Introducción a PL/SQL. Parte I.

Ayuda para prácticas. 2º cuatrimestre.

PL/SQL

Es un lenguaje procedimental que amplía la funcionalidad de SQL añadiendo estructuras habituales en otros lenguajes de programación como:

-Variables y tipos (predefinidos y definidos por el usuario)
-Estructuras de control (bucles y condiciones IF-THEN-ELSE)
-Procedimientos y funciones.
-Tipos de objetos y métodos

Creación de programas PL/SQL.

Podemos crear programas con cualquier editor y ejecutarlos desde el prompt de sql con STAR o @. Los ficheros creados serán de texto y tendrán la extensión sql.
Para que un fichero se ejecute correctamente debe tener en su última línea el símbolo /.

Características de PL/SQL

La unidad básica en PL/SQL es el *bloque*. Todos los programas de PL/SQL están compuestos por bloques que pueden estar anidados.

Estructura de un Bloque:

```
DECLARE
 /*Aquí se declaran las variables, tipos cursores y subprogramas locales*/

BEGIN
/* Aquí se incluyen las órdenes SQL y PL/SQL, es obligatoria y debe contener, al
menos una orden ejecutable */
END;

EXCEPTION
 /* Sección para el manejo de excepciones (errores)*/
END;
```

-Podemos crear diferentes tipos de bloques:

Bloques anónimos: Se construyen de forma dinámica y se suelen ejecutar una sola vez.
Bloques nominados: Igual que los anónimos pero con una etiqueta que les da nombre.
Subprogramas: Procedimientos, paquetes y funciones, almacenados en la BD y que se ejecutan en múltiples ocasiones. Los subprogramas se ejecutarán mediante una llamada.
Disparadores ("Triggers"): Bloques nominados que se almacenan en la BD y se ejecutan ante algún suceso.

- Para dar nombre a un bloque se le pone una etiqueta antes del DECLARE . **<<etiqueta>>**
- Para transformar un bloque en un procedimiento almacenado reemplazamos la palabra clave **DECLARE** por **CREATE OR REPLACE PROCEDURE** *nombre_procedimiento* **AS**

Variables y tipos:

Podemos declarar variables, cada una debe tener su tipo asociado.
Las variables pueden ser de los mismos tipos que las columnas de una base de datos:

```
DECLARE
v_NombreEstudiante   VARCHAR2(20);
v_FechaActual        DATE;
v_Puntos             NUMBER(3);
```

- También existen otros tipos adicionales (enteros binarios y lógicos):

```
DECLARE
v_ContadorBucle      BINARY_INTEGER;
v_Registrado         BOOLEAN;
```

BINARY_INTEGER: Se usa para almacenar valores que sólo van a ser utilizados en cálculos y no se van a almacenar en la BBDD.
BOOLEAN: Puenden contener los valores TRUE,FALSE o NULL.

Sintaxis: *nombre_variable tipo* [CONSTANT] [NOT NULL] [:=*valor*]

- Además admite tipos definidos por el usuario (tablas y registros):

```
DECLARE
 TYPE t_RegistroEstudiante IS RECORD (
   Nombre            VARCHAR2(10),
   Apellido1         VARCHAR2(15),
   Edad              NUMBER(3)
); v_Estudiante t_RegistroEstudiante;
```

- PL/SQL 8 también admite tipos de objetos

- El atributo %TYPE permite declarar una variable del mismo tipo que otra ya existente, especialmente útil para declarar variables del mismo tipo que atributos de una tabla.

NombreVariable *variableReferencia***%TYPE;**

Identificiadores

Los identificadores válidos empiezan por una letra que puede ser seguida de una secuencia de caracteres que puede incluir letras, números,$,_ ,#. La longitud máxima de un identificador es de 30 caracteres.

Literales

Podremos utilizar literales numéricos (enteros y reales), booleanos (TRUE,FALSE Y NULL) y de carácter (uno o más caracteres delimitados por comillas simples).

Estructuras de bucle

Existen cuatro tipos de bucles (LOOP,WHILE,FOR y FOR de Cursor)

- **Bucle simple LOOP**

```
LOOP
  Secuencia de ordenes;
END LOOP;
```

```
DECLARE
  v_ContBucle BINARY_INTEGER := 1;
BEGIN
  LOOP
    INSERT INTO temp_table (num_col)
      VALUES (v_ContBucle);
    v_ContBucle := v_ContBucler + 1;
    EXIT WHEN v_ContBucle > 50;
  END LOOP;
END;
```

El bucle se repite indefinidamente, debemos añadir una condición de salida mediante la orden EXIT

```
EXIT[WHEN condición];
```

Ejemplo (Bucle que inserta 50 filas en la tabla temp_table)

```
DECLARE
  V_Contador BINARY_INTEGER := 1;
BEGIN
  WHILE V_Contador <= 50 LOOP
    INSERT INTO temp_table
      VALUES (V_Contador, 'Indice bucle');
    V_Contador := V_Contador + 1;
  END LOOP;
END;
```

- **Bucle WHILE**

```
WHILE condición LOOP
  Secuencia de ordenes;
END LOOP;
```

El funcionamiento es el habitual de los bucles con condición inicial.
Podemos salirnos del bucle en cualquier momento incluyendo una sentencia EXIT.

- **Bucle FOR numérico**

```
FOR contador_bucle IN [REVERSE] v_inicial..v_final LOOP
  Secuencia de ordenes;
END LOOP;
```

```
BEGIN
  FOR v_Contador IN 1..50 LOOP
    INSERT INTO temp_table
      VALUES (v_Contador, 'Indice Bucle');
  END LOOP;
END;
```

-La variable índice de un bucle FOR se declara implícitamente como BINARY_INTEGER.
-Si se incluye la palabra REVERSE el bucle se ejecuta desde el valor final hasta el inicial.
-Los valores inicial y final pueden ser cualquier expresión que pueda ser convertida a un valor numérico.

Estructura IF_THEN-ELSE

Sintaxis:

IF *expresión2* **THEN**
  secuencia de ordenes1;
[**ELSIF** *expresión2* **THEN**
  secuencia de ordenes2;]
....
[**ELSE**
  secuencia de ordenes3;]
END IF;

```
DECLARE
  v_NumberSeats rooms.number_seats%TYPE;
  v_Comment VARCHAR2(35);
BEGIN
  /* Extrae el número de asientos de la habitación
     cuyo identificador es 99999, almacenando el
     resultado en v_NumberSeats*/
  SELECT number_seats
    INTO v_NumberSeats
    FROM rooms
    WHERE room_id = 99999;
  IF v_NumberSeats < 50 THEN
    v_Comment := 'Pequeña;
  ELSIF v_NumberSeats < 100 THEN
    v_Comment := 'Mediana;
  ELSE
    v_Comment := 'Grande';
  END IF;
END;
```

En cada sección del IF pueden aparecer tantas órdenes como se quiera.
Su funcionamiento es el habitual de estas construcciones.

Depuración de programas, entrada y salida de datos por consola.

La forma normal de introducir datos en una BBDD no es a través de un programa en PL/SQL, éstos suelen realizar operaciones específicas dentro de la BBDD sin interacción con un operador. En cualquier caso existen algunas funciones que nos pueden ayudar a depurar programas mostrando datos por pantalla o leyendo datos.

- Para mostrar un valor cadena podemos utilizar:

  DBMS_OUTPUT.PUT_LINE(*cadena)*

En caso de que el valor a mostrar no sea una cadena puedes utilizar la función TO_CHAR() para transformarlo.

El paquete DBMS_OUTPUT simplemente implementa una cola, si queremos que los datos aparezcan por pantalla tenemos que activar la opción SERVEROUTPUT mediante la orden de SQL*Plus:

SET SERVEROUTPUT ON [SIZE *tamaño del buffer*]

Esta orden llama de forma implícita a DBMS_OUTPUT.ENABLE que configura el buffer interno, ademas con la opción SERVEROUTPUT activada SQL*Plus llamará a DBMS_OUTPUT.GET_LINES después de que se haya completado el bloque PL/SQL. De forma que la salida se visualizará en pantalla una vez completado el bloque PL/SQL.

- Una forma sencilla de leer valores de pantalla para utilizarlos puede ser con la instrucción de SQL **ACCEPT** y variables de sustitución. (Variables disponibles en SQL distintas de las de PL/SQL).

Una variable de sustitución puede aparecer directamente en una sentencia SELECT sin necesidad de definirla, va precedida del símbolo & y SQL nos preguntará que valor queremos asignarle (la variable es sustituida por el literal que escribamos, si lo que escribimos queremos considerarlo como caracteres o fechas **la pondremos entre comillas**, para formar una cadena con el literal sustituido):

```
SELECT empnum, enom, sal, dptmto
FROM emp
WHERE empnum = &num_emp;
```

```
SELECT empnum, enom, sal, dptmto
FROM emp
WHERE emnom = '&nombre_emp';
```

Al ejecutarse una instrucción SQL como las anteriores nos mostrará un mensaje en el PROMPT pidiéndonos el valor de la variable :

Enter value for num_emp:

Enter value for nombre_emp:

👌 Especificaremos **SET VERIFY OFF,** si no queremos que SQL nos muestre por pantalla el valor anterior que tenía la variable y el nuevo que toma.

ACCEPT permite declarar una variable de SQL y leer su valor poniendo un mensaje en el Prompt.

ACCEPT *variable* [NUMBER|CHAR|DATE] [FORMAT] format]
[PROMPT *text*]] [HIDE]

● Para utilizar la variable accedemos a ella anteponiéndole el símbolo &.

☹ No podemos utilizar ACCEPT para leer variables dentro de un bloque PL/SQL, si queremos utilizarlo debemos hacerlo fuera!

```
SET VERIFY OFF
ACCEPT producto PROMPT ' Introduce el precio: '
ACCEPT iva PROMPT ' Introduce el IVA: '
SET SERVEROUTPUT ON
DECLARE
v_producto NUMBER := &producto;
v_iva NUMBER := &iva;
BEGIN
  dbms_output.put_line('Valor producto:'¦¦to_char(v_producto)) ;
…….
…….
END;
/
```

Registros y Tablas

PL/SQL permite utilizar tipos compuestos definidos por el usuario (Registros y Tablas)

Registros

La sintaxis general para definir un tipo registro es:

```
TYPE tipo_registro IS RECORD(
 campo1 tipo1 [NOT NULL][:= expr1],
 campo2 tipo2 [NOT NULL][:= expr2],
.....
 campoN tipoN [NOT NULL][:= exprN]);
```

Ejemplo:

```
DECLARE
TYPE t_EjRegistro IS RECORD(
 Cont NUMBER (4),
 Nombre VARCHAR(10) NOT NULL := 'Ana',
 Fecha DATE,
 Descripcion VARCHAR (45) := 'Ejemplo');

v_Ejemplo1 t_EjRegistro;
v_Ejemplo2 t_Ejregistro;
```

- Para hacer referencia a los campos de un registro se utiliza el punto (.) ***nombre_registro.nombre_campo***.
- Para poder asignar un registro a otro ambos deben ser del mismo tipo.
- También se pueden asignar valores a un registro completo mediante la orden SELECT que extraería datos de la BD y los almacenaría en el registro:

```
DECLARE
  --Define un registro con algunos campo de la tabla students

  TYPE t_StudentRecord IS RECORD (
    FirstName  students.first_name%TYPE,
    LastName   students.last_name%TYPE,
    Major      students.major%TYPE);

  -- Declara una variable para recibir los datos
  v_Student  t_StudentRecord;
BEGIN
  -- Recupera información del estudiante conID 10,000.
    SELECT first_name, last_name, major
    INTO v_Student
    FROM students
    WHERE ID = 10000;
END;
/
```

- Es bastante habitual definir un registro en PL/SQL con los mismos tipos que una fila de una base de datos. Esto se puede realizar directamente con el operador %ROWTYPE

Ejemplo

```
DECLARE
 v_RegAlumno     alumno%ROWTYPE
```

Tablas:

La sintaxis general para definir una tabla es:

TYPE *tipotabla* **IS TABLE** ***OF*** *tipo* **INDEX BY BINARY_INTEGER**

Ejemplos:

```
DELCARE
TYPE t_tablaNombres IS TABLE OF students.first_name%TYPE
  INDEX BY BINARY INTEGER;
TYPE t_tablaFechas IS TABLE OF DATE
  INDEX BY BINARY INTEGER;

v_Nombres t_tablaNombres;
v  fechas     t  tablaFechas;
```

Ordenes SQL en PL/SQL

Las únicas órdenes SQL permitidas en un programa PL/SQL con las del DML, y las de control de transacciones.

Las órdenes DML permitidas con SELECT, INSERT, UPDATE y DELETE.

PL/SQL permite utilizar variables dentro de una orden SQL allí donde este permitido usar una expresión.(no todo componente de una orden SQL se puede reemplazar con una variable, sólo las expresiones. En particular los nombres de tabla y de columna deben ser conocidos).

SELECT

Una orden SELECT extrae datos de la base de datos y los almacena en variables PL/SQL, podemos utilizar una lista de variables separadas por comas o una variable registro. Si la lista de selección es simplemente *, podría definirse este registro mediante *nombre_tabla*%ROWTYPE.

Esta forma de la orden SELECT no debería devolver más de una fila, si no es así PL/SQL nos devolverá un mensaje de error. Para extraer más de una fila en una consulta debe utilizarse un cursor para extraer individualmente cada fila.

Ejemplo uso:

```
DECLARE
  v_StudentRecord  students%ROWTYPE;
  v_Department     classes.department%TYPE;
  v_Course         classes.course%TYPE;
BEGIN
  -- Recupera un registro de la tabla students y lo almacena
  --  en v_StudentRecord. La cláusula WHERE sólo
  -- se corresponderá conuna fila.
  -- a consulta devuelve todos los campos (*) por ello el registro
  -- en que almacenamos el
  -- resultado se define como students%ROWTYPE.
  SELECT *
    INTO v_StudentRecord
    FROM students
    WHERE id = 10000;
```

```
  --Recupera dos campos y los almacena
  --en dos variables
  --La cláusula WHERE sólo
  --  debecorresponderse con una fila
  SELECT department, course
    INTO v_Department, v_Course
    FROM classes
    WHERE room_id = 99997;
END;
/
```

INSERT

Ejemplo del uso de INSERT:

```
DECLARE
  v_StudentID  students.id%TYPE;
BEGIN
  -- Extrae un nuevo número de estudiante
  SELECT student_sequence.NEXTVAL
    INTO v_StudentID
    FROM dual;

  -- Añade una fila a la tabla students
  INSERT INTO students (id, first_name, last_name)
    VALUES (v_StudentID, 'Timothy', 'Taller');
```

```
  -- Añade otra fila usando directamente el número
  -- de secuencia en la orden INSERT

  INSERT INTO students (id, first_name, last_name)
    VALUES (student_sequence.NEXTVAL,
'Patrick', 'Poll');
END;
/
```

Si la orden INSERT contiene un SELECT, la lista de selección debe corresponderse con las columnas que van a ser insertadas.

UPDATE

Ejemplo de uso:

Si la orden contiene un SELECT, la lista de selección debe corresponderse con las columnas de la cláusula SET.

```
DECLARE
  v_Major          students.major%TYPE;
  v_CreditIncrease  NUMBER := 3;
BEGIN
  -- Esta orden añade 3 al campo current_credits
  -- de todos los estudiantes de la especialidad de ‘History’
  v_Major := 'History';
  UPDATE students
    SET current_credits = current_credits + v_CreditIncrease
    WHERE major = V_Major;
END;
/
```

DELETE

Elimina filas de una tabla, indicando la cláusula WHERE de la orden qué filas hay que eliminar.
Ejemplo:

```
DECLARE
  v_StudentCutoff  NUMBER;
BEGIN
  v_StudentCutoff := 10;
  -- borra todas las clases que tengan menos de un número de
  -- estudiantes

  DELETE FROM classes
    WHERE current_students < v_StudentCutoff;

  --Borra todos los estudiantes de economía sin créditos
DELETE FROM students
    WHERE current_credits = 0
    AND   major = 'Economics';
END;
/
```

Introducción a PL/SQL. Parte II.

Cursores

Para procesar una orden SQL, Oracle asigna un área de memoria que recibe el nombre de área de

```
DECLARE
  /* Variables de salida para almacenar los resultados de la consulta */
  v_StudentID    students.id%TYPE;
  v_FirstName    students.first_name%TYPE;
  v_LastName     students.last_name%TYPE;

  /* Variable de acoplamiento utilizada en la consulta */
  v_Major        students.major%TYPE := 'Computer Science';

  /* Declaración del cursor */
  CURSOR c_Students IS
    SELECT id, first_name, last_name
      FROM students
      WHERE major = v_Major;
BEGIN
  /* Identifica las filas en el conjunto activo y prepara el
     posterior procesamiento de los datos */
  OPEN c_Students;
  LOOP
    /* Recupera cada fila del conjunto activo y
       almacena los datos en variables */
    FETCH c_Students INTO v_StudentID, v_FirstName, v_LastName;

    /* Si no hay más filas que recuperar salir del bucle */
    EXIT WHEN c_Students%NOTFOUND;
  END LOOP;

  /* Libera los recursos utilizados en la consulta */
  CLOSE c_Students;
END;
/
```

contexto. Esta área contiene información sobre el procesamiento, como el número de filas procesadas por la orden y en caso de consultas el *conjunto activo*, que es el conjunto de filas resultado de la consulta.

Un ***cursor*** es un puntero al área de contexto. Mediante un cursor, un programa PL/SQL puede controlar el área de contexto, tendremos que utilizar un cursor, por ejemplo, para procesar las distintas filas de datos que devuelva una consulta:

Este ejemplo muestra el uso de un ***cursor explícito*** donde se asigna explícitamente el nombre de un cursor a una orden SELECT.

Los cuatro pasos PL/SQL necesarios para el procesamiento de un cursor explícito son:

1 Declaración del Cursor.
2 Apertura del cursor para una consulta.
3 Recogida de los resultados en variables PL/SQL.
4 Cierre del cursor

La declaración del cursor es el único paso que se lleva a cabo en la sección declarativa de un bloque.

Declaración del cursor

Define su nombre y asocia el cursor con una orden SELECT. La sintaxis es:

CURSOR *nombre_cursor* IS *orden_SELECT;*

Una declaración de cursor puede hacer referencia a variables PL/SQL en la cláusula WHERE, cumpliéndose las reglas de ámbito y visibilidad conocidas.

Para asegurarse de que todas las variables referenciadas en una declaración de cursor son declaradas antes de la referencia, pueden declararse todos los cursores al final de la sección declarativa. El propio nombre del cursor puede emplearse en una referencia con el atributo %ROWTIPE. En este caso, el cursor debe ser declarado antes de hacer referencia a él.

Apertura de un cursor

La sintaxis para abrir un cursor es:

OPEN *nombre_cursor*;

Al abrir un cursor suceden tres cosas:

- Se examinan los valores de las variables acopladas.
- Se determina el conjunto activo.
- Se hace apuntar el puntero del conjunto activo a la primera fila.

Podemos reabrir un cursor que ya estaba abierto, PL/SQL ejecutará implícitamente una orden CLOSE antes de reabrirlo. También podemos tener varios cursores abiertos al mismo tiempo.

Extracción de los datos de un cursor

La cláusula INTO de la consulta se incluye en la orden FETCH. Esta orden tiene dos formas posibles:

FETCH *nombre_cursor* INTO *lista_variables;*

FETCH *nombre_cursor* INTO *registro;*

Después de cada FETCH, se incrementa el puntero del conjunto activo para que apunte a la siguiente fila. De esta forma con un bucle cada FETH devolverá filas sucesivas del conjunto activo.

El atributo de cursores %NOTFOUND se utiliza para determinar cuando se ha terminado de extraer todo el conjunto activo.

Cierre de un cursor

Cuando se ha terminado de extraer el conjunto activo debe cerrarse el cursor. La sintaxis para el cierre de un cursor es:

CLOSE *nombre_cursor*;

Atributos de los cursores

Existen cuatro atributos en PL/SQL que pueden aplicarse a los cursores para obtener valores sobre ellos. Los atributos son %FOUND, %NOTFOUND, %ISOPEN, y %ROWCOUNT. Los atributos se unen al nombre del cursor y el resultado que devuelven es el siguiente:

%FOUND es un atributo booleano, devuelve TRUE si la última orden FETCH devolvió una fila y FALSE en caso contrario.

%NOTFOUND se comporta de forma opuesta a %FOUND. Este atributo se utiliza comúnmente como condición de salida para un bucle de extracción.

%ISOPEN nos indica si el cursor está o no abierto**.**

%ROWCOUNT Este atributo numérico devuelve el número de filas extraídas por el cursor hasta ese momento.

Cursores parametrizados

Permiten utilizar la orden OPEN para enviar las variables de acoplamiento en un cursor:

```
DECLARE

  /* Declaración del cursor */
  CURSOR c_Students (v_Major students.major%TYPE)IS
    SELECT id, first_name, last_name
      FROM students
      WHERE major = v_Major;
BEGIN
  /* Utilizamos la orden OPEN para enviar el valor de
      v_Major */

  OPEN c_Students('Computer Science');
```

Cursores Implícitos

Los cursores explícitos sirven para procesar órdenes SELECT que devuelven más de una fila. Sin embargo, todas las órdenes SQL se ejecutan dentro de un área de contexto y tienen, por tanto, un cursor asociado. Este cursor se conoce con el nombre de cursor SQL y sirve para procesar las órdenes INSERT, UPDATE, DELETE, y las órdenes SLECT-INTO de una sola fila. Éste se abre y cierra automáticamente, pero podemos acceder a sus atributos.

Se muestran dos ejemplos de su uso en los que se ejecuta una orden INSERT si la orden UPDATE no encuentra ninguna fila coincidente:

```
BEGIN
  UPDATE rooms
    SET number_seats = 100
    WHERE room_id = 99980;
  /*Si la anterior orden UPDATE
no se aplica a ninguna fila, inserta
una nueva fila en la tabla*/
  IF SQL%NOTFOUND THEN
    INSERT INTO rooms (room_id,
number_seats)
      VALUES (99980, 100);
  END IF;
END;
/
```

```
BEGIN
  UPDATE rooms
    SET number_seats = 100
    WHERE room_id = 99980;
 /*el mismo ejemplo pero consultando
%ROWCOUNT */
  IF SQL%ROWCOUNT = 0 THEN
    INSERT INTO rooms (room_id,
number_seats)
      VALUES (99980, 100);
  END IF;
END;
/
```

Aunque se puede emplear SQL%NOTFOUND con las órdenes SELECT-INTO, realmente no resulta útil porque se producirá un error Oracle cuando no encuentre ninguna fila coincidente.

Bucles de cursor FOR

Podemos crear bucles de extracción mediante cursor con LOOP-END LOOP como se mostraba en el ejemplo inicial o también con WHILE, pero requieren un procesamiento explícito de las órdenes OPEN,

FETCH y CLOSE. PL/SQL proporciona un bucle más simple que realiza un procesamiento implícito del cursor. Este bucle se llama ***Bucle de cursor FOR.***

Veamos su uso con un ejemplo:

```
DECLARE
  -- Cursor que recupera la información de los estudiantes de Historia
  CURSOR c_HistoryStudents IS
    SELECT id, first_name, last_name
      FROM students
      WHERE major = 'Historia';
BEGIN
  -- un OPEN de c_HistoryStudents se ejecuta implicitamente al comienzo
  -- del bucle
    FOR v_StudentData IN c_HistoryStudents LOOP
    -- un FETCH implicito se ejecuta aquí.

    -- Procesa las filas recuperadas. En este caso matrirula
    -- a cada estudiente en HIS 301, insertándolo en la
    -- tabla registered_students. Guarda también el nombre y
    -- el apellido en la tabla temp_talbe
    INSERT INTO registered_students (student_id, department, course)
      VALUES (v_StudentData.ID, 'HIS', 301);

    INSERT INTO temp_table (num_col, char_col)
      VALUES (v_StudentData.ID,
              v_StudentData.first_name || ' ' || v_StudentData.last_name);

    -- Antes de hacer otro ciclo aquí se hace una comprovbación
    -- implicita de c_HistoryStudents%NOTFOUND
  END LOOP;
  -- cuando el bucle termina se hace un CLOSE implicito

  -- confirma el trabajo.
  COMMIT;
END;
/
```

.

- El registro utilizado lo declara implícitamente PL/SQL como ***nombre_curosor*%ROWTYPE** y no es necesario declararlo en sección declarativa del bloque.

Cursores SELECT FOR UPDATE

Es habitual que el procesamiento que se lleve a cabo en un bucle de extracción modifique las filas extraídas por el cursor. PL/SQL proporciona una sintaxis específica para estos casos. El método consta de dos partes: la cláusula FOR UPDATE en la declaración del cursor, y la cláusula WHERE CURRENT OF en la orden UPDATE o DELETE.

Si se declara el cursor con la cláusula FOR UPDATE, puede emplearse la cláusula WHERE CURRENT OF en una orden UPDATE o DELETE.

Ejemplo de uso:

```
DECLARE
  -- Céditos a añadir al total de cada estudiante
  v_NumCredits  classes.num_credits%TYPE;

  -- El cursos selecciona los alumnos matriculados en HIS 101
  CURSOR c_RegisteredStudents IS
    SELECT *
      FROM students
      WHERE id IN (SELECT student_id

                     FROM registered_students
                     WHERE department= 'HIS'
                     AND course = 101)
      FOR UPDATE OF current_credits;
```

```
BEGIN
  -- Crea el bucle de extracción
  FOR v_StudentInfo IN c_RegisteredStudents LOOP
  -- obtiene el número de créditos HIS 101.
  SELECT num_credits
    INTO v_NumCredits
    FROM classes
    WHERE department = 'HIS'
    AND course = 101;

  -- Actualiza la fila que acaba de recuperar con el cursor.
  UPDATE students
    SET current_credits = current_credits + v_NumCredits
    WHERE CURRENT OF c_RegisteredStudents;
  END LOOP;

  -- Confirma el trabajo.
  COMMIT;
END;
/
```