Universidad Nacional del Litoral

Facultad de Ingeniería y Ciencias Hídricas

Departamento de Informática

Bases de Datos

SQL: Guía de Trabajo Nro. 5

Cursores y loops

Msc. Lic. Hugo Minni

2015

1. Cursores y loops

SQL fue diseñado como un lenguaje orientado a conjuntos. Puede suceder a veces que la operación a realizar sea tan compleja que no la podamos resolver a través de este enfoque de conjuntos y necesitemos recorrer los datos secuencialmente, fila a fila.

Al permitirnos recorrer los datos fila a fila, podemos entonces detenernos en cada una, realizar tal vez varias operaciones, y luego movernos a la próxima.

En esta Guía de Trabajo veremos qué enfoques nos proporcionan T-SQL y PL/pgSQL para resolver este tipo de problemas.

Ejemplo 1

Demostraremos el recorrido fila a fila con un ejemplo simple. Supongamos que necesitamos obtener el máximo valor de la columna `price` en la tabla `titles` (algo que podríamos obtener directamente con `SELECT MAX(price) FROM titles`, pero vale como ejemplo).

Vamos a recorrer fila a fila la tabla `titles` e iremos actualizando una variable con el valor máximo obtenido. Al final del recorrido, tendremos el máximo:

La siguiente es la solución T-SQL del Ejemplo 1. Necesitamos usar un **cursor T-SQL:**

```
DECLARE curPrecios CURSOR                    (A)
   FOR
      SELECT Price                           (B)
         From Titles

Declare @price money,
        @priceMaximo FLOAT;

SET @priceMaximo = 0;

OPEN curPrecios                              (C)

FETCH NEXT
   FROM curPrecios                           (D)
   INTO @price

                                             (E)
WHILE @@fetch_status = 0                     (F)
   BEGIN
      IF @price IS NOT NULL
         IF @price > @priceMaximo
            SET @priceMaximo = @price;
         --END IF;
      -- END IF;

      FETCH NEXT
         FROM curPrecios                     (G)
         INTO @price

   END
-- END WHILE

CLOSE curPrecios                             (H)

DEALLOCATE curPrecios                        (I)

SELECT @priceMaximo
--22.95
```

A es la declaración del cursor.

B. `FOR <sentencia-select>` especifica el conjunto de datos vamos a recuperar para recorrer.

C. `OPEN <nombre-cursor>` abre el cursor. En este punto el DBMS ejecuta la sentencia `Select` especificada en **B** y ya conoce la cantidad de filas recuperadas.

D es la operación `FETCH`. Equivale a posicionarse en **la próxima tupla recuperada** (en este caso será la primera) y "bajar" los valores de sus componentes a variables locales. Para ello tendremos que tener definidas previamente una variable local por cada componente recuperado por la sentencia `SELECT` de **B**. Si tenemos más de una variable local, se separan con coma. Por ejemplo: `INTO @price, @pubdate`

E. La variable del sistema `@@fetch_status` nos proporciona el código de status de la operación `FETCH`. Su valor cambia por supuesto con cada `FETCH`.
Un valor 0 indica que el `FETCH` fue exitoso. Veremos los demás valores más adelante.

F es el procesamiento fila a fila. Finaliza cuando `FETCH` no encuentra más filas (`@@fetch_status` tendrá un valor diferente de cero). Antes de finalizar el bucle `WHILE` tenemos que hacer por supuesto un nuevo `FETCH` (**G**).

Una vez finalizado el trabajo tenemos que hacer el "cleanup". T-SQL distingue entre cerrar el cursor y elimnarlo de memoria. Son las operaciones `CLOSE` (**H**) y `DEALLOCATE` (**I**)

Aquí estamos creando un `CURSOR` en un batch. Por supuesto también lo podemos crear en un procedimiento almacenado T-SQL.

La siguiente es la solución con cursores PL/pgSQL del Ejemplo 1:

```
CREATE FUNCTION test()
   RETURNS FLOAT
   LANGUAGE plpgsql
   AS
   $$
   DECLARE
      vPrice FLOAT;
      vPriceMaximo FLOAT;
      cursorPrice CURSOR FOR Select price
                             From Titles;        (A)
   BEGIN
      vPriceMaximo := 0;
      OPEN cursorPrice;                          (B)

   (C) LOOP
         FETCH NEXT FROM cursorPrice INTO vPrice;   (D)
         EXIT WHEN NOT FOUND;                       (E)

         IF vPrice IS NOT NULL THEN
            IF vPrice > vPriceMaximo THEN
               vPriceMaximo := vPrice;
            END IF;
         END IF;

      END LOOP;
      CLOSE cursorPrice;                         (F)
      RETURN vPriceMaximo;

   END
   $$;
```

(**A**) es la declaración del cursor.

En (**B**) abrimos el cursor.

Para iterar sobre el cursor usamos una construcción `LOOP` como la que vimos en La Guía de Trabajo Nro. 4, sección 13.1 (**C**).

En (**D**) realizamos la operación `FETCH`.

(**E**) Cuando en un `LOOP` estamos trabajando con cursores –o hemos disparado una sentencia SQL- podemos usar el valor booleano `FOUND` o `NOT FOUND` para que la cláusula `WHEN` evalúe la continuidad o salida del bucle.
También podríamos escribir:

```
IF NOT FOUND THEN
   EXIT
END IF;
```

En (**F**) cerramos el cursor.

Recordemos que en la Guía de Trabajo Nro. 4 –*Sección 13.3 – Otros iteradores*- vimos una construcción `FOR` con la siguiente sintaxis:

```
suma:=0;
FOR i IN 1..10 LOOP
   suma := suma + i;
END LOOP;
```

Esta construcción `FOR` en realidad tiene un soporte más extendido, de la forma:

```
FOR <target> IN <query> LOOP
   -- procesar...
END LOOP
```

A esta forma de `LOOP` se le llama "cursor implícito". El bucle finaliza automáticamente cuando no se encuentran más filas en `<query>`.

La siguiente es la solución PL/pgSQL del Ejemplo 1. Usamos un bucle `FOR..IN..LOOP`:

```
CREATE FUNCTION test700()
   RETURNS FLOAT
   LANGUAGE plpgsql
   AS
   $$
   DECLARE
      vPrice FLOAT;
      vPriceMaximo FLOAT;
   BEGIN
    vPriceMaximo := 0;
          A              B
    FOR vPrice IN Select price
                  From Titles LOOP

       IF vPrice IS NOT NULL THEN
          IF vPrice > vPriceMaximo THEN
             vPriceMaximo := vPrice;
          END IF;
       END IF;

    END LOOP;

    RETURN vPriceMaximo;
   END
   $$;

SELECT * from test700();
-- 16,73
```

A es el `<target>`. En este caso, una variable escalar.
B es el `<query>`, que debe retornar por supuesto un componente del mismo tipo de la variable escalar.

2. Loops y tipos compuestos

Loops y composite types

En PL/pgSQL tenemos la posibilidad de usar composite types en un bucle `FOR..IN.. LOOP`. En el ejemplo 1 estamos recuperando un solo componente, pero podría darse el caso de que necesitemos muchos componentes y deberíamos declarar una variable local para cada uno.

2.1. Tipo de datos para la tupla completa

<tabla>%rowtype datatype

El tipo de dato `<tabla>%rowtype` define una tupla con el schema de `<tabla>`.

La siguiente es la solución del Ejemplo 1 usando una variable de tipo `<tabla>%rowtype`:

```
CREATE FUNCTION test702()
   RETURNS FLOAT
   LANGUAGE plpgsql
   AS
   $$
   DECLARE
      tuplaTitles titles%rowtype;
      vPriceMaximo FLOAT;
   BEGIN
    vPriceMaximo := 0;
    FOR tuplaTitles IN Select *
                       From Titles LOOP
       IF tuplaTitles.price IS NOT NULL THEN
          IF tuplaTitles.price > vPriceMaximo THEN
             vPriceMaximo := tuplaTitles.price;
          END IF;
       END IF;
    END LOOP;
    RETURN vPriceMaximo;
   END
   $$;
```

2.2. Tipo de datos para composite types específicos

En la Guía de Trabajo Nro. 4 –*Secciónes 14.3 – Retornando sets de proyecciones de tuplas* y *14.4 – Retornar composite types*- vimos como definir **composite types específicos** que se adapten a subconjuntos de tuplas o tuplas completamente nuevas.
La siguiente es la solución del Ejemplo 1 usando un composite type específico:

```
CREATE TYPE titlesCT
   AS (
       title_id CHAR(6),
       price numeric
      );

CREATE FUNCTION test703()
   RETURNS FLOAT
   LANGUAGE plpgsql
   AS
   $$
   DECLARE
      tuplaTitlesCT titlesCT%rowtype;
      vPriceMaximo FLOAT;
   BEGIN
    vPriceMaximo := 0;
    FOR tuplaTitlesCT IN Select title_id, price
                         From Titles LOOP
       IF tuplaTitlesCT.price IS NOT NULL THEN
          IF tuplaTitlesCT.price > vPriceMaximo THEN
             vPriceMaximo := tuplaTitlesCT.price;
          END IF;
       END IF;
    END LOOP;
    RETURN vPriceMaximo;
   END
   $$;
```

2.3. Tipo de dato para estructuras flexibles

RECORD datatype

El tipo de dato `RECORD` no posee un formato de tupla determinado. Adopta el "schema" de la tupla en el momento de la asignación.

En el caso del bucle `FOR..IN..<query>` la operación fetch implícita define la estructura del `RECORD`.

La siguiente es la solución del **Ejemplo 1** usando una variable de tipo `RECORD`:

```
CREATE FUNCTION test701()
   RETURNS FLOAT
   LANGUAGE plpgsql
   AS
   $$
   DECLARE
      recTupla RECORD;
      vPriceMaximo FLOAT;
   BEGIN
    vPriceMaximo := 0;
    FOR recTupla IN Select price
                    From Titles LOOP
       IF recTupla.price IS NOT NULL THEN
          IF recTupla.price > vPriceMaximo THEN
             vPriceMaximo := recTupla.price;
          END IF;
       END IF;
    END LOOP;
    RETURN vPriceMaximo;
   END
   $$;
```

3. Cursores for update

En general los cursores pueden ser utilizados para realizar modificaciones sobre las tablas. Es decir, nos permiten realizar operaciones `UPDATE` o `DELETE` sobre tuplas recuperadas por la operación `FETCH`.
Esto es posible ya que un cursor es una estructura compleja que mantiene todo el tiempo un enlace entre la tupla recuperada por una operación `FETCH` y los datos físicos subyacentes en la base de datos.
De esta manera, las operaciones de modificación que realicemos sobre la tupla actualmente en `FETCH` puede ser trasladada a la tabla física subyacente.

3.1. La cláusula CURRENT OF

Para soportar la modificación o eliminación de filas en un cursor, tanto las sentencias `UPDATE` como `DELETE` soportan una sintaxis especial de la cláusula `WHERE` que indica que **el contexto del query se reduce a la fila correspondiente al último FETCH realizado**.

Declaración de Cursores for update

Declaramos un cursor for update con la siguiente sintaxis:

```
DECLARE curPrecios CURSOR
   FOR
      SELECT Price
         From Titles
   FOR UPDATE
```

También podemos restringir las columnas que pueden soportar update:

```
DECLARE curPrecios CURSOR
   FOR
      SELECT Price
         From Titles
   FOR UPDATE OF price
```

(A)

En este caso, solo estamos permitiendo la modificación de la columna `price` (**A**)

Ejemplo 2

Se necesita modificar el título de las publicaciones que poseen un título que comienza con la string `'The gourmet'` por ese título concatenado con la string `' Second Edition'`.

La siguiente es la solución T-SQL del **Ejemplo 2**:

```
SELECT title
   FROM titles
   WHERE title LIKE 'The gourmet%'
-- The Gourmet Microwave


Declare curTitles Cursor
   For
      Select title
         From Titles
   FOR UPDATE

Declare @title VARCHAR(255)


OPEN curTitles
FETCH NEXT
   FROM curTitles
   INTO @title

WHILE @@fetch_status = 0
   BEGIN
      IF @title LIKE 'The gourmet%'
         UPDATE titles
            SET title = title + ' Second Edition'
            WHERE CURRENT OF curTitles
      --END IF;

      FETCH NEXT
         FROM curTitles
         INTO @title

   END
-- END WHILE

CLOSE curTitles
DEALLOCATE curTitles
```

La siguiente es la solución del **Ejemplo 2** usando PL/pgSQL:

```
SELECT title
   FROM titles2
   WHERE title LIKE 'The Gourmet%'
-- The Gourmet Microwave


CREATE FUNCTION test()
   RETURNS VOID
   LANGUAGE plpgsql
   AS
   $$
   DECLARE
      vTitle VARCHAR(255);
      curTitles CURSOR FOR Select title
                           From Titles2;

   BEGIN
      OPEN curTitles;

      LOOP
         FETCH NEXT FROM curTitles INTO vTitle;
         EXIT WHEN NOT FOUND;

         IF vTitle LIKE 'The Gourmet%' THEN
            UPDATE titles2
               SET title = title || ' Second Edition'
               WHERE CURRENT OF curTitles;
         END IF;

      END LOOP;
      CLOSE curTitles;
      RETURN;

   END
   $$;
```

4. SCROLL CURSORS

Los cursores que hemos visto hasta ahora son "forward only". Solo hemos ejecutado `FETCH NEXT` para obtener la próxima tupla del conjunto resultado.
T-SQL nos permite crear cursores que permiten hacer un "scroll" hacia adelante y hacia atrás. Estos cursores son mucho más costosos en recursos y deberían utilizase solo si no hay otra alternativa.

4.1. Operaciones FETCH

Si el cursor es de tipo `SCROLL`, tenemos la posibilidad de realizar los siguientes `FETCH` adicionales:
`FETCH PRIOR` (para ir a la tupla anterior a la actual), `FETCH FIRST` (0ara ir a la primera tupla), `FETCH LAST` (para ir a la última tupla).

Declaración de un scroll cursor

Declaramos un scroll cursor con la siguiente sintaxis:

```
DECLARE curPrecios CURSOR
   SCROLL
   FOR
      SELECT Price From Titles
```

Declaración de un scroll cursor

Declaramos un scroll cursor con la sintaxis:

```
DECLARE curPrecios SCROLL CURSOR
   FOR
      SELECT Price From Titles
```

Ejercicio 1.

Implemente un batch T-SQL que actualice los precios de las publicaciones de la editorial '0736'.
Por cada publicación de esta editorial, se desea incrementar en un 25% el precio de las publicaciones que cuestan $10 o menos y decrementar también en un 25% las publicaciones que cuestan más de $10.

Ejercicio 2.

Resuelva el Ejercicio 1 utilizando loops PL/pgSQL.

Ejercicio 3

En T-SQL, Obtenga un listado con las tres publicaciones más caras de cada tipo (columna `type`).

```
Publicaciones más caras de tipo business

----------------------
2,99
11,95
19,99

Publicaciones más caras de tipo mod_cook

----------------------
NULL
2,99
19,99

Publicaciones más caras de tipo popular_comp

----------------------
NULL
20,00
22,95

Publicaciones más caras de tipo psychology

----------------------
10,95
19,99
21,59

Publicaciones más caras de tipo trad_cook

----------------------
11,95
14,99
20,95

Publicaciones más caras de tipo UNDECIDED

----------------------
NULL
```

Ejercicio 4

Usando PL/pgSQL, obtenga un listado como el siguiente para los autores que viven en ciudades donde se ubican las editoriales que publican sus libros.

```
El autor: Carson reside en la misma ciudad que la editorial que lo edita
El autor: Bennet reside en la misma ciudad que la editorial que lo edita
```

Nota: existen soluciones sin loops/cursores para este problema. Resuelva el ejercicio usando loops/cursores.

Ejercicio 5

En la tabla `Employee` hay varios empleados que son editores (columna `job_id` con valor `5`):

```
Select * from employee where job_id = 5
--PXH22250M de 0877
--CFH28514M de 9999
--JYL26161F de 9901
--LAL21447M de 0736
--RBM23061F de 1622
--SKO22412M de 1389
--MJP25939M de 1756
```

Se deben analizar los empleados con `job_id 5` y, de los que pertenezcan a las dos editoriales que menos han facturado (en dinero) a lo largo del tiempo, se debe seleccionar el más antiguo (columna `hire_date`).
Este empleado debe pasar a formar parte de la editorial que más ha facturado (en dinero) a lo largo del tiempo.

Por ejemplo, la editorial que más ha vendido es la `'1389'`.

Las dos que menos han vendido son las `'0736'` y `'0877'`

Terminan siendo evaluados dos empleados:

```
-- PXH22250M de editorial 0877 contratado el 1993-08-19 00:00:00.000
-- LAL21447M de editorial 0736 contratado el 1990-06-03 00:00:00.000
```

El más antiguo es el empleado `LAL21447M`, contratado en `1990`. Este empleado debe pasara trabajar en la editorial `'1389'`.
Resuélva el ejercicio utilizando T-SQL

Nota: existen soluciones sin cursores para este problema. Resuelva el Ejercicio con algún uso de los mismos.

Ejercicio 6

Resuelva el ejercicio 5 utilizando PL/pgSQL.