

**MP0484. Bases de datos
UF7. Programación avanzada
de acceso a datos**

7.2. Paquetes

Índice

☰	Objetivos	3
☰	Definición y creación del paquete	4
☰	Ventajas de los Paquetes PL/SQL	19

Objetivos

Un paquete sirve para encerrar la lógica del negocio, es decir las consultas y/o actualizaciones de mis tablas para evitar accesos indebidos.

Objetivos:

- Conocer las partes en que se divide la creación de un paquete.
- Qué se puede escribir en la especificación del paquete.
- El cuerpo del paquete subprogramas públicos y privados.
- Desarrollo de un ejemplo de Paquete.
- Trabajar con cursores variables en un paquete.
- Ver las ventajas de trabajar con paquetes.

Definición y creación del paquete



Un paquete es un objeto del esquema que agrupa lógicamente variables, constantes, tipos de datos y subprogramas PL/SQL. Sirven para encapsular en ellos la lógica de los accesos a tablas que tengan que ver con un esquema de negocio, o con una lógica de proceso(tratamiento de ficheros, funciones standard, salida por consola...).

Al ser un bloque almacenado, se usa la sentencia SQL/DDL, CREATE.

Los paquetes se dividen en:

- **Especificación:** es la zona de declaración de las variables, tipos, constantes, excepciones, cursos y subprogramas disponibles para ser usados. Para crear un paquete se utiliza la sentencia **CREATE OR REPLACE PACKAGE nombre_package**.
- **Cuerpo:** zona en la que se implanta el código de los cursos y subprogramas definidos en la especificación, también puede contener otras declaraciones y otros subprogramas que no están definidos en la especificación. Para crear el cuerpo se utiliza la sentencia **CREATE OR REPLACE PACKAGE BODY nombre_package**, con la implementación del código.

Crear la especificación y el cuerpo

La forma genérica de crear una especificación de Paquete es:

```
CREATE [OR REPLACE] PACKAGE Nombre_paquete
    [AUTHID {CURRENT_USER | DEFINER}]
{IS | AS}
    [PRAGMA SERIALLY_REUSABLE;]
    [Definición_Tipo_Colección ...]
    [Definición_tipo_Registro ...]
    [Definición_Subtipos ...]
    [Declaración_Colección ...]
    [Declaración_constante ...]
    [Declaracion_Excepción ...]
    [Declaración_Objeto ...]
    [Declaración_Registro ...]
    [Declaración_Variable ...]
    [Especificación_Cursor ...]
    [Especificación_Función...]
    [Especificación_Procedimiento ...]
    [Especificación_Llamada ...]
    [PRAGMA RESTRICT_REFERENCES(Tipos_Comportamiento) ...]
END [Nombre_paquete];
/
```

La especificación contiene la parte pública del paquete la cual es visible desde otras aplicaciones. Los procedimientos tienen que ser declarados al final de la zona de especificación, excepto las PRAGMAS que hacen referencia a alguna función.

Una vez creada la cabecera del paquete, posteriormente se crearía el cuerpo.

```

CREATE [OR REPLACE] PACKAGE BODY Nombre_paquete
{IS | AS}
    [PRAGMA SERIALLY_REUSABLE;]
    [Definición_Tipo_Colección ...]
    [Definición_tipo_Registro ...]
    [Definición_Subtipos ...]
    [Declaración_Colección ...]
    [Declaración_constante ...]
    [Declaracion_Excepción ...]
    [Declaración_Objeto ...]
    [Declaración_Registro ...]
    [Declaración_Variable ...]
    [Especificación_Cursor ...]
    [Especificación_Función...]
    [Especificación_Procedimiento ...]
    [Especificación_Llamada ...]

[BEGIN
    Sentencias procedurales
[EXCEPTION
    Tratamiento de excepciones]]
END [Nombre_paquete];
/

```

Los procedimientos y/o funciones de un paquete se pueden **sobrecargar**, es decir, crear el mismo nombre, pero admitiendo distintos tipos de parámetros y/o distinto número de parámetros.

El cuerpo del paquete contiene la implementación de cada cursor y subprograma declarado en la parte de las especificaciones. Todos los subprogramas que estén declarados en las especificaciones serán públicos, el resto serán privados y no podrán ser accedidos fuera del paquete.

Para concordar los procedimientos declarados en la zona de especificaciones y el cuerpo se hace una comparación carácter a carácter. La única excepción es el espacio en blanco. En caso de que no coincidiera Oracle levantaría una excepción.

El begin de un paquete es opcional y éste se ejecuta una sola vez por sesión.

La zona de excepciones del paquete trata las excepciones originadas sólo por instrucciones del begin del paquete.

La invocación a un tipo de datos avanzado, subprograma de un paquete desde otro paquete, procedimiento o función se puede realizar, siempre y cuando estos subprogramas sean públicos:

```

DECLARE
    variable      nommbre_paquete.tipo_avanzado;

BEGIN
    Nombre_paquete.nombre_procedimiento(parametros);

    Variable:= Nombre_paquete.nombre_funcion(parametros);

    IF Nombre_paquete.nombre_funcion(parámetros) < 10 THEN
        ...
    END IF;

```

Ejemplo de paquete

Para la gestión de las tablas del esquema HR, vamos a crear un paquete llamado: PKT_GESTION_EMPL. Y vamos a crear los siguientes Subprogramas y tipos necesarios:

Procedimientos:

- PROCEDURE IMP_EMPLEADOS(P_DEP EMPLOYEES.DEPARTMENT_ID%TYPE, P_CUANTOS OUT PLS_INTEGER);
- PROCEDURE CUR_ESPECIFICO(P_CURSOR IN OUT CUR_2V2, OPCION PLS_INTEGER);

Funciones:

- FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE, P_FILA OUT DEPARTMENTS%ROWTYPE) RETURN BOOLEAN;
- FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE) RETURN BOOLEAN;

Y los siguientes Tipos avanzados:

- TYPE REG_2V2 IS RECORD (CAMPO_1 VARCHAR2(4000), CAMPO_2 VARCHAR2(4000));
- TYPE CUR_2V2 IS REF CURSOR RETURN REG_2V2;

```

CREATE OR REPLACE PACKAGE PKT_GESTION_EMPL AS

    TYPE REG_2V2 IS RECORD (CAMPO_1 VARCHAR2(4000), CAMPO_2 VARCHAR2(4000));
    TYPE CUR_2V2 IS REF CURSOR RETURN REG_2V2;

    PROCEDURE IMP_EMPLEADOS(P_DEP EMPLOYEES.DEPARTMENT_ID%TYPE, P_CUANTOS OUT
    PLS_INTEGER);
    FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE, P_FILA OUT DEPART-
    MENTS%ROWTYPE) RETURN BOOLEAN;
    FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE) RETURN BOOLEAN;

    PROCEDURE CUR_ESPECIFICO( P_CURSOR IN OUT CUR_2V2, OPCION PLS_INTEGER);

END PKT_GESTION_EMPL;
/
Package created.

```

PKT_GESTION_EMPL

Una vez compilada la especificación del paquete, procedemos a crear el body con la implementación de cada procedimiento y cada función, si lo hacemos con el sqldeveloper, este es el cuerpo que genera:

```

CREATE OR REPLACE
PACKAGE BODY PKT_GESTION_EMPL AS

    PROCEDURE IMP_EMPLEADOS(P_DEP EMPLOYEES.DEPARTMENT_ID%TYPE, P_CUANTOS OUT PLS_-
    INTEGER) AS
    BEGIN
        -- TAREA: Se necesita implantación para PROCEDURE PKT_GESTION_EMPL.IMP_EMPLEA-
        DOS
        NULL;
    END IMP_EMPLEADOS;

    FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE, P_FILA OUT DEPART-
    MENTS%ROWTYPE) RETURN BOOLEAN AS
    BEGIN
        -- TAREA: Se necesita implantación para FUNCTION PKT_GESTION_EMPL.EXISTE_DEP
        RETURN NULL;
    END EXISTE_DEP;

```

```

FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE) RETURN BOOLEAN AS
BEGIN
    -- TAREA: Se necesita implantación para FUNCTION PKT_GESTION_EMPL.EXISTE_DEP
    RETURN NULL;
END EXISTE_DEP;

PROCEDURE CUR_ESPECIFICO( P_CURSOR IN OUT CUR_2V2, OPCION PLS_INTEGER) AS
BEGIN
    -- TAREA: Se necesita implantación para PROCEDURE PKT_GESTION_EMPL.CUR_ESPE-
CIFICO
    NULL;
END CUR_ESPECIFICO;

END PKT_GESTION_EMPL;

```

PKT_GESTION_EMPL BODY



Es una gran ayuda, porque nos crea un cuerpo, totalmente compilable, sin errores y a partir de ahí podemos ir dando forma a cada uno de los subprogramas.

1

**PROCEDURE IMP_EMPLEADOS(P_DEP EMPLOYEES.DEPARTMENT_ID%TYPE,
P_CUANTOS OUT PLS_INTEGER) AS**

Recibe por parámetro un código de departamento, e imprime los empleados pertenecientes a este departamento. En una variable OUT vuelca el número de empleados se han procesado.

El procedimiento no verifica si el departamento existe o no existe.

Inténtalo para comprobar:

```

CREATE OR REPLACE
PACKAGE BODY PKT_GESTION_EMPL AS

    PROCEDURE IMP_EMPLEADOS(P_DEP EMPLOYEES.DEPARTMENT_ID%TYPE, P_CUANTOS OUT
PLS_INTEGER) AS
        CURSOR CUR_EMPL IS
            SELECT LAST_NAME, SALARY, HIRE_DATE
            FROM EMPLOYEES
            WHERE DEPARTMENT_ID = P_DEP;

        BEGIN
            P_CUANTOS := 0;
            FOR REG_EMPL IN CUR_EMPL LOOP
                DBMS_OUTPUT.PUT('APELLIDO : ' || RPAD(REG_EMPL.LAST_NAME,15));
                DBMS_OUTPUT.PUT('SALARIO : ' || LPAD(REG_EMPL.SALARY,8));
                DBMS_OUTPUT.PUT_LINE(' FECHA : ' || TO_CHAR(REG_EMPL.HIRE_DA-
TE, 'DD-MM-YYYY'));
            --     P_CUANTOS := CUR_EMPL%ROWCOUNT;
                P_CUANTOS := P_CUANTOS + 1;
            END LOOP;
        END IMP_EMPLEADOS;

        FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE, P_FILA OUT DEPART-
MENTS%ROWTYPE) RETURN BOOLEAN AS
        BEGIN
            -- TAREA: Se necesita implantación para FUNCTION PKT_GESTION_EMPL.EXISTE_DEP
            RETURN NULL;
        END EXISTE_DEP;

        FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE) RETURN BOOLEAN AS
        BEGIN
            -- TAREA: Se necesita implantación para FUNCTION PKT_GESTION_EMPL.EXISTE_DEP
            RETURN NULL;
        END EXISTE_DEP;

        FUNCTION GET_EMPLEADOS(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE) RETURN TAB_EMPL
AS
        BEGIN
            -- TAREA: Se necesita implantación para FUNCTION PKT_GESTION_EMPL.GET_EMPLEA-
DOS
            RETURN NULL;
        END GET_EMPLEADOS;

        PROCEDURE CUR_ESPECIFICO( P_CURSOR IN OUT CUR_2V2, OPCION PLS_INTEGER) AS
        BEGIN
            -- TAREA: Se necesita implantación para PROCEDURE PKT_GESTION_EMPL.CUR_ESPE-
CIFICO
            NULL;
        END CUR_ESPECIFICO;

        PROCEDURE INSERT_DEPAR(P_FILA_DEP DEPARTMENTS%ROWTYPE) AS
        BEGIN
            -- TAREA: Se necesita implantación para PROCEDURE PKT_GESTION_EMPL.INSERT_DE-
PAR
            NULL;
        END INSERT_DEPAR;

    END PKT_GESTION_EMPL;

```

PROCEDURE IMP_EMPLEADOS

Y hacemos un bloque anónimo para probar el procedimiento:

```

SET SERVEROUTPUT ON
DECLARE
    V_CUANTOS PLS_INTEGER;
    V_DEP EMPLOYEES.DEPARTMENT_ID%TYPE;
    F_DEP DEPARTMENTS%ROWTYPE;
BEGIN
    V_DEP := &DEP;
    PKT_GESTION_EMPL.IMP_EMPLEADOS(V_DEP,V_CUANTOS);
    IF V_CUANTOS = 0 THEN
        DBMS_OUTPUT.PUT_LINE('DEPARTAMENTO SIN EMPLEADOS');
    ELSE
        DBMS_OUTPUT.PUT_LINE('EMPLEADOS LEIDOS : ' || V_CUANTOS);
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('ERROR GENERAL TEST_imp_empls : ' || SQLERRM);

END;
/
Enter value for dep: 30
old   6:  V_DEP := &DEP;
new   6:  V_DEP := 30;
APELIDO : Raphaely      SALARIO :     11000  FECHA : 07-12-2002
APELIDO : Khoo          SALARIO :      3100  FECHA : 18-05-2003
APELIDO : Baida         SALARIO :      2900  FECHA : 24-12-2005
APELIDO : Tobias        SALARIO :      2800  FECHA : 24-07-2005
APELIDO : Himuro        SALARIO :      2600  FECHA : 15-11-2006
APELIDO : Colmenares    SALARIO :      2500  FECHA : 10-08-2007
EMPLEADOS LEIDOS : 6

PL/SQL procedure successfully completed.

```

2

**FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE,
P_FILA OUT DEPARTMENTS%ROWTYPE) RETURN BOOLEAN AS**

Función que pide un código de departamento y si existe en la tabla devuelve true y además vuelca la fila obtenida en un parámetro de departments%rowtype.

Si no existe el departamento devuelve FALSE.

Inténtalo para comprobar:

```

CREATE OR REPLACE
PACKAGE BODY PKT_GESTION_EMPL AS

    PROCEDURE IMP_EMPLEADOS(P_DEP EMPLOYEES.DEPARTMENT_ID%TYPE, P_CUANTOS OUT
PLS_INTEGER) AS
        CURSOR CUR_EMPL IS
            SELECT LAST_NAME, SALARY, HIRE_DATE
            FROM EMPLOYEES
            WHERE DEPARTMENT_ID = P_DEP;

        BEGIN
            P_CUANTOS := 0;
            FOR REG_EMPL IN CUR_EMPL LOOP
                DBMS_OUTPUT.PUT('APELLIDO : ' || RPAD(REG_EMPL.LAST_NAME,15));
                DBMS_OUTPUT.PUT('SALARIO : ' || LPAD(REG_EMPL.SALARY,8));
                DBMS_OUTPUT.PUT_LINE(' FECHA : ' || TO_CHAR(REG_EMPL.HIRE_DA-
TE, 'DD-MM-YYYY'));
                -- P_CUANTOS := CUR_EMPL%ROWCOUNT;
                P_CUANTOS := P_CUANTOS + 1;

            END LOOP;
        END IMP_EMPLEADOS;

        FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE, P_FILA OUT DEPART-
MENTS%ROWTYPE) RETURN BOOLEAN AS
        BEGIN
            SELECT *
            INTO P_FILA
            FROM DEPARTMENTS
            WHERE DEPARTMENT_ID = P_DEP;
            RETURN TRUE;
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                RETURN FALSE;
        END EXISTE_DEP;

        FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE) RETURN BOOLEAN AS
        BEGIN
            RETURN NULL;
        END EXISTE_DEP;

        PROCEDURE CUR_ESPECIFICO( P_CURSOR IN OUT CUR_2V2, OPCION PLS_INTEGER) AS
        BEGIN
            -- TAREA: Se necesita implantación para PROCEDURE PKT_GESTION_EMPL.CUR_ESPE-
CIFICO
            NULL;
        END CUR_ESPECIFICO;

    END PKT_GESTION_EMPL;

```

FUNCTION EXISTE_DEP, con 2 parámetros.

Y el correspondiente bloque anónimo para probar la función EXISTE_DEP:

```

SET SERVEROUTPUT ON
DECLARE
    FILA_DEP DEPARTMENTS%ROWTYPE;
BEGIN
    IF PKT_GESTION_EMPL.EXISTE_DEP(30, FILA_DEP) THEN
        DBMS_OUTPUT.PUT_LINE('DEPARTAMENTO : ' || FILA_DEP.DEPARTMENT_NAME);
    ELSE
        DBMS_OUTPUT.PUT_LINE('DEPARTAMENTO NO ENCONTRADO');
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('ERROR GENERAL TEST_FUN_EXISTE_DEP : ' || SQLERRM);
END;
/
DEPARTAMENTO : Purchasing

PL/SQL procedure successfully completed.

```

TES_EXISTE_DEP

3

Sobrecargar la Función EXISTE_DEP

Definimos el mismo nombre de función EXISTE_DEP, pero solo admitimos como parámetro de entrada el código de departamento, si existe devolvemos TRUE y si no existe devolvemos FALSE.

Para probar vamos a usar el mismo de antes, para saber el mensaje de qué función es, en los DBMS especificamos fun 2 param, fun 1 param.

```

CREATE OR REPLACE
PACKAGE BODY PKT_GESTION_EMPL AS

PROCEDURE IMP_EMPLEADOS(P_DEP EMPLOYEES.DEPARTMENT_ID%TYPE, P_CUANTOS OUT

```

```

PLS_INTEGER) AS
CURSOR CUR_EMPL IS
SELECT LAST_NAME, SALARY, HIRE_DATE
FROM EMPLOYEES
WHERE DEPARTMENT_ID = P_DEP;

BEGIN
    P_CUANTOS := 0;
    FOR REG_EMPL IN CUR_EMPL LOOP
        DBMS_OUTPUT.PUT('APELLIDO : ' || RPAD(REG_EMPL.LAST_NAME,15));
        DBMS_OUTPUT.PUT('SALARIO : ' || LPAD(REG_EMPL.SALARY,8));
        DBMS_OUTPUT.PUT_LINE(' FECHA : ' || TO_CHAR(REG_EMPL.HIRE_DATE, 'DD-MM-YYYY'));
    --    P_CUANTOS := CUR_EMPL%ROWCOUNT;
        P_CUANTOS := P_CUANTOS + 1;
    END LOOP;
END IMP_EMPLEADOS;

FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE, P_FILA OUT
DEPARTMENTS%ROWTYPE) RETURN BOOLEAN AS
BEGIN
    SELECT *
    INTO P_FILA
    FROM DEPARTMENTS
    WHERE DEPARTMENT_ID = P_DEP;
    RETURN TRUE;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END EXISTE_DEP;

FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE) RETURN BOOLEAN
AS
    V_CUANTOS pls_integer := 0;
BEGIN
    SELECT COUNT(*)
    INTO V_CUANTOS
    FROM EMPLOYEES
    WHERE DEPARTMENT_ID = P_DEP;
    IF V_CUANTOS = 0 THEN
        DBMS_OUTPUT.PUT_LINE('EXISTE_DEP 1 PARAM');
        RETURN FALSE;
    ELSE
        DBMS_OUTPUT.PUT_LINE('EXISTE_DEP 1 PARAM');
        RETURN TRUE;
    END IF;
END EXISTE_DEP;

PROCEDURE CUR_ESPECIFICO( P_CURSOR IN OUT CUR_2V2, OPCION PLS_INTEGER) AS
BEGIN
    -- TAREA: Se necesita implantación para PROCEDURE PKT_GESTION_EMPL. -
CUR_ESPECIFICO
    NULL;
END CUR_ESPECIFICO;

END PKT_GESTION_EMPL;

```

Y el bloque anónimo de prueba:

```

SET SERVEROUTPUT ON

BEGIN
    IF PKT_GESTION_EMPL.EXISTE_DEP(430)  THEN
        null;
    ELSE
        DBMS_OUTPUT.PUT_LINE('DEPARTAMENTO NO ENCONTRADO');
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('ERROR GENERAL TEST_FUN_EXISTE_DEP : ' || SQLERRM);
END;
/
EXISTE_DEP 1 PARAM
DEPARTAMENTO NO ENCONTRADO

PL/SQL procedure successfully completed.

```

Trabajando con cursosres variables

Vamos a generar el procedimiento llamado:

```
PROCEDURE CUR_ESPECIFICO( P_CURSOR IN OUT CUR_2V2, OPCION PLS_INTEGER) AS
```

Este procedimiento trabaja con un cursor variable IN OUT, definido como TYPE en la especificación del paquete, que devuelve consultas formadas por 2 variables de tipo VARCHAR2, definidas en un registro del paquete.

El procedimiento recibe además de una variable REF CURSOR del paquete, un número natural de forma que si recibimos en la OPCION los valores:

1. Seleccionamos el last_name y el first_name de employees para los empleados del departamento 30.
2. Seleccionamos el job_id y job_title de la tabla jobs, de aquellos cuyo job_id comience con la letra 'S'.

Cualquier otro número levantamos la excepción -20100, 'OPCION INCORRECTA, SOLO 1 o 2'. El cursor se abre, y se pasa por IN OUT al proceso que lo solicita, y la lectura y cierre lo hace el Test.

Este es el tratamiento típico de cursores variables. La consulta se hace en el Bloque anónimo, y el tratamiento en el bloque o programa que solicita la información (p.ej, Java).

```

CREATE OR REPLACE
PACKAGE BODY PKT_GESTION_EMPL AS

    PROCEDURE IMP_EMPLEADOS(P_DEP EMPLOYEES.DEPARTMENT_ID%TYPE, P_CUANTOS OUT PLS_INTEGER) AS
        CURSOR CUR_EMPL IS
            SELECT LAST_NAME, SALARY, HIRE_DATE
            FROM EMPLOYEES
            WHERE DEPARTMENT_ID = P_DEP;

        BEGIN
            P_CUANTOS := 0;
            FOR REG_EMPL IN CUR_EMPL LOOP
                DBMS_OUTPUT.PUT('APELIDO : ' || RPAD(REG_EMPL.LAST_NAME,15));
                DBMS_OUTPUT.PUT('SALARIO : ' || LPAD(REG_EMPL.SALARY,8));
                DBMS_OUTPUT.PUT_LINE(' FECHA : ' || TO_CHAR(REG_EMPL.HIRE_DATE, 'DD-MM-YYYY'));
                --      P_CUANTOS := CUR_EMPL%ROWCOUNT;
                P_CUANTOS := P_CUANTOS + 1;
            END LOOP;
        END IMP_EMPLEADOS;

        FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE, P_FILA OUT DEPARTMENTS%ROWTYPE) RETURN BOOLEAN AS
        BEGIN
            SELECT *
            INTO P_FILA
            FROM DEPARTMENTS
            WHERE DEPARTMENT_ID = P_DEP;
            RETURN TRUE;
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                RETURN FALSE;
        END EXISTE_DEP;

        FUNCTION EXISTE_DEP(P_DEP DEPARTMENTS.DEPARTMENT_ID%TYPE) RETURN BOOLEAN AS
            v_cuantos pls_integer := 0;
        BEGIN
            SELECT COUNT(*)
            INTO V_CUANTOS
            FROM EMPLOYEES

```

```

WHERE DEPARTMENT_ID = P_DEP;
IF V_CUANTOS = 0 THEN
    DBMS_OUTPUT.PUT_LINE('EXISTE_DEP 1 PARAM');
    RETURN FALSE;
ELSE
    DBMS_OUTPUT.PUT_LINE('EXISTE_DEP 1 PARAM');
    RETURN TRUE;
END IF;
END EXISTE_DEP;

PROCEDURE CUR_ESPECIFICO( P_CURSOR IN OUT CUR_2V2, OPCION PLS_INTEGER) AS

BEGIN
CASE OPCION
WHEN 1 THEN
    OPEN P_CURSOR FOR SELECT LAST_NAME, FIRST_NAME FROM EMPLOYEES WHERE
DEPARTMENT_ID = 30;

```

Este es el bloque anónimo para probar el procedimiento, con las tres salidas correspondientes a las opciones:

- 4 (excepción opción incorrecta).
- 2 (lectura de trabajos que empiecen por 'S'.
- 1 (lectura de empleados del departamento 30.



Fíjate que las dos variables declaradas son de los tipos definidos en el paquete, tanto el cursor como el registro. Es una forma cómoda de trabajar, no me tengo que inventar tipos nuevos, y el paquete trabaja con sus tipos de forma óptima.



Al llamar al procedimiento

```
PKT_GESTION_EMPL.CUR_ESPECIFICO(MI_CURSOR, 3);
```

El procedimiento me vuelca, en la variable mi_cursor, la dirección de memoria del fichero que ha generado. Y ahora con un bucle loop end loop; trato la información.

```

SET SERVEROUTPUT ON
DECLARE
    MI_CURSOR  PKT_GESTION_EMPL.CUR_2V2;
    MI_REG      PKT_GESTION_EMPL.REG_2V2;
BEGIN
    PKT_GESTION_EMPL.CUR_ESPECIFICO(MI_CURSOR, 3);

    LOOP
        FETCH MI_CURSOR INTO MI_REG;
        EXIT WHEN MI_CURSOR%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('APELLIDO 0 JOB_ID : ' || MI_REG.CAMPO_1);
    END LOOP;

    IF MI_CURSOR%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('CURSOR VACÍO');
    END IF;

    CLOSE MI_CURSOR;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('ERROR GENERAL TEST_CUR_ESPECIFICO : ' || SQLERRM);
END;
/
Enter value for opcion: 4
ERROR GENERAL TEST_CUR_ESPECIFICO : ORA-20100: OPCION INCORRECTA, SOLO 1 o 2

PL/SQL procedure successfully completed.

Enter value for opcion: 2
APELIDO 0 JOB_ID : SA_MAN
APELIDO 0 JOB_ID : SA_REP
APELIDO 0 JOB_ID : ST_MAN
APELIDO 0 JOB_ID : ST_CLERK
APELIDO 0 JOB_ID : SH_CLERK

PL/SQL procedure successfully completed.

Enter value for opcion: 1
APELIDO 0 JOB_ID : Raphaely
APELIDO 0 JOB_ID : Khoo
APELIDO 0 JOB_ID : Baida
APELIDO 0 JOB_ID : Tobias
APELIDO 0 JOB_ID : Himuro
APELIDO 0 JOB_ID : Colmenares

PL/SQL procedure successfully completed.

```



Y ahora el paquete estaría completo.

Ventajas de los Paquetes PL/SQL

1

Modularidad.

Los paquetes pueden encapsular lógicamente tipos de datos y subprogramas en un módulo PL/SQL con nombre. Cada paquete es fácil de entender, y las interfaces con los paquetes son simples, claras y bien definidas, esto facilita el desarrollo de la aplicación.

2

Facilidad en el Diseño de la Aplicación.

Cuando diseñamos una aplicación, todo lo que inicialmente se necesita es la información de la interfaz en la especificación del paquete. No se necesita definir completamente el cuerpo del paquete hasta que no se complete la definición de la aplicación.

3

Ocultamiento de la Información.

En los paquetes, se pueden especificar tipos de datos y subprogramas para que sean públicos (visibles y accesibles) o privados (invisibles e inaccesibles). Por ejemplo, si tenemos un paquete que contiene cuatro subprogramas, tres públicos y uno privado. El paquete oculta la especificación del subprograma privado y solo implementa su código en el cuerpo del paquete.

4

Funcionalidad Agregada.

Las variables son persistentes para la sesión. Es decir, su valor se mantiene para toda la sesión del usuario que ejecuta ese paquete. Si otro usuario invoca el paquete, las variables contendrán el valor que inicialice el paquete, no los valores modificados por otro usuario.

5

Mejora Ejecución.

En la parte declarativa del cuerpo del paquete, opcionalmente, se puede inicializar las variables globales del cuerpo del paquete. Esta inicialización se ejecutará la primera vez que el paquete se coloque en memoria, es decir, la primera vez que un procedimiento del paquete sea invocado

6

Sobrecarga de subprogramas en paquetes.

PL/SQL permite dos o más subprogramas con el mismo nombre dentro del mismo paquete. Esta opción es usada cuando se necesita un subprograma igual que acepte parámetros que tienen diferentes tipos de datos.

