CURSORES

## Introducción:

* Definimos un cursor como un área de memoria que se reserva para ejecutar sentencias SQL y para almacenar información de procesamiento de las mismas
* Con los cursores se puede acceder al área de trabajo o contexto para procesar la información recuperada en una consulta
* Existen dos tipos de cursores:

Explícitos.- Son definidos por el programador para acceder a las filas recuperadas en una consulta.

Implícitos.- Generados automáticamente por Oracle al ejecutar una instrucción no asociada a un cursor por el programador. Es decir, cuando se lanza una sentencia SQL tiene un cursor asociado

* Para una mejor comprensión, podríamos decir que utilizar cursores explícitos es como trabajar con ficheros virtuales en memoria, donde el fichero sería el cursor y cada una de las filas recuperadas en la consulta sería un registro.
* Cuando lanzamos una sentencia SELECT esta debe devolver una única fila , de lo contrario obtendremos la excepción TOO\_MANY\_ROWS. Pero en muchas ocasiones existe la necesidad de trabajar con más de una fila, para ello utilizamos los cursores explícitos

## Declaración:

* Para declarar un cursor utilizaremos el siguiente formato:

# DECLARE

tipo :=

CURSOR nom\_cur [(param1 { } [{ }valor], ...)]

var %TYPE DEFAULT

IS instrucción\_select;

nom\_cur.- Nombre del cursor a declarar.

param1.- Variable usada como parámetro en la instrucción SELECT. Puede aparecer más de un parámetro. Se debe especificar su tipo de forma obligatoria, siendo opcional la posibilidad de darle un valor por defecto.

Instrucción\_select.- Instrucción SELECT que especifica las distintas condiciones y procedencias que deben cumplir las filas recuperadas para el cursor. En caso de que el

cursor incorpore algún parámetro, éste aparecerá en cualquier parte de dicha instrucción.

Ejemplo:

CURSOR c\_vecinos IS SELECT \* FROM vecinos; CURSOR c\_libros IS

SELECT nom\_autor, titulo, editorial FROM libros;

CURSOR c\_facturas IS

SELECT \* FROM facturas WHERE unidades > 70;

CURSOR c\_alumnos (nota notas.ev1 %TYPE) IS SELECT nom\_alu, ape1\_alu, ape2\_alu, ev1

FROM alumnos a, notas n

WHERE a.num\_alu = n.num\_alu AND ev1>=nota;

* Una vez declarado el cursor, para poder acceder a sus datos debemos abrirlo. El formato es:

# BEGIN

OPEN nom\_cur [(valor1[, ...])]; END;

nom\_cur.- Nombre del cursor a abrir.

valor1.- Valor que tomará el parámetro. Es de obligada especificación únicamente en el caso de que, al declarar el cursor, se especificará al menos un parámetro y no se le hubiera asignado valor por defecto.

Ejemplo: OPEN c\_vecinos;

OPEN c\_libros; OPEN c\_facturas; OPEN c\_alumnos(7);

* OPEN asigna de forma dinámica una area de memoria para que sea utilizada por el cursor
* Obtiene lo valores de la sentencia select
* Posiciona el puntero del cursor en la primera fila de las filas recuperadas
* Una vez abierto el cursor, podemos acceder a las filas en él almacenadas, para ello utilizaremos la instrucción FECHT, cuyo formato es:

# BEGIN

.....

FETCH nom\_cur INTO {variable1[, variable2 ...]| registro};

# .... END;

/

* La sentencia FECHT lee la fila marcada por el puntero, poniendo el valor de sus columnas en las variables que siguen a la palabra clave INTO.
* La lectura de una fila mediante FECHT hace que el puntero pase a señalar la fila siguiente, con lo que el acceso se realizará secuencialmente en el orden en que se recuperaron las filas.
* Debe existir una correspondencia en cuanto a número y tipo entre las columnas de la consulta y las variables de FECHT.
* A continuación ponemos un ejemplo de un cursor al que hay que pasarle un parámetro. No debemos olvidar que para abrir un cursor con valores diferentes para los parámetros, debemos cerrarlo antes de abrirlo si ya ha sido utilizado con anterioridad.

Ejemplo:

CREATE OR REPLACE PROCEDURE ver\_emple\_apell(cadena VARCHAR2) AS

cad varchar2(70);

CURSOR c\_emple (cad varchar2)IS SELECT ename, empno FROM emp WHERE ename LIKE cad;

vr\_emple c\_emple%ROWTYPE;

/\*\*arriba es un rowtype del mismo tipo cursor\*/

BEGIN

cad:=upper(cadena); cad:='%'||cad||'%';

OPEN c\_emple(cad);

FETCH c\_emple INTO vr\_emple; WHILE (c\_emple%FOUND) LOOP

DBMS\_OUTPUT.PUT\_LINE(vr\_emple.empno||'\*'||vr\_emple.ename); FETCH c\_emple INTO vr\_emple;

### END LOOP;

DBMS\_OUTPUT.PUT\_LINE('NUMERO DE EMPLEADOS: '|| c\_emple

### %ROWCOUNT);

### /\*\*Rowcount te de el numero de fila \*/

CLOSE c\_emple; END ver\_emple\_apell;

/

Ejemplo:

### DECLARE

CURSOR c\_notas IS SELECT \* FROM notas; num NUMBER(2);

nota1 NUMBER(4,2); nota2 NUMBER(4,2);

### BEGIN

OPEN c\_notas;

FETCH c\_notas INTO num, nota1, nota2;

### ... END;

/

* Una vez finalizada el acceso a las filas el cursor debe cerrarse, para ello utilizaremos la instrucción CLOSE , cuyo formato es:

CLOSE nom\_cur;

nom\_cur.- Nombre del cursor a cerrar. Ejemplo: CLOSE c\_notas;

* Cada cursor tiene asociados cuatro atributos predefinidos por el sistema. Cuando se trabaja con el cursor, estos atributos devuelven información sobre el resultado de la sentencia ejecutada sobre el cursor. Los atributos son:

#### %ISOPEN

Retorna TRUE si el cursor está abierto y FALSE si está cerrado.

#### %FOUND

Devuelve FALSE si el último FECHT ejecutado no encontró más filas y en caso de encontrar una fila devuelve TRUE. Después de abrir el cursor y antes del primer FECHT retorna NULL.

##### EJEMPLO BEGIN

OPEN cur1;

FETCH cur1 INTO v\_nom,v\_loc; WHILE cur1%FOUND LOOP

Dbms\_output.put\_line(v\_nom|| v\_loc); FETCH cur1 INTO v\_nom,v\_loc;

END LOOP; CLOSE cur1; END;

#### %NOTFOUND

Devuelve FALSE si FECHT encontró una fila y TRUE en caso de no encontrarla. Es exactamente lo contrario del atributo anterior. Después de abrir el cursor y antes del primer FECHT retorna NULL.

##### EJEMPLO:

….

EXIT WHEN CUR1%notfound; -- Se suele utilizar como salida de los bucles

#### %ROWCOUNT

Devuelve el número de filas accedidas por FECHT hasta el momento.

* Para utilizar estos atributos nombre\_cursor%vble
* El atributo %ISOPEN es realmente útil con cursores explícitos, ya que el control de apertura y cierre de los cursores explícitos lo tiene el programador, mientras que los cursores implícitos los controla el servidor de Oracle

## BUCLE FOR PARA TRATAR CURSORES MEDIANTE REGISTROS.

* Cuando se trabaja con cursores y registros, existe una versión del bucle FOR específica para el tratamiento de la información. Su formato es:

FOR nom\_registro IN nom\_cursor [(valor1[,...])] LOOP sentencias;

##### END LOOP;

nom\_registro.- Nombre del registro que generará automáticamente el sistema para el control del cursor asociado.

nom\_cursor.- Nombre del cursor que el bucle controlará su tratamiento. valor1,....- Valores de los parámetros del cursor.

* La sentencia FOR se encarga de abrir el cursor y recupera la primera fila en su ejecución. En sucesivas ejecuciones, va recuperando cada una de las filas del cursor. Cuando ya no hay más filas, se cierra automáticamente el cursor y se termina la ejecución del bucle.
* Los valores de las filas del cursor son accesibles a través de los campos del registro, el cual, lo declara el sistema automáticamente y es del tipo nom\_cursor %ROWTYPE.
* Al registro no se le puede hacer referencia fuera del bucle FOR.

Ejemplo:

### DECLARE

CURSOR c\_notas (nota notas.ev1 %TYPE) IS

SELECT num\_alu, ev1 FROM notas WHERE ev1 >= nota

### FOR UPDATE;

BEGIN

FOR var\_notas IN c\_notas(7) LOOP

IF var\_notas.ev1>7 AND var\_notas.ev1<10 THEN var\_notas.ev1 := var\_notas.ev1 + 1;

UPDATE notas SET ev1 = var\_notas.ev1 WHERE CURRENT OF c\_notas;

### END IF; END LOOP; COMMIT;

END;

/ El bucle FOR se encarga generar el cursor, de abrirlo, posicionar el puntero en todas y cada una de las filas del cursor y de cerrarlo después de procesar la última fila.

## Bucles For para Cursores

* Procesan filas de un cursor explícito.
* Cuando usamos un bucke For de estas características no tenemos que preocuparnos de abrir,cerrar y realizar la instrucción fetch.
* Lo único que hacemos es declararlo

FOR id\_reg IN id\_cursor LOOP Sentencias;

##### END LOOP;

Id\_reg: es una variable de tipo registro donde se almacenan los datos abiertos por el cursor. No hace falta declararlo y su ambito es unicamente el bucle

Id\_cursor: Es el identificador del cursor que previamente hemos declarado

* El bucle se termina automáticamente cuando se recupera la última fila

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | %FOUND | %ISOPEN | %NOTFOUND | %ROWCOUNT |
| open | Antes | Invalid\_cursor | F | Invalid\_cursor | Invalid |
| Después | NULL | T | NULL | 0 |
| Primer fetch | Antes | NULL | T | NULL | 0 |
| Después | T | T | F | 1 |
| Ultimo fetch | Antes | T | T | F | N |
| Después | F | T | T | N |
| close | Antes | T | T | T | N |
| Después | Invalid\_cursor | F | invalid | invalid |

### EJEMPLO:

DECLARE

CURSOR MICUR IS SELECT \* FROM scott.salgrade WHERE hisal>=2000; BEGIN

### FOR REG\_EMP IN MICUR LOOP

DBMS\_OUTPUT.PUT\_LINE(to\_char(micur%rowcount,’99.’)||REG\_EMP.GRADE||':'|| reg\_emp.grade);

end loop; end;

/

## Atributos en cursores implícitos

* Se abre implícitamente un cursor cuando se procesa un comando SQL que no está asociado a un cursor explícito
* El cursor implícito se llama SQL y dispone de los 4 atributos vistos anteriormente

SQL%NOTFOUND: TRUE si el último INSERT,UPDATE,DELETE o SELECT han fallado o no hay filas

SQL%FOUND: TRUE si el último INSERT,UPDATE,DELETE o SELECT han afectado a una o mas filas

SQL%ROWCOUNT devuelve el número de filas afectadas en el último INSERT,UPDATE,DELETE o SELECT

SQL%ISOPEN siempre devuelve falso ya que oracle cierra automáticamente el cursor después de hacer la sentencia SQL asociada

## Diferencias del cursor implicito con un cursor explicito

* + En el caso de SELECT INTO debe devolver una y solo una fila de lo contrario se producen errores y se levantan automáticamente las excepciones:

NO\_DATA\_FOUND: Ninguna fila TOO\_MANY\_ROWS: Más de una fila

Se detiene la ejecución del programa y se bifurca a la sección de EXCEPTION

* + Esto no es aplicable a INSERT,UPDATE,DELETE, por eso tiene sentido SQL

%ROWCOUNT

* LOS CURSORES NO LEVANTA ECEPCIONES\*\*

**Utilización de una subconsulta en un bucle for**

* Si utilizamos una subconsulta en un bucle FOR de cursor no hace falta declararse el cursor en la zona declarativa

##### EJEMPLO

BEGIN

FOR REG\_EMP IN (SELECT \* FROM scott.salgrade WHERE hisal>=2000) LOOP DBMS\_OUTPUT.PUT\_LINE(REG\_EMP.GRADE||':'|| reg\_emp.grade);

end loop; end;

/

## alias en las columnas

* Cuando utilizamos variables registro declaradas del mismo tipo que el cursor o que la tabla, los campos tiene el mismo nombre que las columnas correpondientes.
* Poner alias para acceder a las funciones grupo sino luego no puedo \*\*

Ejemplo:

CURSOR C1 Select dept\_no, count(\*) from emple group by dept\_no; OPEN C1;

FETCH c1 INTO v\_reg;

* Para visualizar una columna hacemos:

V\_reg.dept\_no

* Para visualizar lo que hemos contado no podríamos por eso debemos ponerles un alias

Ejemplo:

CURSOR C1 Select dept\_no, count(\*) total from emple group by dept\_no; OPEN C1;

FETCH c1 INTO v\_reg;

* Para visualizar el total hacemos:

V\_reg.total

EJERCICIO:

Escribir un bloque PL/SQL que visualice el ename y hiredate(fecha de alta) de la empresa, ordenados por fecha:

1. Mediante for de cursor
2. Mediante while

SOLUCIONES:

##### DECLARE

CURSOR c\_emple IS SELECT ename,hiredate\_from emple order by 2; V\_reg\_emp c\_emp%rowtype;

Begin

For v\_reg\_emp IN c\_emple LOOP DBMS\_OUTPUT.PUT\_LINE(v\_reg\_emp.ename||v\_reg\_emp.hiredate)

##### END LOOP; END;

/

##### DECLARE

CURSOR c\_emple IS SELECT ename,hiredate\_from emple order by 2; V\_reg\_emp c\_emp%rowtype;

Begin

OPEN c\_emple;

FETCH c\_emple INTOv\_reg\_emp; WHILE c\_emple%FOUND LOOP

DBMS\_OUTPUT.PUT\_LINE(v\_reg\_emp.ename||v\_reg\_emp.hiredate) FETCH c\_emple INTOv\_reg\_emp;

##### END LOOP;

Close c\_emple; END;

/

## FOR UPDATE :

* A veces podemos requerir bloquear las filas devueltas por el cursor para realizar alguna modificación en los datos.
* Para mantener bloqueadas las filas durante el proceso se utiliza FOR UPDATE en la sentencia SELECT asociada al cursor

##### DECLARE

CURSOR nombre IS SELECT \* FROM notas FOR UPDATE;

* La clausula FOR UPDATE es la última de la sentencia SELECT incluso después de ORDER BY si lo hubiese
* Los bloqueos de las filas se realizan en la apertura del cursor, no en las recuperaciones
* No se debe lanzar una sentencia Commit antes de cerrar un cursor que lleve la cláusula FOR UPDATE ya que en ese caso se liberan los bloqueos realizados por el cursor

Ejemplo:

##### DECLARE

CURSOR c\_dept IS SELECT \* FROM emp where deptno=10 FOR UPDATE;

##### BEGIN

OPEN c\_dept;

FETCH c\_dept INTO v\_reg; WHILE c\_dept %FOUND LOOP

IF v\_reg.comm IS NULL THEN UPDATE emp SET comm=300

WHERE empno=v\_reg.empno; END IF;

FETCH c\_dept INTO v\_reg; END LOOP;

CLOSE c\_notas; COMMIT;

##### END;

/

#### WHERE CURRENT OF nom\_cursor;

nom\_cursor.- Nombre del cursor que recuperó las filas a modificar o borrar.

* Se utiliza WHERE CURRENT OF para hacer referencia a la fila que se está utilizando actualmente sin necesidad de hacer referencia explícitamente con ROWID o algún identificativo de la fila
* Para utilizar esta claúsula , la sentencia select debe incluir la cláusula FOR UPDATE
* No se puede utilizar WHERE CURRENT OF cuando la select accede a más de una tabla
* Sintaxis:

WHERE CURRENT OF id\_cursor;

Ejemplo:

##### DECLARE

CURSOR c\_notas (nota notas.ev1 %TYPE) IS SELECT nom\_alu, ape1\_alu, ape2\_alu, ev1

FROM alumnos a, notas n WHERE a.num\_alu = n.num\_alu

AND ev1 >= nota FOR UPDATE;

nombre VARCHAR2(9); ape1 nombre %TYPE; ape2 nombre %TYPE; nota1 NUMBER(4,2);

##### BEGIN

OPEN c\_notas(7);

FETCH c\_notas INTO nombre, ape1, ape2, nota1; WHILE c\_notas %FOUND LOOP

IF nota1 > 7 AND nota1 < 10 THEN nota1 := nota1 + 1;

UPDATE notas SET ev1 = nota1 WHERE CURRENT OF c\_notas;

##### END IF;

FETCH c\_notas INTO nombre, ape1, ape2, nota1; END LOOP;

CLOSE c\_notas; COMMIT;

##### END;

/

 ERROR, ya que el **cursor está formado por más de una tabla** y por tanto no puede coincidir con la tabla **notas** que intentamos actualizar.

* La solución al caso anterior pasa por no utilizar la opción WHERE CURRENT OF nom\_cursor y utilizar otra condición en la cláusula WHERE.

Ejemplo:

##### DECLARE

CURSOR c\_notas (nota notas.ev1 %TYPE) IS SELECT n.num\_alu,nom\_alu,ape1\_alu,ape2\_alu,ev1

FROM alumnos a, notas n

WHERE a.num\_alu=n.num\_alu AND ev1>=nota;

num NUMBER(2);

nombre VARCHAR2(9); ape1 nombre %TYPE; ape2 nombre %TYPE; nota1 NUMBER(4,2);

##### BEGIN

OPEN c\_notas(7);

FETCH c\_notas INTO num,nombre,ape1,ape2,nota1; WHILE c\_notas %FOUND LOOP

IF nota1 > 7 AND nota1 < 10 THEN nota1 := nota1 + 1;

UPDATE notas SET ev1 = nota1 WHERE num\_alu = num;

##### END IF;

FETCH c\_notas INTO num, nombre, ape1, ape2, nota1; END LOOP;

CLOSE c\_notas; COMMIT;

##### END;

/

Ejemplo correcto:

DECLARE CURSOR c1 IS SELECT \*

FROM emp

WHERE deptno=10 FOR UPDATE;

BEGIN

FOR REG IN C1 LOOP

IF REG.COMM IS NULL THEN

UPDATE EMP SET COMM=300 WHERE CURRENT OF C1; END IF;

END LOOP;

END;

/