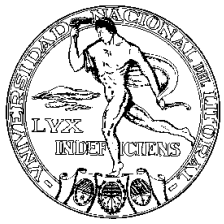


Universidad Nacional del Litoral

Facultad de Ingeniería y Ciencias Hídricas

Departamento de Informática

Bases de Datos

SQL: Guía de Trabajo Nro. 4

Cursores y loops

Parte 2: Ejercicios

Msc. Lic. Hugo Minni

2020

Ejercicios

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.

Guía para resolver el ejercicio

Trabajamos con la tabla *Titles*

Un cursor siempre va atado a una sentencia `SELECT`. Mientras el cursor esté abierto, la sentencia `SELECT` probablemente esté bloqueando filas en la base de datos, máxime si se trata de un cursor sobre el que vamos a actualizar datos.

Si el cursor está bloqueando filas, muchos otros usuarios no podrán acceder a esas filas hasta que nuestra sesión finalice.

Por eso, como regla general:

A) El cursor debe mantenerse abierto el menor tiempo posible.

B) Debemos ser económicos en que datos recuperamos en el cursor. Nuestro `SELECT` debe recuperar la menor cantidad de datos necesarios para realizar la tarea.
Y hablamos no solo de la menor cantidad de filas posible sino también de la menor cantidad de columnas posibles.

Un cursor que esté asociado a un `SELECT` * seguramente estará mal diseñado.

En nuestro caso, debemos evaluar precios. Entonces... ¿necesitamos otro dato además del precio de la publicación?.

Nos piden analizar las publicaciones de la editorial `'0736'`. Entonces, ¿será correcto un `SELECT` que traiga filas de otras editoriales?

Finalmente, necesitamos actualizar el precio de determinadas publicaciones. Esto significa que:

A) El cursor debe ser declarado para actualizaciones.

B) Podemos aprovechar la característica de los cursores de hacer una actualización **para la fila actual** (`CURRENT OF`).

Ejercicio 2.

Resuelva el Ejercicio 1 utilizando la estructura `FOR <target> IN <query> LOOP`.

Guía para resolver el ejercicio

Para recibir la tupla en cada iteración podemos usar un tipo de dato `RECORD`.

Al usar un cursor implícito no podremos aprovechar la característica de hacer una actualización para la fila actual.

La consecuencia directa de esto es que necesitamos especificar una condición para el `UPDATE`. Esa condición debería estar dada por la Primary Key de la tabla.

Por lo tanto nuestro cursor necesita recuperar más columnas que el que creamos en T-SQL en el Ejercicio 1.

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
```

Guía para resolver el ejercicio

Por cada tipo de publicación debemos analizar los precios de sus publicaciones.
Los tipos de publicación se repiten, así que debemos tener en cuenta recuperar **una vez** cada tipo de publicación.

Para obtener las tres publicaciones más caras hay varias alternativas.

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.

Guía para resolver el ejercicio

Nuevamente debemos evaluar qué columnas son indispensables para el `SELECT` asociado al cursor.

Seguramente por cada autor deberemos hacer la evaluación del lugar donde vive.

Respecto a la salida o resultado de la función, debemos tener en cuenta que PL/pgSQL no nos permite retornar un mensaje en una expresión `SELECT` como sí lo permite T-SQL.
Por lo tanto, tendremos que buscar la forma de retornar todos los mensajes:

```
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
```
etc.

...como salida de la función.

Una alternativa es definir un composite-type. Por ejemplo:

```
CREATE TYPE Mensaje
   AS (
       columna1 VARCHAR(255)
      );
```

...y para cada autor que cumpla con lo especificado en el enunciado "armar" sobre `columna1` una cadena con la forma `'El autor: '  <autor>  'reside en la misma ciudad que la editorial que lo edita '`

y retornar inmediatamente esta cadena utilizando la forma que aprendimos en la *Guía de trabajo Nro. 2 - Parte 2*, en la sección *5.5. Return Next*.

`RETURN NEXT` es la forma de salida que debemos usar cuando lo que queremos retornar NO ES resultado de un query, sino que se trata de algo que estamos armando por nosotros mismos, como es el caso en este ejercicio.

Recordemos que debemos definir la función para que retorne un **setof** del composite type.

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:

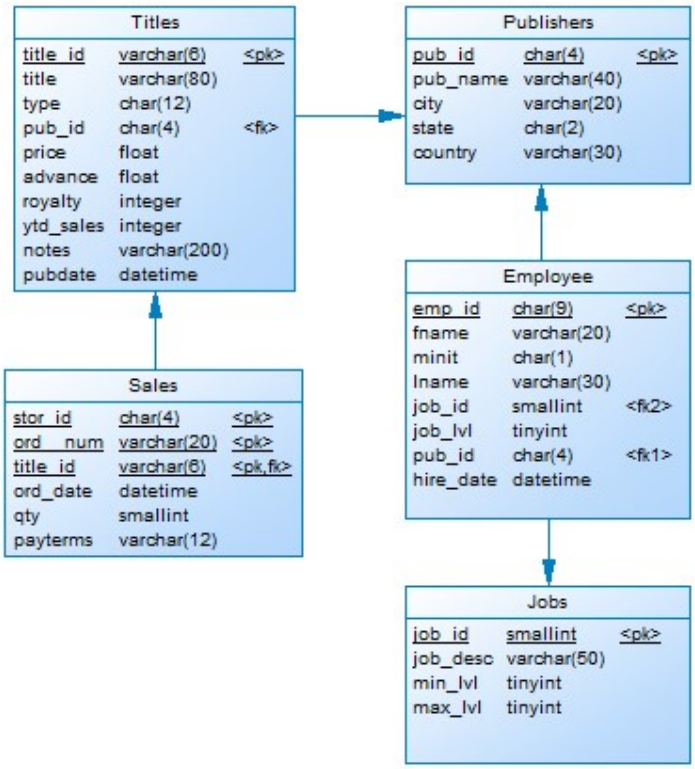
```
-- PXH22250M de editorial 0877 contratado el 1993-08-19 00:00:00.000
-- LAL21447M de editorial 0736 contratado el 1990-06-03 00:00:00.000
```

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.

Guía para resolver el ejercicio

Trabajamos con las siguientes tablas:



...ya que necesitamos *Employee* y *Publishers* para analizar qué empleado trabaja en qué editorial, con qué *job_id* y con qué antigüedad (basada en su columna *hire_date*).

Pero también necesitamos conocer las editoriales que más y menos han facturado a lo largo del tiempo. Esto lo debemos calcular a partir de sus ventas y de los precios de las publicaciones (consideraremos los precios actuales de la publicaciones, -columna *price* de la tabla *Titles*-). Por lo tanto debemos hacer un análisis también sobre *Sales* y *Titles*.

Por un lado necesitamos conocer y guardar la editorial que más ha facturado a lo largo del tiempo.

Por otro lado necesitamos conocer y guardar las *dos* editoriales que menos han facturado a lo largo del tiempo.

Luego necesitaremos realizar un procesamiento empleado por empleado. Esto implica un cursor.

Recordemos lo que dijimos al principio de la Guía:

Debemos ser económicos en que datos recuperamos en el cursor. Nuestro `SELECT` debe recuperar **la menor cantidad de datos necesarios para realizar la tarea**.

Entonces, ¿es una buena idea recuperar todos los empleados y analizar su situación?. No, ya que en una base de datos real estaremos bloqueando esas filas a potenciales usuarios concurrentes.

En las bases de datos en producción habrá uno o más DBAs (Database Administrators) que estarán monitoreando las curvas de consumo de recursos en el servidor de bases de datos. Si disparamos una consulta excesivamente pesada probablemente recibamos una advertencia.

Necesitamos basar el cursor en una sentencia `SELECT` óptima que recupere los empleados con el job_id especificado que **pertenezcan** a las dos editoriales que menos han facturado a lo largo del tiempo. Ese sería un cursor óptimo, con el mínimo de tuplas recuperadas.

Debemos evaluar también qué columnas recuperar. ¿Cuáles son imprescindibles?.

Queda pendiente -sobre los empleados que resultan "pre-seleccionados"- evaluar su antiguedad.
Para esto simplemente podemos usar la estrategia básica de guardar la fecha actual, por ejemplo, y comparar las fechas de contratación contra ella, e ir guardando la más antigua.

Finalmente, el empleado que cumpla con la condición de pertenecer a una de las editoriales que menos han facturado y que a su vez sea el empleado más antiguo de los "pre-seleccionados", debe cambiar de editorial.

Aquí otra salvedad. En nuestro trabajo de todos los días no es buena idea realizar una operación tan delicada como un `UPDATE` "a ciegas" dentro de un cursor.
Una buena idea es ir mostrando mensajes ilustrativos de los que el cursor va haciendo.
Por ejemplo:

```
PRINT  'La editorial de mayor venta es ' + @pub_idMayorVenta;

PRINT 'Procesando Empleado: ' + @emp_id + ' de editorial ' + @pub_id

PRINT 'Termino el procesamiento. El empleado a modificar es ' + @min_emp_id +
'que pasa a la editorial ' + @pub_idMayorVenta
```

Para armar los mensajes necesitaremos seguramente recuperar un par más de columnas en el `SELECT` asociado al cursor, pero es un costo/beneficio que vale la pena pagar.

Sin embargo, aún así hay operaciones que son arriesgadas de realizar.
Una buena idea es utilizar el cursor como un "generador de código SQL". En nuestro caso, podemos decidir que el cursor **no ejecute** el `UPDATE`, sino que genere **la sentencia precisa que debemos ejecutar**.

Esta generación la hacemos simplemente armando una cadena y mostrándola en un mensaje con `PRINT`.