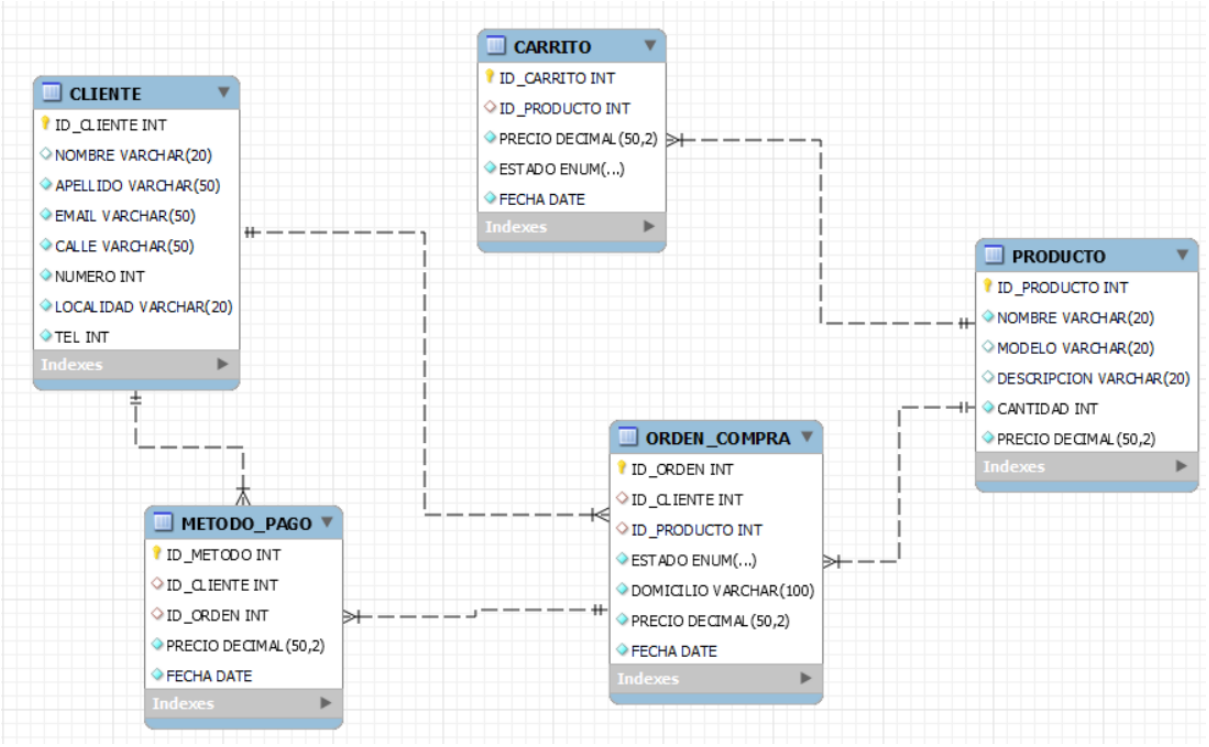
Proyecto: Base de Datos para E-commerce

La idea es diseñar una base de datos de un e-commerce de electrodomésticos (proyecto final presentado en el curso de JavaScript realizado anteriormente).
Dentro de la base de datos se almacenarán los clientes (id, nombre, e-mail, productos comprados, domicilio) y los productos (id, nombre producto, modelo, descripción, imagen, cantidad, precio).

La finalidad es por un lado que el usuario tenga registro de los productos que ya compró y que a su vez el e-commerce lleve un registro de los productos vendidos, cantidad en stock, variación de precios, etc.

Diagrama ER:



Tablas:
En la tabla Cliente se almacenarán los datos de los clientes que se vayan registrando en la plataforma ya que es un requisito necesario para comprar productos. Esta tabla estará asociada a la orden de compra que generará el cliente y al método de pago que elija (puede ser Débito o Crédito). La tabla Productos almacena los datos de los productos (id, descripcion, stock). La orden de compra será generada a partir de los productos que el cliente agregue al carrito y se relaciona con el método de pago elegido para la compra.

Vistas Funciones y Stored Procedures:

Las vistas creadas son las siguientes:

- ESTADO_OC_CANCEL: Permite ver el estado de las Órdenes de Compra que fueron canceladas:

```
CREATE VIEW ESTADO_OC_CANCEL AS (
SELECT ID_ORDEN, ESTADO, FECHA FROM ORDEN_COMPRA
WHERE ESTADO = "CANCELADA"
);
```

- CLIENTES_CABA: Permite ver los clientes que residen en la Ciudad Autónoma de Bs As.

```
CREATE VIEW CLIENTES_CABA AS (
SELECT ID_CLIENTE, APELLIDO, EMAIL, LOCALIDAD FROM CLIENTE
WHERE LOCALIDAD = "CABA"
);
```

- PRODUCTOS_FALTA: Permite ver los productos que estén por debajo de X cantidad, con el fin de ver si hay que reponer el stock.

```
CREATE VIEW PRODUCTOS_FALTA AS (
SELECT ID_PRODUCTO, NOMBRE, CANTIDAD FROM PRODUCTO
WHERE CANTIDAD <= 30
ORDER BY CANTIDAD
);
```

- DEBITO_2023: Muestra aquellas compras que fueron realizadas con débito durante el 2023.

```
CREATE VIEW DEBITO_2023 AS (
SELECT ID_METODO, DESCRIPCION, PRECIO, FECHA
FROM METODO_PAGO
WHERE DESCRIPCION = "DEBITO" && FECHA >= "2023-01-01"
ORDER BY FECHA
);
```

- OC_COMPLETA_CREDITO_2023: Muestra aquellas Órdenes de Compra que se han completado durante el 2023.

```
CREATE VIEW  OC_COMPLETA_CREDITO_2023 AS (
SELECT ORDEN_COMPRA.ID_ORDEN, ORDEN_COMPRA.ESTADO,
ORDEN_COMPRA.PRECIO, ORDEN_COMPRA.FECHA,
METODO_PAGO.DESCRIPCION
FROM ORDEN_COMPRA, METODO_PAGO
WHERE METODO_PAGO.DESCRIPCION = "CREDITO" AND
ORDEN_COMPRA.FECHA > "2023-01-01" AND ORDEN_COMPRA.ESTADO =
"COMPLETA"
GROUP BY ID_ORDEN
);
```

Las funciones creadas son las siguientes:

- total_cancelados_2023(): ver el monto total de las órdenes de compra canceladas en 2023.

```
delimiter $$
CREATE FUNCTION total_cancelados_2023()
RETURNS DECIMAL(50,2)
DETERMINISTIC
BEGIN
   DECLARE monto_total DECIMAL(50,2);
        SELECT SUM(precio) INTO monto_total
   FROM metodo_pago
   WHERE YEAR(fecha) = 2023
   AND descripcion = 'CANCELADO';
   RETURN monto_total;
END $$
```

- clientes_caba(): pertmite ver la cantidad de clientes en CABA.

```
delimiter $$
CREATE FUNCTION clientes_caba()
RETURNS INT
DETERMINISTIC
BEGIN
  DECLARE total_caba INT;
  SELECT COUNT(*) INTO total_caba
  FROM cliente
  WHERE localidad = 'CABA';
  RETURN total_caba;
END $$
```

- precio_final_aumento2: calcular el precio final de un producto a partir de su id; en caso de querer hacer un aumento % de precio:

```
delimiter $$
CREATE FUNCTION precio_final_aumento2 (id_producto1 INT,
                                                    aumento float)
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
   DECLARE precio_original DECIMAL(10, 2);
   DECLARE precio_final DECIMAL(10, 2);
      SELECT precio INTO precio_original FROM producto WHERE id_producto =
id_producto1;
      SET precio_final = precio_original + (precio_original * (aumento / 100));
      RETURN precio_final;
END $$
```

Los Stored Procedures creados son los siguientes:

- stock_productos: ver la cantidad de productos que tienen stock <= "X":

```
delimiter $$
CREATE PROCEDURE stock_productos (in p_dato_stock int)
begin
        select * from producto
   where cantidad <= p_dato_stock;
end $$

call stock_productos(10)
```

- oc_estados: obtener las Órdenes de Compra según el estado en el que se encuentre, puede ser "COMPLETA" "CANCELADA" "EN PROCESO":

```
delimiter $$
CREATE PROCEDURE oc_estados (IN p_estado VARCHAR(50))
BEGIN
   SELECT * FROM orden_compra WHERE estado = p_estado;
END $$

call oc_estados("COMPLETA") -- puede ser "COMPLETA" "CANCELADA" "EN
PROCESO"
```

- InsertarCliente: agregar un cliente, en un caso ideal el ID_Cliente sería autoincremental:

```
DELIMITER $$
CREATE PROCEDURE InsertarCliente(
        IN id_cliente int,
   IN nombre VARCHAR(20),
   IN apellido VARCHAR(20),
   IN email VARCHAR(100),
   IN calle VARCHAR(20),
   IN numero INT,
   IN localidad VARCHAR(20),
   IN tel INT
)
BEGIN
   INSERT INTO cliente (id_cliente, nombre, apellido, email, calle, numero, localidad,
tel)
   VALUES (id_cliente, nombre, apellido, email, calle, numero, localidad, tel);
END $$
CALL InsertarCliente(16,'Leandro', 'Traficante', 'leatrafi@gmail.com', 'Llalal', '23',
'Martinez', '5555555');
select * from cliente
```

- sp_ordenar_clientes: ordenar la tabla segun el parámetro deseado y en ASC / DSC:

```
DELIMITER //
CREATE PROCEDURE sp_ordenar_clientes(IN campo VARCHAR(20), IN orden
VARCHAR(4))
```

```
BEGIN
   SET @query = CONCAT('SELECT * FROM cliente ORDER BY ', campo, ' ',
orden);
   PREPARE stmt FROM @query;
   EXECUTE stmt;
   DEALLOCATE PREPARE stmt;
END //
DELIMITER ;

CALL sp_ordenar_clientes('calle', 'desc');
```

Link al repositorio GitHub: https://github.com/Leatraficante/SQL-Coder.git