# Database Programming with PL/SQL

## 10-1: Creating Packages Practice

## Activities

### Vocabulary

Identify the vocabulary word for each definition below:

| | |
|---|---|
| **Package specification** | The interface to your applications that declares the constructs (procedures, functions, variables, and so on) which are visible to the calling environment. |
| **Encapsulation** | A two-part structure in which the detailed package body code is invisible to the calling environment, which can only see the specification. If changes to the code are needed, the body can be edited and recompiled without having to edit or recompile the specification. |
| **OR REPLACE** | An option that drops and re-creates the package body. |
| **Package body** | This contains the executable code of the subprograms which were declared in the package specification. It may also contain its own variable declarations. |
| **PL/SQL packages** | Containers that enable you to group together related PL/SQL subprograms, variables, cursors, and exceptions. |

### Try It / Solve It

1. Name at least four objects that can be put into a package.
**Variables, constantes, procedimientos y funciones.**

2. Name the two components of a package and explain the difference between them.
   **Package specification: En esta parte se hace la declaración del package, se declaran variables, constantes, funciones, procedimientos, etc. En esta parte solo se declara el package, no se agrega código.**
   **Package body: En esta parte del package está todo el código de cada función o procedimiento que se declara en package specification.**

3. Which component must be created first, and why?

**El package specification se crea primero, porque ahí es donde se declara todo lo que contendrá el package.**

4. Write a query against a Data Dictionary to display the list of packages you own.

**SELECT * FROM USER_OBJECTS WHERE OBJECT_TYPE LIKE 'PACKAGES';**

5. Create the specification for the check_emp_pkg which you studied in this lesson. The specification should declare a constant and two procedures, as follows:

   - g_max_length_of_service, datatype NUMBER, initialized to 100
   - chk_hiredate with one input parameter having the same datatype as employees.hire_date
   - chk_dept_mgr with two input parameters having the same datatypes as employees.employee_id and employees.manager_id.

**CREATE OR REPLACE PACKAGE CHECK_EMP_PKG AS**

**G_max_length_of_service NUMBER:=100;**

**PROCEDURE chk_hiredate(p_hire_date IN employees.hire_date%type);**

**PROCEDURE chk_dept_mgr(p_employee_id IN employees.employee_id%type, p_manager_id IN employees.manager_id%type);**

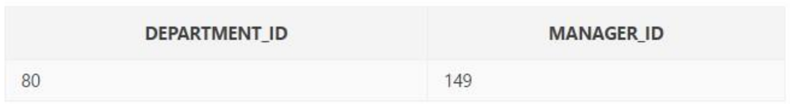
**END;**

6. Create the package body for check_emp_pkg. Remember that the names and parameters of the procedures in the body must be identical to those in the specification, or the body will not compile.

   The code for chk_hiredate should RAISE_APPLICATION_ERROR if the employee was hired more than 100 years ago (hint: use MONTHS_BETWEEN, comparing with g_max_length_of_service * 12). The second procedure, chk_dept_mgr, accepts two input parameters: an employee_id and a manager_id. The code should find the manager of the employee's department and check whether this manager has the same manager_id as the second parameter. If the manager_id is the same, display a suitable "success" message; if they are different, raise an application error. Include an exception handler for NO_DATA_FOUND.

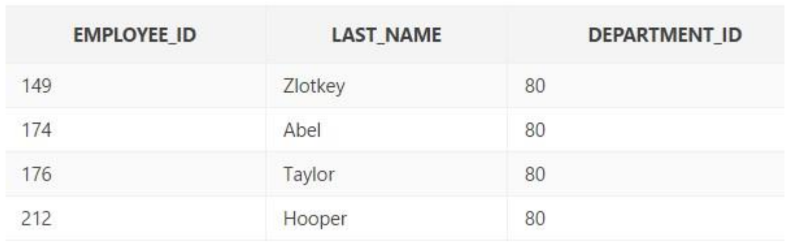
   The following sample data from the employees and departments tables may help you:

   Departments:



| DEPARTMENT_ID | MANAGER_ID |
|---|---|
| 80 | 149 |

   Employees:



| EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID |
|---|---|---|
| 149 | Zlotkey | 80 |
| 174 | Abel | 80 |
| 176 | Taylor | 80 |
| 212 | Hooper | 80 |

```
CREATE OR REPLACE PACKAGE BODY check_emp_pkg IS
        PROCEDURE chk_hiredate(p_date IN employees.hire_date%TYPE) IS
                v_meses NUMBER;
        BEGIN
                v_meses := MONTHS_BETWEEN(SYSDATE,p_date );
                IF (v_meses > g_max_length_of_service * 12) THEN
                RAISE_APPLICATION_ERROR(-20000,'Fecha No Valida.
                                        El empleado no puede tener mas de 100 años');
        END IF;

        DBMS_OUTPUT.PUT_LINE('A�os trabajados: ' || v_meses/12);

        END chk_hiredate;


        PROCEDURE chk_dept_mgr(p_emp_id IN employees.employee_id%TYPE, p_mgr IN
employees.manager_id%TYPE) IS

                v_manager employees.manager_id%TYPE;

        BEGIN

                SELECT d.manager_id INTO v_manager FROM employees e inner join
departments d on e.department_id=d.department_id  WHERE e.employee_id=p_emp_id;

                IF(v_manager = p_mgr) THEN

                        DBMS_OUTPUT.PUT_LINE('Es correcto!');

                ELSE

                        RAISE_APPLICATION_ERROR(-20001,'El manager con ID ' || p_mgr || ' no
es el jefe del empleado con ID ' || p_emp_id);

                END IF;


        EXCEPTION

                WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('No se encontro
empleado con el ID especificado');

        END chk_dept_mgr;


END check_emp_pkg;
```

Passing parameters (174,149) would be successful, while (174,176) would raise an error.

```
BEGIN

check_emp_pkg.chk_dept_mgr(174,149);

END;
```

```
BEGIN
check_emp_pkg.chk_dept_mgr(174,176);
END;
```

7. Test the chk_hiredate procedure using input value 17-Jan-1987 (it should succeed).

```
BEGIN
CHECK_EMP_PKG.chk_hiredate('17-JAN-1987');
END;
```

8. Test the chk_dept_mgr procedure twice using input values (174,149) and (174,176). The first should succeed while the second should fail.

```
 BEGIN
check_emp_pkg.chk_dept_mgr(174,149);
END;

BEGIN
check_emp_pkg.chk_dept_mgr(174,176);
END;
```

9. Now you want to modify the package so that the chk_dept_mgr procedure displays a different error message if the two manager_ids are different. What do you need to recreate: the Specification, the Body, or both?

   **Solo se necesita modificar el body, ya que en la especificación del procedimiento no se cambia nada.**