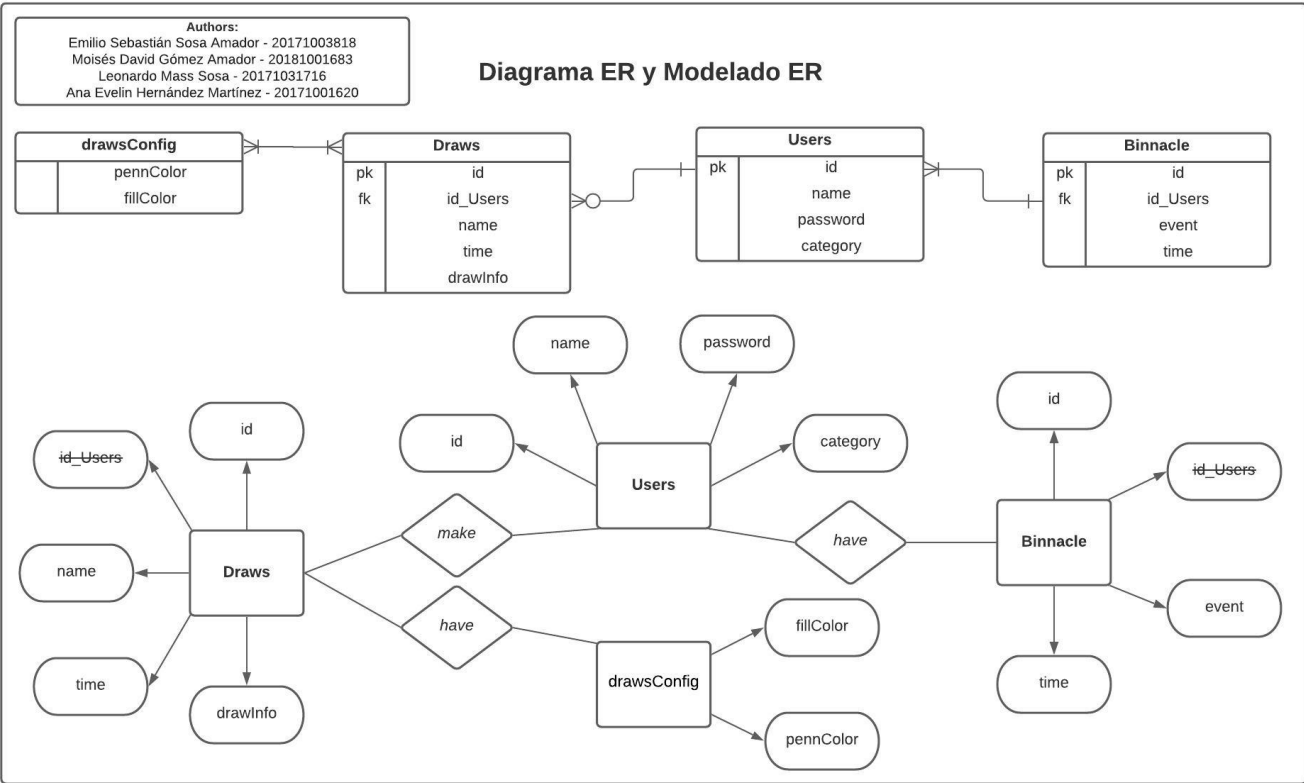


Base de Datos A - Análisis

Al realizar un pequeño y minucioso estudio de los elementos que se deben representar en esta base de datos, se logró distinguir una serie de atributos necesarios para lograr un mejor desempeño en el programa.

A continuación se presenta una explicación detallada de todos los elementos que se encuentran en la Base de Datos A del archivo *DDS_version_1.sql*.

Modelado ER y Diagrama ER



Creados bajo una percepción general de las entidades, atributos y relaciones que componen la Base de Datos - A, se definen los siguientes:

Entidades y sus Atributos:

Entidad	Atributos	Entidad	Atributos	Entidad	Atributos	Entidad	Atributos
User	id	Draws	id	Binnacle	id	drawsConfig	pennColor
	name		userId		userId		fillcolor
	password		name		time		
	category		time		event		
			drawInfo				

Podemos definir el conjunto de relaciones de la siguiente manera:

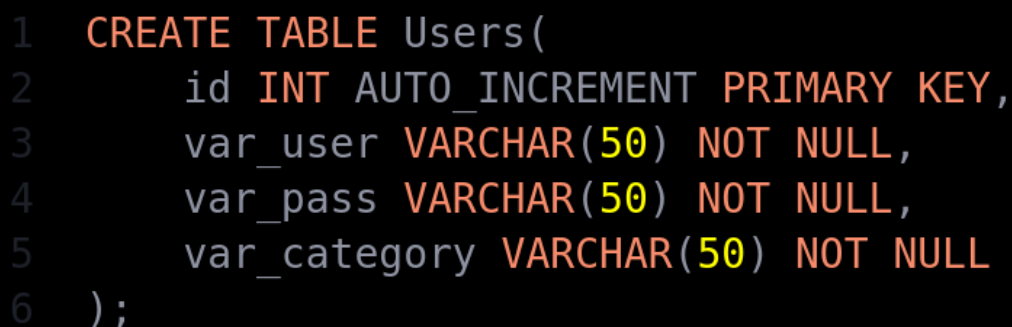
- **Relación: *Users-Draws*:** Todas las relaciones de la forma Users-Draws, permiten obtener la información de los usuarios y sus respectivos dibujos. Esta relación es de cero a muchos puesto que un usuario puede ser ingresado a la base de datos pero no necesariamente tener algún dibujo realizado.
- **Relación: *Users-Binnacle*:** Todas las relaciones de la forma Users-Binnacle, permiten obtener la información de los usuarios y las actividades que desarrollan dentro del programa y guardarlas en la bitácora que representa el historial de todos los eventos. Esta relación es de uno a muchos porque es necesario que exista un usuario que interactúe con el programa para que se pueda registrar todo lo que hace.
- **Relación: *Draws-drawsConfig*:** Todas las relaciones de la forma Draws-drawsConfig, permiten obtener la información de los dibujos en cuanto a sus valores de configuración. Esta relación es de uno a muchos puesto que debe existir un dibujo para poder realizar estos cambios en la configuración.

NOTA: *Esta es la versión 3.0 del análisis de la base de datos.*

Análisis de las Tablas y sus Atributos

Tabla *Users*

Esta tabla representa los usuarios ingresados a la base de datos. Y que serán los encargados de la creación de dibujos, así como también de las actividades de modificación para el caso de los administradores.



```
1 CREATE TABLE Users(  
2     id INT AUTO_INCREMENT PRIMARY KEY,  
3     var_user VARCHAR(50) NOT NULL,  
4     var_pass VARCHAR(50) NOT NULL,  
5     var_category VARCHAR(50) NOT NULL  
6 );
```

Se definen cuatro atributos para la tabla usuario los cuales son:

- ***id*:** Un atributo de tipo de dato entero con valor autoincremental, definido como la llave primaria de la tabla; este dato es único para cada usuario que se ingresa en la base de datos y nos permite

identificarlo en cualquier otra instancia necesaria.

- **var_user:** Atributo de tipo de dato varchar no nulo, el cual almacena los nombre de los usuarios; estos nombres son ingresados por el mismo usuario o por el usuario administrador.
- **var_pass:** Atributo de tipo de dato varchar no nulo, en el que se almacena la contraseña de cada usuario en la tabla; estas contraseñas son definidas por el mismo usuario o por el usuario administrador.
- **var_category:** Atributo de tipo de dato varchar no nulo, en el que se define si un usuario es o no administrador.

Estos atributos son las propiedades de cada usuario necesarias para su creación y validación dentro del programa y la base de datos, los cuales solo pueden ser modificados por el administrador del programa.

Tabla *Draws*

Esta tabla representa los dibujos creados por los usuarios que pasan a la base de datos como archivos JSON.

```
1  CREATE TABLE Draws(  
2      id INT AUTO_INCREMENT PRIMARY KEY,  
3      userId INT NOT NULL,  
4      var_name VARCHAR(50) NOT NULL,  
5      tim_time TIMESTAMP DEFAULT NOW(),  
6      jso_drawInfo JSON NOT NULL,  
7  
8      FOREIGN KEY (userId)  
9          REFERENCES Users(id)  
10         ON DELETE CASCADE  
11         ON UPDATE CASCADE  
12 );
```

Se definen cinco atributos para la tabla *Draws* los cuales son:

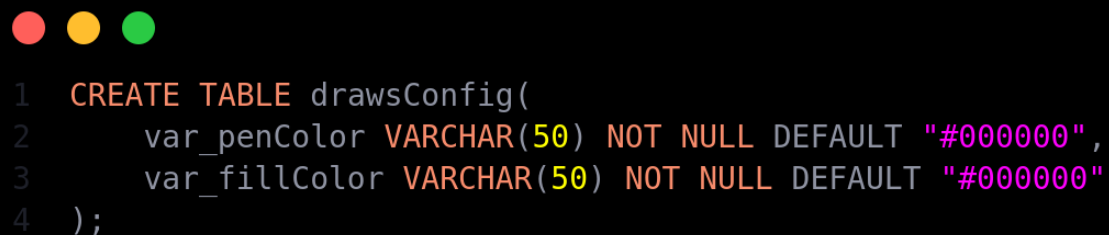
- **id:** Un atributo de tipo de dato entero con valor autoincremental, definido como la llave primaria de la tabla; este dato es único para cada dibujo generado por los usuarios, que se ingresa en la base de datos y nos permite identificarlo en cualquier otra instancia necesaria.

- **userId:** Atributo de tipo de dato entero no nulo, definido en la tabla users y llave foránea de la tabla draws; este dato permite vincular a un usuario con sus dibujos en la tabla draws ya que el usuario puede realizar uno o varios dibujos.
- **var_name:** Atributo de tipo de dato varchar no nulo, que corresponde al nombre del dibujo creado por el usuario, este dato es asignado por el usuario ya sea un usuario normal o el administrador.
- **tim_time:** Atributo de tipo de dato timestamp no nulo, que guarda el registro de la hora y fecha instantánea en la que fue creado el dibujo por defecto.
- **jso_drawInfo:** Atributo de tipo de dato JSON no nulo, que se encarga de manejar la información de los dibujos que son agregados a la base de datos.

Estos atributos son las propiedades de cada dibujo necesarias para su creación y validación dentro del programa y la base de datos, los cuales solo pueden ser modificados por el administrador del programa.

Tabla *drawsConfig*

Esta tabla representa las configuraciones con respecto al color para los dibujos creados por los usuarios.



```
1 CREATE TABLE drawsConfig(  
2     var_penColor VARCHAR(50) NOT NULL DEFAULT "#000000",  
3     var_fillColor VARCHAR(50) NOT NULL DEFAULT "#000000"  
4 );
```

Se definen tres atributos para la tabla drawsConfig los cuales son:

- **var_pennColor:** Atributo de tipo varchar no nulo, que por defecto es el color negro y que permite saber la configuración de color de lápiz que está utilizando un usuario en su dibujo.
- **var_fillColor:** Atributo de tipo varchar no nulo, que por defecto es el color negro y que permite saber la configuración de color de fondo que está utilizando un usuario en su dibujo.

Estos atributos son las propiedades de la configuración del dibujo necesarias para su creación y validación dentro del programa y la base de datos, los cuales solo pueden ser modificados por el administrador del programa.

Tabla *Binnacle*

Esta tabla representa la bitácora que funciona como módulo de registro de actividades dentro del programa.

```
1 CREATE TABLE Binnacle(  
2     id INT AUTO_INCREMENT PRIMARY KEY,  
3     userId INT NOT NULL,  
4     tex_event TEXT NOT NULL,  
5     tim_time TIMESTAMP DEFAULT NOW(),  
6  
7     FOREIGN KEY (userId)  
8         REFERENCES Users(id)  
9         ON DELETE CASCADE  
10        ON UPDATE CASCADE  
11 );
```

Se definen cinco atributos para la tabla Binnacle los cuales son:

- **id:** Un atributo de tipo de dato entero con valor autoincremental, definido como la llave primaria de la tabla; este dato es único para cada bitácora, que se ingresa en la base de datos y nos permite identificar el usuario y sus respectivos dibujos en cualquier otra instancia necesaria.
- **userId:** Atributo de tipo de dato entero no nulo, definido en la tabla users y llave foránea de la tabla binnacle; este dato permite vincular a un usuario con sus distintas acciones y estas serán guardadas como parte de un historial de trabajo en la bitácora.
- **tex_event:** Atributo de tipo de dato text no nulo, que se encarga de determinar cuál es la acción realizada dentro del programa que puede variar entre la creación, modificación y eliminación de usuarios o dibujos.
- **tim_time:** Atributo de tipo de dato timestamp no nulo, que guarda el registro de la hora y fecha instantánea en la que fue realizado algún evento.

Estos atributos son las propiedades de la bitácora necesarias para su creación y validación dentro del programa y la base de datos, los cuales no son modificables por ningún usuario del programa.

La bitácora es capaz de guardar todas las acciones del usuario, incluyendo entre ellas la autenticación, visualización, creación, modificación, eliminación de dibujos, configuración de colores y usuarios. Estas actividades son las que realizan todos los usuarios sin importar la categoría.

Análisis de Triggers

Para la creación de los trigger se utilizan algunas funciones de sql como:

- **CURRENT_USER():** Se encarga de devolver el nombre del contexto suplantado, que para este proyecto se solicita con el id del usuario y lo que retorna es el nombre del mismo.
- **SUBSTRING_INDEX():** Es una función que devuelve una subcadena de una cadena antes de que ocurra un número específico de delimitadores. Para este caso el delimitador es el @(arroba).

NOTA: Se utiliza el delimitador "\$\$" (dos signos de dólar) para especificar que las sentencias terminarán con el mismo; lo que nos sirve para pasar un trigger como una sola sentencia. Y al final de los trigger se vuelve a declarar como delimitador ";" (punto y coma) para las próximas sentencias de ejecución.

1.- Trigger bin_addUser

```
1 DELIMITER $$
2
3 DROP TRIGGER IF EXISTS bin_addUser $$
4 CREATE TRIGGER bin_addUser AFTER INSERT ON Users
5     FOR EACH ROW
6     BEGIN
7         INSERT INTO Binnacle(userId, tex_event) VALUES(
8             (SELECT Us.id
9              FROM Users Us
10             WHERE ((SUBSTRING_INDEX(CURRENT_USER(), "@",1))) = Us.var_user),
11             "Inserción de Usuario"
12         );
13     END $$
```

Se elimina el trigger *bin_addUser* si es que este existe y posteriormente se crea; este trigger trabaja con la tabla *Users* y la acción que realiza es que después de insertar en la tabla *Users* a un usuario, agregar el evento al historial de actividades que se encuentra en la tabla *Binnacle*.

El evento que desempeña un usuario al agregarse al programa, en este caso en específico al ser agregado a la base de datos, se hace con los valores de nombre del usuario (*var_user*) obteniendolo de la tabla *Users* e identificandolo con su id y el evento que en este caso se define como *"Inserción de Usuario"*.

2.- Trigger bin_deleteUser

```
1 DROP TRIGGER IF EXISTS bin_deleteUser $$
2 CREATE TRIGGER bin_deleteUser AFTER DELETE ON Users
3 FOR EACH ROW
4 BEGIN
5     INSERT INTO Binnacle(userId, tex_event) VALUES(
6         (SELECT Us.id
7          FROM Users Us
8          WHERE ((SUBSTRING_INDEX(CURRENT_USER(), "@",1))) = Us.var_user),
9         "Eliminación de Usuario"
10    );
11 END $$
```

Se elimina el trigger *bin_deleteUser* si es que este existe y posteriormente se crea; este trigger trabaja con la tabla *Users* y la acción que realiza es que después de eliminar en la tabla *Users* al usuario indicado, agrega el evento al historial de actividades que se encuentra en la tabla *Binnacle*.

Para este caso en específico al ser eliminado de la base de datos se debe identificar con los valores de nombre del usuario (*var_user*) obteniendolo de la tabla *Users* e identificandolo con su id y el evento que en este caso se define como *"Eliminación de Usuario"*.

3.- Trigger *bin_updateUser*

```
1 DROP TRIGGER IF EXISTS bin_updateUser $$
2 CREATE TRIGGER bin_updateUser AFTER UPDATE ON Users
3 FOR EACH ROW
4 BEGIN
5     INSERT INTO Binnacle(userId, tex_event) VALUES(
6         (SELECT Us.id
7          FROM Users Us
8          WHERE ((SUBSTRING_INDEX(CURRENT_USER(), "@",1))) = Us.var_user),
9         "Modificación de Usuario"
10    );
11 END $$
```

Se elimina el trigger *bin_updateUser* si es que este existe y posteriormente se crea; este trigger trabaja con la tabla *Users* y la acción que realiza es que después de actualizar en la tabla *Users* al usuario indicado, agrega el evento al historial de actividades que se encuentra en la tabla *Binnacle*.

Al ser actualizado en la base de datos se debe identificar con los valores de nombre del usuario (*var_user*) obteniendolo de la tabla *Users* e identificandolo con su id y el evento que en este caso se define como *"Modificación de Usuario"*.

4.- Trigger *bin_addDraw*

```
1 DROP TRIGGER IF EXISTS bin_addDraw $$
2 CREATE TRIGGER bin_addDraw AFTER INSERT ON Draws
3 FOR EACH ROW
4 BEGIN
5     INSERT INTO Binnacle(userId, tex_event) VALUES(
6         (SELECT Us.id
7          FROM Users Us
8          WHERE ((SUBSTRING_INDEX(CURRENT_USER(), "@",1))) = Us.var_user),
9         "Inserción de Dibujo"
10    );
11 END $$
```

Se elimina el trigger *bin_addDraw* si es que este existe y posteriormente se crea; este trigger trabaja con la tabla *Draws* y la acción que realiza es que después de insertar en la tabla *Draws* un dibujo, agrega el evento al historial de actividades que se encuentra en la tabla *Binnacle*.

El evento que desempeña un usuario al agregar un dibujo al programa, en este caso en específico al agregarlo a la base de datos, se hace con los valores de nombre del usuario (*var_user*) obteniendolo de la tabla *Users* e identificandolo con su id y el evento que en este caso se define como *"Inserción de Dibujo"*.

5.- Trigger *bin_deleteDraw*

```
1 DROP TRIGGER IF EXISTS bin_deleteDraw $$
2 CREATE TRIGGER bin_deleteDraw AFTER DELETE ON Draws
3 FOR EACH ROW
4 BEGIN
5
6     INSERT INTO Binnacle(userId, tex_event) VALUES(
7         (SELECT Us.id
8          FROM Users Us
9          WHERE ((SUBSTRING_INDEX(CURRENT_USER(), "@",1))) = Us.var_user),
10        "Eliminación de Dibujo"
11    );
12 END $$
```

Se elimina el trigger *bin_deleteDraw* si es que este existe y posteriormente se crea; este trigger trabaja con la tabla *Draws* y la acción que realiza es que después de eliminar en la tabla *Draws* al dibujo indicado, agrega el evento al historial de actividades que se encuentra en la tabla *Binnacle*.

Para este caso en específico al ser eliminado de la base de datos un dibujo, se debe identificar con los valores de nombre del usuario (*var_user*) obteniendolo de la tabla *Users* e identificandolo con su id y el evento que en este caso se define como *"Eliminación de Dibujo"*.

6.- Trigger *bin_modifyDraw*


```
1 DROP TRIGGER IF EXISTS bin_modifyDraw $$
2 CREATE TRIGGER bin_modifyDraw AFTER UPDATE ON Draws
3 FOR EACH ROW
4 BEGIN
5
6     INSERT INTO Binnacle(userId, tex_event) VALUES(
7         (SELECT Us.id
8          FROM Users Us
9          WHERE ((SUBSTRING_INDEX(CURRENT_USER(), "@",1))) = Us.var_user),
10        "Modificación de Dibujo"
11    );
12 END $$
```

Se elimina el trigger *bin_modifyDraw* si es que este existe y posteriormente se crea; este trigger trabaja con la tabla *Draws* y la acción que realiza es que después de actualizar en la tabla *Draws* al dibujo indicado, agrega el evento al historial de actividades que se encuentra en la tabla *Binnacle*.

Al ser actualizado en la base de datos se debe identificar con los valores de nombre del usuario (*var_user*) obteniendolo de la tabla *Users* e identificandolo con su id y el evento que en este caso se define como "Modificación de Dibujo".

7.- Trigger *bin_modifyDrawConfig*

```
1 CREATE TRIGGER bin_modifyDrawConfig AFTER UPDATE ON drawsConfig
2 FOR EACH ROW
3 BEGIN
4
5     INSERT INTO Binnacle(userId, tex_event) VALUES(
6         (SELECT Us.id
7          FROM Users Us
8          WHERE ((SUBSTRING_INDEX(CURRENT_USER(), "@",1))) = Us.var_user),
9        "Modificación de Configuración de Colores de Dibujo"
10    );
11
12 END $$
```

Se elimina el trigger *bin_modifyDrawConfig* si es que este existe y posteriormente se crea; este trigger trabaja con la tabla *drawConfig* y la acción que realiza es que después de modificar en la tabla *drawConfig* la configuración del dibujo indicado, agrega el evento al historial de actividades que se encuentra en la tabla *Binnacle*.

Al ser modificado en la base de datos se debe identificar con los valores de nombre del usuario (*var_user*) obteniendolo de la tabla *Users* e identificandolo con su id y el evento que en este caso se define como "Modificación de Configuración de Colores de Dibujo".

8.- Trigger *tg_addCodedUser*

```
1 DROP TRIGGER IF EXISTS tg_addCodedUser $$
2 CREATE TRIGGER tg_addCodedUser BEFORE INSERT ON Users
3 FOR EACH ROW
4 BEGIN
5
6     SET NEW.var_pass = AES_ENCRYPT(NEW.var_pass, 'admin');
7     SET NEW.var_user = AES_ENCRYPT(NEW.var_user, 'admin');
8     SET NEW.var_category = AES_ENCRYPT(NEW.var_category, 'admin');
9
10 END $$
```

Se elimina el trigger *tg_addCodedUser* si es que este existe y posteriormente se crea; este trigger trabaja con la tabla *Users* y la acción que realiza es la de encriptar los nuevos datos agregados a la tabla de usuarios cuando detecta una inserción.

9.- Trigger *tg_updateCodedUser*

```
1 DROP TRIGGER IF EXISTS tg_updateCodedUser $$
2 CREATE TRIGGER tg_updateCodedUser BEFORE UPDATE ON Users
3 FOR EACH ROW
4 BEGIN
5     IF OLD.var_user <> NEW.var_user THEN
6         SET NEW.var_user = AES_ENCRYPT(NEW.var_user, 'admin');
7     END IF;
8
9     IF OLD.var_pass <> NEW.var_pass THEN
10        SET NEW.var_pass = AES_ENCRYPT(NEW.var_pass, 'admin');
11    END IF;
12 END $$
```

Se elimina el trigger *tg_updateCodedUser* si es que este existe y posteriormente se crea; este trigger trabaja con la tabla *Users* y la acción que realiza es la de encriptar los nuevos datos agregados a la tabla de usuarios cuando detecta una modificación.

10.- Trigger *tg_addCodedDraw*

```
1 DROP TRIGGER IF EXISTS tg_addCodedDraw $$
2 CREATE TRIGGER tg_addCodedDraw BEFORE INSERT ON Draws
3 FOR EACH ROW
4 BEGIN
5     SET NEW.var_name = AES_ENCRYPT(NEW.var_name, 'admin');
6
7     -- SET NEW.json_drawInfo = AES_ENCRYPT(NEW.json_drawInfo, "admin");
8 END $$
```

Se elimina el trigger *tg_addCodedDraw* si es que este existe y posteriormente se crea; este trigger trabaja con la tabla *Draws* y la acción que realiza es la de encriptar los nuevos datos agregados a la tabla de dibujos cuando detecta una inserción.

11.- Trigger *tg_updateCodedDraw*

```
1 DROP TRIGGER IF EXISTS tg_updateCodedDraw $$
2 CREATE TRIGGER tg_updateCodedDraw BEFORE UPDATE ON Draws
3 FOR EACH ROW
4 BEGIN
5     IF OLD.var_name <> NEW.var_name THEN
6         SET NEW.var_name = AES_ENCRYPT(NEW.var_name, "admin");
7     END IF;
8
9     IF OLD.json_drawInfo <> NEW.json_drawInfo THEN
10        SET NEW.json_drawInfo = AES_ENCRYPT(NEW.json_drawInfo, "admin");
11    END IF;
12 END $$
```

Se elimina el trigger *tg_updateCodedDraw* si es que este existe y posteriormente se crea; este trigger trabaja con la tabla *Draws* y la acción que realiza es la de encriptar los nuevos datos agregados a la tabla de dibujos cuando detecta una modificación.

12.- Trigger *tg_updateCodedColorConfiguration*

```
1 DROP TRIGGER IF EXISTS tg_updateCodedColorConfiguration $$
2 CREATE TRIGGER tg_updateCodedColorConfiguration BEFORE UPDATE ON drawsConfig
3     FOR EACH ROW
4
5     BEGIN
6         IF OLD.var_penColor <> NEW.var_penColor THEN
7             SET NEW.var_penColor = AES_ENCRYPT(NEW.var_penColor, "admin");
8         END IF;
9
10        IF OLD.var_fillColor <> NEW.var_fillColor THEN
11            SET NEW.var_fillColor = AES_ENCRYPT(NEW.var_fillColor, "admin");
12        END IF;
13    END $$
```

Se elimina el trigger *tg_updateCodedColorConfiguration* si es que este existe y posteriormente se crea; este trigger trabaja con la tabla *drawsConfig* y la acción que realiza es la de encriptar los nuevos datos agregados a la tabla de configuración para los dibujos cuando detecta una modificación.

Análisis de Procedimientos

```
1 DROP PROCEDURE IF EXISTS sp_addMainUser $$
2 CREATE PROCEDURE sp_addMainUser(IN category VARCHAR(50))
3     BEGIN
4
5         INSERT INTO Users(var_user, var_pass, var_category) VALUES
6         (((SUBSTRING_INDEX(CURRENT_USER(), "@", 1))), 'admin', category);
7
8         CREATE USER IF NOT EXISTS 'admin'@'localhost' IDENTIFIED BY 'admin';
9         GRANT ALL PRIVILEGES ON *.* TO 'admin'@'localhost' WITH GRANT OPTION;
10    END $$
```

Se debe eliminar el procedimiento almacenado *sp_addMainUser* en caso de que ya exista y posteriormente se crea, este procedimiento recibe como parámetro la categoría para validar al usuario administrador.

Dentro de la lógica del procedimiento lo que se hace es crear (si no es que ya existe) al usuario administrador y otorgarle todos los privilegios que va a tener dentro del programa; dentro de los que reacaen cosas como, agregar, modificar y eliminar tanto usuarios como dibujos.