Ampliación de Bases de Datos

Práctica 3: PL/SQL

Introducción

- SQL es un lenguaje 4GL
  - describe lo que debe hacerse, pero no cómo hacerse

relación OPERACIÓN relación = relación

- C, JAVA, COBOL,... son más procedimentales (3GL)
  - Poseen variables, estructuras de control, bucles,...

- PL/SQL combina 3GL y 4GL

Características principales

- La unidad básica de PL/SQL es el **bloque**.
- Tipos de bloques:
  - Anónimos
    - Se construyen de forma dinámica
    - Se ejecutan 1 vez
  - Nominados
    - Anónimo + <<etiqueta>>
  - Subprogramas
    - Nominados = Procedimientos, funciones, paquetes
    - Se almacenan en la BD
    - Se ejecutan múltiples veces, mediante llamadas
  - Disparadores
    - Nominados
    - Se almacenan en la BD
    - Se ejecutan cada vez que tiene lugar el suceso de disparo

Estructura de un bloque PL/SQL

```
[DECLARE]
    /* Sección declarativa - variables, tipos, cursores y subprogramas */

BEGIN
    /* Sección ejecutable - órdenes PL/SQL */

[EXCEPTION]
    /* Sección de manejo de excepciones */

END;
```

Ejemplos de bloques

```
BEGIN
        NULL;
END;
/
```

No hace nada

Provoca la ejecución

```
SET SERVEROUTPUT ON
BEGIN
  DBMS_OUTPUT.PUT_LINE('Estamos a '|| SYSDATE);
  DBMS_OUTPUT.PUT('a ');
  DBMS_OUTPUT.PUT('b');
  DBMS_OUTPUT.NEW_LINE;
END;
/
Estamos a 10-ENE-05
a b
```

escritura en pantalla

≅ *println*

concatena cadenas

≅ *print*

salto de línea

Unidades léxicas

- Etiquetas
  - Identifican partes del código
  - NO usar para GOTO
- Literales
  - De carácter: ‘literal’
  - Numérico
  - Booleano: true, false, null
- Comentarios
  - Monolínea: --
  - Multilínea: /* */

Variables y tipos

<nombre_variable> [CONSTANT] <tipo> [NOT NULL] [{:= | DEFAULT} <valor por defecto>];

letras, números, $, _, #

inicialización

- Tipos escalares
  - Numéricos (enteros o reales): NUMBER, DECIMAL, DOUBLE PRECISION, INTEGER, NUMERIC, REAL, SMALLINT, BINARY_INTEGER
  - De carácter: VARCHAR2, CHAR
  - Raw: almacena datos binarios de longitud fija
  - Fecha: DATE
  - Rowid: almacena identificadores de tuplas
  - Booleanos: almacenan true, false o null
- Tipos compuestos
  - Registros, tablas y arrays de variables escalares
- Tipos de referencia
  - ≅ puntero
- Tipos LOB
  - Almacena objetos de hasta 4 GB
- Tipos definidos por el usuario

Ejemplo de declaración de variables

```
DECLARE
    num3            NUMBER := 3;
    num6            CONSTANT NUMBER NOT NULL := 6;

    TYPE t_RegistroEstudiante IS RECORD (   Nombre    VARCHAR2(20),
                                            Apellido1 VARCHAR2(20),
                                            Apellido2 VARCHAR2(20),
                                            DNI       VARCHAR2(10));

    v_Estudiante    t_RegistroEstudiante;

    Sueldo          JUGADOR.SALARIO%TYPE;
    OtroSueldo      Sueldo%TYPE;
    regEquipo       EQUIPO%ROWTYPE;        -- Ahora podemos acceder a regEquipo.Categoría, etc.
BEGIN
    ............

END;
```

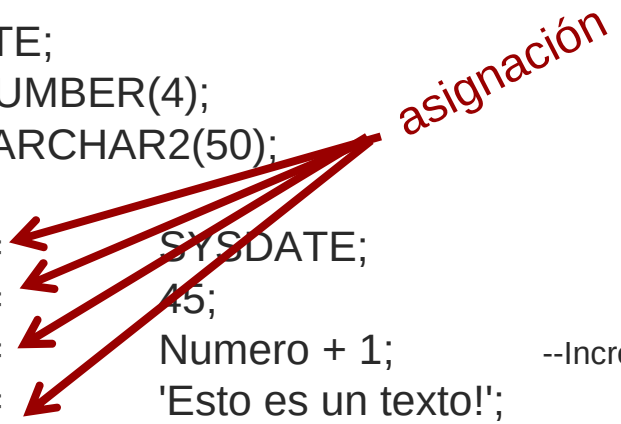
constante

Tipo definido por el usuario

Manipulación de variables

1. Asignación Directa

```
DECLARE
  Fecha DATE;
  Numero NUMBER(4);
  Cadena VARCHAR2(50);
BEGIN
  Fecha    :=          SYSDATE;
  Numero   :=          45;
  Numero   :=          Numero + 1;          --Incrementa Numero en 1, ahora vale 46
  Cadena   :=          'Esto es un texto!';
  DBMS_OUTPUT.PUT_LINE('Estamos a '|| SYSDATE);
  DBMS_OUTPUT.PUT_LINE(Numero);
  DBMS_OUTPUT.PUT_LINE(Cadena);
END;
```



Ejemplo de un bloque PL/SQL

1. Asignación Directa
2. Mediante sentencias DML de SQL
   PERO... NO SE PUEDEN EJECUTAR SENTENCIAS DDL

```
DECLARE
      regEquipo      EQUIPO%ROWTYPE; -- Ahora podemos acceder a regEquipo.Categoría, etc.
BEGIN
      SELECT *       INTO regEquipo
      FROM EQUIPO
      WHERE CODEQUIPO = 'HCL';
END;
```

Ejemplo de manipulación de variables

Usando variables:

-- Modificar los datos del jugador de DNI 12.345.678, indicando que ha estado en 3 equipos

-- Insertar un nuevo equipo

```
DECLARE
        NumEquipos EQUIPO.OTROSEQUIPOS%TYPE;
        regEquipo EQUIPO%ROWTYPE;
BEGIN
        NumEquipos := 3;
        UPDATE JUGADOR
                SET OTROSEQUIPOS = NumEquipos
                WHERE DNI = '12.345.678';

        regEquipo.CODEQUIPO             :='INV';
        regEquipo.NOMBRE_EQUIPO         :='Equipo Inventado';
        regEquipo.CATEGORIA             :='No hay';
        regEquipo.PRESUPUESTO           :=0;
        regEquipo.EQPRINCIPAL           :=NULL;
        regEquipo.CP                    :=NULL;
        INSERT INTO EQUIPO
                VALUES regEquipo;
END;
```

Estructuras de control

```
IF <condición 1> THEN
    /* secuencia de acciones a ejecutar si condición 1 es cierta */
[ELSIF <condición 2> THEN
    /* secuencia de acciones a ejecutar si condición 2 es cierta (y condición 1 no) */
 ... ]
[ELSE
    /* secuencia de acciones a ejecutar si todas las condiciones son falsas o nulas */
END IF;
```

Estructuras de control

Imprimir la variable Numero si vale 3.
Indicar también si la condición no es cierta

```
IF Numero = 3 THEN
        DBMS_OUTPUT.PUT_LINE('La variable Numero vale 3');
ELSE
        DBMS_OUTPUT.PUT_LINE('La variable Numero NO vale 3');
END IF;
```

Indicar si un número es menor que 5, está entre 5 y 10, o es mayor que 10:

```
DECLARE
  Numero NUMBER;
BEGIN
        /*  ... La variable Numero toma un valor ... */
  IF Numero < 5 THEN              -- Si (Numero<5) es cierto...
        DBMS_OUTPUT.PUT_LINE('Es menor que 5');
  ELSIF Numero <= 10 THEN         -- Si (Numero < 5) no es cierto y (Numero <= 10) sí lo es...
        DBMS_OUTPUT.PUT_LINE('Está entre 5 y 10');
  ELSE                            -- Si todas las condiciones anteriores son falsas...
        DBMS_OUTPUT.PUT_LINE('Es mayor que 10');
  END IF;
END;
```

Estructuras de control

```
IF <condición 1> THEN
    /* secuencia de acciones a ejecutar si condición 1 es cierta */
[ELSIF <condición 2> THEN
    /* secuencia de acciones a ejecutar si condición 2 es cierta (y condición 1 no) */
 ... ]
[ELSE
    /* secuencia de acciones a ejecutar si todas las condiciones son falsas o nulas */
END IF;
```

```
LOOP
        /* sentencias */
    EXIT [<etiqueta>] WHEN <condición>
END LOOP;
```

Ejemplos de utilización de LOOP

Mostrar secuencialmente los números comenzando en 1

```
DECLARE
  contador BINARY_INTEGER := 1;
BEGIN
  LOOP
        DBMS_OUTPUT.PUT_LINE(contador);
        contador := contador + 1;
  END LOOP;
END;
```

Bucle infinito !!

Imprimir los números del 1 al 10

```
DECLARE
  contador BINARY_INTEGER := 1;
BEGIN
  LOOP
        DBMS_OUTPUT.PUT_LINE(contador);
        contador := contador + 1;

        IF contador > 10 THEN
              EXIT;
        END IF;
  END LOOP;
END;
```

EXIT WHEN contador >10

Ejemplos de utilización de LOOP

Ejemplo con bucles anidados

```
DECLARE
        cont1 BINARY_INTEGER := 1;
        cont2 BINARY_INTEGER := 1;
BEGIN
        <<externo>>
         LOOP
                DBMS_OUTPUT.PUT_LINE('Bucle externo: ‘ || cont1);
                cont2 := 1;
                <<interno>>
                LOOP
                        DBMS_OUTPUT.PUT(' Int:‘ || cont2);
                        cont2 := cont2 + 1;
                        EXIT interno WHEN cont2 > cont1;
                END LOOP;
                DBMS_OUTPUT.NEW_LINE;
                cont1 := cont1 + 1;
                EXIT externo WHEN cont1 > 10;
        END LOOP;
END;
```

Estructuras de control

```
IF <condición 1> THEN
    /* secuencia de acciones a ejecutar si condición 1 es cierta */
[ELSIF <condición 2> THEN
    /* secuencia de acciones a ejecutar si condición 2 es cierta (y condición 1 no) */
 ... ]
[ELSE
    /* secuencia de acciones a ejecutar si todas las condiciones son falsas o nulas */
END IF;
```

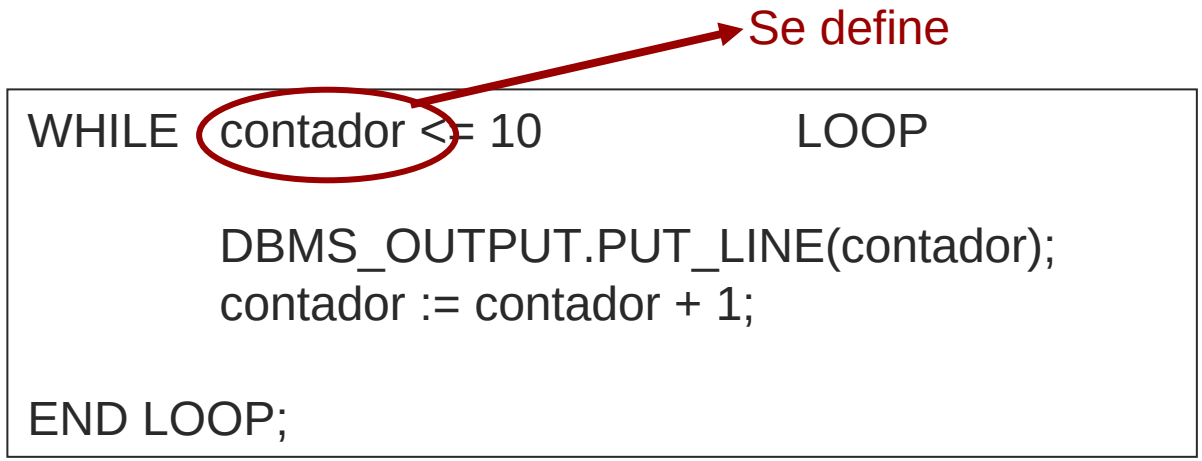
```
LOOP
        /* sentencias */
    EXIT [<etiqueta>] WHEN <condición>
END LOOP;
```

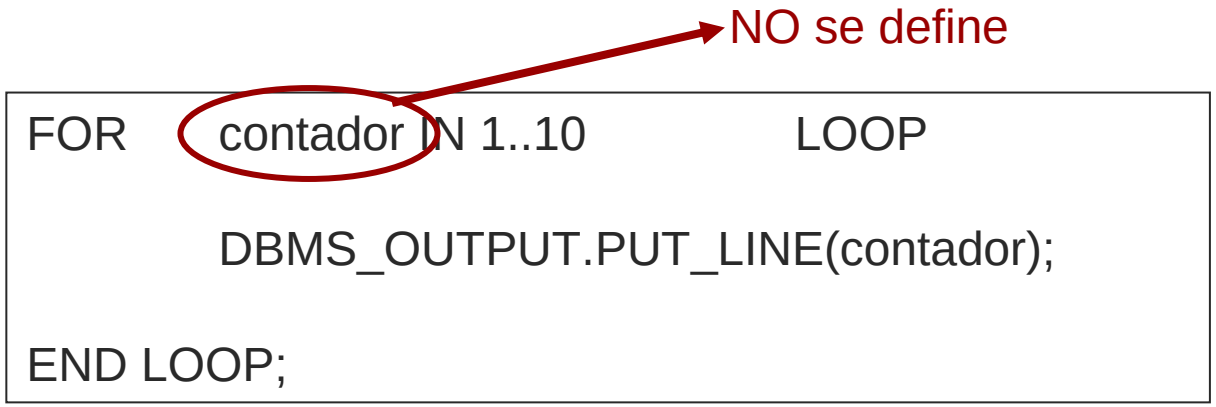
```
WHILE <condición> LOOP
        /* sentencias */
END LOOP;
```

```
FOR <variable> IN [REVERSE] <limite_inferior>..<limite_superior> LOOP
        /* sentencias */
END LOOP;
```

Ejemplos de utilización de WHILE y FOR

Imprimir los números del 1 al 10



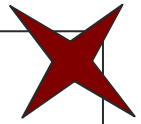


Consultas que devuelven 1 fila

- Se puede recoger el resultado de SELECT **si devuelve 1 fila**:

```
DECLARE
        NumEquipos NUMBER;
        regEquipo EQUIPO%ROWTYPE;
BEGIN
        SELECT OTROSEQUIPOS
                INTO NumEquipos
                FROM JUGADOR
                WHERE DNI='12.345.678';
         SELECT  *
                INTO regEquipo
                FROM EQUIPO
                WHERE CODEQUIPO='HCL';
END;
```

- En caso contrario (0 ó *n* tuplas) .... ERROR!!



```
DECLARE
  regEquipo EQUIPO%ROWTYPE;
BEGIN
  SELECT *
        INTO regEquipo
        FROM EQUIPO
        WHERE CODEQUIPO='ABC';
END;
```

```
DECLARE
  regEquipo EQUIPO%ROWTYPE;
BEGIN
  SELECT *
        INTO regEquipo
        FROM EQUIPO;
END;
```

Consultas que devuelven 1 fila

- Número total de jugadores existentes y la suma de sus salarios
- Información sobre el equipo cuyo código es HCL

```
DECLARE
   CantidadDeJugadores          NUMBER(2);
   SumaDeSalario                NUMBER(8,2);
   regEquipo                    EQUIPO%ROWTYPE;
BEGIN
   SELECT COUNT(*), SUM(SALARIO)
            INTO CantidadDeJugadores, SumaDeSalario
            FROM JUGADOR;
   DBMS_OUTPUT.PUT_LINE('Hay '|| CantidadDeJugadores);
   DBMS_OUTPUT.PUT_LINE('En total cobran '|| SumaDeSalario);

   SELECT *
            INTO regEquipo
            FROM EQUIPO
            WHERE CODEQUIPO = 'HCL';
   DBMS_OUTPUT.PUT_LINE(    'Código: ‘ || regEquipo.CODEQUIPO ||
                            ' Nombre:‘ || regEquipo.NOMBRE_EQUIPO);
   DBMS_OUTPUT.PUT_LINE(    'Categoría: ‘ || regEquipo.CATEGORIA ||
                            ‘ Presupuesto:‘ || regEquipo.PRESUPUESTO);
   DBMS_OUTPUT.PUT_LINE(    'Equipo Principal: ‘ || regEquipo.EQPRINCIPAL || ' CP: ‘ || regEquipo.CP);
END;
```

Cursores

- Se emplean para procesar múltiples tuplas
- Oracle asigna a cada consulta un *área SQL*
- Cursor = puntero al *área SQL*
- Manejo de un cursor:

1. Declarar

```
CURSOR <nombre_cursor> IS <sentencia select>;
```

2. Abrir (ejecutar la consulta)

```
OPEN <nombre_cursor>;
```

3. Procesar

```
LOOP
        FETCH <nombre_cursor> INTO <lista_variables>|<registro>;
        EXIT WHEN <nombre_cursor>%NOTFOUND;
END LOOP;
```

4. Cerrar

```
CLOSE <nombre_cursor>;
```

Ejemplo de cursor

Nombre, salario y número de equipos en los que han estado los jugadores del equipo HCL (variable)

```
DECLARE
        Nombre              JUGADOR.NOMBRE_JUGADOR%TYPE;
        Salario             JUGADOR.SALARIO%TYPE;
        NumeroEquipos       JUGADOR.OTROSEQUIPOS%TYPE;
        CodigoEquipo        JUGADOR.CODEQUIPO%TYPE;
        CURSOR C_JUG IS
                SELECT NOMBRE_JUGADOR, SALARIO, OTROSEQUIPOS
                FROM JUGADOR
                WHERE CODEQUIPO = CodigoEquipo;
BEGIN
        CodigoEquipo := 'HCL';
        OPEN C_JUG;
        LOOP
                FETCH C_JUG INTO Nombre, Salario, NumeroEquipos;
                EXIT WHEN C_JUG%NOTFOUND;
                DBMS_OUTPUT.PUT_LINE(Nombre || ' ‘ || Salario || ' ‘ || NumeroEquipos);
        END LOOP;
        CLOSE C_JUG;
END;
```

Cursores

- Atributos:
  - <nombre_cursor>%FOUND => cierto si el último FETCH devolvió una fila
    falso en caso contrario
  - <nombre_cursor>%NOTFOUND => cierto si FETCH no devuelve una fila
    falso si la devuelve
  - <nombre_cursor>%ISOPEN => cierto si el cursor está abierto
    falso en caso contrario
  - <nombre_cursor>%ROWCOUNT => nº filas extraídas por el cursor

- Cursores implícitos (cursor SQL)
  - Para INSERT, UPDATE, DELETE y SELECT…INTO (de 1 fila)
  - No se utilizan CURSOR, OPEN, FETCH ni CLOSE
  - Se pueden aplicar atributos del cursor (SQL%NOTFOUND, SQL%FOUND, …)

Cursores implícitos

FACTURA (Codigo, Fecha, Total)
LINEA_FACTURA (Codigo, Linea, Precio)

Se desea:
-- insertar una línea e incrementar el total de la factura con el precio de la nueva línea,
-- si es la primera línea de la factura crear la tupla en la tabla de facturas

```
DECLARE
  CodigoFactura       LINEA_FACTURA.CODIGO%TYPE := ...;
  Fecha               DATE :=SYSDATE;
  NumeroLinea         LINEA_FACTURA.LINEA%TYPE := ...;
  PrecioLinea         LINEA_FACTURA.PRECIO%TYPE := ...;
BEGIN
  INSERT INTO LINEA_FACTURA (CODIGO, LINEA, PRECIO)
            VALUES(CodigoFactura, NumeroLinea, PrecioLinea);
  UPDATE FACTURA
            SET TOTAL = TOTAL + PrecioLinea
            WHERE CODIGO = CodigoFactura;
       -- La variable SQL%NOTFOUND se activa si no se han procesado filas
  IF SQL%NOTFOUND THEN
            INSERT INTO FACTURA(CODIGO, FECHA, TOTAL)
                      VALUES(CodigoFactura, Fecha, PrecioLinea);
  END IF;
END;
```

Bucles sobre cursores

"Nombre y salario de los jugadores"

- WHILE

```
DECLARE
  Nombre    JUGADOR.NOMBRE_JUGADOR%TYPE;
  Salario   JUGADOR.SALARIO%TYPE;
  CURSOR C_JUG IS
            SELECT NOMBRE_JUGADOR, SALARIO
            FROM JUGADOR;
BEGIN
  OPEN      C_JUG;
  FETCH     C_JUG INTO Nombre, Salario;
  WHILE     C_JUG%FOUND       LOOP
            DBMS_OUTPUT.PUT_LINE(Nombre || ' cobra  ' || Salario);
            FETCH C_JUG INTO Nombre, Salario;
  END LOOP;
  CLOSE C_JUG;
END;
```

- FOR

```
FOR <registro> IN <nombre cursor> LOOP
            /* Procesar la fila actual */
END LOOP;
```

- No se define

- No utiliza OPEN / FETCH / CLOSE

```
DECLARE
  CURSOR C_JUG IS
            SELECT NOMBRE_JUGADOR, SALARIO
            FROM JUGADOR;
BEGIN
  FOR regJUGADOR IN C_JUG LOOP
    DBMS_OUTPUT.PUT_LINE(regJUGADOR.nombre_jugador
            || ' cobra ' || regJUGADOR.salario);
  END LOOP;
END;
```

Cursores actualizables

- Se utilizan las sentencias:
  - CURSOR <nombre_cursor> IS SELECT … FOR UPDATE
  - UPDATE … WHERE CURRENT OF <nombre_cursor>

Ej: Aumentar el presupuesto en un 6% para equipos con menos de 3 000 000

```
DECLARE
   CURSOR C_EQUIPO IS
              SELECT *
              FROM EQUIPO
              WHERE PRESUPUESTO < 3000000
              FOR UPDATE;
BEGIN
   FOR regEquipo IN C_EQUIPO LOOP
              DBMS_OUTPUT.PUT_LINE(regEquipo.codequipo || ' ' || regEquipo.presupuesto);
              UPDATE EQUIPO SET PRESUPUESTO = PRESUPUESTO * 1.06
                        WHERE CURRENT OF C_EQUIPO;
    END LOOP;
END;
```

Manejo de excepciones

- Variables:
  - SQLCODE = código de error (0 si es correcto, nº negativo si error)
  - SQLERRM = mensaje de error
- Manejo de excepciones:

  WHEN <nombre_excepción> THEN <sentencias>;
- Excepciones con nombre:
  - NO_DATA_FOUND (si no devuelve filas)
  - TOO_MANY_ROWS (si devuelve más de 1 fila)
  - OTHERS (en otro caso)

Ej: Mensaje de error si el resultado de una consulta a EQUIPO devuelve más de 1 fila

```
DECLARE
   regEquipo EQUIPO%ROWTYPE;
BEGIN
   SELECT *
            INTO regEquipo
            FROM EQUIPO;
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
            DBMS_OUTPUT.PUT_LINE('Hay más de una fila');
END;
```

Manejo de excepciones

- Variables:
  - SQLCODE = código de error (0 si es correcto, nº negativo si error)
  - SQLERRM = mensaje de error
- Manejo de excepciones:

  WHEN <nombre_excepción> THEN <sentencias>
- Excepciones con nombre:
  - NO_DATA_FOUND (si no devuelve filas)
  - TOO_MANY_ROWS (si devuelve más de 1 fila)
  - OTHERS (en otro caso)

Ej: Controlar la excepción al realizar una consulta de los equipos cuyo presupuesto sea ‘ABC’

```
DECLARE
   regEquipo EQUIPO%ROWTYPE;
BEGIN
   SELECT *
     INTO regEquipo
     FROM EQUIPO
     WHERE PRESUPUESTO = 'ABC';
EXCEPTION
   WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE(‘No existe presupuesto');
   WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Se ha producido un error');
END;
```

Manejo de excepciones

- Variables:
  - SQLCODE = código de error (0 si es correcto, nº negativo si error)
  - SQLERRM = mensaje de error
- Manejo de excepciones:

  WHEN <nombre_excepción> THEN <sentencias>
- Excepciones con nombre:
  - NO_DATA_FOUND (si no devuelve filas)
  - TOO_MANY_ROWS (si devuelve más de 1 fila)
  - OTHERS (en otro caso)

Ej: Gestión genérica

```
DECLARE
   regEquipo EQUIPO%ROWTYPE;
BEGIN
   SELECT *
     INTO regEquipo
     FROM EQUIPO
     WHERE PRESUPUESTO = 'ABC';
EXCEPTION
     WHEN OTHERS THEN
          DBMS_OUTPUT.PUT_LINE('Código: ‘ || SQLCODE);
          DBMS_OUTPUT.PUT_LINE('Mensaje: ‘ || SUBSTR(SQLERRM, 11,100));
END;
```

Manejo de excepciones

- Excepciones de usuario:
  - Declarar una variable de tipo EXCEPTION
  - La excepción se provoca con RAISE <nombre_excepción>

Ej: Emitir un mensaje si hay más de 5 jugadores almacenados

```
DECLARE
   E_MUCHOS_JUGADORES EXCEPTION;
   NumJugadores NUMBER;
BEGIN
   SELECT COUNT(*)       INTO NumJugadores
                FROM JUGADOR;
   IF NumJugadores>5 THEN
                RAISE E_MUCHOS_JUGADORES;
   END IF;
EXCEPTION
   WHEN E_MUCHOS_JUGADORES THEN
                DBMS_OUTPUT.PUT_LINE('Hay demasiados jugadores');
END;
```

```
BEGIN
   RAISE_APPLICATION_ERROR(-20000, 'Excepción creada por el usuario');
EXCEPTION
   WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('Código: ' || SQLCODE);
                DBMS_OUTPUT.PUT_LINE('Mensaje: ' || SUBSTR(SQLERRM, 11,100));
END;
```

Declarar procedimientos y funciones

- Declarar y usar un procedimiento

```
PROCEDURE procedimiento
    [(parámetro  [IN | OUT | IN OUT]  tipo  [{DEFAULT | :=} expresion]      [,...])]

    IS
            declaraciones ...
    BEGIN
            sentencias ...
    END [procedimiento];
```

Declarar procedimientos y funciones

- Declarar y usar un procedimiento

```
DECLARE
   PROCEDURE printLine(width IN INTEGER, chr IN CHAR DEFAULT '-')
   IS
   BEGIN
            FOR i IN 1 .. width LOOP
                     DBMS_OUTPUT.PUT(chr);
            END LOOP;
            DBMS_OUTPUT.PUT_LINE(‘ ');
   END printLine;

BEGIN
   printLine(40, '*');                          - imprime una línea con 40 asteriscos
   printLine(width=>>20, chr=>>'=');            - imprime una línea con 20 signos igual
   printLine(10);                               - imprime una línea con 10 guiones
END;
```

notación posicional

notación por nombre

Declarar procedimientos y funciones

- Declarar y usar una función

```
FUNCTION función
   [(parámetro    [IN | OUT | IN OUT]    tipo    [{DEFAULT | :=} expresion]      [,...])]
         RETURN tipo
   IS
            declaraciones ...
   BEGIN
            sentencias ...
   END [función];
```

Declarar procedimientos y funciones

- Declarar y usar una función

```
DECLARE
        tempPresup NUMBER;
        FUNCTION sumaPresup (codigo IN INTEGER)
                RETURN NUMBER
        IS
                sumaPresup NUMBER;
        BEGIN
                SELECT SUM(presupuesto) INTO sumaPresup
                        FROM EQUIPO
                        WHERE cp = codigo;
                RETURN sumaPresup;
        END sumaPresup;
BEGIN
   tempPresup := sumaPresup(15000);
   DBMS_OUTPUT.PUT_LINE('Presupuesto total Coruña‘ || tempPresup);
   DBMS_OUTPUT.PUT_LINE('Presupuesto total Madrid‘ || sumaPresup(28000));
END;
```

Procedimientos y funciones almacenados

- Crear y usar un procedimiento almacenado

```
CREATE OR REPLACE PROCEDURE printLine
                        (width IN INTEGER, chr IN CHAR DEFAULT '-')
  IS
  BEGIN
          FOR i IN 1 .. width LOOP
                  DBMS_OUTPUT.PUT(chr);
          END LOOP;
          DBMS_OUTPUT.PUT_LINE(‘ ');
  END printLine;
```

```
BEGIN
   printLine(40, '*');                          - imprime una línea con 40 asteriscos
END;
/
BEGIN
   printLine(width=>>20, chr=>>'=');            - imprime una línea con 20 signos igual
END;
/
BEGIN
   printLine(10);                               - imprime una línea con 10 guiones
END;
/
```

Procedimientos y funciones almacenados

- Crear y usar funciones almacenadas

```
CREATE OR REPLACE FUNCTION sumaPresup (codigo IN INTEGER)
    RETURN NUMBER
IS
    sumaPresup NUMBER;
BEGIN
    SELECT SUM(presupuesto) INTO sumaPresup
        FROM EQUIPO
        WHERE cp = codigo;
    RETURN sumaPresup;
END sumaPresup;
```

```
SELECT *
    FROM CIUDAD
    WHERE sumaPresup(cp) > 3000000;
```

Paquetes PL/SQL

- Estructura PL/SQL para almacenar de forma conjunta varios objetos PL/SQL relacionados
- Tiene dos partes: especificación “cabecera” y cuerpo
- Deben ser almacenados en el gestor (no pueden ser locales)
- Permite el uso de variables globales por sesión
- Cada sesión dispone de un área de memoria privada para el uso del paquete.
- Pueden disponer de un código de inicialización.

Declaración de Paquetes

- Declaración de la *cabecera del paquete*
  - Información acerca del contenido del paquete
    - Procedimientos y Funciones
    - Excepciones, Cursores, Tipos y Variables
    - Pragmas
  - No contiene código
- Declaración del *cuerpo del paquete*
  - Requiere de una compilación previa de la cabecera
  - Puede incluir declaraciones adicionales
    - Son globales respecto al cuerpo del paquete pero no son visibles
  - Puede no ser necesario si no hay procedimientos o funciones que implementar

Declaración de la cabecera del paquete

- Declaración de la cabecera del paquete

```
CREATE [OR REPLACE] PACKAGE paquete AS

    defínción_tipo |
    especificación_procedimientos |
    especificación_funciones |
    declaración_variables |
    declaración_excepciones |
    declaración_cursores |
    declaración_pragma

END [paquete];
```

Declaración de la cabecera del paquete

- Ej:

```
CREATE OR REPLACE PACKAGE equipos_pkg AS

    PROCEDURE printLine
        (width IN INTEGER, chr IN CHAR DEFAULT '-');
    FUNCTION sumaPresup
        (codigo IN INTEGER) RETURN NUMBER;

    E_MUCHOS_JUGADORES_EXCEPTION EXCEPTION;

    VARIABLE_GLOBAL NUMBER;

END equipos_pkg;
```

Declaración de cuerpo de paquete

- Declaración del cuerpo del paquete

```
CREATE [OR REPLACE] PACKAGE BODY paquete AS
        .....

BEGIN
        <código de inicialización>
END [paquete];
```

Declaración de cuerpo de paquete

- Ej:

```
CREATE OR REPLACE PACKAGE BODY equipos_pkg AS

   FUNCTION sumaPresup (codigo IN INTEGER)
      RETURN NUMBER
   IS
      sumaPresup NUMBER;
   BEGIN
      SELECT SUM(presupuesto) INTO sumaPresup
            FROM EQUIPO
            WHERE cp = codigo;
      RETURN sumaPresup;
   END sumaPresup;

END equipos_pkg;
```

Otras operaciones con paquetes

- Recompilar un paquete:

```
ALTER PACKAGE [esquema.]paquete
        COMPILE PACKAGE;

ALTER PACKAGE [esquema.]paquete
                        COMPILE BODY;
```

- Eliminar un paquete:

```
DROP PACKAGE [esquema.]paquete;

DROP PACKAGE BODY [esquema.]paquete;
```