13. PL/SQL Avanzado

14.1. CURSORES

Cursores implícitos – los utilizados hasta ahora (SELECT ….INTO) para los casos en los que las **consultas devuelven una fila**.

CURSORES EXPLICITOS

Se utilizan para consultas que devuelven más de una fila.

Hay 4 operaciones básicas al trabajar con cursores:

- Declaración :

  Irá en la zona de declaraciones.

  Formato: CURSOR <nombre_cursor> IS <sentencia_select>;

  (*) La SELECT puede utilizar variables declaradas antes.

- Apertura:

  En la zona de instrucciones

  Formato: OPEN <nombre_cursor>;

  Al abrirlo se ejecuta la SELECT y los datos se almacenan en memoria, el puntero apuntará a la 1ª fila.

- Recogida de información

  FETCH <nombre_cursor> INTO {<variable> | <lista variables>};

  Variables es del tipo %ROWTYPE

  Cada FETCH recuperará una fila y el cursor avanzará a la fila siguiente.

- Cierre del cursor

  CLOSE < nombre_cursor>

ATRIBUTOS DEL CURSOR

Uso cursor %atributo.

Para conocer el estado del cursor en cada momento existen 4 atributos.

ATRIBUTO	DESCRIPCIÓN	EJEMPLO

%FOUND	Devuelve TRUE si el último FETCH ha devuelto una fila; sino devuelve FALSE.
	Antes de ningún FETCH devuelve NULL.
	Antes de abrir el cursor devuelve ERROR (invalid_cursor):
		LOOP
		FETCH cur1 INTO variable;
		IF cur1%FOUND THEN
		------ 	
		ELSE
		EXIT;
		END IF;
		END LOOP;

%NOTFOUND	Contrario a %FOUND. Se utiliza como condición de salida.	LOOP FETCH ..... EXIT WHEN cur1%NOTFOUND; - - - - - - END LOOP;

%ROWCOUNT	Devuelve nº de filas recuperadas hasta el momento.
	En la apertura se encuentra a 0.
		LOOP
		FETCH cur1 INTO variable;
		IF cur1%ROWCOUNT = 5 THEN
		EXIT;
		END IF;
		END LOOP;

%ISOPEN	Devuelve TRUE si el cursor está abierto.	IF cur1%ISOPEN THEN
		CLOSE cur1;
		ELSE
		OPEN cur1;
		END IF;

VARIABLE DE ACOPLAMIENTO: Variable que se utiliza en la condición. En el siguiente ejemplo SELECT APELLIDO FROM EMPLE WHERE DEPT_NO=20 el cursor debe seleccionar filas de acuerdo a una condición (DEPT_NO=20) en un programa PL las condiciones van cambiando para ello utilizamos las variables de acoplamiento que se declaran y se utilizan en la condición.

ESTRUCTURA CURSOR FOR .. LOOP

Estructura que simplifica las tareas, haciendo todas las operaciones excepto la declaración del cursor.

Formato y uso:

1. Se declara el CURSOR en la sección correspondiente.
2. Se procesa el CURSOR:

   FOR <nombrevareg> IN <nombrecursor> LOOP

   - - - - - -

   END LOOP;

14.2. EXCEPCIONES

Sirven para tratar errores u otras situaciones en tiempo de ejecución, así como errores y situaciones definidas por el usuario.

Funcionamiento:

Al producirse el error PL pasa el control a la zona de excepciones y dentro de ella a la excepción correspondiente; actuará según lo establecido y dará por finalizada la ejecución del bloque.

Formato:

```
EXCEPTION
        WHEN <nombre_excepción1> THEN
                <instrucciones>;
        WHEN <nombre_excepción 2> THEN
                <instrucciones>;
        [WHEN OTHERS THEN
                <instrucciones>;]
END <nombre_programa>;
```

Tipos de excepciones:

- Internas predefinidas en PL/SQL.
- Internas definidas por el ususario
- Otras asociadas a errores internos de Oracle.

14.2.1.INTERNAS PREDEFINIDAS

Excepción	Descripción
CURSOR_ALREADY_OPEN	Se activa si intentamos abrir un cursor que ya ha sido abierto con anterioridad.
DUP_VAL_ON_INDEX	Intento de almacenar en una tabla un valor que ya existe en un índice único.
INVALID_CURSOR	Operación ilegal sobre un cursor.
INVALID_NUMBER	Fallo de conversión de un caracter a un número.
LOGIN_DENIED	Nombre de usuario/contraseña no válido.
NO_DATA_FOUND	Una orden de tipo SELECT INTO no ha devuelto ningún valor.
PROGRAM_ERROR	Problema interno.
STORAGE_ERROR	Fallo de Memoria

TOO_MANY_ROWS	Una orden SELECT INTO ha devuelto más de una fila.
VALUE_ERROR	Error aritmético, de conversión, de truncamiento o de restricción.
ZERO_DIVIDE	Error al intentar dividir un número por cero.

Ejemplos:

a)
```
DECLARE
.....
BEGIN
.............
EXCEPTION
        WHEN NOT_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('Error datos no encontrados ');
        WHEN TOO_MANY_ROWS THEN
                DBMS_OUTPUT.PUT_LINE('Error Demasiados filas
recuperadas ');
        WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('Error');
END;
```

14.2.2. EXCEPCIONES INTERNAS DEFINIDAS POR EL USUARIO

- Se deben declarar

  DECLARE

  <nombre_excepción> EXCEPTION;

- En la zona de excepciones se determinará lo que hacer cuando la excepción se active.
- Para arrancar una excepción de este tipo utilizaremos la orden

  RAISE <nombre_excepción>;

Ejemplo:

```
IF <condición> THEN
        RAISE <nombre_excepción>;
END IF;
```

(*) Puede haber varios RAISE en el mismo bloque con la misma o distinta excepción.

Ejemplo:

```
DECLARE
        CURSOR alu IS SELECT codigo,ape1,nombre FROM Alumnos;
        Error_ape EXCEPTION;
BEGIN
        FOR x IN alu LOOP
                IF x.ape1 IS NULL THEN
                        RAISE error_ape;
                END IF;
                INSERT INTO tem(col1,col2) VALUES (x.codigo,x.ape||x.nombre);
        END LOOP;
EXCEPTION
        WHEN error_ape THEN
                INSERT INTO temp(col1,col2) VALUES (x.codigo,'!!!ERROR!!!
                Algunos de los apellidos está a nulos');
END;
```

14.2.3. EXCEPCIONES ASOCIADAS A ERRORES INTERNOS DE ORACLE.

Existen errores de ORACLE que en vez de tener asociada una excepción nos proporcionan un ERROR con un código y un mensaje.

Estos errores se tratan en la cláusula WHEN OTHERS.

Las funciones con las que accedemos al código y el mensaje de error son SQLCODE y SQLERRM respectivamente.

Ejemplo 1:

```
        WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('ERROR: ' || SQLCODE || ' – ' ||
SQLERRM);
```

También podemos asociar una excepción a alguno de estos errores internos de ORACLE que no tienen excepciones predefinidas asociadas. Para ello haremos lo siguiente:

a.- Definiremos una excepción en el DECLARE como las definidas por el usuario:

```
        <nombre_excepción> EXCEPTION;
```

b.- Asociaremos esa excepción a un código de error determinado, así:

```
        PRAGMA EXCEPTION_INIT (nombre_excepción> , <nº_error_Oracle>);
```

c.- Indicaremos el tratamiento a la excepción en la zona EXCEPTIONS como para cualquier otra excepción.

```
        WHEN <nombre_excepción>;
```

Esto es especialmente útil cuando queremos tratar varias excepciones de tipo interno.

Ejemplo 2: Crear un bloque donde se define la excepcion err_blancos asociada con un error definido por el programador y la excepcion no hay espacio asociándolo con el error numero –1547 de Oracle.

```
DECLARE
    Cod_err number(6);
    Men_err varchar2(30);
    Vcod varchar2(10);
    Vnom varchar2(15);
    Err_blancos EXCEPTION;
    No_hay_espacio EXCEPTION;
    PRAGMA EXCEPTION_INIT(no_hay_espacio,-1547);
BEGIN
    SELECT emp_no,apellido INTO vcod, vnom FROM EMPLE;
    IF SUBSTR(vnom,1,1) = ‘  ‘ THEN
        RAISE err_blancos;
    END IF;
    UPDATE clientes SET nombre=vnom WHERE nif=vnif;


EXCEPTION
    WHEN err_blancos THEN
        INSERT INTO terror(col1,col2) VALUES (vcod,‘ERR blancos’);
    WHEN no_hay_espacio THEN
        INSERT INTO terror(col2) VALUES (‘ERR tablespace’);
    WHEN NO_DATA_FOUND THEN
        INSERT INTO terror(col2) VALUES (‘ERR no habia datos’);
    WHEN TOO_MANY_ROWS THEN
        INSERT INTO terror(col2) VALUES (‘ERR demasiados datos’);
    WHEN OTHERS THEN
        Cod_err := SQLCODE;
        Men_err := SQLERRM;
        INSERT INTO terror(col1,col2) VALUES(cod_err , men_err);
END;
```

UTILIZACIÓN DE RAISE_APPLICATION_ERROR. Sirve para levantar errores y definir y enviar mensajes de error. Su formato es:

RAISE_APPLICATION_ERROR(número_error, mensaje):

Num_error entre –20000 y –20999

Mensaje hasta 512 bytes.

Cuando un subprograma hace esta llamada, se levanta la excepción y produce la salida del programa.

14.3. TRIGGERS DE BASE DE DATOS

Los triggers o disparadores de base de datos son bloques PL/SQL almacenados asociados a una tabla que se ejecutan o disparan automáticamente cuando se producen ciertos eventos o sucesos que afectan a la tabla, como inserción, borrado o modificación de filas.

Se utilizan para proporcionar mayor seguridad a la base de datos, para auditar actualizaciones, y otras medidas de seguridad.

Hay tres tipos de disparadores de bases de datos:

- DISPARDORES DE TABLAS, asociados a una tabla y se disparan cuando insertamos, borramos o modificamos en dicha tabla.
- DISPARADORES DE SUSTITUCIÓN, asociados a vistas.
- DISPARADORES DE SISTEMA.

DISPARDORES DE TABLAS.

Formato:

```
CREATE [OR REPLACE] TRIGGER <nombre_trigger>
   {BEFORE | AFTER} {DELETE | INSERT | UPDATE [OF
<lista_columnas>}
         [OR {BEFORE | AFTER} {DELETE | INSERT | UPDATE [OF
         <lista_columnas>}]..
   ON <nombre_tabla>
   [FOR EACH { STATEMENT | ROW [WHEN (<condición>)] } ]
/* aquí comienza el bloque del cuerpo trigger PL/SQL */
[DECLARE
.....
END;]
```

14.3.1. ELEMENTOS DEL TRIGGER

- *Nombre del trigger*
- *Evento del disparador* : INSERT, DELETE o UPDATE en esta última se puede especificar a que columnas afecta. Se pueden especificar varios eventos de disparo en un mismo trigger.
- *Tipo de trigger*: Hace referencia a dos cuestiones:
  - El momento de ejecución AFTER/BEFORE de la orden de manipulación.
  - El nivel de disparo :

    . A nivel de orden (por defecto, STATEMENT), se activará una sola vez en cada orden.

. A nivel de fila: se dispara una vez por cada fila afectada en la orden (FOR EACH ROW).

- *Restricción del trigger*: La cláusula WHEN restringe la ejecución del disparo al cumplimiento de una condición. Limitaciones de la condición:
  . Solamente se puede utilizar con triggers a nivel de fila
  . Es una condición SQL, no PL/SQL.
  . No puede incluir una consulta.
- *Cuerpo del trigger:* Código que se ejecutará cuando se dispare el trigger.

14.3.2. VALORES NEW Y OLD.

Al trabajar en el trigger podremos hacer referencia a valores anteriores o posteriores a una actualización referidos a una fila.

Formato :

[:] {new|old}.<columna>

(*) Cuando hagamos referencia a new y old en el WHEN del trigger no poner :

Ejemplo: IF :new.salario>:old.salario .....

Tener en cuenta:
- Solo se pueden utilizar en disparadores a nivel de fila.
- Cuando el evento es un DELETE sólo se puede utilizar old., new es NULL
- Cuando es un INSERT, sólo new.
- Para UPDATE , se puede ambos.

Si queremos sustituir new u old por otras palabras especificarlo en el REFERENCING antes del WHEN.

REFERENCING new AS n, old AS o...

14.3.3. ALGUNAS CONSIDERACIONES.

- Para crear un Trigger hay que tener privilegio CREATE TRIGGER.
- Un Trigger se puede asociar a una tabla o a una vista
- No se puede manipular en el cuerpo del trigger la tabla o vista que disparo el trigger.
- Si falla la ejecución del Trigger, se considera fallida la actualización a la que estaba asociado y se realizará un ROLLBACK de toda ella, incluso si esta hubiera afectado a varias filas.
- No se deben usar indiscriminadamente, ya que pueden degradar el comportamiento de la B.D., Recordar que hay controles que se pueden realizar utilizando otras herramientas, CHECK, PRIMARY KEY, FOREIGN KEY, SET DEFAULT.....
- Para borrar un trigger utilizaremos : DROP TRIGGER nombretrigger.
- para compilarlo: ALTER TRIGGER nombretrigger COMPILE
- Para activarlo: ALTER TRIGGER nombretrigger ENABLE.
- Vistas con información sobre TRIGGERS: DBA_TRIGGERS, USER_TRIGGERS.

14.3.3. ORDEN DE EJECUCIÓN DE DISPARADORES.

Si una tabla tiene asociados varios disparadores, el orden de ejecución es:

1. BEFORE FOR EACH STATEMENT
2. BEFORE FOR EACH ROW
3. Ejecución de la actual
4. AFTER FOR EACH ROW
5. AFTER FOR EACH STATEMENT

14.3.4.MULTIPLES EVENTOS EN EL DISPARO.

Cuando se contemplan varios eventos que disparan un trigger, por ejemplo BEFORE INSERT OR DELETE OR UPDATE, puede ser que haya distintas acciones en el cuerpo siguiente al evento. Para facilitar este control Oracle permite la utilización de predicados condicionales que devolveran un valor veradadero o falso para cada una de las posibles operaciones:

INSERTING, devuelve true si el evento que se disparo fue un INSERT.

DELETING, ‘’’’ DELETE.

UPDATING, ‘’’’ UPDATE.

UPDATING (‘columna’), ‘’’ UPDATE y la columna especificada ha sido actualizada.

EJEMPLO:

```
CREATE TRIGGER.....
BEFORE INSERT OR UPDATE OR DELETE ON EMPLEADOS
BEGIN
        IF INSERTING THEN.....
        ......
        ELSIF DELETING THEN
        ......
        ELSIF UPDATING('SALARIO') THEN
        .....
        END IF
        ....
END;
```

DISPARADORES DE SUSTITUCIÓN, estan asociados a vistas.
FORMATO:

```
CREATE [OR REPLACE] TRIGGER <nombre_trigger>
INSTEAD OF
   {DELETE | INSERT | UPDATE [OF <lista_columnas>}
   ON <nombre_vista>
   [FOR EACH  ROW]    [WHEN (<condición>)]
/* aquí comienza el bloque del cuerpo trigger PL/SQL */
[DECLARE
.................
END;]
```

DISPARADORES DE SISTEMA, se disparan cuando ocurre un evento del sistema(arranque o parada de la bd, entrada o salida de un ususario...) o una instrucción de definición de datos(creación, modificación o eliminación de un objeto).

FORMATO:

```
CREATE [OR REPLACE] TRIGGER <nombre_trigger>
{BEFORE | AFTER} {<lista eventos definicion> | <lista eventos del
sistema>}
ON {DATABASE | SCHEMA} [WHEN (condicion)]
/* aquí comienza el bloque del cuerpo trigger PL/SQL */
[DECLARE
.................
END;]
```