Tema 12.Programación en Bases De Datos (II)

Contenido

1. Cursores, Excepciones y Control De Transacciones En PL/SQL

1.1 Cursores

Hasta el momento hemos venido utilizando **cursores implícitos**. Este tipo de cursores es muy sencillo y cómodo de usar, pero plantea diversos problemas.

El más importante es que **la consulta debe devolver una fila** (y sólo una), de lo contrario, se produciría un error. Por ello, dado que normalmente una consulta devolverá varias filas, se suelen manejar cursores explícitos.

1.1.1. Cursores explícitos

Se utilizan para trabajar con consultas que pueden devolver más de una fila.

El cursor es un identificador, no es una variable. Solamente se puede usar para hacer referencia a una consulta. No se le pueden asignar valores ni utilizar en expresiones. No obstante, el cursor tiene, al igual que las variables, su ámbito.

Hay cuatro operaciones básicas para trabajar con un cursor explícito:

1. **Declaración del cursor**. El cursor se declara en la zona de declaraciones según el siguiente formato:

   CURSOR <nombrecursor> IS SELECT <sentencia select>;

2. **Apertura del cursor**. En la zona de instrucciones hay que abrir el cursor:

   OPEN <nombrecursor>

   Al hacerlo se ejecuta automáticamente la sentencia SELECT asociada y sus resultados se almacenan en las estructuras internas de memoria manejadas por el cursor. No obstante, para acceder a la información debemos realizar el paso siguiente:

3. **Recogida de información**. Para recoger información almacenada en el cursor utilizaremos el siguiente formato:

   FETCH <nombrecursor> INTO {<variable> | <listavariables>};

   Después del INTO figurará una variable que recogerá la información de todas las columnas. En este caso, la variable puede ser declarada de esta forma:

   <variable> <nombrecursor>%ROWTYPE

   O una lista de variables. Cada una recogerá la columna correspondiente de la claúsula SELECT, por tanto, serán del mismo tipo que las columnas.

   Cada FETCH recupera una fila y el cursor avanza automáticamente a la fila siguiente.

4. **Cierre del cursor**. Cuando el cursor no se va a utilizar hay que cerrarlo:

   CLOSE <nombrecursor>;

Ejemplo: esta forma de recuperar información con un cursor aparece en algunos libros y páginas web, pero es mejor utilizar bucles while, como en el ejemplo del apartado 3.1.2.

```
DECLARE
        CURSOR cur1 IS
        SELECT dnombre, loc FROM depart;
        v_nombre                VARCHAR2(14);
        v_localidad             VARCHAR2(14);
BEGIN
        OPEN cur1;
        LOOP
                FETCH cur1 INTO v_nombre, v_localidad;
                EXIT WHEN cur1%NOTFOUND;
                DBMS_OUTPUT.PUT_LINE(v_nombre || ' * ' ||  v_localidad);
        END LOOP;
        CLOSE cur1;
END;
```

La salida de este procedimiento será

```
CONTABILIDAD * SEVILLA
INVESTIGACIÓON * MADRID
VENTAS * BARCELONA
PRODUCCIÓN * BILBAO

Procedimiento PL/SQL terminado con éxito.
```

Podemos observar que, a diferencia de lo que ocurre en los cursores implícitos, la sentencia SELECT en la declaración del cursor no contiene la cláusula INTO para indicar las variables que recibirán la información. Esta cláusula INTO se especifica en FETCH.

Después de un FETCH debe comprobarse el resultado, con alguno de los atributos del cursor

Si utilizamos variable tipo cursor:

```
DECLARE
        CURSOR cur1 IS
                SELECT dnombre, loc FROM depart;
        v_cur1  cur1%ROWTYPE;
BEGIN
        OPEN cur1;
        LOOP
                FETCH cur1 INTO v_cur1;
                EXIT WHEN cur1%NOTFOUND;
                DBMS_OUTPUT.PUT_LINE (v_cur1.dnombre|| ' * ' || v_cur1.loc);
        END LOOP;
        CLOSE cur1;
END;
```

1.1.2. Atributos del cursor

Para conocer detalles respecto a la situación del cursor hay cuatro atributos para consultar:

- **%FOUND** Devuelve verdadero si el último FETCH ha recuperado algún valor; en caso contrario, devuelve falso. Si el cursor no estaba abierto devuelve error, y si estaba abierto pero no se había ejecutado aún ningún FETCH, devuelve NULL. Se suele utilizar como condición de continuación en bucles para

recuperar información. En el ejemplo anterior se puede sustituir el bucle y la condición de salida por:

```
…
BEGIN
        OPEN cur1;
        FETCH cur1 INTO v_nombre, v_localidad;
        WHILE cur1%FOUND  LOOP
                DBMS_OUTPUT.PUT_LINE (v_nombre || ' * ' || v_localidad);
                FETCH cur1 INTO v_nombre, v_localidad;
        END LOOP;
        CLOSE cur1;
END;
```

- **%NOTFOUND.** Hace lo contrario que el atributo anterior. Se suele utilizar como condición de salida en bucles:

```
…
EXIT WHEN cur1%NOTFOUND;
…
```

- **%ROWCOUNT** Devuelve el número de filas recuperadas hasta el momento por el cursor (número de FETCH realizados satisfactoriamente)

- **%ISOPEN** Devuelve verdadero si el cursor está abierto

1.1.3. Variables de acoplamiento en el manejo de cursores

En muchas ocasiones la cláusula SELECT del cursor deberá seleccionar las filas de acuerdo con una condición. Cuando se trabaja con SQL interactivo se introducen los términos exactos de la condición (visualizar los apellidos del empleado 7521).

Cuando se escribe un programa PL/SQL se suele utilizar un diseño más abierto, por lo que los términos exactos de esta condición solamente se conocen en tiempo de ejecución.

Ejemplos

Podemos observar que en la cláusula **WHERE** se incluye una variable que se deberá haber declarado previamente. Este tipo de variables recibe el nombre de **variables de acoplamiento**. El programa la sustituirá por su valor en el momento en que se abre el cursor, y se seleccionarán las filas según dicho valor. Aunque ese valor cambie durante la recuperación de los datos con FETCH, el conjunto de filas que contiene el cursor no variará

1.1.4. Cursor FOR..LOOP

Como ya hemos visto, el trabajo con un cursor consiste en:

- Declarar el cursor.
- Declarar una variable que recogerá los datos del cursor.
- Abrir el cursor.
- Recuperar con FETCH una a una las filas extraídas, introduciendo los datos en la variable, procesándolos, y comprobando también si se han recuperado datos o no.
- Cerrar el cursor.

Por eso PL/SQL proporciona la estructura cursor, FOR.. LOOP, que simplifica estas tareas realizando todas ellas, excepto la declaración del cursor, de manera implícita

El formato y el uso de esta estructura es:

1. Se declara la información del cursor en la sección correspondiente (como cualquier otro cursor).
2. Se procesa el cursor utilizando el siguiente formato:

```
FOR  nombrevareg  IN nombrecursor  LOOP
        ….
END LOOP;
```

Al entrar en el bucle, se abre el cursor de manera automática, se declara implícitamente la variable **nombrevareg** de tipo nombrecursor%ROWTYPE y se ejecuta el primer FETCH, cuyo resultado quedará en **nombrevareg**.

A continuación se realizarán las acciones que correspondan, hasta procesar la última fila de la consulta, momento en el que se producirá la salida del bucle y se cerrará automáticamente el cursor.

```
CREATE OR REPLACE PROCEDURE ver_emple_por_depart (p_dep number) AS
        v_dept NUMBER;
        CURSOR c1 IS SELECT apellido
                FROM emple WHERE dept_no = v_dept;
        v_apellido varchar2(100);
BEGIN
        v_dept:=p_dep;
        FOR vreg_c1 in c1 LOOP
                DBMS_OUTPUT.PUT_LINE( vreg_c1.apellido);
        END LOOP;
END;
```

También se podría salir del bucle FOR utilizando EXIT

La variable vreg_c1 se declara implícitamente y es local al bucle, por tanto, al salir del bucle la variable de registro no estará disponible.

Dentro del bucle se puede hacer referencia a la variable de registro y a sus campos (cuyo nombre se corresponde con las columnas de la consulta) usando la notación de punto:

```
DECLARE
        CURSOR c_emple IS
                select apellido, fecha_alta FROM emple;
BEGIN
        FOR v_reg_emp IN c_emple LOOP
                DBMS_OUTPUT.PUT_LINE('apellido:  '||  v_reg_emp.apellido  ||  ',
fecha de alta: '||v_reg_emp.fecha_alta);
        END LOOP;
END;
```

```
DECLARE
        CURSOR c_emple IS
                select apellido, fecha_alta FROM emple;
        v_reg_emp c_emple%rowtype;
BEGIN
        open c_emple;
        fetch c_emple into v_reg_emp;
        while(c_emple%FOUND) loop
```

```
                DBMS_OUTPUT.PUT_LINE('apellido: '|| v_reg_emp.apellido || ',
fecha de alta: '||v_reg_emp.fecha_alta);
                fetch c_emple into v_reg_emp;
        end loop;
        close c_emple;
END;
```

1.1.5. Uso de alias en las columnas de selección del cursor

Ya hemos indicado que cuando utilizamos variables de registro declaradas del mismo tipo que el cursor o que la tabla, los campos tienen el mismo nombre que las columnas correspondientes.

Cuando esas consultas son expresiones, se puede presentar un problema y debemos colocar alias en las columnas

```
CURSOR c1 IS
        SELECT dept_no, count(*)  n_emp, sum(salario+NVL(comision,0)) suma
        FROM emple
        GROUP BY dept_no;
```

1.1.6. Cursores con parámetros

El cursor puede tener parámetros; en este caso se aplicará el siguiente formato genérico:

```
CURSOR nombrecursor [(parámetro1, parámetro2,…)]  IS
        SELECT <sentencia select en la que intervendrán los parámetros>;
```

Los parámetros tienen la siguiente sintaxis:

```
        Nombredevariable [IN] tipodedato [{:= |DEFAULT} valor]
```

Todos los parámetros formales de un cursor son parámetros de entrada. El ámbito de estos parámetros es local al cursor, por eso solamente pueden ser referenciados dentro de la consulta.

```
DECLARE
v_dep  emple.dept_no%TYPE;
v_ofi    emple.oficio%TYPE;
Cursor CUR1 (p_departamento NUMBER, p_oficio VARCHAR2 DEFAULT 'DIRECTOR')
                IS SELECT apellido, salario  FROM  emple
                WHERE dept_no = p_departamento AND oficio = p_oficio;
```

Para abrir un cursor con parámetros:

```
        OPEN nombrecursor [(parámetro1, parámetro2, …)];
```

No tiene por qué ser los mismos nombres de las variables indicadas como parámetros al declarar el cursor; es más si lo fueran, serían consideradas como variables distintas

Para abrir el cursor, las siguientes líneas son correctas y válidas:

```
OPEN  cur1(v_dep);
OPEN cur1(v_dep,v_ofi);
OPEN cur1(20,'VENDEDOR');
```

En el caso de la instrucción FOR..LOOP, puesto que la orden OPEN va implícita, el paso de parámetros se hará a continuación del identificador del cursor:

```
….
        FOR  reg_emple  IN  cur1(20,'DIRECTOR')  LOOP
….
```

Notas importantes:

- Los parámetros formales de un cursor son siempre IN y no devuelven ningún valor ni pueden afectar a los parámetro actuales.
- La recogida de datos se hará, igual que en otros cursores explícitos, con FETCH.
- La cláusula WHERE (y las variables que en ella intervienen) asociada al cursor se evalúa solamente en el momento de abrir el cursor.

1.1.7. Atributos en cursores implícitos

Oracle abre implícitamente un cursor cuando procesa un comando SQL que no esté asociado a un cursor explícito. El cursor implícito se llama SQL y dispone también de los cuatro atributos mencionados, que pueden facilitarnos información sobre la ejecución de los comandos SELECT INTO, INSERT, UPDATE y DELETE

El valor de los atributos del cursor SQL se refiere, en cada momento, a la última orden SQL:

- SQL%NOTFOUND dará TRUE si el último INSERT, UPDATE O DELETE han fallado (no han afectado a ninguna fila). En el caso de que SELECT INTO no devuelva datos se levanta una excepción NO_DATA_FOUND y nunca podremos evaluar a continuación este atributo.
- SQL%FOUND dará TRUE si el último INSERT, UPDATE, DELETE o SELECT INTO han afectado a una o más filas. En el caso de que SELECT INTO devuelva más de una fila tampoco podremos evaluar a continuación este atributo porque se levanta una excepción TOO_MANY_ROWS.
- SQL%ROWCOUNT devuelve el número de filas afectadas por la última orden.
- SQL%ISOPEN siempre devolverá FALSO, ya que ORACLE cierra automáticamente el cursor después de cada orden SQL.

Estos atributos solamente están disponibles desde PL/SQL, no en órdenes SQL

El comportamiento de los atributos en los cursores implícitos es distinto al de los cursores explícitos

1. Devolverán un valor relativo a la última orden, aunque el cursor esté cerrado
2. SELECT.. INTO ha de devolver una fila y sólo una, pues de lo contrario se producirá un error y se levantará automáticamente una excepción:
   - NO_DATA_FOUND
   - TOO_MANY_ROWS

   Se detendrá la ejecución normal del programa y bifurcará a la sección EXCEPTION

3. Lo anterior no es aplicable a las órdenes INSERT, UPDATE, DELETE, ya que en estos casos no se levantan las excepciones correspondientes

Ejemplo

```
DECLARE
        v_dpto  depart.dnombre%TYPE;
        v_loc    depart.loc%TYPE;
BEGIN
        v_dpto := 'MARKETING';
        UPDATE depart SET loc='SEVILLA'
```

```
                WHERE DNOMBRE=v_dpto;
        IF SQL%NOTFOUND THEN
                DBMS_OUTPUT.PUT_LINE ('Error en la actualización');
        END IF;
        DBMS_OUTPUT.PUT_LINE ('Continúa el programa');
        SELECT loc INTO v_loc
                FROM depart
                WHERE dnombre=v_dpto;
        IF SQL%NOTFOUND THEN
                DBMS_OUTPUT.PUT_LINE ('IMPOSIBLE  nunca pasa por aquí');
        END IF;
END;
```

Cuando un SELECT INTO hace **referencia a una función de grupo** nunca se levantará la excepción NO_DATA_FOUND y SQL%FOUND siempre será verdadero. Esto se debe a que las funciones de grupo siempre retornan algún valor (aunque sea NULL).

1.2 Excepciones

Las excepciones sirven para tratar errores en tiempo de ejecución, así como errores y situaciones definidas por el usuario. Cuando se produce un error, PL/SQL levanta una excepción y pasa el control a la sección EXCEPTION correspondiente del bloque PL, que actuará según lo establecido y dará por finalizada la ejecución del bloque actual.

Para controlar posibles situaciones de error en otros lenguajes que no disponen de gestión de excepciones, se debe controlar después de cada orden cada una de las posibles condiciones de error.

El formato de la sección EXCEPTION es:

```
EXCEPTION
        WHEN <nombredeExcepción1> THEN
                <instrucciones1>;
        WHEN <nombredeExcepción2> THEN
                <instrucciones2>;
        …

        [WHEN OTHERS THEN
                <instrucciones>;]
END<nombre de programa>;
```

Hay tres tipos de excepciones:

- Excepciones internas predefinidas
- Excepciones definidas por el usuario
- Otras excepciones

1.2.1. Excepciones internas predefinidas

Están predefinidas por Oracle. Se disparan automáticamente al producirse determinados errores. En el cuadro adjunto se incluyen las excepciones más frecuentes con los códigos de error correspondientes:

Código error Oracle	Valor de SQL CODE	Excepción	Se disparan cuando….
ORA-06530	-6530	ACCESS_INTO_NULL	Se intenta acceder a los atributos de un objeto no inicializado.
ORA-06531	-6531	COLLECTION_IS_NULL	Se intenta acceder a elementos de una colección que no ha sido inicializada.
ORA-06511	-6511	CURSOR_ALREADY_OPEN	Intentamos abrir un cursor que ya se encuentra abierto.
ORA-00001	-1	DUP_VAL_ON_INDEX	Se intenta almacenar un valor que crearía duplicados en la clave primaria o en una columna con la restricción UNIQUE.
ORA-01001	-1001	INVALID_CURSOR	Se intenta realizar una operación no permitida sobre un cursor (por ejemplo, cerrar un cursor que no se ha abierto).
ORA-01722	-1722	INVALID_NUMBER	Fallo al intentar convertir una cadena a un valor numérico.
ORA-01017	-1017	LOGIN_DENIED	Se intenta conectar a ORACLE con un usuario o una clave no válidos.
ORA-01012	-1012	NOT_LOGGED_ON	Se intenta acceder a la base de datos sin estar conectado a Oracle.
ORA-01403	+100	NO_DATA_FOUND	Una sentencia SELECT ... INTO ... no devuelve ninguna fila.
ORA-06501	-6501	PROGRAM_ERROR	Hay un problema interno en la ejecución del programa. ;
ORA-06504	-6504	ROWTYPE_MISMATCH	La variable del cursor del HOST y la variable del cursor PL/SQL pertenecen a tipos incompatibles.
ORA-06533	-6533	SUBSCRIPT_OUTSIDE_LIMIT	Se intenta acceder a una tabla anidada o a un array con un valor de índice ilegal (p ejemplo, negativo).
ORA-06500	-6500	STORAGE_ERROR	El bloque PL/SQL se ejecuta fuera de memoria (o hay algún otro error de memoria).
ORA-00051	-51	TIMEOUT_ON_RESOURCE	Se excede el tiempo de espera para un recurso.
ORA-01422	-1422	TOO_MANY_ROWS	Una sentencia SELECT ... INTO ... devuelve más de una fila.
ORA-06502	-6502	VALUE_ERROR	Un error de tipo aritmético, de conversión, de truncamiento...
ORA-01476	-1476	ZERO_DIVIDE	Se intenta la división entre cero.

No hay que declararlas en la sección DECLARE.

```
DECLARE
        ….
BEGIN
….
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE ('ERROR datos no encontrados');
        WHEN TOO_MANY_ROWS THEN
                DBMS_OUTPUT.PUT_LINE ('ERROR demasiadas filas recuperadas');
```

```
END;
```

1.2.2. Excepciones definidas por el usuario

Para su utilización hay que dar tres pasos:

1. Se deben declarar en la sección DECLARE de la forma siguiente:

```
<nombreexcepción> EXCEPTION;
```

2. Se disparan o levantan en la sección ejecutable del programa con la orden RAISE;

```
RAISE <nombreexcepción>;
```

3. Se tratan en la sección EXCEPTION según el formato ya conocido:

```
WHEN <nombreexcepción> THEN<tratamiento>;
```

```
DECLARE
        .....
        importe_erroneo  EXCEPTION;
BEGIN
        ....
        IF precio NOT BETWEEN precio_min AND precio_maximo THEN
                RAISE importe_erroneo;
        END IF;
        ....
EXCEPTION
        .....
        WHEN importe_erroneo THEN
                DBMS_OUTPUT.PUT_LINE('Importe erróneo. Venta cancelada.');
END;
```

La cláusula RAISE <nombreexcepción> se puede usar varias veces en el mismo bloque con la misma o con distintas excepciones:

```
DECLARE
        venta_erronea EXCEPTION;
        importe_erroneo EXCEPTION;
BEGIN
        .....
        RAISE venta_erronea;
        RAISE importe_erroneo;
        RAISE venta_erronea;
                .....
EXCEPTION
        WHEN importe_erroneo THEN
                .....;
        WHEN venta_erronea THEN
                ......
END;
```

El siguiente ejemplo utiliza una excepción predefinida y otra definida por el programador:

```
CREATE OR REPLACE PROCEDURE subir_comision (p_num_empleado INTEGER,
p_incremento REAL)
IS
        v_comision_actual REAL;
        COMISION_NULA EXCEPTION;
BEGIN
        SELECT comision INTO v_comision_actual
        FROM emple
         WHERE emp_no = p_num_empleado;
         IF v_comision_actual IS NULL THEN
                RAISE COMISION_NULA;
        ELSE
                UPDATE emple SET comision = comision + p_incremento
                WHERE emp_no = p_num_empleado;
        END IF;
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE(p_num_empleado||  '*Error.  Empleado
no encontrado') ;
        WHEN COMISION_NULA THEN
                DBMS_OUTPUT.PUT_LINE(p_num_empleado||  '*Error.  Comision
nula');
END subir_comision;
```

1.2.3. Otras excepciones

Existen otros errores internos de Oracle, similares a los asociados a las excepciones internas pero que no tienen asignada una excepción, sino un código de error y un mensaje de error, a los que se accede mediante las funciones SQLCODE y SQLERRM.

Cuando se produce uno de estos errores se transfiere el control a la sección EXCEPTION, donde se tratará el error en la cláusula WHEN OTHERS:

```
        EXCEPTION
        …..
                WHEN OTHERS THEN
                        DBMS_OUTPUT.PUT_LINE('Error:  ' ||SQLCODE ||SQLERRM);
        ….
        END;
```

Mostrará el texto ‘Error: ’ con el código de error y el mensaje de error utilizando las funciones correspondientes.

```
…
        WHEN OTHERS THEN
                :cod_err := SQLCODE;
                :msg_err := SQLERRM;
                ROLLBACK;
                EXIT;
        END;
```

También podemos asociar una excepción a alguno de estos errores internos que no tienen excepciones predefinidas asociadas. Siguiendo los siguientes pasos:

1. Definimos una excepción en la sección de declaraciones como si fuese una excepción definida por el usuario:

```
<nombre_excepción> EXCEPTION;
```

2. Asociamos esa excepción a un determinado código de error mediante la directiva del compilador PRAGMA EXCEPTION_INIT, según el formato siguiente:

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

3. Indicamos el tratamiento que recibirá la excepción en la sección EXCEPTION como si se tratase de cualquier otra excepción definida o predefinida:

```
DECLARE
        ….
        ERR_EXTERNO EXCEPTION;    ---- Se define la excepción de usuario
  …
        PRAGMA EXCEPTION_INIT (ERR_EXTERNO, -1547);  /* Se asocia con un error de
Oracle*/
        ….
BEGIN
        ….
        /*  no hay que levantar la excepción, ya que llegado el caso Oracle lo hará  */
        ….
EXCEPTION
        …
        WHEN ERR_EXTERNO  THEN              -- Se trata como cualquier otra
                <tratamiento>;
END;
```

Ejemplo. Trata todos los tipos de excepciones. Antes de ejecutar el siguiente ejemplo crear la siguiente tabla y hacer la siguiente inserción:

```
CREATE TABLE TEMP2 (
   col1  VARCHAR2(25),
   col2  VARCHAR2(25)
 );

insert into temp2 values('888H',' MARIA');
```

```
DECLARE
        cod_err                 NUMBER(6);
        v_nif                   VARCHAR2(25);     /* igual tamaño que en TEMP2*/
        v_nom                   VARCHAR2(25);
        ERR_BLANCOS             EXCEPTION;  /* excepción definida por el usuario  */
        NO_HAY_ESPACIO          EXCEPTION;    /* excepción asociada a un error
interno  */
        PRAGMA  EXCEPTION_INIT (NO_HAY_ESPACIO, -1547);
BEGIN
        SELECT col1, col2  INTO  v_nif, v_nom FROM  TEMP2;
        IF  SUBSTR(v_nom, 1, 1) = ' ' THEN
                        RAISE ERR_BLANCOS;
        END IF;
        UPDATE clientes SET nombre = v_nom WHERE nif=v_nif;
```

```
EXCEPTION
        WHEN ERR_BLANCOS THEN
                INSERT INTO temp2 (col1) VALUES ('ERR blancos');
        WHEN NO_HAY_ESPACIO THEN
                INSERT INTO temp2 (col1) VALUES ('ERRtablespace');
        WHEN NO_DATA_FOUND THEN
                INSERT INTO temp2 (col2) VALUES (' ERR no había datos');      /*
dejar un espacio en blanco delante  */
        WHEN TOO_MANY_ROWS  THEN
                INSERT INTO temp2 (col1) VALUES ('ERR demasiados datos');
        WHEN OTHERS THEN
                cod_err := SQLCODE;
                INSERT  INTO  temp2(col1) VALUES (cod_err);
END;
```

Si queremos que dos o más excepciones ejecuten la misma secuencia de instrucciones, podremos indicarlo en la cláusula WHEN indicando las excepciones unidas por el operador OR:

```
        WHEN exc1 OR exc2 OR exc3 …. THEN ….
```

No obstante, en la lista no podrá aparecer WHEN OTHERS

1.2.4. Programación y ámbito de las excepciones

Algunos tipos de excepciones se han de tratar con mucha precaución, ya que se puede caer en un bucle infinito (por ejemplo NOT_LOGGED_ON).

La gestión de excepciones tiene las siguientes reglas:

- Cuando se levanta una excepción, el programa bifurca a la sección EXCEPTION del bloque actual. Si no está definida en ella, la excepción se propaga al bloque que llamó al actual, pasando el control a la sección EXCEPTION de dicho bloque y así hasta encontrar tratamiento para la excepción o devolver el control al programa Host.
- Una vez tratada la excepción en un bloque, se devuelve el control al bloque que llamó al que trató la excepción, con independencia del que la disparó.
- Si, después de tratar una excepción, queremos volver a la línea siguiente a la que se produjo, no podemos hacerlo directamente, pero sí es posible diseñar el programa para que funciones así. Esto se consigue encerrando el comando o comandos que pueden levantar la excepción en un subbloque junto con el tratamiento para la excepción:

```
….
SELECT  INTO …..     ---→ Puede levantar NO_DATA_FOUND
….
```

Para que el control del programa no salga del bloque actual, cuando se produzca una excepción, podemos encerrar el comando en un bloque y tratar la excepción en ese bloque:

```
BEGIN
        SELECT  INTO ….. → Puede levantar  NO_DATA_FOUND
EXCEPTION
        WHEN  NO_DATA_FOUND  THEN
        …..  ;       --   tratamiento para la excepción
END;
```

- La cláusula WHEN OTHERS tratará cualquier excepción que no aparezca en las cláusulas WHEN correspondientes, con independencia del tipo de excepción. De este modo se evita que la excepción se propague a los bloques de nivel superior.

  Tenemos el siguiente ejemplo:

  En el siguiente ejemplo, si se levanta la excepción ex1, se tratará en el mismo bloque, pero el control pasará después al bloque <<exterior>> en la línea siguiente a la llamada.

  Si se hubiese levantado NO_DATA_FOUND, al no encontrar tratamiento, la excepción se propaga al bloque <<exterior>>, donde será tratada por WHEN OTHERS (suponiendo que no exista un tratamiento específico), devolviendo el control al bloque o la herramienta que llamó a <<exterior>>.

```
<<exterior>>
DECLARE
        …
BEGIN
        ….
        <<interior>>
        DECLARE
                …
                ex1  EXCEPTION;
        BEGIN
        …..
                RAISE ex1;
                ….
                SELECT col1,col2  INTO  ….;   -- > levanta  NO_DATA_FOUND
                …
        EXCEPTION
                ….
                WHEN ex1 THEN              -- > Tratamiento para ex1
                        ROLLBACK;                  -- > Después pasará el control a (1)
                …..
        END;
        /*  (1)  */
        ….
EXCEPTION
        ….
        WHEN OTHERS  THEN …
END;
```

- Si la excepción se levanta en la sección declarativa (por un fallo al inicializar una variable, por ejemplo) automáticamente se propagará al bloque que llamó al actual, sin comprobar si existe tratamiento para la excepción en el mismo bloque que se levantó.

- También se puede levantar una excepción en la sección EXCEPTION de forma voluntaria o por un error que se produzca al tratar una excepción. En este caso, se propagará de forma automática al bloque que llamó al actual, sin comprobar si existe tratamiento para la excepción en el mismo bloque que se levantó. La excepción original (la que se estaba tratando cuando se produjo nueva excepción) se perderá, ya que solamente puede estar activa una excepción.

```
EXCEPTION
        WHEN  INVALID_NUMBER  THEN
                INSERT  INTO ….          -- podría levantar DUP_VAL_ON_INDEX
        WHEN  DUP_VAL_ON_INDEX  THEN              -- no atrapará la excepción
                …..
END;
```

- En ocasiones, puede resultar conveniente, después de tratar una excepción, volver a levantar la misma excepción para que se propague al bloque de nivel superior. Esto puede hacerse indicando al final de los comandos de manejo de la excepción el comando RAISE sin parámetros:

```
…
WHEN  TOO_MANY_ROWS  THEN
        ….;                -- instrucciones de manejo de error
        RAISE;   -- levanta de nuevo y la propaga
…
```

- Las excepciones declaradas en un bloque son locales al bloque, por tanto, no son conocidas en bloques de nivel superior. No se puede declarar dos veces la misma excepción en el mismo bloque, pero sí en distintos bloques. En este caso, la excepción del subbloque prevalecerá sobre la del bloque, aunque esta última se puede levantar desde el subbloque utilizando la notación de punto (RAISE nombrebloque. nombreexcepción).

- Las variables locales, las globales, los atributos de un cursor, etc. Se pueden referenciar desde la sección EXCEPTION según las reglas de ámbito que rigen para los objetos del bloque. Pero si la excepción se ha disparado dentro de un bucle cursor FOR … LOOP no se podrá acceder a los atributos del cursor, ya que Oracle cierra este cursor entes de disparar la excepción

1.2.5. Utilización de RAISE_APPLICATION_ERROR

En el paquete DBMS_STANDARD se incluye un procedimiento muy útil llamado RAISE_APPLICATION_ERROR que sirve para levantar errores y definir y enviar mensajes de error. Su formato es el siguiente:

RAISE_APPLICATION_ERROR (número_de_error, mensaje_de_error)

Número_de_error es un número comprendido entre –20000 y –20999, y mensaje_de_error es una cadena de hasta 512 bytes

Cuando un subprograma hace esta llamada, se levanta la excepción y se deshacen los cambios realizados por el subprograma. Esta excepción solamente puede ser manejada con WHEN OTHERS.

Ejemplo:

Modificar el procedimiento anterior subir_comision utilizando en vez de la excepción creada por el usuario la de RAISE_APPLICATION_ERROR, con el número –20010

```
CREATE OR REPLACE PROCEDURE subir_comision2 (p_num_empleado INTEGER,
p_incremento REAL)
IS
        v_comision_actual REAL;
  /*COMISION_NULA EXCEPTION;*/
BEGIN
        SELECT comision INTO v_comision_actual
        FROM emple
        WHERE emp_no = p_num_empleado;
        IF v_comision_actual IS NULL THEN
                RAISE_APPLICATION_ERROR(-20010, ' Error. Comision nula');
        ELSE
                UPDATE emple SET comision = comision + p_incremento
                WHERE emp_no = p_num_empleado;
END IF;
```

```
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE(p_num_empleado||  '*Error.  Empleado
no encontrado') ;
        /*WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('UN ERROR');*/
END subir_comision2;
```

Probar la ejecución con **`EXECUTE subir_comision2(7369, 100)`** y ver qué ocurre si se incluyen las líneas que están comentadas en el código.

1.3 Control de transacciones

Una transacción se puede definir como un conjunto de operaciones dependientes unas de otras que se realizan en la base de datos. Una transacción por tanto, no se circunscribe al ámbito de una orden SQL o al de un bloque PL/SQL, sino que es el usuario (el programador, en este caso) quien decide cuáles serán las operaciones que compondrán la transacción.

Oracle garantiza la consistencia de los datos en una transacción en términos de VALE TODO o NO VALE NADA, es decir, o se ejecutan todas las operaciones que componen una transacción o no se ejecuta ninguna. Así pues, la base de datos tiene un estado antes de la transacción y un estado después de la transacción, pero no hay estados intermedios.

Una transacción comienza con la primera orden SQL de la sección del usuario o con la primera orden SQL posterior a la finalización de la transacción anterior.

La transacción finaliza cuando se ejecuta un comando de control de transacciones (COMMIT o ROLLBACK), una orden de definición de datos (DDL) o cuando finaliza la sesión.

Una vez completada la transacción ya no puede deshacerse.

```
BEGIN
        …
        UPDATE cuentas SET saldo=saldo –v_importe_tranfer
                WHERE num_cta = v_cta_origen;
        UPDATE cuentas SET saldo = saldo + v_importe_tranfer
                WHERE num_cta = v_cta_destino;
        COMMIT;
        …
EXCEPTION
        WHEN OTHERS  THEN
                ROLLBACK;
END;
```

En el ejemplo anterior se garantiza que la transferencia se llevará a cabo totalmente o que no se realizará ninguna operación, pero en ningún caso se quedará a medias.

- **ROLLBACK** implícitos. Cuando un subprograma almacenado falla y no se controla la excepción que produjo el fallo. Oracle automáticamente ejecuta ROLLBACK sobre todo lo realizado por el subprograma, salvo que en el subprograma hubiese algún COMMIT, en cuyo caso lo confirmado no sería deshecho.

- **SAVEPOINT** Se utiliza para poner marcas o puntos de salvaguarda al procesar transacciones. Se utiliza con ROLLBACK TO. Esto permite deshacer parte de una transacción

Ejemplo:

```
CREATE OR REPLACE PROCEDURE  prueba_savepoint (p_numfilas  POSITIVE)
AS
BEGIN
        SAVEPOINT ninguna;
        INSERT  INTO temp2 (col1) VALUES ('PRIMERA  FILA');
        SAVEPOINT  UNA;
        INSERT INTO temp2(col1) VALUES ('SEGUNDA FILA');
        SAVEPOINT  DOS;
        IF p_numfilas = 1 THEN
                ROLLBACK  TO UNA;
        ELSIF pnumfilas = 2 THEN
                ROLLBACK TO DOS;
        ELSE
                ROLLBACK TO NINGUNA;
        END IF;
        COMMIT;
EXCEPTION
        WHEN OTHERS  THEN
                ROLLBACK;
END;
```

Podemos observar que **ROLLBACK TO** <punto_de_salvaguarda> deshace el trabajo realizado sobre la base de datos después del punto indicado, incluyendo posibles bloqueos. No obstante, tampoco se confirma el trabajo hecho hasta el punto_de_salvaguarda. La transacción no finaliza hasta que se ejecuta un comando de control de transacciones COMMIT o ROLLBACK, o hasta que finaliza la sesión (o se ejecuta una orden de definición de datos DDL).

Oracle establece un **punto de salvaguarda implícito** cada vez que se ejecuta una sentencia de DDL de datos. En el caso de que la sentencia falle, Oracle restaurará automáticamente los datos a sus valores iniciales.

Por omisión, el número de **SAVEPOINT** está limitado a **cinco por sesión**, pero se puede cambiar en el parámetro SAVEPOINT del fichero de inicialización de Oracle hasta 255.

Cuando se ejecuta un ROLLBACK TO <marca>, todas las marcas después del punto indicado desaparecen (la indicada no desaparece). También desaparecen todas las marcas cuando se ejecuta un COMMIT.

Los nombres de las marcas son identificadores no declarados y se pueden reutilizar.

- **SET TRANSACTION READ ONLY**. Establece el comienzo de una **transacción de sólo lectura**. Se utiliza para garantizar la consistencia de los datos recuperados entre distintas consultas. Todas las consultas que se ejecutan a continuación solamente verán aquellos cambios confirmados antes del comienzo de la transacción: es como si se hiciese una fotografía de la base de datos.

  Antes de usar SET TRANSACTION READ ONLY debemos confirmar o rechazar la transacción en curso estableciendo así el comienzo de la nueva transacción. Una vez efectuadas todas las operaciones de consulta cuya consistencia queremos garantizar, introduciremos COMMIT para dar por finalizada la transacción de sólo lectura

```
DECLARE
        v_num_ventas_dia  NUMBER;
        v_num_ventas_semana  NUMBER;
BEGIN
        COMMIT; --Confirma transacción anterior e inicia la nueva
        SET TRANSACTION READ ONLY; -- indica que la transacción será de sólo
lectura
        SELECT count(*) INTO v_num_ventas_dia
```

```
        FROM  ventas
        WHERE fecha =TO_DATE ('18/10/97');
        dbms_output.put_line('ventas día ' || v_num_ventas_dia);
        SELECT count(*) INTO v_num_ventas_semana
        FROM  ventas
        WHERE fecha between TO_DATE ('18/10/97') - 7 and TO_DATE ('18/10/97');
        dbms_output.put_line(' ventas semana ' || v_num_ventas_semana);
        COMMIT;
END;
```

1.4 Uso de cursores para actualizar filas

- **Cursor FOR UPDATE.** Hasta el momento hemos venido utilizando los cursores sólo para seleccionar datos, pero también se puede usar el nombre de un cursor que apunta a una fila para realizar una operación de actualización en esa fila.

  Cuando se prevea esa posibilidad, a la declaración del cursor habrá que añadirle FOR UPDATE al final:

```
CURSOR nombrecursor <declaración del cursor> FOR UPDATE
```

FOR UPDATE indica que las filas seleccionadas por el cursor van a ser actualizadas o borradas. Todas las filas seleccionadas serán bloqueadas tan pronto se abra (OPEN) el cursor y serán desbloqueadas al terminar las actualizaciones (al ejecutar COMMIT explícito o implícitamente)

Una vez declarado un cursor FOR UPDATE, se añadirá el especificador CURRENT OF nombredecursor en la cláusula WHERE para actualizar (UPDATE) o borrar (DELETE) la última recuperada mediante la orden FETCH

```
{UPDATE | DELETE } … WHERE CURRENT OF nombredecursor
```

Ejemplo:

Realizar un procedimiento que suba el salario a todos los empleados del departamento indicado en la llamada (dos parámetros: número de departamento y aumento en tanto por ciento)

```
CREATE OR REPLACE PROCEDURE subir_salario_dpto (p_num_dpto    NUMBER,
        p_pct_subida   NUMBER)
AS
        v_inc    number(8,2);
        CURSOR c_emple IS SELECT oficio, salario FROM emple
                        WHERE dept_no=p_num_dpto FOR UPDATE;
        v_reg_c_emple  c_emple%ROWTYPE;
BEGIN
        OPEN c_emple;
        FETCH c_emple  INTO  v_reg_c_emple;
        WHILE c_emple%FOUND  LOOP
                v_inc := (v_reg_c_emple.salario/100) * p_pct_subida;
                UPDATE emple  SET salario= salario + v_inc
                WHERE CURRENT  OF c_emple;
                FETCH c_emple  INTO  v_reg_c_emple;
        END  LOOP;
        CLOSE c_emple;
END subir_salario_dpto;
```

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

```
CURSOR  nombredecursor <declaracióndelcursor> FOR UPDATE OF nombredecolumna
```

```
DECLARE
        …
        CURSOR c_emple  IS SELECT  oficio, salario
                                FROM  emple, depart
                                WHERE emple.dept_no=depart.dept_no
          FOR UPDATE OF salario;        /* bloquea sólo la tabla emple  */
        ….
```

- **Uso de ROWID en lugar de FOR UPDATE**. La utilización de la cláusula FOR UPDATE plantea algunas cuestiones que, algunas veces, pueden ser problemáticas:
  - Se bloquean todas las filas de la SELECT, no sólo la que se está actualizando en un momento dado
  - Si se ejecuta un COMMIT, después ya no se puede ejecutar FETCH. Es decir, tenemos que esperar a que estén todas las filas actualizadas para confirmar los cambios.

Para evitar estos problemas se puede utilizar el ROWID, que indicará la fila que se va a actualizar en lugar de FOR UPDATE. Para ello, al declarar el cursor en la cláusula SELECT indicaremos que seleccione también el identificador de fila o ROWID:

```
CURSOR nombredecursor  IS SELECT  col1,col2,… ROWID  FROM tabla;
```

Al ejecutar el FETCH se guardará el número de la fila en una variable o en un campo de la variable, y después ese número se podrá usar en la cláusula WHERE de la actualización:

```
{ UPDATE | DELETE}  … WHERE  ROWID = variable_que_guarda_rowid
```

Ejemplo:

Modificar el procedimiento anterior utilizando ROWID

```
CREATE OR REPLACE  PROCEDURE  subir_salario_dpto_b (
        p_num_dpto    NUMBER,
        p_pct_subida    NUMBER)
AS
        v_inc    number(8,2);
        CURSOR c_emple IS SELECT oficio, salario, ROWID  FROM emple
                        WHERE dept_no=p_num_dpto  ;
        vreg_c_emple  c_emple%ROWTYPE;
BEGIN
        OPEN c_emple;
        FETCH c_emple  INTO  vreg_c_emple;
        WHILE c_emple%FOUND  LOOP
                v_inc := (vreg_c_emple.salario/100) * p_pct_subida;
                UPDATE emple  SET salario= salario + v_inc
                WHERE rowid = vreg_c_emple.ROWID;
                FETCH c_emple  INTO  vreg_c_emple;
        END  LOOP;
        CLOSE c_emple;
END subir_salario_dpto_b;
```