CURSORES

Hasta el momento hemos venido utilizando *cursores implícitos*. Son muy cómodos y sencillos pero plantean diversos problemas. Lo más importante es que la subconsulta debe envolver 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.

A. Cursores explícitos

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

Las operaciones permitidas con el cursor son: *declarar, abrir, recoger información* y *cerrar el cursor*. No se le pueden asignar valores ni utilizarlo en expresiones.

Hay cuatro operaciones básicas para trabajar con un cursor explícito:

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

   ```
   CURSOR <nombrecursor> IS <sentencia SELECT>;
   ```

2. **Apertura** del cursor en la zona de instrucciones:

   ```
   OPEN <nombrecursor>;
   ```

   La instrucción OPEN ejecuta automáticamente la sentencia SELECT asociada y sus resultados se almacenan en las estructuras internas de memoria manejadas por el cursor.

   En realidad, un **cursor** es una estructura que apunta a una región de la PGA que contiene toda la información relativa a la consulta asociada, en especial, las filas de datos recuperadas.

   No obstante, para acceder a la información debemos dar el paso siguiente.

3. **Recogida de información almacenada** en el cursor. Se usa el comando FETCH con el siguiente formato:

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

   Después de INTO figurará:

- Una variable que recogerá la información de todas las columnas. Puede declararse de esta forma:

```
<variable> <nombrecursor>%ROWTYPE;
```

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

Cada FETCH recupera una fila y el cursor avanza automáticamente a la fila siguiente en cada nueva instrucción FETCH.

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

```
CLOSE <nombrecursor>;
```

El siguiente ejemplo utiliza un cursor explícito para visualizar el nombre y la localidad de todos los departamentos de la tabla DEPART.

```
DECLARE
    CURSOR cur1 IS
       SELECT dnombre, loc FROM depart;      --1.Declarar
    v_nombre VARCHAR2(14);
    v_localidad VARCHAR2(14);
BEGIN
   OPEN cur1;                                --2.Abrir
   LOOP
    FETCH cur1 INTO v_nombre, v_localidad;   --3.Recoger
     EXIT WHEN cur1%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE (v_nombre ||'*'||v_localidad);
   END LOOP;
   CLOSE cur1;                               --4.Cerrar
END;
```

El resultado de la ejecución de este bloque será:

```
CONTABILIDAD*SEVILLA
INVESTIGACION*MADRID
VENTAS*BARCELONA
PRODUCCION*BILBAO
```

Observamos que:

- La sentencia SELECT en la declaración del cursor no contiene la cláusula INTO a diferencia de lo que ocurre en los cursores implícitos.

- La instrucción FETCH se usa para recuperar cada una de las filas seleccionadas por la consulta. Esta información se deposita en las variables que siguen a la cláusula INTO.

- Después de un FETCH debe comprobarse el resultado, lo que se suele hacer preguntando por el valor de alguno de los atributos del cursor que se mencionan en el siguiente apartado.

B. Atributos del cursor

Hay cuatro atributos para consultar detalles de la situación del cursor:

- **%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. 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.

La siguiente tabla muestra los valores de retorno de los atributos del cursor en diferentes situaciones:

		%FOUND	%ISOPEN	%NOTFOUND	%ROWCOUNT
OPEN	Antes	Invalid_cursor	F	Invalid_cursor	Invalid_cursor
	Después	NULL	T	NULL	0
PRIMER FETCH	Antes	NULL	T	NULL	0
	Después	T	T	F	1
SIGUIENTES FETCH	Antes	T	T	F	1
	Después	T	T	F	…
ULTIMO FETCH	Antes	T	T	F	N
	Después	F	T	T	N
CLOSE	Antes	F	T	T	N
	Después	Invalid_cursor	F	Invalid_cursor	Invalid_cursor

Tabla 10.1. *Valores de retorno de los atributos del cursor en diferentes situaciones.*

Caso práctico

1 El siguiente ejemplo ilustra lo que hemos visto hasta ahora respecto a cursores y atributos de cursor: se trata de visualizar los apellidos de los empleados pertenecientes al departamento 20 numerándolos secuencialmente.

```
DECLARE
   CURSOR c1 IS
   SELECT apellido FROM emple WHERE dept_no=20;
   v_apellido VARCHAR2(10);
BEGIN
   OPEN c1;
   LOOP
      FETCH c1 INTO v_apellido;
      DBMS_OUTPUT.PUT_LINE(TO_CHAR(c1%ROWCOUNT,'99.')
                        ||v_apellido);
      EXIT WHEN c1%NOTFOUND;
   END LOOP;
   CLOSE C1;
END;
```

El resultado de la ejecución será:

```
1.SANCHEZ
2.JIMENEZ
3.GIL
4.ALONSO
5.FERNANDEZ
5.FERNANDEZ
```

En este ejemplo observamos que el último FETCH no devuelve ningún valor, no incrementa el atributo %ROWCOUNT y no sobrescribe el valor de la variable del cursor. Es evidente que el programa está mal diseñado, ya que procesa (visualiza) la información supuestamente recuperada antes de comprobar si se ha recuperado información nueva.

En caso de que FETCH no recupere una nueva fila:

- No se incrementará el atributo %ROWCOUNT.
- No se sobrescribe el valor de la variable del cursor.

Actividades propuestas

1 Escribe el ejercicio anterior subsanando el error de diseño para que no aparezca el último empleado duplicado. Hacerlo primero manteniendo el bucle LOOP...EXIT WHEN, y posteriormente probar con un bucle WHILE. Observa las diferencias en ambos casos.

C. Variables de acoplamiento en el manejo de cursores

La cláusula SELECT del cursor deberá seleccionar frecuentemente las filas de acuerdo con una condición. Cuando se trabaja con SQL interactivo se introducen los términos exactos de la condición. Veamos un ejemplo:

```
SQL> SELECT apellido FROM emple
     WHERE dept_no = 20;
```

Pero en un programa PL/SQL los términos exactos de esta condición solamente se conocen en tiempo de ejecución. Esta circunstancia obliga a utilizar un diseño más abierto. Las variables de acoplamiento cumplen esta función. Su forma de uso suele ser:

1. Se declara la variable como cualquier otra.

2. Se utiliza la variable en la sentencia SELECT como parte de la expresión.

```
CURSOR nombrecursor IS clausulaselectconvariableacoplam;
```

Caso práctico

2 En el siguiente ejemplo se visualizan los empleados de un departamento cualquiera usando variables de acoplamiento:

```
CREATE OR REPLACE PROCEDURE ver_emple_por_dept (
dep VARCHAR2)
AS
   v_dept NUMBER(2);
   CURSOR c1 IS
       SELECT apellido FROM emple WHERE dept_no = v_dept;
   v_apellido VARCHAR2(10);
BEGIN
   v_dept := dep;
   OPEN c1;
   FETCH c1 INTO v_apellido;
   WHILE c1%FOUND LOOP
       DBMS_OUTPUT.PUT_LINE(v_apellido);
       FETCH c1 INTO v_apellido;
   END LOOP;
  CLOSE c1;
END;
```

El programa sustituirá la variable 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.

También podíamos haber usado directamente el parámetro formal *dep* en lugar de la variable *v_dept*. El resultado es el mismo.

(Continúa)

(Continuación)

Una vez creado el procedimiento, se puede ejecutar:

```
SQL> EXECUTE ver_emple_por_dept(30);
  ARROYO
  SALA
  MARTIN
  NEGRO
  TOVAR
  JIMENO
```

Actividades propuestas

2 **Escribe un procedimiento que reciba una cadena y visualice el apellido y el número de empleado de todos los empleados cuyo apellido contenga la cadena especificada. Al finalizar, visualiza el número de empleados mostrados. El procedimiento empleará variables de acoplamiento para la selección de filas y los atributos del cursor estudiados en el epígrafe anterior.**

D. Cursores FOR...LOOP

Al escribir la sentencia SELECT del cursor debemos cuidar que seleccione únicamente aquellas filas con las que va a trabajar el programa pues de lo contrario sobrecargaremos el sistema y deberemos emplear código adicional para filtrar o tratar las filas requeridas.

Como ya hemos visto, en muchas ocasiones 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.

La estructura de cursores FOR...LOOP 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 el cursor en la sección declarativa (como cualquier otro cursor).

```
CURSOR <nombrecursor> IS <sentencia SELECT>;
```

2. Se procesa el cursor utilizando el siguiente formato:

```
FOR <nombrevareg>  IN <nombrecursor>  LOOP
        ...
END LOOP;
```

Donde *nombrevareg* es el nombre de la variable de registro que creará el bucle para recoger los datos del cursor.

Al entrar en el bucle:

- Se abre el cursor de manera automática.
- Se declara implícitamente la variable *nombrevareg* de tipo *nombrevareg* %ROWTYPE y se ejecuta un FETCH implícito, cuyo resultado quedará en *nombrevareg*.
- A continuación, se realizarán las acciones que correspondan hasta llegar al END LOOP, que sube de nuevo al FOR...LOOP ejecutando el siguiente FETCH implícito, y depositando otra vez el resultado en *nombrevareg*, y así sucesivamente, hasta procesar la última fila de la consulta. En ese momento, se producirá la salida del bucle y se cerrará automáticamente el cursor.

En el siguiente ejemplo, se visualizan el apellido, el oficio y la comisión de los empleados cuya comisión supera 500 € utilizando CURSOR FOR...LOOP:

```
DECLARE
   CURSOR mi_cursor IS
       SELECT apellido, oficio, comision FROM emple
         WHERE comision > 500;
BEGIN
   FOR v_reg IN mi_cursor LOOP
       DBMS_OUTPUT.PUT_LINE(v_reg.apellido||'*'||
         v_reg.oficio||'*'||TO_CHAR(v_reg.comision));
   END LOOP;
END;
```

El resultado será:

```
SALA*VENDEDOR*650
MARTIN*VENDEDOR*1020
```

Cabe subrayar que la variable *nombrevareg* 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.

El siguiente ejemplo muestra las diferencias en el uso de cursores FOR...LOOP con otras estructuras convencionales.

Caso práctico

3 **Escribiremos un bloque PL/SQL que visualice el apellido y la fecha de alta de todos los empleados ordenados por fecha de alta.**

1. Mediante una estructura cursor FOR...LOOP.

```
DECLARE
   CURSOR c_emple IS
       SELECT apellido, fecha_alt FROM emple
       ORDER BY fecha_alt;
BEGIN
   FOR v_reg_emp IN c_emple LOOP
       DBMS_OUTPUT.PUT_LINE(v_reg_emp.apellido||'*'||
          v_reg_emp.fecha_alt);
   END LOOP;
END;
```

2. Utilizando un bucle WHILE.

```
DECLARE
   CURSOR c_emple IS
       SELECT apellido, fecha_alt FROM emple
       ORDER BY fecha_alt;
   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(v_reg_emp.apellido||'*'||
           v_reg_emp.fecha_alt);
       FETCH c_emple INTO v_reg_emp;
   END LOOP;
   CLOSE c_emple;
END;
```

Podemos observar que el bucle FOR...LOOP realiza implícitamente la mayoría de las operaciones con el cursor (abrir, comprobar, recuperar fila y cerrar).

E. 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 al tratar de referenciarlas:

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

En estos casos, debemos indicar un alias en la columna:

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

F. Cursores con parámetros

En lugar de usar variables de acoplamiento podemos usar parámetros para determinar los términos exactos de la sentencia SELECT cada vez que se abre el cursor.

El uso de parámetros permite que cualquier programa pueda llamar al cursor sin necesidad de conocer y/o manipular las variables de acoplamiento.

La declaración de un cursor con parámetros se realiza indicando la lista de parámetros entre paréntesis a continuación del nombre del cursor. Los parámetros declarados se usarán en la sentencia SELECT de la declaración tal como se muestra a continuación:

```
CURSOR <nombrecursor>  [ (parámetro1, parámetro2, ...)  ]
IS SELECT <sentencia select con parámetros>;
```

Los parámetros formales indicados después del nombre del cursor tienen la siguiente sintaxis:

```
<Nombredeparámetro>  [IN]  <tipodedato>  [{ := | DEFAULT }
<valor>]
```

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
  ...
  CURSOR cur1
    (v_departamento NUMBER,
     v_oficio VARCHAR2 DEFAULT 'DIRECTOR')
     IS SELECT apellido, salario FROM emple
     WHERE dept_no = v_departamento AND oficio = v_oficio;
```

La apertura del cursor pasándole parámetros se hará:

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

Donde parámetro1, parámetro2, ... son expresiones que contienen los valores que se pasarán al cursor. No tienen 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.

Supongamos, por ejemplo, las siguientes declaraciones:

```
DECLARE
   v_dep emple.dept_no%TYPE;
   v_ofi emple.oficio%TYPE;
   CURSOR cur1
       (v_departamento NUMBER,
        v_oficio VARCHAR2 DEFAULT 'DIRECTOR')
       IS SELECT apellido, salario FROM emple
       WHERE dept_no = v_departamento AND oficio = v_oficio;
 ...
```

El uso de parámetros en cursores es muy habitual en conjunción con paquetes. En la próxima unidad estudiaremos más ejemplos y ejercicios de este tipo.

Cualquiera de los siguientes comandos abrirá el cursor:

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

Debemos recordar que:

- Los parámetros formales de un cursor son siempre IN y no devuelven ningún valor ni pueden afectar a los parámetros actuales.
- La recogida de datos se hará, igual que en otros cursores explícitos, con FETCH.
- La cláusula WHERE asociada al cursor se evalúa solamente en el momento de abrir el cursor. En ese momento es cuando se sustituyen las variables por su valor.

En el caso de los cursores FOR...LOOP, puesto que la instrucción OPEN va implícita, el paso de parámetros se hará a continuación del identificador del cursor en la instrucción FOR...LOOP, tal como se muestra en el ejemplo:

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

E. Cursores y rupturas de secuencia

Como acabamos de estudiar, el uso de cursores facilita el procesamiento secuencial de los datos recuperados. Esto incluye la posibilidad de rupturas de secuencia según el criterio o criterios determinados por el problema, que deberán ser los criterios de ordenación. A continuación, escribiremos un ejemplo de aplicación de esta técnica.

Caso práctico

4 Escribe un programa que muestre, en formato similar a las rupturas de control o secuencia vistas en SQL*Plus los siguientes datos:

- Para cada empleado: apellido y salario.
- Para cada departamento: número de empleados y suma de los salarios del departamento.
- Al final del listado: número total de empleados y suma de todos los salarios.

```
CREATE OR REPLACE PROCEDURE listar_emple
AS
 CURSOR c1 IS
       SELECT apellido, salario, dept_no FROM emple
ORDER BY dept_no, apellido;
   vr_emp c1%ROWTYPE;
 dep_ant EMPLE.DEPT_NO%TYPE DEFAULT 0;
 cont_emple NUMBER(4) DEFAULT 0;
 sum_sal NUMBER(9,2) DEFAULT 0;
 tot_emple NUMBER(4) DEFAULT 0;
 tot_sal NUMBER(10,2) DEFAULT 0;
BEGIN
 OPEN c1;
 LOOP
      FETCH c1 INTO vr_emp;

/* Si es el primer Fetch inicializamos dep_ant */
      IF c1%ROWCOUNT = 1 THEN
          dep_ant := vr_emp.dept_no;
      END IF;

/* Comprobación nuevo departamento (o finalización) y resumen del anterior e ini-
cialización de contadores y acumuladores parciales  */
IF dep_ant <> vr_emp.dept_no OR c1%NOTFOUND THEN
       DBMS_OUTPUT.PUT_LINE('*** DEPTO: ' || dep_ant ||
           ' NUM. EMPLEADOS: ' || cont_emple ||
                ' SUM. SALARIOS:  ' || sum_sal);
dep_ant := vr_emp.dept_no;
                    tot_emple := tot_emple + cont_emple;
```

(Continúa)

(Continuación)

```
            tot_sal := tot_sal + sum_sal;
            cont_emple := 0;
            sum_sal := 0;
            END IF;
      EXIT WHEN c1%NOTFOUND; /* Condición de salida del bucle */

   /* Escribir Líneas de detalle incrementar y acumular */
       DBMS_OUTPUT.PUT_LINE(RPAD(vr_emp.apellido,10) || ' * '
              ||LPAD(TO_CHAR(vr_emp.salario,'999,999'),12));
         cont_emple := cont_emple + 1;
            sum_sal := sum_sal + vr_emp.salario;

    END LOOP;
    CLOSE c1;

   /* Escribir totales informe */
      DBMS_OUTPUT.PUT_LINE(' ****** NUMERO TOTAL EMPLEADOS: '
         || tot_emple || ' TOTAL SALARIOS: ' || tot_sal);

   END listar_emple;
```

F. 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, DELETE o SELECT INTO ha fallado (no ha afectado a ninguna fila).

- **SQL%FOUND** dará TRUE si el último INSERT, UPDATE, DELETE o SELECT INTO ha afectado a una o más filas.
- **SQL%ROWCOUNT** devuelve el número de filas afectadas por el último INSERT, UPDATE, DELETE o SELECT INTO.
- **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.

Es preciso hacer algunas observaciones respecto al uso de atributos en cursores implícitos pues su comportamiento difiere, en algunos casos, al de los cursores explícitos. De hecho, como acabamos de ver, el cursor estará cerrado inmediatamente después de procesar la orden SELECT...INTO, que es cuando se pregunta por su atributo o atributos (lo cual no determina un error como pasaba en los cursores explícitos).

A continuación, se especifican algunas peculiaridades en el comportamiento y uso de los atributos en cursores implícitos:

- Devolverán un valor relativo a la última orden INSERT, UPDATE, DELETE o SELECT...INTO, aunque el cursor esté cerrado.
- En el caso de que se trate de un SELECT...INTO, debemos tener en cuenta que 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, si la consulta no devuelve ninguna fila.
  - TOO_MANY_ROWS, si la consulta devuelve más de una fila.

  Se detendrá la ejecución normal del programa y bifurcará a la sección EXCEPTION. Por tanto, cualquier comprobación de la situación del cursor en esas circunstancias resulta inútil.
- Lo indicado en el párrafo anterior no es aplicable a las órdenes INSERT, UPDATE, DELETE, ya que en estos casos no se levantan las excepciones correspondientes.

El siguiente ejemplo ilustra estas observaciones: se trata de un bloque que pretende cambiar la localidad de un departamento cuyo nombre en este caso es MARKETING (sabiendo que no existe ese departamento).

```
DECLARE
  v_dpto depart.dnombre%TYPE := 'MARKETING'; --(NO existe)
  v_loc  depart.loc%TYPE;
BEGIN
  UPDATE depart SET loc = 'SEVILLA'
     WHERE dnombre = v_dpto;  /* no actualiza ninguna fila
                              pero no levanta excepción */
    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;   /* fallará y levantará
                                     NO_DATA_FOUND */
    IF SQL%NOTFOUND THEN
       DBMS_OUTPUT.PUT_LINE('Nunca pasará por aquí');
    END IF;
END;
```

El resultado de la ejecución del programa será:

```
Error en la actualización
Continúa el programa
...
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line
```

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 o cero).

```
BEGIN
    ...
    SELECT MAX(salario) INTO v_max
      FROM emple
      WHERE dept_no = num_depart;  -- > nunca levantará
                                   --  NO_DATA_FOUND
    IF SQL%NOTFOUND THEN           -- > nunca será cierto
        ...                        -- > nunca se ejecutará
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN        -- > no será invocado
      ...
```

Esta característica resulta útil para comprobar la existencia o no de una determinada fila sin que se levante NO_DATA_FOUND. Por ejemplo, para comprobar si existe un determinado departamento podemos escribir:

```
SELECT COUNT (*) INTO v_dummy
   FROM depart WHERE dept_no = v_depart;
IF v_dummy > 0 THEN ...
```

G. Uso de cursores para actualizar filas

Los cursores se pueden usar para actualizar filas. En este caso, tenemos las siguientes opciones:

- **Cursor FOR UPDATE.** Son cursores que permiten y facilitan la actualización (modificación o eliminación) de las filas seleccionadas por el cursor. Todas las filas seleccionadas serán bloqueadas tan pronto se abra el cursor (OPEN) y serán desbloqueadas al terminar las actualizaciones (al ejecutar COMMIT explícita o implícitamente).

  Para crearlos únicamente habrá que añadirle FOR UPDATE al final de la declaración:

  ```
  CURSOR <nombrecursor> IS <sentencia SELECT del cursor>
  FOR UPDATE;
  ```

  Los cursores así declarados se usan exactamente igual que los cursores explícitos estudiados (OPEN, FETCH, etcétera). Pero además de estas operaciones, permiten actualizar la última fila recuperada con FETCH mediante el comando (UPDATE o DELETE) incluyendo el especificador WHERE CURRENT OF *nombrecursor* en la cláusula para actualizar (UPDATE) o borrar (DELETE).

  El formato para actualizar la fila seleccionada por un cursor FOR UPDATE es:

  ```
  {UPDATE | DELETE} ... WHERE CURRENT OF <nombrecursor>
  ```

  El siguiente procedimiento subirá el salario todos los empleados del departamento indicado en la llamada. La subida será el porcentaje indicado en la llamada:

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

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 <sentencia SELECT del cursor>
FOR UPDATE OF <nombrecolumna>

DECLARE
  ...
  CURSOR c_emple IS SELECT  oficio, salario
    FROM emple, depart
    WHERE emple.dept_no = depart.dept_no
    FOR UPDATE OF salario;
  ...
```

El bloqueo de filas y la determinación de una única transacción de los cursores FOR UPDATE suele ser lo más adecuado cuando queremos actualizar todas las filas seleccionadas por el cursor.

La utilización de la cláusula FOR UPDATE, en ocasiones, puede ser problemática pues:

- 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.

- **Uso de ROWID.** Consiste en usar el identificador de fila (ROWID) como condición de selección para actualizar filas. Para ello procederemos:
  - Al declarar el cursor en la cláusula SELECT, indicaremos que seleccione también el identificador de fila o ROWID:

    ```
    CURSOR nombrecursor 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 del cursor. Después ese número se usará en la cláusula WHERE de la actualización:

    ```
    {UPDATE | DELETE} ... WHERE ROWID =
    <variable_que_guarda_rowid>
    ```

La actualización de filas con ROWID realiza bloqueos individuales de fila. Normalmente optaremos por ella cuando se prevean pocas actualizaciones en relación con el número de filas seleccionado por el cursor.

En el ejemplo del epígrafe anterior:

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

En caso de que tengamos que declarar explícitamente la variable que recogerá el ROWID debemos tener en cuenta que esta pseudocolumna tiene su propio tipo con el mismo nombre (ROWID), tal como estudiamos en la unidad anterior. Por tanto, declararemos la variable así:

```
<nombredevariable> ROWID;
```

Además del uso de cursores FOR UPDATE y del ROWID, podemos seguir usando cualquier condición que permita la selección de las filas a actualizar.