## Índice

## 1. DECLARACIÓN DE UN CURSOR

Para poder procesar una orden SQL, Oracle asigna un área de memoria que recibe el nombre de ***área de contexto***. Esta área contiene informaciones necesarias para completar el procesamiento, incluyendo el número de filas procesadas por la orden, un puntero a la versión analizada de la orden y, en el caso de las consultas, el ***conjunto activo***, que es el conjunto de filas resultado de la consulta.

Un *cursor* es un puntero al área de contexto. Mediante el cursor, un programa PL/SQL puede controlar el área de contexto y lo que en ella sucede a medida que se procesa la orden.

La declaración de un cursor define su nombre, y asocia el cursor con una orden SELECT. La sintaxis es:

```
CURSOR nombre_cursor IS orden_SELECT;
```

Hay que tener en cuenta que el nombre del cursor es un identificador PL/SQL, y ha de ser declarado antes de poder hacer referencia a él. Podemos usar cualquier orden SELECT incluyendo uniones (joins) y órdenes con las cláusulas UNION o MINUS.

Por ejemplo:

```
DECLARE
   CURSOR Mi_Cursor
IS
   SELECT num_emp, nombre, puesto, salario
   FROM empleados
   WHERE num_dept = 'informatica'
```

Este comando es meramente declarativo, simplemente especifica las filas y columnas que se van a recuperar. La consulta se ejecuta cuando se abre o se activa el cursor. La cláusula [ORDER BY] es opcional y especifica una ordenación para las filas del cursor; si no se especifica, la ordenación de las filas es definida el gestor de SGBD

## 2. APERTURA DEL CURSOR

La sintaxis para abrir un cursor es:

```
OPEN nombre_cursor;
```

Al abrir un cursor suceden tres cosas:

1.- Se examinan los valores de las variables aceptadas

2.- Se determina el conjunto activo, basándose en los valores de dichas variables

3.- Se hace apuntar el puntero del conjunto activo a la primera fila

El conjunto activo, es decir, el conjunto de filas que satisfacen los criterios de la consulta se determina en el momento de abrir un cursor. La cláusula WHERE se evalúa para la tabla o tablas referenciadas en la cláusula FROM de la consulta, y todas las filas para las cuales se evalúe la condición como TRUE son añadidas al conjunto activo. También se establece un puntero al conjunto activo en el momento de abrir el cursor. **Este puntero indica cuál es la siguiente fila que el cursor extraerá.**

**El puntero se posiciona delante de la primera fila de datos (registro actual), esta sentencia no recupera ninguna fila.**

Es legal abrir un cursor que ya esté abierto. PL/SQL ejecutará implícitamente una orden CLOSE antes de reabrir el cursor con la segunda orden OPEN. También puede haber más de un cursor abierto al mismo tiempo.

## 3. EXTRACCIÓN DE LOS DATOS DEL CURSOR (LEER)

La cláusula INTO de la consulta es parte de la orden FETCH. Dicha orden tiene dos formas posibles:

```
FETCH nombre_cursor INTO lista_variables;

FETCH nombre_cursor INTO registro_PL/SQL;
```

*Nombre_cursor.-* Identifica a un cursor abierto y ya declarado

*Lista_variables.-* Es una lista de variables PL/SQL previamente declaradas y separadas por comas

*Registro_PL/SQL.-* Es un registro PL/SQL previamente declarado

En ambos casos la variable o variables de la cláusula INTO deben ser compatibles en cuanto a tipo con la lista de selección de la consulta. Después de cada FETCH, se incrementa el puntero activo, para que apunte a la siguiente fila. De esta forma, cada FETCH devolverá filas sucesivas del conjunto activo, hasta que se devuelve el conjunto completo.

## 4. CIERRE DEL CURSOR

Cuando se ha terminado de extraer el conjunto activo, debe cerrarse el cursor. Esta acción informa a PL/SQL de que el programa ha terminada de utilizar el cursor, y de que se pueden liberar los recursos con él asociados. Estos recursos incluyen las áreas de almacenamiento empleadas para contener el conjunto activo, así como cualquier espacio temporal utilizado en la determinación de dicho conjunto. La sintaxis para el cierre del cursor es:

```
CLOSE nombre_cursor;
```

Si intentamos cerrar un cursor que ya está cerrado nos daría un error ORA-1001.

## 5. ATRIBUTOS: NO_DATA_FOUND, %NOTFOUND %ISOPEN

La excepción NO_DATA_FOUND se produce sólo en las órdenes SELECT..INTO, cuando la cláusula WHERE de la consulta no se corresponde con ninguna fila. En el caso de los cursores explícitos, por el contrario, cuando la cláusula WHERE no se corresponde con ninguna fila, lo que sucede es que el atributo **%NOTFOUND** toma el valor TRUE. Si la cláusula WHERE de una orden UPDATE o DELETE no se corresponde con ninguna fila, el atributo SQL %NOTFOUND toma el valor TRUE, en lugar de producirse la excepción NO_DATA_FOUND.

Debido a esta circunstancia DEBE UTILIZARSE EN todos los bucles de extracción **%NOTFOUND o %FOUND** para determinar la condición de salida del bucle, en lugar de la excepción NO_DATA_FOUND.

**%ISOPEN** Este atributo toma el valor verdadero (TRUE) cuando un cursor se encuentra **abierto**. De otra manera, retorna FALSO.

***%ROWCOUNT*** Cuando un cursor se abre este atributo se inicializar en 0 (cero). En adelante, cada vez que se recuperen filas exitosamente con un FETCH, este valor se irá incrementando en uno. Cuando se utiliza con cursores implícitos, este **atributo devuelve el total de filas afectadas por una instrucción del tipo INSERT, UPDATE o DELETE.**

**Ejemplo 1.- Abrir un cursor y recorrerlo**

```
DECLARE
CURSOR Employee_Cursor IS
        SELECT Last_Name, First_Name
        FROM EMPLOYEE
        WHERE Last_Name like 'B%';
  Apellido employee.last_name%type;
  Nombre employee.first_name%type;
  begin
   OPEN Employee_Cursor;

fetch  Employee_Cursor  into  apellido, nombre;
        while Employee_Cursor%found loop

 fetch  Employee_Cursor  into  apellido, nombre;
   end  loop;
CLOSE Employee_Cursor;
end;
/
```

**Ejemplo 2.- utilización del %FOUND %ROWCOUNT**

```
begin
 update employee
   set commission=commission*1.10
 where department_id in(select department_id
 from department where name='SALES');
  if SQL%FOUND then
       dbms_output.put_line('se incremente 10% a'||to_char(SQL%ROWCOUNT));
  end if;
 end;
/
```

**Ejemplo 3.- Abrir un cursor e imprimir su contenido**

```
DECLARE
        CURSOR Employee_Cursor IS
        SELECT Last_Name, First_Name
        FROM EMPLOYEE
        WHERE Last_Name like 'B%';
        Apellido employee.last_name%type;
         Nombre employee.first_name%type;
 begin
        OPEN Employee_Cursor;
        fetch  Employee_Cursor  into  apellido, nombre;
        while Employee_Cursor%found loop
                dbms_output.put_line(apellido||' '||nombre);
                fetch  Employee_Cursor  into  apellido, nombre;
        end  loop;
        CLOSE Employee_Cursor;
end;
/
```

**Ejemplo 3.1.- Cursores y Records. Abrir un cursor e imprimir su contenido utilizando variable tipo record.**

```
DECLARE
        CURSOR Employee_Cursor IS
        SELECT Last_Name, First_Name
        FROM EMPLOYEE
        WHERE Last_Name like 'B%';
        var_cur Employee_cursor%rowtype;
 begin
        OPEN Employee_Cursor;
        fetch  Employee_Cursor  into  var_cur;
        while Employee_Cursor%found loop
                dbms_output.put_line(var_cur.last_name||' '||var_cur.first_name);
                fetch  Employee_Cursor  into  var_cur;
        end  loop;
        CLOSE Employee_Cursor;
end;
/
```

## 6. BUCLES FOR PARA CURSORES

Hay una manera de manejar cursores sin tener que realizar los pasos de declaración, apertura, mover y cierre de un cursor. Esto se consigue mediante la utilización de bucles FOR. Ejemplo:

```
DECLARE
        CURSOR employee_cursor IS
        SELECT Last_Name, First_Name
        FROM EMPLOYEE
        WHERE Last_Name like 'B%';

 begin
        --se realiza el open implícito--
        for VAR in employee_cursor loop
                --el fetch lo hace de forma implícita--
                dbms_output.put_line(var.last_name||'  '||var.first_name);
                --ahora chequea si hay datos de forma implícita--

          end  loop;
        --cierra de forma implícita el cursor--

end;
/
```

**No se ha declarado la variable VAR, pero el propio compilador la declara al analizar el bucle FOR. Esta variable es de tipo cursor, es decir tiene todos los campos del cursor. Para **visualizar un campo** hemos de poner el nombre de la **variable.nombre del campo**
Es una forma más sencilla y clara para el uso de los cursores.

## 7. SELECT… FOR UPDATE (CURSORES)

Si hay varias sesiones en una Base de Datos, existe la posibilidad de que las filas de una tabla en particular se actualizan después de haber abierto un cursor. De modo que usted ve los datos actualizados sólo cuando vuelve a abrir el cursor. Por lo tanto, es mejor bloquear las filas antes de actualizarlas o eliminarlas. Es posible bloquear las filas con la cláusula ***FOR UPDATE*** en la consulta del cursor.

La cláusula FOR UPDATE es parte de una orden SELECT. Es la última cláusula de la orden, después de la cláusula ORDER BY (si es que existe), su sintaxis es:

```
SELECT… FROM… FOR UPDATE [OF referencia_columna] [NOWAIT]
```

La sentencia ***SELECT ... FOR UPDATE*** identifica las filas que se van a actualizar o eliminar y, a continuación, bloquea cada fila. Esto es útil cuando se desea realizar una actualización en los valores existentes en una fila. En ese caso, se debe asegurar de que la fila no sea cambiada por otra sesión antes de la actualización.

Ejemplo

```
DECLARE
   CURSOR c_emp_cursor IS
   SELECT employee_id, last_name, salary
   FROM employee
      WHERE department_id = 20
      FOR UPDATE OF


        WHERE CURRENT OF cursor
```

- Usar cursores para actualizar o eliminar la fila actual.
- Incluir la cláusula ***FOR UPDATE*** en la consulta de cursor para primero bloquear la fila.
- Utilice la cláusula ***WHERE CURRENT OF*** para hacer referencia a la fila actual de un cursor explícito.

La cláusula ***WHERE CURRENT OF*** se utiliza en conjunto con la cláusula ***FOR UPDATE*** para referirse a la fila actual de un cursor explícito. La cláusula ***WHERE CURRENT OF*** se utiliza en las sentencias ***UPDATE*** o ***DELETE***; primero es necesario especificar la cláusula ***FOR UPDATE*** en la declaración del cursor. Se puede utilizar la combinación de actualización y supresión de la fila actual de la tabla de Base de Datos correspondiente. Esto le permite aplicar las actualizaciones y eliminar la fila en cuestión, sin la necesidad de hacer referencia explícita al *ID* de la misma.

***Sintaxis***:

```
UPDATE table_name
SET column_name = ...
WHERE CURRENT OF cursor_name;
```

```
DELETE FROM table_name
WHERE CURRENT OF cursor_name;
```

Ejemplo completo

Actualizamos el salario de los empleados cuyos apellidos empiezan por B un 10%, y vamos visualizando las actualizaciones

```
DECLARE
        CURSOR c_emp IS
        SELECT  salary
        FROM  employee
        WHERE  last_name like 'B%';
        salario employee.salary%type;
BEGIN
        OPEN c_emp;
        FETCH c_emp INTO  salario;
        while c_emp%found loop
                UPDATE employee
                SET salary = salario*1.1;
                dbms_output.put_line('anterior: '||salario);
                FETCH c_emp INTO  salario;
        end loop;
        CLOSE c_emp;
END;
/
```

Comprobamos que está mal porque como no hemos bloqueado ninguna fila todas se actualizan al mismo valor
Asi si hacemos

```
Select salary
From employee;
```

Obtenemos la siguiente salida

| | SALARY |
|---|---|
| 1 | 3135 |
| 2 | 3135 |
| 3 | 3135 |
| 4 | 3135 |
| 5 | 3135 |
| 6 | 3135 |
| 7 | 3135 |
| 8 | 3135 |
| 9 | 3135 |
| 10 | 3135 |
| 11 | 3135 |
| 12 | 3135 |

En la que **todos los salarios son iguales**. Sin embargo, si hacemos este otro cursor

```
DECLARE
   CURSOR c_emp IS
   SELECT  salary
   FROM  employee
   WHERE  last_name like 'B%'
     FOR UPDATE;

   salario employee.salary%type;
BEGIN
   OPEN c_emp;
   FETCH c_emp INTO  salario;
   while c_emp%found loop
      UPDATE employee
     SET salary = salario*1.1
     WHERE CURRENT OF c_emp;
        dbms_output.put_line('anterior: '||salario);

     FETCH c_emp INTO  salario;
 end loop;
   CLOSE c_emp;
END;
/
```

Si ejecutamos la sentencia

```
SELECT salary
 FROM employee
 WHERE last_name like 'B%';
```

Obtenemos como salida

| | SALARY |
|---|---|
| 1 | 2420 |
| 2 | 3135 |

Que es a los dos que ha cambiado

Y si hicieramos

```
SELECT salary
FROM employee;
```

Comprobamos ahora que no todos son iguales, sino que cada uno tiene su salario salvo los dos cambios.

| | SALARY |
|---|---|
| 1 | 600 |
| 2 | 1600 |
| 3 | 2650 |
| 4 | 2750 |
| 5 | 2420 |
| 6 | 1250 |
| 7 | 1250 |
| 8 | 1250 |
| 9 | 1250 |
| 10 | 1250 |
| 11 | 2975 |
| 12 | 3000 |
| 13 | 1250 |
| 14 | 1600 |
| 15 | 1250 |
| 16 | 1650 |
| 17 | 3135 |
| 18 | 2450 |
| 19 | 3000 |
| 20 | 1500 |

Si la consulta del cursor hace referencia a múltiples tablas, se deberá usar FOR UPDATE OF nombreColumna, con lo que únicamente se bloquearán las filas correspondientes de la tabla que tenga la columna especificada.

**CURSOR nombreCursor IS SELECT ... FROM… FOR UPDATE** [**OF** *nombreColumna*]

Actualizamos el salario de los empleados cuyos apellidos empiezan por B un 10%, y vamos visualizando las actualizaciones

```
DECLARE
        CURSOR c_emp IS
        SELECT  salary,name
        FROM  employee,department
        WHERE  employee.department_id=department.department_id and last_name like 'B%'
        FOR UPDATE;
         salario employee.salary%type;
        nombre department.name%type;

BEGIN
        OPEN c_emp;
        FETCH c_emp INTO  salario, nombre;
        while c_emp%found loop
                dbms_output.put_line('anterior: '||salario||'  '||nombre);
                UPDATE employee
                SET salary = salario*1.1
                WHERE CURRENT OF c_emp;
                dbms_output.put_line('actualizado');
                FETCH c_emp INTO  salario, nombre;
        end loop;
        CLOSE c_emp;
END;
/
```

Comprobamos que si hacemos la consulta

```
SELECT  salary,name
FROM  employee,department
WHERE  employee.department_id=department.department_id and last_name like 'B%'
```

| | SALARY | NAME |
|---|---|---|
| 1 | 2200 | OPERATIONS |
| 2 | 2850 | SALES |

No nos ha actualizado el campo salary. Sin embargo, en el siguiente caso

```
DECLARE
        CURSOR c_emp IS
        SELECT  salary,name
        FROM  employee,department
        WHERE  employee.department_id=department.department_id and last_name like 'B%'
        FOR UPDATE of salary;
        salario employee.salary%type;
        nombre department.name%type;

BEGIN
        OPEN c_emp;
        FETCH c_emp INTO  salario, nombre;
        while c_emp%found loop
                dbms_output.put_line('anterior: '||salario||'  '||nombre);
                UPDATE employee
                SET salary = salario*1.1
                WHERE CURRENT OF c_emp;
                dbms_output.put_line('actualizado');
                FETCH c_emp INTO  salario, nombre;
        end loop;
        CLOSE c_emp;
END;
/
```

Si hacemos ahora

```
SELECT  salary,name
FROM  employee,department
WHERE  employee.department_id=department.department_id and last_name like 'B%';
```

| | SALARY | NAME |
|---|---|---|
| 1 | 2420 | OPERATIONS |
| 2 | 3135 | SALES |

Vemos que se ha hecho el update.

En el ejemplo anterior, si quisiéramos crear un cursor para acceder a campos de dos tablas y modificar algún campo tendremos que utilizar FOR UPDATE OF nombreCampo.
En el ejemplo siguiente se trata de modificar el salario y el código de función:

```
DECLARE
        CURSOR c_emp IS
        SELECT  employee_id, job_id, salary,name
        FROM  employee ,department
        WHERE  employee.department_id=department.department_id and last_name like 'B%'
        FOR UPDATE of salary,job_id;
        cod_empleado employee.employee_id%type;
        cod_funcion employee.job_id%type;
        salario employee.salary%type;
        nombre_dep department.name%type;

BEGIN
        OPEN c_emp;
        FETCH c_emp INTO  cod_empleado, cod_funcion, salario, nombre_dep;
        while c_emp%found loop
                dbms_output.put_line('anterior: '||cod_empleado||' '||salario||' '||cod_funcion);
                UPDATE employee
                SET salary = salario*1.1, job_id=667
                WHERE CURRENT OF c_emp;
                dbms_output.put_line('actualizado');
                FETCH c_emp INTO  cod_empleado, cod_funcion, salario, nombre_dep;
        end loop;
        CLOSE c_emp;
END;
/
```

Si antes de ejecutar el cursor realizamos

```
SELECT  employee_id, job_id, salary,name
FROM  employee ,department
WHERE  employee.department_id=department.department_id and last_name like 'B%';
```

Obtenemos la siguiente salida

| | EMPLOYEE_ID | JOB_ID | SALARY | NAME |
|---|---|---|---|---|
| 1 | 7507 | 671 | 2200 | OPERATIONS |
| 2 | 7698 | 671 | 2850 | SALES |

Una vez ejecutado el cursor volvemos a ejecutar la misma select

```
SELECT  employee_id, job_id, salary,name
FROM  employee ,department
WHERE  employee.department_id=department.department_id and last_name like 'B%';
```

Obtenemos la siguiente salida

| | EMPLOYEE_ID | JOB_ID | SALARY | NAME |
|---|---|---|---|---|
| 1 | 7507 | 667 | 2420 | OPERATIONS |
| 2 | 7698 | 667 | 3135 | SALES |

Con lo que podemos observar que se han actualizado el salario y el job_id.

## 8. EJERCICIOS SOBRE CURSORES, BASADOS EN LAS TABLAS

TABLA: **EMPLOYEE**

| Name | Null? | Type |
|---|---|---|
| EMPLOYEE_ID | NOT NULL | NUMBER(4) |
| LAST_NAME | | VARCHAR2(15) |
| FIRST_NAME | | VARCHAR2(15) |
| MIDDLE_INITIAL | | VARCHAR2(1) |
| JOB_ID | | NUMBER(3) |
| MANAGER_ID | | NUMBER(4) |
| HIRE_DATE | | DATE |
| SALARY | | NUMBER(7,2) |
| COMMISSION | | NUMBER(7,2) |
| DEPARTMENT_ID | | NUMBER(2) |

TABLA: **DEPTARTMENT**

| Name | Null? | Type |
|---|---|---|
| DEPARTMENT_ID | NOT NULL | NUMBER(2) |
| NAME | | VARCHAR2(14) |
| LOCATION_ID | | NUMBER(3) |

**Ejercicio número: 1**

Realizar un programa que calcule el presupuesto del departamento para el año próximo, esto quedará almacenado en la tabla **DEPARTMENT** en la columna **PRESUP**. Hay que tener en cuenta las siguientes subidas de sueldo:

671 + 20%
669 + 15%

Los demás empleados que no estén en ninguna de las categorías anteriores se les subirá el sueldo un 10%.

LO PRIMERO CON ALTER TABLE CREAMOS EN LA TABLA DEPARTMENT EL CAMPO PRESUP

```
Alter table department
Add (presup number(10,2));
```

```
declare

        cursor  c1  is  select  salary,  job_id,  department_id
        from  employee;
        salario         employee.salary % type;
        trabajo         employee.job_id % type;
        cod_dep         employee.department_id % type;
begin
        update  department
        set  presup = 0;    -- el nuevo campo se inicializa a 0
        open  c1;
        fetch  c1  into  salario,  trabajo,  cod_dep;
        while c1%found loop
                if  trabajo = 671
                        then
                                salario := salario * 1.2;
                                        elsif  trabajo  =  669
                                                then
                                                        salario  :=  salario * 1.15;

                                                else
                                                        salario  :=  salario * 1.1 ;
                end if;
                update  department
                set  presup  =  presup  +  salario
                where  department_id = cod_dep;
                fetch  c1  into  salario,  trabajo,  cod_dep;
        end loop;
        close  c1;
end;
/
```

**Ejercicio número: 2**

Programa que actualice el campo TOTAL_SALARIO y el campo MEDIA_SALARIOS de la tabla **DEPTARTMENT**, siendo el total la suma del salario de todos los empleados, igualmente con la media.

1.- Crear un cursor **C1**, que devuelva todos los departamentos
2.- Crear un cursor **C2**, que devuelva el salario y el código de todos los empleados de su departamento

LO PRIMERO CON ALTER TABLE CREAMOS EN LA TABLA DEPARTMENT LOS CAMPOS STOTAL Y MTOTAL

```
Alter table department
Add (stotal number(10,2), mtotal number(10,2));
```

```
declare
        cursor  c1   is
        select distinct department_id
        from  employee;

cursor  c2  (departamento   department.department_id % type )  is
        select    salary
        from  employee
        where  department_id  =  departamento;

departamento  department.department_id % type;
salario       employee.salary % type;
ssal          number (10);
msal          employee.salary % type;

        begin

                open  c1;
                fetch  c1  into  departamento;
                while c1%found loop
                   msal  :=  0;
                   ssal  :=  0;
                   open  c2(departamento);
                   fetch  c2  into  salario;
                   while c2%found loop
                         ssal  :=  ssal  +  salario;
                         fetch  c2  into  salario;
                   end loop;
                   if  ssal  <>  0
                   then
                      msal  :=  ssal  /  c2 %rowcount;
                   end if;

                   update  department
                   set  stotal  =  ssal,  mtotal  =  msal
                   where department_id=departamento;
                  close  c2;
                   fetch  c1  into  departamento;
                 end loop;
        close  c1;
        end;
 /
```

## 9.- EJERCICIOS PROPUESTOS

1º.- Realizar un bloque pl/sql que permita visualizar el apellido y la fecha de alta de todos los empleados de la tabla employee ordenados por apellido.

2º.- Realizar un bloque pl/sql que permita visualizar el nombre de cada departamento y el número de empleados que tiene.

3º.- Realizar un bloque pl/sql que reciba una cadena por teclado y visualice el apellido y el número de empleado de todos los empleados cuyo apellido coincida con la cadena especificada. Al finalizar visualizar el número de empleados mostrados.

4º.- Realizar un bloque pl/sql que visualice el apellido y el salario de los cinco empleados que tienen el salario más alto.

5º.- Codifica un bloque pl/sql que visualice los dos empleados que ganan menos en cada tipo de trabajo.

6º.- Escribir un bloque pl/sql que muestre en formato similar a las rupturas de control los siguientes datos:

Para cada empleado: apellido y salario.

Para cada departamento: número de empleados y suma de salarios del departamento.

Al final del listado: número total de empleados y suma de todos los salarios.

7º.- Codificar un bloque pl/sql que permita actualizar la comisión de los empleados con arreglo a las siguientes premisas:

A los empleados cuyo salario esté entre 0 y 1000 la comisión será de 200.
A los empleados cuyo salario esté entre 1001 y 2500 la comisión será 300.
A los empleados cuyo salario sea mayor que 2500 la comisión será 400.

8º.- Realizar un bloque pl/sql que permita actualizar el límite de crédito de los clientes con los siguientes criterios:

Entre 0 y 1 pedidos limite de crédito = 0.
Entre 2 y 5 pedidos limite de crédito= 15% gastos.
6 o más pedidos limite de crédito = 20% gastos.

9º.- Realizar un bloque Pl/Sql que permita borrar todos los clientes de la tabla customer que no tengan dependencias y visualice por pantalla el código del cliente y el nombre de los clientes borrados.

10º.- Realizar un bloque Pl/Sql que visualice el código y el nombre de los clientes, así como el código del pedido y su valor de todos los pedidos que ha realizado cada cliente con un valor total inferior a 600 euros.