-

TEMA 8.
PL/SQL. CURSORES Y DISPARADORES

Francisco Aldarias Raya

paco.aldarias@ceedcv.es

Base de Datos
CFGS DAW

2019/2020

Fecha 05/05/20

Versión:200505.1301

Licencia

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.CURSORES....4
1.1 Cursores implícitos....4
1.2 Cursores explícitos....4
1.3 Atributos....6
1.3.1 Ejemplo 1....6
1.4 Registros....7
1.4.1 Ejemplo 2....8
1.4.2 Ejemplo 3....8
1.5 Bucle FOR de cursor....9
1.5.1 Ejemplo 4....9
2.TRIGGERS....10
2.1 Sintaxis....10
2.2 Orden de ejecución de los Triggers....11
2.3 Restricciones en la utilización de Triggers....11
2.4 Creando Triggers I....11
2.4.1 Ejemplo 5....13
2.5 Creando Triggers II....13
2.6 Gestionando los triggers....14
3.EXCEPCIONES....15
3.1 Ejemplo 6....16

TEMA 8. PL/SQL. CURSORES Y DISPARADORES

1. CURSORES

> **Importante**
>
> Los cursores son áreas de memoria que almacenan datos extraídos de la base de datos.

Hay dos tipos los cursores implícitos y los explícitos.

1.1 Cursores implícitos

> **Importante**
>
> Los cursores implícitos sirven para todas aquellas consultas que devuelven únicamente una fila.

No es necesario declararlos pues se crean de forma implícita (son los que hemos utilizado hasta ahora).

Repetimos, solamente los podemos utilizar en instrucciones SELECT donde se devuelva una única filas si no devuelve ninguna o devuelve más de una se producirá un error. Estos errores se controlarán en la zona EXCEPTION mediante las excepciones NO_DATA_FOUND y TOO_MANY_ROWS.

1.2 Cursores explícitos

> **Importante**
>
> Los cursores explícitos deben ser declarados por el programador y se emplearán en todas las consultas que puedan devolver más de una fila en el resultado.

La sintaxis para la declaración de un cursor se colocará dentro del apartado DECLARE de la siguiente forma:

```
CURSOR nombre_cursor IS sentencia_select_del_cursor;
```

> Atención
>
> La sentencia SELECT del cursor explícito NO lleva INTO para pasar el resultado a las variables del procedimiento o función.

Por ejemplo:

```
DECLARE CURSOR herramientas IS
        SELECT Nombre, PrecioVenta
        FROM PRODUCTOS
        WHERE GAMA='Herramientas';
```

La declaración del cursor aún no ha realizado ninguna operación para que se ejecute la consulta se tiene que abrir el cursor. En este casos a diferencia de los cursores implícitos si no se devuelve ninguna fila no se producirá una excepción.

Para abrir el cursor bastará con ejecutar una instrucción OPEN seguida del nombre del cursor de esta manera:

```
OPEN herramientas;
```

Una vez abierto el cursor, procederemos a recorrer las filas que se han obtenido como resultado de la consulta que tiene asociada. Para ello emplearemos la instrucción FETCH.

La sintaxis será:

```
FETCH nombre_cursor INTO [var1,var2,var3,... ] | registro;
```

Cada vez que hacemos un FETCH recibiremos una fila y se guardarán los datos en las variables o en el registro que indiquemos. La asignación de valores se realizará posicionalmente la primera columna a la primera variables la segunda columna a la segunda variables etc. Por ello debe haber el mismo número de variables que el número de columnas que se devuelve en cada fila.

Es labor del programador comprobar si la consulta ha devuelto filas o si el procesamiento de las filas ya ha concluido.
Por último, una vez procesadas las filas que nos interesen debemos cerrar el cursor. Para ello emplearemos la instrucción CLOSE seguida del nombre del cursor de la siguiente forma:

```
CLOSE herramientas;
```

Por supuesto, una vez cerrados, no se podrán recuperar filas del cursor a no ser que volvamos a abrirlo. También hay que tener en cuenta que los gestores de bases de datos suelen tener un parámetro (v$parameter) para indicar cuántos cursores pueden estar abiertos de forma simultánea.

Si en algún momento tenemos cursores que a veces funcionan y otras veces nos puede ser que dicho parámetro deba ser incrementado. Os si por ejemplos aparece un error como SQL ORA-01000 (número máximo de cursores abiertos excedido).
Para conocer el número máximo de cursores podemos hacer:

```
select value from v$parameter where name='open_cursors';
```

Y para actualizar el número máximo de cursores as por ejemplos 1000:

```
alter system set open_cursors = 1000;
```

1.3 Atributos

Los cursores nos proporcionan un conjunto de atributos que nos permitirán controlar el funcionamiento de los cursores. Los atributos que vamos a utilizar son:

- %ISOPEN es de tipo booleano y nos permite comprobar si un cursor está abierto. El atributo tendrá el valor TRUE si el cursor está abierto y FALSE en caso contrario.
- %NOTFOUND también de tipo booleano y nos permite comprobar si la última recuperación devuelve fila (FALSE) o no (TRUE).
- %FOUNDs igualmente de tipo booleanos devolverá TRUE si la recuperación más reciente devuelve fila y FALSE si no lo hace.
- %ROWCOUNT número de filas devueltas hasta ese momento.

Veamos cómo utilizarlos con un ejemplo (pl23.sql)

1.3.1 Ejemplo 1

```
SET SERVEROUTPUT ON
DECLARE CURSOR herramientas IS
        SELECT Nombre, PrecioVenta
        FROM PRODUCTOS
        WHERE GAMA='Herramientas';
mi_nombre varchar2(70);
mi_precioventa number(10,2);
BEGIN
        OPEN herramientas;
        IF herramientas%ISOPEN THEN
                FETCH herramientas INTO mi_nombre, mi_precioventa;
                WHILE herramientas%FOUND LOOP
                        DBMS_OUTPUT.PUT_LINE('Fila: '||herramientas
                        %ROWCOUNT||'-'||mi_nombre||'-'||mi_precioventa);
                        FETCH herramientas INTO mi_nombre, mi_precioventa;
                END LOOP;
                CLOSE herramientas;
        END IF;
END;
/
```

Hemos declarado el cursor y dos variables que van a recoger los resultados de las filas que devuelve. Después abrimos el cursor y comprobamos que se ha abierto correctamente. Si está abierto solicitamos la primera fila con un FETCH y si la ha encontrado entramos en el WHILE (si no hubiese filas en el resultado no entraría en el bucle). Una vez dentro del bucle mostramos los datos recogidos y pedimos la siguiente fila con otro FETCH. Realizaremos el bucle hasta que se hayan procesado todas las filas del cursor. Posteriormente cerramos el cursor y terminará la ejecución del script.

El resultado será:

1.4 Registros

Los registros son elementos bastante utilizados en PL/SQL.

> **Importante**
>
> Un registro es un grupo de variables relacionadas bajo un mismo nombres cada una de las cuales tiene su propio nombre y su tipo de dato.

La forma de declarar un registro es la siguiente:

```
TYPE direccion IS RECORD
(calle varchar2(30),
numero int,
localidad varchar2(30),
codpostal varchar2(5));
```

Ya tenemos declarado el tipo de registros ahora ya podemos declarar variables de dicho tipo.

mi_direccion direccion;

Donde mi_direccion es la variable y dirección es el tipo de dato definido como registro.
Para poder acceder a los campos del registro que hemos creado bastará con utilizar el operador punto de la siguiente forma:

```
mi_direccion.calle
mi_direccion.numero
mi_direccion.localidad
mi_direccion.codpostal
```

Podremos asignar valores a los campos:

```
mi_direccion.calle := 'Avda. la Consttución';
mi_direccion.numero := 20;
```

```
mi_direccion.localidad := 'Madrid';
mi_direccion.codpostal := '28003';
```

Aunque la forma más sencilla de crear un registro es utilizando los atributos de tipo (%ROWTYPE) creando un registro con los mismos campos que la fila que devolverá un cursor de la siguiente forma:

```
DECLARE CURSOR pagos_clientes IS
        SELECT codigocliente, sum(cantdad) as total_pagado
        FROM pagos group by codigocliente
        ORDER BY codigocliente;
reg_pagos pagos_clientes%ROWTYPE;
```

Veamos cómo utilizarlos con un script (pl24.sql)

1.4.1 Ejemplo 2

```
SET SERVEROUTPUT ON
DECLARE CURSOR pagos_clientes IS
        SELECT codigocliente, sum(cantidad) as total_pagado
        FROM pagos group by codigocliente
        ORDER BY codigocliente;
reg_pagos pagos_clientes%ROWTYPE;
BEGIN
        OPEN pagos_clientes;
        IF pagos_clientes%ISOPEN THEN
                FETCH pagos_clientes INTO reg_pagos;
                WHILE pagos_clientes%FOUND LOOP
                        DBMS_OUTPUT.PUT_LINE('Fila: '||pagos_clientes
                        %ROWCOUNT||'- Cliente:'||reg_pagos.codigocliente||'- Total:'||
                        reg_pagos.total_pagado);
                        FETCH pagos_clientes INTO reg_pagos;
                END LOOP;
CLOSE pagos_clientes;
END IF;
END;
/
```

El resultado será:

1.4.2 Ejemplo 3

Los registros también los podemos utilizar con cursores implícitos. El uso más habitual es cuando les asignamos el atributo de tipo de una tabla. Veamos un ejemplo sencillo (pl24a.sql)

```
SET SERVEROUTPUT ON
DECLARE Reg_cliente Clientes%ROWTYPE;
BEGIN
        SELECT * INTO Reg_cliente
        FROM clientes WHERE codigocliente=6;
        DBMS_OUTPUT.PUT_LINE(Reg_cliente.nombrecliente||' - '||Reg_cliente.telefono);
END;
/
```

El resultado será:

1.5 Bucle FOR de cursor

El bucle FOR funciona de una forma especial con los cursores explícitos. Al crear una instrucción FOR con un cursor el sistema realiza de forma implícita la apertura del cursor (OPEN) la recuperación de las filas una a una (FETCH) y el cierre del cursor (CLOSE). Además la variable índice del FOR ahora actuará como un registro con el atributo de tipo de la fila recuperada en el cursor. Es decir lo hace prácticamente todo de forma automáticas únicamente debemos procesar las líneas.

La sintaxis será:

```
FOR nombre_registro IN nombre_cursor LOOP
instrucción_1;
instrucción_2;
....
END LOOP;
```

Veamos un ejemplo (pl25.sql)

1.5.1 Ejemplo 4

```
SET SERVEROUTPUT ON
DECLARE
Total_Pedidos NUMBER(10,2):=0;
Reg_p10p detallepedidos%ROWTYPE;
CURSOR primeros_10_pedidos IS
        SELECT Codigopedido, sum(cantdad * preciounidad) as total
        FROM detallepedidos
        WHERE codigopedido BETWEEN 1 and 10
        GROUP BY codigopedido
        ORDER BY codigopedido;
BEGIN
FOR Reg_p10p IN primeros_10_pedidos LOOP
DBMS_OUTPUT.PUT_LINE('Pedido: '||Reg_p10p.codigopedido ||' Total: '||
Reg_p10p.total);
Total_pedidos := Total_pedidos + Reg_p10p.total;
```

```
END LOOP;
DBMS_OUTPUT.PUT_LINE('El total de los pedidos es: '||Total_Pedidos);
END;
/
```

El resultado será:

2. TRIGGERS

> **Importante**
>
> Los disparadores (o triggers) son bloques de código asociados a una tabla que se ejecutan automáticamente como reacción a una operación específica (INSERT,UPDATE o DELETE) sobre dicha tabla.

Los triggers pueden ejecutarse antes o después (BEFORE, AFTER) de que se produzca la operación indicada sobre la tabla.

2.1 Sintaxis

La sintaxis de los triggers en Oracle es:

```
CREATE {OR REPLACE} TRIGGER nombre_disparador [BEFORE|AFTER]
[DELETE|INSERT|UPDATE {OF columnas}]
[OR [DELETE|INSERT|UPDATE {OF columnas}]...]
ON tabla
[FOR EACH ROW [WHEN condicion disparo]]
[DECLARE]
        -- Declaración de variables locales
BEGIN
        -- Instrucciones de ejecución
[EXCEPTION]
        -- Instrucciones de excepción
END;
```

Como podemos deducir de la sintaxis se puede crear un disparador o reemplazar uno que ya exista. Debemos elegir si se debe ejecutar antes o después de que se realice la operación a la que está asociado con las palabras BEFORE o AFTER.
También debemos indicar la operación sobre la que se va a aplicar INSERT, DELETE o UPDATE. Hay que indicar la tabla sobre la que se realiza la operación: ON tabla.
Además debemos indicar si es un disparador a nivel de fila con FOR EACH ROWs eso quiere decir que el disparador se activará una vez por cada fila afectada en la operación que provoca el disparo.
También existen los Triggers a nivel de orden que se activan sólo una vez después de la ejecución completa de la instrucción.

2.2 Orden de ejecución de los Triggers

En una misma tabla podemos tiene asociados varios Triggers, si esto ocurre, debemos conocer el orden en el que se ejecutarán.
Los disparadores se activarán al comenzar la instrucción SQL y se ejecutarán en el siguiente orden:

- Primero se ejecutará el disparador a nivel de orden con el tipo BEFORE.
- Después para cada filas se ejecutará el disparador de tipo BEFORE con nivel de fila (row).
- Lo siguiente será ejecutar la instrucción SQL con la fila correspondiente.
- Después se ejecutará el disparador de tipo AFTER con nivel de fila (row).
- Por último se ejecutará el disparador a nivel de orden con el tipo AFTER.

2.3 Restricciones en la utilización de Triggers

El cuerpo del disparador, es un bloque PL/SQL, luego cualquier instrucción de las que conocemos hasta ahora la podemos emplear en un Trigger. Solamente debemos tener en cuenta algunas restricciones:

- Un disparador no puede incluir control de transacciones ni puede llamar a funciones o procedimientos que puedan incluir un control de este tipo.
- Además aunque hay alguna excepción como regla general nunca modificaremos la tabla que se está viendo afectada por la instrucción SQL.

2.4 Creando Triggers I

Para comenzar vamos a crear un Trigger que nos permita controlar el número de empleados que tiene un departamento. Para ello vamos a modificar la tabla DEPARTAMENTOS añadiendo el campo NumEmp con valor 0 por defecto:

Después vamos a actualizar los registros de los departamentos incluyendo el número de empleados que hay de cada uno de ellos:

Quedando al final los datos:

Ahora vamos a crear un Trigger de tal forma que cada vez que se realice un INSERT en la tabla empleados se aumente en uno el número de empleados del departamento al que pertenece. Fíjate que para hacer esto necesitaremos acceder a los campos que se insertan pues necesitamos conocer el código del departamento del nuevo empleado. Cuando trabajamos a nivel de fila (row) para acceder al contenido de los campos Oracle proporciona las referencias NEW y OLD para acceder a los nuevos datos (NEW.Nombre_Campo) o a los antiguos (OLD.Nombre_Campo) respectivamente. Debemos tiener en cuenta que:

- En un INSERT solo tenemos datos NEW y si nuestro Trigger es de tipo BEFORE podemos modificar esos valores antes de que se almacenen en la tabla.
- En un UPDATE tenemos datos NEW y OLDs los primeros son los nuevos datos y los segundos son los existentes en la tabla. Igual que en el caso anterior si es de tipo BEFORE también se podrán modificar los datos de NEW.
- En un DELETE solo tenemos datos tipo OLD.

Cuando vamos a utilizar estos campos dentro del cuerpo del Trigger será necesario colocar delante de ellos el signo dos puntos (:), excepto en la cláusula WHEN que veremos después.

Bien vamos con nuestro ejemplos se trata de aumentar de forma automática con un Trigger el campo NUMEMP del departamentos cuando se inserta un nuevo empleado. Nuestro código será:

```
CREATE OR REPLACE TRIGGER nuevo_empleado
AFTER INSERT
ON EMPLEADOS
FOR EACH ROW
BEGIN
UPDATE DEPARTAMENTOS SET NUMEMP = NUMEMP + 1
WHERE CODDPTO = :NEW.DPTO;
END;
/
```

Prestad atención al modo en el que se emplea el código de departamento del nuevo empleado con :NEW.DPTO

Si al crear el Trigger hubiera errores podríamos verlos con el comando:

```
Show Errors;
```

Una vez creado el script lo ejecutamos y se creará el Trigger.

Para comprobar su funcionamiento tenemos que insertar un nuevo registro en empleados y comprobar sis de forma automáticas se aumenta el campo NUMEMP del departamento al que pertenece el nuevo empleado.

Como podemos observar se ha aumentado en 1 el NUMEMP del departamento de Informática al que pertenece el nuevo empleado.
¿Cómo sería un trigger que al eliminar un empleado actualizase correctamente el número de empleados del departamento al que pertenece? Crea el Trigger indicado y probarlo. Si no te sales, compártelo con tus compañeros en el foro.

2.4.1 Ejemplo 5

Ahora vamos a crear otro Trigger que al actualizar un empleado si se cambia el departamentos actualice de forma correcta el campo NUMEMP de los departamentos involucrados:

```
CREATE OR REPLACE TRIGGER cambio_en_empleado
AFTER UPDATE
ON EMPLEADOS
FOR EACH ROW
BEGIN
        IF :NEW.DPTO != :OLD.DPTO THEN
                UPDATE DEPARTAMENTOS SET NUMEMP = NUMEMP - 1
                WHERE CODDPTO = :OLD.DPTO;
                UPDATE DEPARTAMENTOS SET NUMEMP = NUMEMP + 1
                WHERE CODDPTO = :NEW.DPTO;
        END IF;
END;
/
```

El resultado será:

2.5 Creando Triggers II

En la creación de Triggers podemos añadir una cláusula WHEN en el procesamiento de cada fila (solo se puede utilizar con triggers a nivel de fila) de tal forma que se evalúe una condición que implique si se debe ejecutar o no el código del Trigger. Su sintaxis sería:

```
FOR EACH ROW WHEN condición
```

Recordemos que si queremos emplear los prefjos NEW y OLD en la condición del WHEN no debemos colocar el signo dos puntos (:) delante del prefijo. Por ejemplo:

```
FOR EACH ROW WHEN (NEW.Especialidad <>'Contabilidad')

BEGIN

INSTRUCCIONES .... ;
```

```
END;
```

Dentro del Trigger podemos detectar si se trata de una instrucción SQL de inserción (INSERTING), modificación (UPDATING) o borrado (DELETING).

```
IF INSERTING THEN
INSTRUCCIONES ....;
END IF;
```

```
IF UPDATING THEN
INSTRUCCIONES ....;
END IF;
```

Además si estamos ante una actualización a nivel de fila podemos comprobar si se está actualizando un determinado campo:

```
IF UPDATING('ESPECIALIDAD') THEN
INSTRUCCIONES ... ;
END IF;
```

2.6 Gestionando los triggers

Para ver los triggers que hay en la base de datos podemos emplear la vista ALL_TRIGGERS que contiene los siguientes campos:

Podemos ver cuáles son los disparadores de nuestro usuario con:

También podemos activar o desactivar un trigger con:

```
ALTER TRIGGER nombre_disparador {ENABLE | DISABLE}
```

Por ejemplo:

Por últimos podemos eliminar un Trigger con:

```
DROP TRIGGER nombre_disparador;
```

3. EXCEPCIONES

> Importante
>
> Una excepción es un identificador PL/SQL que aparece cuando se produce un error durante la ejecución.

Cuando se produce un error se detiene la ejecución normal del programa y se lanza una excepción transfiriendo el control del programa al manejador de excepciones (sección EXCEPTION del procedimiento o función). Eso significa que una vez que ha ocurrido el error ya no podremos continuar la ejecución por el lugar donde estábamos antes de que ocurriera.

Las excepciones más comunes son las que produce la base de datos, aunque también el programador puede lanzar una excepción cuando detecta una situación anómala mediante la instrucción RAISE. Para ello debemos declarar la nueva excepción en el bloque DECLARE con la siguiente sintaxis:

```
Nombre_nueva_excepción_usuario EXCEPTION;
```

Para gestionar las excepciones Oracle proporciona su propia instrucción EXCEPTION que es situada al final del bloque de código. Si en un bloque se produce una excepción y no es tratada en él, la excepción se propaga al proceso padre.

Para manejar las excepciones emplearemos la siguiente sintaxis:

```
EXCEPTION
WHEN excepción [OR excepton2…] THEN
        instruccion1;
        instruccion2;
[……..
WHEN OTHERS THEN
instruccion3;
instruccion4;]
```

La palabra reservada EXCEPTION, inicia la sección de gestión de excepciones.

En el bloque podemos manejar varios tipos de excepciones, pero sólo se procesará un manejador (el primero por el que entre) antes de salir del bloque.

La cláusula WHEN OTHERS es la última cláusula y se entrará en ella si no se ha entrado por ninguna de las anteriores.

~ Oracle proporciona dos funciones predefinidas para trabajar con las excepciones: SQLCODE nos devuelve el número de error que se ha producido y SQLERRM nos devuelve el mensaje de error asociado a ese número de error. Los números de error son negatvos excepto el de NO_DATA_FOUND que es el 100.

3.1 Ejemplo 6

Veamos un ejemplo en el que vamos a provocar un error de división por cero para que se produzca una excepción:

```
SET SERVEROUTPUT ON
DECLARE
        NUMERO_ERROR NUMBER;
        MENSAJE_ERROR VARCHAR2(255);
        res NUMBER;
BEGIN
        SELECT n/0 INTO res FROM DUAL;
EXCEPTION
        WHEN OTHERS THEN
                NUMERO_ERROR := SQLCODE;
                MENSAJE_ERROR := SQLERRM;
                DBMS_OUTPUT.put_line('Error: '||NUMERO_ERROR);
                DBMS_OUTPUT.put_line('Mensaje: '||MENSAJE_ERROR);
END;
/
```

El resultado será:

Oracle tiene predefinidas la mayoría de las excepciones, aquí te presentamos algunas de ellas:

Vamos a modificar el ejemplo anterior para añadir la excepción de división por cero:

```
SET SERVEROUTPUT ON
DECLARE
        NUMERO_ERROR NUMBER;
        MENSAJE_ERROR VARCHAR2(255);
        res NUMBER;
BEGIN
        SELECT n/0 INTO res FROM DUAL;
EXCEPTION
        WHEN ZERO_DIVIDE THEN
        DBMS_OUTPUT.put_line('Hola, tenes un Error. ESTAS DIVIDIENDO X 0.');
WHEN OTHERS THEN
        NUMERO_ERROR := SQLCODE;
        MENSAJE_ERROR := SQLERRM;
        DBMS_OUTPUT.put_line('Error: '||NUMERO_ERROR);
        DBMS_OUTPUT.put_line('Mensaje: '||MENSAJE_ERROR);
END;
/
```

El resultado será:

Pueden ocurrir otras muchas excepciones que no están en la lista anterior. En ese caso, aunque no tienen un nombre asignado, sí tenen un número asignado. Ese número es el que aparece cuando Oracle muestra el mensaje de error tras la palabra ORA.

Supongamos que tenemos nuestra base de datos inicial con departamentos, empleados y proyectos.

Recuerda que en Oracle no tenemos el ON UPDATE CASCADE, luego si intentamos actualizar el código del departamento debe mostrar un error de integridad referencial. Veamos qué error muestra y cómo capturar ese error para manejarlo.

Si intentamos modificar desde la línea de comandos veremos el error que se produce:

Podemos ver que se devuelve un error con el código -2292 indicando que no se cumplen las condiciones de la integridad referencial. Ese código no es ninguno de los que hemos colocado en la tabla anterior.

Para capturarlo haremos lo siguiente:

```
SET SERVEROUTPUT ON
DECLARE
BEGIN
```

```
        UPDATE DEPARTAMENTOS SET CODDPTO = 'INF'
        WHERE CODDPTO ='IT';
EXCEPTION
        WHEN OTHERS THEN
        IF SQLCODE = -2292 THEN
                DBMS_OUTPUT.PUT_LINE('ERROR DE INTEGRIDAD. HAY EMPLEADOS
        ASIGNADOS');
ELSE
        DBMS_OUTPUT.PUT_LINE(SQLCODE ||' - '||SQLERRM);
END IF;
END;
/
```

El resultado será:

Podemos encontrar los errores más frecuentes en la página web:

http://ora.u440.com/errores/.