Practices for Lesson 11: Design Considerations for the PL/SQL Code

Chapter 11

Practices for Lesson 11: Overview

Overview

In this practice, you create a package that performs a bulk fetch of employees in a specified department. The data is stored in a PL/SQL table in the package. You also provide a procedure to display the contents of the table. In addition, you create the `add_employee` procedure that inserts new employees. The procedure uses a local autonomous subprogram to write a log record each time the `add_employee` procedure is called, whether it successfully adds a record or not.

Note:

1. Before starting this practice, execute `/home/oracle/labs/plpu/code_ex/cleanup_scripts/cleanup_11.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 11-1: Using Bulk Binding and Autonomous Transactions

Overview

In this practice, you create a package that performs a bulk fetch of employees in a specified department. The data is stored in a PL/SQL table in the package. You also provide a procedure to display the contents of the table. In addition, you create the `add_employee` procedure that inserts new employees. The procedure uses a local autonomous subprogram to write a log record each time the `add_employee` procedure is called, whether it successfully adds a record or not.

Note: Execute `cleanup_11.sql script` from `/home/oracle/labs/plpu/code_ex/cleanup_scripts/` before performing the following tasks.

Task

1. Update the `EMP_PKG` package with a new procedure to query employees in a specified department.
   a. In the package specification:
      1) Declare a `get_employees` procedure with a parameter called `dept_id`, which is based on the `employees.department_id` column type
      2) Define a nested PL/SQL type as a `TABLE OF EMPLOYEES%ROWTYPE`
   b. In the package body:
      1) Define a private variable called `emp_table` based on the type defined in the specification to hold employee records
      2) Implement the `get_employees` procedure to bulk fetch the data into the table
   c. Create a new procedure in the specification and body, called `show_employees`, which does not take arguments. The procedure displays the contents of the private PL/SQL table variable (if any data exists). Use the `print_employee` procedure that you created in an earlier practice. To view the results, click the Enable DBMS Output icon on the DBMS Output tab in SQL Developer, if you have not already done so.
   d. Enable `SERVEROUTPUT`. Invoke the `emp_pkg.get_employees` procedure for department `30`, and then invoke `emp_pkg.show_employees`. Repeat this for department `60`.
2. Your manager wants to keep a log whenever the `add_employee` procedure in the package is invoked to insert a new employee into the `EMPLOYEES` table.
   a. First, load and execute the code under Task 2_a from the `/home/oracle/labs/plpu/solns/sol_11.sql` script to create a log table called `LOG_NEWEMP`, and a sequence called `log_newemp_seq`.
   b. In the `EMP_PKG` package body, modify the `add_employee` procedure, which performs the actual `INSERT` operation. Add a local procedure called `audit_newemp` as follows:
      1) The `audit_newemp` procedure must use an autonomous transaction to insert a log record into the `LOG_NEWEMP` table.
      2) Store the `USER`, the current time, and the new employee name in the log table row.
      3) Use `log_newemp_seq` to set the `entry_id` column.

      Note: Remember to perform a `COMMIT` operation in a procedure with an autonomous transaction.

c. Modify the `add_employee` procedure to invoke `audit_newemp` before it performs the insert operation.

d. Invoke the `add_employee` procedure for these new employees: `Max Smart` in department 20 and `Clark Kent` in department 10. What happens?

e. Query the two `EMPLOYEES` records added, and the records in the `LOG_NEWEMP` table. How many log records are present?

f. Execute a `ROLLBACK` statement to undo the insert operations that have not been committed. Use the same queries from step 2 e. as follows:

   1) Use the first query to check whether the employee rows for `Smart` and `Kent` have been removed.

   2) Use the second query to check the log records in the `LOG_NEWEMP` table. How many log records are present? Why?

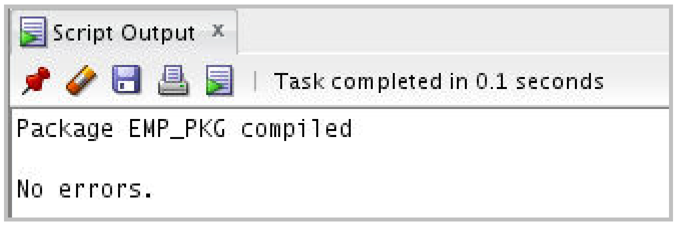
Solution 11-1: Using Bulk Binding and Autonomous Transactions

In this practice, you create a package that performs a bulk fetch of employees in a specified department. The data is stored in a PL/SQL table in the package. You also provide a procedure to display the contents of the table. In addition, you create the `add_employee` procedure that inserts new employees. The procedure uses a local autonomous subprogram to write a log record each time the `add_employee` procedure is called, whether it successfully adds a record or not.

1. Update the `EMP_PKG` package with a new procedure to query employees in a specified department.
   a. In the package specification:
      1) Declare a `get_employees` procedure with a parameter called `dept_id`, which is based on the `employees.department_id` column type
      2) Define a nested PL/SQL type as a `TABLE OF EMPLOYEES%ROWTYPE`

      Open the `/home/oracle/labs/plpu/solns/sol_11.sql` script. Uncomment and select the code under Task 1_a. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to create and compile the specification. The code and the results are displayed as follows. The newly added code is highlighted in bold letters in the code box below.

```
CREATE OR REPLACE PACKAGE emp_pkg IS

  TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

  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 get_employees(p_dept_id
employees.department_id%type);

  PROCEDURE init_departments;

  PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS
```



```
Package EMP_PKG compiled

No errors.
```

b. In the package body:

1) Define a private variable called `emp_table` based on the type defined in the specification to hold employee records

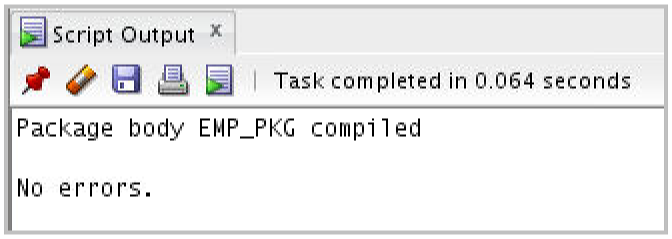
2) Implement the `get_employees` procedure to bulk fetch the data into the table

Uncomment and select the code under Task 1_b. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to create and compile the package body. The code and the results are shown below. The newly added code is highlighted in bold letters in the code box below.

```
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;
  emp_table         emp_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;

-- New get_employees procedure.

PROCEDURE get_employees(p_dept_id employees.department_id%type)
IS
  BEGIN
    SELECT * BULK COLLECT INTO emp_table
    FROM EMPLOYEES
    WHERE department_id = p_dept_id;
  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;

BEGIN
  init_departments;

END emp_pkg;
/
SHOW ERRORS
```



```
Package body EMP_PKG compiled

No errors.
```

c. Create a new procedure in the specification and body, called `show_employees`, which does not take arguments. The procedure displays the contents of the private PL/SQL table variable (if any data exists). Use the `print_employee` procedure that you created in an earlier practice. To view the results, click the Enable DBMS Output icon in the DBMS Output tab in SQL Developer, if you have not already done so.

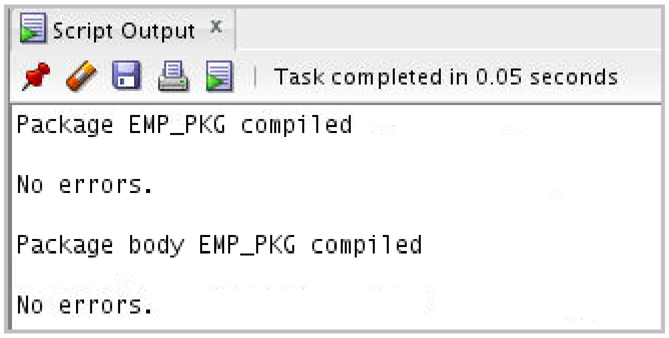
Uncomment and select the code under Task 1_c. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to re-create and compile the package with the new procedure. The code and the results are shown below.

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS
  TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;
  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 get_employees(p_dept_id
employees.department_id%type);

  PROCEDURE init_departments;

  PROCEDURE print_employee(p_rec_emp employees%rowtype);

  PROCEDURE show_employees;

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;
  emp_table         emp_tab_type;
  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
      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 get_employees(p_dept_id
  employees.department_id%type) IS
    BEGIN
      SELECT * BULK COLLECT INTO emp_table
      FROM EMPLOYEES
      WHERE department_id = p_dept_id;
    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;

    PROCEDURE show_employees IS
    BEGIN
      IF emp_table IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('Employees in Package table');
        FOR i IN 1 .. emp_table.COUNT
        LOOP
          print_employee(emp_table(i));
        END LOOP;
      END IF;
    END show_employees;

    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
```

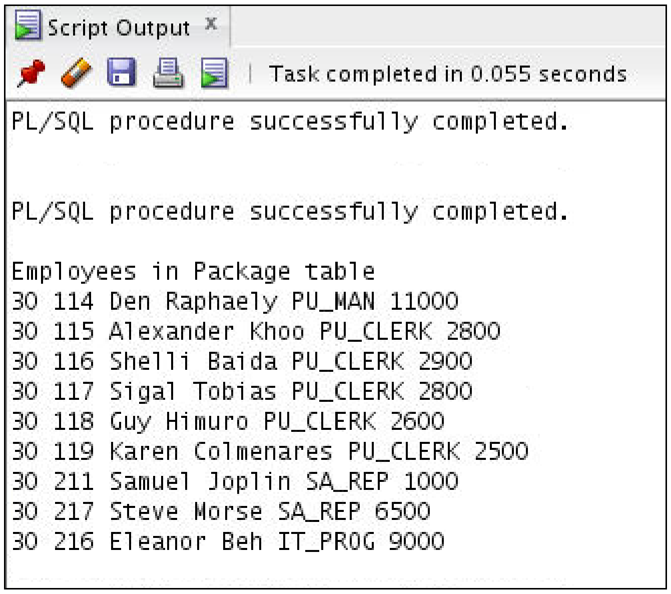


```
Package EMP_PKG compiled

No errors.

Package body EMP_PKG compiled

No errors.
```

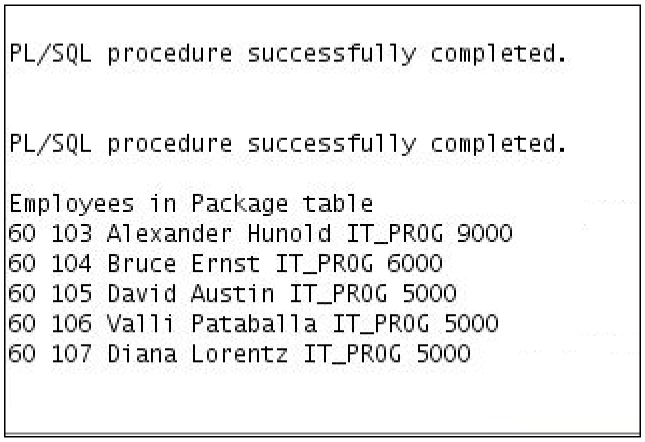
d. Enable `SERVEROUTPUT`. Invoke the `emp_pkg.get_employees` procedure for department `30`, and then invoke `emp_pkg.show_employees`. Repeat this for department `60`.

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 procedures. The code and the results are shown below:

```
SET SERVEROUTPUT ON

EXECUTE emp_pkg.get_employees(30)
EXECUTE emp_pkg.show_employees

EXECUTE emp_pkg.get_employees(60)
EXECUTE emp_pkg.show_employees
```



```
PL/SQL procedure successfully completed.


PL/SQL procedure successfully completed.

Employees in Package table
30 114 Den Raphaely PU_MAN 11000
30 115 Alexander Khoo PU_CLERK 2800
30 116 Shelli Baida PU_CLERK 2900
30 117 Sigal Tobias PU_CLERK 2800
30 118 Guy Himuro PU_CLERK 2600
30 119 Karen Colmenares PU_CLERK 2500
30 211 Samuel Joplin SA_REP 1000
30 217 Steve Morse SA_REP 6500
30 216 Eleanor Beh IT_PROG 9000
```

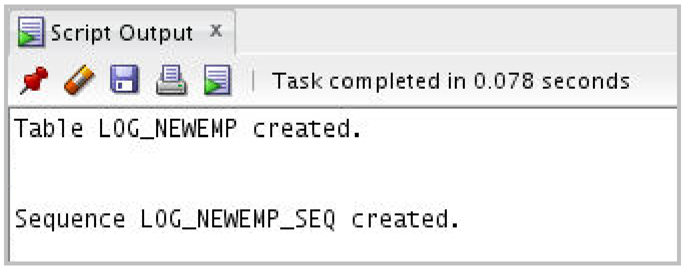


```
PL/SQL procedure successfully completed.


PL/SQL procedure successfully completed.

Employees in Package table
60 103 Alexander Hunold IT_PROG 9000
60 104 Bruce Ernst IT_PROG 6000
60 105 David Austin IT_PROG 5000
60 106 Valli Pataballa IT_PROG 5000
60 107 Diana Lorentz IT_PROG 5000
```

2. Your manager wants to keep a log whenever the `add_employee` procedure in the package is invoked to insert a new employee into the `EMPLOYEES` table.
   a. First, load and execute the code under Task 2_a from `/home/oracle/labs/plpu/solns/sol_11.sql` script to create a log table called `LOG_NEWEMP`, and a sequence called `log_newemp_seq`.

      Uncomment and select the code under Task 2_a. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
CREATE TABLE log_newemp (
  entry_id  NUMBER(6) CONSTRAINT log_newemp_pk PRIMARY KEY,
  user_id   VARCHAR2(30),
  log_time  DATE,
  name      VARCHAR2(60)
);

CREATE SEQUENCE log_newemp_seq;
```



```
Table LOG_NEWEMP created.


Sequence LOG_NEWEMP_SEQ created.
```

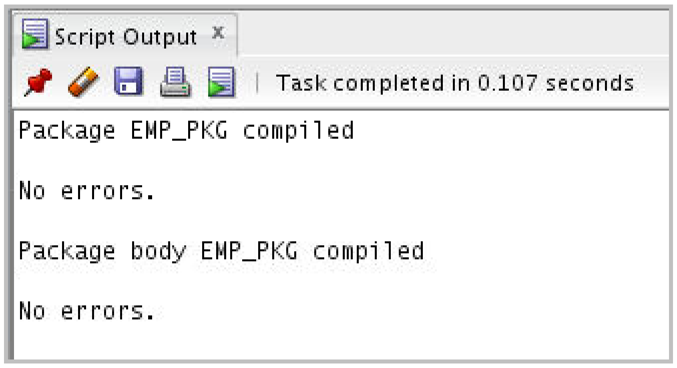
   b. In the `EMP_PKG` package body, modify the `add_employee` procedure, which performs the actual `INSERT` operation. Add a local procedure called `audit_newemp` as follows:
      1) The `audit_newemp` procedure must use an autonomous transaction to insert a log record into the `LOG_NEWEMP` table.
      2) Store the `USER`, the current time, and the new employee name in the log table row.
      3) Use `log_newemp_seq` to set the `entry_id` column.

Note: Remember to perform a `COMMIT` operation in a procedure with an autonomous transaction.

Uncomment and select the code under Task 2_b. The newly added code is highlighted in bold letters in the following code box. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to run the script. The code and the results are displayed as follows:

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS


  TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

  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 get_employees(p_dept_id
employees.department_id%type);
```

```
  PROCEDURE init_departments;

  PROCEDURE print_employee(p_rec_emp employees%rowtype);

  PROCEDURE show_employees;

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;
  emp_table         emp_tab_type;

  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

-- New local procedure

    PROCEDURE audit_newemp IS
      PRAGMA AUTONOMOUS_TRANSACTION;
      user_id VARCHAR2(30) := USER;
    BEGIN
      INSERT INTO log_newemp (entry_id, user_id, log_time,
                              name)
      VALUES (log_newemp_seq.NEXTVAL, user_id,
              sysdate,p_first_name||' '||p_last_name);
      COMMIT;
    END audit_newemp;

  BEGIN
  -- add_employee
    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 get_employees procedure.

  PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
  BEGIN
    SELECT * BULK COLLECT INTO emp_table
    FROM EMPLOYEES
    WHERE department_id = p_dept_id;
  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;

  PROCEDURE show_employees IS
  BEGIN
    IF emp_table IS NOT NULL THEN
      DBMS_OUTPUT.PUT_LINE('Employees in Package table');
      FOR i IN 1 .. emp_table.COUNT
      LOOP
        print_employee(emp_table(i));
      END LOOP;
    END IF;
  END show_employees;

  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
```



```
Package EMP_PKG compiled

No errors.

Package body EMP_PKG compiled

No errors.
```

c. Modify the `add_employee` procedure to invoke `audit_newemp` before it performs the insert operation.

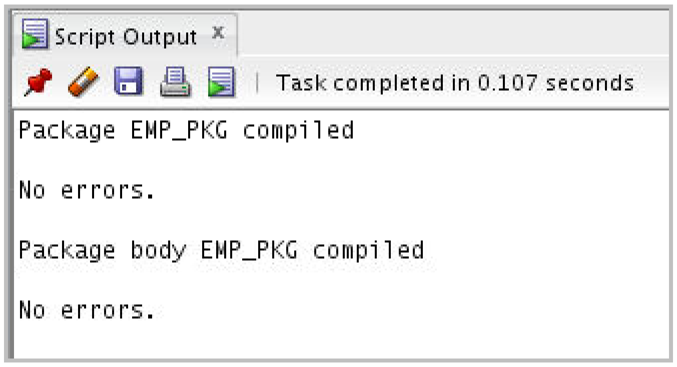
Uncomment and select the code under Task 2_c. The newly added code is highlighted in bold letters in the following code box. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS


  TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

  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 get_employees(p_dept_id
employees.department_id%type);
```

```
  PROCEDURE init_departments;

  PROCEDURE print_employee(p_rec_emp employees%rowtype);

  PROCEDURE show_employees;

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;
  emp_table         emp_tab_type;

  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

    PROCEDURE audit_newemp IS
      PRAGMA AUTONOMOUS_TRANSACTION;
      user_id VARCHAR2(30) := USER;
    BEGIN
      INSERT INTO log_newemp (entry_id, user_id, log_time, name)
      VALUES (log_newemp_seq.NEXTVAL, user_id,
sysdate,p_first_name||' '||p_last_name);
      COMMIT;
    END audit_newemp;
  BEGIN -- add_employee
    IF valid_deptid(p_deptid) THEN
      audit_newemp;
      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 get_employees(p_dept_id
employees.department_id%type) IS
  BEGIN
    SELECT * BULK COLLECT INTO emp_table
    FROM EMPLOYEES
    WHERE department_id = p_dept_id;
  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;

  PROCEDURE show_employees IS
  BEGIN
    IF emp_table IS NOT NULL THEN
      DBMS_OUTPUT.PUT_LINE('Employees in Package table');
      FOR i IN 1 .. emp_table.COUNT
      LOOP
        print_employee(emp_table(i));
      END LOOP;
    END IF;
  END show_employees;

  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
```

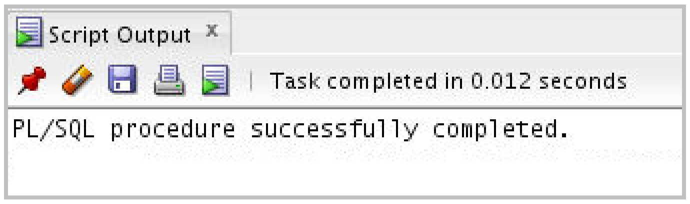


```
Package EMP_PKG compiled

No errors.

Package body EMP_PKG compiled

No errors.
```

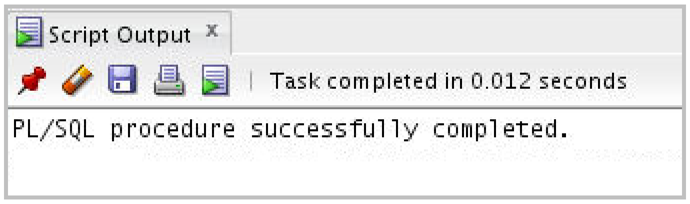
d. Invoke the `add_employee` procedure for these new employees: `Max Smart` in department 20 and `Clark Kent` in department 10. What happens?

Uncomment and select the code under Task 2_d. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to run the script. The code and the results are as follows.

```
EXECUTE emp_pkg.add_employee('Max', 'Smart', 20)
EXECUTE emp_pkg.add_employee('Clark', 'Kent', 10)
```



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



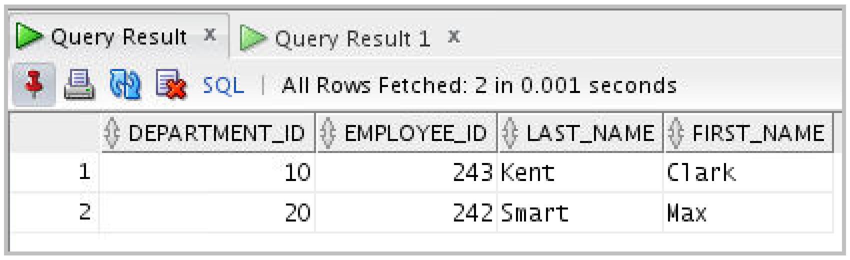
```
PL/SQL procedure successfully completed.
```

Both insert statements complete successfully. The log table has two log records as shown in the next step.

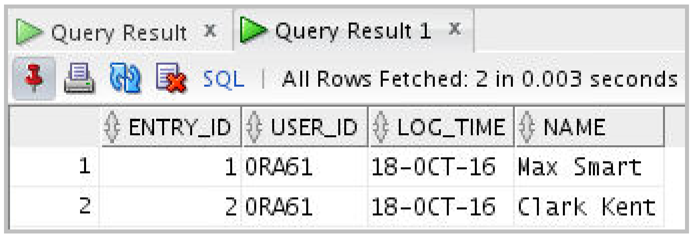
e. Query the two `EMPLOYEES` records added, and the records in the `LOG_NEWEMP` table. How many log records are present?

Uncomment and select the code under Task 2_e. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to run the script. The code and the results are displayed as follows:

```
select department_id, employee_id, last_name, first_name
from employees
where last_name in ('Kent', 'Smart');

select * from log_newemp;
```



	DEPARTMENT_ID	EMPLOYEE_ID	LAST_NAME	FIRST_NAME
1	10	243	Kent	Clark
2	20	242	Smart	Max



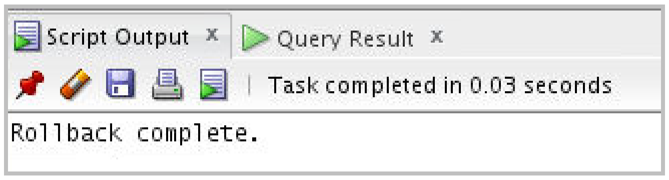
	ENTRY_ID	USER_ID	LOG_TIME	NAME
1	1	ORA61	18-OCT-16	Max Smart
2	2	ORA61	18-OCT-16	Clark Kent

There are two log records, one for `Smart` and another for `Kent`.

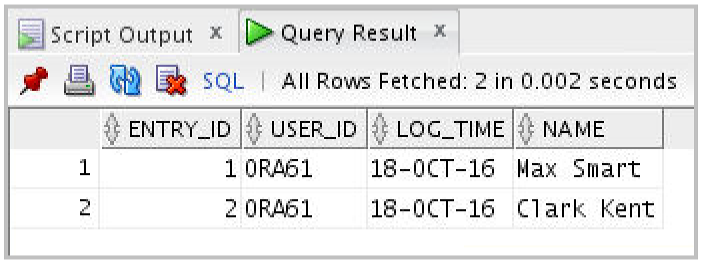
f. Execute a `ROLLBACK` statement to undo the insert operations that have not been committed. Use the same queries from step 2 e. as follows:

1) Use the first query to check whether the employee rows for `Smart` and `Kent` have been removed.
2) Use the second query to check the log records in the `LOG_NEWEMP` table. How many log records are present? Why?

```
ROLLBACK;
select * from log_newemp;
```



```
Rollback complete.
```



	ENTRY_ID	USER_ID	LOG_TIME	NAME
1	1	ORA61	18-OCT-16	Max Smart
2	2	ORA61	18-OCT-16	Clark Kent

The two employee records are removed (rolled back). The two log records remain in the log table because they were inserted using an autonomous transaction, which is unaffected by the rollback performed in the main transaction.