materiales didácticos de aula
formación profesional específica

Desarrollo de Aplicaciones Informáticas

CICLO FORMATIVO DE GRADO SUPERIOR

FORMACIÓN PROFESIONAL A DISTANCIA

Unidad 6

PL/SQL – Programación avanzada

MÓDULO

Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas CASE

FORMACIÓN PROFESIONAL
Principado de Asturias

Título del Ciclo: DESARROLLO DE APLICACIONES INFORMATICAS

Título del Módulo: DESARROLLO DE APLICACIONES EN ENTORNOS DE CUARTA GENERACIÓN Y CON HERRAMIENTAS CASE

Unidad 6: **PL/SQL - Programación avanzada**

Dirección: Dirección General de Políticas Educativas, Ordenación Académica y Formación Profesional
Servicio de Formación Profesional Inicial y Aprendizaje Permanente

Dirección de la obra:
Alfonso Careaga Herrera
Antonio Reguera García
Arturo García Fernández
Ascensión Solís Fernández
Juan Carlos Quirós Quirós
Luís Mª Palacio Junquera
Yolanda Álvarez Granda

Coordinador de los contenidos:

Juan Manuel Fernández Gutiérrez

Autores:
Juan Manuel Fernández Gutiérrez
Rodrigo Fernández Martínez

Colección:
Materiales didácticos de aula

Serie:
Formación profesional especifica

Edita:
Consejería de Educación y Ciencia

ISBN: 978-84-692-3443-2

Deposito Legal: AS-3564-2009

Copyright:
2009. Consejería de Educación y Ciencia

Esta publicación tiene fines exclusivamente educativos. Queda prohibida la venta, así como la reprodución total o parcial de sus contenidos sin autorización expresa de los autores y del Copyright

Introducción

En la unidad anterior hemos visto la funcionalidad básica del lenguaje PL/SQL. En esta unidad incorporamos aspectos más avanzados del lenguaje, como son: la gestión de excepciones, definición de procedimientos y funciones, procedimientos almacenados, paquetes estándar y disparadores.

Objetivos

- Controlar las excepciones.
- Crear subprogramas y paquetes almacenados en el catálogo de la base de datos.
- Construir disparadores de base de datos.
- Manejar paquetes estándar existentes en una base de datos Oracle.

Contenidos Generales

1. *Control de errores. Sección de excepciones*

En PL/SQL se denomina **excepción** a cualquier error que se produzca durante la ejecución del programa. Las excepciones pueden ser generadas internamente por el sistema o definidas por el usuario.

Cuando se produce un error, se activa una excepción y la ejecución normal del programa se detiene, pasándo el control a la sección de excepciones del bloque. Las excepciones internas son activadas automáticamente cuando se produce el error correspondiente. Las excepciones definidas por el usuario deben ser activadas explícitamente con la sentencia **RAISE.**

En la parte de excepciones del programa existirán distintas rutinas o ***exception handlers,*** cada una dedicada a procesar una determinada excepción. Cada rutina comienza con una cláusula **WHEN** donde se especifica la excepción o excepciones a las que se aplicará dicha rutina. Después de ejecutarse la rutina correspondiente a un error el proceso continúa con la siguiente sentencia del bloque contenedor. En el ejemplo siguiente, después de ejecutarse una rutina de excepción en el bloque ***dos***, se ejecutaría la siguiente sentencia (la que va detrás de este bloque) del bloque ***uno***.

```
<<uno>>
DECLARE
.............
BEGIN
sentencias;
<<dos>>
  DECLARE
    .............
  BEGIN
    sentencias
  EXCEPTION
    WHEN  excepción1  THEN sentencias;
    WHEN  excepción2  THEN sentencias;
     ...................
    WHEN  OTHERS  THEN sentencias;
  END;
sentencias;
EXCEPTION
...................
END;
```

Si en la parte de excepciones de un bloque no existe rutina de tratamiento de un determinado error, o si no existe sección de excepciones, el error pasa sin tratar al bloque contenedor, que intentará tratarlo en su sección de excepciones. De esta forma, un error no tratado va pasando de bloque en bloque hasta que es tratado o hasta que termina el programa con error. Si no queremos que se produzca la salida de un bloque con un error no tratado, incluiremos una rutina **WHEN OTHERS** en su sección de excepciones. En esta rutina se tratará cualquier excepción para la que no exista tratamiento específico.

El tratamiento de excepciones de PL/SQL clarifica el código del programa al separarlo del resto del proceso y definiendo claramente las acciones a realizar al producirse cada error.

1.1. Excepciones predefinidas.

Una excepción interna es activada automáticamente cuando el programa PL/SQL viola una regla de Oracle. Un ejemplo de excepción interna sería la que se produce cuando al ejecutar una sentencia SELECT ... INTO se recupera más de una fila. Cada error Oracle tiene un número, pero las cláusulas WHEN reconocen las excepciones por nombre. PL/SQL tiene predefinidas excepciones correspondientes a los errores más comunes, así, la excepción predefinida correspondiente al ejemplo anterior es TOO_MANY_ROWS.

EXCEPCIONES PREDEFINIDAS

Nombre de la Excepción	***Descripción***
CURSOR_ALREADY_OPEN	Se intenta abrir un cursor ya abierto.
DUP_VAL_ON_INDEX	Intento de insertar o actualizar violando la condición de unicidad de índice.
INVALID_CURSOR	Operación ilegal sobre un cursor.
INVALID_NUMBER	En una sentencia SQL se intentó realizar una conversión de cadena de caracteres a número y la cadena contenía caracteres no numéricos.
LOGIN_DENIED	Se intento iniciar una sesión de base de datos con nombre de usuario o password no válidos.
NO_DATA-FOUND	Una sentencia SELECT INTO no devuelve ninguna fila.
NOT_LOGGED_ON	PL/SQL realiza una llamada a Oracle y no existe conexión.
PROGRAM_ERROR	Error interno de PL/SQL.
STORAGE_ERROR	Se produce un error de memoria.

TIMEOUT_ON_RESOURCE	Se termina el tiempo de espera por un recurso.
TOO_MANY_ROWS	Una SELECT INTO devuelve más de una fila.
VALUE_ERROR	Se produjo un error aritmético o de conversión, un truncamiento o la violación de una restricción.
ZERO_DIVIDE	Se intentó dividir por cero.

Para manejar otros errores se puede usar la rutina **OTHERS**. Las funciones **SQLCODE** y **SQLERRM** son muy útiles en **OTHERS** porque devuelven el código del error Oracle y el mensaje correspondiente al error. También, se puede usar la pseudoinstrucción o **PRAGMA EXCEPTION_INIT** para asociar nombres de excepción a códigos de error ORACLE.

1.2. Excepciones definidas por el usuario.

El programador puede definir sus propias excepciones. Estas excepciones, al contrario que las internas, deben ser declaradas y activadas.

```
DECLARE
    cuota_nula EXCEPTION;  -- Declaración de la excepción.
    ..................
    ..................

BEGIN
    ..................
    ..................
       IF registro.cuota IS NULL THEN
          RAISE  cuota_nula;  -- Se activa la excepción cuota_nula.
    ..................
    ..................
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
          ................
          ................
    WHEN cuota_nula THEN      -- Rutina de tratamiento de la excepción cuota_nula.
          sentencias;
    WHEN OTHERS THEN
          sentencias;
END;
```

En el ejemplo, el programador trata la condición **registro.cuota IS NULL** como una excepción.

La sentencia **RAISE** se utiliza para activar excepciones previamente declaradas, aunque también admite el nombre de una excepción predefinida. Ya sabemos que las excepciones predefinidas se activan automáticamente, pero si queremos tratar una determinada condición con la rutina que trata un error predefinido podemos activar dicha excepción en la sentencia que

analiza la condición. Por ejemplo, si queremos tratar la condición **registro.cuota IS NULL** igual que la situación **NO_DATA_FOUND**, pondríamos

```
IF registro.cuota IS NULL THEN
    RAISE  NO_DATA_FOUND;
END IF;
```

1.3 Tratamiento de excepciones.

Cuando se activa una excepción el control pasa a la sección de excepciones. En esta sección habrá una o más cláusulas **WHEN**, cuyo formato es el siguiente:

WHEN nombre_excepción THEN
sentencias;

Las **sentencias** constituyen la rutina que tratará la excepción **nombre_excepción**. Cada cláusula WHEN termina cuando empieza la siguiente o cuando termina el bloque. Si queremos que una misma rutina trate varias excepciones, detrás de la palabra WHEN pondremos los nombres de esas excepciones separados por **OR**. La cláusula WHEN que como **nombre_excepción** lleva OTHERS contiene la rutina que tratará aquellas excepciones para las que no se dispusieron rutinas específicas.

Un error producido en la ejecución de la sección de excepciones no puede ser tratado en ella. En este caso se termina la ejecución del bloque con un error que se intentará tratar en algunos de los bloques externos.

En la declaración de variables pueden producirse excepciones al realizar inicializaciones incorrectas. Las excepciones producidas en la parte de declaraciones no pueden ser tratadas en el bloque y su activación da lugar a la finalización de la ejecución del bloque con una excepción no tratada. La siguiente declaración de variable daría lugar a la activación de una excepción:

```
maximo  CONSTANT  NUMBER(4) := 10000;
```

1.4. Tratamiento de excepciones con SQLCODE y SQLERRM.

En cualquier rutina de tratamiento de excepciones se puede utilizar **SQLCODE** para saber el código de error asociado a la excepción y **SQLERRM** para conocer el mensaje de error

estándar correspondiente a un determinado error, la longitud de SQLERRM es de 512 caracteres (puede variar con las versiones de Oracle).

En el caso de excepciones internas SQLCODE devuelve el número de error, número que es siempre negativo excepto en el caso de NO_DATA_FOUND que es el +100.

SQLERRM devuelve el correspondiente mensaje de error, que comienza siempre por el código de error de Oracle. En el caso de excepciones definidas por el usuario, SQLCODE devuelve +1 y SQLERRM el mensaje 'Excepción definida por el usuario'.

En el siguiente ejemplo se ve como se puede utilizar SQLCODE y SQLERRM

```
DECLARE
 num_error  NUMBER;
 men_error  CHAR(100);
BEGIN
 .........
 .........
EXCEPTION
 .........
 WHEN OTHERS THEN
       num_error :=  SQLCODE;
       men_error :=  SUBSTR(SQLERRM,1,100);
       INSERT INTO errores VALUES (num_error, men_error);
END;
```

Si a la función SQLERRM le pasamos un número de error como argumento, devuelve el mensaje de error asociado a ese número.

Ejemplos.

```
DECLARE
   mensaje VARCHAR2(80);
   BEGIN
   mensaje := SQLERRM(-2147);
   INSERT INTO errores VALUES (mensaje);
END;
/
Procedimiento PL/SQL terminado con éxito.
DECLARE
   mensaje VARCHAR2(80);
   BEGIN
   mensaje := SQLERRM(+100);
   INSERT INTO errores VALUES (mensaje);
END;
Procedimiento PL/SQL terminado con éxito.
```

```
SQL> SELECT * FROM errores;

NOMBRE
------------------------------------------------------------
ORA-02147: las opciones SHARED/EXCLUSIVE son incompatibles
ORA-01403: no se han encontrado datos
```

Aquellas excepciones internas que no tienen nombre asociado pueden tratarse con estas funciones desde la rutina WHEN OTHERS.

1.5. Tratamiento de excepciones con EXCEPTION_INIT.

Para tratar excepciones internas que carecen de nombre se puede utilizar una rutina OTHERS o también se puede utilizar la pseudoinstrucción EXCEPTION_INIT. Esta pseudoinstrucción o ***pragma***, le indica al compilador de PL/SQL que asocie un nombre de excepción con un número de error determinado. De esta forma, si en la ejecución del programa se produce ese error, lo podremos tratar como una excepción más con la correspondiente rutina detrás de **WHEN nombre_de_excepción THEN**.

En el siguiente bloque se trata el error Oracle -1626, que se produce cuando un segmento de rollback no tiene más capacidad para tratar transacciones. A través de la pseudoinstrucción EXCEPTION_INIT le asociamos la excepción **error_segmento**. Obsérvese que hay que declarar la excepción antes de asociarla con el error.

El formato de la pseudointrucción es:

PRAGMA EXCEPTION_INIT (nombre_de_excepción, número_error_Oracle)

La palabra PRAGMA indica al compilador que lo que sigue es una orden para él y no de una sentencia ejecutable.

```
DECLARE
     error_segmento  EXCEPTION;
     PRAGMA EXCEPTION_INIT (error_segmento, -1626);
BEGIN
     sentencias;
      /* Aquí irían algunas  de las sentencias que  pueden causar el error.
             * La excepción se dispara sola, ya que es una
             * excepción interna.
      */
```

```
EXCEPTION
      .............
      WHEN error_segmento THEN
            sentencias de tratamiento del error;
      .............
END;
```

ACTIVIDADES

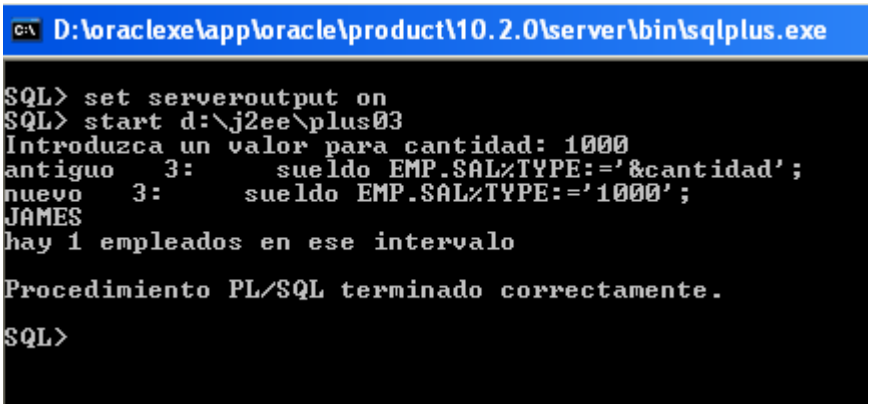
Obtener el nombre de los empleados que ganan +-100 euros de la cantidad que se introduce por teclado, junto con los nombres se visualizará el total de empleados que están en ese intervalo. Si no hay ninguno se visualizará: *"No hay ningún empleado en ese intervalo"*. Controlar cualquier otro error que se produzca.

SOLUCION

```
DECLARE

sueldo EMP.SAL%TYPE:='&cantidad';
CURSOR c1 IS SELECT ename FROM emp WHERE sal<sueldo+100 AND
                                              sal>sueldo-100;
contador NUMBER(2):=0;
num_error NUMBER:=0;
nom_error CHAR(100);

BEGIN

 FOR reg IN c1 LOOP
      DBMS_OUTPUT.PUT_LINE(reg.ename);
     Contador:=contador + 1;
 END LOOP;
 IF contador = 0 THEN
      RAISE NO_DATA_FOUND;
 END IF;
DBMS_OUTPUT.PUT_LINE('hay '||contador||' empleados en ese
intervalo');
 EXCEPTION
WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('no hay ningún empleado en ese
intervalo');
      WHEN OTHERS THEN
            num_error:=SQLCODE;
            nom_error:=SUBSTR(SQLERRM,1,100);
            DBMS_OUTPUT.PUT_LINE('error: '||num_error||' -
'||nom_error);
END;
/
```

'&sueldo' es una de las formas de cargar una variable en SqlPLus. También se puede cargar con *accept* fuera del bloque PL/SQL, en este caso hay que utilizar el entorno SQL*Plus.

DBMS_OUTPUT.PUT_LINE subprograma standard put_line del paquete dbms_output que sirve para visualizar el contenido de variables declaradas en el bloque PL/SQL y/ literales. En este caso visualiza el contenido del contador y en caso de que se produzcan excepciones los literales correspondientes. Este programa va asociado a la activación en el entorno de la variable **set serveroutput on,** ver la imagen capturada. Lo mismo que con el comando *accept* estos comandos hay que manejarlos desde el entorno SQL*Plus.



El programa opera de la siguiente forma: primero se carga la variable sueldo con la cantidad que se teclea, luego se define un cursor para recuperar los nombres de los empleados que están en el intervalo válido, a continuación se recorre el cursor con la sentencia *for* y se van contando los empleados al mismo tiempo que se visualizan. Por último si se produce alguna excepción el programa bifurca a la sección de excepciones.

2. SUBPROGRAMAS.

Hasta aquí hemos estado tratando con un tipo de bloque PL/SQL denominados bloques anónimos. Otro tipo de bloques son los subprogramas que se diferencian de los anteriores en que tienen un nombre por el que pueden ser invocados y en esta invocación se les pueden pasar parámetros.

Hay dos tipos de subprogramas: procedimientos y funciones. Los procedimientos se utilizan normalmente para realizar una acción y las funciones para realizar un cálculo.

2.1. Procedimientos.

Un procedimiento tiene la siguiente sintaxis:

```
PROCEDURE nombre  [(parámetro [,parámetro, ..])]  IS
[declaración local]
 BEGIN
      Sentencias;
[EXCEPTION
      tratamiento de excepciones]
END [nombre];
```

A su vez, **parámetro** tiene la siguiente sintaxis:

```
Parámetro [IN  | OUT | IN OUT] tipo_dato [{:=  | DEFAULT } expresión]
```

En el **tipo_dato** no se permite especificar la longitud, por ejemplo, se especificará NUMBER y no NUMBER(8). En los parámetros tampoco se permite la restricción NOT NULL.

Un procedimiento tiene dos partes: la especificación y el cuerpo. La especificación va entre las palabras PROCEDURE e IS y en ella se indica el nombre del procedimiento y, opcionalmente, los parámetros a través de los que se pasarán valores cuando se le invoque. Si un procedimiento carece de parámetros no llevará paréntesis tras el nombre.

El cuerpo comienza a continuación de IS y consta de las tres partes de cualquier bloque: parte declarativa, opcional, que se extiende hasta BEGIN; parte ejecutable, obligatoria, entre BEGIN y EXCEPTION; y parte de manejo de excepciones, opcional, hasta END.

Los parámetros de un procedimiento o función pueden ser de tipo IN, que es el tipo por defecto, de tipo OUT y de tipo IN OUT. Los de tipo IN se utilizan para pasar valores al procedimiento en la invocación, los de tipo OUT para que el procedimiento devuelva valores a quien le invoca y los de tipo IN OUT realizan la doble función de pasar y devolver valores.

Los parámetros son variables accesibles desde el cuerpo del procedimiento y que admiten distintas formas de uso según el tipo.

Los procedimientos, igual que las funciones, deben ser declarados antes de poder invocarse. Se les llama desde cualquier parte de la sección ejecutable del bloque PL/SQL.

```
DECLARE
.................

PROCEDURE  proc1 (cuotas   NUMBER,  nombre  VARCHAR2  OUT)  IS

BEGIN
  ...............
  ...............
END proc1;
PROCEDURE  proc2 .........
  ...............
  ...............
END  proc2;

---------------------------- programa principal ----------------

BEGIN
  ................
  proc1 (789, denominacion);
  ...........
  proc2;
  ..........
END;
```

2.2. Funciones

Las funciones son subprogramas que realizan una serie de operaciones y que devuelven un único valor. Las funciones pueden formar parte de expresiones. Su sintaxis es semejante a la de los procedimientos, aunque tienen una cláusula RETURN.

```
FUNCTION nombre [(parámetros [,parámetro, …])] RETURN tipo_dato IS
[declaraciones locales]
BEGIN
      Sentencias;
[EXCEPTION
      tratamiento  de excepciones;]
END  [nombre];
```

A su vez, **parámetro** tiene la siguiente sintaxis:

```
Parámetro [IN  | OUT | IN OUT] tipo_dato [{:=  | DEFAULT } expresión]
```

Como los procedimientos, las funciones tienen dos partes: especificación y cuerpo. La especificación se define entre FUNCTION e IS y el cuerpo comprende el resto. Una función debe tener en su parte ejecutable al menos una sentencia **RETURN expresión_PL/SQL**

Al ejecutarse la sentencia RETURN termina la ejecución de la función y el valor de la expresión es asignado al identificador de la función en la expresión del bloque desde el que se invocó.

Puede haber varias sentencias RETURN con lo que la función dispondrá de varios puntos de finalización.

La cláusula RETURN de la especificación de la función indica el tipo de dato devuelto por la función. El valor resultante de la evaluación de la expresión de la sentencia RETURN debe ser compatible con ese tipo. Un procedimiento puede llevar también una o más sentencias RETURN, pero sin expresión.

Los subprogramas admiten sobrecarga, o sea, la declaración con el mismo nombre de varios subprogramas que difieren en el número, orden o tipo de sus parámetros. PL/SQL distingue las llamadas a los distintos subprogramas por las características de sus argumentos.

Ejemplo

```
DECLARE
       Total  NUMBER(8);
       FUNCTION  neto
              (bp   facturas.base_imponible%TYPE,
               iv    facturas.iva%TYPE,
               ds   facturas.descuento%TYPE)
               RETURN  NUMBER IS
                 resul   NUMBER(8);
       BEGIN
              IF ds IS NULL
                     resul := bp ;
              ELSE
                     resul := bp – (bp * ds  / 100);
              END IF;
                     resul := resul + resul * iv /100;
              RETURN resul;
       END;

/*La sentencia RETURN puede devolver una expresión tipo: RETURN  (bp – bp * ds  /  100)  */
BEGIN
       .......
       total := neto(base_imponible, iva, descuento);
       .......
END;
```

2.3. Declaración de subprogramas.

Los subprogramas se pueden declarar en cualquier bloque PL/SQL, subprograma o package (paquete), pero se deben declarar después de todos los objetos que se hayan definido en la sección DECLARE

No se puede llamar a un subprograma que no este previamente declarado, pero a veces la lógica del programa obliga a llamar a un subprograma que aún no se declaró. Este es el caso de dos subprogramas que se invocan uno a otro.

Ejemplo.

```
DECLARE
      .......
      PROCEDURE proc1   ......  IS
      .......
      BEGIN
            .........
            proc2  ;   -- no está declarado, luego no se puede llamar.
            ........
      END;

      PROCEDURE  proc2  ......IS
      ...........
      BEGIN
            ..........
      END;
```

Se puede solucionar realizando una declaración **previa** o **forward declaration**. Este tipo de declaraciones contiene solo la especificación del subprograma e indica al compilador que se hará una declaración completa mas adelante.

```
DECLARE

      PROCEDURE proc2  (.....);  -- Declaración previa.

      PROCEDURE proc1   ...... IS
      ……….
```

```
BEGIN
      proc2  ;   -- sí está previamente declarado, luego se puede llamar.
END;

PROCEDURE  proc2  ..... IS
.............
BEGIN
      .......  -- declaración completa  del procedimiento previamente declarado
END;
```

En ambas declaraciones, la previa y la completa, debe aparecer integra la especificación del subprograma.

Los subprogramas pueden ser almacenados en la base de datos como un objeto mas. Una vez almacenados en el catálogo pueden ser invocados desde cualquier herramienta o programa.

Para crear un subprograma y almacenarlo en el catálogo de la base de datos se debe ejecutar la sentencia CREATE PROCEDURE o CREATE FUNCTION desde SQL*Plus o cualquier otra herramienta que las admita.

La sintaxis de estas sentencias es la misma que las de las sentencias PROCEDURE y FUNCTION de PL/SQL, con la excepción de la palabra CREATE que las inicia.

La siguiente sentencia crea y almacena un procedimiento para borrar las actividades que no tienen cuota.

```
CREATE PROCEDURE  borra_actividades  AS
BEGIN
      DELETE FROM actividades WHERE cuota IS NULL;
END;
```

Para llamar al procedimiento

```
BEGIN
      borra_actividades
END;
```

No deben aparecer variables globales en los procedimientos y tampoco se deben pasar variables con el mismo nombre, puede dar lugar a que el resultado sea indeterminado.

2.4. Uso de los parámetros.

Los subprogramas reciben y devuelven información a través de los parámetros. Los parámetros que aparecen en la especificación del subprograma pueden utilizarse, con ciertas restricciones, en su cuerpo.

Los argumentos son las variables o expresiones que aparecen en la invocación o llamada al subprograma. Existe una correspondencia entre argumentos y parámetros. Cada argumento debe tener un tipo compatible con el de su parámetro.

```
DECLARE
      .........
      .........
PROCEDURE incr_cuota ( coeficiente NUMBER, acti  CHAR)  IS
      ............
      ............
BEGIN
      ............
END;
BEGIN
      ...........
      incr_cuota(valor/10,  'JUDO');

/*
Se evalúan los argumentos y sus valores se asignan a los correspondientes
parámetros
*/
END;
```

En este ejemplo *coeficiente* y *acti* son parámetros, *valor/10* y 'JUDO' son argumentos. En este caso existe una correspondencia posicional entre argumentos y parámetros pero se podría haber usado una correspondencia por nombre

```
incr_cuota(acti => 'JUDO', coeficiente => valor/10);
```

o mixta

```
incr_cuota(valor/10, acti =>'JUDO');
```

Cuando se declara algún parámetro con valor inicial, este es el valor que toma el parámetro si no se le pasa argumento. O sea, se puede omitir algún argumento en la llamada a

un subprograma si el parámetro correspondiente fue definido con valor inicial. Solo admiten inicialización los parámetros de tipo IN.

Ejemplo

```
DECLARE
      base         facturas.base_imponible%TYPE;
      desc         facturas.descuento%TYPE
      PROCEDURE  neto (bs  INTEGER,  ds  REAL DEFAULT 5.0)  IS
            .........
      BEGIN
            ........
      END;
BEGIN
      ..........
      neto(base);  -- Solo se pasa valor para bs, luego ds tomará el valor 5.0
      .........
END;
```

Un parámetro tiene distintas características según se declare de tipo IN, OUT o IN OUT. Sabemos que el tipo por defecto es IN.

En el cuadro siguiente vemos las características de los distintos tipos.

IN	OUT	IN OUT
Pasa un valor al subprograma	Devuelve un valor al módulo llamante	Inicialmente pasa un valor al subprograma y al final devuelve otro valor al módulo llamante
Dentro del subprograma actúa como una constante	Actúa como una variable no inicializada	Actúa como una variable inicializada
No se le puede asignar valor dentro del cuerpo del subprograma	No puede ser usado en expresiones y debe asignársele un valor dentro del cuerpo del subprograma	Puede asignársele o no un valor dentro del cuerpo del subprograma
El argumento puede ser una constante, variable inicializada, literal o expresión	El argumento debe ser una variable	El argumento debe ser una variable

ACTIVIDAD

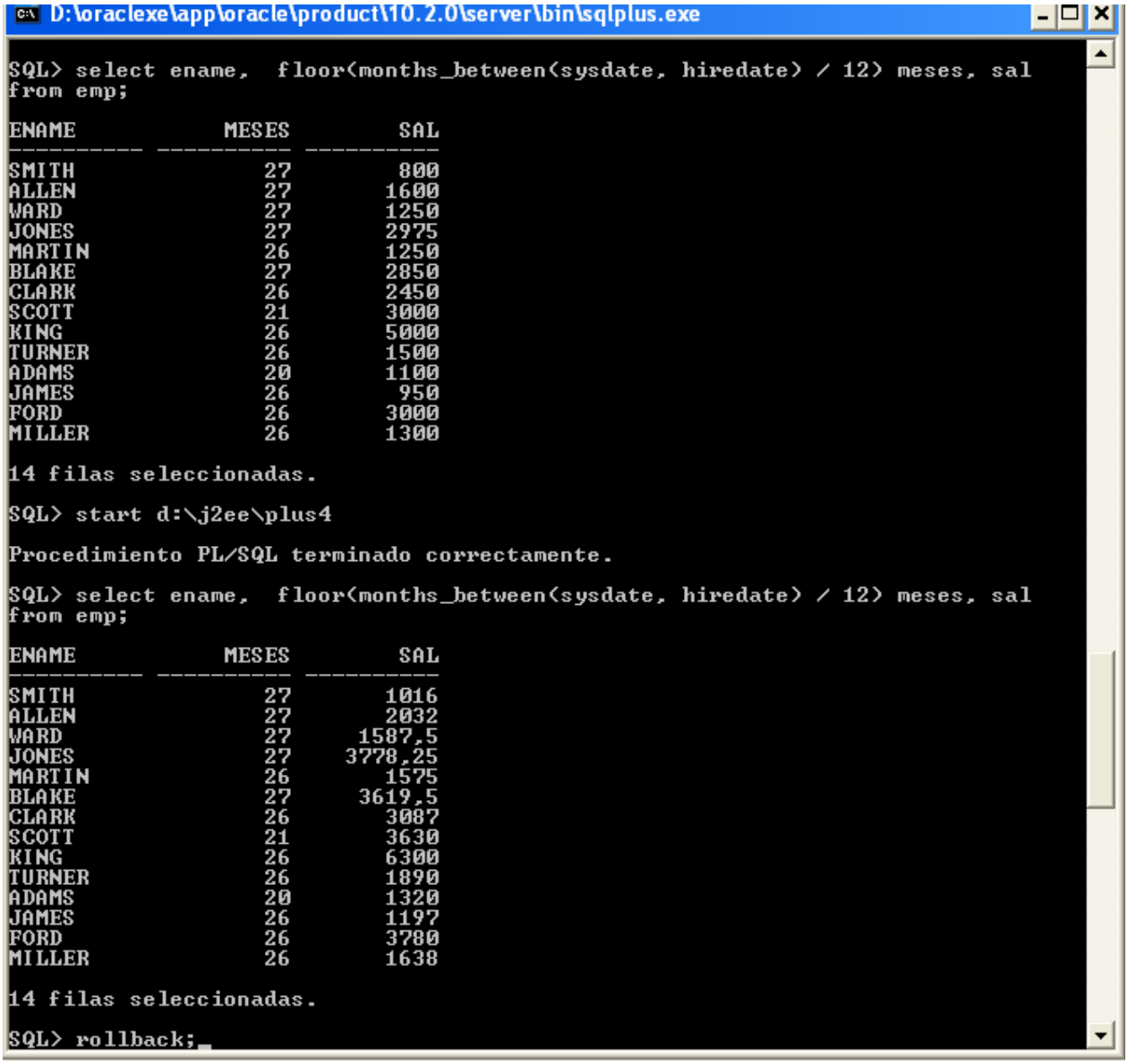
Programa que actualiza el salario de los empleados incrementándolo en el tanto por ciento equivalente a su antigüedad en años. La antigüedad la calcula una función.

SOLUCION

```
DECLARE
ant   NUMBER(4);
CURSOR c1  IS   SELECT  * FROM emp FOR  UPDATE;
//función que se define en la declaración del programa principal
FUNCTION antig(fecha_alta DATE) RETURN NUMBER IS
BEGIN
       RETURN (FLOOR(MONTHS_BETWEEN(SYSDATE, fecha_alta) / 12));
END antig;
// programa principal
BEGIN
FOR reg IN c1 LOOP
  ant := antig(reg.hiredate);
  UPDATE emp SET sal =(sal * (1 + ant/100)) WHERE CURRENT OF c1;
END LOOP;
END;
```

En la imagen capturada de SqlPLus se ve como está la tabla antes de ejecutar el programa, se detalla para cada empleado los meses de antigüedad y el salario.

Se ejecuta el bloque, se guardó en el archivo *plus4.sql,* y se repite la consulta de la tabla donde se puede ver el incremento de los salarios en base a la antigüedad en años.



Vamos hacer un ejemplo con funciones y procedimientos almacenados. Como ya hemos comentado constituyen objetos del esquema.

Creamos un procedimiento llamado *nuevo_emp* para insertar un empleado nuevo en la tabla *emp.* El procedimiento llamará a una función *valida_dept* para comprobar que el departamento del nuevo empleado existe en la tabla departamentos.

Primero se crea la función *valida_dept* para comprobar que existe el departamento. Recibe como parámetro el número del departamento.

```
CREATE OR REPLACE FUNCTION valida_dept (num_dept IN dept.deptno%TYPE)
                                                    RETURN BOOLEAN IS
x CHAR;
BEGIN
     SELECT 'x' INTO x FROM dept WHERE deptno = num_dept;
     RETURN (TRUE);
EXCEPTION
      WHEN NO_DATA_FOUND THEN RETURN (FALSE);
END valida_dept;
/
```

Suponiendo que lo hayamos guardado como *plus05.sql,* se ejecuta con *start d:\j2ee\plus05.sql*

Si hubiese errores se visualizan con SHOW ERRORS.

El código del procedimiento almacenado que inserta el nuevo empleado es el siguiente:

```
CREATE OR REPLACE PROCEDURE nuevo_emp (num emp.empno%TYPE, nom
                            emp.ename%TYPE, depar emp.deptno%TYPE) IS
BEGIN
IF valida_dept(depar) THEN
   INSERT INTO emp (empno,ename,deptno) VALUES (num,nom,depar);
ELSE
          DBMS_OUTPUT.PUT_LINE('Departamento inválido ');
END IF;
END nuevo_emp;
/
```

El procedimiento inserta los valores que recibe, el resto de las columnas asumen nulos.

Se le puede llamar desde SqlPlus de dos formas:

Como bloque PL/SQL

```
BEGIN
  nuevo_emp(11,'ANA',20);
END;
```

Con el comando de SqlPlus EXECUTE

```
EXECUTE nuevo_emp(11,'ANA',20);
```

Las siguientes capturas muestra los dos casos posibles uno que inserta y otro no

La siguiente pantalla muestra la inserción del empleado 11 en el departamento 20 que existe y por lo tanto lo da de alta.

```
SQL> EXECUTE nuevo_emp(11,'ANA',20);

Procedimiento PL/SQL terminado correctamente.
```

La siguiente inserción intenta dar de alta un empleado con el departamento 88, al no existir no lo inserta y visualiza el mensaje *departamento inválido*

```
SQL> BEGIN
  2  nuevo_emp(12,'LAURA',88);
  3  END;
  4  /
Departamento invßlido

Procedimiento PL/SQL terminado correctamente.
```

3. Paquetes

Un paquete es un objeto de la base de datos que agrupa lógicamente definiciones de tipos de datos, variables, constantes, excepciones, cursores y subprogramas, que tienen cierta relación entre ellos.

Un paquete tiene dos partes que se definen y almacenan por separado: la especificación y el cuerpo. Los paquetes que no incluyen cursores ni subprogramas carecen de cuerpo.

La especificación contiene la declaración de todos los objetos del paquete. En el caso de los subprogramas solo contiene una declaración previa.

El cuerpo contienen la especificación completa de cursores y subprogramas.

La especificación del paquete es pública, o sea, es visible para las aplicaciones, mientras que el cuerpo, que contiene los detalles de la implementación de los objetos, es privado y no visible para las aplicaciones.

Para crear y almacenar la especificación de un paquete se usan las sentencias CREATE PACKAGE y CREATE PACKAGE BODY, cuya sintaxis es:

```
CREATE [OR REPLACE] PACKAGE  nombre   {IS | AS}
 Declaraciones de tipos
 Variables
 Constantes
 Excepciones
 Especificación de cursores
 Especificaciones de subprogramas
END  [nombre];
      CREATE  [OR  REPLACE]  PACKAGE  BODY  nombre  {AS  |  IS}
 Declaraciones privadas de variables, constantes, cursores, excepciones
 Declaración de cuerpos de cursores
subprograma (procedimiento, función)
BEGIN
             Sentencias del subprograma
END;
   [BEGIN
             ..........................
             ..........................]  -- Parte de inicialización (*)
    END [nombre ];
```

La parte de inicialización del cuerpo (*) se ejecuta solo la primera vez que se invoca un objeto del paquete y suele utilizarse para inicializar ciertos objetos del paquete. Esta parte es opcional.

En el cuerpo del paquete se pueden declarar objetos como variables, constantes, excepciones, etc. que son privados y no accesibles por las aplicaciones.

La definición de un cursor que se realiza en la especificación del paquete tiene la siguiente sintaxis:

```
CURSOR nombre_cursor [(parámetro[, parámetro]...)]
RETURN tipo_registro;
```

El tipo de registro se puede especificar dando el nombre de un tipo de registro ya definido o mediante el atributo %ROWTYPE.

El cuerpo del cursor se declara en el cuerpo del paquete según la siguiente sintaxis:

```
CURSOR nombre_cursor [(parámetro [, parámetro]...)]
            RETURN tipo_registro    IS    consulta;
```

Ejemplo.

El siguiente paquete está compuesto de un procedimiento que actualiza la cuota de los usuarios que tienen código de banco y son de un determinado distrito, y una función que calcula la antigüedad en trienios.

```
CREATE  OR REPLACE PACKAGE   pq1   AS
    PROCEDURE actualizar
        (distrito      VARCHAR2,
     cuota        NUMBER);
     FUNCTION trienio  (fecha DATE) RETURN NUMBER;
     END pq1;

CREATE  OR REPLACE PACKAGE BODY  pq1  AS
     PROCEDURE actualizar
          (distrito      VARCHAR2,
           cuota         NUMBER)   IS
     BEGIN
            UPDATE usuarios SET cuota_socio = cuota
              WHERE distrito = cp AND codigo_banco IS NOT NULL;
     END  actualizar;

     FUNCTION   trienio  (fecha  DATE)  RETURN  NUMBER  IS
      resul NUMBER;
     BEGIN
      resul :=TRUNC((TO_CHAR(SYSDATE,'YYDDD')- TO_CHAR(fecha,'YYDDD'))/3000);
      return  resul;
     END   trienio;
END pq1;
```

Para referenciar los objetos de un paquete se empleará la notación

nombre_paquete.nombre_objeto

En el ejemplo anterior se hará referencia al procedimiento y a la función como sigue:

```
pq1.actualizar('33440', 7000);

antiguedad := pq1.trienio(fecha_alta);
```

4. Algunos paquetes disponibles en Oracle

En una base de datos Oracle que disponga de PL/SQL existirá un paquete de nombre STANDARD. Este paquete contiene las declaraciones de los tipos de datos, excepciones y subprogramas utilizables automáticamente desde cualquier programa PL/SQL. Por ejemplo, todas las funciones predefinidas de PL/SQL se encuentran en el paquete STANDARD. La especificación de este paquete es visible para cualquier aplicación PL/SQL.

También existen en Oracle otros paquetes que facilitan la construcción de aplicaciones PL/SQL, como DBMS_STANDARD, DBMS_XDB, DBMS_SQL o DBMS_OUTPUT.

El paquete **DBMS_STANDARD** suministra facilidades que ayudan a las aplicaciones a interactuar con Oracle. Entre los procedimientos incluidos en este paquete nos vamos a fijar en RAISE_APPLICATION_ERROR. Este procedimiento cuya especificación es:

procedure raise_application_error(num binary_integer, msg varchar2,
keeperrorstack boolean default FALSE);

facilita al programador la generación de errores Oracle desde subprogramas almacenados. El parámetro num es el número de error devuelto y es definido por el programador en el rango -20000 .. -20999. El parámetro msg es el mensaje de error devuelto y es una cadena de caracteres que puede tener una longitud de hasta 2048 bytes. El tercer parámetro, keeperrorstack, es opcional ya que tiene un valor por defecto, marca la forma de pasar el error al sistema.

Cuando se invoca, RAISE_APPLICATION_ERROR termina el subprograma, realiza un ROLLBACK y devuelve el número y mensaje de error pasados como argumentos. Este error puede ser tratado como cualquier otro error Oracle.

RAISE_APPLICATION_ERROR solo puede ser invocado desde subprogramas almacenados.

El paquete **DBMS_OUTPUT** permite que un subprograma almacenado, paquete o disparador emita mensajes. Los procedimientos de este paquete PUT y PUT_LINE permiten escribir los mensajes en un buffer, que pueden ser leídos por otro subprograma, paquete o disparador.

Desde SQL*Plus se pueden visualizar mensajes introducidos en el buffer por los procedimientos anteriores. Para que dicha visualización sea posible la variable de SQL*Plus SERVEROUTPUT debe estar en ON.

Para acceder a la información contenida en el buffer desde un bloque PL/SQL se emplean los procedimientos GET_LINE o GET_LINES.

Los mensajes enviados mediante DBMS_OUTPUT no se envían realmente hasta que la ejecución del subprograma o disparador termina.

Este paquete es especialmente útil para mostrar información mientras se realiza la depuración del programa PL/SQL.

El paquete **DBMS_SQL** permite la ejecución desde PL/SQL de sentencias DDL de SQL y sentencias SQL dinámicas.

El ejemplo que sigue hace uso de este paquete para ejecutar, desde un procedimiento PL/SQL, una sentencia DDL. La sentencia es una DROP SYNONYM cuyo argumento, el nombre del sinónimo a eliminar, no se conoce hasta el momento de su ejecución.

El ejemplo hace uso de tres subprogramas definidos en DBMS_SQL:

La función OPEN_CURSOR, que permite crear un cursor para ejecutar posteriormente una sentencia SQL. Esta función devuelve un entero que identifica al cursor abierto.

El procedimiento PARSE, que compila una sentencia SQL y si se trata de una sentencia DDL la ejecuta también. Al llamar a este procedimiento se le deben dar tres argumentos:

1º.- El número del cursor donde queremos que compile y ejecute.

2º.- La sentencia a compilar.

3º.- La especificación de cómo Oracle debe manejar la sentencia. Puede especificarse con las constantes V6, V7 o NATIVE.

El procedimiento CLOSE_CURSOR que cierra el cursor cuyo número se le pasa como argumento.

Mediante este procedimiento se eliminan del catálogo de la base de datos todos los sinónimos sobre objetos del usuario SYSCASE. El procedimiento debe ser ejecutado por un usuario con los privilegios adecuados.

```
CREATE PROCEDURE borrar_sinonimos AS
       CURSOR sinonimo IS
              SELECT synonym_name, owner
              FROM sys.dba_synonyms
              WHERE table_owner = 'SYSCASE';
/* El cursor sinonimo permite obtener los nombres de todos
```

```
 * los sinónimos propiedad de SYSCASE.
 */
      cid    INTEGER;
BEGIN
      FOR registro IN sinonimo LOOP
        cid := DBMS_SQL.OPEN_CURSOR; -- Se abre el cursor.
        DBMS_SQL.PARSE(cid, 'DROP SYNONYM '|| registro.owner||
       '.'||registro.synonym_name, DBMS_SQL.V7); --Se borra un sinónimo.
        DBMS_SQL.CLOSE_CURSOR(cid);
      END LOOP;
END
/
```

El siguiente procedimiento elimina cualquier objeto del esquema cuyo tipo se le pasa como parámetro, devuelve el número de objetos borrados. No elimina tablas maestras relacionadas dado que le falta la opción *cascade constraints* para este tipo de objetos.

```
CREATE OR REPLACE PROCEDURE borra_objetos
      (tipo USER_OBJECTS.OBJECT_TYPE%TYPE, contador OUT number ) AS
id INTEGER;
CURSOR c1 IS SELECT object_name
                  FROM   user_objects
                  WHERE  object_type=tipo;
BEGIN
 contador := 0;
 FOR reg IN c1 LOOP
      id:=DBMS_SQL.OPEN_CURSOR;
      DBMS_SQL.PARSE(id,'DROP '|| tipo || ' ' || reg.object_name, DBMS_SQL.V7);
      contador := contador + 1;
      DBMS_SQL.CLOSE_CURSOR(id);
 END LOOP;
END;
/
```

El procedimiento se puede ejecutar desde el entorno SqlPlus de la siguiente forma:

```
    DECLARE
    borrados NUMBER;
    BEGIN
       borra_objetos('VIEW',borrados);
       DBMS_OUTPUT.PUT_LINE(borrados);
    END;
    /
```

El primer parámetro que se le pasa es el tipo tipo de objeto que se quiere borrar, en el segundo se recibe el número de objetos eliminados que muestra con el subprograma PUT_LINE .

5. Disparadores de la base de datos.

Un disparador es un procedimiento PL/SQL almacenado asociado a una tabla que Oracle ejecuta automáticamente cuando se realiza una determinada operación sobre la tabla. Con un disparador, por ejemplo, se puede controlar que a determinadas tablas no se acceda en determinados días, o bien que no se realicen determinadas operaciones por determinados usuarios, etc

Son tres los eventos que hacen que se dispare un trigger: ***insert, delete, update*.** Pudiendo elegir si se dispara antes o después de producirse uno o varios de los eventos anteriores.

Sintaxis para crear disparadores de la base de datos

```
CREATE [OR REPLACE] TRIGGER nombre {BEFORE | AFTER}
{
[DELETE]
[[OR] INSERT]
[[OR] UPDATE [ OF columnas...]]}
}
ON tabla
      [REFERENCING {OLD AS nuevo_nombre | NEW AS nuevo_nombre ... ]
                        [FOR EACH ROW [ WHEN condición]]

DECLARE
   Declaraciones;
BEGIN
   Sentencias;
END;
```

Con las opciones BEFORE o AFTER se le indica si se dispara antes o después de producirse el evento.

Los eventos pueden ser: la inserción de nuevas filas, la actualización de todas o algunas de las columnas y el borrado de todas o algunas filas. Se puede especificar más de un evento separándolos por

OR.

Cuando el trigger solo se dispara para determinadas filas se empleará la opción **FOR EACH ROW WHEN condición,** en este caso podemos hacer referencia al valor nuevo o viejo de la columna que interviene en la comparación utilizando como prefijo :NEW o :OLD respectivamente(p. e. :NEW.columna > expresión). En el caso de que se quiera cambiar los prefijos NEW y OLD por otros nombres utilizamos la cláusula REFERENCING (p.e. REFERENCING NEW AS nuevo_nombre) Ahora ya se puede utilizar *nuevo_nombre* como prefijo de la columna en la condición.

Ejemplo.

Vamos a controlar qué usuarios manipulan la tabla *actividades* (insertan, borran o actualizan) y cuándo lo hacen.

```
1º Creamos el disparador, para lo que es necesario tener el privilegio CREATE TRIGGER
si lo creamos en nuestro esquema, o el de CREATE ANY TRIGGER si se crea en cualquier
otro esquema.

CREATE TRIGGER controla_act
    BEFORE   DELETE OR INSERT OR UPDATE ON  actividades
      DECLARE
        id_us  number(8);
        nom_us varchar2(30);
      BEGIN
        id_us := UID;
        nom_us := USER;
        INSERT INTO  tab_controla  VALUES (id_us, nom_us, sysdate);
END;
/

Disparador creado.
```

Cuando se inserta, actualiza o borran filas de la tabla de actividades se activa el disparador e inserta una fila en la tabla tab_controla con la identificación del usuario y la fecha.

TABLAS UTILIZADAS EN LOS EJEMPLOS

```
CREATE TABLE BANCOS
(
  ENT_SUC      NUMBER(8) NOT NULL,
  NOMBRE       VARCHAR2(50),
  DIRECCION    VARCHAR2(50),
  LOCALIDAD    VARCHAR2(30),
  TELEFONOS    VARCHAR2(30),
  CONSTRAINT CLAVE_PRIMARIA_BANCOS PRIMARY KEY(ENT_SUC)
);

CREATE TABLE USUARIOS
(
  NUM_SOCIO       VARCHAR2(7) NOT NULL,
  DNI             VARCHAR2(9) NOT NULL,
  NOMBRE          VARCHAR2(20),
  APELLIDOS       VARCHAR2(30),
  FOTOGRAFIA      LONG RAW,
  DOMICILIO       VARCHAR2(40),
  LOCALIDAD       VARCHAR2(50),
  CP              VARCHAR2(5),
  FECHA_NACIMIENTO  DATE,
  TELEFONO        VARCHAR2(20),
  TAQUILLA        VARCHAR2(15),
  HORARIO         VARCHAR2(15),
  FECHA_ALTA      DATE,
  FECHA_BAJA      DATE,
  CUOTA_SOCIO     NUMBER(7),
  CUOTA_FAMILIAR  NUMBER(7),
  PAGA_BANCO      CHAR NOT NULL
       CONSTRAINT CHEQUEO1 CHECK (PAGA_BANCO IN ('S','N')),
  CODIGO_BANCO    NUMBER(8),
  CUENTA          NUMBER(10),
  DIGITO_CONTROL  NUMBER(2),
  OBSERVACIONES   VARCHAR2(500),
  CONSTRAINT CLAVE_PRIMARIA_USUARIOS PRIMARY KEY (NUM_SOCIO),
```

```
  CONSTRAINT CLAVE_ALTERNATIVA_USUARIOS UNIQUE (DNI),
  CONSTRAINT CLAVE_AJENA_BANCOS FOREIGN KEY (CODIGO_BANCO)
            REFERENCES BANCOS(ENT_SUC)
);

CREATE TABLE ACTIVIDADES
(
  CODIGO_ACTIVIDAD   VARCHAR2(7) NOT NULL,
  DESCRIPCION        VARCHAR2(50),
  CUOTA              NUMBER(7),
  CONSTRAINT CLAVE_PRIMARIA_ACTIVIDADES PRIMARY KEY (CODIGO_ACTIVIDAD)
);

CREATE TABLE ACTIVIDADES_USUARIOS
(
  CODIGO_ACTIVIDAD    VARCHAR2(7) NOT NULL,
  CODIGO_USUARIO      VARCHAR2(7)NOT NULL,
  FECHA_ALTA          DATE,
  FECHA_BAJA          DATE,
  CONSTRAINT CLAVE_PRIMARIA_ACT_USU
          PRIMARY KEY(CODIGO_ACTIVIDAD,CODIGO_USUARIO),
  CONSTRAINT CLAVE_AJENA_ACT  FOREIGN KEY (CODIGO_ACTIVIDAD)
            REFERENCES  ACTIVIDADES(CODIGO_ACTIVIDAD),
  CONSTRAINT CLAVE_AJENA_USU FOREIGN KEY(CODIGO_USUARIO)
          REFERENCES  USUARIOS(NUM_SOCIO)
);









CREATE TABLE PAGOS
(
  CODIGO_USUARIO   VARCHAR2(7) NOT NULL,
  NUMERO_MES       NUMBER(2) NOT NULL,
  CUOTA            NUMBER(7),
  OBSERVACIONES    VARCHAR2(500),
```

```
  CONSTRAINT CLAVE_PRIMARIA_PAGOS
          PRIMARY KEY(CODIGO_USUARIO, NUMERO_MES),
  CONSTRAINT CLAVE_AJENA_PAG_USU FOREIGN KEY(CODIGO_USUARIO)
      REFERENCES USUARIOS(NUM_SOCIO) ON DELETE CASCADE
);

CREATE TABLE USUARIOS_ASOCIADOS
(
  CODIGO_USUARIO    VARCHAR2(7) NOT NULL,
  USUARIO_ASOCIADO  VARCHAR2(7) NOT NULL,
  CONSTRAINT CLAVE_PRIMARIA_USUARIOS_ASOC
          PRIMARY KEY(CODIGO_USUARIO, USUARIO_ASOCIADO),
  CONSTRAINT CLAVE_AJENA_USU_USU FOREIGN KEY(CODIGO_USUARIO)
          REFERENCES USUARIOS(NUM_SOCIO),
  CONSTRAINT CLAVE_AJENA_USU_ASOC FOREIGN KEY(USUARIO_ASOCIADO)
          REFERENCES USUARIOS(NUM_SOCIO)
);

Insert into bancos (ent_suc, nombre, direccion, localidad) values (
30700018,'BANESTO','MANUEL LLANEZA, 33','MIERES');

Insert into bancos (ent_suc, nombre, direccion, localidad) values (
20480070,'CAJA DE ASTURIAS','MANUEL LLANEZA, 17','MIERES');

Insert into bancos (ent_suc, nombre, direccion, localidad) values (
43000250,'HERRERO','MANUEL LLANEZA, 22','MIERES');

Insert into bancos (ent_suc, nombre, direccion, localidad) values (
85002222,'SANTANDER','LA CAMARA, 13','AVILES');

Insert into bancos (ent_suc, nombre, direccion, localidad) values (
22223333,'BBV','LA RIBERA, 17','LUANCO');

Insert into bancos (ent_suc, nombre, direccion, localidad) values (
33334444,'ATLANTICO','GIJON, 56','LUANCO');

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
     localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
```

```
    cuenta, digito_control)
    values ('A1111','01111111','JUAN LUIS','ARIAS ALVAREZ',
    'C/ LA VEGA, 18','MIERES','33600',to_date('10/01/1996','dd/mm/yyyy'),
    5500,'S',30700018,1111,10);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'A2222','02222222','INES','PEREZ DIAZ',
    'C/DOCTOR FLEMING, 14','MIERES','33600',to_date('10/01/1996','dd/mm/yyyy'),
    6800,'S',20480070,2222,47);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco)
    values ( 'A3333','03333333','JOSE','RUIZ PEÑA',
    'C/ AVILES, 18','LUANCO', '33400',to_date('21/05/1996','dd/mm/yyyy'),
    6000,'N');

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'I1111','00111111','MARIA','FERNANDEZ SANTAMARINA',
    'PLAZA DE LA CONSTITUCION, 3','MIERES','33600',
    to_date('10/11/1998','dd/mm/yyyy'),7000,'S',43000250,1234567890,21);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'I2222','77896542Y','MARTA','ARIAS SANTOLAYA',
    'C/ ASTURIAS, 23 4º','MIERES','33600',to_date('12/12/1997','dd/mm/yyyy'),
    4500,'S',20480070,2242,41);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J1111','11111111','ANA','GUTIERREZ ALONSO',
    'C/ ASTURIAS, 51 4º','LUANCO','33440',to_date('02/03/1998','dd/mm/yyyy'),
    5000,'S',85002222,2342,61);
```

```
Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J2222','22222222','LAURA','FERNANDEZ ALONSO',
    'C/ TEVERGA, 17 2º','LUANCO','33440',to_date('11/12/1997','dd/mm/yyyy'),
    5000,'S',85002222,234255,34);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J3333','33333333','MARIA','ALONSO GUTIERREZ',
    'C/ OVIEDO, 8 1º','LUANCO','33440',to_date('10/11/1998','dd/mm/yyyy'),
    6000,'S',22223333,425566,43);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J4444','44444444','MANUEL','ALONSO OVIES',
    'C/ OVIEDO, 18 4º','LUANCO','33440',to_date('05/06/1996','dd/mm/yyyy'),
    6000,'S',22223333,425566,43);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J5555','55555555','RAMON','ARBOLEYA GARCIA',
    'C/ LANGREO, 9 1º','AVILES','33400',to_date('03/04/1995','dd/mm/yyyy'),
    4000,'S',85002222,4343,12);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J6666','66666666','DOLORES','MORENO RODRIGUEZ',
    'C/ OVIEDO, 23 6º','AVILES','33400',to_date('03/03/1998','dd/mm/yyyy'),
    4000,'S',22223333,6675,21);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
```

```
        cuenta, digito_control)
        values ( 'J7777','77777777','PABLO','RODRIGUEZ ARIAS',
        'C/ LA FLORIDA, 3 6º','AVILES','33400',to_date('06/09/1997','dd/mm/yyyy'),
        4000,'S',33334444,6775,12);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
        localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
        cuenta, digito_control)
        values ( 'J8888','88888888','MARTA','FERNANDEZ GONZALEZ',
        'PLAZA SAN JUAN, 9','MIERES','33600',to_date('06/09/1997','dd/mm/yyyy'),
        7000,'S',33334444,9975,11);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
        localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
        cuenta, digito_control)
        values ( 'J9999','99999999','LUIS','BULNES BALBIN',
        'C/ LA VEGA, 49 2º','MIERES','33600',to_date('14/11/1996','dd/mm/yyyy'),
        7000,'S',33334444,39975,22);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
        localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
        cuenta, digito_control)
        values ( 'J1010','10101010','JOSE','ALVAREZ CASTRO',
        'C/ OÑON, 23 2º','MIERES','33600',to_date('01/04/1995','dd/mm/yyyy'),
        6000,'S',33334444,93375,42);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
        localidad, cp, fecha_alta, cuota_socio, paga_banco)
        values ( 'J0011','11011011','PELAYO','ESPANDES CASARIEGO',
        'C/ LA PISTA, 14 1º','MIERES','33600',to_date('01/05/1997','dd/mm/yyyy'),
        8000,'N');

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
        localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
        cuenta, digito_control)
        values ( 'J0012','12121212','VICTOR','ALBA PRIETO',
        'C/ LA LILA, 49 2º','AVILES','33400',to_date('01/05/1998','dd/mm/yyyy'),
        8000,'S',85002222,54679,32);
```

```
Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J0013','13131313','LUZDIVINA','CUETO ARROYO',
    'C/ NAYOR, 91 5º','AVILES','33400',to_date('01/06/1995','dd/mm/yyyy'),
    7000,'S',22223333,66785,32);

Insert into usuarios (num_socio, dni, nombre, apellidos, domicilio,
    localidad, cp, fecha_alta, cuota_socio, paga_banco, codigo_banco,
    cuenta, digito_control)
    values ( 'J0014','14141414','MARIO','FERNANDEZ VEGA',
    'C/ PEZ, 19 2º','AVILES','33400',to_date('01/04/1998','dd/mm/yyyy'),
    3000,'S',33334444,3354679,53);

Insert into actividades values ('G000001', 'GIMNASIA DE MANTENIMIENTO', 1000);

Insert into actividades values ('G000002','GIMNASIA RITMICA',800);

Insert into actividades values ('AM00001','JUDO',1200);

Insert into actividades values ('AM00002','KARATE',1100);

Insert into actividades values ('N000001','NATACION 1',900);

Insert into actividades values ('N000002','NATACION 2',1000);

Insert into actividades values ('G000003','MUSCULACION',1300);

Insert into actividades_usuarios (codigo_actividad, codigo_usuario, fecha_alta) values (
'G000001','A1111',to_date('30/03/1999','dd/mm/yyyy'));

Insert into actividades_usuarios (codigo_actividad, codigo_usuario, fecha_alta) values (
'AM00002','A1111',to_date('30/03/1999','dd/mm/yyyy'));

Insert into actividades_usuarios (codigo_actividad, codigo_usuario, fecha_alta) values (
'AM00001','A2222',to_date('30/03/1999','dd/mm/yyyy'));
```

```
Insert into actividades_usuarios (codigo_actividad, codigo_usuario, fecha_alta) values (
'N000001','A2222',to_date('30/03/1999','dd/mm/yyyy'));

Insert into actividades_usuarios (codigo_actividad, codigo_usuario, fecha_alta) values (
'N000002','I2222',to_date('01/03/1999','dd/mm/yyyy'));

Insert into actividades_usuarios (codigo_actividad, codigo_usuario, fecha_alta) values (
'AM00002','I2222',to_date('11/02/1999','dd/mm/yyyy'));

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A1111',1,5000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A1111',2,5000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A1111',3,5000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A1111',4,5500);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A1111',5,5500);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A1111',6,5000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',1,7800);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',2,7800);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',5,7800);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',6,8000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',3,7800);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',4,9000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I1111',5,4500);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I1111',7,5000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I1111',1,3456);
```

```
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A2222',7,8000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('A3333',2,4500);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I2222',1,5000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I2222',2,5000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I2222',3,5500);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I2222',4,5500);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('I2222',6,6000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J1111',1,1000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J1111',2,10000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J1111',4,10000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J1111',5,10000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J1111',12,10000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J2222',12,8000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J2222',1,1000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J2222',3,10000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J2222',11,10000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',1,1000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',2,2000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',3,3000);
```

```
Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',4,4000);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',5,500);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',11,800);

Insert into pagos (codigo_usuario, numero_mes,cuota) values ('J3333',12,1500);

commit;
```

Desarrollo de Aplicaciones Informáticas

materiales didácticos de aula

UNIÓN EUROPEA
Fondo Social Europeo

GOBIERNO DEL PRINCIPADO DE ASTURIAS
CONSEJERÍA DE EDUCACIÓN Y CIENCIA

FORMACIÓN PROFESIONAL
Principado de Asturias