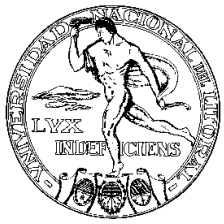


Universidad Nacional del Litoral

Facultad de Ingeniería y Ciencias Hídricas

Departamento de Informática

Bases de Datos

SQL: Guía de Trabajo Nro. 5

Cursores y loops

Parte 1

2023

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 eliminarlo 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.

En PL/pgSQL

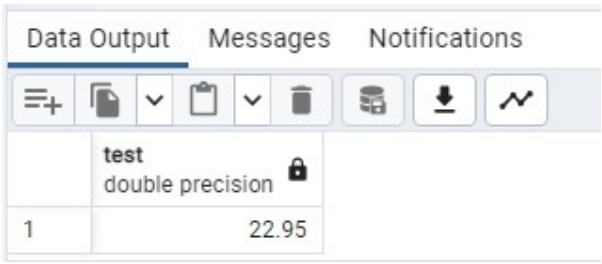
La siguiente es la solución PL/pgSQL del Ejemplo 1:

```
CREATE OR REPLACE FUNCTION test
    ()
    RETURNS FLOAT
    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;
    $$
    LANGUAGE plpgsql


SELECT test ();
```



(**A**) es la declaración del cursor.

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

Para iterar sobre el cursor usamos una construcción `LOOP` (**C**).

Ver *Loop en Guía de Trabajo Nro. 4*.

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.

Vimos la variable `FOUND` en la sección *2.2. Sentencias SELECT Single-row en Guía de Trabajo Nro. 4 - Parte 2*.
`FOUND` es una variable especial que nos indica si un sentencia SQL encontró o no tuplas como resultado de su ejecución

También podríamos escribir:

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

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

También en la *Guía de Trabajo Nro. 4* - 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>`.

Cursor explícito

Generalmente los lenguajes de bases de datos llaman CURSOR EXPLÍCITO al cursor como el del Ejemplo 1: un cursor que se declara, posee una sentencia `OPEN`, posee varias sentencias `FETCH` y una sentencia `CLOSE`.

Como contrapartida, un cursor que se crea con una sentencia `FOR <target> IN <query> LOOP` generalmente se denomina cursor implícito.

Sin embargo, la interpretación de cursor implícito varía entre fabricantes. Para Oracle, por ejemplo, toda sentencia `SELECT...INTO` crea un cursor implícito.

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

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.

```
CREATE OR REPLACE FUNCTION test700
    ()
    RETURNS FLOAT
    AS
    $$
    DECLARE
       vPrice FLOAT;
       vPriceMaximo FLOAT;
    BEGIN
       vPriceMaximo := 0;

       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;
    $$
    LANGUAGE plpgsql


SELECT test700 ();
```

B

A

2. Loops y estructuras de datos para sentencias SELECT single-row

Como en cada iteración del Loop estaremos recuperando los datos de una única fila, podemos valernos de las declaraciones ancladas que vimos en la Guía de Trabajo Nro. 4, en la sección *Sentencias SELECT Single-Row*.

Como el cursor recupera una única fila por vez, el comportamiento es análogo al de un query cualquiera que retorna una única tupla:

2.1. Cursor que retorna una tupla completa

Record anchoring
Table-based record

Si el cursor recupera una fila completa, podemos usar un Table-based record como el que vimos en la Sección *Sentencias SELECT Single-row que retornan una tupla completa* de la Guía de Trabajo Nro. 4:

```
DECLARE
   titleRec titles%ROWTYPE;
```

La siguiente es la solución del Ejemplo 1 usando table-based anchoring:

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

2.2. Cursor que retorna una Projection de una tupla

Composite-type

Si el cursor recupera una Projections de una tupol, podemos definir un composite-type, como aprendimos en la Sección *2.4 Sentencias SELECT Single-row que retornan una Projection de una tupla* de la *Guía de Trabajo Nro. 4 - parte 2*, y luego declarar una variable anclada a ese composite type:

```
CREATE TYPE titleCT
   AS (
       title_id char(6),
       price numeric
      );

DECLARE
   titleRec titleCT%ROWTYPE;
```

La siguiente es la solución del Ejemplo 1 usando una variable anclada a un composite type:

```
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. Records

RECORD datatype

Sin embargo la solución más práctica es usar un tipo de dato **RECORD datatype** como el que aprendimos en la Sección *2.5 Records* de la *Guía de Trabajo Nro. 4 - parte 2.*

Recordemos que el tipo de dato `RECORD` no posee un formato de tupla determinado. Adopta la forma 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
   $$
   DECLARE
      recTupla RECORD;
      vPriceMaximo FLOAT;
   BEGIN
    vPriceMaximo := 0;
    FOR recTupla IN Select price
                    From Titles LOOP
       IF recTupla.price IS NOT NULL THEN
          IF recTupla.price > vPriceMaximo THEN
             vPriceMaximo := recTupla.price;
          END IF;
       END IF;
    END LOOP;
    RETURN vPriceMaximo;
   END
   $$;
```

También podríamos haber podido usar un tipo de dato `RECORD` en el Ejemplo 1:

```
CREATE FUNCTION test500()
   RETURNS FLOAT
   LANGUAGE plpgsql
   AS
   $$
   DECLARE
      recTitles RECORD;
      vPriceMaximo FLOAT;
      cursorPrice CURSOR FOR Select price
                             From Titles;
   BEGIN
      vPriceMaximo := 0;
      OPEN cursorPrice;

      LOOP
         FETCH NEXT FROM cursorPrice INTO recTitles;
         EXIT WHEN NOT FOUND;

         IF recTitles IS NOT NULL THEN
            IF recTitles.price > vPriceMaximo THEN
               vPriceMaximo := recTitles.price;
            END IF;
         END IF;

      END LOOP;
      CLOSE cursorPrice;
      RETURN vPriceMaximo;

   END
   $$;

SELECT * from test500()
```

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
```

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
```