# Practices for Lesson 6: Working with Packages

**Chapter 6**

## Practices for Lesson 6: Overview

### Overview

In this practice, you modify an existing package to contain overloaded subprograms and you use forward declarations. You also create a package initialization block within a package body to populate a PL/SQL table.

**Note:**

1. Before starting this practice, execute `/home/oracle/labs/plpu/code_ex/cleanup_scripts/cleanup_06.sql` script.
2. If you missed a step in a practice, please run the appropriate solution script for that practice step before proceeding to the next step or the next practice.

## Practice 6-1: Working with Packages

### Overview

In this practice, you modify the code for the EMP_PKG package that you created earlier, and then overload the ADD_EMPLOYEE procedure. Next, you create two overloaded functions called GET_EMPLOYEE in the EMP_PKG package. You also add a public procedure to EMP_PKG to populate a private PL/SQL table of valid department IDs and modify the VALID_DEPTID function to use the private PL/SQL table contents to validate department ID values. You also change the VALID_DEPTID validation processing function to use the private PL/SQL table of department IDs. Finally, you reorganize the subprograms in the package specification and the body so that they are in alphabetical sequence.

**Note:** Execute `cleanup_06.sql script` from `/home/oracle/labs/plpu/code_ex/cleanup_scripts/` before performing the following tasks.

### Task

1. Modify the code for the EMP_PKG package that you created in Practice 5, and overload the ADD_EMPLOYEE procedure.
   a. In the package specification, add a new procedure called ADD_EMPLOYEE that accepts the following three parameters:
      1) First name
      2) Last name
      3) Department ID
   b. Click the Run Script icon (or press F5) to create and compile the package.
   c. Implement the new ADD_EMPLOYEE procedure in the package body as follows:
      1) Format the email address in uppercase characters, using the first letter of the first name concatenated with the first seven letters of the last name.
      2) The procedure should call the existing ADD_EMPLOYEE procedure to perform the actual INSERT operation using its parameters and formatted email to supply the values.
      3) Click Run Script to create the package. Compile the package.
   d. Invoke the new ADD_EMPLOYEE procedure using the name `Samuel Joplin` to be added to department 30.
   e. Confirm that the new employee was added to the EMPLOYEES table.
2. In the EMP_PKG package, create two overloaded functions called GET_EMPLOYEE:
   a. In the package specification, add the following functions:
      1) The GET_EMPLOYEE function that accepts the parameter called `p_emp_id` based on the `employees.employee_id%TYPE` type. This function should return `EMPLOYEES%ROWTYPE`.
      2) The GET_EMPLOYEE function that accepts the parameter called `p_family_name` of type `employees.last_name%TYPE`. This function should return `EMPLOYEES%ROWTYPE`.
   b. Click Run Script to re-create and compile the package.
   c. In the package body:

1) Implement the first `GET_EMPLOYEE` function to query an employee using the employee's ID.
2) Implement the second `GET_EMPLOYEE` function to use the equality operator on the value supplied in the `p_family_name` parameter.

d. Click Run Script to re-create and compile the package.

e. Add a utility procedure `PRINT_EMPLOYEE` to the `EMP_PKG` package as follows:

1) The procedure accepts an `EMPLOYEES%ROWTYPE` as a parameter.
2) The procedure displays the following for an employee on one line, using the `DBMS_OUTPUT` package:
   - `department_id`
   - `employee_id`
   - `first_name`
   - `last_name`
   - `job_id`
   - `salary`

f. Click the Run Script icon (or press F5) to create and compile the package.

g. Use an anonymous block to invoke the `EMP_PKG.GET_EMPLOYEE` function with an employee ID of `100` and family name of `'Joplin'`. Use the `PRINT_EMPLOYEE` procedure to display the results for each row returned.

3. Because the company does not frequently change its departmental data, you can improve performance of your `EMP_PKG` by adding a public procedure, `INIT_DEPARTMENTS`, to populate a private PL/SQL table of valid department IDs. Modify the `VALID_DEPTID` function to use the private PL/SQL table contents to validate department ID values.

**Note:** The code under Task 3 contains the solution for steps a, b, and c.

a. In the package specification, create a procedure called `INIT_DEPARTMENTS` with no parameters by adding the following to the package specification section before the `PRINT_EMPLOYEES` specification:

```
PROCEDURE init_departments;
```

b. In the package body, implement the `INIT_DEPARTMENTS` procedure to store all department IDs in a private PL/SQL index-by table named `valid_departments` containing `BOOLEAN` values.

1) Declare the `valid_departments` variable and its type definition `boolean_tab_type` before all procedures in the body. Enter the following at the beginning of the package body:

```
TYPE boolean_tab_type IS TABLE OF BOOLEAN
INDEX BY BINARY_INTEGER;
valid_departments boolean_tab_type;
```

2) Use the `department_id` column value as the index to create the entry in the index-by table to indicate its presence, and assign the entry a value of `TRUE`. Enter the `INIT_DEPARTMENTS` procedure declaration at the end of the package body (right after the `print_employees` procedure) as follows:

```
PROCEDURE init_departments IS
BEGIN
```

```
          FOR rec IN (SELECT department_id FROM departments)
            LOOP
              valid_departments(rec.department_id) := TRUE;
            END LOOP;
        END;
```

c. In the body, create an initialization block that calls the `INIT_DEPARTMENTS` procedure to initialize the table as follows:

```
BEGIN
  init_departments;
END;
```

d. Click the Run Script icon (or press F5) to create and compile the package.

4. Change the `VALID_DEPTID` validation processing function to use the private index-by table of department IDs.
   a. Modify the `VALID_DEPTID` function to perform its validation by using the index-by table of department ID values. Click the Run Script icon (or press F5) to create the package. Compile the package.
   b. Test your code by calling `ADD_EMPLOYEE` using the name `James Bond` in department `15`. What happens?
   c. Insert a new department. Specify 15 for the department ID and `'Security'` for the department name. Commit and verify the changes.
   d. Test your code again, by calling `ADD_EMPLOYEE` using the name `James Bond` in department 15. What happens?
   e. Execute the `EMP_PKG.INIT_DEPARTMENTS` procedure to update the internal index-by table with the latest departmental data.
   f. Test your code by calling `ADD_EMPLOYEE` by using the employee name `James Bond`, who works in department `15`. What happens?
   g. Delete employee `James Bond` and department `15` from their respective tables, commit the changes, and refresh the department data by invoking the `EMP_PKG.INIT_DEPARTMENTS` procedure. Make sure you enter `SET SERVEROUTPUT ON` first.

5. Reorganize the subprograms in the package specification and the body so that they are in alphabetical sequence.
   – Edit the package specification and reorganize subprograms alphabetically. Click Run Script to re-create the package specification. Compile the package specification. What happens?
   – Edit the package body and reorganize all subprograms alphabetically. Click Run Script to re-create the package specification. Re-compile the package specification. What happens?
   – Correct the compilation error using a forward declaration in the body for the appropriate subprogram reference. Click Run Script to re-create the package, and then recompile the package. What happens?