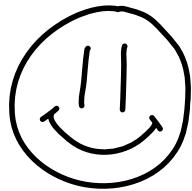


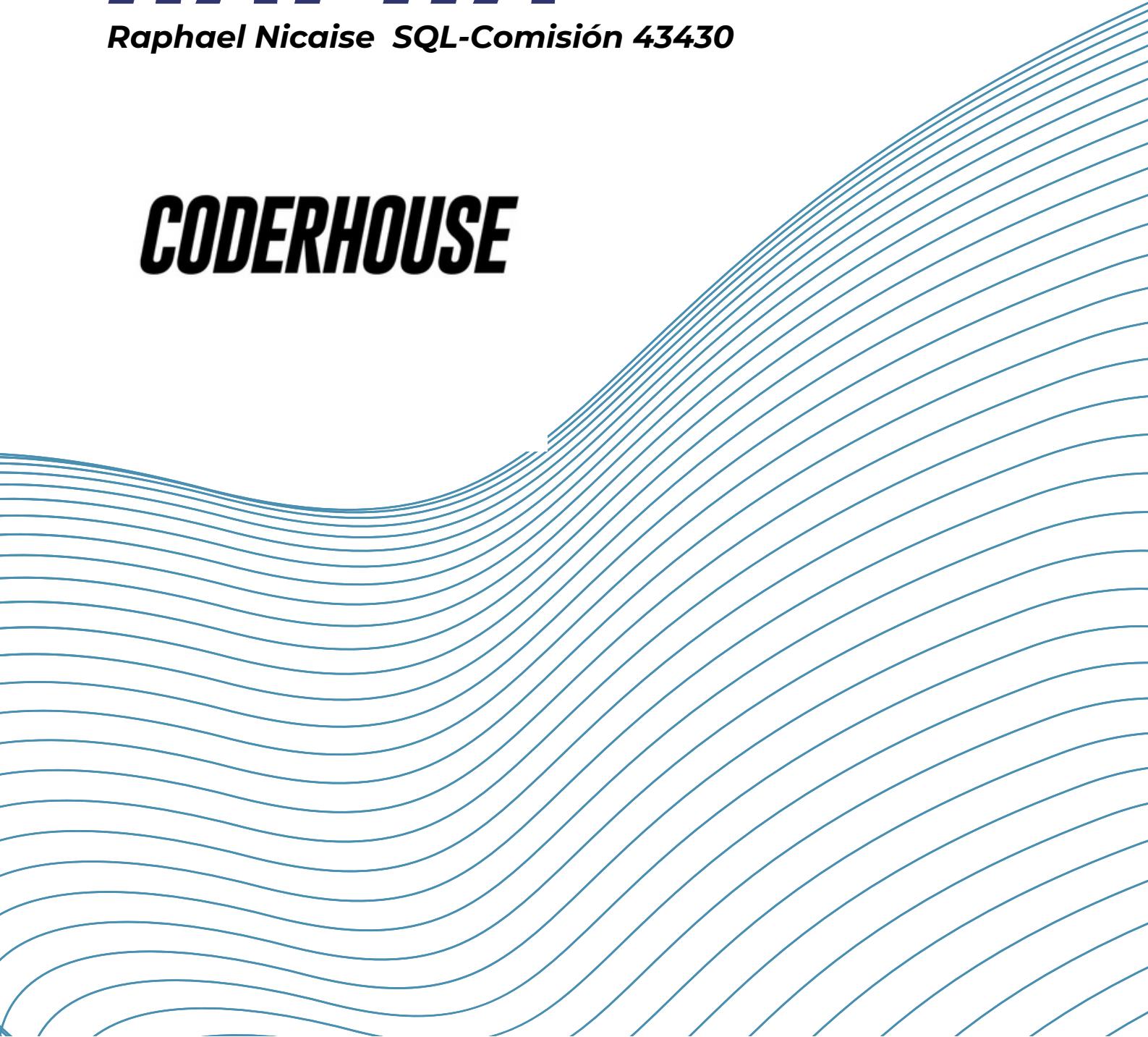
REMERAS

RAPHA



Raphael Nicaise SQL-Comisión 43430

CODERHOUSE



REPOSITORIO:



ÍNDICE

INTRODUCCIÓN

- OBJETIVO
- PROBLEMATICA
- MODELO DE NEGOCIO

MODELOS

- MODELO E-R
- MODELO-RELACIONAL

DESCRIPCION DE TABLAS

SCRIPTS

- CREATION & INSERT
- VIEWS
- FUNCTIONS
- STORED PROCEDURES
- TRIGGERS

OBJETIVO

El objetivo principal de esta DB es: Gestionar, Manejar y Analizar de manera eficaz un E-Commerce de remeras personalizadas por los mismos usuarios.

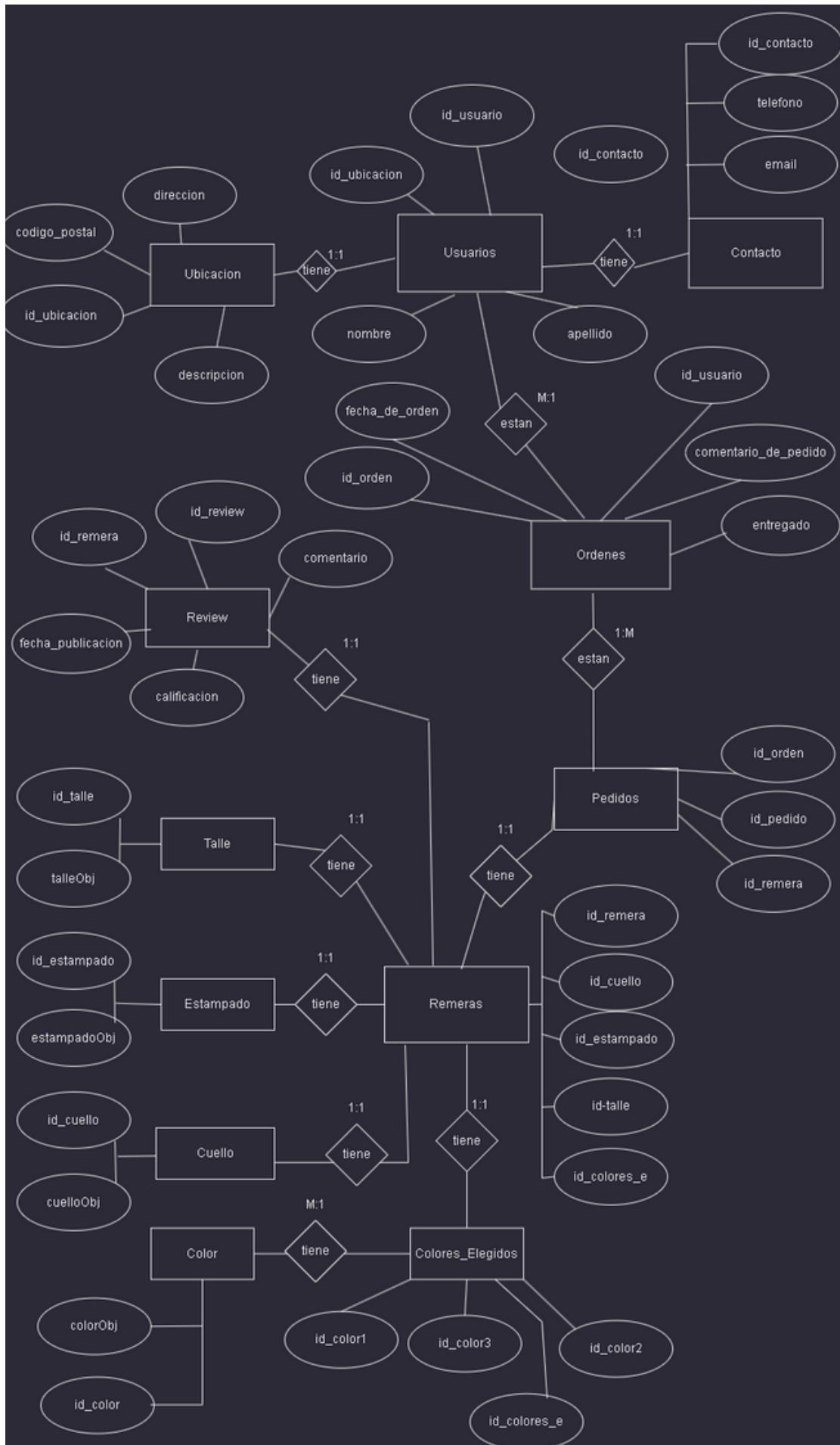
PROBLEMATICA

La problemática principal que encontré es la complicidad que tendría un modelo de negocio como este E-Commerce, sin una gestión inteligente. Esta complicidad podría resultar en una serie de problemas que afectarían la eficiencia, la experiencia del cliente y la rentabilidad

MODELO DE NEGOCIO

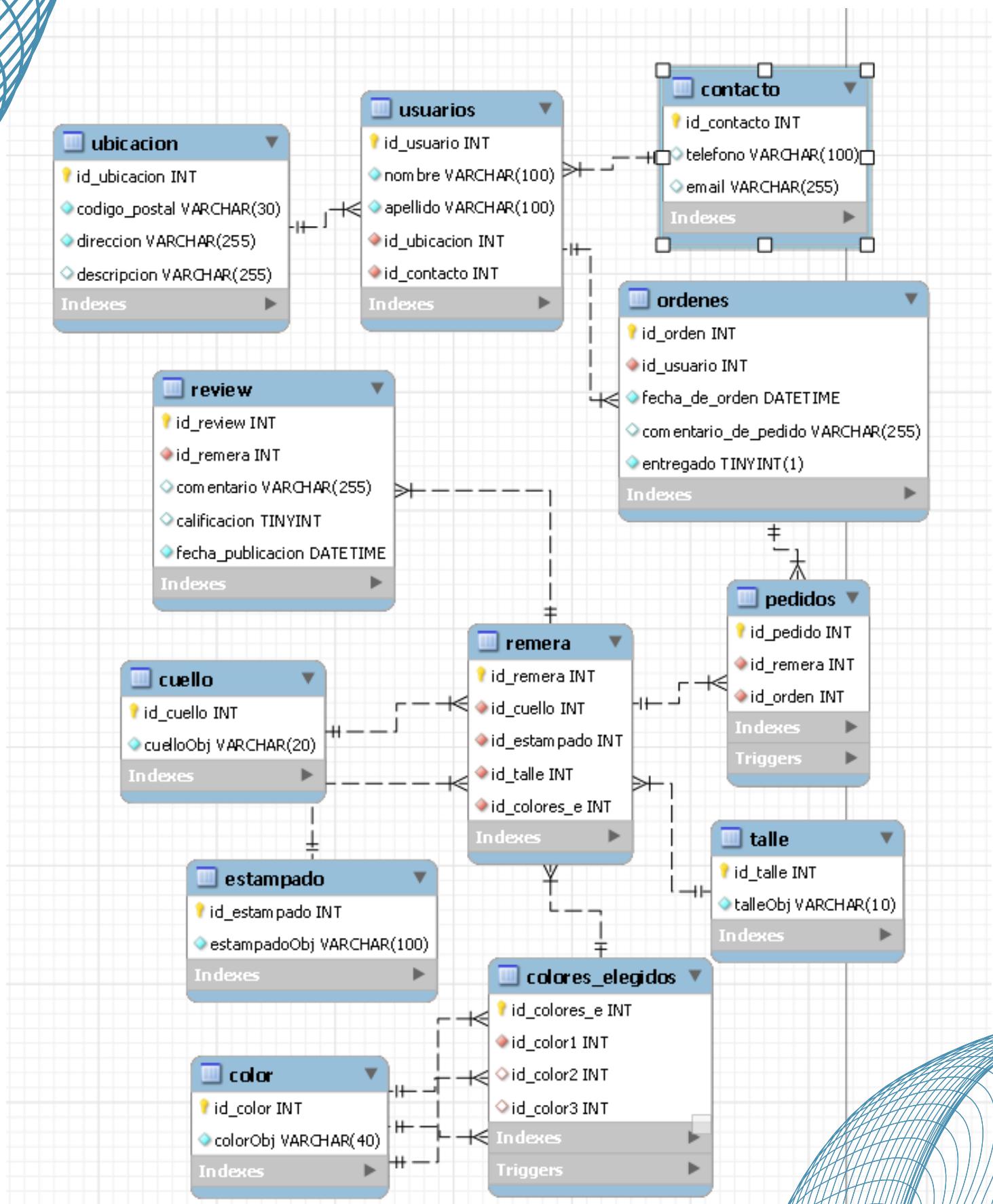
El modelo de negocio se basa en una plataforma de E-Commerce especializada en la venta de remeras personalizadas. A través de la base de datos, se agiliza la creación, gestión y análisis de un negocio que permite a los usuarios diseñar y adquirir remeras únicas. La base de datos es el núcleo del proceso al optimizar pedidos, producción, entrega y análisis de datos para decisiones estratégicas. En esencia, se crea una experiencia personalizada y eficiente para los usuarios, impulsada por la misma base de datos.

MODELO E-R



MODELO RELACIONAL

del workbench



DESCRIPCION DE TABLAS



Tabla Cuello

Tabla en la que se guardan los cuellos

PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN	NOTES
TRUE	id_cuello	INT	TRUE		AI	ID del cuello
	cuelloObj	VARCHAR	TRUE	TRUE	20	El cuello

Tabla Estampado

Tabla en la que se guardan los estampados

PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN	NOTES
TRUE	id_estampado	INT	TRUE		AI	ID del estampado
	estampadoObj	VARCHAR	TRUE	TRUE	100	El estampado

Tabla Talle

Tabla en la que se guardan los talles

PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN	NOTES
TRUE	id_talle	INT	TRUE		AI	ID del talle
	talleObj	VARCHAR	TRUE	TRUE	10	El talle

Tabla Color

Tabla en la que se guardan los colores

PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN	NOTES
TRUE	id_color	INT	TRUE		AI	ID del color
	colorObj	VARCHAR	TRUE	TRUE	40	El color

Tabla Colores_Elegidos

Tabla en la que se guardan los colores elegidos

PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN	NOTES
TRUE	id_colores_e	INT	TRUE		AI	ID de los colores elegidos
	id_color1	INT	TRUE			Foreign Key id_color1 referencia a id_color de la tabla Color
	id_color2	INT				Foreign Key id_color2 referencia a id_color de la tabla Color (opcional)
	id_color3	INT				Foreign Key id_color3 referencia a id_color de la tabla Color (opcional)

Tabla Remera

Tabla en la que se guardan los datos de la remera

PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN	NOTES
TRUE	id_remera	INT	TRUE		AI	ID de la remera
	id_cuello	INT	TRUE			Foreign Key id_cuello de la tabla Cuello
	id_estampado	INT	TRUE			Foreign Key id_estampado de la tabla Estampado
	id_talle	INT	TRUE			Foreign Key id_talle de la tabla Talle
	id_colores_e	INT	TRUE			Foreign Key id_colores_e de la tabla Colores_Elegidos

Tabla Ubicacion

Tabla en la que se guardan los datos de ubicacion de cierto usuario

PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN	NOTES
TRUE	id_ubicacion	INT	TRUE		AI	ID de la ubicacion
	codigo_postal	VARCHAR	TRUE		30	Codigo postal
	direccion	VARCHAR	TRUE		255	Direccion
	descripcion	VARCHAR			255	Descriptional (opcional)

Tabla Contacto

Tabla en la que se guardan los datos de contacto de cierto usuario

PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN	NOTES
TRUE	id_contacto	INT	TRUE		AI	ID del contacto
	telefono	VARCHAR			100	Telefono
	email	VARCHAR			255	Email

Tabla Usuarios

Tabla en donde se guardan los usuarios

PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN	NOTES
TRUE	id_usuario	INT	TRUE		AI	ID del usuario
	nombre	VARCHAR	TRUE		100	Nombre del usuario
	apellido	VARCHAR	TRUE		100	Apellido del usuario
	id_ubicacion	INT	TRUE			Foreign Key id ubicacion de la tabla Ubicacion
	id_contacto	INT	TRUE			Foreign Key id contacto de la tabla Contacto

Tabla Ordenes

Tabla donde se registran las ordenes

PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN	NOTES
TRUE	id_orden	INT	TRUE		AI	ID de la orden
	id_usuario	INT	TRUE			Foreign Key id_usuario de la tabla Usuario
	fecha_de_orden	DATETIME	TRUE			Fecha de Orden
	comentario_de_pedido	VARCHAR			255	Comentario de pedido (opcional)
	entregado	BOOLEAN	TRUE			True = entregado False = no entregado

Tabla Pedidos

Tabla en donde se registran los pedidos

PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN	NOTES
TRUE	id_pedido	INT	TRUE		AI	ID del pedido
	id_remera	INT	TRUE			Foreign Key id_remera de la tabla Remera
	id_orden	INT	TRUE			Foreign Key id_orden de la tabla Ordenes

Tabla Review

Tabla en la que se registran las review

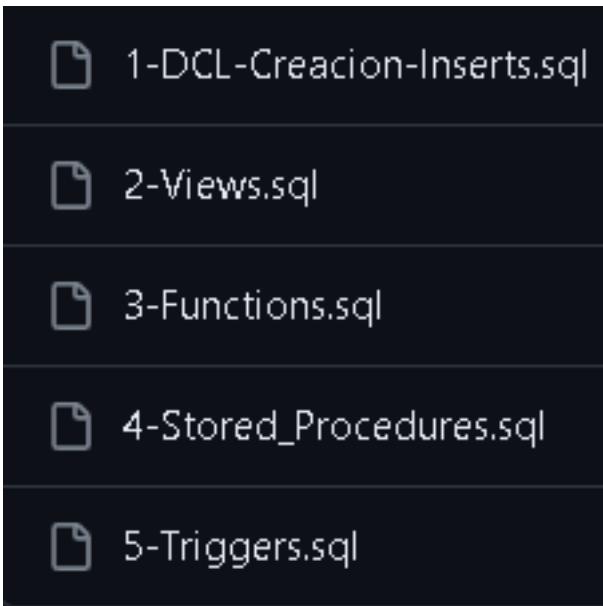
PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN	NOTES
TRUE	id_review	INT	TRUE			ID de la review
	id_remera	INT	TRUE			Foreign Key id_remera de la tabla Remera
	comentario	VARCHAR			255	Comentario (opcional)
	calificacion	TINYINT				CHECK (calificacion >= 1 AND calificacion <= 5)
	fecha_publicacion	DATETIME	TRUE			Fecha en la que se publica la review



SCRIPTS



ORDEN DE EJECUCION:



Los 5 Scripts están preparados para ejecutarse de una vez

SCRIPT 1: 1-DCL-CREACION-INSERTS.SQL



En este Script, primero te encontraras con la creación de tablas, y después las inserciones de datos sobre ellas.

SCRIPT 2: 2-VIEWS.SQL



En este Script se encuentran todas las Vistas.

VISTA 1

-La vista Diseños_remeras muestra el cuello, el estampado, el talle y los colores, de cada remera



```
CREATE OR REPLACE VIEW Diseños_remeras AS
(SELECT R.id_remera,c.cuelloObj, Es.estampadoObj,T.talleObj
,(c1.colorObj) AS color1,
(c2.colorObj) AS color2,
(c3.colorObj) AS color3
FROM remera R
JOIN cuello C ON C.id_cuello = R.id_cuello
JOIN estampado Es ON Es.id_estampado = R.id_estampado
JOIN talle T ON T.id_talle = R.id_talle
JOIN colores_elegidos CE ON CE.id_colores_e = R.id_colores_e
LEFT JOIN color c1 ON CE.id_color1 = c1.id_color
LEFT JOIN color c2 ON CE.id_color2 = c2.id_color
LEFT JOIN color c3 ON CE.id_color3 = c3.id_color
ORDER BY id_remera);
```

-La vista Colores_remeras muestra los colores de cada remera

VISTA 2

```
CREATE OR REPLACE VIEW Colores_remeras AS (
SELECT R.id_remera,R.id_colores_e,
(c1.colorObj) AS color1,
(c2.colorObj) AS color2,
(c3.colorObj) AS color3
FROM remera R
JOIN colores_elegidos CE ON CE.id_colores_e = R.id_colores_e
LEFT JOIN color c1 ON CE.id_color1 = c1.id_color
LEFT JOIN color c2 ON CE.id_color2 = c2.id_color
LEFT JOIN color c3 ON CE.id_color3 = c3.id_color
ORDER BY id_remera);
```

VISTA 3

-La vista Calificacion_remeras muestra la clasificación, el comentario de cada remera , el teléfono y el mail.



```
CREATE OR REPLACE VIEW Calificacion_remeras AS  
(SELECT REV.* ,co.email,co.telefono FROM review REV join remera REM ON REV.id_remara = REM.id_remara  
join pedidos p on p.id_remara = REM.id_remara  
join ordenes o on o.id_orden = p.id_orden  
join usuarios us on o.id_usuario = us.id_usuario  
join contacto co on us.id_contacto = co.id_contacto);
```

VISTA 4

-La vista Infopararepartidor muestra información que le va a servir al repartidor Nombre, Apellido, dirección, descripción y el teléfono.



```
CREATE OR REPLACE VIEW infoparaRepartidor AS (  
SELECT ord.id_orden,Concat(US.nombre,' ',US.apellido)  
AS Nombre_Apellido ,UB.direccion, UB.descripcion, CO.telefono  
FROM ubicacion UB  
JOIN usuarios US ON US.id_ubicacion = UB.id_ubicacion  
JOIN contacto CO ON CO.id_contacto = US.id_contacto  
join ordenes ord ON ord.id_usuario = US.id_usuario  
where entregado = false Order By id_orden );
```

VISTA 5

-La vista informacionpedido muestra información general de las ordenes



```
CREATE OR REPLACE VIEW InformacionOrdenes AS  
(SELECT  
p.id_remara,o.id_orden,c.email,o.fecha_de_orden,  
o.comentario_de_pedido  
FROM pedidos p JOIN ordenes o  
ON p.id_orden = o.id_orden  
JOIN usuarios u ON o.id_usuario = u.id_usuario  
JOIN contacto c ON u.id_contacto = c.id_contacto  
ORDER BY fecha_de_orden ASC);
```

VISTA 6

-La vista cant_Ordenes_Entregadas devuelve la cantidad de ordenes Entregadas

```
CREATE OR REPLACE VIEW cant_Ordenes_Entregadas AS (
select count(*) from ordenes where entregado =
true);
```

VISTA 7

-Esta vista devuelve la cantidad de ordenes que quedan por entregar.

```
CREATE OR REPLACE VIEW cant_Ordenes_para_entregar AS (
select count(*) as Cantidad from ordenes where
entregado = false);
```

VISTA 8

-Devuelve una tabla con la cantidad de pedidos por cada usuario



```
CREATE OR REPLACE VIEW Cant_pedidos_por_usuario AS (
SELECT concat(u.nombre,' ',u.apellido) as Nombre,
count(id_pedido) as cantidadPedida FROM usuarios u
JOIN ordenes o ON o.id_usuario = u.id_usuario
JOIN pedidos p ON p.id_orden = o.id_orden
group by u.id_usuario
order by cantidadPedida desc);
```

VISTA 9

-Devuelve una tabla con la cantidad de ordenes por cada usuario



```
CREATE OR REPLACE VIEW Cant_ordenes_por_usuario AS (
SELECT concat(u.nombre,' ',u.apellido) as Nombre,
count(id_orden) as cantidadOrdenes FROM usuarios u
JOIN ordenes o ON o.id_usuario = u.id_usuario
group by u.id_usuario
order by cantidadOrdenes desc);
```

VISTA 10

-Devuelve la cantidad de remeras registradas en algún pedido



```
CREATE OR REPLACE VIEW Cant_remerasRegistradas AS (select count(*) as Cantidad from pedidos);
```

-Devuelve la cantidad de remeras ya entregadas

```
CREATE OR REPLACE VIEW Cant_remerasEntregadas AS (select count(*) as Cantidad from pedidos p join ordenes o on p.id_orden = o.id_orden where o.entregado = true);
```

VISTA 11

SCRIPT 3: 3-FUNCTIONS.SQL



En este Script se encuentran todas las Funciones.

FUNCION 1

-Toma como variable la id de un estampado y devuelve la cantidad de remeras con ese estampado.

```
DELIMITER //  
CREATE FUNCTION cantVentas_x_estampado(estampado_id  
INT)  
RETURNS INT READS SQL DATA  
BEGIN  
    DECLARE cantidad INT;  
  
    SELECT COUNT(*) INTO cantidad  
    FROM Remera R  
    JOIN Pedidos P ON R.id_remara = P.id_remara  
    WHERE R.id_estampado = estampado_id;  
  
    RETURN cantidad;  
END //
```



```
CREATE FUNCTION email_usuario(usuario_id INT)
RETURNS VARCHAR(255)
READS SQL DATA
BEGIN
    return(select c.email from usuarios u
           join contacto c on c.id_contacto = u.id_contacto
           where id_usuario = usuario_id);
END $$
```

FUNCION 2

-Toma como variable la id de un usuario, y devuelve su email.

FUNCION 3

-Toma como variable la id de algún color y devuelve la cantidad de remeras que tengan ese color.

```
DELIMITER $$
CREATE FUNCTION cantRemerascon_xcolor(color_id INT)
RETURNS int READS SQL DATA
BEGIN
    return(SELECT count(*) FROM colores_elegidos ce
           where ce.id_color1 = color_id or ce.id_color2 =
           color_id or ce.id_color3 = color_id);
END $$
```

SCRIPT 4: 4-STORED_PROCEDURES.SQL

En este Script se encuentran todos los Stored Procedures



SP 1

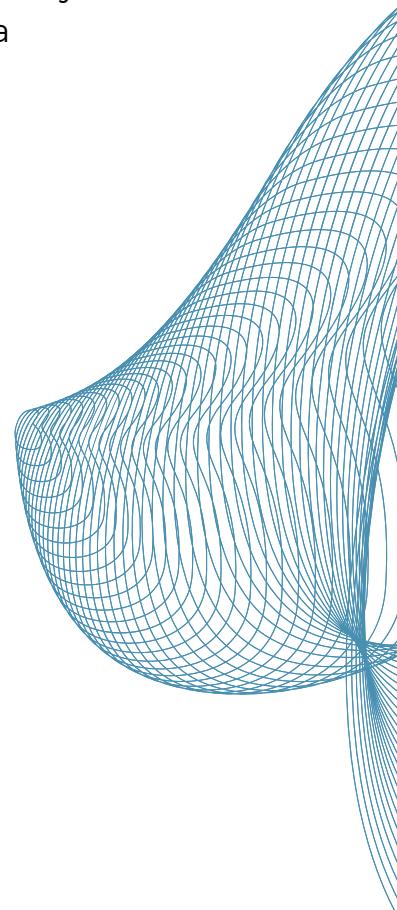
-Este SP toma como variable a id_orden, si esta orden no fue entregada, le coloca entregado = 1 (True), Si la orden ingresada ya es true devuelve que esta ya fue entregada

```
DELIMITER //
CREATE PROCEDURE confirmarEntregaOrden(IN orderID
INT)
BEGIN

DECLARE entregado_value INT;

SELECT entregado INTO entregado_value
FROM ordenes
WHERE id_orden = orderID;

IF entregado_value = 0 THEN
    UPDATE ordenes
    SET entregado = 1 where id_orden = orderID;
    SELECT 'Entrega Realizada Correctamente!' as
comentario;
ELSE
    SELECT 'El pedido YA fue entregado' as
comentario;
END IF;
END //
```



-El SP Usuarios_ordenamiento toma como variable, a la columna por la que queremos ordenar a la tabla, y otra variable si la queremos ordenar asc o desc.

SP 2

```
DELIMITER //
create procedure Usuarios_Oordenamiento(IN columna varchar(50),IN asc_desc char(4))
BEGIN

DECLARE clausula VARCHAR(200);
DECLARE orderbycolumna VARCHAR(100);
DECLARE tipodeordenamiento varchar(200);

IF columna < '' THEN
    set @orderbycolumna = concat('Order by ', columna);
ELSE
    set @orderbycolumna = '';
END IF;

IF asc_desc IN ('ASC','asc','DESC','desc') THEN
    SET @tipodeordenamiento = asc_desc;
ELSE
    set @tipodeordenamiento = '';
END IF;

SET @clausula = concat('Select * from Usuarios ', @orderbycolumna,' ', @tipodeordenamiento);
PREPARE runSQL FROM @clausula;
EXECUTE runSQL;
DEALLOCATE PREPARE runSQL;
END //
```

SP 3

-Este SP toma como variable uno de los 3 accesorios que hay en una remera, y hace un top con los mas usado

```
DELIMITER //
CREATE PROCEDURE top_de_cada_accesorio(
IN tipoAccesorio ENUM('cuelloObj','estampadoObj','talleObj'))

BEGIN
DECLARE clausula1 varchar(255);
DECLARE accesorioelegido varchar(50);

SET @accesorioelegido = tipoAccesorio;
SET @clausula1 = concat('SELECT ',@accesorioelegido,',count(*) as
CantUsada FROM remera R JOIN cuello C ON C.id_cuello = R.id_cuello
JOIN estampado Es ON Es.id_estampado = R.id_estampado
JOIN talle T ON T.id_talle = R.id_talle
group by ',@accesorioelegido , ' order by CantUsada desc;');

PREPARE runSQL FROM @clausula1;
EXECUTE runSQL;
DEALLOCATE PREPARE runSQL;
END //
```

-Este SP agrega un usuario, y tambien agrega su respectiva informacion en la tabla ubicacion y contacto

SP 4

```
DELIMITER $$
CREATE PROCEDURE add_usuario(
    IN nombreNuevo VARCHAR(100),
    IN apellidoNuevo VARCHAR(100),
    IN telefonoNuevo VARCHAR(100),
    IN emailNuevo VARCHAR(255),
    IN codigo_postalNuevo VARCHAR(30),
    IN direccion VARCHAR(255))

BEGIN

    DECLARE contacto_id INT;
    DECLARE ubicacion_id INT;

    INSERT INTO ubicacion (codigo_postal, direccion, descripcion)
    VALUES (codigo_postalNuevo, direccion, '');
    SET ubicacion_id = LAST_INSERT_ID();

    INSERT INTO contacto (telefono, email)
    VALUES (telefonoNuevo, emailNuevo);
    SET contacto_id = LAST_INSERT_ID();

    INSERT INTO usuarios (nombre, apellido, id_contacto, id_ubicacion)
    VALUES (nombreNuevo, apellidoNuevo, contacto_id, ubicacion_id);

    SELECT 'Nuevo usuario agregado: ' AS Mensaje,
    nombreNuevo AS Nombre, apellidoNuevo AS Apellido;

END $$
```

SP 5

-Este SP agrega una remera, colocando el id de cada accesorio, ademas de agregar los colores en la tabla colores_elegidos

```
DELIMITER //
CREATE PROCEDURE add_remera(
    IN cuello_id INT, IN estampado_id INT, IN talle_id INT,
    IN color1_id INT, IN color2_id INT, IN color3_id INT )
BEGIN
    DECLARE remera_id INT;
    DECLARE colores_elegidos_id INT;

    INSERT INTO Colores_Elegidos (id_color1, id_color2, id_color3)
    VALUES (color1_id, IFNULL(color2_id, NULL), IFNULL(color3_id, NULL));
    SET colores_elegidos_id = LAST_INSERT_ID();

    INSERT INTO Remera (id_cuello, id_estampado, id_talle, id_colores_e)
    VALUES (cuello_id, estampado_id, talle_id, colores_elegidos_id);
    SET remera_id = LAST_INSERT_ID();

    SELECT concat('Remera agregada correctamente: id_remera ', remera_id) as Comentario;
END //
```

SP 6

-Este SP relaciona a el usuario con una remera.
La lógica de esta BD es que en una orden pueden haber muchos pedidos (ósea que a una orden se le pueden asignar varios pedidos, siempre que sea del mismo usuario). Entonces puse una regla que fue, si se hacia un pedido, y después otro en menos de 24 horas, estos se guardarían en una misma orden.

```
DELIMITER //
CREATE PROCEDURE add_pedido(
    IN usuario_id INT,
    IN id_remera INT
)
BEGIN
    DECLARE orden_id INT;
    DECLARE ultima_orden24hs_id INT;
    DECLARE mensaje VARCHAR(255);

    SELECT id_orden INTO ultima_orden24hs_id
    FROM Ordenes
    WHERE id_usuario = usuario_id
    AND fecha_de_orden >= DATE_SUB(NOW(), INTERVAL 1 DAY)
    ORDER BY fecha_de_orden DESC
    LIMIT 1;

    IF ultima_orden24hs_id IS NULL THEN
        INSERT INTO Ordenes (id_usuario, fecha_de_orden, entregado)
        VALUES (usuario_id, NOW(), false);
        SET orden_id = LAST_INSERT_ID();
    ELSE
        SET orden_id = ultima_orden24hs_id;
    END IF;

    INSERT INTO Pedidos (id_remera, id_orden) VALUES (id_remera, orden_id);
    SET mensaje = CONCAT('Pedido agregado correctamente en la ID de Orden: ', orden_id);
    SELECT mensaje AS comentario;
END //
```

SP 7

-Este SP modifica los colores de la id_remera ingresada, colocando los id_color que quieras
(Si a uno de los colores los queres dejar vacios, colocar NULL)

```
•••  
delimiter //  
CREATE PROCEDURE edit_Colores_Elegidos  
(IN remera_id INT, IN color1_id INT, IN color2_id INT, IN color3_id INT)  
BEGIN  
  
    UPDATE colores_elegidos co JOIN remera r ON r.id_colores_e = co.id_colores_e  
    SET id_color1 = color1_id, id_color2 = color2_id, id_color3 = color3_id  
    WHERE co.id_colores_e = remera_id;  
    SELECT CONCAT('La id_remera: ', remera_id, ' a sido modificada a los colores: ') AS Comentario,  
    color1_id AS id_color1,  
    color2_id AS id_color2,  
    color3_id AS id_color3 FROM colores_elegidos  
    LIMIT 1;  
END //
```

-Este SP devuelve la cantidad de remeras vendidas

SP 8

```
•••  
DELIMITER //  
CREATE PROCEDURE total_remeras_vendidas(OUT total_remeras_v INTEGER)  
BEGIN  
    SELECT COUNT(*) INTO total_remeras_v FROM pedidos p  
    JOIN ordenes o ON p.id_orden = o.id_orden WHERE o.entregado = TRUE;  
END //
```

SP 9

-Este SP toma a un id_usuario y devuelve la cantidad de ordenes de ese usuario

```
•••  
DELIMITER //  
CREATE PROCEDURE get_totalOrdenesxUsuario(  
    IN usuario_id INT,  
    OUT totalOrdenes INT  
)  
BEGIN  
    SELECT COUNT(*) INTO totalOrdenes  
    FROM Ordenes  
    WHERE id_usuario = usuario_id ;  
END //
```

SP 10

-Este SP toma a un id_usuario y la nueva descripción que le vamos a agregar a su ubicación.

```
DELIMITER //
CREATE PROCEDURE agregarDescripcion_Ubicacion(IN usuario_id INT,
                                              IN nueva_descripcion_ubicacion varchar(255))
BEGIN
    UPDATE ubicacion ub join usuarios us
    ON ub.id_ubicacion = us.id_ubicacion
    SET descripcion = nueva_descripcion_ubicacion where id_usuario = usuario_id;
END //
```

SP 11

-Este SP toma la id_usuario y devuelve toda su información: Nombre, Apellido, Teléfono, Email, Dirección y Código Postal

```
DELIMITER //
CREATE PROCEDURE informacionCompleta_usuario (IN usuario_id INT)
BEGIN
    SELECT concat(nombre,' ',apellido) as usuario ,
    co.telefono,co.email,ub.direccion, ub.codigo_postal as CódigoPostal
    FROM usuarios us join ubicacion ub
    ON us.id_ubicacion = ub.id_ubicacion
    JOIN contacto co on us.id_contacto = co.id_contacto
    where id_usuario = usuario_id;
END //
```

SP 12

-Este SP toma el id_estampado y devuelve el diseño de todas las remeras con ese estampado

```
DELIMITER //
CREATE PROCEDURE remerasxestampado(IN estampado_id INT)
BEGIN
    SELECT R.id_remera,c.cuelloObj, Es.estampadoObj,T.talleObj, R.id_colores_e
    ,(c1.colorObj) AS color1,
    (c2.colorObj) AS color2,
    (c3.colorObj) AS color3
    FROM remera R
    JOIN cuello C ON C.id_cuello = R.id_cuello
    JOIN estampado Es ON Es.id_estampado = R.id_estampado
    JOIN talle T ON T.id_talle = R.id_talle
    JOIN colores_elegidos CE ON CE.id_colores_e = R.id_colores_e
    LEFT JOIN color c1 ON CE.id_color1 = c1.id_color
    LEFT JOIN color c2 ON CE.id_color2 = c2.id_color
    LEFT JOIN color c3 ON CE.id_color3 = c3.id_color
    where Es.id_estampado = estampado_id
    ORDER BY id_remera;
END //
```

SP 13

-Este SP, ejecutándolo devuelve todos los usuarios con su nombre y apellido correspondiente

```
DELIMITER //
CREATE PROCEDURE get_usuarios()
BEGIN
    SELECT id_usuario,nombre,apellido FROM USUARIOS;
END //
```

-Este SP, ejecutandolo devuelve los colores de todas las remeras

SP 14

```
DELIMITER //
CREATE PROCEDURE get_colores_remera ()
BEGIN
    SELECT R.id_remara,R.id_colores_e,
    (c1.colorObj) AS color1,
    (c2.colorObj) AS color2,
    (c3.colorObj) AS color3
    FROM remera R
    JOIN colores_elegidos CE ON CE.id_colores_e = R.id_colores_e
    LEFT JOIN color c1 ON CE.id_color1 = c1.id_color
    LEFT JOIN color c2 ON CE.id_color2 = c2.id_color
    LEFT JOIN color c3 ON CE.id_color3 = c3.id_color
    ORDER BY id_remara asc;
END //
```

SP 15

-Este SP agrega un nuevo Estampado

```
DELIMITER //
CREATE PROCEDURE add_Estampado(in nuevoEstampado varchar(50))
BEGIN
    IF nuevoEstampado not in (select estampadoObj from estampado) then
        INSERT INTO estampado (estampadoObj) values (nuevoEstampado);
        SELECT concat('Nuevo estampado: ',nuevoEstampado,' fue agregado correctamente!') as Comentario;
    ELSE
        SELECT 'Este estampado ya existe' as comentario;
    END IF;
END //
```

SP 16

-Este SP le agrega un comentario a cierta id_orden

```
DELIMITER //
CREATE PROCEDURE add_comentario_orden(IN orden_ID INT, IN Comentario_Orden varchar(255))
BEGIN
    update ordenes
    set comentario_de_pedido = Comentario_Orden
    WHERE id_orden = orden_id;
END //
```

SP 17

-Este SP agrega una remera a la tabla review, con su respectiva calificación y comentario

```
DELIMITER //
CREATE PROCEDURE add_review(IN remera_id INT, IN comentarioNew VARCHAR(255), IN calificacionNew TINYINT)
BEGIN
    if calificacionNew >= 1 AND calificacionNew <= 5 THEN
        INSERT INTO review (id_remera,comentario,calificacion,fecha_publicacion)
        values (remera_id,comentarioNew,calificacionNew,NOW());
        SELECT * from review where id_remera = remera_id;
    ELSE
        select concat('La calificacion solo esta permitida desde 1 a 5');
    END IF;
END //
```

SP 18

-Este SP toma una id_remera y devuelve su review (o reseña)

```
DELIMITER //
CREATE PROCEDURE obtenerReseñaPorRemera(IN remeraID INT)
BEGIN
    if remeraID in (select id_remera from review) then
        SELECT r.id_remera, r.comentario, r.calificacion, r.fecha_publicacion
        FROM Review r
        WHERE r.id_remera = remeraID;
    else
        select 'Esta remera no tiene review' as Mensaje;
    END IF;
END //
```

SCRIPT 5: 5-TRIGGERS.SQL



En este Script se encuentran todos los Triggers.

```
CREATE TABLE Pedidos_log
(
    id_pedido INT NOT NULL AUTO_INCREMENT,
    fecha_hora DATETIME NOT NULL,
    usuario VARCHAR(255) NOT NULL,
    PRIMARY KEY (id_pedido)
);

delimiter //
CREATE TRIGGER log_pedido
AFTER INSERT ON pedidos
FOR EACH ROW
BEGIN
    INSERT INTO Pedidos_log
    (id_pedido, fecha_hora, usuario)
    VALUES
    (NEW.id_pedido, NOW(), USER());
END //
```

TRIGGER 2

-Este Trigger guarda en la tabla de auditoria colores_elegidos_log si se hace un update de los colores_elegidos y guarda los anteriores al update

TRIGGER 1

-Este Trigger guarda en la tabla de auditoria Pedidos_log

Devuelve el id_pedido, la fecha y la hora, y el usuario de la BD que registro el pedido.

```
CREATE TABLE colores_elegidos_log (
    timestamp DATETIME,
    usuario VARCHAR(255),
    action VARCHAR(50),
    old_id_colores_e INT,
    old_id_color1 INT,
    old_id_color2 INT,
    old_id_color3 INT
);

DELIMITER //
CREATE TRIGGER log_colores_elegidos_act
BEFORE UPDATE ON Colores_Elegidos
FOR EACH ROW
BEGIN
    INSERT INTO colores_elegidos_log
    VALUES (NOW(), USER(), 'UPDATE',
    NEW.id_colores_e, NEW.id_color1, NEW.id_color2,
    NEW.id_color3);
END //
```