Universidad Nacional del Litoral

Facultad de Ingeniería y Ciencias Hídricas

Departamento de Informática

Bases de Datos

Guía de Trabajo Nro. 7

SQL: Triggers

Msc. Lic. Hugo Minni

2015

Triggers

Los triggers, también llamados *event-condition actions*, o *ECA rules*, son diferentes de otros tipos de constraints o procedimientos de SQL:

1. Se activan cuando ocurren ciertos **eventos** (**triggering events**), especificados por el desarrollador. Los tipos de eventos permitidos son normalmente insert, update o delete sobre una determinada tabla.

2. Una vez activado por su triggering event, el trigger evalúa una **Condition**. Si la condición no aplica, no sucede nada más.

3. Si la condición del trigger es satisfecha, el DBMS lleva a cabo la **Action** asociada al trigger.

Entonces tenemos las siguientes definiciones:

Eventos
Triggering event

Activan el trigger.
Los tipos de eventos son normalmente `INSERT`, `UPDATE` o `DELETE` sobre una determinada tabla.

En el caso particular de los eventos `UPDATE`, podemos especificar que los mismos deben estar limitados a un atributo (columna) o conjunto de atributos en particular.

Condition

Es una condición que se debe cumplir a fin de que el trigger lleve a cabo alguna acción. Si no se cumple, no sucede nada.
Esta condition es opcional.
Si no está presente, la **Action** es ejecutada cada vez que el trigger es activado. Si está presente, la action es ejecutada sólo si la Condition es verdadera.
Su función es básicamente evitar la ejecución innecesaria de un bloque de código.

Action

Es un bloque de código que puede modificar en alguna manera los efectos del evento. Inclusive puede abortar la transacción de la cual el evento forma parte.

No es necesario que este código esté vinculado necesariamente al evento que activó el trigger.
El código puede llevar a a cabo cualquier secuencia de operaciones de base de datos.

Ejemplo 1

Queremos crear un trigger que cree una copia de la tabla `Titles` por cada modificación que hagamos en el año 2015:

```
A  CREATE TRIGGER trTitles
       [BEFORE | AFTER] UPDATE ON Titles  B
C  WHEN (date_part('year', now()) = '2015')
          INSERT INTO auditoria SELECT * FROM titles;  D
```

A. La sentencia `CREATE TRIGGER` crea el trigger `trTitles`.

B. La cláusula que indica el **Triggering event:** En este caso, una sentencia `UPDATE` sobre la tabla `Titles`. Por ahora ignoraremos las cláusulas `BEFORE` y `AFTER`.

C. La **Condition** usa la keyword `WHEN` y una expresión booleana. En este caso estamos diciendo que sólo ejecutaremos la **Action** cuando el año actual sea `2015`.

D. La **Action**, consistente de una o más sentencias SQL. En este caso es una sentencia `INSERT` sobre una tabla de auditoría.
En esta caso tenemos como action una única sentencia SQL, pero podemos tener todos las sentencias que queramos, separadas por punto y coma y encerradas en un bloque `BEGIN...END`.

Action - Trigger function

PostgreSQL no sigue el estándar SQL en el sentido en que no permite la ejecución de una sentencia o grupo de sentencias SQL como Action del trigger.
En cambio, se debe crear una función PostgreSQL como las que vimos en la Guía de Trabajo Nro. 4, que, cuando es utilizada por un trigger, se denomina **trigger function**.

La trigger function es una función como cualquier otra, pero con las siguientes restricciones:

- Se debe declarar **sin parámetros**.
- Su cláusula `RETURNS` debe especificar `trigger`

La siguiente sería la solución PL/pgSQL del **Ejemplo 1**:

```
CREATE TRIGGER trTitles
   {BEFORE | AFTER}
   UPDATE
   ON Titles
   WHEN (date_part('year', now()) = '2015')
      EXECUTE PROCEDURE test200();
```

B **A**

En el Action del trigger, la trigger function (**A**) es precedida por la cláusula `EXECUTE PROCEDURE` (**B**).

Más de un triggering event

Si tenemos más de un triggering event, los mismos se separan con `OR`. Esta es una característica no estándar:

```
CREATE TRIGGER trTitles
   [BEFORE | AFTER] UPDATE OR INSERT ON Titles
    ...
```

Triggers T-SQL

T-SQL posee su propia forma de trabajar con triggers. La sintaxis apenas se puede relacionar con la definida por ANSI SQL.
La siguiente sería la solución T-SQL del Ejemplo 1:

```
CREATE TRIGGER trTitles
   ON Titles          (A)
   FOR UPDATE         (B)
(C)
   AS             (D)
      IF (YEAR(CURRENT_TIMESTAMP) = '2015')
         SELECT * INTO auditoria FROM titles      (E)
      RETURN
                 (F)
```

A. Debemos especificar antes que nada sobre que tabla se crea el trigger.

B. T-SQL incorpora la keyword `FOR` antes del nombre del o los triggering events. `FOR` se puede reemplazar por `AFTER`.
Si tenemos más de un triggering event, los mismos se separan con coma. Por ejemplo: `FOR UPDATE, INSERT, DELETE`

C. T-SQL no soporta la especificación de **Condition**.

D. La **Action** es precedida por la keyword `AS`.

E. Siguiendo el estándar, permite, en la **Action**, la ejecución directa de sentencias SQL.

La sentencia `RETURN` (**F**) no puede especificar valor de retorno.

1. El trigger y el estado de la base de datos

Estado de la base de datos

Llamamos estado de la base de datos al conjunto de las *instancias actuales* de todas las tablas de la base de datos.

La evaluación de la **Condition** y la ejecución de la **Action** del trigger puede tener efecto sobre el estado de la base de datos **anterior** a la ocurrencia del **triggering event** (`BEFORE`) o sobre el estado de la base de datos **posterior** a la ocurrencia del **triggering event** (`AFTER`).

BEFORE trigger
AFTER trigger

Se denomina **BEFORE TRIGGER** a un trigger que evalúa su **Condition** y ejecuta su **Action** sobre el estado de la base de datos previo a la ejecución del Triggering event.

Se denomina **AFTER TRIGGER** a un trigger que evalúa su **Condition** y ejecuta su **Action** sobre el estado de la base de datos posterior a la ejecución del Triggering event.

En nuestro ejemplo:

```
CREATE TRIGGER trTitles
A  AFTER UPDATE ON Titles
   WHEN (date_part('year', now()) = '2015')
      INSERT INTO auditoria SELECT * FROM titles;
```

A. Especificamos que el trigger utiliza el estado de la base de datos **posterior** al **triggering event**.

Triggers y el estado de la base de datos

En T-SQL **todos los triggers son AFTER triggers**. **T-SQL no soporta BEFORE triggers**. La keyword `FOR` ha sido sustituida por `AFTER` en las nuevas versiones de SQL Server a fines de que quede más claro que se trata de un AFTER trigger:

```
CREATE TRIGGER trTitles
   ON Titles
   AFTER UPDATE
   AS
      IF (YEAR(CURRENT_TIMESTAMP) = '2015')
         SELECT * INTO auditoria FROM titles
```

2. Granularidad del trigger

El programador tiene la opción de especificar que el trigger se ejecuta:

a) Una vez por cada tupla modificada

b) Una vez para todas las tuplas que han sido modificadas en la sentencia SQL. (recordemos que una sentencia de modificación SQL puede afectar muchas tuplas).

Entonces tenemos las siguientes definiciones:

row-level trigger

Es un trigger que se ejecuta una (1) vez por cada tupla modificada.

statement-level trigger

Es un trigger que se ejecuta una (1) vez no importa cuantas hayan sido las tuplas modificadas.

Por ejemplo, si actualizamos una tabla completa con una sentencia SQL update, un *statement-level update trigger* se ejecutaría sólo una vez, mientras que un *row-level trigger* se ejecutaría una vez por cada tupla a la cual es aplicado el update.

En el caso del **Ejemplo 1**:

```
CREATE TRIGGER trTitles
   AFTER UPDATE ON Titles
A  FOR EACH ROW
   WHEN (date_part('year', now()) = '2015')
      INSERT INTO auditoria SELECT * FROM titles;
```

A. La cláusula indica que el trigger se ejecuta una (1) vez por cada tupla modificada.

En T-SQL todos los triggers son statement-level triggers. **T-SQL no soporta row-level triggers**.

Los triggers PL/pgSQL son por omisión **statement-level triggers**.

3. Triggering events de tipo UPDATE

Cuando el Triggering event es de tipo `UPDATE`, podemos especificar una cláusula adicional (opcional) para acotar el o los atributos (columnas) que deben ser modificados para que el trigger sea activado.

Ejemplo 2

Supongamos que queremos impedir que se disminuyan los precios de las publicaciones. Podemos escribir la segunda línea como:

```
CREATE TRIGGER trTitles2
   AFTER UPDATE OF price ON Titles
   FOR EACH ROW
   WHEN (Condition)
      Action
```

Estamos especificando que el triggering event debe ser un update sólo de las columnas listadas luego de la keyword `OF`[1].

Si se debe especificar más de una columna, éstas se separan con coma.

[1] La cláusula `OF` obviamente no es válida para eventos `INSERT` o `DELETE`, ya que éstos eventos tienen sentido solo para tuplas completas.

T-SQL no soporta la cláusula `OF` para especificar el update de sólo una serie de atributos listados.

Vinculado a esta característica no proporcionada, en cambio, T-SQL permite que dentro del cuerpo de la **Action** utilicemos la función `UPDATE()`.`UPDATE()` permite determinar si una columna dada ha sido afectada por una sentencia `INSERT`, `UPDATE` o `DELETE` que es triggering event de un trigger. Esta función es accesible únicamente dentro de la **Action** de un trigger:

```
UPDATE (nombre-columna)
```

`nombre-columna` es el nombre de la columna para la cual se desea testear la existencia de modificaciones. La función devuelve un valor verdadero si la columna especificada ha sido afectada por una sentencia `INSERT` o `UPDATE`.

`UPDATE()` considera que la columna ha sido afectada por un `UPDATE` cuando la cláusula `SET` afecta directamente la columna en cuestión.

`UPDATE()` considera que la columna ha sido afectada por un `INSERT` en los siguientes casos:

- La columna en cuestión está incluida en la lista de columnas o no existe lista de columnas (se incluyen todas las columnas)
- La columna no está incluida pero posee definido una constraint `DEFAULT`.
- La columna posee el status de `IDENTITY`.

4. Valores de tuplas anteriores y posteriores

Los triggers permiten que tanto la **Condition** como la **Action** puedan hacer referencia a antiguos valores de tuplas y/o nuevos valores de tuplas en dependencia de las alteraciones provocadas por el **Triggering event**.

El estándar SQL define para ello la keyword `REFERENCING`.

En el caso del **Ejemplo 2**:

```
CREATE TRIGGER trTitles2
   AFTER UPDATE OF price ON Titles
A  REFERENCING
      OLD ROW AS OldTupla
      NEW ROW AS NewTupla
   FOR EACH ROW
   WHEN (OldTupla.price > NewTupla.price)
      UPDATE Titles
         SET price = OldTupla.price
         WHERE title_id = NewTupla.title_id;
```

La cláusula `REFERENCING` (**A**) permite que la **condition** y **action** del trigger hagan referencia a la tupla que está siendo modificada.
En el caso de un update –como en este ejemplo- esta cláusula nos permite proporcionar **nombres** a la tupla anterior y posterior a la modificación.
Tanto la condition como la action del trigger puedan hacer referencia a la tupla antigua (la tupla previa al update) como la nueva tupla (la tupla posterior al update). En nuestro ejemplo, estas tuplas serán referenciadas como `OldTupla` y `NewTupla` respectivamente.

En la condition y la action, estos nombres pueden ser utilizados como si se tratase de *tuple variables* (alias) declaradas en la cláusula `FROM` de un query SQL común y corriente.

En el ejemplo el Action es una sentencia SQL `UPDATE` que restituye el precio de la publicación al valor anterior al update.

Valores de tuplas anteriores y posteriores

PostgreSQL no sigue el estándar SQL en este aspecto. No permite la definición de "alias" para las tuplas antiguas y nuevas.
En cambio, la trigger function dispone de dos variables de tipo `RECORD`:

`NEW`
Es una variable de tipo `RECORD` con exactamente la misma estructura de una tupla de la tabla a la que se asocia el trigger, y que contiene la nueva tupla para triggering events `INSERT` o `UPDATE` en row-level triggers.
Esta variable es `NULL` para operaciones `DELETE`.

`OLD`
Es una variable de tipo `RECORD` con exactamente la misma estructura de una tupla de la tabla la que se asocia el trigger, y que contiene la tupla antigua para triggering events `UPDATE` o `DELETE` en row-level triggers.
Esta variable es `NULL` para Triggering events de tipo `INSERT`.

Estas variables **sólo están disponibles en row-level triggers**.

En la **Condition**, PL/pgSQL también permite, en row-level triggers, que hagamos referencia a las columnas antiguas y nuevas con la sintaxis `OLD.nombre-columna` y `NEW.nombre-columna` respectivamente. Por supuesto, los triggers con triggering event `INSERT` no pueden referenciar a `OLD` y los triggers con triggering event `DELETE` no pueden referenciar a `NEW`.

Triggers AFTER row-level

El valor de retorno de una trigger function en un AFTER trigger row-level es ignorado. Puede ser `NULL`.

La siguiente sería la solución PL/pgSQL al **Ejemplo 2**:

```
CREATE FUNCTION test()
   RETURNS trigger
   LANGUAGE plpgsql
   AS
   $$
   DECLARE
   BEGIN
      UPDATE Titles
         SET price = OLD.price
         WHERE title_id = NEW.title_id;
      RETURN NULL;
   END
   $$;


CREATE TRIGGER trTitles3
   AFTER UPDATE OF price ON Titles
   FOR EACH ROW
   WHEN (OLD.price > NEW.price)
      EXECUTE PROCEDURE test();
```

Cuando el triggering event de un row-level trigger es un `UPDATE`, tendremos una tupla antigua y una nueva (las tuplas previas y posteriores al `UPDATE`, respectivamente).
En ANSI SQL podemos dar a estas tuplas nombres a través de las cláusulas `OLD ROW AS` y `NEW ROW AS`, como vimos.

Si el triggering event es un `INSERT`, podemos dar un nombre a la tupla insertada a través de la cláusula `NEW ROW AS`. Obviamente, el uso de `OLD ROW AS` no es válido en este caso.

A la inversa, en el caso de una eliminación podemos usar `OLD ROW AS` para dar un nombre a la tupla eliminada, y `NEW ROW AS` no es válido en ese caso.

Valores de tuplas anteriores y posteriores

T-SQL tampoco sigue el estándar SQL en este aspecto. No permite la definición de "alias" para las tuplas antiguas y nuevas. Pero tampoco tendría sentido, ya que no soporta row-level triggers.

5. Uso de los triggers BEFORE

Un uso importante de los **triggers BEFORE** es corregir o retocar algún aspecto de las tuplas que forman parte de una sentencia `INSERT` o `UPDATE` **antes** de que sean realmente plasmadas en la base de datos.

Como vimos, en los triggers BEFORE, la **Condition** es evaluada sobre el estado de la base de datos previo a la ejecución del **Triggering event**.
Si la **Condition** es verdadera, la **Action** del trigger es ejecutada ***sobre ese estado***.

Finalmente, se ejecuta el evento que activó la ejecución del trigger, más allá de que la **Condition** sea o no verdadera en ese momento.

Triggers BEFORE row-level

El *valor de retorno* de una trigger function de un *row-level BEFORE trigger* asume un rol determinante en el trabajo con triggers en PL/pgSQL. Lo revisaremos para cada caso:

A. Cancelar un Triggering Event
Si se trata de un Triggering event `INSERT`, `UPDATE` o `DELETE`, y deseamos cancelar el mismo para la tupla actual, el valor de retorno debe ser `NULL`. Esto equivale a deshacer la transacción.

B. Aprobar el Triggering event sin intervenir

Aquí tenemos dos sub-alternativas:

B.1. Si se trata de un triggering event `DELETE` y deseamos que el mismo efectivamente ocurra para la tupla actual, el valor de retorno no posee relevancia, pero debe ser diferente de `NULL`. Normalmente se estila retornar la table row `OLD`.

B.2. Si se trata de un triggering event `INSERT` o `UPDATE` y deseamos que el mismo efectivamente ocurra para la tupla actual -y sin intervención- el valor de retorno debe ser la table row `NEW`.

C. Aprobar el Triggering event interviniendo

Si se trata de un triggering event `INSERT` o `UPDATE` y deseamos que el mismo efectivamente ocurra para la tupla actual, tenemos la posibilidad de intervenir modificando o ajustando algún aspecto de la tupla que está por ser insertada o modificada.
Una vez realizada esta intervención, el valor de retorno puede ser la tupla `NEW` "adulterada" u otra table row completamente nueva con la misma estructura.
La tupla retornada se convierte en la tupla que será insertada o que reemplazará a la tupla que está siendo actualizada.

Ejemplo 3

Supongamos que queremos insertar tuplas de publicaciones, pero existen algunas de las que desconocemos el precio inicial.

Lo que podemos hacer es implementar un trigger que verifique que el atributo `price`, y, si el mismo es nulo, lo reemplace por un valor adecuado (tal vez uno que calculamos de alguna manera compleja).

En el siguiente ejemplo queremos reemplazar el precio por el valor `15` (algo que podríamos haber implementado a través de una constraint `DEFAULT`, pero sirve como ejemplo):

```
CREATE FUNCTION test()
   RETURNS trigger
   LANGUAGE plpgsql
   AS
   $$
   DECLARE
   BEGIN
      NEW.price:=15;
      RETURN NEW;
   END
   $$;

CREATE TRIGGER trTitles4
   BEFORE INSERT ON Titles
   FOR EACH ROW
   EXECUTE PROCEDURE test();
```

Como se mencionó antes, T-SQL no soporta BEFORE triggers.

6. Statement-level triggers

Creamos un statement-level trigger especificando la cláusula `FOR EACH STATEMENT`.

Valores de tuplas anteriores y posteriores

Es imposible acceder a las tuplas anteriores o posteriores desde un statement-level trigger.

Como dijimos antes, todos los triggers T-SQL son statement-level AFTER triggers. T-SQL sí permite acceder al conjunto de tuplas "anteriores" a la aplicación del triggering event a través de una "tabla virtual" denominada **deleted**, y al conjunto de tuplas "posteriores" a la aplicación del triggering event a través de una "tabla virtual" denominada **inserted**.

Estas tablas virtuales pueden ser manipuladas como cualquier tabla, con la salvedad de que son de solo lectura.

inserted posee entonces la estructura de la tabla a la que se asocia el trigger, y contiene las nuevas tuplas para triggering events `INSERT` o `UPDATE`.
inserted no está definida para triggering events `DELETE`.

deleted posee la misma estructura de la tabla a la que se asocia el trigger, y que contiene las antiguas tuplas para triggering events UPDATE o DELETE.
deleted no está definida para triggering events de tipo `INSERT`.

Triggers T-SQL y transacciones

Un uso típico en un trigger T-SQL es la implementación de constraints complejas. En estos casos, cuando no se cumple determinada condición, que debemos evaluar en el cuerpo de la Action, se procede a abortar la sentencia del triggering event a través de un `ROLLBACK TRANSACTION`.

Triggers statement-level BEFORE o AFTER

El valor de retorno de una trigger function asociada a un statement-level trigger (BEFORE o AFTER) es ignorado. Puede ser `NULL`.

Ejemplo 4

El promedio actual de precios de publicaciones es $15. Supongamos que queremos impedir que este promedio aumente.

La siguiente es la solución T-SQL:

```
CREATE TRIGGER AveragePriceTrigger
   ON Titles
   AFTER UPDATE
   AS
      IF(15 < (SELECT AVG(price) FROM Titles))
         ROLLBACK TRANSACTION
```

A

Estamos evaluando (**A**) si el promedio de precios de publicaciones quedó por encima de los $15 **después de la actualización**.

7. Trabajo con triggers

Eliminar un trigger

```
DROP TRIGGER <Nombre-trigger>
```

En PostgreSQL la sintaxis especifica la tabla sobre la que se aplica el trigger a eliminar:

```
DROP TRIGGER <Nombre-trigger>
   ON <Tabla-asociada>
```

8. Información disponible en triggers PL/pgSQL

En una trigger function PL/PgSQL disponemos de una variable de tipo `text` llamada `TG_OP`. Esta variable posee un valor string `"INSERT"`, `"UPDATE"`o `"DELETE"` indicando cuál fue el triggering event que desencadenó el trigger.

Otras variables que pueden resultar útiles son las siguientes:
`TG_NAME` proporciona el nombre del trigger en ejecución.
`TG_WHEN` proporciona el tipo de trigger: `BEFORE`, `AFTER` o `INSTEAD OF`.
`TG_LEVEL` proporciona la granularidad del trigger: `ROW` o `STATEMENT`.
`TG_TABLENAME` proporciona el nombre de la tabla a la cual está asociada el trigger.

Ejercicio 1.

En SQL Server, cree una copia (`Autores`) de la tabla `Authors`. Luego defina un trigger llamado `tr_ejercicio1` asociado al evento `DELETE` sobre la misma. El trigger debe retornar un mensaje (usando PRINT) `"Se eliminaron n filas"` indicando la cantidad de filas afectadas en la operación. Dispare luego la siguiente sentencia SQL.

```
DELETE
   FROM autores
   WHERE au_id = "172-32-1176" or
         au_id = "213-46-8915"
```

Ejercicio 2.

Implemente en T-SQL un trigger (`tr_ejercicio2`) asociado a la tabla `autores` para inserción y actualización. El trigger debe mostrar un mensaje `'Datos insertados en transaction log'`, y a continuación los datos insertados. Luego `'Datos eliminados en transaction log'` y a continuación los datos eliminados.

Modifique la configuración del menú *Query* a fin de que pueda visualizar la salida de la ejecución como texto. Luego inserte la siguiente fila y evalúe los resultados:

```
INSERT INTO autores
   SELECT '111-11-1111', 'Lynne', 'Jeff', '415 658-9932',
          'Galvez y Ochoa', 'Berkeley', 'CA', '94705', 1
```

Modifique la fila insertada y evalúe los resultados:

```
UPDATE Autores
   SET au_fname = 'Nicanor' WHERE au_id = '111-11-1111'
```

Ejercicio 3.

Implemente un trigger T-SQL (`tr_ejercicio3`) para inserción sobre la tabla `productos` que, ante la inserción de un producto con stock negativo, dispare el error de aplicación `'El stock debe ser positivo o cero'` con una severidad `12` y contexto `1` y deshaga la transacción. Testee su funcionamiento disparando la siguiente sentencia `INSERT`:

```
INSERT INTO Productos
   VALUES (10, 'Producto 10', 200, -6)
```

Ejercicio 4.

Implemente un trigger T-SQL (`tr_ejercicio4`) que impida insertar publicaciones de editoriales que no hayan vendido por más de `$1500` (tabla `Sales`).

Por ejemplo, La editorial `'1389'` posee un monto de ventas que debería permitir la inserción de sus publicaciones. En cambio, para la editorial `'0736'` seguramente se debería impedir la inserción de publicaciones.

Puede probar el trigger con las siguientes sentencias `INSERT`:

```
INSERT INTO titles
   SELECT 'PC4545', 'Prueba 1', 'trad_cook', '1389',
          14.99, 8000.00, 10, 4095, 'Prueba 1', '06/12/91'


INSERT INTO titles
   SELECT 'PC4646', 'Prueba 2', 'trad_cook', '0736',
          14.99, 8000.00, 10, 4095, 'Prueba 1', '06/12/91'
```

Ejercicio 5.

Escriba el mismo trigger como `tr_ejercicio5` en PL/pgSQL.

Ejercicio 6.

En PostgreSQL, agregue dos columnas adicionales a la tabla `Publishers`. `FechaHoraAlta` está destinada a guardar la fecha y hora en que se da de alta una editorial. `UsuarioAlta` se utilizará para registrar el usuario que realizó la operación de inserción:

```
ALTER TABLE publishers
   ADD COLUMN FechaHoraAlta DATE NULL;

ALTER TABLE publishers
   ADD COLUMN UsuarioAlta VARCHAR(255) NULL;
```

Defina un trigger (`tr_ejercicio6`) que, ante la inserción de una editorial, permita registrar la fecha y hora de la operación (función `CURRENT_TIMESTAMP`) y el usuario que llevó a cabo la operación (función `SESSION_USER`);

Ejercicio 7.

Tenemos una tabla de registro con la siguiente estructura:

```
CREATE TABLE Registro
    (
     fecha DATE NULL,
     tabla varchar(100) NULL,
     operacion varchar(30) NULL,
     CantFilasAfectadas Integer NULL
    )
```

Se desean registrar en ella algunos movimientos que afectan varias filas, como `UPDATE` y `DELETE`.

Se desea crear un trigger (`tr_Ejercicio7`) para `DELETE` sobre la tabla `Employee` que por cada sentencia `DELETE` **que afecte más de una tupla** genere una entrada en la tabla `Registro`.

Defina el trigger en T-SQL ó PL/pgSQL. Encuentra inconvenientes en alguno de los dos DBMS?

Ejercicio 8.

Queremos auditar, por cada fila insertada en la tabla `Publishers`, fecha y hora de alta y usuario que realizó la misma (tal como hicimos en el Ejercicio 6). Podemos optar por:

A. Agregar dos columnas adicionales a la tabla `Publishers` (tal como hicimos en el Ejercicio 6) o bien

B. Tener una tabla de log independiente (como la tabla `Registro` del Ejercicio 8).

¿Que configuración de trigger PL/pgSQL elegiría si se adopta la opción A?

¿Que configuración de trigger PL/pgSQL elegiría si se adopta la opción B?