



Clase 07. SQL

# ***OBJETOS DE UNA BASE DE DATOS***

***RECUERDA PONER A GRABAR LA  
CLASE***





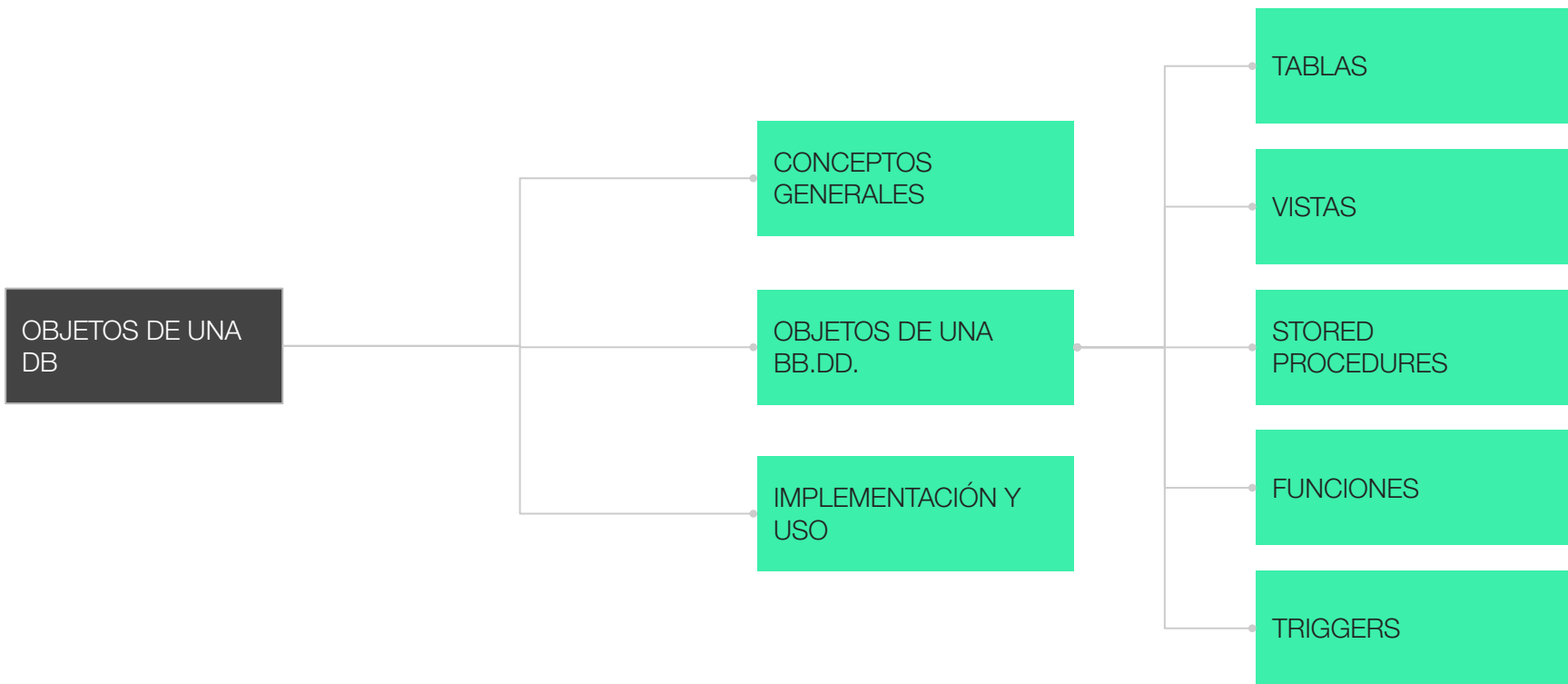
## ***OBJETIVOS DE LA CLASE***

- Reconocer los objetos que componen una base de datos.
- Identificar para qué y en qué momento se debe implementar cada uno de ellos.

# ***MAPA DE CONCEPTOS***

# MAPA DE CONCEPTOS CLASE 7

¡Para  
recordar!



Realicemos un repaso general por todos los objetos que componen una DB.

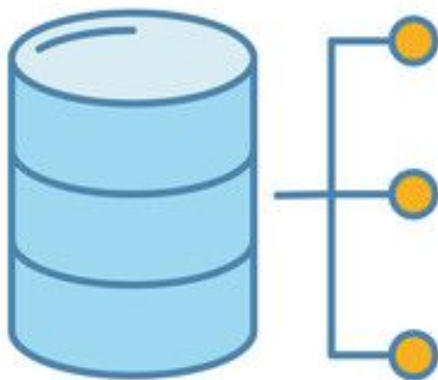
Reconozcamos a los mismos, y entendamos cómo y para qué se usan, además de cuándo se deben implementar.

# ***OBJETOS DE UNA BASE DE DATOS***

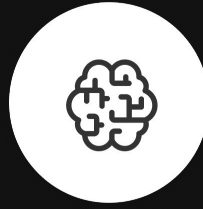
# ***CONCEPTOS GENERALES***



# OBJETOS DE UNA BASE DE DATOS



- 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.



***¡PARA PENSAR!***

*¿Cuáles eran los objetos que componen una base de datos relacional?*

# OBJETOS DE UNA BASE DE DATOS

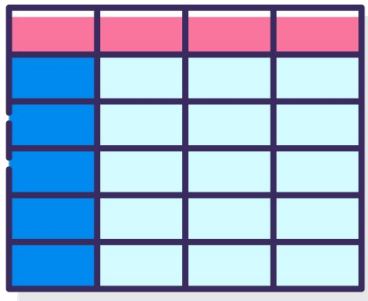
## ***Definición***

- Una DB 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***

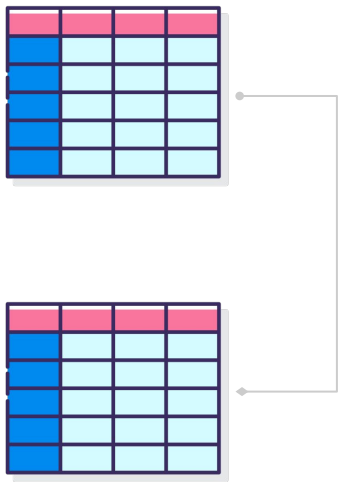
***CODER HOUSE***

# ***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.

# ***Tablas relacionales***





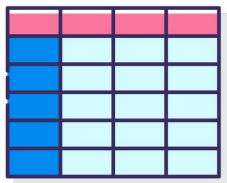




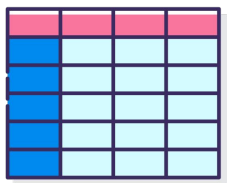


Cada tabla puede trabajar de forma autónoma, aunque en una DB relacional suele establecerse al menos una relación entre la tabla más importante y una o más tablas secundarias, terciarias, etcétera.

# ***Tipos de claves***



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 UN POCO!***



## ***PASO 1: crear la tabla 'Friend'***

Ejemplo  
en vivo



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

```
Query 1 x
1 • CREATE TABLE friend(
2     id INT,
3     first_name VARCHAR(30),
4     last_name VARCHAR(30),
5     troop INT,
6     PRIMARY KEY(id)
7 );
```

## PASO 2: agreguemos algunos registros

Ejemplo  
en vivo



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.

```
Query 1 x
[Icons: Folder, Save, Run, Copy, Paste, Find, Undo, Redo, Refresh, Close, Limit to 1000 rows, Star, Share, Search, Print, Full Screen]
1 • INSERT INTO friend(id, first_name, last_name, troop) VALUES
2   (1, 'Rick', 'Hunter', 2),
3   (2, 'Roy', 'Fokler', 2),
4   (3, 'Max', 'Sterling', 2),
5   (4, 'Kramer', 'Key', 2),
6   (5, 'Mriya', 'Sterling', 2),
7   (6, 'Ben', 'Dixon', 1);
```

## ***PASO 3: creamos la tabla 'Troops'***

Ejemplo  
en vivo



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

```
Query 1 x
1 CREATE TABLE troops(
2     id INT,
3     description VARCHAR(45),
4     PRIMARY KEY(id),
5     FOREIGN KEY(id) REFERENCES friend(id)
6 );
```

## PASO 4: agregamos algunos registros

Ejemplo  
en vivo



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

```
Query 1 x
1 • INSERT INTO troops (id, description) VALUES
2 (1, 'Vermillion'),
3 (2, 'Skull');
```

# Ejemplo de tablas con relación

Ejemplo  
en vivo



	id	first_name	last_name	troop
▶	1	Rick	Hunter	2
	2	Roy	Fokler	2
	3	Max	Sterling	2
	4	Kramer	Key	2
	5	Mriya	Sterling	2
	6	Ben	Dixon	1
★	NULL	NULL	NULL	NULL

	id	description
▶	1	Vermillion
	2	Skull
★	NULL	NULL

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**.

# Ejemplo de tablas con relación

Ejemplo  
en vivo



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?*

	id	first_name	last_name	troop
▶	1	Rick	Hunter	2
	2	Roy	Fokler	2
	3	Max	Sterling	2
	4	Kramer	Key	2
	5	Mriya	Sterling	2
	6	Ben	Dixon	1
*	NULL	NULL	NULL	NULL



# Ejemplo de tablas con relación

Ejemplo  
en vivo



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

*¿Se imaginan por qué?*

	id	description
▶	1	Vermillion
	2	Skull
✱	NULL	NULL



# Normalización de datos



- Es importante definir siempre de forma lógica a las tablas que normalizan la información en una DB.

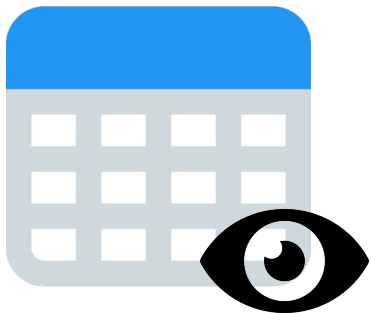
De esta forma evitaremos que los registros queden huérfanos en el caso que se eliminen de una tabla secundaria, como es en este caso la tabla **troops**.



***VISTAS***

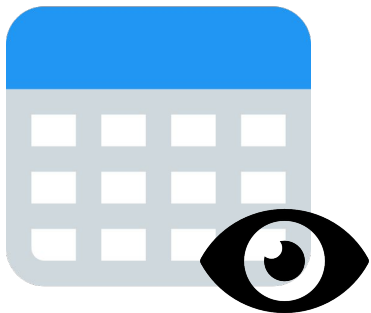
# ***DEFINICIÓN DE VISTAS***

# ***VISTAS: definición***



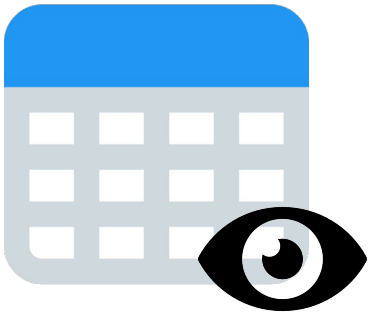
- Una Vista es un **conjunto de resultados de una tabla o más tablas** de un DB.
- Podemos definirlas también como “**una tabla virtual**”, que se genera a partir de una o más tablas de una BD relacional.

## ***VISTAS: definición de su nombre***



- 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.

# ***VISTAS: tipo de interacción***



- 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.

# VISTAS: ejecución

Ejemplo  
en vivo



Puedes ejecutar una **Vista** tal como invocas a una **Tabla** convencional.

Incluso puedes sumarle condicionales mediante la sentencia **WHERE**, sobre la información que mostrará.

```
Query 1 x
1 • CREATE VIEW friends_view AS
2     SELECT * FROM friend;
3
4 • SELECT * FROM friends_view;
```

Result Grid

	id	first_name	last_name	troop
▶	1	Rick	Hunter	2
	2	Roy	Fokler	2
	3	Max	Sterling	2
	4	Kramer	Key	2
	5	Mriya	Sterling	2
	6	Ben	Dixon	1

# Uso de Tablas versus Vistas

¡Para  
recordar!



- Si somos responsables de la Administración y Mantenimiento de una DB 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 DB.



***BREAK***

**¡5/10 MINUTOS Y VOLVEMOS!**



**Si todavía no conoces nuestros servicios  
de desarrollo profesional y querés saber  
más hacé clic [aquí](#).**

**¡5/10 MINUTOS Y VOLVEMOS!**

***CODER HOUSE***

# ***FUNCIONES***

# ***DEFINICIÓN Y USO DE LAS FUNCIONES***

# ***FUNCIONES: definición***



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 retornar un resultado determinado.

# ***FUNCIONES: definición***



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.

# ***FUNCIONES: estructura***

Aceptan sólo 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.

# ***FUNCIONES; ejemplo de uso***

Ejemplo  
en vivo



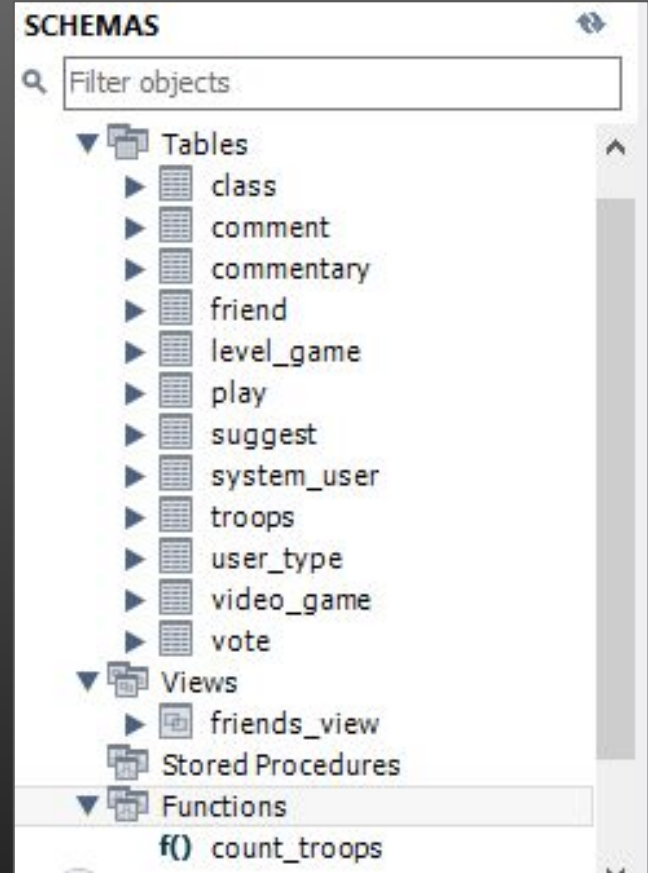
Puedes crear, por ejemplo, una función que retorne la cantidad de integrantes que tiene una **troop**, pasando como parámetro el identificador de una **troop**.

```
Query 1 x
[Icons] Limit to 1000 rows [Icons]
1 • CREATE FUNCTION count_troops (id_troop INT)
2   RETURNS INT
3   DETERMINISTIC
4   RETURN (SELECT COUNT(*) FROM friend WHERE troop = id_troop);
5
6 • SELECT count_troops(2);
```

# ***FUNCIONES: disponibilidad***

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

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

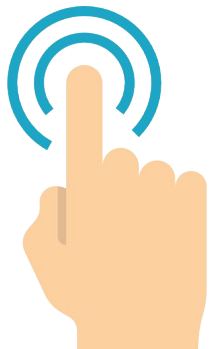




# ***TRIGGERS***

# ***DEFINICIÓN DE TRIGGER***

## ***TRIGGER: definición***



Un trigger es una aplicación almacenada (**stored program**), creado 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.

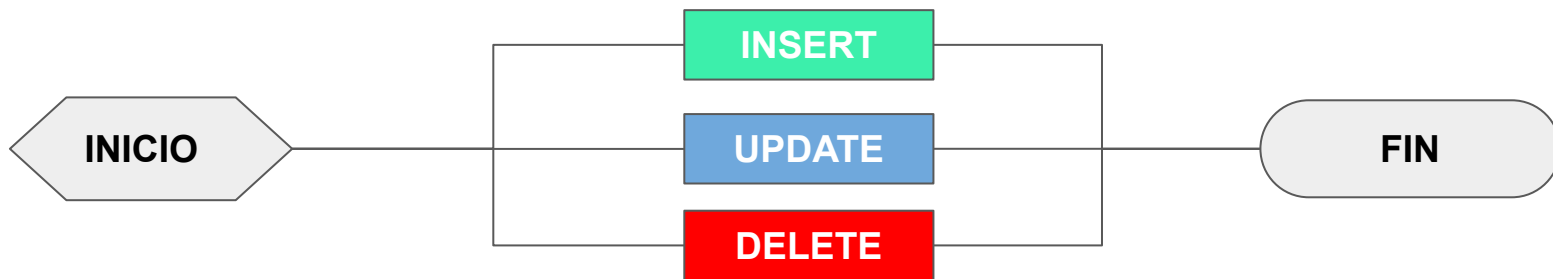
# ***TRIGGER: definición***

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.

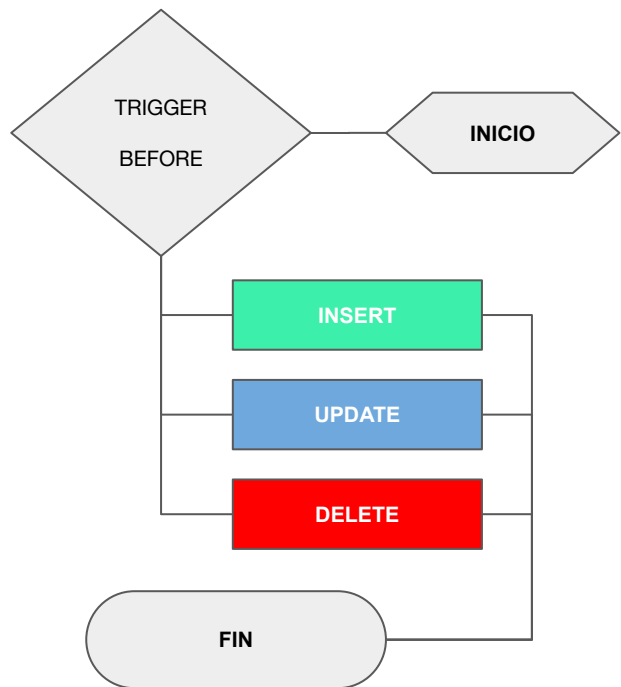
## ***TRIGGER: esquema tradicional***



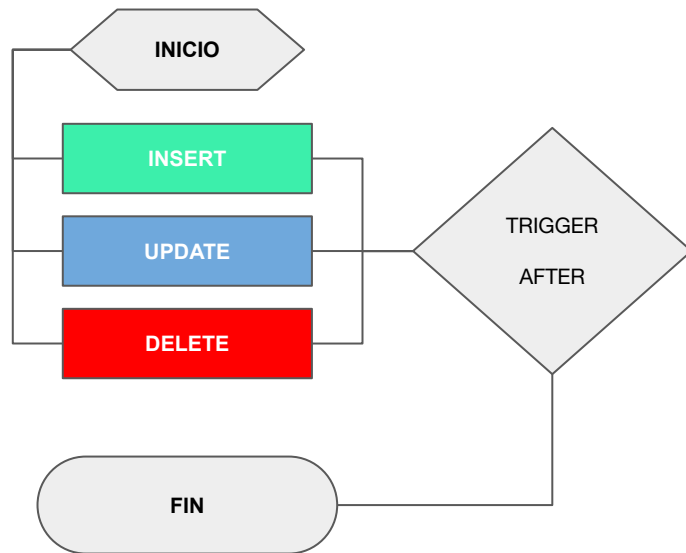
Este es un esquema tradicional de trabajo, cuando se ejecuta un comando DML y cualquier acción adicional a realizar sobre esta acción debe ser controlada por un programador.

# ***TRIGGER: esquema con triggers***

BEFORE



AFTER



# Esquema con interacción de un trigger

Ejemplo  
en vivo



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'**.

```
Query 1 x
Limit to 1000 rows
1 DELIMITER $$
2 • CREATE TRIGGER tr_troops_default_description
3 BEFORE INSERT
4 ON troops FOR EACH ROW
5 BEGIN
6     IF NEW.description is null THEN
7         SET NEW.description = 'default description';
8     END IF;
9 END$$
10 DELIMITER ;
```

# Esquema con interacción de un trigger

Ejemplo  
en vivo



Ejemplo para testear el disparador:

Query 1 x

Limit to 1000 rows

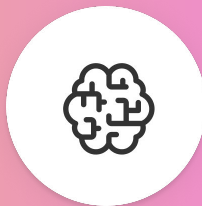
```
1 • INSERT INTO troops (id, description) VALUES (3, null);
2
3 SELECT * FROM troops
4
5
```

Result Grid

	id	description
▶	1	Vermillion
	2	Skull
	3	default description

**CODER HOUSE**





***¡EJERCITEMOS UN POCO LA MENTE!***

## ***Piensa un esquema de DB***

¡Para  
pensar!



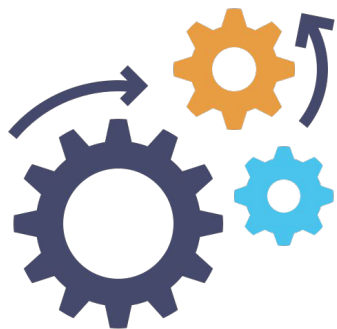
Entendiendo la lógica de los Triggers y cómo aplicarlos, pensemos un esquema (o *dos*) donde debamos aplicar un Trigger utilizando **AFTER**, y uno utilizando **BEFORE**.

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***

## ***STORED PROCEDURE: definición***



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.

# ***STORED PROCEDURE: definición***

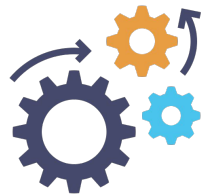


Son especialmente útiles cuando:

- 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**.

## ***STORED PROCEDURE: 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.

## ***STORED PROCEDURE: 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.



## ***STORED PROCEDURE: 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.

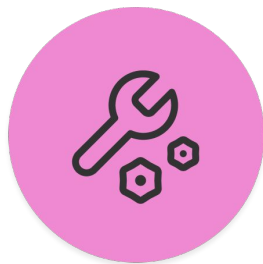
# STORED PROCEDURE: estructura

¡Para  
pensar!



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

```
Query 1 x
[Icons] Limit to 1000 rows
1 DELIMITER $$
2 • CREATE PROCEDURE country_hos (d VARCHAR(45))
3 BEGIN
4     SELECT *
5     FROM troops
6     WHERE description = d;
7 END $$
8 DELIMITER ;
9
10 • CALL country_hos('Skull');
```



# ***IMPLEMENTACIÓN Y USO DE UN DIAGRAMA E-R***

Tiempo estimado: 15 minutos

# ***IMPLEMENTACIÓN Y USO DE UN DIAGRAMA E-R***

Desafío  
generico



Recuperando el **Diagrama E-R que realizamos en la Clase 2**, y con los conocimientos adquiridos hasta ahora, crear una base de datos nueva, integrando a la misma las tablas realizadas en este diagrama E-R.

Tengan presente que cada una de las tablas deberá tener, como mínimo, unos 3 campos. Si tienen o desean agregarles más, mejor aún.

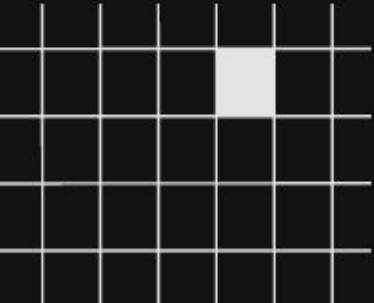
***¿PREGUNTAS?***





# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Objetos de una DB.
  - Tablas y Vistas.
  - Stored Procedures.
  - Funciones.
  - Triggers.
- 



***OPINA Y VALORA ESTA CLASE***

***#DEMOCRATIZANDO LA EDUCACIÓN***

***CODER HOUSE***