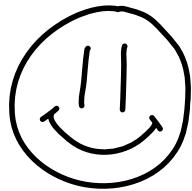


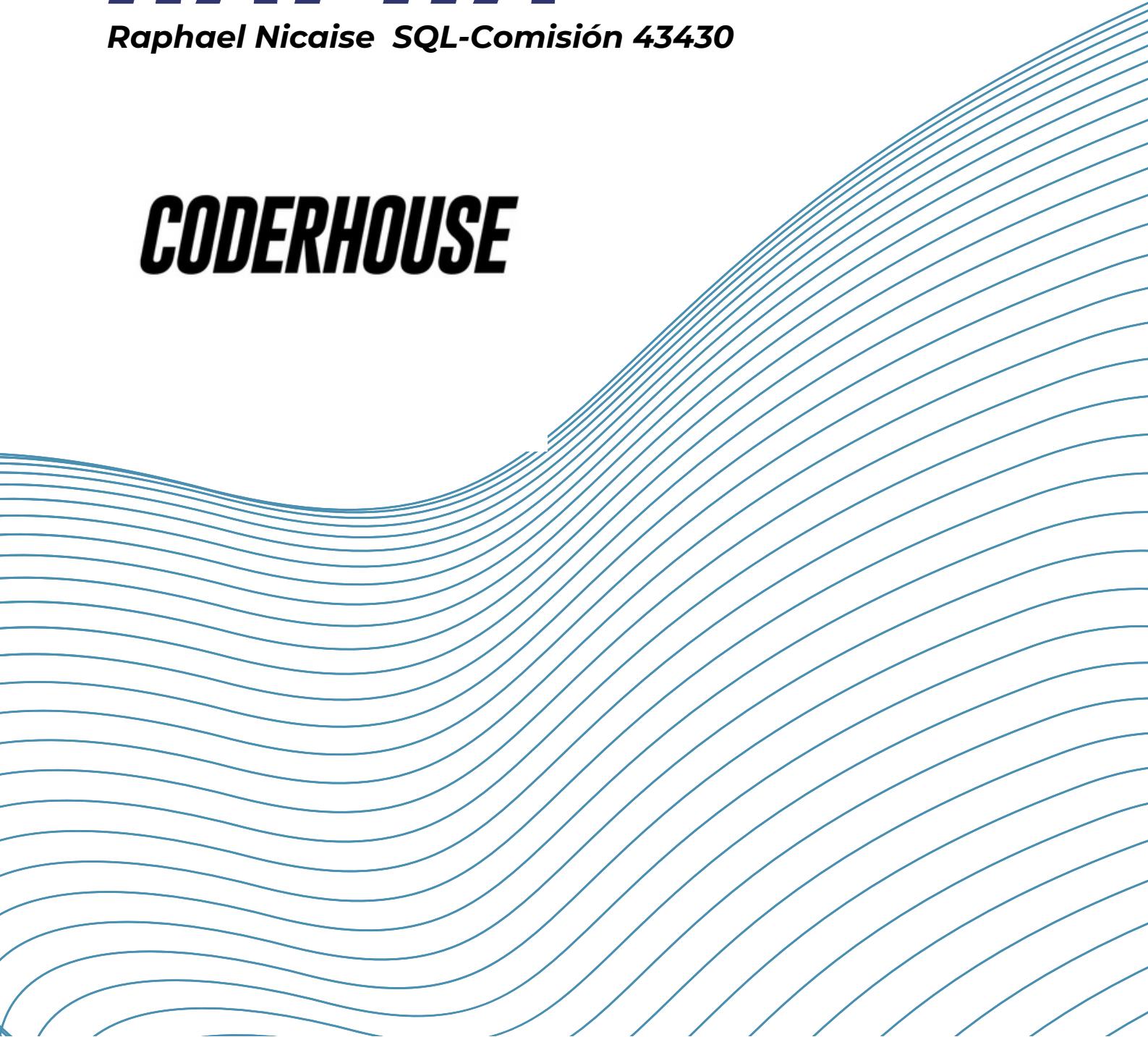
# **REMERAS**

# **RAPHA**



*Raphael Nicaise SQL-Comisión 43430*

**CODERHOUSE**



# REpositorio:



## ÍNDICE

### INTRODUCCIÓN

- OBJETIVO
- PROBLEMATICA
- MODELO DE NEGOCIO

### BASE DE DATOS

- MODELO E-R
- MODELO-RELACIONAL
- DESRIPCION DE TABLAS

### SCRIPTS

- DDL - TABLAS
- VISTAS
- FUNCIONES
- STORED PROCEDURES
- TRIGGERS
- PRIVILEGE-USERS
- INSERTS
- EJECUTABLE

### FINAL

- INFORMES
- A FUTURO

# **EN QUE CONSISTE LA BASE DE DATOS**

Objetivo | Problemática | Modelo de Negocio

El **objetivo principal** de esta base de datos es la **buenas gestión, control y análisis** profundo de todas las operaciones que conllevan a un **E-Commerce especializado en la venta de remeras, personalizadas por los propios usuarios**. Mi meta fue crear una Base de datos, que permita al cliente **diseñar y pedir remeras personalizadas**, (según un catalogo establecido), y que mediante varios procesos, se pueda llegar a tener una **buenas resolución de problemas**, con la información dada y manejada por los mismos procesos, para que el negocio funcione de manera **excelente**.

La **problemática principal** que conlleva a realizar la base de datos, se relaciona con la complejidad que tiene un modelo de negocio de E-Commerce, centrado en la **personalización de las remeras**. Si no hay una gestión inteligente y una base de datos efectiva, se pueden dar **casos de fallos**, como la mezcla de remeras a cada usuario, o remeras mal registradas, etc.

Uno de los problemas clave que resuelve la DB, es la de permitir al usuario poder tener una remera completamente personalizada, resguardando todos los datos de esta, para que no se pierda, y pueda llegar al cliente de forma segura.

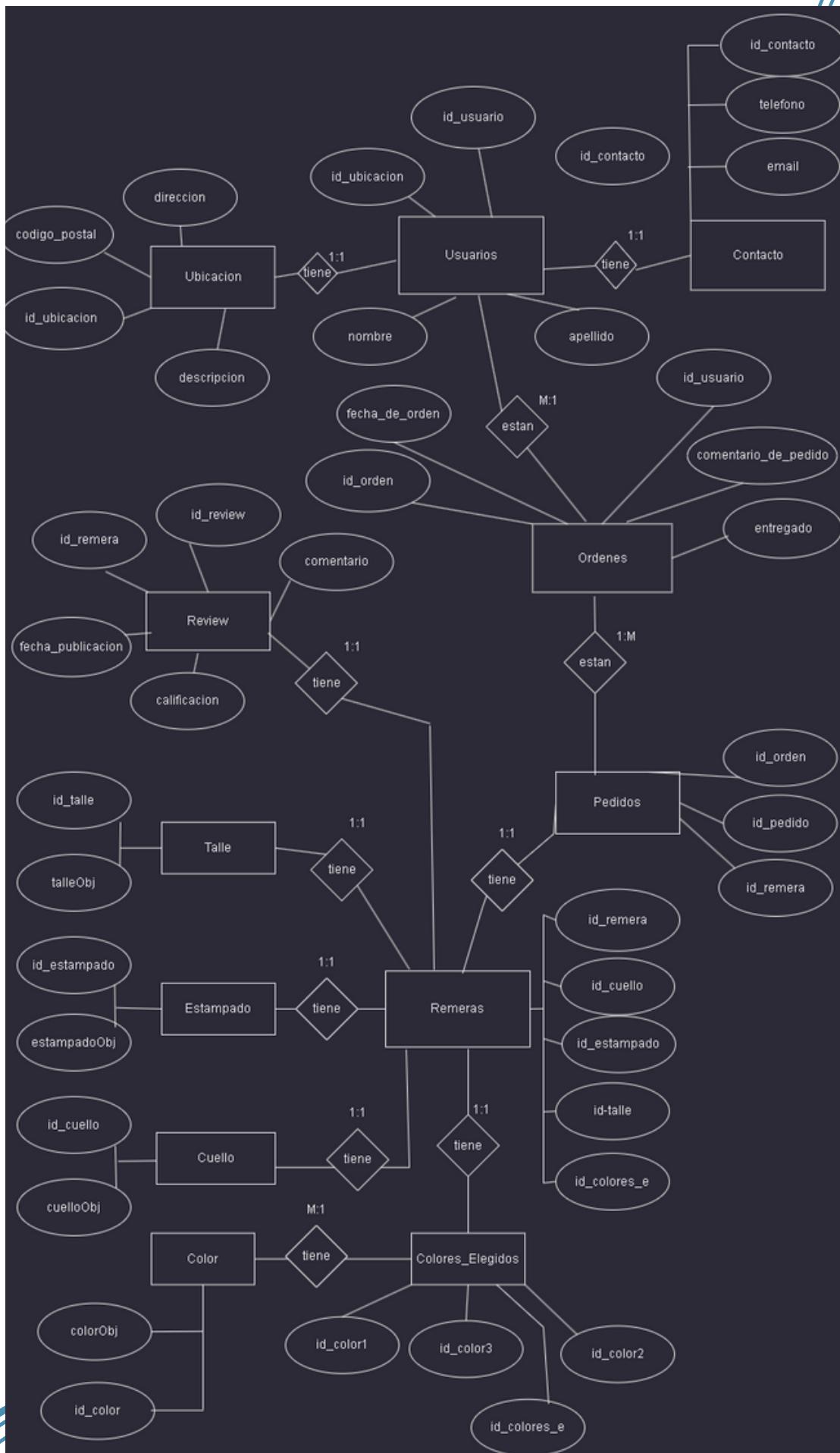
Además, **la producción y la logística** de entrega también pueden volverse **un problema sin una base de datos buena**. Por lo que se brindan **vistas**, que serían **utilizadas por proveedores, o por los fabricantes** de la remera, a los que le enviamos todos los datos, para que la **producción y la entrega se ejecute**, también de forma **segura y eficiente**.

Por último, y no menos importante, esta DB nos permite **analizar el negocio de distintas maneras**, por ejemplo: cuales son los accesorios mas usados, cuales son los usuarios con mas ordenes, como también nos brinda información de fechas en distintas tablas, para que posteriormente **podamos usarlas y filtrarlas para distintos procesos estadísticos**, como por ejemplo, la cantidad de ordenes por mes.

## **COSAS A TENER EN CUENTA:**

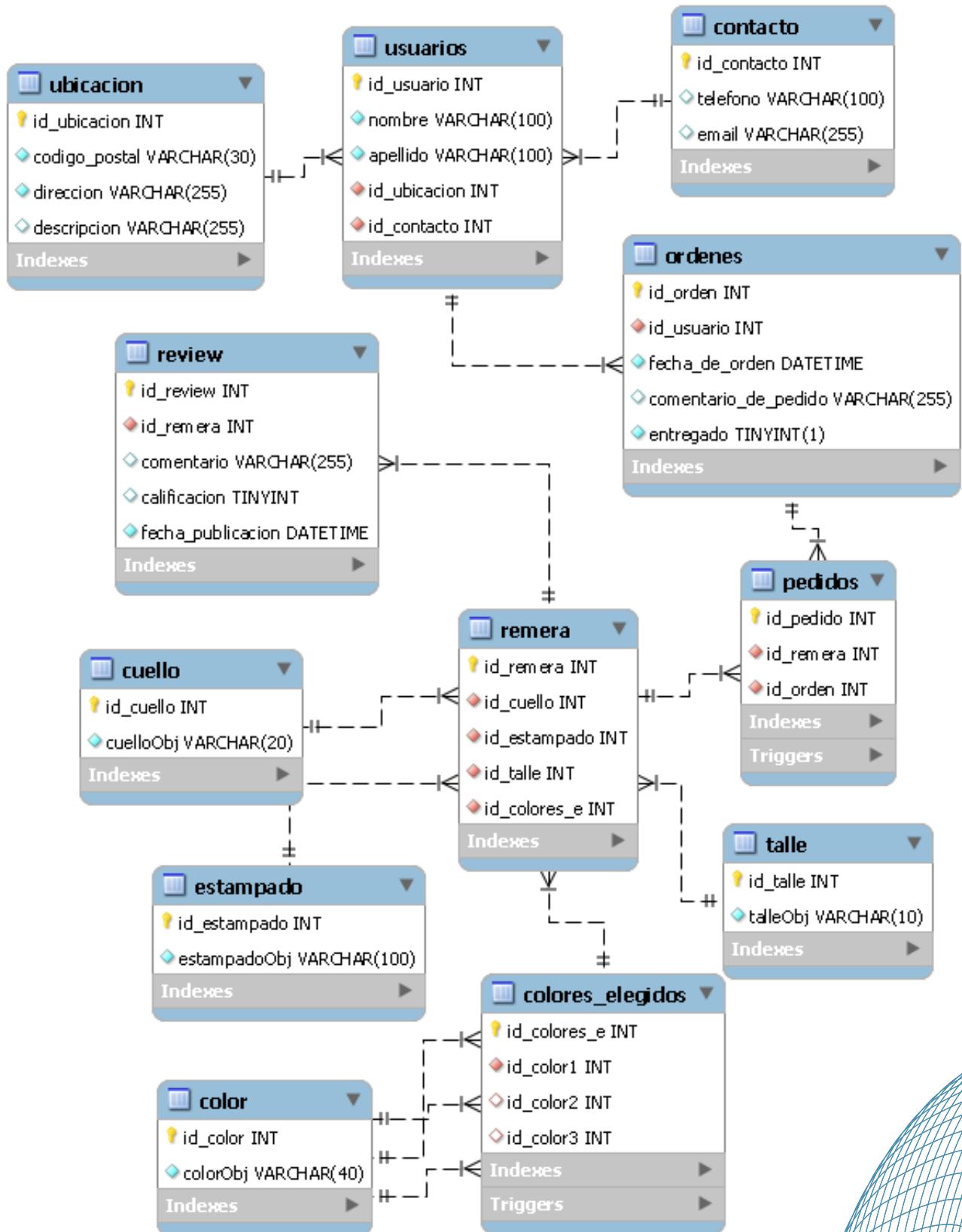
Resumidamente, la base de datos contiene una tabla de usuarios, de contacto y de ubicación, en la que se guarda información de cada usuario. Este usuario puede crear remeras, que se agregan automáticamente a un pedido, y estos pedidos, con una lógica que explico mas adelante, se agregan a una orden, esta orden puede contener varios pedidos. (ósea varias remeras). Cada remera tiene foreign keys, de las tablas, estampado, colores\_elegidos, cuello y talle. Cada remera, tiene la posibilidad de tener una Review dada por el usuario.

# 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	Descriptinal (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

### Tabla Colores\_elegidos\_log

Tabla en la que se registran los cambios en colores\_elegidos existentes.

PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN	NOTES
	timestamp	datetime				Fecha y Hora en la que se ejecuta el trigger
	usuario	VARCHAR			255	"Usuario" de la DB que lo provoco USER()
	action	VARCHAR				255 Guarda la palabra "Update"
	old_id_colores_e	INT				Guarda el id_colores_e que se cambio
	old_id_color1	INT				id_color1 antiguo
	old_id_color2	INT				id_color2 antiguo
	old_id_color3	INT				id_color3 antiguo
	new_id_color1	INT				id_color1 nuevo
	new_id_color2	INT				id_color2 nuevo
	new_id_color3	INT				id_color3 nuevo

### Tabla log\_usuarios

Tabla en la que se registra cuando se inserta un nuevo pedido

PK	COLUMN	TYPE	NOT NULL	UNIC	LEN	NOTES
	id_usuario	INT	TRUE			ID del nuevo usuario
	creacion	TIMESTAMP				Fecha y hora en el que se creo el usuario

### Tabla Pedidos\_log

Tabla en la que se registra cuando se inserta un nuevo pedido

PK	COLUMN	TYPE	NOT NULL	LEN	NOTES
TRUE	id_pedido	INT	TRUE		ID del pedido insertado
	fecha_hora	DATE TIME	TRUE		fecha y hora en la que se ejecuta el trigger
	usuario	VARCHAR	TRUE	255	Usuario de la DB que lo provoco USER()

### Tabla Log\_entrega\_orden

Tabla en la que se registra cuando una orden se cambia a "entregado"

PK	COLUMN	TYPE	NOT NULL	UNIC	LEN	NOTES
	id_orden	INT				ID de la orden entregada
	creacion	TIMESTAMP				fecha y hora en la que se ejecuta el trigger
	estado	VARCHAR			10	DEFAULT "Entregad"

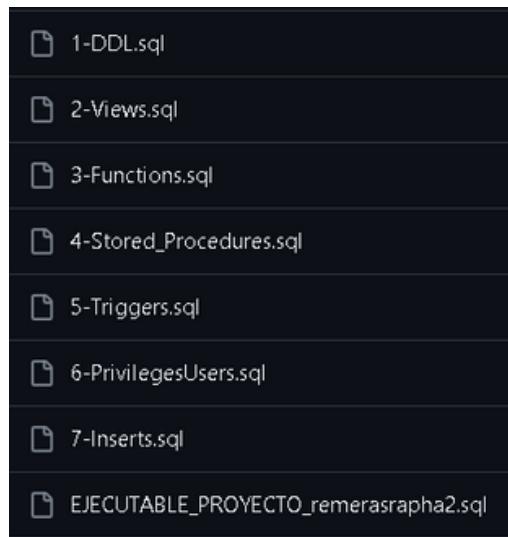
# SCRIPTS



EJECUTAR ESTE ARCHIVO:

EJECUTABLE\_PROYECTO\_REMERASRAPHA2.SQL

**ESTE ARCHIVO CONTIENE TODOS LOS SCRIPTS JUNTOS, MOSTRADOS AQUI DEBAJO, PARA EJECUTAR TODO DE UNA VEZ.**



## SCRIPT 1: 1-DDL



**En este Script, primero te encontraras con la creación de las 12 tablas. (No están incluidas la de los Triggers)**

# TABLAS:

## Tabla - Cuello

Esta tabla almacena diferentes tipos de cuellos para las remeras, con un ID único para cada tipo de cuello.



```
CREATE TABLE Cuello (
    id_cuello INT NOT NULL AUTO_INCREMENT,
    cuelloObj VARCHAR(20) NOT NULL,
    PRIMARY KEY (id_cuello)
);
```



```
CREATE TABLE Estampado (
    id_estampado INT NOT NULL
    AUTO_INCREMENT,
    estampadoObj VARCHAR(100) NOT NULL,
    PRIMARY KEY (id_estampado)
);
```

## Tabla - Talle

Guarda los tamaños disponibles para las remeras, con un ID único para cada talla.



```
CREATE TABLE Talle (
    id_talle INT NOT NULL AUTO_INCREMENT,
    talleObj VARCHAR(10) NOT NULL,
    PRIMARY KEY (id_talle)
);
```



```
CREATE TABLE Color (
    id_color INT NOT NULL AUTO_INCREMENT,
    colorObj VARCHAR(40) NOT NULL,
    PRIMARY KEY (id_color)
);
```

## Tabla - Color

Almacena la variedad de colores que se pueden utilizar en las remeras.

## **Tabla - Colores\_Elegidos**

Esta tabla registra la combinación de colores seleccionada para una id\_colores\_e específica. Puede incluir hasta tres colores diferentes y se asocia a través de una foreign keys provenientes de la tabla color.

```
CREATE TABLE Colores_Elegidos (
    id_colores_e INT NOT NULL
    AUTO_INCREMENT,
    id_color1 INT NOT NULL,
    id_color2 INT ,
    id_color3 INT ,
    PRIMARY KEY (id_colores_e),
    FOREIGN KEY (id_color1)
        REFERENCES Color (id_color),
    FOREIGN KEY (id_color2)
        REFERENCES Color (id_color),
    FOREIGN KEY (id_color3)
        REFERENCES Color (id_color)
);
```



```
CREATE TABLE Remera (
    id_remera INT NOT NULL AUTO_INCREMENT,
    id_cuello INT NOT NULL,
    id_estampado INT NOT NULL,
    id_talle INT NOT NULL,
    id_colores_e INT NOT NULL,
    PRIMARY KEY (id_remera),
    FOREIGN KEY (id_cuello)
        REFERENCES Cuello (id_cuello),
    FOREIGN KEY (id_estampado)
        REFERENCES Estampado (id_estampado),
    FOREIGN KEY (id_talle)
        REFERENCES Talle (id_talle),
    FOREIGN KEY (id_colores_e)
        REFERENCES Colores_Elegidos (id_colores_e)
);
```

## **Tabla - Remera**

Contiene Foreign keys de las tablas, cuello, estampado, talle y colores\_elegidos, para que estas se combinen creando una id\_remera, haciendo que esta se pueda colocar en un pedido.

## Tabla - Ubicacion

Guarda la información de un usuario, como el código postal y la dirección.

```
CREATE TABLE Ubicacion (
    id_ubicacion INT NOT NULL
    AUTO_INCREMENT,
    codigo_postal VARCHAR(30) NOT NULL,
    direccion VARCHAR(255) NOT NULL,
    descripcion VARCHAR(255),
    PRIMARY KEY (id_ubicacion)
);
```

```
CREATE TABLE Contacto (
    id_contacto INT NOT NULL
    AUTO_INCREMENT,
    telefono VARCHAR(100),
    email VARCHAR(255),
    PRIMARY KEY (id_contacto)
);
```

## Tabla - Usuarios

Esta tabla contiene el nombre y apellido del usuario, como también claves foráneas de su id\_ubicacion y id\_contacto.

## Tabla - Contacto

Registra información de contacto de cada usuario, como el email y el teléfono.

```
CREATE TABLE Usuarios (
    id_usuario INT NOT NULL AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    id_ubicacion INT NOT NULL,
    id_contacto INT NOT NULL,
    PRIMARY KEY (id_usuario),
    CONSTRAINT FK_ContactodeUsuario FOREIGN KEY (id_contacto)
        REFERENCES Contacto (id_contacto),
    CONSTRAINT FK_UbicacionUsuario FOREIGN KEY (id_ubicacion)
        REFERENCES Ubicacion (id_ubicacion)
);
```

```
CREATE TABLE Ordenes (
    id_orden INT NOT NULL AUTO_INCREMENT,
    id_usuario INT NOT NULL,
    fecha_de_orden DATETIME NOT NULL,
    comentario_de_pedido VARCHAR(255),
    entregado BOOLEAN NOT NULL,
    PRIMARY KEY (id_orden),
    FOREIGN KEY (id_usuario)
        REFERENCES Usuarios (id_usuario)

);
```

## Tabla - Ordenes

Contiene registros de las órdenes realizadas por los usuarios, con detalles como la fecha de la orden, un comentario opcional y un boolean que indica si la orden fue sido entregada.

## Tabla - Pedidos

Esta tabla relaciona mediante una *id\_pedido*, a una remera con una orden. Ya que el usuario puede hacer muchos pedidos en una misma orden.

```
CREATE TABLE Pedidos (
    id_pedido INT NOT NULL AUTO_INCREMENT,
    id_remera INT NOT NULL,
    id_orden INT NOT NULL,
    PRIMARY KEY (id_pedido),
    FOREIGN KEY (id_remera)
        REFERENCES Remera (id_remera),
    FOREIGN KEY (id_orden)
        REFERENCES ordenes (id_orden)

);
```

## Tabla - Review

```
CREATE TABLE Review (
    id_review INT NOT NULL AUTO_INCREMENT,
    id_remera INT NOT NULL,
    comentario VARCHAR(255),
    calificacion TINYINT CHECK
    (calificacion ≥ 1 AND calificacion ≤ 5),
    fecha_publicacion DATETIME NOT NULL,
    PRIMARY KEY (id_review),
    FOREIGN KEY (id_remera)
        REFERENCES Remera (id_remera)

);
```

Registra opiniones y calificaciones (del 1 al 5) de remeras. Incluye campos para el comentario, la fecha de publicación y se relaciona con las remeras mediante su ID. Las reseñas ayudan a los fabricantes y al negocio en general a mejorar.

# SCRIPT 2: 2-VIEWS



En este script se encuentran todas las VISTAS



```
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);
```

## Vista - Diseños\_remeras

- Esta vista devuelve información sobre las remeras, incluyendo su diseño, cuello, estampado, talle y hasta tres colores de la remera.
- Tablas: remera, cuello, estampado, talle, colores\_elegidos, color.

id_remera	cuelloObj	estampadoObj	talleObj	color1	color2	color3
1	Cuello Redondo	The Backyardingans	S	Rojo	Azul	Verde
2	Cuello Polo	Mickey Mouse	M	Azul	Amarillo	NULL
3	Cuello Polo	Darth Vader	XL	Verde	NULL	NULL
4	Cuello Alto	Lu la mejor Profes	XL	Blanco	Negro	NULL
5	Cuello Redondo	Artic Monkeys	M	Amarillo	Gris	NULL



## Vista - Colores\_remeras

- Esta vista devuelve información sobre los colores de cada remera, ya que en cada remera se usa un id\_colores\_e, esta contiene id\_color numéricos, que se relacionan con colores en si, y esa es la función que cumple la vista, mostrarnos los colores tal cual.
- Tablas: remera, colores\_elegidos, color.

```
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);
```

id_remera	id_colores_e	color1	color2	color3
1	1	Rojo	Azul	Verde
2	2	Azul	Amarillo	NULL
3	3	Verde	NULL	NULL



## Vista

### Calificacion\_remeras

- Esta vista devuelve información sobre las remeras y sus calificaciones, incluyendo el correo electrónico y el teléfono del usuario que realizó la Review.
- Tablas utilizadas: review, remera, pedidos, ordenes, usuarios, contacto.

```
CREATE OR REPLACE VIEW Calificacion_remeras AS
(SELECT REV.* ,co.email,co.telefono FROM review REV join remera REM ON REV.id_remera = REM.id_remera
join pedidos p on p.id_remera = REM.id_remera
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);
```

id_review	id_remera	comentario	calificacion	fecha_publicacion	email	telefono
1	1	Excelente remera muy comoda	5	2023-07-26 08:15:00	acaush0@shutterfly.com	45 790 7322
2	2	El estampado es genial	4	2023-07-27 14:30:00	atrenholm1@epa.gov	57 818 2814
3	3	Dudosa calidad de la remera	3	2023-07-30 09:30:00	gvenus2@ehow.com	33 211 5197
4	4	Muy comoda y elegante	5	2023-07-31 12:00:00	rchingedahls3@domainmarket.com	32 321 6187
5	6	Tremenda remera muy feliz estoy	5	2023-08-01 13:00:00	brolstone5@nps.gov	54 597 8524

### Vista - InfoparaRepartidor

- Esta vista proporciona información útil para el repartidor, incluyendo la dirección de entrega y el teléfono de los usuarios, a los cuales su orden, todavía no fue entregada.
- Tablas: ubicacion, usuarios, contacto, ordenes.

```
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 );
```

id_orden	Nombre_Apellido	direccion	descripcion	telefono
5	Bobbi Maddicks	424 Kim Court		36 654 4278
7	Ashil Scard	7 Vidon Terrace	Tocar la puerta fuerte	54 152 7355
8	Winne Oggars	22 Pleasure Street	Llamar (no funciona timbre)	54 678 7616
9	Herman Brantl	61 Melby Junction	Al lado de la puerta roja)	53 523 9950

### Vista - InformacionOrdenes

- Esta vista devuelve información general sobre los pedidos, incluyendo el email, la fecha de la orden y un comentario (si es que el usuario hizo uno).
- Tablas: pedidos, ordenes, usuarios, contacto.

```
CREATE OR REPLACE VIEW InformacionOrdenes AS
(SELECT
p.id_remera,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);
```

id_remera	id_orden	email	fecha_de_orden	comentario_de_pedido
1	1	acaush0@shutterfly.com	2023-07-24 12:34:56	Pedido para ocasión especial
2	2	atrenholm1@epa.gov	2023-07-25 10:00:00	Pedido para regalo
3	3	gvenus2@ehow.com	2023-07-28 15:45:00	Pedido para un evento
5	5	mberetta4@unc.edu	2023-07-28 15:45:00	Pedido para un evento

## Vista

### Cant\_pedidos\_por\_usuario

- Esta vista muestra la cantidad de pedidos realizados por cada usuario, ordenando la cantidad de mayor a menor, para saber quien hizo mas pedidos (hay que recordar que cada pedido es una remera).
- Tablas: usuarios, ordenes, pedidos.



```
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);
```

Nombre	cantidadPedida
Bobbi Maddicks	4
Aubree Scriver	3
Herman Brantl	2
Cherice Pettican	2

## Vista

### Cant\_ordenes\_por\_usuario

- En cambio, esta vista muestra la cantidad de órdenes que se realizaron por cada usuario (hay que recordar que en una orden, puede haber muchos pedidos, ósea muchas remeras).
- Tablas: usuarios, ordenes.



Nombre	cantidadOrdenes
Bobbi Maddicks	3
Herman Brantl	2
Aubree Scriver	2
Jobyna Trowill	1

## Vista - infoUsuarios

- Esta vista muestra información detallada sobre los usuarios, incluyendo su nombre, apellido, teléfono, correo electrónico, código postal y dirección.
- Tablas: contacto, usuarios, ubicacion.



```
CREATE OR REPLACE VIEW infoUsuarios AS (
select us.id_usuario,nombre,apellido,telefono,email,
codigo_postal,direccion from contacto c
join usuarios us on us.id_contacto = c.id_contacto
join ubicacion u on us.id_ubicacion = u.id_ubicacion
);
```

id_usuario	nombre	apellido	telefono	email	codigo_postal	direccion
1	Jobyna	Trowill	45 790 7322	acaush0@shutterfly.com	2342	111 Southridge Circle
2	Taber	Malinowski	57 818 2814	atrenholm1@epa.gov	342423	069 Becker Center
3	Kate	Pikhno	33 211 5197	gvenus2@ehow.com	67543	29 Sunnyside Terrace
4	Aubree	Scriver	32 321 6187	rchingedehals3@domainmarket.com	662133	917 Moose Alley



```
CREATE OR REPLACE VIEW top_estampados AS
SELECT estampadoObj as estampado, count(r.id_estampado) as Cantidad
FROM remera r JOIN estampado e ON r.id_estampado = e.id_estampado
GROUP BY (estampadoObj) ORDER BY count(r.id_estampado) DESC
```

estampado	Cantidad
Lu la mejor Profe	3
Queen	3
Logo Argentina	2
Artic Monkeys	2
The Backvardinians	2



### Vista - top\_estampados

- Esta vista devuelve una lista de los estampados, y al lado la cantidad de remeras, que usan este estampado, ordenado de mayor a menor, de manera que podemos saber cuales son los mas usados.
- Tablas: remera, estampado.

## SCRIPT 3: 3-FUNCTIONS



**En este script se  
encuentran todas las Funciones**



```
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_remera = P.id_remera
    WHERE R.id_estampado = estampado_id;

    RETURN cantidad;
END //
```



### Func.

#### **cantVentas\_x\_estampado**



Esta función toma como entrada el *id\_estampado* de un estampado (valga la redundancia) y devuelve la cantidad de remeras asociadas a ese estampado. En la variable *cantidad* se almacena el *count(\*)* de las remeras con ese *id\_estampado*, y se retorna esa cantidad.

remerasrapha2.cant_remeras_entregadas()
---

5
---

## Func. email\_usuario

Esta función toma como entrada el *id\_usuario* y devuelve su dirección de correo electrónico.

retornando la selección del mail de un usuario, usando un *join* para unir la tabla usuarios y contacto

```
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 $$
```

email\_usuario(4)



rchingedeals3@domainmarket.com

## Func.

### cantRemerascon\_xcolor

Esta función toma como entrada un *id\_color* y devuelve la cantidad de remeras que tienen ese color en alguna de sus tres opciones. Retorna el recuento de la selección de remeras que contengan ese color en alguno de los tres campos.



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

cantRemerascon\_xcolor(3)



7

## Func. telefono\_usuario

Esta función toma como entrada un *id\_usuario* y devuelve su número de teléfono. Lo mismo que la del mail, hace un *join* de la tabla usuarios y tabla contacto.



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

telefono\_usuario(7)



54 152 7355



```
DELIMITER $$  
CREATE FUNCTION cant_remeras_registradas()  
RETURNS INT  
READS SQL DATA  
BEGIN  
    DECLARE remeras_registradas INT;  
    SELECT COUNT(*) INTO remeras_registradas FROM remera;  
    RETURN remeras_registradas;  
END $$
```



## Func. *cant\_remeras\_registrada s*

*Esta función no tiene valor de entrada, solo devuelve la cantidad total de remeras registradas en la base de datos. Hace un count(\*) de todas las remeras en la base de datos.*

remerasrapha2.cant_remeras_registradas()
17

## Func. *cant\_remeras\_entregadas*

*Esta función tampoco tiene valor de entrada, y devuelve la cantidad de remeras que han sido entregadas.*

*Selecciona todas las remeras, y para saber si están entregadas, unea la tabla ordenes, condiciona a que entregado = TRUE, retornando así las remeras que han sido entregadas.*



```
CREATE FUNCTION cant_remeras_entregadas()  
RETURNS INT  
READS SQL DATA  
BEGIN  
    DECLARE remeras_entregadas INT;  
    SELECT count(*) INTO remeras_entregadas FROM pedidos p JOIN ordenes o ON p.id_orden = o.id_orden  
    WHERE entregado = TRUE;  
    RETURN remeras_entregadas;  
END $$
```

remerasrapha2.cant_remeras_entregadas()
5



```
DELIMITER $$  
CREATE FUNCTION cant_remeras_para_entregar()  
RETURNS INT  
READS SQL DATA  
BEGIN  
    DECLARE remeras_para_entregada INT;  
    SELECT count(*) INTO remeras_para_entregada  
    FROM pedidos p JOIN ordenes o ON p.id_orden = o.id_orden  
    WHERE entregado = FALSE;  
    RETURN remeras_para_entregada;  
END $$
```

## Func.

### *cant\_remeras\_para\_entregar*

*Lo mismo que la anterior, pero cambia en el condicionamiento, esta retorna las remeras que tengan su campo de orden entregado = FALSE.*

remerasrapha2.cant_remeras_para_entregar()
12



# SCRIPT 4:

## 4-STORED\_PROCEDURES



**En este script se  
encuentran todos los  
procedimientos almacenados**

### SP 1

#### **confirmarEntregaOrden**

Este SP permite registrar como entregada una cierta *id\_orden*, la lógica es: Si el valor booleano de entregado es 0 (FALSE), se coloca como 1 (TRUE), pero si este ya estaba entregado, te lo advierte, y deja el valor como estaba.

- Parámetros: *orderID*

```
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 //
```

orderID 9 [IN] INT



comentario  
Entrega Realizada Correctamente!

comentario  
El pedido YA fue entregado

entregado  
1

## SP 2 Usuarios\_Ordenamiento

Seleccionar y ordenar a los usuarios según nombre o apellido, y el orden (asc o desc) especificado (este campo puede ser no especificado)

- Parámetros: columna, asc\_desc.



```
DELIMITER //
create procedure Usuarios_Ordenamiento(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 //
```

columna	apellido	[IN]	varchar(50)
asc_desc	desc	[IN]	char(4)



nombre	apellido
Jobyna	Trowill
Arielle	Timny
Aubree	Scriver
Ashil	Scard
Kate	Pikhno
Cherice	Pettican
Winne	Oggers
Taber	Malinowski
Bobbi	Maddicks
Herman	Brantl

## SP 3 top\_de\_cada\_accesorio

Seleccionar y mostrar el top de un cierto accesorio seleccionado, estos pueden ser el estampado, el talle o el cuello. Este Stored Procedure contiene un “prepared statement”, que permite personalizar ampliamente las consultas.

- Parámetros: tipoAccesorio: (cuelloObj,estampadoObj,tall eObj).

tipoAccesorio estampadoObj [IN]



estampadoObj	CantUsada
Queen	3
Lu la mejor Profe	3
Darth Vader	2
Logo Argentina	2
...	...



```

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);

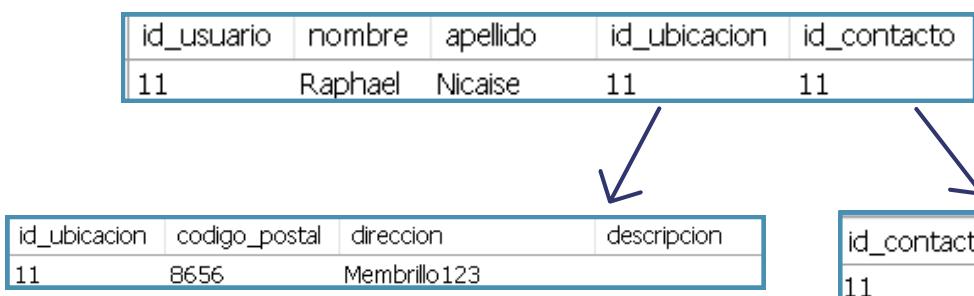
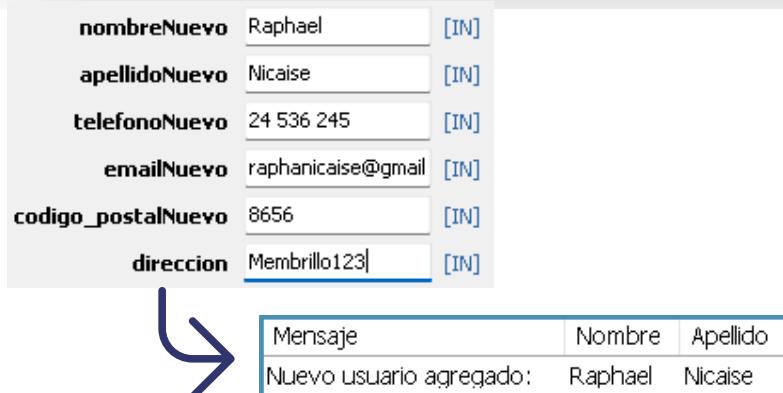
    SELECT 'Nuevo usuario agregado: ' AS Mensaje,
    nombreNuevo AS Nombre, apellidoNuevo AS Apellido;

END $$
```

## SP 4 add\_usuario

Agregar un nuevo usuario con sus datos en las tablas de contacto, ubicación, y su nombre y apellido. Si uno de estos campos esta vacío, el SP devuelve un mensaje de error. [El last\\_insert\\_id\(\)](#). lo implemento ya que al insertar valores en distintas tablas, apenas se inserta, esta función devuelve la columna auto\_increment del ultimo registro insertado. Despues estos ID se agregan al insert de la tabla usuarios.

- Parámetros:  
**nombreNuevo,**  
**apellidoNuevo,**  
**telefonoNuevo,**  
**emailNuevo,**  
**codigo\_postalNuevo,**  
**direccion .**



## SP 5 add\_remera

Agregar una nueva remera con los id de cada accesorio, incluyendo los colores, en la tabla remera. La lógica de el negocio me permite colocar solo un color, como también dos y 3. Pero si o si una, las otras pueden ser NULL.

Parámetros: cuello\_id, estampado\_id, talle\_id, color1\_id, color2\_id, color3\_id.

Detalles a agregar, las variables color2\_id y color3\_id estan seteadas como char(2), con el objetivo de, si en la llamada al SP, no se coloca ningun valor, osea '', ese id\_color se inserta como NULL. Cuando el SP espera un INT, este valor SI O SI requiere un valor como tal. Por otro lado, al insertar los ID, verificar antes cuantos registros tiene cada objeto, es decir, no vas a poder agregar un id\_cuello 54,por ejemplo.

```
DELIMITER //
CREATE PROCEDURE add_remera(
    IN cuello_id INT, IN estampado_id INT, IN talle_id INT,
    IN color1_id INT, IN color2_id CHAR(2), IN color3_id CHAR(2))
BEGIN
    DECLARE remera_id INT;
    DECLARE colores_elegidos_id INT;
    DECLARE color2 INT;
    DECLARE color3 INT;

    IF color2_id like '' THEN
        SET color2 = NULL;
    ELSE SET color2 = color2_id;
    END IF;

    IF color3_id like '' THEN
        SET color3 = NULL;
    ELSE SET color3 = color3_id;
    END IF;

    INSERT INTO Colores_Elegidos (id_color1, id_color2, id_color3)
    VALUES (color1_id, color2, color3);
    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('id_remera: ',remera_id,' agregada correctamente') as
    Mensaje;

END //
```

cuello_id	3	[IN] INT
estampado_id	2	[IN] INT
talle_id	3	[IN] INT
color1_id	5	[IN] INT
color2_id	2	[IN] INT
color3_id	4	[IN] INT



id_remera	id_cuello	id_estampado	id_talle	id_colores_e
18	3	2	3	18



id_colores_e	id_color1	id_color2	id_color3
18	5	2	4

## SP 6 add\_pedido

Asignar una remera a un usuario, permitiendo que los usuarios puedan realizar múltiples pedidos dentro de una orden. La lógica del procedimiento consiste en verificar si el usuario hace un pedido en las últimas 24 horas, si no lo hizo, se crea una nueva orden y se agrega el pedido a esta orden recién creada. Si el usuario ya hizo un pedido en ese período, el nuevo pedido se asigna a la misma orden existente. Esto facilita la agrupación de pedidos similares en una sola orden cuando son realizados en el periodo de 24hs, mejorando la eficiencia en la gestión de pedidos.

- Parámetros: `usuario_id`, `id_remera`.



```
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 //
```

<b>usuario_id</b>	11	[IN]
<b>id_remera</b>	18	[IN]



	<b>id_pedido</b>	<b>id_remera</b>	<b>id_orden</b>
	20	18	15



<b>id_orden</b>	<b>id_usuario</b>	<b>fecha_de_orden</b>	<b>comentario_de_pedido</b>	<b>entregado</b>
15	11	2023-09-25 02:33:23	NULL	0

## SP 7 add\_remera\_a\_pedido

Este SP es conveniente a usar ya que utiliza `add_remera` y `add_pedido` en una sola llamada, esto facilita el pedido, ya que creas la remera, insertas el `id_usuario` y automáticamente se hace la gestión. Cabe recalcar que este SP, llama a los 2 SP anteriormente mencionados, así que la lógica seguirá funcionando igual. Al llamar `add_remera`, se inserta la remera, `LAST_INSERT_ID()` agarra como valor el `id auto_incremental`, que posteriormente es agregado a la llamada de `add_pedido`.

Parámetros: `usuario_id_`, `estampado_id_`, `talle_id_`, `cuello_id_`, `color1_id_`, `color2_id_`, `color3_id_` (estos dos últimos como `char(2)` por lo mencionado anteriormente)

```
DELIMITER //
CREATE PROCEDURE add_remera_a_pedido (
IN usuario_id_ INT, IN estampado_id_ INT, IN talle_id_ INT,
IN cuello_id_ INT, IN color1_id_ INT, IN color2_id_ CHAR(2), IN color3_id_ CHAR(2))
BEGIN
    DECLARE nueva_id_remera INT;
    CALL add_remera(cuello_id_, estampado_id_, talle_id_, color1_id_, color2_id_, color3_id_);
    SET nueva_id_remera = LAST_INSERT_ID();

    CALL add_pedido (usuario_id_, nueva_id_remera);

    SELECT concat('La id_remera: ',nueva_id_remera,
    ' a sido pedida por el usuario: ',usuario_id_) as Message;
END //
```

usuario_id_	11	[IN] INT
estampado_id_	2	[IN] INT
talle_id_	3	[IN] INT
cuello_id_	4	[IN] INT
color1_id_	2	[IN] INT
color2_id_	6	[IN] CHAR(2)
color3_id_		[IN] CHAR(2)



Message

La id\_remera: 25 a sido pedida por el usuario: 11

id_remera	id_cuello	id_estampado	id_talle	id_colores_e
25	4	2	3	26

id_pedido	id_remera	id_orden		
21	25	15		
id_orden   id_usuario   fecha_de_orden   comentario_de_pedido		entregado		
15	11	2023-09-25 02:33:23	NULL	0

Este pedido se agrego automáticamente a la orden 15, que ya existia, ya que esta habia sido creada por el mismo usuario "11" en el plazo menor a las 24hs.

## SP 8

### edit\_colores\_elegidos

Modificar los colores de una remera en la tabla colores\_elegidos. Si queres setearle a un color -> null, no ingresar valor, osea '', la misma logica utilizada en add\_remera.

- Parámetros:

remera\_id, color1\_id,  
color2\_id, color3\_id .

```
delimiter //
CREATE PROCEDURE edit_Colores_Elegidos
(IN remera_id INT, IN color1_id INT, IN color2_id CHAR(2), IN color3_id CHAR(2))
BEGIN

DECLARE color2 INT;
DECLARE color3 INT;

IF color2_id like '' THEN
    SET color2 = NULL;
ELSE SET color2 = color2_id;
END IF;

IF color3_id like '' THEN
    SET color3 = NULL;
ELSE SET color3 = color3_id;
END IF;

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_color3 = color3
WHERE co.id_colores_e = remera_id;
SELECT concat('La id_remara: ', 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 //
```

remera_id	24	[IN] INT
color1_id	1	[IN] INT
color2_id	7	[IN] char(2)
color3_id		[IN] char(2)



Comentario	id_color1	id_color2	id_color3
La id_remara: 24 a sido modificada a los colores: 1 7			



```
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 9 get\_totalOrdenesxUsuario

Devuelve la cantidad de ordenes realizadas por el id\_usuario indicado.

- Parámetros: usuario\_id , (totalOrdenes como valor de salida).

usuario_id	11	[IN] INT
totalOrdenes		[OUT] INT



@totalOrdenes
1

## SP 10

### AgregarDescripcion\_Ubicacion

Este SP agrega una descripción de la ubicación, a un cierto *id\_usuario*.

- Parámetros: *usuario\_id*, *nueva\_descripcion\_ubicacion*.

```
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 //
```

usuario_id	11	[IN]
nueva_descripcion_ubicacion	Calle 123	[IN]



id_ubicacion	codigo_postal	direccion	descripcion
11	8656	Membrillo123	Calle 123



```
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 11 remerasxestampado

Ingresas un *id\_estampado*, y este devuelve toda la info de todas las remeras que contengan el *id\_estampado* especificado.

- Parámetros: *estampado\_id*.

estampado_id	2	[IN]
--------------	---	------



id_remera	cuelloObj	estampadoObj	talleObj	id_colores_e	color1	color2	color3
9	Cuello Alto	Queen	M	9	Verde	Verde	NULL
15	Cuello Alto	Queen	S	15	Amarillo	Verde	Gris
17	Cuello Alto	Queen	M	17	Rojo	Rojo	Blanco
18	Cuello Polo	Queen	L	18	Blanco	Azul	Amarillo
22	Cuello Redondo	Queen	L	23	Amarillo	Rojo	NULL
23	Cuello Redondo	Queen	L	24	Rojo	Negro	NULL
25	Cuello Alto	Queen	L	26	Azul	NULL	NULL
26	Cuello Redondo	Queen	L	27	Amarillo	Azul	NULL

## SP 12 get\_usuarios

Un SP muy sencillo, que al llamarlo devuelve el *id\_usuario*, *nombre* y *apellido* de todos los usuarios.



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

id_usuario	nombre	apellido
1	Jobyna	Trowill
2	Taber	Malinowski
3	Kate	Pikhno
4	Aubree	Scriver



```
DELIMITER //
CREATE PROCEDURE get_colores_remara ()
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 13 get\_colores\_remara

Otro SP muy sencillo en el que al llamarlo devuelve los colores de todas las remeras.

id_remara	id_colores_e	color1	color2	color3
1	1	Rojo	Azul	Verde
2	2	Azul	Amarillo	NULL
3	3	Verde	NULL	NULL
4	4	Blanco	Negro	NULL
5	5	Amarillo	Gris	NULL
6	6	Amarillo	Blanco	Gris

## SP 14 get\_usuarios

Agregar un nuevo estampado a la tabla estampado (si no existe).

Si este existe, devuelve "Este estampado ya existe"

- Parámetros:  
*nuevoEstampado*.



```
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 //
```

**nuevoEstampado** Phineas & Ferb [IN]



id_estampado	estampadoObj
11	Phineas & Ferb

## SP 15

### **add\_comentario\_orden**

Agregar un comentario a una orden específica mediante su *id\_orden*.  
 Parámetros: *orden\_ID*, *Comentario\_Oorden*.

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



orden_ID	15	[IN]	Comentario_Oorden	Entregar Rapido	[IN]

## SP 16 add\_review

Agregar una review

(comentario y calificación)

a una *id\_remera* específica.

- Parámetros: *remera\_id*, *comentarioNew*, *calificacionNew*.

```
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 //
```



remera_id	18	[IN]	INT	comentarioNew	Muy buena Remera	[IN]	VARCHAR(255)	calificacionNew	5	[IN]	TINYINT

## SP 17

### **obtenerReseñaPorRemera**

Obtener la reseña (review) de una remera específica mediante su ID de remera, pero antes verifica si esta *id\_remera* esta en la tabla reviews..

- Parámetros: *remeraID*.

```
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 //
```



remeraID	18	[IN]	id_remera	comentario	calificacion	fecha_publicacion
			18	Muy buena Remera	5	2023-09-25 19:59:48

## SP 18 infodeusuario

Obtiene información detallada de un usuario proveniente de la vista 'infousuarios'.

- Parámetros: *usuario\_id*

```
DELIMITER //
CREATE PROCEDURE infodeusuario (IN usuario_id INT)
BEGIN
    SELECT * FROM remerasrapha2.infousuarios
    where id_usuario = usuario_id;
END //
```



usuario_id	11	[IN]	id_usuario	nombre	apellido	telefono	email	codigo_postal	direccion
			11	Raphael	Nicase	24 536 245	raphanicaise@gmail	8656	Membrillo123

# SCRIPT 5:5-TRIGGERS



En este script se encuentran los Triggers, con sus respectivas tablas de auditoria.

## TRIGGER 1: log\_pedido

- Este trigger se dispara después de que se inserte un nuevo pedido en la tabla pedidos. Su propósito es registrar la fecha, la hora, y el usuario que inserto el pedido, en la tabla pedidos\_log como forma de auditoria.

id_pedido	fecha_hora	usuario
22	2023-09-25 12:42:08	root@localhost

Tabla Pedidos_log					
Tabla en la que se registra cuando se inserta un nuevo pedido					
PK	COLUMN	TYPE	NOT NULL	LEN	NOTES
TRUE	id_pedido	INT	TRUE		ID del pedido insertado
	fecha_hora	DATE TIME	TRUE		fecha y hora en la que se ejecuta el trigger
	usuario	VARCHAR	TRUE	255	Usuario de la DB que lo provoco USER()

```
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 //
```



```
CREATE TABLE IF NOT EXISTS 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,
    new_id_color1 INT,
    new_id_color2 INT,
    new_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', OLD.id_colores_e,
OLD.id_color1, OLD.id_color2, OLD.id_color3,
    NEW.id_color1, NEW.id_color2, NEW.id_color3);
END //
```

## TRIGGER 2: log\_colores\_elegidos\_act

- Este trigger se dispara antes de que se realice una actualización en la tabla colores\_elegidos. Su objetivo es auditar los cambios en los colores elegidos y registrarlos en la tabla colores\_elegidos\_log, en la que se guarda la fecha y hora en la que se hizo, los tres id\_color antiguos, y al lado, los 3 nuevos.

timestamp	usuario	action	old_id_colore	old_id_c	old_id_	old_id_	new_i	new_	new_
2023-09-25 03:30:13	root@localhost	UPDATE	24	1	4	NULL	1	6	NULL

Tabla Colores_elegidos_log					
Tabla en la que se registran los cambios en colores_elegidos existentes.					
PK	COLUMN	TYPE	NOT NULL	UNIQUE	LEN
	timestamp	datetime			
	usuario	VARCHAR			255 "Usuario" de la DB que lo provoco USER()
	action	VARCHAR			255 Guarda la palabra "Update"
	old_id_colores_e	INT			Guarda el id_colores_e que se cambio
	old_id_color1	INT			id_color1 antiguo
	old_id_color2	INT			id_color2 antiguo
	old_id_color3	INT			id_color3 antiguo
	new_id_color1	INT			id_color1 nuevo
	new_id_color2	INT			id_color2 nuevo
	new_id_color3	INT			id_color3 nuevo

### TRIGGER 3: log\_usuarios

- Este trigger se dispara después de que se inserte un nuevo usuario en la tabla usuarios. Su función es registrar el id del usuario y el momento en el que se creo (fecha y hora), en la tabla usuarios\_log.

```

CREATE TABLE usuarios_log (
    id_usuario INT NOT NULL,
    creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

DELIMITER //
CREATE TRIGGER log_usuarios
AFTER INSERT ON usuarios
FOR EACH ROW
BEGIN
    INSERT INTO usuarios_log
    (id_usuario) values (NEW.id_usuario);
END //

```

id_usuario	creacion
12	2023-09-25 12:40:36

Tabla log_usuarios					
Tabla en la que se registra cuando se inserta un nuevo pedido					
PK	COLUMN	TYPE	NOT NULL	UNIC LEN	NOTES
	id_usuario	INT	TRUE		ID del nuevo usuario
	creacion	TIMESTAMP			Fecha y hora en el que se creo el usuario

### TRIGGER 4: log\_entrega\_ordenes

- Este Trigger se dispara cuando se actualiza la tabla ordenes, mas específicamente cuando se actualiza la condición de entregado, por cada vez que suceda esto, chequea que cada campo de entregado nuevo, sea distinto al entregado viejo. (OLD.entregado <> NEW.entregado). Si es así, este inserta en la tabla log\_entrega\_ordenes la id\_orden que se entrego, y el momento en el que se entrego. Si OLD y NEW son iguales entonces no se procede a nada. El condicional fue puesto ya que, hay un Stored Procedure, que actualiza la tabla ordenes, pero al campo de comentario, dando a que si agregábamos un comentario, este se registraría también en log\_entrega\_ordenes, lo cual era un problema grave.

```

CREATE TABLE log_entrega_orden (
    id_orden INT ,
    creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    estado varchar(10) DEFAULT 'Entregado'
);

DELIMITER //
CREATE TRIGGER log_entrega_ordenes
AFTER UPDATE ON ordenes
FOR EACH ROW
BEGIN
    IF OLD.entregado <> NEW.entregado THEN
        INSERT INTO log_entrega_orden (id_orden) VALUES (new.id_orden);
    END IF;
END //

```

Tabla Log_entrega_orden					
Tabla en la que se registra cuando una orden se cambia a "entregado"					
PK	COLUMN	TYPE	NOT NULL	UNIC LEN	NOTES
	id_orden	INT			ID de la orden entregada
	creacion	TIMESTAMP			fecha y hora en la que se ejecuta el trigger
	estado	VARCHAR			10 DEFAULT "Entregad"

id_orden	creacion	estado
15	2023-09-25 19:58:09	Entregado

# **SCRIPT 6:**

## **6-PRIVILEGESUSERS**



**En este script se encuentra la garantizacion de privilegios a distintos usuarios en la base de datos.**

```
CREATE USER administracion@localhost IDENTIFIED BY '123456789';  
  
GRANT SELECT,INSERT,UPDATE ON remerasrapha2.* TO administracion@localhost;
```

- Crea al usuario *administracion* en el *localhost*, le asigna su contraseña, y le da la posibilidad de hacer *selects*, *inserts*, y *update* a todas las tablas de la Base de Datos.

```
CREATE USER fabrica_remeras@localhost IDENTIFIED BY 'qwertyuiop';  
  
GRANT SELECT ON colores_remeras TO fabrica_remeras@localhost;  
GRANT SELECT ON diseños_remeras TO fabrica_remeras@localhost;
```

- Crea al usuario *fabrica\_remeras* en el *localhost*, le asigna su contraseña y le da la posibilidad de hacer *Selects* a Vistas que les sirven para fabricar y diseñar las remeras.

# **SCRIPT 7:**

## **7-INSERTS**



**En este script se encuentran varios inserts con datos para hacer pruebas.**

```
(7, 54 152 7355 , 'preyshonob@nns.gov' ),  
(8,'54 678 7616','tlenard7@wikispaces.com'),  
(9,'53 523 9950','agrishakin8@redcross.org'),  
(10,'55 241 8560','dvosse9@dailymotion.com');  
  
INSERT INTO usuarios (id_usuario,nombre,apellido,id_ubicacion,id_contacto)  
VALUES (1,'Jobyna','Trowill',1,1),  
(2,'Taber','Malinowski',2,2),  
(3,'Kate','Pikhno',3,3),  
(4,'Aubree','Scriver',4,4),  
(5,'Bobbi','Maddicks',5,5),  
(6,'Arielle','Timny',6,6),  
(7,'Ashil','Scard',7,7),  
(8,'Winne','Oggers',8,8),  
(9,'Herman','Brantl',9,9),  
(10,'Cherice','Pettican',10,10);  
  
INSERT INTO ordenes (id_orden,id_usuario,fecha_de_orden,comentario_de_pedido,entregado)  
VALUES (1,1,"2023-07-24 12:34:56","Pedido para ocasión especial",1),  
(2,2,"2023-07-25 10:00:00","Pedido para regalo",1),  
(3,3,"2023-07-28 15:45:00","Pedido para un evento",1),
```

# **SCRIPT 8:**



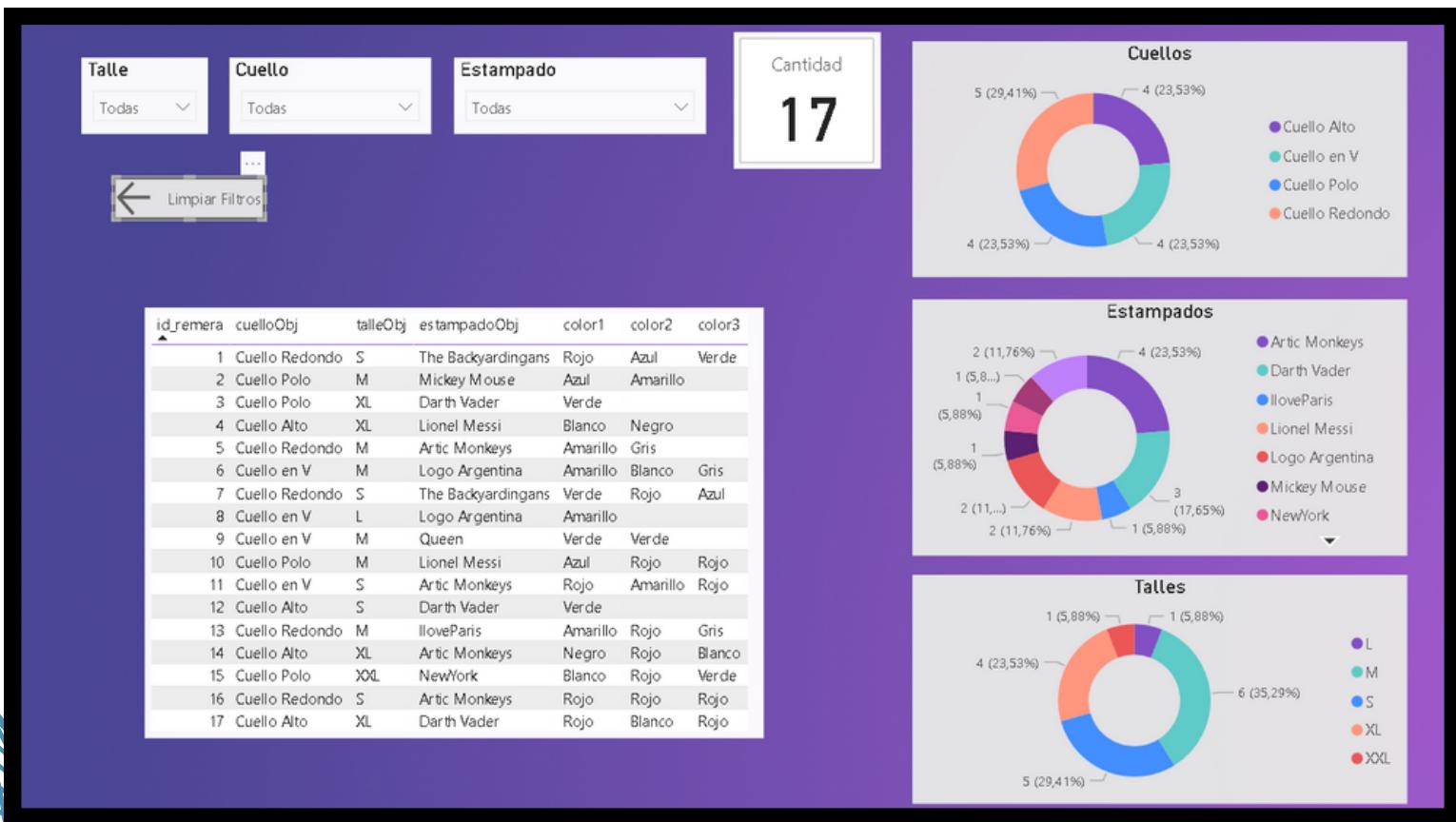
**EJECUTABLE\_PROYECTO\_REMERASRAPHA2**

**En este script se encuentran TODOS los scripts, contenido:**

- Las Tablas.
- Las Vistas.
- Las Funciones.
- Los Stored Procedures.
- los Triggers.
- Los Privilegios de usuarios.
- Los inserts

# INFORMES QUE DAN VALOR A LA DB

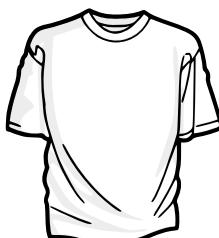
- Este DASHBOARD, creado en **Power BI**, mediante una conexión a la base de datos **MySQL**, permite visualizar todas las remeras registradas, con algunos graficos, ademas de tener unos filtros, permitiendo asi ver cuantas remeras tienen cierto estampado, o cierto talle.



- Con la vista Clasificación Remeras, se pueden observar las calificaciones (valga la redundancia), de todas las remeras. Al contener id\_remera, esto abre mil puertas a fines estadísticos, como por ejemplo: Cual es el promedio de clasificación según el estampado, según que cantidad de colores tiene, etc. Fines estadísticos con el objetivo de mejorar la calidad y la productividad del negocio

id_review	id_remera	comentario	calificacion	fecha_publicacion	email	telefono
1	1	Excelente remera muy comoda	5	2023-07-26 08:15:00	acaush0@shutterfly.com	45 790 7322
2	2	El estampado es genial	4	2023-07-27 14:30:00	atrenholm1@epa.gov	57 818 2814
3	3	Dudosa calidad de la remera	3	2023-07-30 09:30:00	gvenus2@ehow.com	33 211 5197
4	4	Muy comoda y elegante	5	2023-07-31 12:00:00	rchingedeals3@domainmarket.com	32 321 6187
5	6	Tremenda remera muy feliz estoy	5	2023-08-01 13:00:00	brolstone5@nps.gov	54 597 8524

- **A futuro**, me gustaría que esta Base de datos desarrolle mas ramas de productos, es decir pantalones, camisas, etc, que también sean personalizables, me gustaría además agregar una Gestión de Compras, para mejorar la seguridad “contable” del negocio, y para tener un informe mas general de cuanto generaría la empresa, las ganancias y perdidas, etc. A esto le implementaría una manera, que, dependiendo que accesorios uses, se le asigne un precio, agregándole los costos de producción y valores agregados.



Diagrams



PowerBI

# GRACIAS!