/\*¿Qué tipo de entidades? Autores

Nombre: autores

comandos:

- USE DATABASE libreria\_cf;

- SHOW DATABASES;

- SHOW TABLES;

- SELECT \* FROM autores; (para obtener todos los registros de una tabla)

\*/

/\*TIP: todas nuestras tablas deben poseer campos que nos permitan saber

cuando fue creado un registro\*/

comandos:

- SELECT current\_timestamp;(es para obtener la fecha exacta)

- UNSIGNED: para no permitir números negativos

+TIPO ENUM

+integridad referencial

Como sabemos, si nosotros necesitamos validar valores únicos usaremos el constraint UNIQUE.

CREATE TABLE usuarios(

usuario\_id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

nombre VARCHAR(50) UNIQUE

);

Otra forma de expresarlo, es de la siguiente manera.

CREATE TABLE usuarios(

usuario\_id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

nombre VARCHAR(50),

CONSTRAINT UNIQUE (nombre)

);

La palabra CONSTRAINT es opcional, sin embargo, por temas de legibilidad recomiendo colocarla.

VALORES MÚTIPLES COLUMNAS

Si necesitamos validar el valor único de una combinación de columnas lo haremos de la siguiente manera.

En este caso queremos validar que la combinación de nombre, apellido y matricula sean unicas en la tabla.

CREATE TABLE usuarios(

usuario\_id INT UNSIGNED NOT NULL AUTO\_INCREMENT,

nombre VARCHAR(50),

apellido VARCHAR(50),

matricula VARCHAR(10),

CONSTRAINT unique\_combinacion UNIQUE (nombre, apellido, matricula),

PRIMARY KEY (usuario\_id)

);

**Modificar tablas**

Abrir temario playlist\_play

Lo lees en 1 Min.

Nosotros podemos modificar la definición de una tabla una vez esta se haya creado, para ello, nos apoyaremos de la sentencia ALTER TABLE.

Aquí un listado de algunas modificaciones que podemos realizar (Las más comunes).

Renombrar tabla.

Renombrar la tabla usuarios por users.

ALTER TABLE usuarios RENAME users;

Agregar una nueva columna.

Agregar a la tabla usuarios, la columna email de tipo VARCHAR con máximo de 50 caracteres.

ALTER TABLE usuarios ADD email VARCHAR(50);

Agregar una nueva columna con constraints.

Agregar a la tabla usuarios, la columna email, validando su presencia.

ALTER TABLE usuarios ADD email VARCHAR(50) NOT NULL DEFAULT '';

Agregar a la tabla usuarios, la columna email, validando su valor único.

ALTER TABLE tabla ADD columna VARCHAR(50) UNIQUE;

Eliminar una columna

Eliminar la columna email de la tabla usuarios.

ALTER TABLE usuarios DROP COLUMN email;

Modificar el tipo de dato de una columna

Modificar el tipo de dato de la columna teléfono, (tabla usuarios) de INT a VARCHAR, máximo 50 caracteres.

ALTER TABLE usuarios MODIFY telefono VARCHAR(50);

Generar una llave primaria.

Generar una llave primaria a la tabla usuarios.

ALTER TABLE usuarios ADD id INT UNSIGNED NOT NULL AUTO\_INCREMENT, ADD PRIMARY KEY (id);

Agregar llave foránea.

Agregar una llave foranea a la tabla usuarios, con referencia a la tabla grupos.

ALTER TABLE usuarios ADD FOREIGN KEY(grupo\_id) REFERENCES grupos(grupo\_id);

Eliminar llaves foráneas

Eliminar la llave foranea grupo\_id de la tabla usuarios.

ALTER TABLE usuarios DROP FOREIGN KEY grupo\_id;

------------------------Lo lees en 2 Min.

Aquí un listado de ejercicios con los cuales puedes practicar a partir de los datos que se encuentran en el archivo sentencias.sql

Tu puedes agregar más datos (reales) si así tú lo deseas. Si aun no poses el archivo puedes descargarlo en el siguiente [link](https://github.com/codigofacilito/curso_base_de_datos)

#### LIBROS

* Obtener todos los libros escritos por autores que cuenten con un seudónimo.
* Obtener el título de todos los libros publicados en el año actual cuyos autores poseen un pseudónimo.
* Obtener todos los libros escritos por autores que cuenten con un seudónimo y que hayan nacido ante de 1965.
* Colocar el mensaje **no disponible** a la columna descripción, en todos los libros publicados antes del año 2000.
* Obtener la llave primaria de todos los libros cuya descripción sea diferente de **no disponible**.
* Obtener el título de los últimos 3 libros escritos por el autor con id 2.
* Obtener en un mismo resultado la cantidad de libros escritos por autores con seudónimo y sin seudónimo.

Ejemplo

| **con seudónimo** | **sin seudónimo** |
| --- | --- |
| 10 | 20 |

* Obtener la cantidad de libros publicados entre enero del año 2000 y enero del año 2005.
* Obtener el título y el número de ventas de los cinco libros más vendidos.
* Obtener el título y el número de ventas de los cinco libros más vendidos de la última década.
* Obtener la cantidad de libros vendidos por los autores con id 1, 2 y 3.

Ejemplo

| **autor** | **ventas** |
| --- | --- |
| 1 | 10 |
| 2 | 20 |
| 2 | 30 |

* Obtener el título del libro con más páginas.
* Obtener todos los libros cuyo título comience con la palabra “La”.
* Obtener todos los libros cuyo título comience con la palabra “La” y termine con la letra “a”.
* Establecer el stock en cero a todos los libros publicados antes del año de 1995
* Mostrar el mensaje **Disponible** si el libro con id 1 posee más de 5 ejemplares en stock, en caso contrario mostrar el mensaje **No disponible**.
* Obtener el título los libros ordenador por fecha de publicación del más reciente al más viejo.

##### AUTORES

* Obtener el nombre de los autores cuya fecha de nacimiento sea posterior a 1950
* Obtener la el nombre completo y la edad de todos los autores.
* Obtener el nombre completo de todos los autores cuyo último libro publicado sea posterior al 2005
* Obtener el id de todos los escritores cuyas ventas en sus libros superen el promedio.
* Obtener el id de todos los escritores cuyas ventas en sus libros sean mayores a cien mil ejemplares.

##### FUNCIONES

* Crear una función la cual nos permita saber si un libro es candidato a préstamo o no. Retornar “Disponible” si el libro posee por lo menos un ejemplar en stock, en caso contrario retornar “No disponible.”

# Ejercicios parte 2

Abrir temario playlist\_play

Lo lees en 1 Min.

Aquí un listado de ejercicios con los cuales puedes practicar a partir de los datos que se encuentran en el archivo sentencias.sql

Tu puedes agregar más datos (reales) si así tú lo deseas. Si aun no poses el archivo puedes descargarlo en el siguiente [link](https://github.com/codigofacilito/curso_base_de_datos)

* Obtener a todos los usuarios que han realizado un préstamo en los últimos diez días.
* Obtener a todos los usuarios que no ha realizado ningún préstamo.
* Listar de forma descendente a los cinco usuarios con más préstamos.
* Listar 5 títulos con más préstamos en los últimos 30 días.
* Obtener el título de todos los libros que no han sido prestados.
* Obtener la cantidad de libros prestados el día de hoy.
* Obtener la cantidad de libros prestados por el autor con id 1.
* Obtener el nombre completo de los cinco autores con más préstamos.
* Obtener el título del libro con más préstamos esta semana.

MOTORES DE ALMACENAMIENTO:

### GESTIÓN

Si nosotros así lo deseamos podemos cambiar el motor de almacenamiento. Existen dos formas de hacer esto. La primera, es modificar el archivo **my.cnf**.

[mysqld]

default-storage-engine = innodb

La segunda forma es hacerlo directamente desde nuestra sección, basta con ejecutar la siguiente sentencia.

SET storage\_engine=INNODB;

En ambos casos modificamos el motor de almacenamiento de MyISAM a InnoDB.

Si nosotros deseamos conocer qué motor de almacenamiento utiliza una tabla en particular, podemos hacerlo ejecutando la siguiente sentencia.

SHOW TABLE STATUS WHERE `Name` = 'tabla' \G;

Si deseamos crear una tabla utilizando un motor en particular, debemos seguir la siguiente estructura.

CREATE TABLE tabla\_innodb (id int, value int) ENGINE=INNODB;

CREATE TABLE tabla\_myisam (id int, value int) ENGINE=MYISAM;

CREATE TABLE tabla\_default (id int, value int);

# Eventos MySQL

Abrir temario playlist\_play

Lo lees en 3 Min.

A partir de la versión 5.1, MySQL añade el concepto de eventos. Un evento no es más que una tarea la cual se ejecuta de forma automática en un momento previamente programado. Si eres un usuarios Linux puedes ver a los eventos cómo los cron jobs .

Los eventos nos permitirán a nosotros cómo administradores de base de datos programar ciertas tareas las cuales queremos que se ejecuten de forma periódica o en un momento en concreto, de tal manera que podamos automatizar ciertos procesos.

[](https://imgur.com/L0j46Th)

#### EJEMPLOS

Para este tutorial me apoyaré de mi tabla test.

CREATE TABLE test(

evento VARCHAR(50),

fecha DATETIME

);

Lo primero que debemos de hacer es habilitar nuestro servidor para que pueda ejecutar eventos.

SET GLOBAL event\_scheduler = ON;

Posteriormente creamos nuestro evento; En mi caso, tendrá el nombre de insertion event. Este evento se ejecutará dentro de 1 min, y lo que hará, será insertar un registro en mi tabla.

CREATE EVENT insertion\_event

ON SCHEDULE AT CURRENT\_TIMESTAMP + INTERVAL 1 MINUTE

DO INSERT INTO test VALUES ('Evento 1', NOW());

El nombre del evento no debe de poseer más de 64 caracteres.

Si queremos que el evento se ejecute en una fecha en concreto, únicamente debemos de especificarlo en ON SCHEDULE AT. Recuerda, el formato es año-mes-día hora:minuto:segundo.

ON SCHEDULE AT '2018-12-31 12:00:00'

Si nuestro evento ejecutará más de una sentencia SQL debemos de apoyarnos de BEGIN y END.

DELIMITER //

CREATE EVENT insertion\_event

ON SCHEDULE AT CURRENT\_TIMESTAMP + INTERVAL 1 MINUTE

DO

BEGIN

INSERT INTO test VALUES ('Evento 1', NOW());

INSERT INTO test VALUES ('Evento 2', NOW());

INSERT INTO test VALUES ('Evento 3', NOW());

END //

DELIMITER ;

Una vez el evento se haya creado, nosotros podemos listarlo.

SHOW events\G;

Si queremos eliminar un evento haremos uso de **DROP**.

DROP EVENT nombre\_evento;

Es importante mencionar que una vez el evento haya expirado, este, será eliminado de forma automática. Si nosotros no queremos que esto ocurra debemos de apoyarnos de **ON COMPLETION**

ON SCHEDULE AT CURRENT\_TIMESTAMP + INTERVAL 1 MINUTE

ON COMPLETION PRESERVE

...

Algo común con los eventos es trabajar con store procedures.

CREATE EVENT nombre\_evento

ON SCHEDULE AT 'fecha de ejeución'

DO

CALL store\_procedure();

#### EVENTOS PERIÓDICOS.

Los eventos los podemos programar para que se ejecuten de forma periódica.

CREATE EVENT insertion\_event

ON SCHEDULE EVERY 1 MINUTE STARTS '2018-07-07 18:30:00'

DO INSERT INTO test VALUES ('Evento 1', NOW());

En esta ocasión, el evento se ejecuta cada minuto después de las 6:30 PM. Podemos programar la ejecución para cada segundo, minuto, hora, semana, mes o año.

Si nosotros queremos que el evento se ejecute entre un rango de fechas debemos de apoyarnos de **ENDS**.

CREATE EVENT insertion\_event

ON SCHEDULE EVERY 1 MINUTE STARTS '2018-07-07 18:30:00'

ENDS '2018-07-07 19:00:00'

DO INSERT INTO test VALUES ('Evento 1', NOW());

Ahora, el evento se ejecuta durante un periodo de 30 minutos (De 6:30 PM a 7:00 PM).

#### EDITAR EVENTOS

Si por alguna razón necesitamos detener un evento, lo que debemos de hacer es deshabilitarlo.

ALTER EVENT nombre\_evento

DISABLE;

Para habilitar nuevamente un evento colocamos **ENABLE**.

ALTER EVENT nombre\_evento

ENABLE;

#### DETENER EVENTOS

Para detener completamente todos los eventos, debemos de ejecutar la siguiente sentencia.

SET GLOBAL event\_scheduler = OFF;

#### ESTRUCTURA DE UN EVENTO

Los eventos en MySQL pueden ser tan complejos como nosotros los deseemos.

CREATE

[DEFINER = { user | CURRENT\_USER }]

EVENT

[IF NOT EXISTS]

event\_name

ON SCHEDULE schedule

[ON COMPLETION [NOT] PRESERVE]

[ENABLE | DISABLE | DISABLE ON SLAVE]

[COMMENT 'string']

DO event\_body;

schedule:

AT timestamp [+ INTERVAL interval] ...

| EVERY interval

[STARTS timestamp [+ INTERVAL interval] ...]

[ENDS timestamp [+ INTERVAL interval] ...]

interval:

quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |

WEEK | SECOND | YEAR\_MONTH | DAY\_HOUR | DAY\_MINUTE |

DAY\_SECOND | HOUR\_MINUTE | HOUR\_SECOND | MINUTE\_SECOND}

-----------------------------------------------CURSORES-------------------------------------

# Cursores MySQL

Abrir temario playlist\_play

Lo lees en 2 Min.

En base de datos un **Cursor** es un mecanismo el cual nos permite procesar fila por fila el resultado de una consulta.

Como sabemos SQL es un lenguaje orientado a conjuntos. Si nosotros deseamos alterar ciertos elementos en nuestra colección tendremos que hacerlo mediante condicione. Única y exclusivamente los elementos que cumpla con dichas condiciones podrán ser alterados. Con los cursores podremos trabajar con cada uno de los elementos (filas) de nuestra consulta sin tener que obtener nuevos conjuntos. Esto nos permitirá ser mucho más flexibles al momento de manipular la información.

Para nosotros poder hacer uso de un cursor será necesario seguir los siguientes pasos.

* Crear un cursor a partir de una sentencia SQL.
* Apertura del cursor.
* Acceso a datos.
* Cierre del cursor.

Es importante mencionar que en MySQL los cursores solo podrán ser utilizados dentro de stored procedures.

Veamos un ejemplo.

1.-Crear un stored procedure el cual incremente en 10 el número de páginas de cada libro (tabla libros). En consola debemos visualizar el título, el anterior número de páginas y el actual número de páginas de cada libro.

Bien, este ejemplo nos permitirá comprender de una mejor manera el uso de cursores.

Veamos como pudiese quedar nuestro stored procedure.

DELIMITER //

DROP PROCEDURE IF EXISTS facilito\_procedure//

CREATE PROCEDURE facilito\_procedure()

BEGIN

DECLARE var\_id INTEGER;

DECLARE var\_paginas INTEGER;

DECLARE var\_titulo VARCHAR(255);

DECLARE var\_final INTEGER DEFAULT 0;

DECLARE cursor1 CURSOR FOR SELECT libro\_id, titulo, paginas FROM libros;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET var\_final = 1;

OPEN cursor1;

bucle: LOOP

FETCH cursor1 INTO var\_id, var\_titulo, var\_paginas;

IF var\_final = 1 THEN

LEAVE bucle;

END IF;

UPDATE libros SET paginas = var\_paginas + 10 WHERE libro\_id = var\_id;

SELECT

var\_titulo AS 'titulo',

var\_paginas AS 'Anterior',

paginas AS 'Incremento'

FROM libros WHERE libro\_id = var\_id;

END LOOP bucle;

CLOSE cursor1;

END//

DELIMITER ;

Dentro del stored procedure lo primero que hacemos es declarar todas las variables que usaremos. En este caso como trabajaremos con el título y el número de páginas de cada libro declaró tres nuevas variables. var\_id, var\_paginas, var\_titulo.

Para tener un control sobre la iteración de cada uno de los elementos en la consulta declaró una cuarta variable llamada var\_final, cuyo valor comienza en 0.

Una vez hemos declarado todas las variables procedemos a crear nuestro cursor. El cursor se creará a partir de una consulta SQL. En este caso la consulta no es demasiado compleja, sin embargo, si así lo deseamos podemos crear un cursos a partir de una consulta con joins, order, group etc...

Con la sentencia

DECLARE CONTINUE HANDLER FOR NOT FOUND SET var\_final = 1;

Indicamos que una vez todos los elementos (filas) dentro de nuestro cursor hayan sido iterados la variable var\_final tomará un nuevo valor, de 0 a 1.

Para comenzar la iteración de los elementos será necesario abrir el cursor.

OPEN cursor1;

La iteración la logramos utilizando un **Loop**, el cual será controlado a partir de la variable var\_final.

IF var\_final = 1 THEN

--Si ya no existen más elementos finalizamos el ciclo.

LEAVE bucle;

END IF;

Al nosotros usar la cláusula **Fetch** obtenemos el elemento (fila) actual de nuestra consulta y avanzamos al siguiente elemento.

En este caso como en nuestra consulta hemos obtenido 3 columnas (id, titulo, páginas) asignamos el valor de cada una de las columnas a nuestras variables. Es importante mencionar que los valores podrán ser asignados únicamente a variables del mismo tipo, no podremos asignar un varchar a una variable de tipo integer.

El orden es importante.

FETCH cursor1 INTO var\_id, var\_titulo, var\_paginas;

Una vez hemos realizado todas las tareas correspondientes y se han iterado todos los elementos (filas) el siguiente paso será cerrar el cursor.

close cursor1

Para ejecutar nuestro stored prcedure ejecutaremos la siguiente sentencia.

call facilito\_procedure();

-------------------------RESPALDO DE INFORMACIÓN------------------------

# Respaldo de información

Abrir temario playlist\_play

Lo lees en 1 Min.

#### RESPALDO DE INFORMACIÓN

En MySQL nosotros podemos realizar respaldos de nuestra información, para ello, utilizaremos la aplicación **mysqldump**.

Si queremos realizar el respaldo de una sola base de datos ejecutaremos el siguiente comando en nuestra terminal.

mysqldump base\_de\_datos > ruta/archivo\_respaldo.sql

Si lo que necesitamos es realizar el respaldo de múltiples base de datos, modificaremos nuestro comando, agregando **--databases**.

mysqldump --databases db1 db2 db3 > ruta/archivo\_respaldo.sql

Separaremos las base de datos mediante un espacio.

Si lo que necesitamos es realizar el respaldo de una o múltiples tablas el comando será el siguiente

Para una sola tabla

mysqldump base\_de\_datos tabla1 > ruta/archivo\_respaldo.sql

Para múltiples tablas

mysqldump base\_de\_datos tabla1 tabla3 > ruta/archivo\_respaldo.sql

# Asignar permisos a usuarios MySQL

Abrir temario playlist\_play

Lo lees en 2 Min.

Cómo mencionamos anteriormente: el servidor de base de datos permite que más de un usuario pueda trabajar con los recursos del servidor (registros, tablas, bases de datos, funciones, etc..).

Hasta este punto del curso, todos los ejercicios los hemos realizado utilizando el usuario **root**.

Si nosotros somos los únicos administradores del servidor (algo que muy pocas veces sucede) no deberíamos tener problemas en seguir utilizando **root**, sin embargo, si más personas trabajarán con el servidor, será necesario que generemos nuevos usuarios y asignemos los permisos pertinentes.

#### AGREGAR NUEVOS USUARIOS

Para que nosotros generemos un nuevo usuario lo primero que debemos de hacer es autenticarnos en el servidor.

mysql -u root -p -h localhost<ip>

Posteriormente debemos de generar un nuevo Usuario, Para esto ejecutamos las siguiente sentencia.

CREATE USER 'usuario'@'localhost' IDENTIFIED BY 'password';

Hasta este punto, nosotros ya podemos autenticarnos con el servidor utilizando el nuevo usuario, sin embargo, una vez autenticado las acciones que podemos hacer son mínimas, debido a que este usuario no posee los permisos necesarios para trabajar con las bases de datos.

#### ASIGNAR PERMISOS

Para poder establecer permisos, las siguiente sentencias deben de ejecutarse utilizando el usuario **root**.

Si queremos que el nuevo usuario tenga permisos de **administrador** (Todos los permisos), debemos de ejecutar la siguiente sentencia.

GRANT ALL PRIVILEGES ON \*.\* TO 'nombre\_usuario'@'localhost';

Los asteriscos indican que los permisos serán asignados a todas las bases de datos y a todas las tablas (primer asteriscos bases de datos, segundo asterisco tablas).

Si queremos asignar permisos para ciertas acciones, la sentencia quedaría de la siguiente manera. Reemplazamos ALL PRIVILEGES y colocamos las acciones que queremos asignar.

GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP

-> ON codigofacilito.\*

-> TO 'nombre\_usuario'@'localhost';

En esta ocasión estamos indicando que el nuevo usuario podrá consultar, crear, actualizar y eliminar registros, así cómo podrá crear o eliminar elementos (tablas, índices, columnas, funciones, stores, etc ...).

Todos estos permisos serán válidos únicamente en la base de datos **codigofacilito** y se aplicarán a todas las tablas.

Si queremos asignar permisos únicamente a una tabla, reemplazamos el asteriscos por el nombre de la tabla.

ejemplo codigofacilito.usuarios

Una vez hayamos finalizado con los permisos, el último paso será refrescarlos.

FLUSH PRIVILEGES;

#### PERMISOS

Aquí un listado de algunos permisos que podemos asignar.

* CREATE permite crear nuevas tablas o bases de datos.
* DROP permite eliminar tablas o bases de datos.
* DELETE permite eliminar registros de tablas.
* INSERT permite insertar registros en tablas.
* SELECT permite leer registros en las tablas.
* UPDATE permite actualizar registros en las tablas.
* GRANT OPTION permite remover permisos de usuarios.
* SHOW DATABASE Permite listar las bases de datos existentes.

#### SENTENCIAS A TENER EN CUENTA

Listado de todos los usuarios.

SELECT User FROM mysql.user;

Eliminar un usuario.

DROP USER 'usuario'@'localhost';

Remover permisos en concreto (Ejemplo update y delete).

REVOKE UPDATE, DELETE ON \*.\* FROM 'usuario'@'localhost';

Remover todos los privilegios.

REVOKE ALL PRIVILEGES ON \*.\* FROM 'usuario'@'localhost';

# Triggers Mysql

Abrir temario playlist\_play

Lo lees en 1 Min.

Un trigger, también conocido como disparador (Por su traducción al español) es un conjunto de sentencias SQL las cuales se ejecutan de forma automática cuando ocurre algún evento que modifique a una tabla. Pero no me refierón a una modificación de estructura, no, me refiero a una modificación en cuando a los datos almacenados, es decir, cuando se ejecute una sentencia **INSERT**, **UPDATE** o **DELETE**.

A diferencia de una función o un store procedure, un trigger no puede existir sin una tabla asociada.

Lo interesante aquí es que podemos programar los triggers de tal manera que se ejecuten antes o después, de dichas sentencias; Dando como resultado seis combinaciones de eventos.

* **BEFORE INSERT** Acciones a realizar antes de insertar uno más o registros en una tabla.
* **AFTER INSERT** Acciones a realizar después de insertar uno más o registros en una tabla.
* **BEFORE UPDATE** Acciones a realizar antes de actualizar uno más o registros en una tabla.
* **AFTER UPDATE** Acciones a realizar después de actualizar uno más o registros en una tabla.
* **BEFORE DELETE** Acciones a realizar antes de eliminar uno más o registros en una tabla.
* **AFTER DELETE** Acciones a realizar después de eliminar uno más o registros en una tabla.

A partir de la versión 5.7.2 de **MySQL** podemos tener la n cantidad de triggers asociados a una tabla. Anteriormente estábamos limitados a tener un máximo de seis trigger por tabla (Uno por cada combinación evento).

Podemos ver esto como una relación uno a muchos, una tabla puede poseer muchos triggers y un trigger le pertenece única y exclusivamente a una tabla.

Algo importante a mencionar es que la sentencia **TRUNCATE** no ejecutará un trigger.

#### VENTAJAS DE UTILIZAR TRIGGERS

* Con los triggers seremos capaces validar todos aquellos valores los cuales no pudieron ser validados mediante un constraints, asegurando así la integreidad de los datos.
* Los triggers nos permitirán ejecutar reglas de negocios.
* Utilizando la combinación de eventos nosotros podemos realizar acciones sumamente complejas.
* Los trigger nos permitirán llevar un control de los cambios realizados en una tabla. Para esto nos debemos de apoyar de una segunda tabla (Comúnmente una tabla log).

#### DESVENTAJAS DE UTILIZAR TRIGGERS

* Los triggers al ejecutarse de forma automática puede dificultar llevar un control sobre qué sentencias SQL fueron ejecutadas.
* Los triggers incrementa la sobrecarga del servidor. Un mal uso de triggers puede tornarse en respuestas lentas por parte del servidor.