

### 3.- AUTOMATIZACIÓN DE TAREAS: CONSTRUCCIÓN DE GUIONES DE ADMINISTRACIÓN.

#### 3.2.- Disparadores (triggers):

A partir de MySQL 5.0.2 se incorporó el soporte básico para disparadores (triggers). Un trigger (o disparador) en una Base de datos, es un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación. En otras palabras, un disparador es un objeto de base de datos con nombre que se asocia a una tabla, y se activa cuando ocurre un evento en particular para la tabla. Algunos usos para los disparadores es verificar valores a ser insertados o llevar a cabo cálculos sobre valores involucrados en una actualización.

Un disparador se asocia con una tabla y se define para que se active al ocurrir una sentencia INSERT, DELETE, o UPDATE sobre dicha tabla. Puede también establecerse que se active antes o después de la sentencia en cuestión. Por ejemplo, se puede tener un disparador que se active antes de que un registro sea borrado, o después de que sea actualizado. También se pueden ejecutar triggers al crear, borrar o editar usuarios, tablas, bases de datos u otros objetos.

Son usados para mejorar la administración de la Base de datos, sin necesidad de contar con que el usuario ejecute la sentencia de SQL. Además, pueden generar valores de columnas, previene errores de datos, sincroniza tablas, modifica valores de una vista, etc.

La estructura básica de un trigger es:

- Llamada de activación: es la sentencia que permite "disparar" el código a ejecutar.
- Restricción: es la condición necesaria para realizar el código. Esta restricción puede ser de tipo condicional o de tipo nulidad.
- Acción a ejecutar: es la secuencia de instrucciones a ejecutar una vez que se han cumplido las condiciones iniciales.

La **sintaxis** para crear una trigger es la siguiente:

```
CREATE TRIGGER nombre_disp momento_disp evento_disp  
ON nombre_tabla FOR EACH ROW sentencia_disp
```

Donde

- ✓ Un disparador es un objeto con nombre (*nombre\_disp*) en una base de datos que se asocia con una tabla, y se activa cuando ocurre un evento en particular para esa tabla.
- ✓ El disparador queda asociado a la tabla *nombre\_tabla*. Esta debe ser una tabla permanente, no puede ser una tabla temporal ni una vista.
- ✓ *momento\_disp* es el momento en que el disparador entra en acción. Puede ser BEFORE (antes) o AFTER (después), para indicar que el disparador se ejecute antes o después que la sentencia que lo activa.

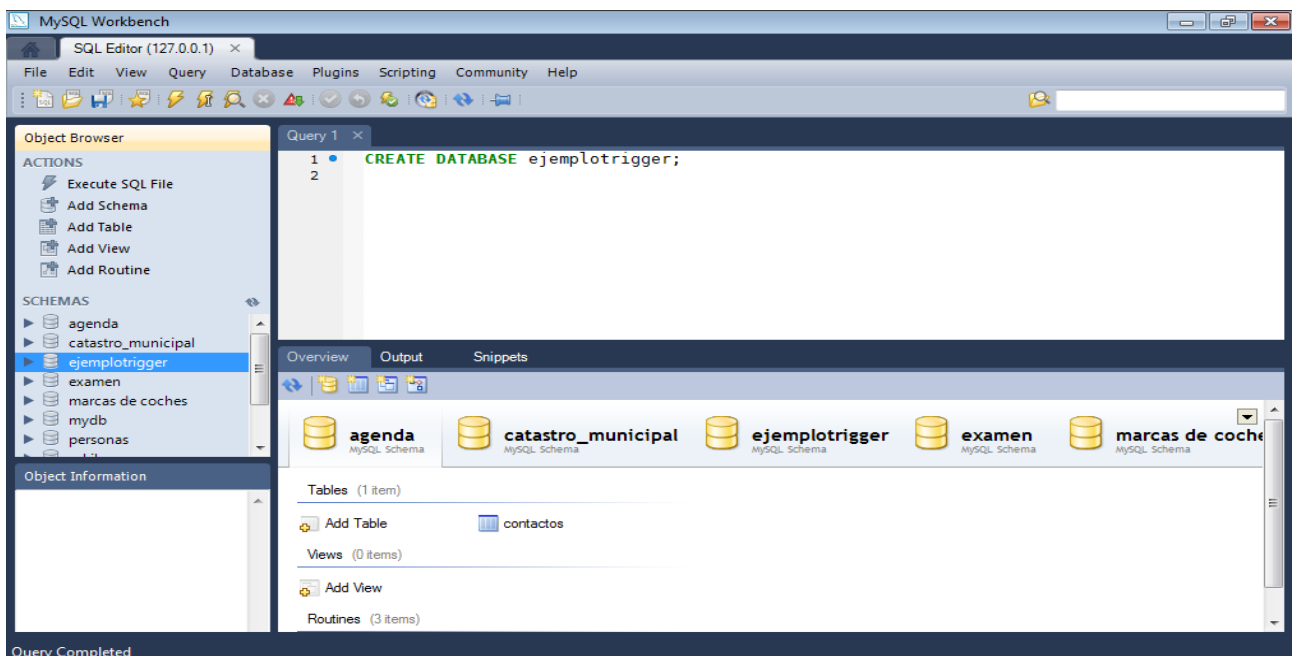
- ✓ `evento_disp` indica la clase de sentencia que activa al disparador. Puede ser INSERT, UPDATE, o DELETE. Por ejemplo, un disparador BEFORE para sentencias INSERT podría utilizarse para validar los valores a insertar.
- ✓ No puede haber dos disparadores en una misma tabla que correspondan al mismo momento y sentencia. Por ejemplo, no se pueden tener dos disparadores BEFORE UPDATE. Pero sí es posible tener los disparadores BEFORE UPDATE y BEFORE INSERT o BEFORE UPDATE y AFTER UPDATE.
- ✓ `sentencia_disp` es la sentencia que se ejecuta cuando se activa el disparador. Si se desean ejecutar múltiples sentencias, deben colocarse entre BEGIN ... END..
- ✓ La sentencia FOR EACH ROW, define lo que se ejecutará cada vez que el disparador se active, lo cual ocurre una vez por cada fila afectada por la sentencia activadora.

Ejemplo:

Vamos a crear un trigger que almacenará la fecha de la inserción, el nombre del usuario, el tipo de proceso y el id del registro agregado del usuario que introduzca un artículo en una tabla que almacena los artículos de la empresa. Es decir cuando un usuario hace una nueva entrada en la tabla artículos se insertará en una tabla llamada log\_articulos un registro con los datos del usuario que insertó ese nuevo artículo.

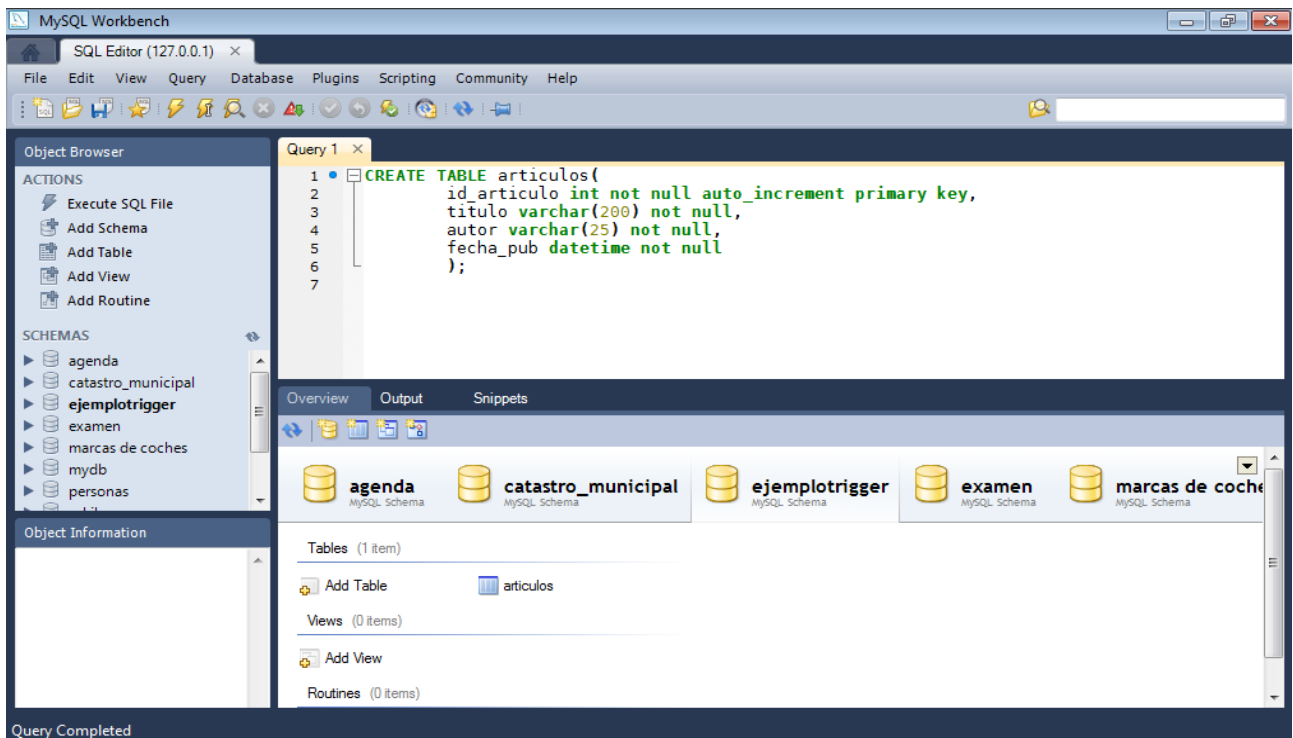
Creamos la base de datos:

```
CREATE DATABASE ejemplotrigger;
```



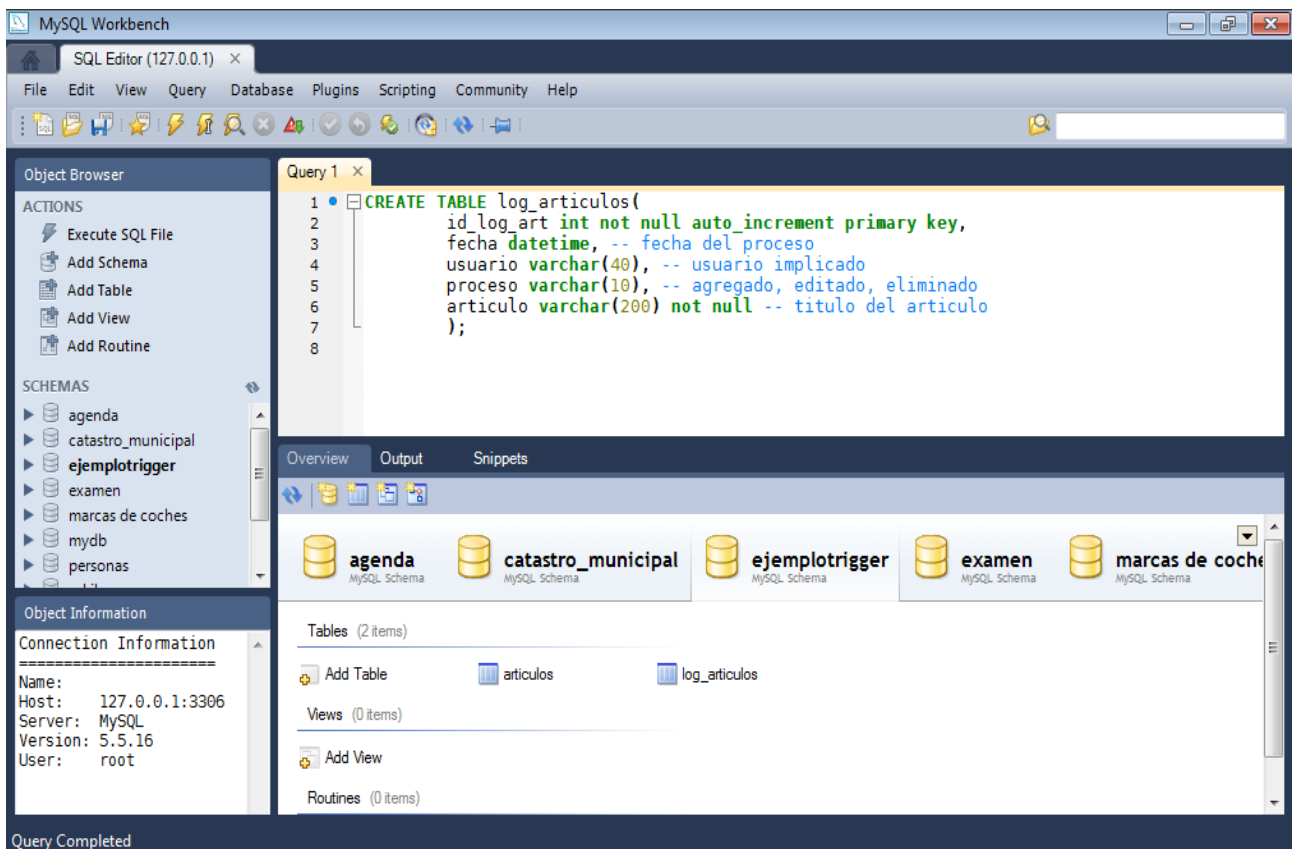
Creamos la tabla donde se almacenan los artículos:

```
CREATE TABLE articulos(  
  id_articulo int not null auto_increment primary key,  
  titulo varchar(200) not null,  
  autor varchar(25) not null, -- podría ser el id de una tabla "usuarios"  
  fecha_pub datetime not null, -- fecha de la publicacion);
```

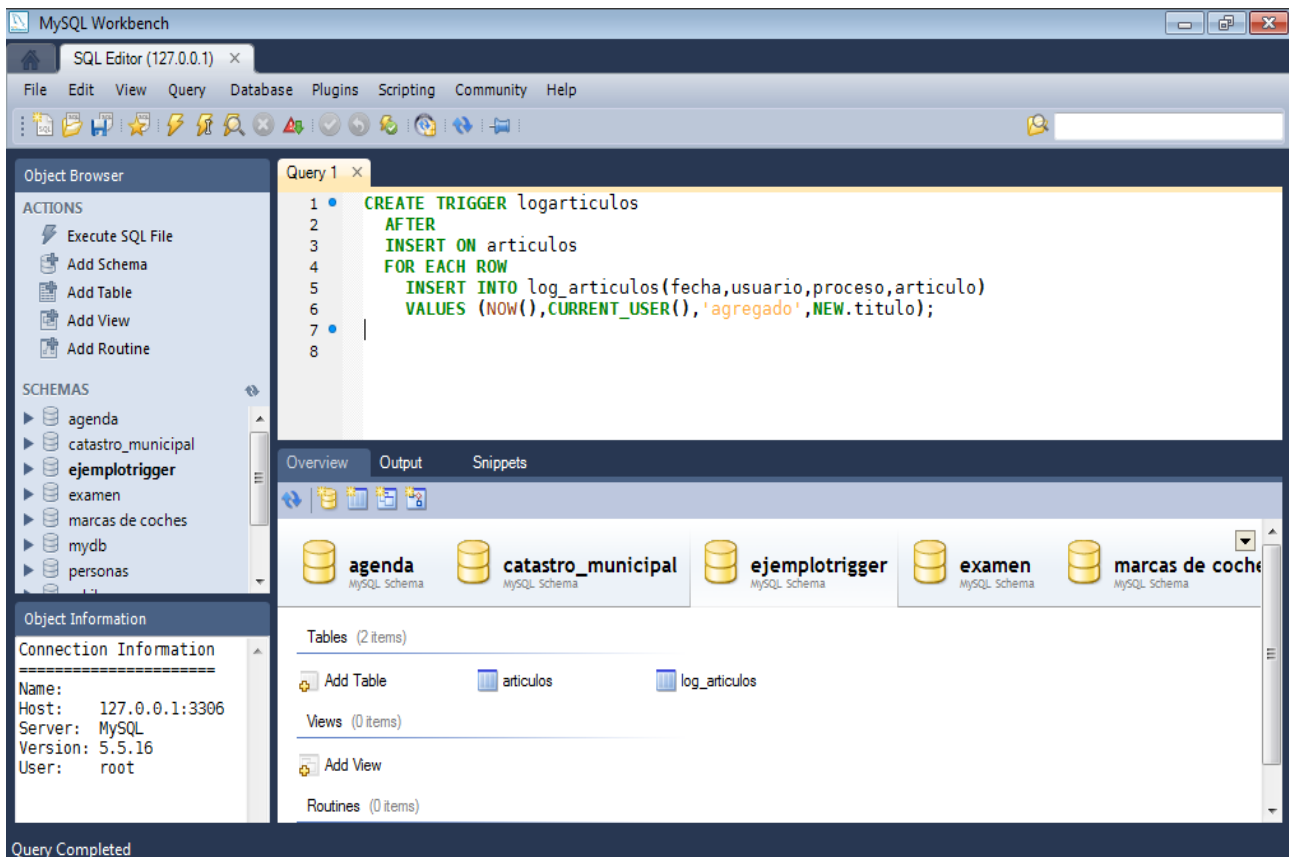


Creamos la tabla que almacena los logeos sobre la tabla "articulos"

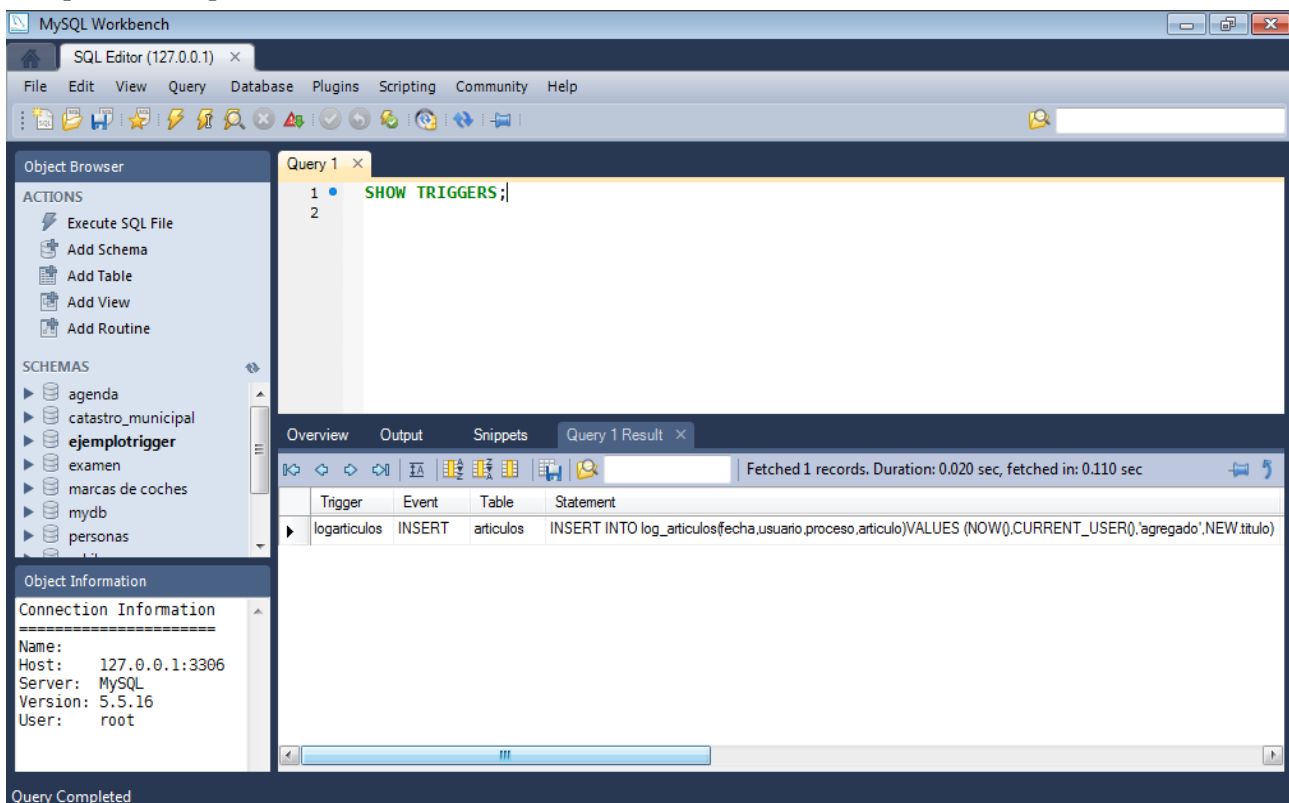
```
CREATE TABLE log_articulos(
    id_log_art int not null auto_increment primary key,
    fecha datetime, -- fecha del proceso
    usuario varchar(40), -- usuario implicado
    proceso varchar(10), -- agregado, editado, eliminado
    articulo varchar(200) not null -- titulo del articulo);
```



Creamos el disparador:

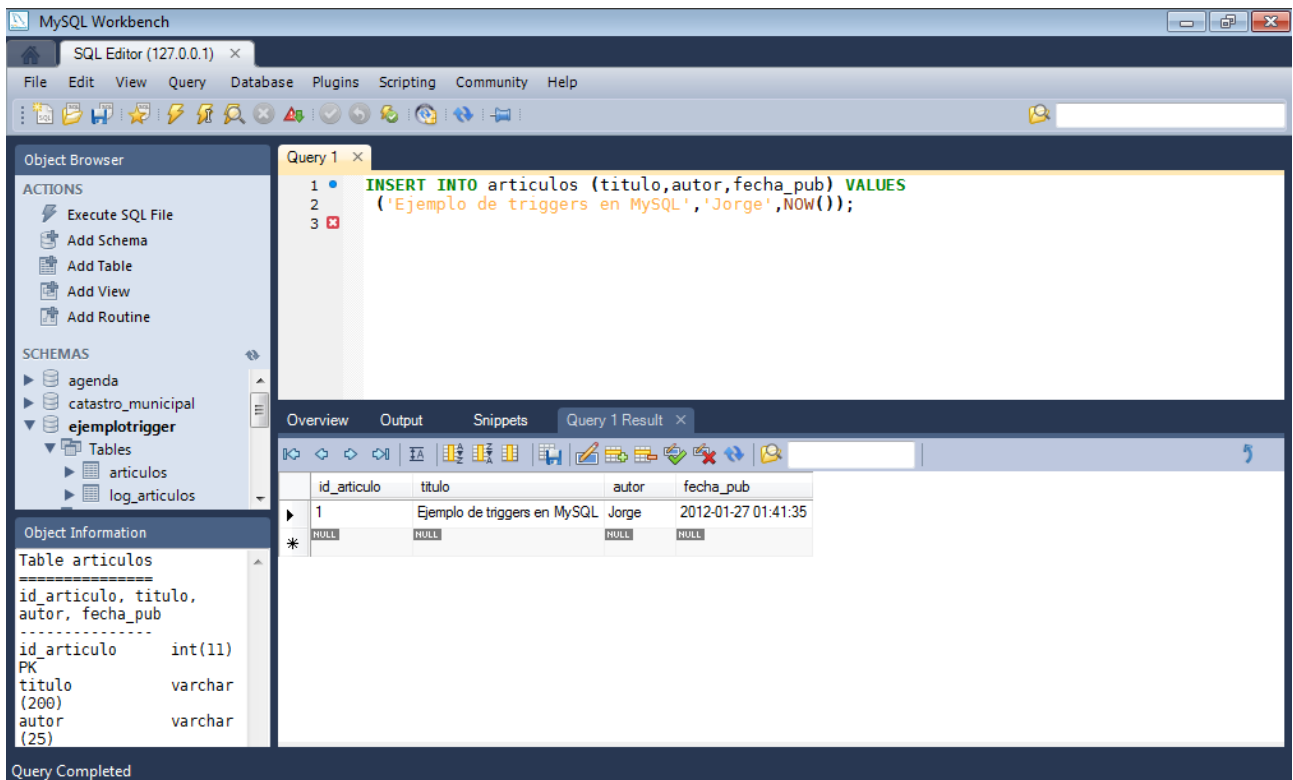


Comprobamos que se ha creado:



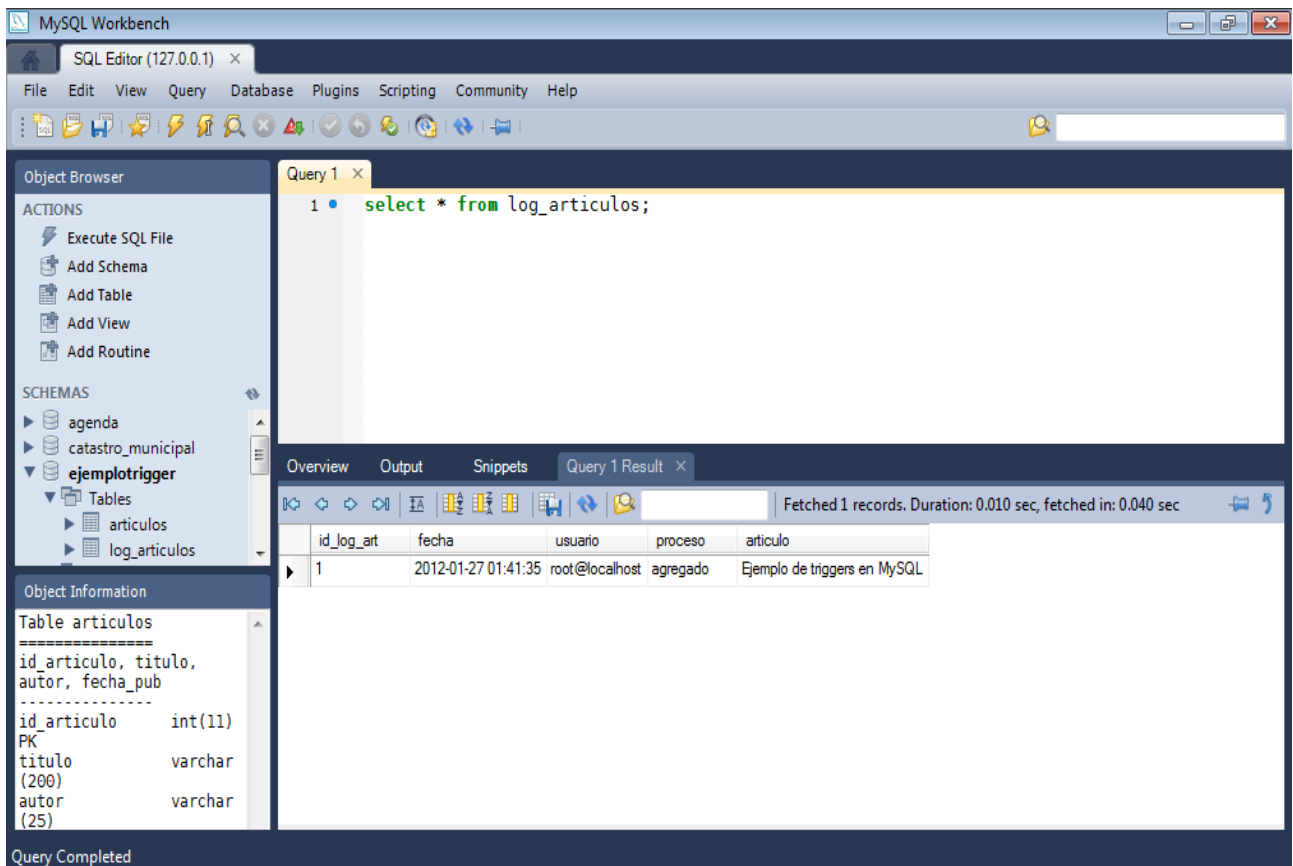
Ahora insertamos un artículo:

```
INSERT INTO articulos (titulo, autor, fecha_pub) VALUES
('Ejemplo de triggers en MySQL', 'Jorge', NOW());
```



Ahora comprobamos que se ha introducido un nuevo registro en la tabla log\_articulos que nos da información sobre el usuario que introdujo el nuevo artículo:

```
select * from log_articulos;
```



Las columnas de la tabla asociada con el disparador pueden referenciarse empleando los alias OLD y NEW. OLD se refiere a un registro existente que va a borrarse o que va a actualizarse antes de que esto ocurra. NEW se refiere a un registro nuevo que se insertará o a un registro modificado luego de que ocurre la modificación.

Las palabras clave OLD y NEW permiten acceder a columnas en los registros afectados por un disparador. En un disparador para INSERT, solamente puede utilizarse NEW.nom\_col; ya que no hay una versión anterior del registro. En un disparador para DELETE sólo puede emplearse OLD.nom\_col, porque no hay un nuevo registro. En un disparador para UPDATE se puede emplear OLD.nom\_col para referirse a las columnas de un registro antes de que sea actualizado, y NEW.nom\_col para referirse a las columnas del registro luego de actualizarlo.

Una columna precedida por OLD es de sólo lectura. Es posible hacer referencia a ella pero no modificarla. Una columna precedida por NEW puede ser referenciada si se tiene el privilegio SELECT sobre ella. En un disparador BEFORE, también es posible cambiar su valor con SET NEW.nombre\_col = valor si se tiene el privilegio de UPDATE sobre ella. Esto significa que un disparador puede usarse para modificar los valores antes que se inserten en un nuevo registro o se empleen para actualizar uno existente.

Para mostrar los triggers dentro de una base de datos, usamos:

```
show triggers;
```

Podemos eliminar un disparador si posee un nombre que vayamos a usar:

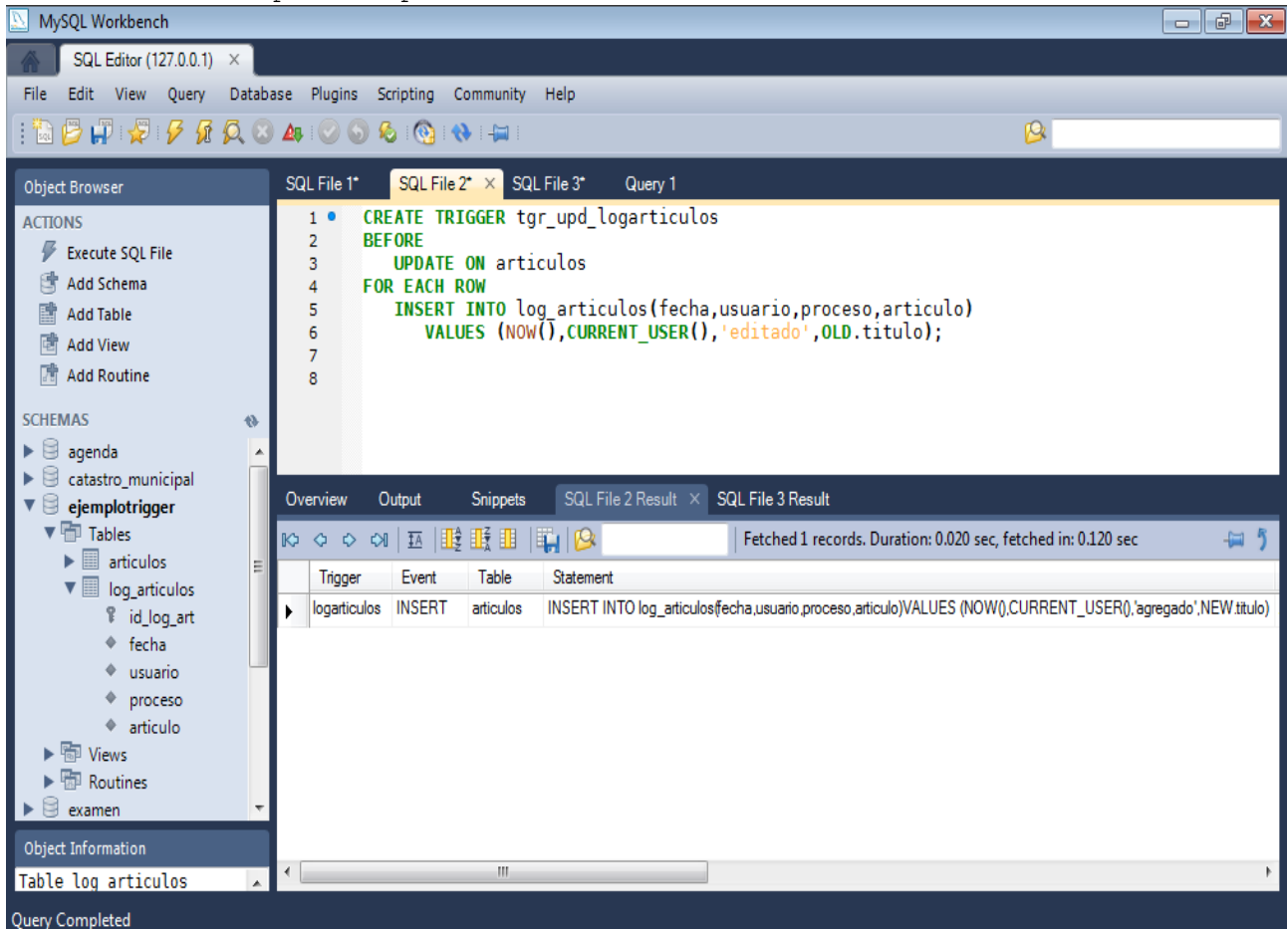
```
DROP TRIGGER IF EXISTS nombre de trigger;
```

Debido a que un disparador está asociado con una tabla en particular, no se pueden tener múltiples disparadores con el mismo nombre dentro de una tabla. También se debería tener en cuenta que el espacio de nombres de los disparadores puede cambiar en el futuro de un nivel de tabla a un nivel de base de datos, es decir, los nombres de disparadores ya no sólo deberían ser únicos para cada tabla sino para toda la base de datos. Para una mejor compatibilidad con desarrollos futuros, se debe intentar emplear nombres de disparadores que no se repitan dentro de la base de datos.

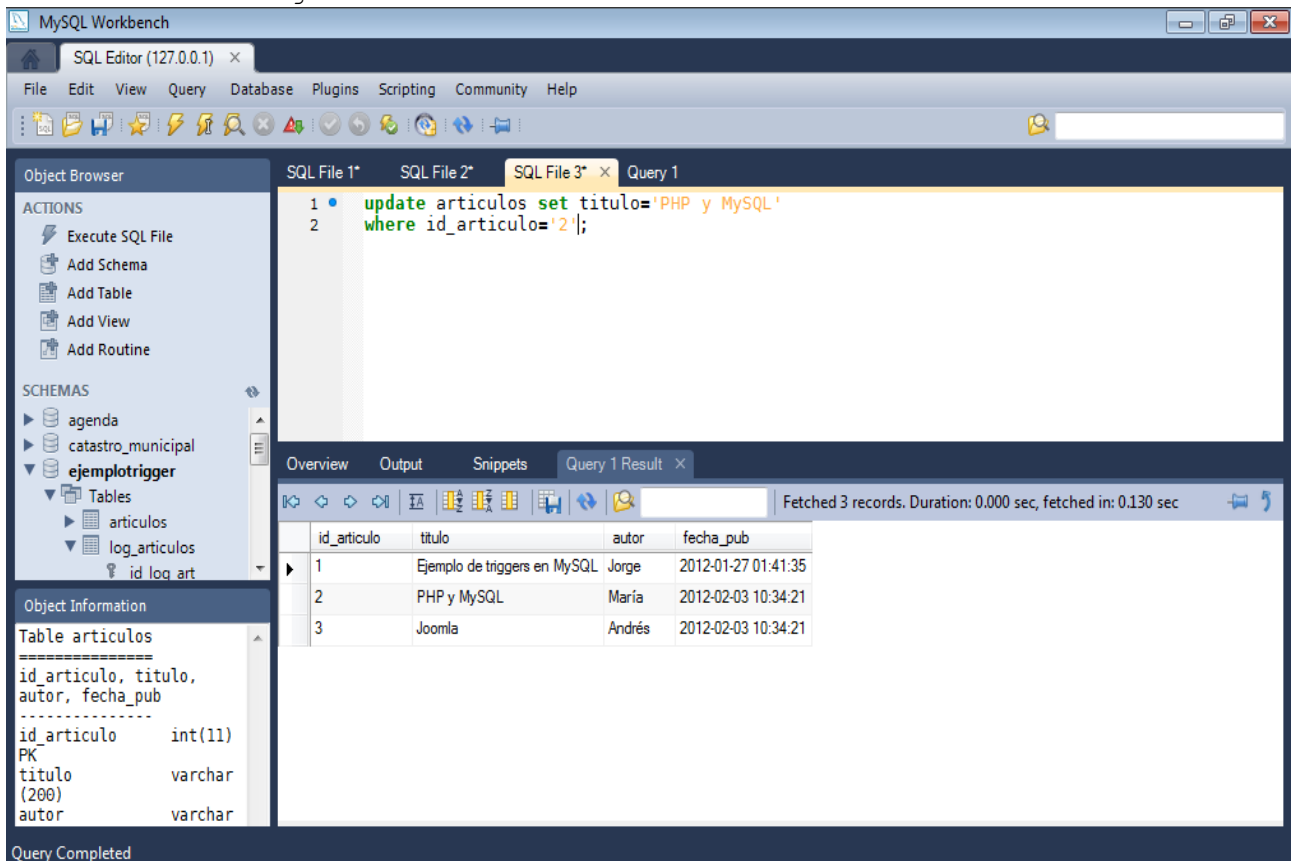
Adicionalmente al requisito de nombres únicos de disparador en cada tabla, hay otras limitaciones en los tipos de disparadores que pueden crearse. En particular, no se pueden tener dos disparadores para una misma tabla que sean activados en el mismo momento y por el mismo evento. Por ejemplo, no se pueden definir dos BEFORE INSERT o dos AFTER UPDATE en una misma tabla. Es improbable que esta sea una gran limitación, porque es posible definir un disparador que ejecute múltiples sentencias empleando el constructor de sentencias compuestas BEGIN ... END luego de FOR EACH ROW.

A continuación seguimos con el ejemplo anterior, pero aplicado a UPDATE Y DELETE.

-- creamos el disparador para las actualizaciones



Actualizamos un registro:



Comprobamos en la tabla log\_articulos el artículo que se ha editado, quién y cuándo se actualizó:

The screenshot shows the MySQL Workbench interface. The SQL Editor contains the query: `select * from log_articulos;`. The Query 1 Result tab displays 15 records. The record for the edited article is highlighted in blue.

id_log_art	fecha	usuario	proceso	articulo
1	2012-01-27 01:41:35	root@localhost	agregado	Ejemplo de triggers en MySQL
2	2012-02-03 10:34:21	root@localhost	agregado	PHP
3	2012-02-03 10:34:21	root@localhost	agregado	Joomla
4	2012-02-04 02:27:42	root@localhost	editado	PHP
5	2012-02-04 02:27:51	root@localhost	editado	PHP y MySQL
6	2012-02-04 02:27:52	root@localhost	editado	PHP y MySQL
7	2012-02-04 02:27:54	root@localhost	editado	PHP y MySQL
8	2012-02-04 02:27:55	root@localhost	editado	PHP y MySQL
9	2012-02-04 02:27:56	root@localhost	editado	PHP v MySQL

-- creamos el disparador para las eliminaciones:

The screenshot shows the MySQL Workbench interface. The SQL Editor contains the following SQL code to create a trigger:

```

1 CREATE TRIGGER tgr_del_logarticulos
2 AFTER
3 DELETE ON articulos
4 FOR EACH ROW
5 INSERT INTO log_articulos(fecha,usuario,proceso,articulo)
6 VALUES (NOW(),CURRENT_USER(),'eliminado',OLD.titulo);
7

```

The SQL File 3 Result tab displays 3 records, showing the trigger creation details.

Trigger	Event	Table	Statement
logarticulos	INSERT	articulos	INSERT INTO log_articulos(fecha,usuario,proceso,articulo)VALUES (NOW(),CURRENT_USER(),'agregado',NEW
tgr_upd_logarticulos	UPDATE	articulos	INSERT INTO log_articulos(fecha,usuario,proceso,articulo) VALUES (NOW(),CURRENT_USER(),'editado',OL
tgr_del_logarticulos	DELETE	articulos	INSERT INTO log_articulos(fecha,usuario,proceso,articulo) VALUES (NOW(),CURRENT_USER(),'eliminado',O



Borramos un registro para comprobarlo:

The screenshot shows the MySQL Workbench interface. In the SQL Editor, a query is entered: `delete from articulos where id_articulo='1';`. The query is executed, and the 'Query 1 Result' tab shows the output. The status bar indicates 'Fetches 2 records. Duration: 0.010 sec, fetched in: 0.120 sec'.

id_articulo	titulo	autor	fecha_pub
2	PHP y MySQL	María	2012-02-03 10:34:21
3	Joomla	Andrés	2012-02-03 10:34:21

Object Information for `ejemplotrigger.log_articulos`:

```

usuario, proceso,
articulo
-----
id_log_art      int(11)
PK
fecha           datetime
usuario         varchar
(40)
proceso         varchar
(10)
  
```

Query Completed

Comprobamos en la tabla `log_articulos` el artículo que se ha borrado, quién y cuándo se eliminó:

The screenshot shows the MySQL Workbench interface. In the SQL Editor, a query is entered: `select * from log_articulos;`. The query is executed, and the 'Query 1 Result' tab shows the output. The status bar indicates 'Fetches 16 records. Duration: 0.010 sec, fetched in: 0.040 sec'.

id_log_art	fecha	usuario	proceso	articulo
9	2012-02-04 02:27:56	root@localhost	editado	PHP y MySQL
10	2012-02-04 02:27:56	root@localhost	editado	PHP y MySQL
11	2012-02-04 02:28:02	root@localhost	editado	PHP y MySQL
12	2012-02-04 02:28:03	root@localhost	editado	PHP y MySQL
13	2012-02-04 02:28:30	root@localhost	editado	PHP y MySQL
14	2012-02-04 02:28:33	root@localhost	editado	PHP y MySQL
15	2012-02-04 02:28:34	root@localhost	editado	PHP y MySQL
16	2012-02-04 02:39:19	root@localhost	eliminado	Ejemplo de triggers en MySQL

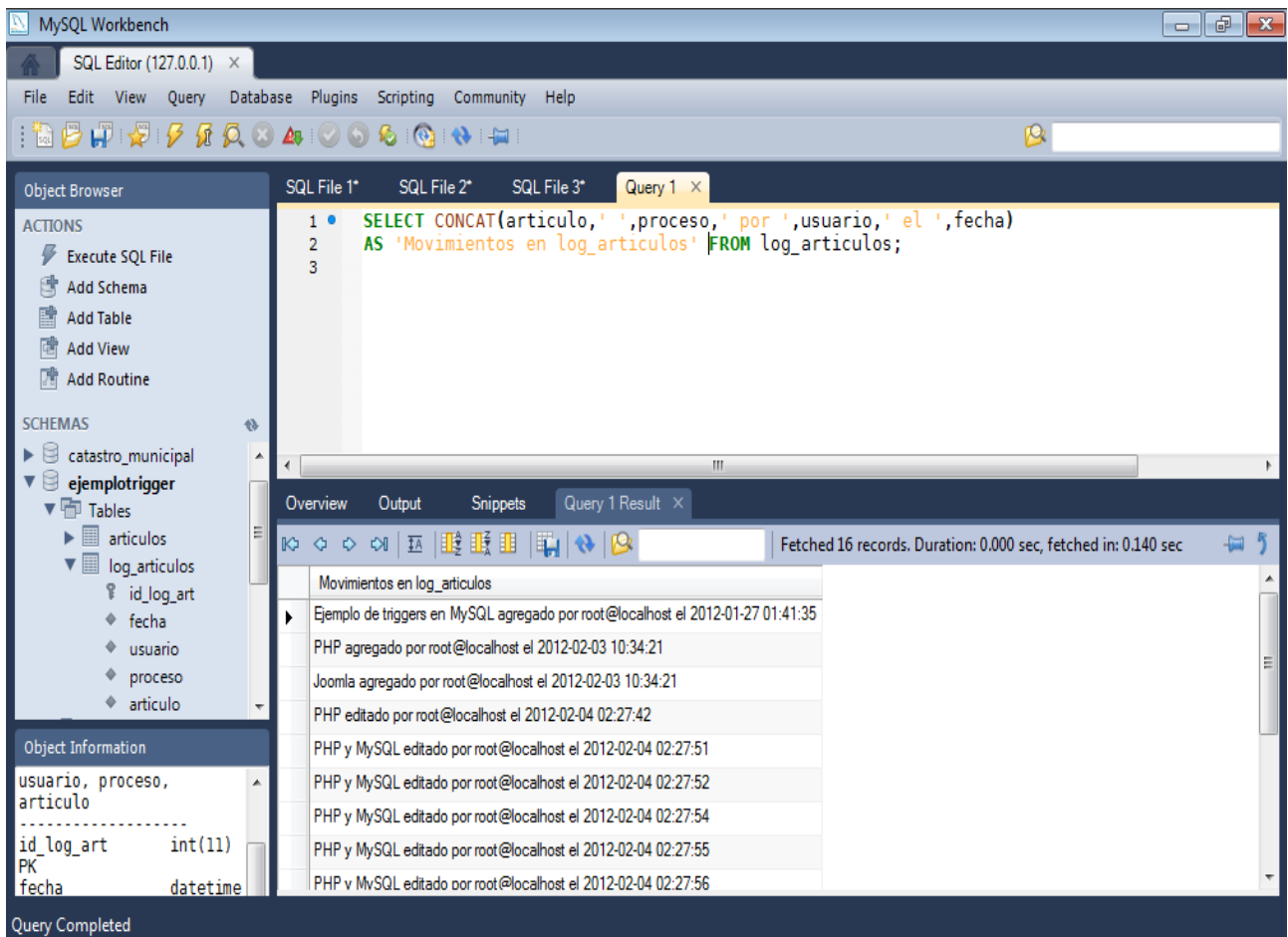
Object Information for `ejemplotrigger.log_articulos`:

```

usuario, proceso,
articulo
-----
id_log_art      int(11)
PK
fecha           datetime
usuario         varchar
(40)
proceso         varchar
(10)
  
```

Query Completed

Ahora hacemos una consulta de la tabla `log_articulos` para ver los movimientos que se han producido en la tabla `articulos`:



The screenshot shows the MySQL Workbench interface. The SQL Editor window contains the following query:

```
1 SELECT CONCAT(articulo, ' ', proceso, ' por ', usuario, ' el ', fecha)
2 AS 'Movimientos en log_articulos' FROM log_articulos;
3
```

The Object Browser on the left shows the database structure, including the `ejemplotrigger` database and the `log_articulos` table. The Object Information panel shows the fields `usuario`, `proceso`, `articulo`, `id_log_art` (PK), and `fecha`.

The Query Results window displays the output of the query, showing 16 records. The records are as follows:

Movimientos en log_articulos
Ejemplo de triggers en MySQL agregado por root@localhost el 2012-01-27 01:41:35
PHP agregado por root@localhost el 2012-02-03 10:34:21
Joomla agregado por root@localhost el 2012-02-03 10:34:21
PHP editado por root@localhost el 2012-02-04 02:27:42
PHP y MySQL editado por root@localhost el 2012-02-04 02:27:51
PHP y MySQL editado por root@localhost el 2012-02-04 02:27:52
PHP y MySQL editado por root@localhost el 2012-02-04 02:27:54
PHP y MySQL editado por root@localhost el 2012-02-04 02:27:55
PHP y MySQL editado por root@localhost el 2012-02-04 02:27:56

Query Completed