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?

Solution 6-1: Working with Packages

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.

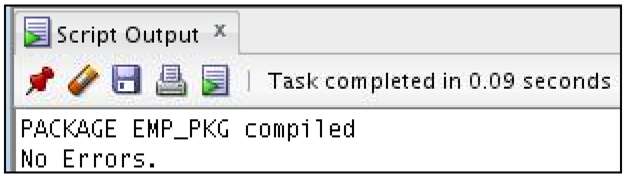
1. Modify the code for the EMP_PKG package that you created in Practice 5 step 2, and overload the ADD_EMPLOYEE procedure.
   a. In the package specification, add a new procedure called ADD_EMPLOYEE that accepts the following three parameters:
      1) First name
      2) Last name
      3) Department ID

      Open the /home/oracle/labs/plpu/solns/sol_06.sql file. Uncomment and select the code under Task 1_a. The code is displayed as follows:

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

-- New overloaded add_employee

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);
END emp_pkg;
/
```

```
SHOW ERRORS
```

b. **Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to create and compile the package.**

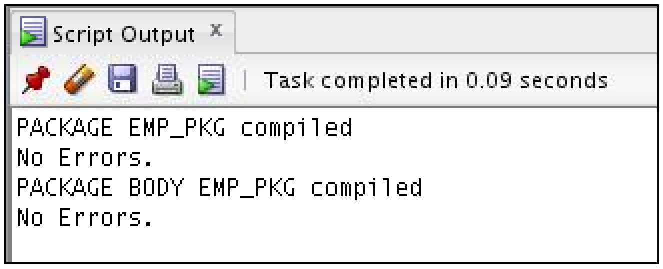


```
PACKAGE EMP_PKG compiled
No Errors.
```

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.

Uncomment and select the code under Task 1_c. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to invoke the package's procedure. The code and the result are displayed as follows (the newly added code is highlighted in bold face text in the code box below):

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

-- New overloaded add_employee

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

-- End of the spec of the new overloaded add_employee

PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);
END emp_pkg;
/
SHOW ERRORS
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
  BEGIN
    SELECT 1
    INTO v_dummy
    FROM departments
    WHERE department_id = p_deptid;
    RETURN TRUE;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END valid_deptid;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,


    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS


BEGIN
  IF valid_deptid(p_deptid) THEN
    INSERT INTO employees(employee_id, first_name, last_name,
      email, job_id, manager_id, hire_date, salary,
      commission_pct, department_id)
      VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
      p_email, p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
      p_deptid);
  ELSE
    RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID. Try
     again.');
  END IF;
  END add_employee;

-- New overloaded add_employee procedure

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
  BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
                     1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
                 p_deptid);
  END;

-- End declaration of the overloaded add_employee procedure
  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
  BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p_empid;
  END get_employee;
END emp_pkg;
/
SHOW ERRORS
```



Script Output

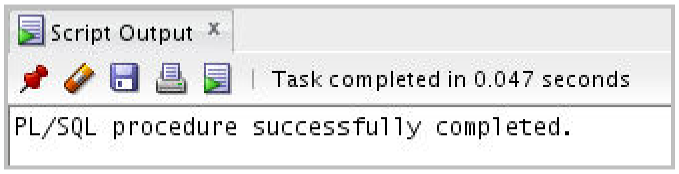
Task completed in 0.09 seconds

```
PACKAGE EMP_PKG compiled
No Errors.
PACKAGE BODY EMP_PKG compiled
No Errors.
```

d. Invoke the new `ADD_EMPLOYEE` procedure using the name `Samuel Joplin` to be added to department 30.

Uncomment and select the code under Task 1_d. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to invoke the package's procedure. The code and the result are displayed as follows:

```
EXECUTE emp_pkg.add_employee('Samuel', 'Joplin', 30)
```



Script Output ×

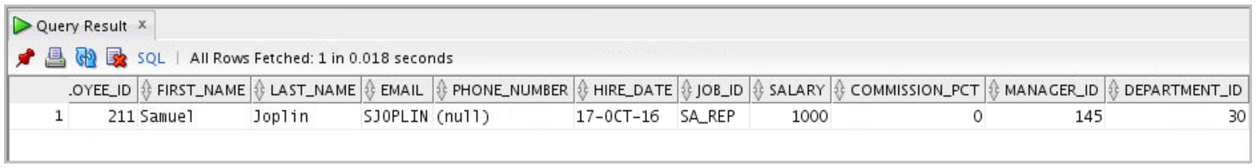
Task completed in 0.047 seconds

```
PL/SQL procedure successfully completed.
```

e. Confirm that the new employee was added to the `EMPLOYEES` table.

Uncomment and select the code under Task 1_e. Click anywhere on the `SELECT` statement, and then click the Execute Statement icon (or press F5) on the SQL Worksheet toolbar to execute the query. The code and the result are displayed as follows:

```
SELECT *
FROM employees
WHERE last_name = 'Joplin';
```



Query Result ×

SQL | All Rows Fetched: 1 in 0.018 seconds

	OYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	211	Samuel	Joplin	SJOPLIN	(null)	17-OCT-16	SA_REP	1000	0	145	30

2. In the `EMP_PKG` package, create two overloaded functions called `GET_EMPLOYEE`:

   a. In the package specification, add the following functions:

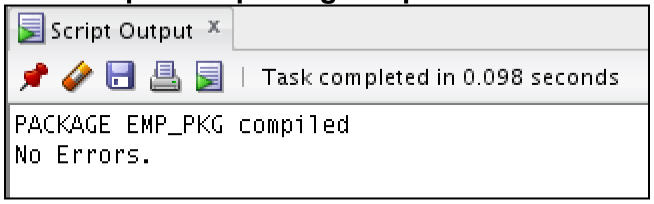
      1) The `GET_EMPLOYEE` function that accepts the parameter called `p_emp_id` based on the `employees.employee_id%TYPE` type. This function should return `EMPLOYEES%ROWTYPE`.

      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`.

      Uncomment and select the code under Task 2_a.

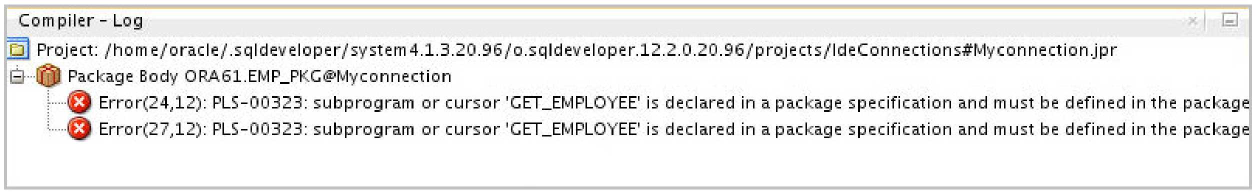
```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

-- New overloaded get_employees functions specs starts here:

  FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

  FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype;

-- New overloaded get_employees functions specs ends here.

END emp_pkg;
/
SHOW ERRORS
```

b. Click Run Script to re-create and compile the package specification.
Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to re-create and compile the package's specification. The result is shown below:



```
PACKAGE EMP_PKG compiled
No Errors.
```

Note: As mentioned earlier, if your code contains an error message, you can recompile the code using the following procedure to view the details of the error or warning in the Compiler – Log tab: To compile the package specification, right-click the package's specification (or the entire package) name in the Object Navigator tree, and then select Compile from the shortcut menu. The warning is expected and is for informational purposes only.



```
Compiler - Log
Project: /home/oracle/.sqldeveloper/system4.1.3.20.96/o.sqldeveloper.12.2.0.20.96/projects/IdeConnections#Myconnection.jpr
Package Body ORA61.EMP_PKG@Myconnection
    Error(24,12): PLS-00323: subprogram or cursor 'GET_EMPLOYEE' is declared in a package specification and must be defined in the package
    Error(27,12): PLS-00323: subprogram or cursor 'GET_EMPLOYEE' is declared in a package specification and must be defined in 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.

Uncomment and select the code under Task 2_c. The newly added functions are highlighted in the following code box.

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);


PROCEDURE get_employee(


    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

-- New overloaded get_employees functions specs starts here:

  FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;
  FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype;

-- New overloaded get_employees functions specs ends here.

END emp_pkg;
/
SHOW ERRORS
```

-- package body

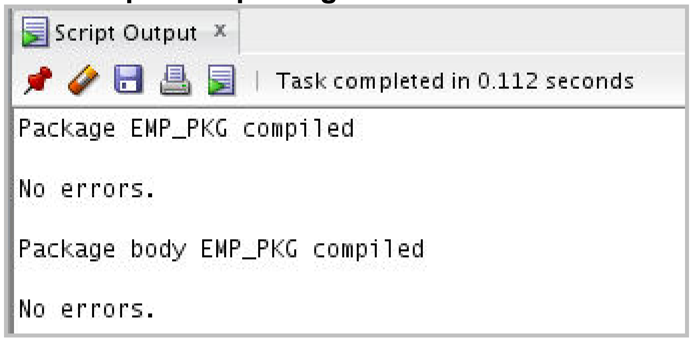
```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
  BEGIN
    SELECT 1
    INTO v_dummy
    FROM departments
    WHERE department_id = p_deptid;
    RETURN TRUE;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END valid_deptid;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
  BEGIN
    IF valid_deptid(p_deptid) THEN
      INSERT INTO employees(employee_id, first_name,
last_name,
   email, job_id, manager_id, hire_date, salary,
   commission_pct, department_id)
```

```
      VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name,
        p_email, p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
        p_deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID.
                            Try again.');
    END IF;
  END add_employee;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
  BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid
=> p_deptid);
  END;

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
  BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p_empid;
  END get_employee;
```

-- New get_employee function declaration starts here

```
FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
  END;

  FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
  END;

-- New overloaded get_employee function declaration ends here

END emp_pkg;
/
SHOW ERRORS
```

d. Click Run Script to re-create the package. Compile the package.
Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to re-create and compile the package. The result is shown below:



```
Package EMP_PKG compiled

No errors.

Package body EMP_PKG compiled

No errors.
```

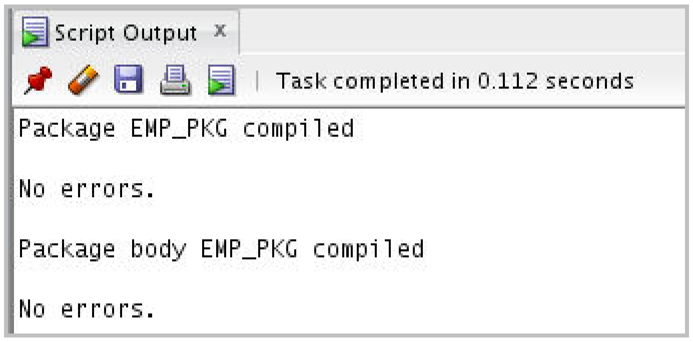
e. Add a utility procedure `PRINT_EMPLOYEE` to the `EMP_PKG` package as follows:
1) The procedure accepts an `EMPLOYEES%ROWTYPE` as a parameter.
2) The procedure displays the following for an employee on one line, by using the `DBMS_OUTPUT` package:
   - `department_id`
   - `employee_id`
   - `first_name`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

Uncomment and select the code under Task 2_e. The newly added code is highlighted in the following code box.

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

  FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

  FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype;

-- New print_employee print_employee procedure spec

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
```

```
  FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
  BEGIN
    SELECT 1
    INTO v_dummy
    FROM departments
    WHERE department_id = p_deptid;
    RETURN TRUE;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END valid_deptid;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
  BEGIN
    IF valid_deptid(p_deptid) THEN
      INSERT INTO employees(employee_id, first_name, last_name,
email,
        job_id, manager_id, hire_date, salary, commission_pct,
department_id)
      VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
        p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
    END IF;
  END add_employee;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
  BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
  END;

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
  BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p_empid;
  END get_employee;

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
  END;

  FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
  END;

-- New print_employees procedure declaration.

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id ||' '||
                                          p_rec_emp.employee_id||' '||
                                          p_rec_emp.first_name||' '||
                                          p_rec_emp.last_name||' '||
                                          p_rec_emp.job_id||' '||
                                          p_rec_emp.salary);
  END;

END emp_pkg;
/
SHOW ERRORS
```

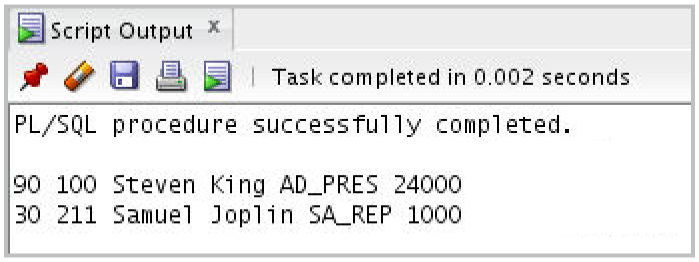
f. Click the Run Script icon (or press F5) to create and compile the package.
   Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to re-create and compile the package.



```
Package EMP_PKG compiled

No errors.

Package body EMP_PKG compiled

No errors.
```

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. Make sure you enter `SET SERVEROUTPUT ON` first.
   Uncomment and select the code under Task 2_g.

```
SET SERVEROUTPUT ON
BEGIN
  emp_pkg.print_employee(emp_pkg.get_employee(100));
  emp_pkg.print_employee(emp_pkg.get_employee('Joplin'));
END;
/
```



```
PL/SQL procedure successfully completed.

90 100 Steven King AD_PRES 24000
30 211 Samuel Joplin SA_REP 1000
```

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 solutions 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;
   ```

   Uncomment and select the code under Task 3. The newly added code is highlighted in the following code box:

   ```
   -- Package SPECIFICATION

   CREATE OR REPLACE PACKAGE emp_pkg IS
     PROCEDURE add_employee(
       p_first_name employees.first_name%TYPE,
       p_last_name employees.last_name%TYPE,
       p_email employees.email%TYPE,
   ```

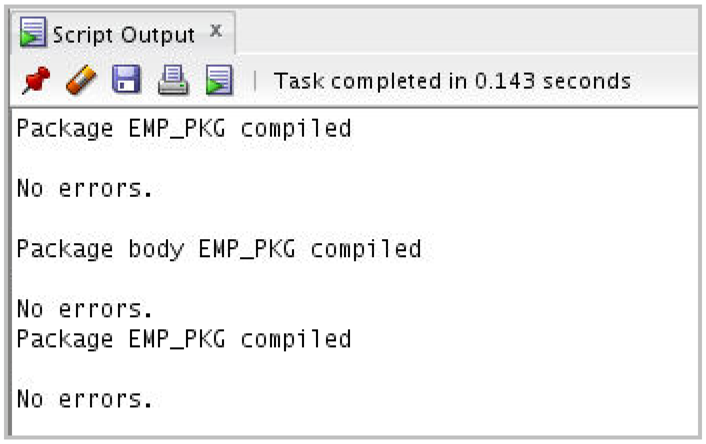
```
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

  FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

  FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype;

-- New procedure init_departments spec
PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS


-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS

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

```
FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
  BEGIN
    SELECT 1
    INTO v_dummy
    FROM departments
    WHERE department_id = p_deptid;
    RETURN TRUE;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END valid_deptid;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
  BEGIN
    IF valid_deptid(p_deptid) THEN

INSERT INTO employees(employee_id, first_name, last_name,
   email, job_id, manager_id, hire_date, salary,
   commission_pct, department_id)
   VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
      p_email, p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
      p_deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
                               Try again.');
    END IF;
  END add_employee;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
  BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
  END;

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
  BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p_empid;
  END get_employee;

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
  END;

  FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
  END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id ||' '||
                         P_rec_emp.employee_id||' '||
                         P_rec_emp.first_name||' '||
                          P_rec_emp.last_name||' '||
                          P_rec_emp.job_id||' '||
                          P_rec_emp.salary);
  END;

-- New init_departments procedure declaration.

PROCEDURE init_departments IS
  BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
      valid_departments(rec.department_id) := TRUE;
    END LOOP;
  END;

-- call the new init_departments procedure.

BEGIN
  init_departments;
END emp_pkg;
/
SHOW ERRORS

CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

  FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

  FUNCTION get_employee(p_family_name
      employees.last_name%type)
    return employees%rowtype;
```

```
--New procedure init_departments spec

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS
```

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

Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to re-create and compile the package.



```
Package EMP_PKG compiled

No errors.

Package body EMP_PKG compiled

No errors.
Package EMP_PKG compiled

No errors.
```

4. Change the `VALID_DEPTID` validation processing function to use the private PL/SQL table of department IDs.

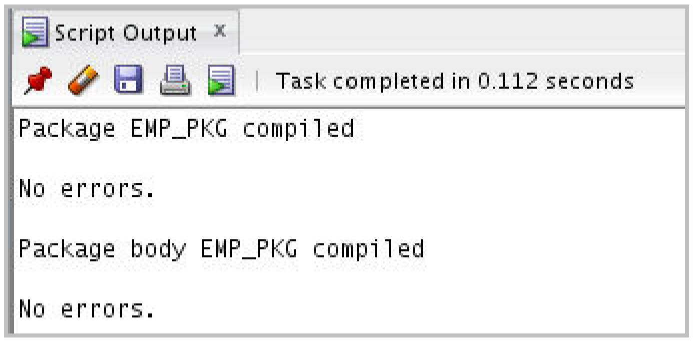
a. Modify the `VALID_DEPTID` function to perform its validation by using the PL/SQL table of department ID values. Click the Run Script icon (or press F5) to create and compile the package.

Uncomment and select the code under Task 4_a. Click the Run Script icon (or press F5) to create and compile the package. The newly added code is highlighted in the following code box.

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

  FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

  FUNCTION get_employee(p_family_name
      employees.last_name%type)
    return employees%rowtype;

-- New procedure init_departments spec
PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS



-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS

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

  FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
  BEGIN
    RETURN valid_departments.exists(p_deptid);
  EXCEPTION
```

```
    WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END valid_deptid;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
  BEGIN
    IF valid_deptid(p_deptid) THEN
      INSERT INTO employees(employee_id, first_name,
        last_name, email, job_id, manager_id, hire_date,
        salary, commission_pct, department_id)
      VALUES (employees_seq.NEXTVAL, p_first_name,
        p_last_name, p_email,
        p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,p_deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
                              Try again.');
    END IF;
  END add_employee;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
  BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
  END;

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
  BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p_empid;
  END get_employee;

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
  END;

  FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
  END;
PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id ||' '||
                         p_rec_emp.employee_id||' '||
                         p_rec_emp.first_name||' '||
                         p_rec_emp.last_name||' '||
                         p_rec_emp.job_id||' '||
                         p_rec_emp.salary);
  END;

-- New init_departments procedure declaration.

PROCEDURE init_departments IS
  BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
```

```
        valid_departments(rec.department_id) := TRUE;
      END LOOP;
    END;

-- call the new init_departments procedure.
BEGIN
  init_departments;
END emp_pkg;

/
SHOW ERRORS
```



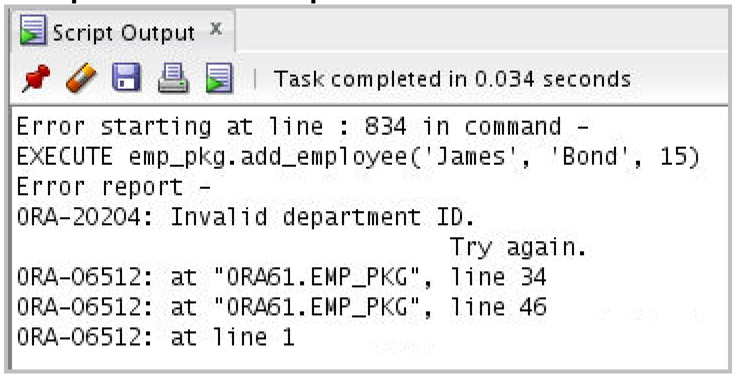
```
Package EMP_PKG compiled

No errors.

Package body EMP_PKG compiled

No errors.
```

b. Test your code by calling `ADD_EMPLOYEE` using the name `James Bond` in department `15`. What happens?

Uncomment and select the code under Task 4_b.

```
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)
```

Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to test inserting a new employee. The insert operation to add the employee fails with an exception because department 15 does not exist.



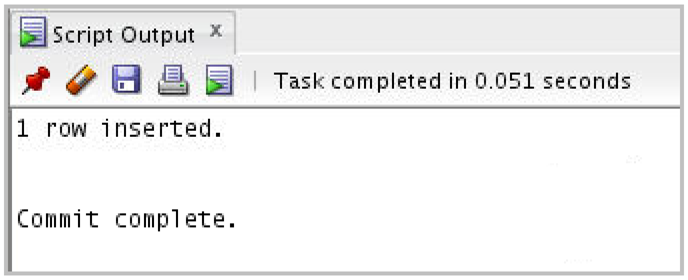
```
Error starting at line : 834 in command -
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)
Error report -
ORA-20204: Invalid department ID.
                              Try again.
ORA-06512: at "ORA61.EMP_PKG", line 34
ORA-06512: at "ORA61.EMP_PKG", line 46
ORA-06512: at line 1
```

c. Insert a new department. Specify 15 for the department ID and `'Security'` for the department name. Commit and verify the changes.

Uncomment and select the code under Task 4_c. The code and result are displayed as follows:

```
INSERT INTO departments (department_id, department_name)
VALUES (15, 'Security');
```

```
COMMIT;
```

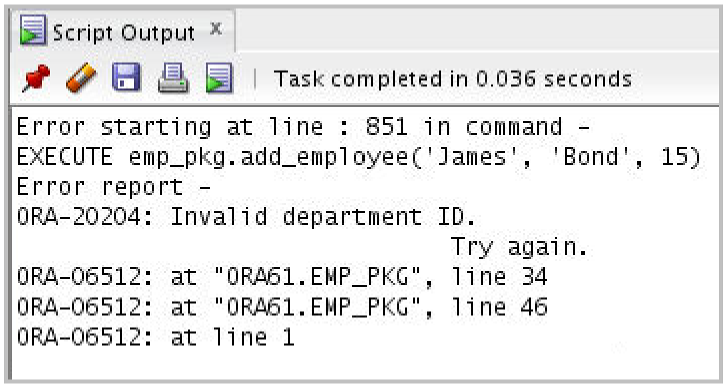


```
1 row inserted.


Commit complete.
```

d. Test your code again, by calling `ADD_EMPLOYEE` using the name `James Bond` in department 15. What happens?

Uncomment and select the code under Task 4_d. The code and the result are displayed as follows:

```
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)
```



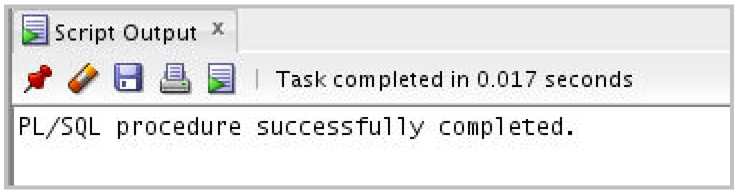
```
Error starting at line : 851 in command -
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)
Error report -
ORA-20204: Invalid department ID.
                              Try again.
ORA-06512: at "ORA61.EMP_PKG", line 34
ORA-06512: at "ORA61.EMP_PKG", line 46
ORA-06512: at line 1
```

The insert operation to add the employee fails with an exception. Department 15 does not exist as an entry in the PL/SQL associative array (index-by table) package state variable.

e. Execute the `EMP_PKG.INIT_DEPARTMENTS` procedure to update the index-by table with the latest departmental data.

Uncomment and select the code under Task 4_e. The code and result are displayed as follows:

```
EXECUTE EMP_PKG.INIT_DEPARTMENTS
```



```
PL/SQL procedure successfully completed.
```

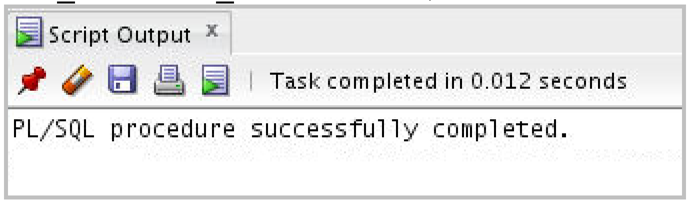
f. Test your code by calling `ADD_EMPLOYEE` using the employee name James Bond, who works in department 15. What happens?

Uncomment and select the code under Task 4_f. The code and the result are displayed as follows.

```
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)
```

The row is finally inserted because the department 15 record exists in the database and the package's PL/SQL index-by table, due to invoking

`EMP_PKG.INIT_DEPARTMENTS`, which refreshes the package state data.



Script Output ×

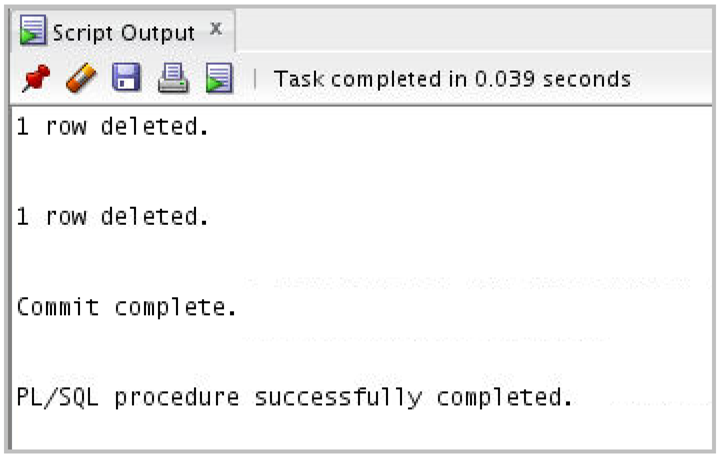
Task completed in 0.012 seconds

```
PL/SQL procedure successfully completed.
```

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.

Open Uncomment and select the code under Task 4_g. The code and the result are displayed as follows.

```
DELETE FROM employees
WHERE first_name = 'James' AND last_name = 'Bond';
DELETE FROM departments WHERE department_id = 15;
COMMIT;
EXECUTE EMP_PKG.INIT_DEPARTMENTS
```



Script Output ×

Task completed in 0.039 seconds

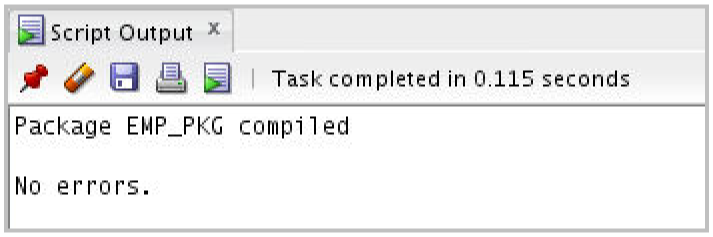
```
1 row deleted.


1 row deleted.


Commit complete.


PL/SQL procedure successfully completed.
```

5. Reorganize the subprograms in the package specification and the body so that they are in alphabetical sequence.

   a. Edit the package specification and reorganize subprograms alphabetically. Click Run Script to re-create the package specification. Compile the package specification. What happens?

   Uncomment and select the code under Task 5_a. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to re-create and compile the package. The code and the result are displayed as follows. The package's specification subprograms are already in an alphabetical order.

```
CREATE OR REPLACE PACKAGE emp_pkg IS

-- the package spec is already in an alphabetical order.

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

  FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

  FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype;

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS
```



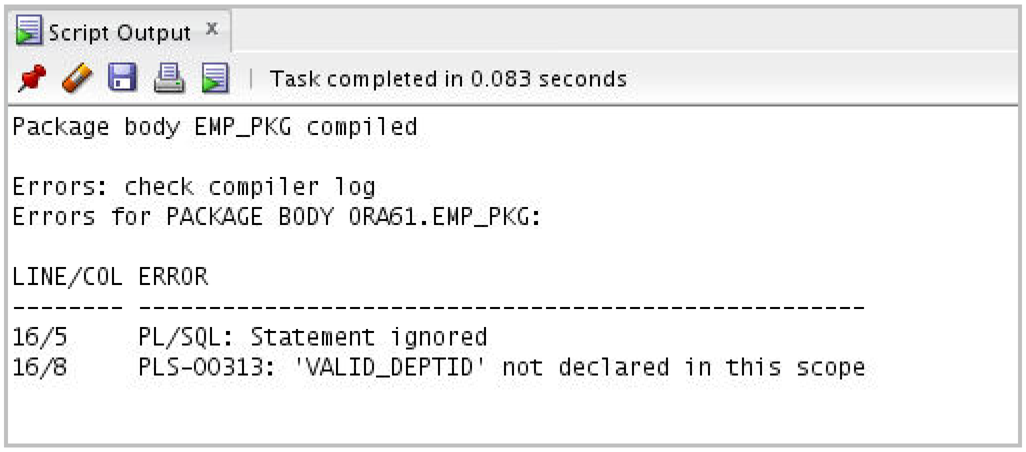
```
Package EMP_PKG compiled

No errors.
```

b. 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?

Uncomment and select the code under Task 5_b. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to re-create the package. The code and the result are displayed as follows.

```
-- Package BODY
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  TYPE boolean_tab_type IS TABLE OF BOOLEAN
     INDEX BY BINARY_INTEGER;
  valid_departments boolean_tab_type;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
  BEGIN
    IF valid_deptid(p_deptid) THEN
      INSERT INTO employees(employee_id, first_name, last_name,
email,
        job_id, manager_id, hire_date, salary, commission_pct,
department_id)
      VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
        p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
    END IF;
  END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
  BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
  END;

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
  BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p_empid;
  END get_employee;

  FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
  END;

  FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
  END;

  PROCEDURE init_departments IS
  BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
      valid_departments(rec.department_id) := TRUE;
    END LOOP;
  END;

  PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id ||' '||
                         p_rec_emp.employee_id||' '||
                         p_rec_emp.first_name||' '||
                         p_rec_emp.last_name||' '||
                           p_rec_emp.job_id||' '||
                           p_rec_emp.salary);
  END;

  FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
  BEGIN
    RETURN valid_departments.exists(p_deptid);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END valid_deptid;


BEGIN
  init_departments;
END emp_pkg;

/
SHOW ERRORS
```

The package does not compile successfully because the `VALID_DEPTID` function is referenced before it is declared.


Script Output
Task completed in 0.083 seconds

```
Package body EMP_PKG compiled

Errors: check compiler log
Errors for PACKAGE BODY ORA61.EMP_PKG:

LINE/COL ERROR
-------- -----------------------------------------------------
16/5     PL/SQL: Statement ignored
16/8     PLS-00313: 'VALID_DEPTID' not declared in this scope
```

c. 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?

Uncomment and select the code under Task 5_c. The function's forward declaration is highlighted in the code box below. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to re-create and compile the package. The code and the result are displayed as follows.

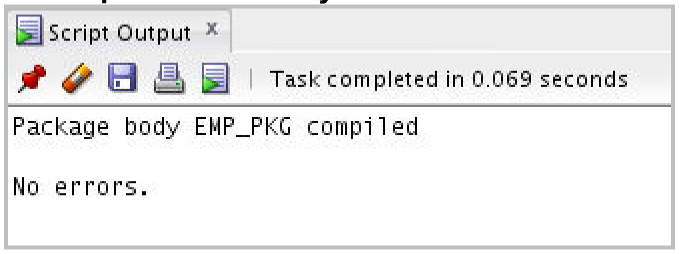
```
-- Package BODY
```

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  TYPE boolean_tab_type IS TABLE OF BOOLEAN
      INDEX BY BINARY_INTEGER;
  valid_departments boolean_tab_type;

 -- forward declaration of valid_deptid

  FUNCTION valid_deptid(p_deptid IN
     departments.department_id%TYPE)
   RETURN BOOLEAN;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
  BEGIN
    IF valid_deptid(p_deptid) THEN -- valid_deptid function
referneced
      INSERT INTO employees(employee_id, first_name, last_name,
email,
        job_id, manager_id, hire_date, salary, commission_pct,
department_id)
      VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
        p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
    END IF;
  END add_employee;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
  BEGIN
```

```
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
  END;

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
  BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p_empid;
  END get_employee;

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
  END;

  FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
  END;

-- New alphabetical location of function init_departments.

PROCEDURE init_departments IS
  BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
```

```
      valid_departments(rec.department_id) := TRUE;
    END LOOP;
  END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id ||' '||
                         p_rec_emp.employee_id||' '||
                         p_rec_emp.first_name||' '||
                         p_rec_emp.last_name||' '||
                         p_rec_emp.job_id||' '||
                         p_rec_emp.salary);
  END;

-- New alphabetical location of function valid_deptid.

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
  BEGIN
    RETURN valid_departments.exists(p_deptid);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END valid_deptid;


BEGIN
  init_departments;
END emp_pkg;

/
SHOW ERRORS
```

A forward declaration for the `VALID_DEPTID` function enables the package body to compile successfully as shown below:



```
Package body EMP_PKG compiled

No errors.
```