

UD 8-3. SOLUCIONES EJERCICIOS. PL SQL. CURSORES, TRIGGERS Y EXCEPCIONES

Base de Datos
CFGS DAW

Francisco Aldarias Raya
paco.aldarias@ceedcv.es

2019/2020

Fecha 13/04/20

Versión:200413.1529


Licencia




Reconocimiento - NoComercial - CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante

 Atención

 Interesante

código

Revisión:

ÍNDICE DE CONTENIDO

1. Ejercicio 1.....	3
2. Ejercicio 2.....	4
3. Ejercicio 3.....	5
4. Ejercicio 4.....	5
5. Ejercicio 5.....	6
6. Ejercicio 6.....	6
7. Ejercicio 7.....	7
8. Ejercicio 8.....	8

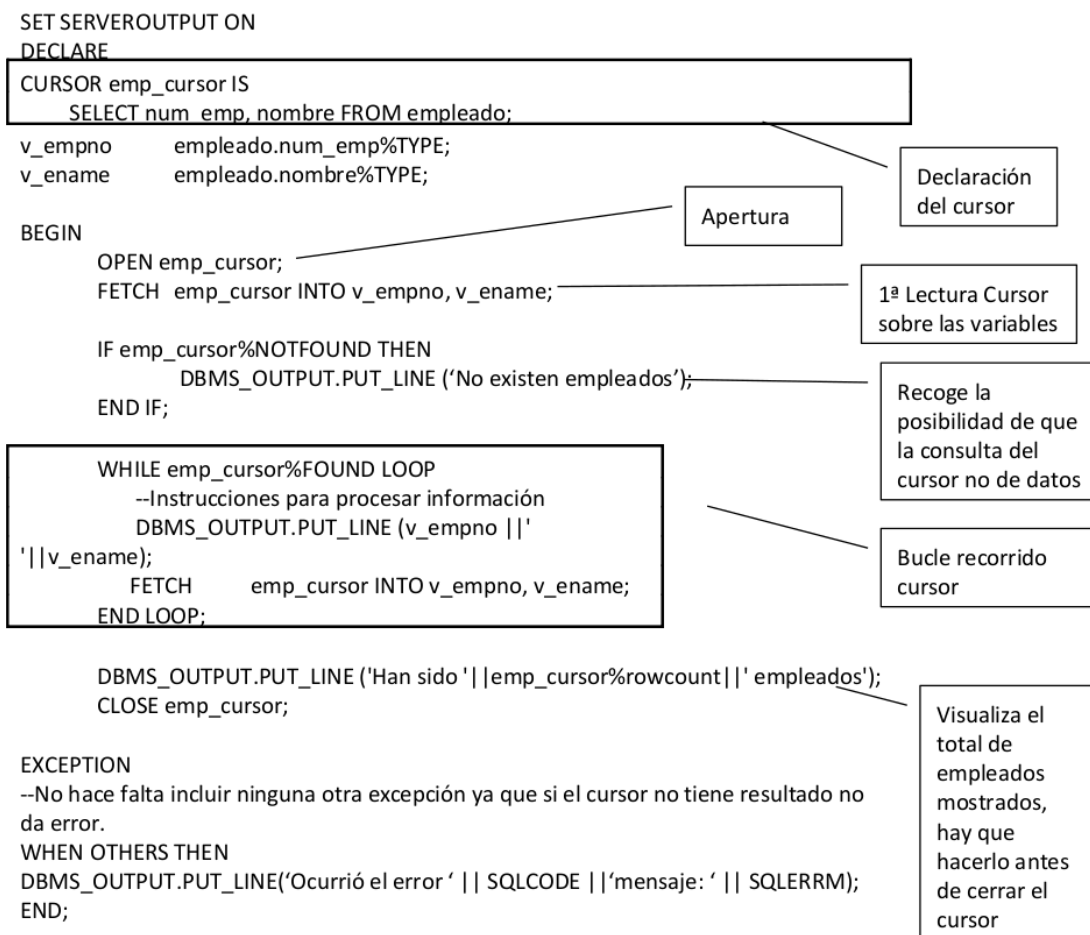
UD08-3. SOLUCIONES EJERCICIOS. PL SQL. CURSORES, TRIGGERS Y EXCEPCIONES

Para la resolución de estos ejercicios se recomienda consultar el Anexo de Funciones predefinidas así como el resto de funciones definidas en los manuales online.

Para todos los ejercicios se utilizará la bbdd de ejercicios de este tema BD_ej_PL_SQL.sql, excepto para el Ejercicio 8, donde utilizaremos la bbdd de jardineria_oracle.zip

1. EJERCICIO 1

Escribe un cursor que muestre todos los empleados de la tabla con un bucle WHILE.



2. EJERCICIO 2

Escribe un cursor que muestre el nombre y salario de los empleados del departamento 10 utilizando una variable registro.

```
SET SERVEROUTPUT ON
DECLARE
    CURSOR emp_cursor IS
    SELECT nombre, salario
    FROM empleado
    WHERE num_dpto = 10;
    v_reg_cursor emp_cursor%ROWTYPE;

BEGIN
    OPEN emp_cursor; --Abrir cursor
    FETCH emp_cursor INTO v_reg_cursor; -- Leer primera fila
    WHILE emp_cursor%FOUND LOOP -- mientras haya filas
        DBMS_OUTPUT.PUT_LINE(v_reg_cursor.nombre || ' gana ' || v_reg_cursor.salario
        ||
        '€');
        FETCH emp_cursor INTO v_reg_cursor; -- leer siguiente
    END LOOP;
    CLOSE emp_cursor; --cerrar cursor

EXCEPTION
--No hace falta incluir ninguna otra excepción ya que si el cursor no tiene resultado no
da error.
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Ocurrió el error ' || SQLCODE || 'mensaje: ' || SQLERRM);
END;
```

Variable registro.
Tienen la misma
estructura que
una fila del cursor.

Acceso a los datos de la
variable registro.
Nomvar.nomcampo

3. EJERCICIO 3

Escribe un cursor que muestre los años de antigüedad de los empleados del departamento 10.

(En esta solución se ha empleado la función MONTHS_BETWEENs pero podríais haber utilizado también otras como DATEDIFF(f1sf2)).

```
SET SERVEROUTPUT ON  
DECLARE
```

```
CURSOR emp_cursor IS  
SELECT nombre, salario, TRUNC(MONTHS_BETWEEN(SYSDATE, fecha_alta)/12) as  
antigüedad  
FROM empleado  
WHERE num_dpto = 10;
```

```
BEGIN
```

```
FOR v_reg_cursor IN emp_cursor LOOP  
    DBMS_OUTPUT.PUT_LINE(v_reg_cursor.nombre || ' * ' || v_reg_cursor.salario || ' *  
    ' || v_reg_cursor.antigüedad);  
END LOOP;
```

```
EXCEPTION
```

```
--No hace falta incluir ninguna otra excepción ya que si el cursor no tiene resultado no  
da error.
```

```
WHEN OTHERS THEN
```

```
DBMS_OUTPUT.PUT_LINE('Ocurrió el error ' || SQLCODE || 'mensaje: ' || SQLERRM);
```

```
END;
```

Bucle FOR, se encarga de abrir el cursor, leerlo en cada vuelta del bucle, declarar la variable registro para volcar el contenido de cada fila y cerrar el cursor

4. EJERCICIO 4

Supongamos que queremos hacer un trigger que no permita a un empleado ianar más de 5000€ si no es el presidente.

Para ello has de crear un trigger que antes de que se inserte o se actualice ún dato de la tabla empleado se compruebe si el salario es mayor de 5000 y si su tarea es diferente de presidente se imprima por pantalla (dado que aún no hemos visto excepciones) que 'No puede ianar tanto si no es el presidente'.

Hacer el trigger y comprobar que funciona correctamente.

```
CREATE OR REPLACE TRIGGER ControlSueldo
BEFORE INSERT OR UPDATE ON Empleado
FOR EACH ROW
BEGIN
    IF :new.salario>5000 AND :new.tarea!='PRESIDENTE' THEN
        --RAISE_APPLICATION_ERROR(-20100, 'No puede ganar tanto si no es presidente');
        DBMS_OUTPUT.PUT_LINE('No puede ganar tanto si no es presidente');
    END IF;
END;
/
```

5. EJERCICIO 5

Crea un disparador de fila para impedir que se modifique el nombres el num_emp o el salario si su nuevo salario es más de un 10% mayor que el anterior. En el caso que se vaya a modificar cualquiera de estos 3 campos y el nuevo salario sea mayor que el 10% del anterior se imprimirá por pantalla que eso no está permitido (dado que aún no hemos visto excepciones).

```
CREATE OR REPLACE TRIGGER fallo_modif
BEFORE UPDATE OF nombre, num_emp, salario
ON empleado
FOR EACH ROW
BEGIN
    IF UPDATING('num_emp') OR UPDATING('nombre')
    OR (UPDATING ('salario') AND :new.salario>:old.salario*1.1)
    THEN
        --RAISE_APPLICATION_ERROR(-20001,'Err. Modificacion no permitida');
        DBMS_OUTPUT.PUT_LINE('Modificacion no permitida');
    END IF;
END;
/
```

6. EJERCICIO 6

Crea un disparador de sentencia para seguir manteniendo información de los

empleados que dejan de trabajar en la empresas para ello necesitamos un disparador que almacene los empleados borrados en una tabla nueva.

Por tanto antes de crear el triggers creamos la nueva tabla como copia de empleados y borramos su contenido para dejarla vacía.

```
CREATE TABLE emple_borrados as (select * from empleado);  
TRUNCATE TABLE emple_borrados;
```

A continuación añadimos la columna fecha_baja que almacenara la fecha en la que se produce la baja.

```
ALTER TABLE EMPLE_BORRADOS ADD FECHA_BAJA DATE;
```

```
CREATE OR REPLACE TRIGGER grabar_borrados  
  BEFORE DELETE  
  ON EMPLEADO  
FOR EACH ROW  
  BEGIN  
    INSERT INTO EMPLE_BORRADOS  
    VALUES(  
      :OLD.num_emp,  
      :OLD.nombre,  
      :OLD.tarea,  
      :OLD.jefe,  
      :OLD.fecha_alta,  
      :OLD.salario,  
      :OLD.num_dpto,  
      sysdate  
    );  
END;  
/
```

7. EJERCICIO 7

Supongamos que queremos hacer un trigger que impida insertar datos en la tabla de departamentos (dpto) fuera del horario normal de oficina.

El horario normal de oficina es de lunes a viernes de 8 a 15:00h.

Dado que todavía no hemos visto las excepciones cuando se intente insertar un departamento fuera de dicho horario se deberá imprimir por pantalla la fecha del sistema en la que se produjo esa inserción no permitida.

```
CREATE OR REPLACE TRIGGER SeguridadEmp
  BEFORE INSERT ON dpto
BEGIN
  IF((TO_CHAR(sysdate, 'DY') IN ('SÁB', 'DOM')) OR
    (TO_CHAR(sysdate, 'HH24') NOT BETWEEN '08' AND '15')) THEN
    --RAISE_APPLICATION_ERROR( -20100, 'No puedes insertar registros fuera
    -- del horario normal de oficina');
    --Hace saltar una excepción de manera que impide la inserción. Más
    -- adelante veremos dónde tratamos la excepción.
    DBMS_OUTPUT.PUT_LINE('Inserción no permitida en la siguiente fecha '||
      to_char(sysdate));
  END IF;
END;
/
```

8. EJERCICIO 8

Para terminar vamos a realizar un ejercicio que aunque no está completo pues no incluye el tratamiento de las excepciones nos puede dar una idea de la potencia que aporta la programación con PL/SQL.

Se trata de realizar un procedimiento que reciba el código de un cliente y nos muestre su estado, es decir, sus datos de cliente los pedidos que ha realizado dicho cliente con el importe total los pagos que ha realizado y el importe que tener pendientes así como si ha superado o no el crédito que tiene indicado en su fecha de cliente.

Una vez creado el procedimiento lo podremos ejecutar en la línea de comandos con el comando EXECUTE seguido del nombre del procedimiento y los parámetros entre paréntesis.

El resultado deberá ser:


```
SQL> @ c:\src\ejer_pl_01.sql

Procedure created.

SQL> EXECUTE ESTADOCLIENTE<1>;
CODIGO CLIENTE:      1
NOMBRE CLIENTE:      DGPRODUCTIONS GARDEN
CONTACTO:            Daniel G
TELEFONO:             5556901745
FAX:                 5556901746
DIRECCION:           False Street 52 2 A
CIUDAD:              San Francisco
REGION:
PAIS:                USA
COD. POSTAL:         24006
-----
Pedido Num.:8 Importe: 1065
Pedido Num.:9 Importe: 2535
Pedido Num.:11 Importe: 820
Pedido Num.:12 Importe: 290
Pedido Num.:25 Importe: 1455
Total Facturado: 6165
-----
Pago 1 FECHA: 10/11/08 Cantidad: 2000
Pago 2 FECHA: 10/12/08 Cantidad: 2000
Total Pagado: 4000
-----
Su saldo es: -2165
NO ha superado su CREDITO que es de 3000 Euros

PL/SQL procedure successfully completed.
```

(Pista: Crear dos cursores (Pedidos_Cliente y Pagos_Cliente) y utilizar FOR de cursor para procesar los resultados)

```
CREATE OR REPLACE PROCEDURE EstadoCliente (Cod IN Clientes.CodigoCliente%TYPE)
IS
    Regcli Clientes%ROWTYPE;

    CURSOR Pedidos_Cliente IS
        SELECT D.CodigoPedido, SUM(D.cantidad * D.PrecioUnidad) as Total_Pedido
        FROM DETALLEPEDIDOS D, PEDIDOS P
        WHERE P.CODIGOCLIENTE = COD AND D.CODIGOPEDIDO=P.CODIGOPEDIDO
        GROUP BY D.CodigoPedido
        ORDER BY D.CodigoPedido;

    CURSOR Pagos_Cliente IS
        SELECT FechaPago, Cantidad
        FROM PAGOS
        WHERE CODIGOCLIENTE = COD
        ORDER BY FechaPago;

    TotalFacturado NUMBER(10,2) DEFAULT 0;
    TotalPagado NUMBER(10,2) DEFAULT 0;
    Saldo NUMBER(10,2) DEFAULT 0;

BEGIN
    /*Seleccionamos el cliente que recibimos como parámetro de entrada*/

    SELECT * INTO RegCli FROM Clientes
    WHERE CODIGOCLIENTE = Cod;
    DBMS_OUTPUT.PUT_LINE('CODIGO CLIENTE: '||RegCli.CodigoCliente);
    DBMS_OUTPUT.PUT_LINE('NOMBRE CLIENTE: '||RegCli.NombreCliente);
    DBMS_OUTPUT.PUT_LINE('CONTACTO: '||RegCli.NombreContacto);
    DBMS_OUTPUT.PUT_LINE('TELEFONO: '||RegCli.Telefono);
    DBMS_OUTPUT.PUT_LINE('FAX: '||RegCli.Fax);
    DBMS_OUTPUT.PUT_LINE('DIRECCION: '||RegCli.LineaDireccion1);
    DBMS_OUTPUT.PUT_LINE('CIUDAD: '||RegCli.Ciudad);
    DBMS_OUTPUT.PUT_LINE('REGION: '||RegCli.Region);
    DBMS_OUTPUT.PUT_LINE('PAIS: '||RegCli.Pais);
    DBMS_OUTPUT.PUT_LINE('COD. POSTAL: '||RegCli.CodigoPostal);
```

```
DBMS_OUTPUT.PUT_LINE('-----');
/*Gestionamos los pedidos del cliente mediante un FOR de cursor*/
FOR RegPed IN Pedidos_Cliente LOOP
    DBMS_OUTPUT.PUT_LINE('Pedido Num.: ' || RegPed.CodigoPedido || ' Importe: ' ||
        RegPed.Total_Pedido);
    TotalFacturado:=TotalFacturado + RegPed.Total_Pedido;
END LOOP;

DBMS_OUTPUT.PUT_LINE('Total Facturado: ' || TotalFacturado);
DBMS_OUTPUT.PUT_LINE('-----');
/*Gestionamos los pagos del cliente mediante un FOR de cursor*/

FOR RegPago IN Pagos_Cliente LOOP
    DBMS_OUTPUT.PUT_LINE('Pago ' || Pagos_Cliente%ROWCOUNT || ' FECHA: ' ||
        RegPago.FechaPago || ' Cantidad: ' || RegPago.Cantidad);
    TotalPagado:=TotalPagado + RegPago.Cantidad;
END LOOP;

DBMS_OUTPUT.PUT_LINE('Total Pagado: ' || TotalPagado);
DBMS_OUTPUT.PUT_LINE('-----');
/*Calculamos el saldo*/
Saldo:=TotalPagado-TotalFacturado;
DBMS_OUTPUT.PUT_LINE('Su saldo es: ' || Saldo);

IF Saldo < 0 THEN
    Saldo := Saldo * -1;
    IF Saldo > RegCli.LimiteCredito THEN
        DBMS_OUTPUT.PUT_LINE('Ha superado su CREDITO en: ' || Saldo
            RegCli.LimiteCredito);
    ELSE
        DBMS_OUTPUT.PUT_LINE('NO ha superado su CREDITO que es de ' ||
            RegCli.LimiteCredito || ' Euros');
    END IF;
END IF;
END;
/
```