 do not require disk input/output (I/O).

Components of a PL/SQL Package



Components of a PL/SQL Package

You create a package in two parts:

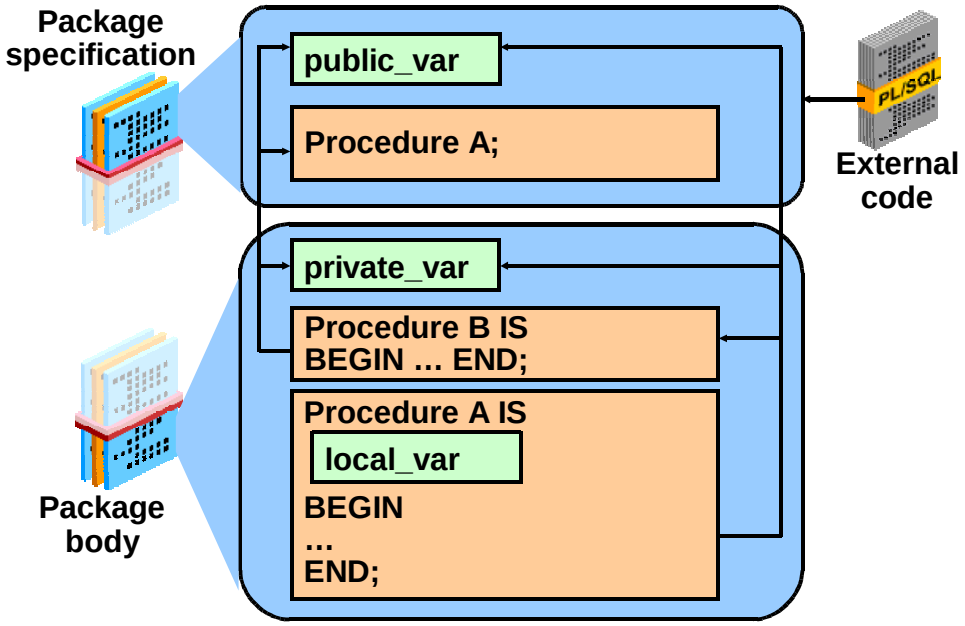
- The package specification is the interface to your applications. It declares the public types, variables, constants, exceptions, cursors, and subprograms available for use. The package specification may also include `PRAGMA`s, which are directives to the compiler.
- The package body defines its own 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 package public components.

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

Visibility of Package Components



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. The visibility of components depends on whether they are locally or globally declared.

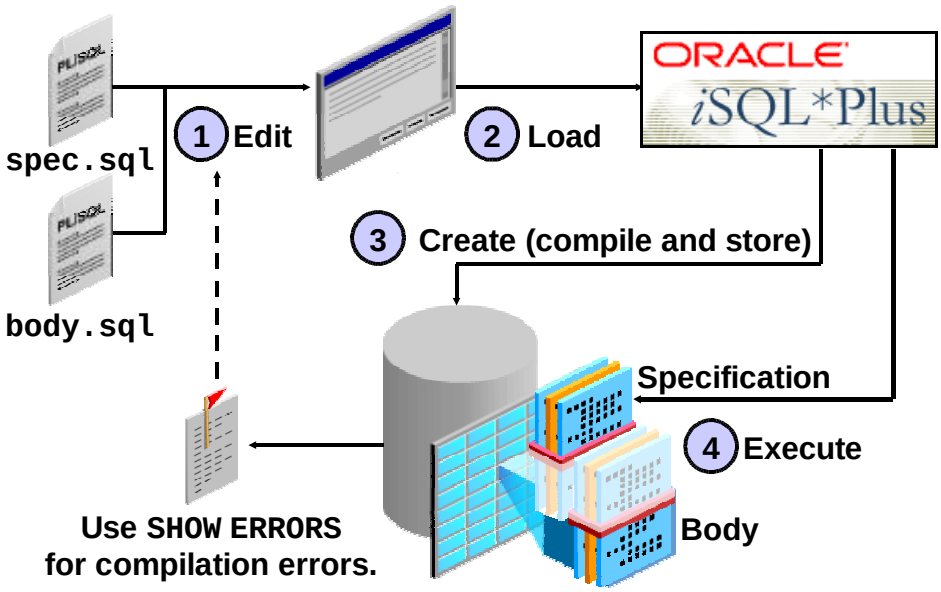
Local components are visible within the structure in which they are declared, such as the following:

- Variables defined in a subprogram can be referenced within that subprogram, and are not visible to external components—for example, `local_var` can be used in procedure `A`.
- Private package variables, which 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. For example, `private_var` can be used by procedures `A` and `B` within the package body, but not outside the package.

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

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

Developing PL/SQL Packages



ORACLE

Developing PL/SQL Packages

To develop a package, perform the following steps:

1. Edit the text for the specification by using the `CREATE PACKAGE` statement within a SQL script file. Edit the text for the body (only if required; see the guidelines below) by using the `CREATE PACKAGE BODY` statement within a SQL script file.
2. Load the script files into a tool such as *i*SQL*Plus.
3. Execute the script files to create (that is, to compile and store) the package and package body in the database.
4. Execute any public construct within the package specification from an Oracle server environment.

Guidelines for Developing Packages

- Consider saving the text for a package specification and a package body in two different script files to facilitate easier modifications to the package or its body.
- A package specification can exist without a package body—that is, when the package specification does not declare subprograms, a body is not required. However, a package body cannot exist without a package specification.

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.

ORACLE

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.

Note: The package specification should contain procedure and function headings terminated by a semicolon, without the IS (or AS) keyword and its

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.

ORACLE

Example of Package Specification: comm_pkg

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

The 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

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.

ORACLE

Creating the Package Body

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.

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;
```


Example of Package Body: comm_pkg

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 lesser than the highest commission among existing employees. The reset_comm procedure invokes the private validation function before changing the standard commission in std_comm. In the example, note the following:

- The std_comm variable referenced in the reset_comm procedure is a public variable. Variables declared in the package specification, such as 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

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)
```


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 *i*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 in a schema user called `SCOTT`. Using *i*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

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))
```

ORACLE

Creating and Using Bodiless Packages

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.

The second code box references the `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 `mtr2yrd` to convert meters to yards, and uses the constant conversion rate `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

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;
```


Removing Packages

When a package is no longer required, you can use a SQL statement in *i*SQL*Plus 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 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';
```

ORACLE

Viewing Packages in the Data Dictionary

The source code for PL/SQL packages is also stored in the data dictionary tables such as 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

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

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.

ORACLE

Guidelines for Writing Packages

Keep your packages as general as possible, so that they can be reused in future applications. Also, avoid writing packages that duplicate features provided by the Oracle server.

Package specifications reflect the design of your application, so define them before defining the package bodies. The package specification should contain only those constructs that must be visible to the users of the package. Thus, other developers cannot misuse the package by basing code on irrelevant details.

Place items in the declaration part of the package body when you must maintain them throughout a session or across transactions. For example, declare a variable called `NUMBER_EMPLOYED` as a private variable if each call to a procedure that uses the variable needs to be maintained. When declared as a global variable in the package specification, the value of that global variable is initialized in a session the first time a construct from the package is invoked.

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.

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

Advantages of Using Packages

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.

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

Advantages of Using Packages (continued)

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."

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

ORACLE

Summary

You group related procedures and function together 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.

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.
DROP PACKAGE BODY	Remove only the package body.

ORACLE

Summary (continued)

You can create, delete, and modify packages. You can remove both package specification and body by using the DROP PACKAGE command. You can drop the package body without affecting its specification.

Practice 3: Overview

This practice covers the following topics:

- **Creating packages**
- **Invoking package program units**

Practice 3: Overview

In this practice, you create package specifications and package bodies. You invoke the constructs in the packages by using sample data.

Note: If you are using SQL Developer, your compile time errors are displayed in the Message Log. If you are using SQL*Plus or *i*SQL*Plus to create your stored code, use the `SHOW ERRORS` to view compile errors.

Practice 3

1. **Create a package specification and body called `JOB_PKG`, containing a copy of your `ADD_JOB`, `UPD_JOB`, and `DEL_JOB` procedures as well as your `GET_JOB` function.**
   Tip: Consider saving the package specification and body in two separate files (for example, `p3q1_s.sql` and `p3q1_b.sql` for the package specification and body, respectively). Include a `SHOW ERRORS` after the `CREATE PACKAGE` statement in each file. Alternatively, place all code in one file.
   Note: Use the code in your previously saved script files when creating the package.

   a. **Create the package specification including the procedures and function headings as public constructs.**

      Note: Consider whether you still need the stand-alone procedures and functions you just packaged.

   b. | JOB_ID | JOB_TITLE | MIN_SALARY | MAX_SALARY |
      |---|---|---|---|
      | IT_SYSAN | Systems Analyst | | |

      the

   c. **Invoke your `ADD_JOB` package procedure by passing the values `IT_SYSAN` and `SYSTEMS ANALYST` as parameters.**

   d. **Query the `JOBS` table to see the result.**

2. **Create and invoke a package that contains private and public constructs.**

   a. **Create a package specification and package body called `EMP_PKG` that contains your `ADD_EMPLOYEE` and `GET_EMPLOYEE` procedures as public constructs, and include your `VALID_DEPTID` function as a private construct.**

   b. **Invoke the `EMP_PKG.ADD_EMPLOYEE` procedure, using department ID `15` for employee `Jane Harris` with the e-mail ID `JAHARRIS`. Because department ID 15 does not exist, you should get an error message as specified in the exception handler of your procedure.**

   c. **Invoke the `ADD_EMPLOYEE` package procedure by using department ID `80` for employee `David Smith` with the e-mail ID `DASMITH`.**