CODERHOUSE

Proyecto Final SQL

Nuñez Juan Ignacio

CODERHOUSE

PROYECTO FINAL SQL

1.DESCRIPCIÓN DE LA TEMÁTICA.

Introducción.

Se creará en una base de datos, en la cual se buscará implementar un modelo relacional que represente procesos de una tienda que comercializa productos para la construcción. La misma lleva un seguimiento rudimentario de los artículos que lleva en stock, así como todavía utiliza fichas para almacenar la información de clientes, proveedores y facturas. Se buscará sistematizar estos procesos permitiendo un mejor funcionamiento general a través del uso del lenguaje SQL.

Objetivos.

- Poder gestionar el stock de mercaderías para la gerencia de comercialización.
- Control de las cobranzas de las ventas y los medios por los cuales se efectúan los pagos.
- Contar un manejo efectivo de los proveedores y los clientes del negocio.

Problemáticas.

Se busca contar con la información ordenada y accesible para responder preguntas tales como:

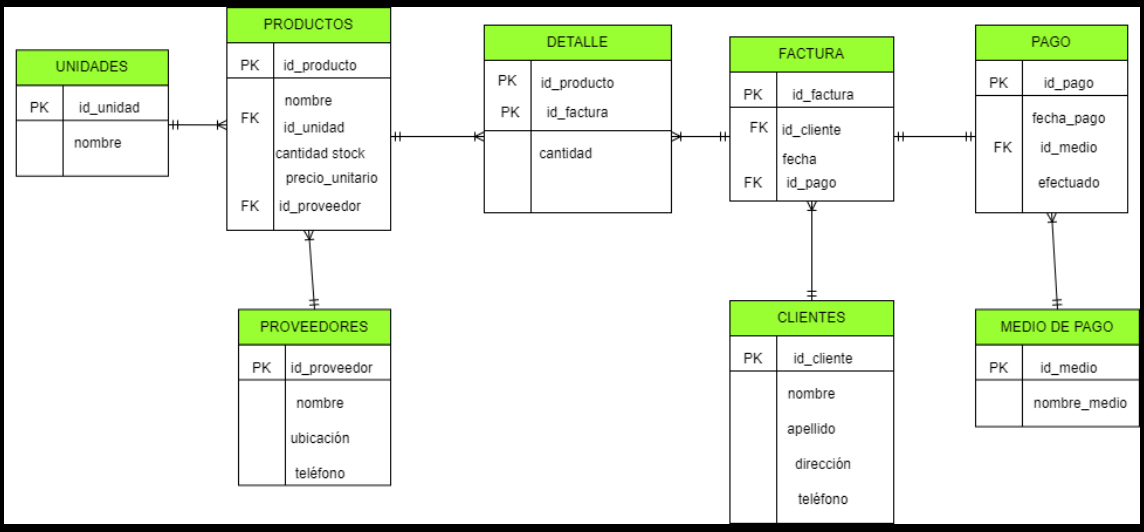
- ¿Cuál es el nivel de ventas que se tienen? ¿Alcanza el stock promedio que presentamos?
- ¿Cuáles son los productos más demandados por los clientes? ¿Qué medios de pagos prefieren?

- ¿Cuál es el nivel de incobrabilidad que están teniendo nuestros clientes?

---

2. DIAGRAMA ENTIDAD RELACIÓN

El modelo de datos entidad-relación (E-R) se basa en la percepción del mundo real que consiste en un conjunto de objetos básicos, denominados entidades, y de las relaciones entre esos objetos. Siguientemente mostramos el diagrama que relaciona a las distintas entidades que conformaran el modelo de negocio del caso analizado.



---

3. TABLAS

Cada una de las entidades incorporadas en el diagrama anterior, van a representar una tabla en nuestro modelo relacional. A continuación, realizamos el listado de las tablas que utilizaremos, con su correspondiente descripción y los elementos que posee cada una, así como sus atributos.

CODERHOUSE

Tabla	UNIDADES							
	Contiene el nombre de las unidades en que se miden los productos							
KEY	**COLUMN**	**TYPE**	**LENGHT**	**NOT NULL**	**UNIQUE**	**DEFAULT**	**AUTO_INCREMENT**	**NOTES**
PK	id_unidad	INT		X	X		X	id de cada unidad
	nombre	VARCHAR	50	X				nombre de la unidad

Tabla	PRODUCTOS							
	Contiene información sobre cada producto en stock							
KEY	**COLUMN**	**TYPE**	**LENGHT**	**NOT NULL**	**UNIQUE**	**DEFAULT**	**AUTO_INCREMENT**	**NOTES**
PK	id_producto	INT		X	X		X	id de cada producto
	nombre	VARCHAR	50	X				nombre del producto
FK	id_unidad	INT		X				id de la unidad
	cant_stock	INT		X				cantidad en stock
	Precio_unitario	DECIMAL (8,2)		X				Precio unitario del producto
FK	id_proveedor	INT		X				id del proveedor

CODERHOUSE

Tabla	PROVEEDORES							
	Contiene información sobre los proveedores							
KEY	**COLUMN**	**TYPE**	**LENGHT**	**NOT NULL**	**UNIQUE**	**DEFAULT**	**AUTO_INCREMENT**	**NOTES**
PK	id_proveedor	INT		X	X		X	id del proveedor
	nombre	VARCHAR	50	X				nombre del proveedor
	ubicación	VARCHAR	50	X				direccion proveedor
	telefono	INT		X				telefono de contacto proveedor

Tabla	DETALLE							
	Contiene los detalles de la venta							
KEY	**COLUMN**	**TYPE**	**LENGHT**	**NOT NULL**	**UNIQUE**	**DEFAULT**	**AUTO_INCREMENT**	**NOTES**
PK-FK	id_producto	INT		X	X			id del producto
PK-FK	id_factura	INT		X	X			id de la factura
	cantidad	INT		X				cantidad comprada

CODERHOUSE

Tabla	FACTURA							
	Contiene la información de las facturas							
KEY	**COLUMN**	**TYPE**	**LENGHT**	**NOT NULL**	**UNIQUE**	**DEFAULT**	**AUTO_INCREMENT**	**NOTES**
PK	id_factura	INT		X	X		X	id de la factura
FK	id_cliente	INT		X				id del cliente
	fecha	DATE		X				fecha de la factura
FK	id_pago	INT		X				id del pago

Tabla	CLIENTES							
	Contiene la información de los clientes							
KEY	**COLUMN**	**TYPE**	**LENGHT**	**NOT NULL**	**UNIQUE**	**DEFAULT**	**AUTO_INCREMENT**	**NOTES**
PK	id_cliente	INT		X	X		X	id del cliente
	nombre	VARCHAR	50	X				nombre del cliente
	apellido	VARCHAR	50	X				apellido del cliente
	direccion	VARCHAR	50	X				dirección del cliente
	telefono	INT		X				telefono de contacto del cliente

CODERHOUSE

Tabla	PAGO							
	Contiene la información de los pagos realizados							
KEY	**COLUMN**	**TYPE**	**LENGHT**	**NOT NULL**	**UNIQUE**	**DEFAULT**	**AUTO_INCREMENT**	**NOTES**
PK	id_pago	INT		X	X		X	id del pago realizado
	fecha_pago	DATE						fecha de pago
FK	id_medio	INT		X	X			id del medio de pago
	efectuado	BOOL		X				Indica si se efectuo o no el pago

Tabla	MEDIO DE PAGO							
	Contiene las distintas opciones de pago							
KEY	**COLUMN**	**TYPE**	**LENGHT**	**NOT NULL**	**UNIQUE**	**DEFAULT**	**AUTO_INCREMENT**	**NOTES**
PK	id_medio	INT		X	X		X	id del medio de pago
	nombre	VARCHAR	50					nombre medio de pago

4. Archivo SQL

En el siguiente link se encuentra el código que nos permite crear nuestro modelo de base de datos relacional y sus correspondientes implementaciones:

https://github.com/juaninz/ProyectoFinalSQL.git

Código de creación de base de datos y tablas:

```
CREATE SCHEMA tiendacorralon;

USE tiendacorralon;

CREATE TABLE unidades (
  id_unidad INT PRIMARY KEY NOT NULL auto_increment UNIQUE,
  nombre VARCHAR(50) NOT NULL
);


CREATE TABLE proveedores (
  id_proveedor INT PRIMARY KEY NOT NULL auto_increment UNIQUE,
  nombre VARCHAR(50) NOT NULL,
  ubicacion VARCHAR(50) NOT NULL,
  telefono INT NOT NULL
);


CREATE TABLE productos (
  id_producto INT PRIMARY KEY NOT NULL auto_increment UNIQUE,
  nombre VARCHAR(50) NOT NULL,
  id_unidad INT NOT NULL,
  cant_stock INT NOT NULL,
  id_proveedor INT NOT NULL,
  FOREIGN KEY (id_unidad) REFERENCES unidades(id_unidad),
  FOREIGN KEY(id_proveedor) REFERENCES proveedores(id_proveedor)
);
```

```
CREATE TABLE clientes (
  id_cliente INT PRIMARY KEY NOT NULL auto_increment UNIQUE,
  nombre VARCHAR(50) NOT NULL,
  apellido VARCHAR(50) NOT NULL,
  direccion VARCHAR(50) NOT NULL,
  telefono INT NOT NULL
);

CREATE TABLE medio_de_pago (
  id_medio INT PRIMARY KEY NOT NULL auto_increment UNIQUE,
  nombre VARCHAR(50) NOT NULL
);

CREATE TABLE pago (
  id_pago INT PRIMARY KEY NOT NULL auto_increment UNIQUE,
  fecha_pago DATE NOT NULL,
  id_medio INT NOT NULL,
  efectuado BOOL NOT NULL,
  FOREIGN KEY(id_medio) REFERENCES medio_de_pago(id_medio)
);

CREATE TABLE factura (
  id_factura INT PRIMARY KEY NOT NULL auto_increment UNIQUE,
  fecha DATE NOT NULL,
  id_cliente INT NOT NULL,
  id_pago INT NOT NULL,
  FOREIGN KEY(id_cliente) REFERENCES clientes(id_cliente),
  FOREIGN KEY(id_pago) REFERENCES pago(id_pago)
);
```

```
CREATE TABLE detalle (
  id_factura INT NOT NULL,
  id_producto INT NOT NULL,
  cantidad INT NOT NULL,
  precio INT NOT NULL,
  FOREIGN KEY(id_factura) REFERENCES factura(id_factura),
  FOREIGN KEY(id_producto) REFERENCES productos(id_producto)
);
```

INSERCIÓN DE REGISTROS

```
INSERT INTO clientes (nombre, apellido, direccion, telefono) VALUES
    ('Juan', 'Pérez', 'Calle 123', '3624123456'),
    ('María', 'González', 'Avenida 456', '3794123456'),
    ('Pedro', 'López', 'Calle 789', '3624987654'),
    ('Luisa', 'Martínez', 'Avenida 012', '3794987654'),
    ('Carlos', 'Rodríguez', 'Calle 345', '3624765432'),
    ('Ana', 'Hernández', 'Avenida 678', '3794765432'),
    ('José', 'Gómez', 'Calle 901', '3624234567'),
    ('Laura', 'Fernández', 'Avenida 234', '3794234567'),
    ('Miguel', 'Sánchez', 'Calle 567', '3624123458'),
    ('Sofía', 'Ramírez', 'Avenida 890', '3794123458'),
    ('Daniel', 'Torres', 'Calle 111', '3624987659'),
    ('Fernanda', 'Silva', 'Avenida 222', '3794987659'),
    ('Alejandro', 'Chávez', 'Calle 333', '3624765430'),
    ('Valentina', 'Rojas', 'Avenida 444', '3794765430'),
    ('Andrés', 'Pereira', 'Calle 555', '3624234561'),
    ('Gabriela', 'López', 'Avenida 666', '3794234561'),
    ('Martín', 'García', 'Calle 777', '3624123472'),
    ('Carolina', 'Torres', 'Avenida 888', '3794123472'),
```

```
    ('Luis', 'Mendoza', 'Calle 999', '3624987613'),
    ('Ana', 'Romero', 'Avenida 000', '3794987613');

INSERT INTO proveedores (nombre, ubicacion, telefono) VALUES
    ('ConstruMax', 'Calle de la Construcción 123', '3624123456'),
    ('Estructuras Sólidas', 'Avenida del Progreso 456', '3624987654'),
    ('Construcciones Modernas', 'Calle de la Innovación 789', '3794123456'),
    ('Todo para su obra', 'Avenida del Arte 012', '3624765432'),
    ('ConstruPlaza', 'Calle del Proyecto 345', '3624234567'),
    ('Materiales Constructivos', 'Avenida de la Edificación 678', '3794987654'),
    ('ConstruTech', 'Calle de la Tecnología 901', '3624321654'),
    ('Bercomat', 'Avenida de la Ingeniería 234', '3794234567'),
    ('ConstruHouse', 'Calle del Hogar 567', '3624123458'),
    ('Urbanización y Obras', 'Avenida de la Urbanidad 890', '3624123459');

INSERT INTO unidades (nombre) VALUES
    ('Kg'),
    ('m3'),
    ('un'),
    ('rollos x 50m'),
    ('m'),
    ('bolsas x 50 kg'),
    ('envase 20lts'),
    ('barras x 12m');

INSERT INTO productos (nombre, id_unidad, cant_stock, precio_unitario,
id_proveedor) VALUES
    ('Cemento Portland', 6 , 200 , 11500 , 2 ),
    ('Ladrillos', 3 , 10000 ,200, 7 ),
```

```
    ('Cal', 6 , 250 , 10700 , 1 ),
    ('Hierro del 6', 8 , 50, 20500 , 6 ),
    ('Hierro del 8', 8 , 25 , 33500, 6 ),
    ('Hierro del 10', 8 , 30 , 48500, 6 ),
    ('Hierro del 12', 8 , 40 , 66700, 6 ),
    ('Baldosa Cerámica 30x30', 3 , 5500, 18000 , 5),
        ('Baldosa de porcelanato 60x60', 3 , 4500 , 25000, 5),
    ('Arena', 2 , 6000 , 20000, 10 ),
    ('Piedra', 2 , 5000 , 25000, 10 ),
    ('Pintura ALBA blanca', 7 , 50 ,35000, 9 ),
    ('Malla Cima', 3 , 500 , 12000, 4),
    ('Chapa Trapezoidal 1x1m', 3 , 800 , 7000, 2 ),
    ('Perfil C 100', 3 , 300 , 27726, 5 ),
    ('Perfil C 120', 3 , 250 , 33519, 5 );

INSERT INTO medio_de_pago(nombre) VALUES
    ('efectivo' ),
    ('tarjeta de debito' ),
    ('tarjeta de credito' ),
    ('cuenta corriente' );

INSERT INTO pago (fecha_pago, id_medio, efectuado) VALUES
    ('2023-05-10', 1, TRUE),
    ('2023-05-11', 3, FALSE),
    ('2023-05-12', 3, TRUE),
    ('2023-05-13', 4, TRUE),
    ('2023-05-14', 4, FALSE),
    ('2023-05-15', 2, TRUE),
    ('2023-05-16', 3, FALSE),
```

```
    ('2023-05-17', 4, TRUE),
    ('2023-05-18', 1, TRUE),
    ('2023-05-19', 4, FALSE);

INSERT INTO factura (fecha, id_cliente, id_pago) VALUES
    ('2023-04-25', 1, 1),
    ('2023-04-26', 2, 2),
    ('2023-04-27', 3, 3),
    ('2023-04-28', 4, 4),
    ('2023-04-29', 5, 5),
    ('2023-04-30', 6, 6),
    ('2023-05-01', 7, 7),
    ('2023-05-02', 8, 8),
    ('2023-05-03', 9, 9),
    ('2023-05-04', 10, 10);

INSERT INTO detalle (id_factura, id_producto, cantidad) VALUES
    (1,1, 50),
    (2, 2, 1200),
    (3, 1, 20),
    (4, 4, 12),
    (5, 4, 15),
    (6, 10, 15),
    (7, 10, 20),
    (8, 11, 10),
    (9, 11, 20),
    (10, 15, 10);
```

VISTAS

top_productos_$: vista para ver cuánto se vendió en términos de dinero cada producto. Trabaja en las tablas de productos y detalle

```
CREATE VIEW top_productos_$ AS

SELECT d.id_producto, SUM(d.cantidad), pr.nombre, pr.precio_unitario,
SUM(d.cantidad * pr.precio_unitario) as total

FROM detalle d

JOIN productos pr ON d.id_producto = pr.id_producto

GROUP BY d.id_producto

ORDER BY total DESC;
```

top_productos_Q : vista para ver cuánto se vendió en términos de cantidad. Trabaja en las tablas de productos y detalle.

```
CREATE VIEW top_productos_Q AS

SELECT d.id_producto, SUM(d.cantidad) as cantidad_total , pr.nombre ,
pr.precio_unitario

FROM detalle d

JOIN productos pr on d.id_producto = pr.id_producto

group by pr.id_producto

order by SUM(d.cantidad) desc;
```

top_clientes: vista para ver que cliente es el que más aportó a las ventas (en este caso cliente con el id 9 total: 50 millones). Trabaja en las tablas de clientes y factura.

```
CREATE VIEW top_clientes AS

SELECT *,

(SELECT precio_unitario FROM productos where productos.id_producto =
detalle.id_producto) as pu,

(SELECT id_cliente FROM factura where detalle.id_factura = factura.id_factura) as
idcliente,

(SELECT precio_unitario FROM productos where productos.id_producto =
detalle.id_producto)*cantidad as total

FROM detalle
```

```
group by (SELECT id_cliente FROM factura where detalle.id_factura =
factura.id_factura)

order by (SELECT precio_unitario FROM productos where productos.id_producto =
detalle.id_producto)*cantidad desc;
```

topMediosDePago: vista para ver que medios de pago fueron los más elegido. Trabaja en las tablas de medio_de_pago y pago.

```
CREATE VIEW topMediosDePago AS

SELECT m.nombre,count(*) as veces_usadas

FROM medio_de_pago m

JOIN pago p on m.id_medio = p.id_medio

group by m.id_medio

order by count(*) desc;
```

PuntoPedidoConstrumax: vista para ver qué productos tienen un stock mayor a 100 y el proveedor es construmax. Trabaja en la tabla proveedores y productos.

```
CREATE VIEW PuntoPedidoConstrumax AS

SELECT * from productos

where cant_stock< 1000 and id_proveedor = (select id_proveedor from
proveedores where nombre="ConstruMax");
```

FUNCIONES

FUNCION PARA CALCULAR EL PRECIO : La misma actúa en las tablas precio y detalle para poder calcular los importes totales netos

```
DELIMITER $$

CREATE FUNCTION calcularPrecioNeto(precio DECIMAL(10, 2), cantidad INT)
RETURNS DECIMAL(10, 2)

DETERMINISTIC

BEGIN

    DECLARE precioNeto DECIMAL(10, 2);
```

```
    SET precioNeto = precio * cantidad;

    RETURN precioNeto;
END$$
DELIMITER ;
```

FUNCIÓN PARA CALCULAR PRECIO CON IVA: La misma actúa en las tablas precio y detalle para poder calcular los importes totales con iva

```
DELIMITER $$
CREATE FUNCTION calcularPrecioConIVA(precio DECIMAL(10, 2), cantidad INT)
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    DECLARE precioNeto DECIMAL(10, 2);
    DECLARE precioConIVA DECIMAL(10, 2);

    SET precioNeto = precio * cantidad;
    SET precioConIVA = precioNeto * 1.21; -- Se asume un IVA del 21%

    RETURN precioConIVA;
END$$
DELIMITER ;

SELECT
    (SELECT precio_unitario FROM productos WHERE productos.id_producto =
detalle.id_producto) AS pu,
    (SELECT nombre FROM productos WHERE productos.id_producto =
detalle.id_producto) AS producto,
    cantidad,
    calcularPrecioNeto((SELECT precio_unitario FROM productos WHERE
productos.id_producto = detalle.id_producto), cantidad) AS precioNeto,
```

```
    calcularPrecioConIVA((SELECT precio_unitario FROM productos WHERE
productos.id_producto = detalle.id_producto), cantidad) AS precioConIVA
FROM detalle;
```

FUNCIÓN PARA DEFINIR SI UN CLIENTE ES O NO MOROSO : la misma actua en la tabla pago y clasifica según criterio de fecha si un cliente es o no moroso

```
delimiter //
create function f_tipoCliente(
  fecha_hoy DATE,
  fecha_pago DATE,
  efectuado bool
  )
  returns varchar(20)
  deterministic
begin
        case
   when fecha_hoy <= fecha_pago then
    return 'con margen';
   when fecha_hoy > fecha_pago  and efectuado = true then
    return 'Cumplidor';
   when fecha_hoy > fecha_pago  and efectuado = false then
    return 'Moroso';
        end case;
 end //
 delimiter ;

SELECT
   f.id_cliente,
   p.fecha_pago,
   p.efectuado,
```

```
    f_tipoCliente('2023-05-15', p.fecha_pago, p.efectuado) AS tipoCliente
FROM pago p
JOIN factura f ON f.id_pago = p.id_pago;
```

STORED PROCEDURES

OrdenarTabla: El siguiente sp permite ordenar una tabla según un campo de ordenamiento y especificar si el orden es ascendente o descendente

```
DELIMITER //

CREATE PROCEDURE OrdenarTabla(
    IN tableName VARCHAR(255),
    IN sortColumn VARCHAR(255),
    IN sortOrder VARCHAR(4)
)
BEGIN
    SET @query = CONCAT('SELECT * FROM ', tableName, ' ORDER BY ', sortColumn, ' ', sortOrder);
    PREPARE stmt FROM @query;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
END //
DELIMITER ;
```

Para usarlo se debe proporcionar el nombre de la tabla en la que se desea ordenar los datos,

el nombre de la columna de ordenamiento y el orden ("ASC" para ascendente o "DESC" para descendente).

```
CALL OrdenarTabla('productos', 'precio_unitario', 'DESC');
```

InsertarEliminarProveedor: El siguiente sp permite insertar registros en una tabla y eliminar un registro específico de la misma

```
DELIMITER //
CREATE PROCEDURE InsertarEliminarProveedor(
```

```
    IN p_id INT,
    IN p_nombre VARCHAR(50),
    IN p_ubicacion VARCHAR(50),
    IN p_telefono VARCHAR(20),
    IN p_accion VARCHAR(10)
)
BEGIN
    IF p_accion = 'INSERT' THEN
        INSERT INTO proveedores (nombre, ubicacion, telefono) VALUES (p_nombre,
p_ubicacion, p_telefono);
    ELSEIF p_accion = 'DELETE' THEN
        DELETE FROM proveedores WHERE id_proveedor = p_id;
    END IF;
END //
DELIMITER;
```

En este sp se utilizan los siguientes parámetros:

p_id: el ID del proveedor que deseas eliminar.

p_nombre: el nombre del proveedor que deseas insertar.

p_ubicacion: la ubicación del proveedor que deseas insertar.

p_telefono: el número de teléfono del proveedor que deseas insertar.

p_accion: la acción que se desea realizar, puede ser 'INSERT' para insertar un nuevo proveedor o 'DELETE' para eliminar un proveedor existente.

En el caso de la inserción se colocará NULL en el id_proveedor debido a que este es autoincremental

```
CALL InsertarEliminarProveedor(NULL, 'GIRONA MATERIALES', 'Ubicación 1',
'1234567890', 'INSERT');
SELECT * FROM proveedores;
```

Para el caso de la eliminación se dejarán vacíos el resto de los campos que nos sean el id que se quiere eliminar.

```
CALL InsertarEliminarProveedor(11, '', '', '', 'DELETE');
```

TRIGGERS

Creacion tabla de auditoría de productos:

```
CREATE TABLE products_audits (
        id_log INT PRIMARY KEY auto_increment,
   entity varchar(100),
   entity_id int,
   insert_dt datetime,
   created_by varchar(100),
   last_update_dt datetime,
   last_updated_by varchar(100)
);
```

Trigger que inserta valores en la tabla de auditoria para registrar los movimientos de insercion en la tabla productos:

```
CREATE TRIGGER `tr_insert_product_aud`
AFTER INSERT ON `productos`
FOR EACH ROW
INSERT INTO `products_audits`(entity, entity_id, insert_dt, created_by, last_update_dt, last_updated_by)
VALUES ('product', NEW.id_producto,CURRENT_TIMESTAMP(), USER(), CURRENT_TIMESTAMP(), USER());


INSERT INTO productos (nombre, id_unidad, cant_stock, precio_unitario, id_proveedor) VALUES
   ('Cinto Métrica', 3 , 20 , 1000 , 2 );
SELECT * FROM products_audits;
```

Trigger que actualiza automáticamente la tabla de auditoría de un producto luego de un update.

```
CREATE TRIGGER `tr_update_product_aud`
AFTER UPDATE ON `productos`
FOR EACH ROW
```

```
UPDATE `products_audits` SET last_update_dt = CURRENT_TIMESTAMP(),
last_updated_by = USER()

WHERE entity_id = OLD.id_producto;

SET SQL_SAFE_UPDATES = 0;

UPDATE productos SET nombre = 'Cinta Multifuncion' WHERE id_producto = 23;


SELECT * FROM products_audits;
```

Trigger de verificación de duplicados

```
DELIMITER //

CREATE TRIGGER trigger_evitar_duplicados

BEFORE INSERT ON productos

FOR EACH ROW

BEGIN

   -- Verificar si ya existe un registro con el mismo nombre

   IF EXISTS (SELECT 1 FROM productos WHERE nombre = NEW.nombre) THEN

      SIGNAL SQLSTATE '45000'

         SET MESSAGE_TEXT = 'No se puede insertar el registro. Ya existe un
producto con el mismo nombre.';

   END IF;

END//

DELIMITER ;


INSERT INTO productos (nombre, id_unidad, cant_stock, precio_unitario,
id_proveedor) VALUES

   ('Cinto Métrica', 3 , 20 , 1500 , 2 );
```

AUDITORIA TABLA PROVEEDOR

creacion tabla de auditoria de proveedores:

```
CREATE TABLE prov_audits (

         id_log INT PRIMARY KEY auto_increment,
```

```
    entity varchar(100),
    entity_id int,
    insert_dt datetime,
    created_by varchar(100),
    last_update_dt datetime,
    last_updated_by varchar(100)
);
```

Trigger que inserta valores en la tabla de auditoria para registrar los movimientos de insercion en la tabla proveedores

```
CREATE TRIGGER `tr_insert_prov_aud`
AFTER INSERT ON `proveedores`
FOR EACH ROW
INSERT INTO `prov_audits`(entity, entity_id, insert_dt, created_by, last_update_dt, last_updated_by)
VALUES ('proveedor', NEW.id_proveedor,CURRENT_TIMESTAMP(), USER(), CURRENT_TIMESTAMP(), USER());
```

```
INSERT INTO proveedores (nombre, ubicacion, telefono) VALUES
    ('Strike Corralon', 'Belgrano 1268', '3624970408');
SELECT * FROM prov_audits;
```

Trigger que actualiza automáticamente la tabla de auditoria de un proveedor luego de un update.

```
CREATE TRIGGER `tr_update_prov_aud`
AFTER UPDATE ON `proveedores`
FOR EACH ROW
UPDATE `prov_audits` SET last_update_dt = CURRENT_TIMESTAMP(), last_updated_by = USER()
WHERE entity_id = OLD.id_proveedor;
```

```
SET SQL_SAFE_UPDATES = 0;

UPDATE proveedores SET nombre = 'Strikes Materiales' WHERE ubicacion =
"Belgrano 1268";


SELECT * FROM prov_audits;


Trigger de verificacion de duplicados

DELIMITER //

CREATE TRIGGER trigger_evitar_duplicados_prov

BEFORE INSERT ON proveedores

FOR EACH ROW

BEGIN

   -- Verificar si ya existe un registro con el mismo nombre

   IF EXISTS (SELECT 1 FROM proveedores WHERE nombre = NEW.nombre) THEN

      SIGNAL SQLSTATE '45000'

         SET MESSAGE_TEXT = 'No se puede insertar el registro. Ya existe un
proveedor con el mismo nombre.';

   END IF;

END//

DELIMITER ;


INSERT INTO proveedores (nombre, ubicacion, telefono) VALUES

   ('Strikes Materiales', 'Belgrano 1268', '3624970408');
```

CODERHOUSE

INFORME GENERADO A PARTIR DE LA INFORMACIÓN DE LA BASE

HERRAMIENTAS Y TECNOLOGÍAS UTILIZADAS

- MySQL
- MySQL Workbench
- Power BI
- Excel

FUTURAS LÍNEAS.

Se pudo realizar la estructura de una base de datos que permita solucionar los requerimientos que posee la empresa de venta de materiales de construcción, pero se podría seguir depurando el proyecto agregando elementos que sean de utilidad tales como la cotización de la moneda extranjera, fechas de entrega de los productos o costos que presenta la empresa además de los de adquisición de las mercaderías.