CLASE 07

Objetos de una base de datos

**CONCEPTOS GENERALES**

* En la primera clase mencionamos que las bases de datos relacionales están compuestas por diferentes objetos.
* Estos representan la información almacenada en la DB, e integran sus características con el lenguaje de programación orientado a objetos.

**Definición**

* Una BD relacional orientada a objetos integra a la base de datos con el software elegido para el desarrollo de aplicaciones a medida.
* A su vez, una DB orientada a objetos resuelve muchas operaciones del lado del motor de la DB, agilizando la lógica de la aplicación en sí.

**TABLAS**

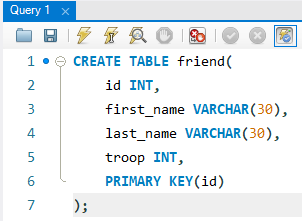
**Comportamiento de las tablas**

* Ya sabemos que las tablas se ocupan de almacenar la información en forma de registros.
* Lo hacen de forma homogénea, respetando la estructura de cada dato de un registro, el cual condice con la definición del campo que lo almacena.
* Cada tabla puede trabajar de forma autónoma, aunque en una DB relaciona, suele establecerse al menos una relación entre la tabla más importante y una o más tablas secundarias, terciarias, etcétera.
* Las claves y/o relaciones entre tablas mantienen la consistencia de datos en una DB.
* Podemos eliminar registros de una tabla principal relacionada a una tabla secundaria, pero no podremos eliminar registros de una tabla secundaria ya relacionados a una tabla principal.

**¡VAMOS A PRACTICAR LO VISTO!**

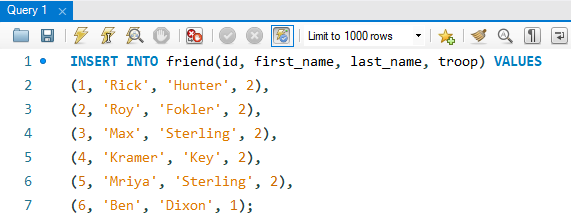
**Paso 1: crear la tabla “Friend”**

Dentro del esquema **Gammers,** en MySQL Workbench,crearemos una tabla denominada **Friend,** utilizando la sentencia **CREATE TABLE Friend**

****

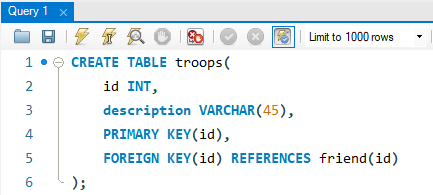
**Paso 2: agregamos algunos registros.**

Agregamos algunos registros a la tabla creada, para darle consistencia en su información. En la columna **troop**, agregamos algunos números que luego haremos coincidir con la otra tabla que debemos crear.



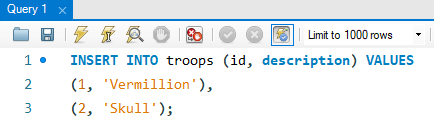
**Paso 3: creamos la tabla “Troops”**

Ahora creamos la tabla troops utilizando la sentencia **CREATE TABLE** troops.



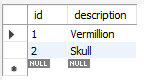
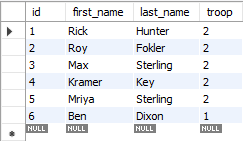
**Paso 4: agregamos algunos registros.**

Hagamos coincidir los números de la columna **id** de la tabla friend, con aquellos números que agregamos en la columna **troops**.

****

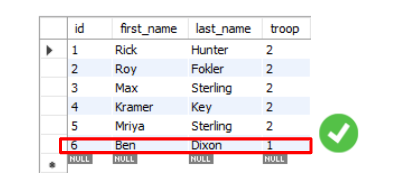
**Ejemplos de tablas con relación**

Disponemos ya de dos tablas: (**friend** y **troops**). Ambas están definidas de forma independiente, con una relación lógica entre dos campos, aunque esta no ha sido definida de forma efectiva en el **Diagrama E-R**.



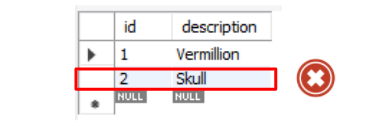
Es posible y lógico que podamos eliminar uno o más registros en la tabla **friend**, a pesar de que están relacionados con la tabla **troops**.

*¿Saben por qué?*

**

Ahora, no es posible, o lógico, que podamos eliminar uno o más registros de la tabla **troops**.

*¿Se imaginan por qué?*

**

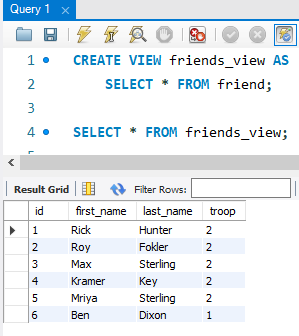
**VISTAS**

**Definición de vistas**

Una Vista es un **conjunto de resultados de una tabla o más tablas** de un BD. Podemos definirlas también como **“una tabla virtual”** que se genera a partir de una o más tablas de una BD relacional.

Están compuestas por la misma estructura que una tabla: **filas** y **columnas**.

Pueden ser almacenadas con el mismo nombre de una tabla, o si se combinan dos o más tablas en la vista, se suele definir combinando ambos nombres. Aunque se utilizan para mostrar datos combinando dos o más tablas, en lugar de tener que elaborar la consulta, las vistas también permiten la **inserción**, **eliminación** y **actualización** de los registros que muestran. Aunque, esto último, queda condicionado a las restricciones de estructura de la Vista cuando es creada. Puedes ejecutar una **Vista** tal como invocas a una **Tabla** convencional. Incluso puedes sumarle condicionales mediante la sentencia **WHERE**, sobre la información que mostrará.



**Uso de Tablas versus Vistas**

* Si somos responsables de la Administración y Mantenimiento de una BD en un equipo de trabajo IT, construyamos siempre **Vistas** que faciliten el acceso a la información de las tablas, por parte de los desarrolladores de software.
* Es la opción más práctica y segura para que otros accedan a la información de dicha BD.

**FUNCIONES**

**Definición y uso de las funciones**

La clase anterior aprendimos sobre **Funciones Escalares**. Ahora nos toca conocer las funciones de usuario en Mysql.

Estas **permiten crear una rutina específica** que procese determinados parámetros, y retorna un resultado determinado.

Las funciones de usuario utilizan el lenguaje **SQL** y permiten incluir sentencias propias creadas por el desarrollador, como también combinar funciones SQL preexistentes.

Podemos combinar estas últimas para crear resultados personalizados que las **funciones integradas** no puedan resolver.

**Estructura:**

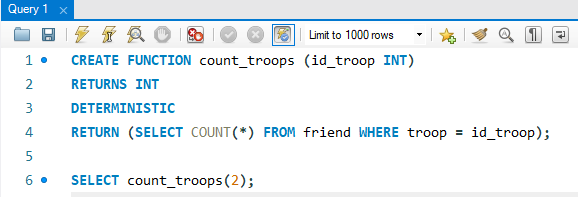
Aceptan solo parámetros de entrada:

* Deben **retornar** siempre **un valor** con un **tipo de dato definido**
* Pueden **usarse en el contexto de una sentencia** SQL
* Retornan un valor individual y **nunca un conjunto de registros**

Si desarrollas en algún lenguaje de programación encontrarás un parecido con las funciones personalizadas que creas en cualquier otro programa.

**Ejemplos de uso:**

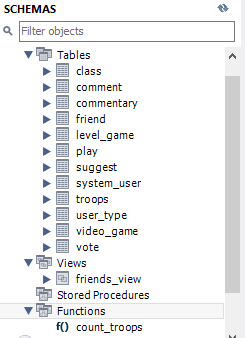
Puedes crear, por ejemplo, una función que retorne la cantidad de integrantes que tiene una **troop** pasando como parámetro el identificador de una **troop**.



**Disponibilidad:**

Las funciones se almacenan en el apartado homónimo de los objetos de la BD.

Solo están disponibles en el **Schema** donde fueron creadas.



**TIGGERS**

**Definición**

Un trigger es una aplicación almacenada (**stored program**), creada para ejecutarse cuando uno o más eventos ocurran en nuestra base de datos.

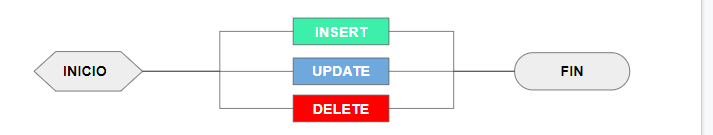
El trigger se dispara cuando ocurre un comando **INSERT**, **UPDATE** o **DELETE**, ejecutando un bloque de instrucciones que proteja o prepare la información de las tablas.

La principal tarea de un trigger es la de mantener la integridad de una bb.dd. aplicando los siguientes casos de uso:

* **Validar la información**
* **Calcular atributos derivados**
* **Seguir movimientos y Logs**

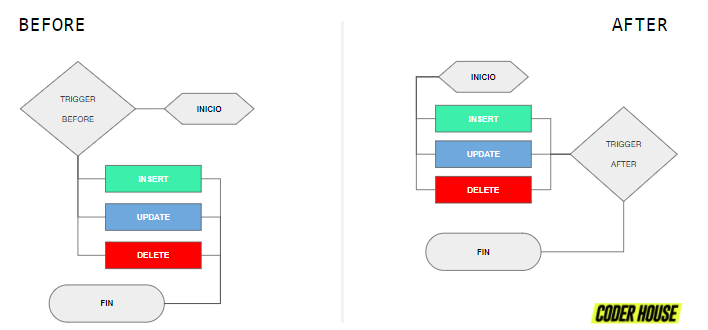
Entre otras tantas necesidades que pueda haber, y que requieran ejecutar una acción implícita sobre los registros de una tabla.

**Esquema tradicional**

****

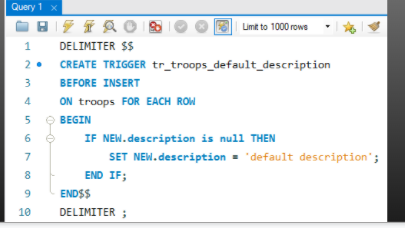
Este es un esquema tradicional de trabajo, cuando se ejecuta un comando DML y cualquier acción adicional a realizar sobre esta acción debe ser controlada por un programador.

**Esquema con triggers**

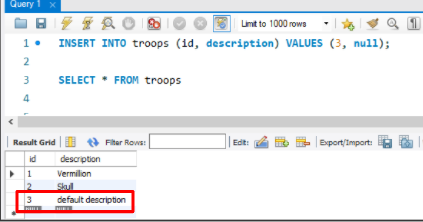
****

**Esquema con interacción de un trigger**

Ejemplo: el trigger siguiente se ejecuta cuando se realiza una inserción sobre la tabla **troops**. Si la **description** es null, el trigger altera su valor ingresando **‘default description’**

****

Ejemplo para testear el disparador:

****

**¡EJERCITEMOS UN POCO LA MENTE!**

**Piensa en un esquema BD**

Entendiendo la lógica de los Triggers y cómo aplicarlos, pensemos un esquema (*o dos*), donde debamos aplicar un Trigger utilizando **AFTER** y un Trigger utilizando **BEFORE**.

Junto con el esquema a pensar, debemos contemplar también la o las tablas involucradas que serán afectadas y/o escuchadas por el trigger.

**STORED PROCEDURES**

**Definición de procedimientos almacenados**

Los **procedimientos almacenados** (*stored procedures*), son parte de Mysql desde su versión 5.0.

Conforman un conjunto de instrucciones escritas en **Transact-SQL** para realizar una tarea determinada pudiendo ser esta una operación simple de resolver o una serie encadenada de tareas complejas.

Son especialmente útiles cuando:

* varias aplicaciones deben realizar una misma consulta
* existen entornos donde la seguridad es importante

Los S.P. se almacenan en el apartado homónimo del esquema visible en **Mysql Workbench**.

**Permisos**

La ejecución de un **Stored Procedure** no está disponible para cualquier usuario. Es necesario que el perfil de éste, en el esquema de base de datos, tenga habilitado el permiso de ejecución (Execute).

Un **SP** puede contener y ejecutar en su interior, cualquier consulta del tipo DML. Incluso puede combinar varias de estas, aplicándolas en diferentes tablas.

**Estructura**

Se inicia con el comando CREATE PROCEDURE nombre\_sp. Recibe parámetros del tipo IN, OUT e INOUT y soporta tipos de datos válidos.

Y, a su vez, los SP pueden tener dos tipos de denominación:

* determinista
* no determinista

**Denominación**

**determinista:** se los denomina así cuando el SP produce el mismo resultado sobre los mismos parámetros de entrada

**no determinista:** cuando produce resultados diferentes a los tipos de parámetros de entrada

Finalmente, un SP se ejecuta siempre del lado del servidor, y devuelve los datos filtrados y procesados al cliente que los solicitó.

**Ejemplo:** SP que retorna las **troops** que coinciden con una descripción parametrizada.

