

Documentación Reto

Para este reto se pide realizar una base de datos para una aplicación sobre una **Tienda Virtual Don pepe (Ejercicio C)** este ejercicio nos pide los siguientes requerimientos:

1. Clientes: se requiere información personal de los clientes como ID, cédula, nombre, dirección, teléfono, email y password para poder registrarse en la aplicación y realizar compras.
2. Categorías de productos: los productos que oferta el supermercado deben estar divididos en diversas categorías con información como el nombre de la categoría, condiciones de almacenamiento (frío, congelado, seco) y observaciones.
3. Productos: se necesita información detallada de cada producto como el nombre, marca, origen, dimensiones (volumen y peso), una fotografía, la categoría y unidades disponibles.
4. Pedido: para generar un pedido se requiere el código del pedido, fecha del pedido, cliente, dirección de entrega, productos pedidos, importe total del pedido y datos de pago (número de tarjeta y fecha de caducidad).
5. Zonas de entrega: los clientes deben pertenecer a una zona (Código Postal) donde existan domiciliarios. Un domiciliario se identifica mediante un nombre, número de matrícula de la furgoneta y zona donde reparte.
6. Stock de productos: se debe controlar el stock de cada producto y asegurarse de que haya unidades suficientes para satisfacer las demandas de cada pedido.

Después de analizar el ejercicio decidí empezar a hacer el modelo entidad relación en el programa día. Una vez tenía las entidades y los atributos listos, empecé a analizar qué relación había entre ellas. Después de tener las relaciones entre entidades me puse a mirar que cardinalidad podía implementar en cada una de ellas y quedo de la siguiente manera:

Podemos identificar las siguientes relaciones entre las entidades:

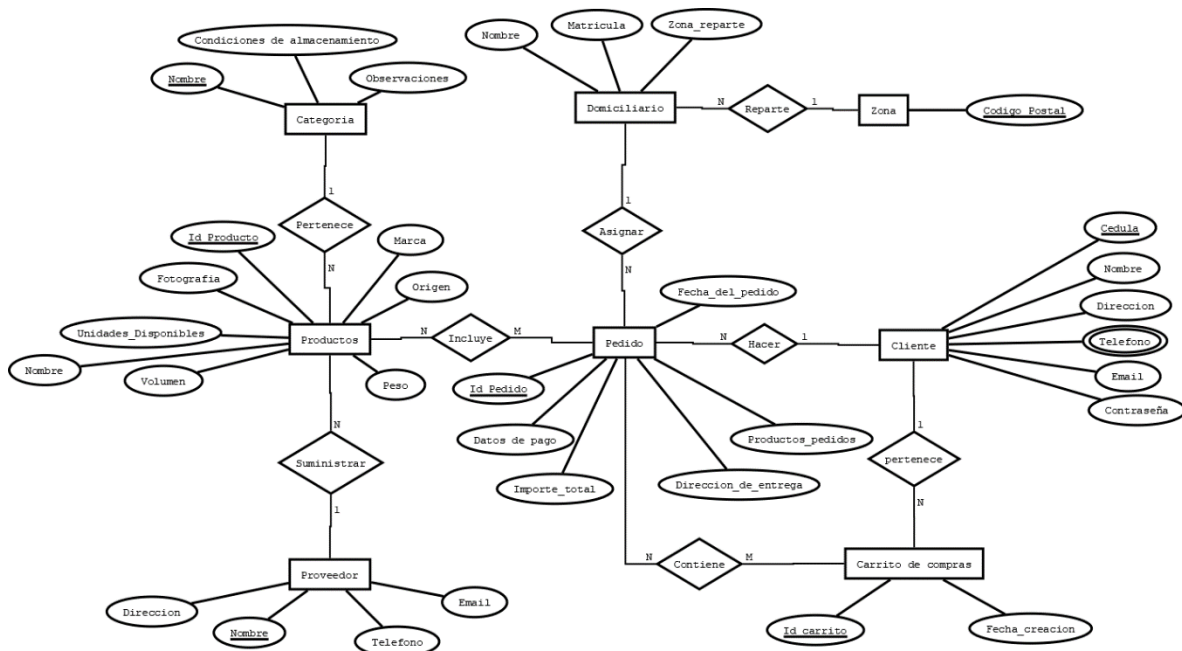
- Un cliente puede realizar muchos pedidos, pero un pedido solo puede ser realizado por un cliente (relación uno a muchos entre Cliente y Pedido).
- Un pedido puede incluir varios productos, y un producto puede estar en varios pedidos (relación muchos a muchos entre Pedido y Producto).
- Un domiciliario puede realizar muchos pedidos, pero un pedido solo puede ser entregado por un domiciliario (relación uno a muchos entre Pedido y Domiciliario).
- Un domiciliario puede trabajar en una zona, y una zona puede tener muchos domiciliarios (relación uno a muchos entre Zona y Domiciliario).

- Un producto pertenece a una categoría, y una categoría puede tener muchos productos (relación uno a muchos entre Categoría y Producto).
- Proveedor: Una relación uno a muchos, ya que un proveedor puede suministrar muchos productos, pero un producto solo puede ser suministrado por un proveedor.
- Un cliente puede tener muchos carritos de compras, pero un carrito de compras solo puede pertenecer a un cliente (relación uno a muchos entre Cliente y Carrito de compras).

NOTA: Sí, hay una relación entre Cliente y Domiciliario a través de la entidad Pedido. En el enunciado se menciona que un pedido debe ser entregado por un domiciliario, por lo que podemos inferir que existe una relación entre Pedido y Domiciliario. A su vez, un pedido es realizado por un cliente, por lo que hay una relación entre Cliente y Pedido.

Entonces, la relación entre Cliente y Domiciliario se da indirectamente a través de la entidad Pedido.

Después de aclarar esto, adjunto una imagen del modelo entidad relación:



Luego de tener listo el modelo entidad-relación, procedí a transformarlo a un modelo relacional y para esto empecé creando las tablas con las entidades principales y sus atributos.

Y después hice la relación correspondiente a cada tabla.

Normalización

Primera Forma Normal (1FN)

Todos los atributos son atómicos, los campos deben ser identificados por la clave. En resumen, se eliminan los valores repetidos dentro de una base de datos.

Segunda Forma Normal (2FN)

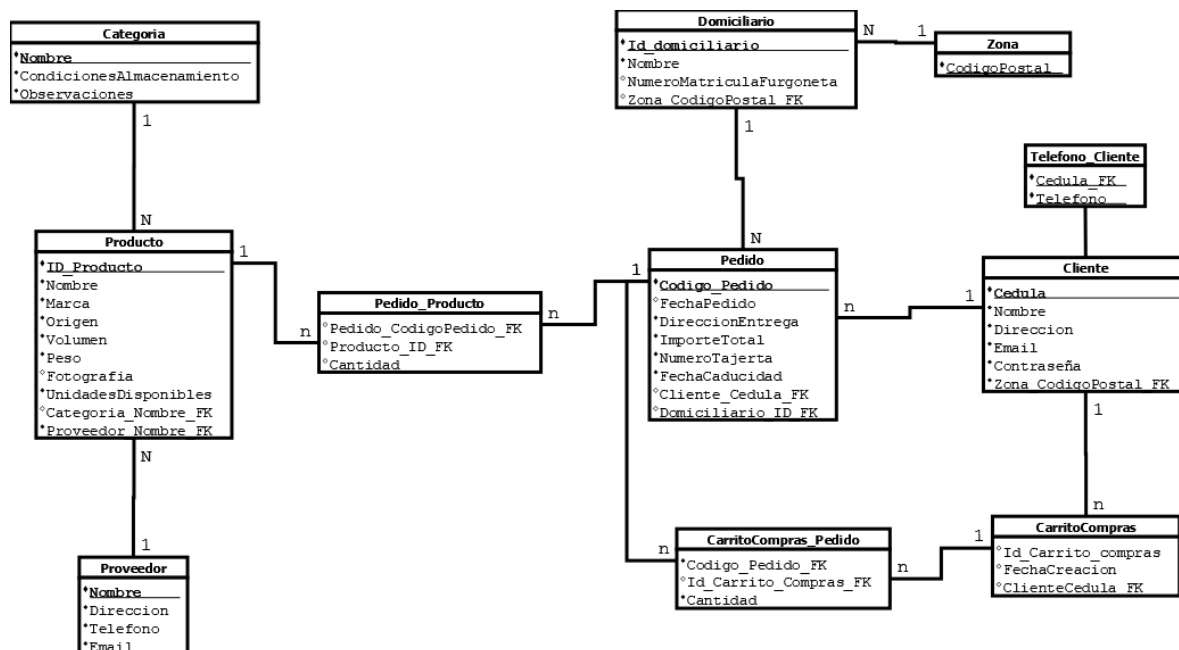
Se eliminan las dependencias parciales. Todos los atributos que no son clave principal deben depender únicamente de la clave principal.

Tercera Forma Normal (3FN)

tiene como objetivo reducir la redundancia y la dependencia funcional no necesaria en una tabla o relación.

En la 3FN, se asegura que cada atributo de una tabla está dependiente funcionalmente únicamente de la clave principal de la tabla y no de otros atributos. Esto significa que cada atributo de la tabla debe ser dependiente funcionalmente de la clave principal de la tabla y no de otras claves, ni de combinaciones de ellas.

Con la Normalización aplicada quedaría de esta manera:



Después de esto empecé a realizar el script en MySQL para poder generar mi base de datos. Para esto tome como referencia mi modelo relacional.

```
1 • CREATE DATABASE TiendaVirtual;
2 • USE TiendaVirtual;
3
4 • CREATE TABLE Zona (
5     CodigoPostal INT PRIMARY KEY
6 );
7
8 • CREATE TABLE Cliente (
9     Cedula INT PRIMARY KEY,
10    Nombre VARCHAR(50),
11    Direccion VARCHAR(100),
12    Email VARCHAR(100),
13    Contraseña VARCHAR(60),
14    Zona_CodigoPostal INT,
15    FOREIGN KEY (Zona_CodigoPostal) REFERENCES Zona(CodigoPostal)
16 );
17
18 • CREATE TABLE Telefono_Cliente (
19     Cedula_FK INT NOT NULL,
20     Telefono_FK VARCHAR(15) NOT NULL,
21     PRIMARY KEY (Cedula_FK, Telefono_FK),
22     FOREIGN KEY (Cedula_FK) REFERENCES Cliente (Cedula)
23 );
24
25 • CREATE TABLE Categoria (
26     Nombre VARCHAR(50) PRIMARY KEY,
27     CondicionesAlmacenamiento VARCHAR(100),
28     Observaciones VARCHAR(200)
29 );
```

En la imagen anterior creo las tablas siguientes: Zona, Cliente, Telefono_Cliente y Categoría.

En la imagen siguiente creo las tablas siguientes: proveedor, Producto (el peso está en Kg) y Domiciliario

```
33 • CREATE TABLE Proveedor (  
34     Nombre VARCHAR(50) PRIMARY KEY,  
35     Direccion VARCHAR(100),  
36     Telefono VARCHAR(15),  
37     Email VARCHAR(50)  
38 );  
39  
40 • CREATE TABLE Producto (  
41     ID INT PRIMARY KEY,  
42     Nombre VARCHAR(50),  
43     Marca VARCHAR(50),  
44     Origen VARCHAR(50),  
45     Volumen DECIMAL(10,2), -- Use DECIMAL para medidas físicas  
46     Peso DECIMAL(10,2), -- Use DECIMAL para medidas físicas  
47     Fotografia VARCHAR(100),  
48     Categoria_Nombre VARCHAR(50),  
49     UnidadesDisponibles INT,  
50     Proveedor_Nombre VARCHAR(50),  
51     FOREIGN KEY (Categoria_Nombre) REFERENCES Categoria(Nombre),  
52     FOREIGN KEY (Proveedor_Nombre) REFERENCES Proveedor(Nombre)  
53 );  
54  
55 • CREATE TABLE Domiciliario (  
56     ID INT PRIMARY KEY,  
57     Nombre VARCHAR(50),  
58     NumeroMatriculaFurgoneta VARCHAR(30),  
59     Zona_CodigoPostal INT,  
60     FOREIGN KEY (Zona_CodigoPostal) REFERENCES Zona(CodigoPostal)  
61 );
```

En la siguiente imagen creo las siguientes tablas: Pedido, Pedido_Producto, carrito_compras y CarritoCompras_Pedido

```
62 • CREATE TABLE Pedido (  
63     CodigoPedido INT PRIMARY KEY,  
64     FechaPedido DATETIME,  
65     Cliente_Cedula INT,  
66     Domiciliario_ID INT,  
67     DireccionEntrega VARCHAR(100),  
68     ImporteTotal DECIMAL(10,2), -- Use DECIMAL para medidas monetarias  
69     NumeroTarjeta VARCHAR(16),  
70     FechaCaducidad DATE,  
71     FOREIGN KEY (Cliente_Cedula) REFERENCES Cliente(Cedula),  
72     FOREIGN KEY (Domiciliario_ID) REFERENCES Domiciliario (ID)  
73 );  
74  
75 • CREATE TABLE Pedido_Producto (  
76     Pedido_CodigoPedido INT,  
77     Producto_ID INT,  
78     Cantidad INT NOT NULL CHECK (Cantidad >= 1), -- La cantidad mínima es 1  
79     PRIMARY KEY (Pedido_CodigoPedido, Producto_ID),  
80     FOREIGN KEY (Pedido_CodigoPedido) REFERENCES Pedido(CodigoPedido),  
81     FOREIGN KEY (Producto_ID) REFERENCES Producto(ID)  
82 );  
83  
84 • CREATE TABLE CarritoCompras (  
85     ID INT PRIMARY KEY,  
86     FechaCreacion DATETIME,  
87     Cliente_Cedula INT,  
88     FOREIGN KEY (Cliente_Cedula) REFERENCES Cliente(Cedula)  
89 );  
90  
90  
91 • CREATE TABLE CarritoCompras_Pedido (  
92     CarritoCompras_ID INT,  
93     CodigoPedido_FK INT,  
94     Cantidad INT NOT NULL CHECK (Cantidad >= 1), -- La cantidad mínima es 1  
95     PRIMARY KEY (CarritoCompras_ID, CodigoPedido_FK),  
96     FOREIGN KEY (CarritoCompras_ID) REFERENCES CarritoCompras(ID),  
97     FOREIGN KEY (CodigoPedido_FK) REFERENCES Pedido(CodigoPedido)  
98 );  
99
```


Para el siguiente paso lo que hice fue poblar mi base de datos solo con dos registros para poder probar las consultas, vistas, procedimientos almacenados y los triggers.

Llene la tabla con los siguientes registros:

```
102 • INSERT INTO Zona (CodigoPostal) VALUES (1010);
103 • INSERT INTO Zona (CodigoPostal) VALUES (1020);
104 • INSERT INTO Zona (CodigoPostal) VALUES (1030);
105
106 • INSERT INTO Cliente (Cedula, Nombre, Direccion, Email, Contraseña, Zona_CodigoPostal)
107 VALUES (1012345678, 'Juan Perez', 'Calle 1 # 23-45', 'juanperez@gmail.com', 'contraseña123', 1010);
108 • INSERT INTO Cliente (Cedula, Nombre, Direccion, Email, Contraseña, Zona_CodigoPostal)
109 VALUES (1023456789, 'Maria Rodriguez', 'Carrera 2 # 34-56', 'mariarodriguez@gmail.com', 'contraseña456', 1020);
110 • INSERT INTO Cliente (Cedula, Nombre, Direccion, Email, Contraseña, Zona_CodigoPostal)
111 VALUES (1023456723, 'Mariaaaaa', 'Carrera 2 # 34-56', 'mariarodriguez@gmail.com', 'contraseña456', 1050);
112
113 • INSERT INTO Telefono_Cliente (Cedula_FK, Telefono_FK) VALUES (1012345678, '3123456789');
114 • INSERT INTO Telefono_Cliente (Cedula_FK, Telefono_FK) VALUES (1023456789, '3001234567');
115
116
117 • INSERT INTO Categoria (Nombre, CondicionesAlmacenamiento, Observaciones)
118 VALUES ('Electrónicos', 'Almacenar en lugares secos y frescos', 'No apilar productos');
119 • INSERT INTO Categoria (Nombre, CondicionesAlmacenamiento, Observaciones)
120 VALUES ('Alimentos', 'Almacenar en lugares frescos y limpios', 'Revisar fecha de caducidad antes de enviar al cliente');
121
122
123 • INSERT INTO Proveedor (Nombre, Direccion, Telefono, Email)
124 VALUES ('Proveedor 1', 'Carrera 4 # 56-78', '3004567890', 'proveedor1@gmail.com');
125 • INSERT INTO Proveedor (Nombre, Direccion, Telefono, Email)
126 VALUES ('Proveedor 2', 'Calle 5 # 67-89', '3101234567', 'proveedor2@gmail.com');
127
```

1. El primer registro crea la tabla "Zona" y agrega tres filas con los códigos postales 1010, 1020 y 1030.
2. El segundo registro crea la tabla "Cliente" y agrega tres filas, cada una representando un cliente diferente. Cada fila tiene una Cedula (número de identificación personal), Nombre, Dirección, Email, Contraseña y el Código Postal de la Zona donde vive el cliente. Los códigos postales hacen referencia a las filas previamente insertadas en la tabla "Zona".
3. El tercer registro crea la tabla "Telefono_Cliente" y agregan dos filas, cada una representando un cliente diferente y su número de teléfono. Cada fila tiene la Cedula del cliente y el número de teléfono.
4. El cuarto registro crea la tabla "Categoria" y agregan dos filas, cada una representando una categoría diferente de productos que se venden. Cada fila tiene el nombre de la categoría, las condiciones de almacenamiento recomendadas y observaciones adicionales.
5. Los últimos dos registros crean la tabla "Proveedor" y agregan dos filas, cada una representando un proveedor diferente. Cada fila tiene el nombre del proveedor, la dirección, el número de teléfono y el correo electrónico.

Se hace el mismo procedimiento con estos registros:

```
129 • INSERT INTO Producto (ID, Nombre, Marca, Origen, Volumen, Peso, Fotografia, Categoria_Nombre, UnidadesDisponibles, Proveedor_Nombre)
130 VALUES (1, 'Celular', 'Samsung', 'Corea del Sur', 200, 0.3, 'celular.jpg', 'Electrónicos', 100, 'Proveedor 1');
131 • INSERT INTO Producto (ID, Nombre, Marca, Origen, Volumen, Peso, Fotografia, Categoria_Nombre, UnidadesDisponibles, Proveedor_Nombre)
132 VALUES (2, 'Arroz', 'Diana', 'Colombia', 1000, 1.5, 'arroz.jpg', 'Alimentos', 500, 'Proveedor 2');
133
134 • INSERT INTO Domiciliario (ID, Nombre, NumeroMatriculaFurgoneta, Zona_CodigoPostal)
135 VALUES (1, 'Pedro Gomez', 'ABC123', 1010);
136 • INSERT INTO Domiciliario (ID, Nombre, NumeroMatriculaFurgoneta, Zona_CodigoPostal)
137 VALUES (2, 'Maria Hernandez', 'XYZ123', 1020);
138
139 • INSERT INTO Pedido (CodigoPedido, FechaPedido, Cliente_Cedula, Domiciliario_ID, DireccionEntrega, ImporteTotal, NumeroTarjeta, FechaCaducida
140 VALUES (1, '2023-02-16 10:00:00', 1012345678, 1, 'Calle 1 # 23-45', 50000.00, '1234567890123456', '2025-01-31');
141 • INSERT INTO Pedido (CodigoPedido, FechaPedido, Cliente_Cedula, Domiciliario_ID, DireccionEntrega, ImporteTotal, NumeroTarjeta, FechaCaducida
142 VALUES (2, '2023-02-16 14:00:00', 1023456789, 2, 'Carrera 2 # 34-56', 35000.00, '9876543210987654', '2024-11-30');
143
144 • INSERT INTO Pedido_Producto (Pedido_CodigoPedido, Producto_ID, Cantidad) VALUES (1, 1, 2);
145 • INSERT INTO Pedido_Producto (Pedido_CodigoPedido, Producto_ID, Cantidad) VALUES (2, 2, 3);
146
147 • INSERT INTO CarritoCompras (ID, FechaCreacion, Cliente_Cedula) VALUES (1, '2023-02-16 16:00:00', 1012345678);
148 • INSERT INTO CarritoCompras (ID, FechaCreacion, Cliente_Cedula) VALUES (2, '2023-02-16 18:00:00', 1023456789);
149
150 • INSERT INTO CarritoCompras_Pedido (CarritoCompras_ID, CodigoPedido_FK, Cantidad) VALUES (1, 1, 1);
151 • INSERT INTO CarritoCompras_Pedido (CarritoCompras_ID, CodigoPedido_FK, Cantidad) VALUES (2, 2, 2);
```

CONSULTAS:

Teniendo el anterior paso listo y con las tablas pobladas con estos registros mi siguiente paso fue hacer las consultas de prueba para ver si todo funcionaba correctamente y este fue el resultado:

1. Obtener el nombre y la dirección de todos los clientes que se encuentran en la zona con código postal 1010:

```
155 -- -----
156 -- ----#CONSULTAS:-----
157 -- -----
158
159 -- -----
160 -- ----#Obtener el nombre y la dirección de todos los clientes que se encuentran en la zona con código postal 1010:-----
161 -- -----
162 • SELECT Nombre, Direccion
163 FROM Cliente
164 WHERE Zona_CodigoPostal = 1010;
165
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
Nombre	Direccion			
Juan Perez	Calle 1 # 23-45			

Esta consulta se utiliza para obtener una lista de los nombres y direcciones de todos los clientes que viven en la zona con código postal 1010.

La consulta utiliza la cláusula WHERE para especificar la condición de búsqueda. En este caso, la condición es "Zona_CodigoPostal = 1010", lo que significa que solo se seleccionarán las filas de la tabla "Cliente" donde el valor de la columna "Zona_CodigoPostal" sea igual a 1010.

La consulta utiliza la cláusula SELECT para especificar las columnas que se deben mostrar en el resultado. En este caso, la consulta solicita las columnas "Nombre" y "Direccion" de la tabla "Cliente".

2. Obtener el nombre, la dirección y el correo electrónico de los clientes que han realizado pedidos:

```
-- -----#Obtener el nombre, la dirección y el correo electrónico de los clientes que han realizado pedidos:-----  
-----  
SELECT DISTINCT c.Nombre, c.Direccion, c.Email  
FROM Cliente c  
INNER JOIN Pedido p ON c.Cedula = p.Cliente_Cedula;
```

3. Obtener el nombre de los proveedores que suministran productos de la categoría "Alimentos":

```
174 -- -----#Obtener el nombre de los proveedores que suministran productos de la categoría "Alimentos":-----  
175 -----  
176 • SELECT DISTINCT Proveedor_Nombre  
177 FROM Producto  
178 WHERE Categoria_Nombre = 'Alimentos';  
179 -----  
180 -----  
181 -- -----#Obtener el nombre y la dirección de los proveedores que suministran productos con un volumen mayor a 500:-----  
-----  
Result Grid | Filter Rows: | Export: | Wrap Cell Content: |  
Proveedor_Nombre  
Proveedor 2
```

En la imagen anterior

4. Obtener el nombre y la dirección de los proveedores que suministran productos con un volumen mayor a 500:

```
-- -----#Obtener el nombre y la dirección de los proveedores que suministran productos con un volumen mayor a 500:-----  
-----  
SELECT Nombre, Direccion  
FROM Proveedor  
WHERE Nombre IN (SELECT DISTINCT Proveedor_Nombre FROM Producto WHERE Volumen > 500);
```

5. Obtener el nombre y la marca de todos los productos con un peso menor a 1 kg:

```
187 -----  
188 -- -----#Obtener el nombre y la marca de todos los productos con un peso menor a 1:-----  
189 -----  
190 • SELECT Nombre, Marca  
191 FROM Producto  
192 WHERE Peso < 1;  
193 -----  
Result Grid | Filter Rows: | Export: | Wrap Cell Content: |  
Nombre Marca  
Celular Samsung
```

La consulta utiliza la cláusula WHERE para especificar la condición de búsqueda. En este caso, la condición es "Peso < 1", lo que significa que solo se seleccionarán las filas de la tabla "Producto" donde el valor de la columna "Peso" sea menor a 1 kg.

La consulta utiliza la cláusula SELECT para especificar las columnas que se deben mostrar en el resultado. En este caso, la consulta solicita las columnas "Nombre" y "Marca" de la tabla "Producto".

El resultado de la consulta es una lista de nombres y marcas de los productos que cumplen con la condición especificada en la cláusula WHERE. Por lo tanto, esta consulta es útil para obtener información específica de los productos que pesan menos de 1 kg.

6. Obtener el número de teléfono de los clientes que han realizado pedidos:

```
194  -----
195  -- ----#Obtener el número de teléfono de los clientes que han realizado pedidos:-----
196  -----
197  • SELECT DISTINCT Telefono_FK
198  FROM Telefono_Cliente
199  WHERE Cedula_FK IN (SELECT DISTINCT Cliente_Cedula FROM Pedido);
200
201  -----
202  -- ----#Obtener el nombre y el número de matrícula de todas las furgonetas de domiciliarios que traba
```

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

Telefono_FK
3123456789
3001234567

7. Obtener el nombre y el número de matrícula de todas las furgonetas de domiciliarios que trabajan en la zona con código postal 1020:

```
201  -----
202  -- ----#Obtener el nombre y el número de matrícula de todas las furgonetas de domiciliarios que trabajan en la zona con código postal 1020:-
203  -----
204  • SELECT Nombre, NumeroMatriculaFurgoneta
205  FROM Domiciliario
206  WHERE Zona_CodigoPostal = 1020;
207
208  -----
```

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

Nombre	NumeroMatriculaFurgoneta
Maria Hernandez	XYZ123

Result Grid

8. Obtener el número total de unidades disponibles de todos los productos:

```
209  -- ----#Obtener el número total de unidades disponibles de todos los productos:-----
210  -- ----
211
212  • SELECT SUM(UnidadesDisponibles) as TotalUnidades
213  FROM Producto;
214
215  -- ----
216  -- ----#Obtener el importe total de todos los pedidos realizados:-----
217  -- ----
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
TotalUnidades			
823			

9. Obtener el importe total de todos los pedidos realizados:

```
215  -- ----
216  -- ----#Obtener el importe total de todos los pedidos realizados:-----
217  -- ----
218  • SELECT SUM(ImporteTotal) as TotalImporte
219  FROM Pedido;
220
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
TotalImporte			
85000.00			

10. Consulta para contar el número total de productos diferentes que han sido pedidos a través de la tabla "Pedido_Producto":

```
222  -- ----
223  -- ----# Consulta para contar el número total de productos diferentes que han sido pedidos a través de la tabla "Pedido_Producto":-----
224  -- ----
225  • SELECT COUNT(DISTINCT Producto_ID) as TotalProductos
226  FROM Pedido_Producto;
227
228
229  -- ----
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
TotalProductos			
2			

Esta consulta se utiliza para contar el número total de productos diferentes que han sido pedidos a través de la tabla "Pedido_Producto".

La función COUNT es utilizada para contar el número de filas en la tabla que cumplen una determinada condición. En este caso, la condición es "DISTINCT Producto_ID", que significa que solo se deben contar los productos únicos y no todas las veces que se han pedido.

El alias "TotalProductos" se utiliza para darle un nombre más amigable al resultado que devuelve la consulta. Esto significa que en lugar de obtener una tabla con una columna llamada "COUNT(DISTINCT Producto_ID)", obtendrás una tabla con una columna llamada "TotalProductos".

En resumen, esta consulta te devolverá un solo número que indica la cantidad total de productos diferentes que han sido pedidos a través de la tabla "Pedido_Producto".

VISTAS:

1. Vista de clientes con sus respectivas zonas y teléfonos:

```
233  -- -----
234  -- ----#Vista de clientes con sus respectivas zonas y teléfonos:-----
235  -- -----
236  • SELECT Cliente.Cedula, Cliente.Nombre, Zona.CodigoPostal, Telefono_Cliente.Telefono_FK
237  FROM Cliente
238  INNER JOIN Zona ON Cliente.Zona_CodigoPostal = Zona.CodigoPostal
239  INNER JOIN Telefono_Cliente ON Cliente.Cedula = Telefono_Cliente.Cedula_FK;
240
```

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	Cedula	Nombre	CodigoPostal	Telefono_FK
▶	1012345678	Juan Perez	1010	3123456789
	1023456789	Maria Rodriguez	1020	3001234567

Al utilizar esta vista, se pueden obtener rápidamente detalles importantes de los clientes, incluyendo su identificación personal, nombre, código postal y número de teléfono, todo en una sola consulta. Esta vista de SQL es una combinación de tres tablas diferentes: "Cliente", "Zona" y "Telefono_Cliente". La vista se crea utilizando las cláusulas INNER JOIN para unir las filas de las tres tablas en una sola vista.

La vista resultante contiene cuatro columnas: "Cedula", "Nombre", "CodigoPostal" y "Telefono_FK".

La columna "Cedula" proviene de la tabla "Cliente" y contiene el número de identificación personal de cada cliente.

La columna "Nombre" también proviene de la tabla "Cliente" y contiene el nombre de cada cliente.



La columna "CodigoPostal" proviene de la tabla "Zona" y contiene el código postal de la zona donde vive cada cliente.


La columna "Telefono_FK" proviene de la tabla "Telefono_Cliente" y contiene el número de teléfono de cada cliente.


2. Vista de productos con su respectiva categoría y proveedor:

```
241  -----
242  -- ----#Vista de productos con su respectiva categoría y proveedor:-----
243  -----
244  • SELECT Producto.ID, Producto.Nombre, Producto.Marca, Producto.Categoria_Nombre, Producto.Proveedor_Nombre
245  FROM Producto
246  INNER JOIN Categoria ON Producto.Categoria_Nombre = Categoria.Nombre
247  INNER JOIN Proveedor ON Producto.Proveedor_Nombre = Proveedor.Nombre;
248
249  -----
```

Result Grid

  Filter Rows:

Export: 

Wrap Cell Content: 

ID	Nombre	Marca	Categoria_Nombre	Proveedor_Nombre
1	Celular	Samsung	Electrónicos	Proveedor 1
2	Arroz	Diana	Alimentos	Proveedor 2
3	Leche	Alquería	Alimentos	Proveedor 2

De esta vista se pueden obtener rápidamente detalles importantes de los productos, incluyendo su identificador único, nombre, marca, categoría y proveedor, todo en una sola consulta. Esta consulta de SQL es una combinación de tres tablas diferentes: "Producto", "Categoria" y "Proveedor". La consulta utiliza las cláusulas INNER JOIN para unir las filas de las tres tablas en una sola vista.

La vista resultante contiene cinco columnas: "ID", "Nombre", "Marca", "Categoria_Nombre" y "Proveedor_Nombre".

La columna "ID" proviene de la tabla "Producto" y contiene el identificador único de cada producto.

La columna "Nombre" también proviene de la tabla "Producto" y contiene el nombre de cada producto.

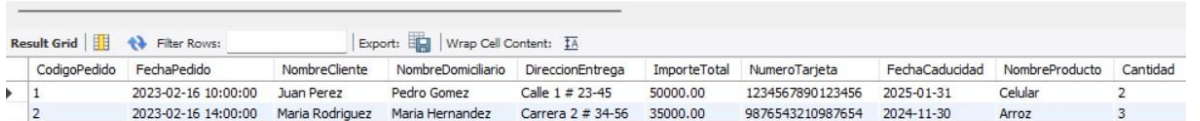
La columna "Marca" proviene de la tabla "Producto" y contiene la marca del producto.

La columna "Categoria_Nombre" proviene de la tabla "Producto" y se une con la columna "Nombre" de la tabla "Categoria" mediante la cláusula INNER JOIN. Esta columna contiene el nombre de la categoría del producto.

La columna "Proveedor_Nombre" proviene de la tabla "Producto" y se une con la columna "Nombre" de la tabla "Proveedor" mediante la cláusula INNER JOIN. Esta columna contiene el nombre del proveedor del producto.

3. Vista de pedidos con información del cliente, domiciliario y productos:

```
250  -- ---#Vista de pedidos con información del cliente, domiciliario y productos:-----
251  -----
252  • SELECT Pedido.CodigoPedido, Pedido.FechaPedido, Cliente.Nombre AS NombreCliente, Domiciliario.Nombre AS NombreDomiciliario, Pedido
253  FROM Pedido
254  INNER JOIN Cliente ON Pedido.Cliente_Cedula = Cliente.Cedula
255  INNER JOIN Domiciliario ON Pedido.Domiciliario_ID = Domiciliario.ID
256  INNER JOIN Pedido_Producto ON Pedido.CodigoPedido = Pedido_Producto.Pedido_CodigoPedido
257  INNER JOIN Producto ON Pedido_Producto.Producto_ID = Producto.ID;
258
```



	CodigoPedido	FechaPedido	NombreCliente	NombreDomiciliario	DireccionEntrega	ImporteTotal	NumeroTarjeta	FechaCaducidad	NombreProducto	Cantidad
▶	1	2023-02-16 10:00:00	Juan Perez	Pedro Gomez	Calle 1 # 23-45	50000.00	1234567890123456	2025-01-31	Celular	2
	2	2023-02-16 14:00:00	Maria Rodriguez	Maria Hernandez	Carrera 2 # 34-56	35000.00	9876543210987654	2024-11-30	Arroz	3

Esta consulta de SQL es una combinación de cinco tablas diferentes: "Pedido", "Cliente", "Domiciliario", "Pedido_Producto" y "Producto". La consulta utiliza las cláusulas INNER JOIN para unir las filas de las cinco tablas en una sola vista.

La vista resultante contiene diez columnas: "CodigoPedido", "FechaPedido", "NombreCliente", "NombreDomiciliario", "DireccionEntrega", "ImporteTotal", "NumeroTarjeta", "FechaCaducidad", "NombreProducto" y "Cantidad".

La columna "CodigoPedido" proviene de la tabla "Pedido" y contiene el código único de cada pedido.

La columna "FechaPedido" también proviene de la tabla "Pedido" y contiene la fecha en que se realizó el pedido.

La columna "NombreCliente" proviene de la tabla "Cliente" y se une con la columna "Cliente_Cedula" de la tabla "Pedido" mediante la cláusula INNER JOIN. Esta columna contiene el nombre del cliente que realizó el pedido.

La columna "NombreDomiciliario" proviene de la tabla "Domiciliario" y se une con la columna "Domiciliario_ID" de la tabla "Pedido" mediante la cláusula INNER JOIN. Esta columna contiene el nombre del domiciliario que entregó el pedido.

La columna "DireccionEntrega" proviene de la tabla "Pedido" y contiene la dirección donde se entregó el pedido.

La columna "ImporteTotal" proviene de la tabla "Pedido" y contiene el importe total del pedido.

La columna "NumeroTarjeta" proviene de la tabla "Pedido" y contiene el número de tarjeta utilizada para pagar el pedido.

La columna "FechaCaducidad" proviene de la tabla "Pedido" y contiene la fecha de caducidad de la tarjeta utilizada para pagar el pedido.

La columna "NombreProducto" proviene de la tabla "Producto" y se une con la columna "Producto_ID" de la tabla "Pedido_Producto" mediante la cláusula INNER JOIN. Esta columna contiene el nombre del producto que se ha pedido.

La columna "Cantidad" proviene de la tabla "Pedido_Producto" y contiene la cantidad del producto que se ha pedido.

La vista es útil porque combina información de varias tablas relacionadas en una sola vista. Al utilizar esta vista, se pueden obtener rápidamente detalles importantes de los pedidos, incluyendo el código del pedido, la fecha del pedido, los nombres del cliente y del domiciliario, la dirección de entrega, el importe total, los detalles de pago y la información de los productos pedidos, todo en una sola consulta.

4. Vista de proveedores con información de sus productos:

```
262 • SELECT Proveedor.Nombre, Proveedor.Direccion, Proveedor.Telefono, Proveedor.Email, Producto.Nombre AS NombreProducto, Producto.Categoria_Nom
263 FROM Proveedor
264 INNER JOIN Producto ON Proveedor.Nombre = Producto.Proveedor_Nombre;
265
266 -- -----
267 -- ----#Vista de carritos de compras con información de los pedidos y el cliente:-----
268 -- -----
```

Nombre	Direccion	Telefono	Email	NombreProducto	Categoria_Nombre
Proveedor 1	Carrera 4 # 56-78	3004567890	proveedor1@gmail.com	Celular	Electrónicos
Proveedor 2	Calle 5 # 67-89	3101234567	proveedor2@gmail.com	Arroz	Alimentos
Proveedor 2	Calle 5 # 67-89	3101234567	proveedor2@gmail.com	Leche	Alimentos

Al utilizar esta vista, se pueden obtener rápidamente detalles importantes de los proveedores, incluyendo su nombre, dirección, número de teléfono, correo electrónico y los detalles de los productos que proporcionan, todo en una sola consulta. Esta consulta de SQL es una combinación de dos tablas diferentes: "Proveedor" y "Producto". La consulta utiliza la cláusula INNER JOIN para unir las filas de las dos tablas en una sola vista.

La vista resultante contiene seis columnas: "Nombre", "Direccion", "Telefono", "Email", "NombreProducto" y "Categoria_Nombre".

La columna "Nombre" proviene de la tabla "Proveedor" y contiene el nombre del proveedor. La columna "Direccion" también proviene de la tabla "Proveedor" y contiene la dirección del proveedor.

La columna "Telefono" proviene de la tabla "Proveedor" y contiene el número de teléfono del proveedor.

La columna "Email" proviene de la tabla "Proveedor" y contiene el correo electrónico del proveedor.

La columna "NombreProducto" proviene de la tabla "Producto" y contiene el nombre de los productos proporcionados por cada proveedor. Esta columna se une con la columna "Proveedor_Nombre" de la tabla "Proveedor" mediante la cláusula INNER JOIN.

La columna "Categoria_Nombre" proviene de la tabla "Producto" y contiene el nombre de la categoría de cada producto proporcionado por cada proveedor.

5. Vista de carritos de compras con información de los pedidos y el cliente:

```
267 -- ----#Vista de carritos de compras con información de los pedidos y el cliente:-----
268 -----
269 • SELECT CarritoCompras.ID, CarritoCompras.FechaCreacion, Cliente.Nombre AS NombreCliente, Pedido.CodigoPedido, Pedido.FechaPedido, Pedido.ImporteTotal
270 FROM CarritoCompras
271 INNER JOIN Cliente ON CarritoCompras.Cliente_Cedula = Cliente.Cedula
272 INNER JOIN CarritoCompras_Pedido ON CarritoCompras.ID = CarritoCompras_Pedido.CarritoCompras_ID
273 INNER JOIN Pedido ON CarritoCompras_Pedido.CodigoPedido_FK = Pedido.CodigoPedido;
274
```

ID	FechaCreacion	NombreCliente	CodigoPedido	FechaPedido	ImporteTotal
1	2023-02-16 16:00:00	Juan Perez	1	2023-02-16 10:00:00	50000.00
2	2023-02-16 18:00:00	Maria Rodriguez	2	2023-02-16 14:00:00	35000.00

Al utilizar esta vista, se pueden obtener rápidamente detalles importantes de los carritos de compras, incluyendo su identificador único, la fecha en que se creó, el nombre del cliente que lo creó y los detalles de los pedidos relacionados con el carrito de compras, todo en una sola consulta.

Esta consulta de SQL es una combinación de cuatro tablas diferentes: "CarritoCompras", "Cliente", "CarritoCompras_Pedido" y "Pedido". La consulta utiliza las cláusulas INNER JOIN para unir las filas de las cuatro tablas en una sola vista.

La vista resultante contiene seis columnas: "ID", "FechaCreacion", "NombreCliente", "CodigoPedido", "FechaPedido" e "ImporteTotal".

La columna "ID" proviene de la tabla "CarritoCompras" y contiene el identificador único de cada carrito de compras.

La columna "FechaCreacion" también proviene de la tabla "CarritoCompras" y contiene la fecha en que se creó el carrito de compras.

La columna "NombreCliente" proviene de la tabla "Cliente" y se une con la columna "Cliente_Cedula" de la tabla "CarritoCompras" mediante la cláusula INNER JOIN. Esta columna contiene el nombre del cliente que creó el carrito de compras.

La columna "CodigoPedido" proviene de la tabla "Pedido" y contiene el código único de cada pedido relacionado con el carrito de compras.

La columna "FechaPedido" también proviene de la tabla "Pedido" y contiene la fecha en que se realizó el pedido relacionado con el carrito de compras.

La columna "ImporteTotal" también proviene de la tabla "Pedido" y contiene el importe total del pedido relacionado con el carrito de compras.

6. Vista de categorías de productos y sus respectivas condiciones de almacenamiento:

```
275  -----
276  -- ---#Vista de categorías de productos y sus respectivas condiciones de almacenamiento:-----
277  -----
278  • SELECT Categoria.Nombre, Categoria.CondicionesAlmacenamiento
279  FROM Categoria;
280
281  -----
282  -- ---#PROCEDIMIENTOS ALMACENADOS:-----
283  -----
```

Result Grid	Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:
Nombre	CondicionAlmacenamiento			
Alimentos	Almacenar en lugares frescos y limpios			
Electrónicos	Almacenar en lugares secos y frescos			
HUELE	HUELE			

Esta consulta de SQL selecciona dos columnas de la tabla "Categoria" y no utiliza ninguna cláusula JOIN, por lo que simplemente devuelve todos los registros de la tabla "Categoria" y muestra solo dos columnas: "Nombre" y "CondicionAlmacenamiento"

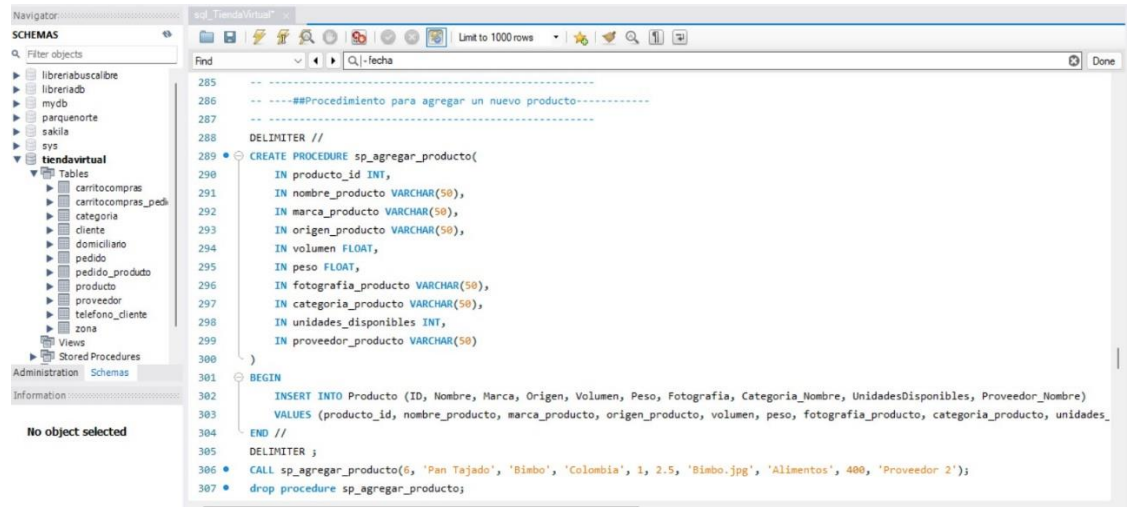
La columna "Nombre" contiene el nombre de cada categoría de productos.

La columna "CondicionAlmacenamiento" contiene información sobre las condiciones de almacenamiento necesarias para los productos de cada categoría.

Esta consulta es útil si se desea ver rápidamente una lista de todas las categorías de productos en la base de datos

PROCEDIMIENTOS ALMACENADOS:

1. Procedimiento para agregar un nuevo producto:



El procedimiento acepta diez parámetros de entrada con los nombres "producto_id", "nombre_producto", "marca_producto", "origen_producto", "volumen", "peso", "fotografia_producto", "categoria_producto", "unidades_disponibles" y "proveedor_producto".

El procedimiento realiza una única operación: inserta una nueva fila en la tabla "Producto" con los valores de los parámetros de entrada proporcionados. La instrucción de inserción se realiza mediante la cláusula INSERT INTO, donde se especifica la tabla "Producto" y los nombres de las columnas a las que se van a agregar los datos. Los valores de las columnas provienen de los parámetros de entrada especificados en la instrucción VALUES.

La instrucción BEGIN y END se utilizan para agrupar el cuerpo del procedimiento almacenado. La cláusula DELIMITER se utiliza para cambiar el delimitador de instrucciones para que el código pueda ser analizado correctamente por el motor de base de datos.

Una vez que el procedimiento almacenado se ha definido, se puede llamar utilizando la instrucción CALL, proporcionando los valores para los parámetros de entrada definidos en el procedimiento. En este caso, el procedimiento se llama con los valores de los parámetros para agregar un nuevo producto a la tabla "Producto".

2. Procedimiento para actualizar la cantidad de unidades disponibles de un producto:

```
308 -----
309 -- ---##Procedimiento para actualizar la cantidad de unidades disponibles de un producto-----
310 -----
311 DELIMITER //
312 • CREATE PROCEDURE sp_actualizar_unidades_disponibles(
313     IN producto_id INT,
314     IN nuevas_unidades INT
315 )
316 • BEGIN
317     UPDATE Producto
318     SET UnidadesDisponibles = nuevas_unidades
319     WHERE ID = producto_id;
320 END //
321 DELIMITER ;
322 • CALL sp_actualizar_unidades_disponibles(1, 123);
```

El procedimiento acepta dos parámetros de entrada: "producto_id" e "nuevas_unidades".

El procedimiento actualiza la columna "UnidadesDisponibles" de la tabla "Producto" con el valor proporcionado en el parámetro "nuevas_unidades" para el producto con el ID especificado en el parámetro "producto_id". La instrucción UPDATE se utiliza para modificar la tabla "Producto" y cambiar el valor de la columna "UnidadesDisponibles" con el nuevo valor especificado.

La cláusula WHERE se utiliza para especificar la fila o filas en la tabla "Producto" que deben actualizarse. En este caso, se especifica que la actualización debe aplicarse a la fila que tenga el ID de producto que se proporciona en el parámetro "producto_id".

La instrucción BEGIN y END se utilizan para agrupar el cuerpo del procedimiento almacenado. La cláusula DELIMITER se utiliza para cambiar el delimitador de instrucciones para que el código pueda ser analizado correctamente por el motor de base de datos.

Una vez que el procedimiento almacenado se ha definido, se puede llamar utilizando la instrucción CALL, proporcionando los valores para los parámetros de entrada definidos en el procedimiento. En este caso, el procedimiento se llama para actualizar las unidades disponibles de un producto específico en la tabla "Producto".

3. Procedimiento para consultar inventario:

```
323  -- -----
324  -- ----##Procedimiento para consultar inventario-----
325  -- -----
326  DELIMITER //
327  • CREATE PROCEDURE sp_consultar_inventario()
328  BEGIN
329      SELECT ID, Nombre, Marca, Origen, Volumen, Peso, Fotografia, Categoria_Nombre, UnidadesDisponibles, Proveedor_Nombre
330      FROM Producto;
331  END //
332  DELIMITER ;
333  • CALL sp_consultar_inventario();
334
```

ID	Nombre	Marca	Origen	Volumen	Peso	Fotografia	Categoria_Nombre	UnidadesDisponibles	Proveedor_Nombre
1	Celular	Samsung	Corea del Sur	200.00	0.30	celular.jpg	Electrónicos	123	Proveedor 1
2	Arroz	Diana	Colombia	1000.00	1.50	arroz.jpg	Alimentos	500	Proveedor 2
3	Leche	Alquería	Colombia	1.00	1.00	leche.jpg	Alimentos	200	Proveedor 2
4	Pan Tajado	Bimbo	Colombia	1.00	0.24	Bimbo.jpg	Alimentos	400	Proveedor 2
5	Pan Tajado	Bimbo	Colombia	1.00	0.60	Bimbo.jpg	Alimentos	400	Proveedor 2
6	Pan Tajado	Bimbo	Colombia	1.00	2.50	Bimbo.jpg	Alimentos	400	Proveedor 2

El procedimiento no acepta parámetros de entrada y simplemente selecciona todos los registros de la tabla "Producto".

La instrucción SELECT se utiliza para obtener los valores de las columnas "ID", "Nombre", "Marca", "Origen", "Volumen", "Peso", "Fotografia", "Categoria_Nombre", "UnidadesDisponibles" y "Proveedor_Nombre" de la tabla "Producto". La instrucción FROM se utiliza para especificar la tabla de la que se seleccionarán los datos.

El resultado de la consulta se muestra al final del procedimiento almacenado. Los resultados de la consulta se muestran en el cliente de MySQL que ejecuta la consulta.

La cláusula BEGIN y END se utilizan para agrupar el cuerpo del procedimiento almacenado. La cláusula DELIMITER se utiliza para cambiar el delimitador de instrucciones para que el código pueda ser analizado correctamente por el motor de base de datos.

Una vez que el procedimiento almacenado se ha definido, se puede llamar utilizando la instrucción CALL, sin proporcionar ningún valor de entrada. En este caso, el procedimiento se llama para mostrar una lista de todos los productos en la tabla "Producto".

4. Procedimiento para eliminar un producto de la tabla producto:

```
335  -----
336  -- ---##Procedimiento para eliminar un producto-----
337  -----
338  DELIMITER //
339  ● CREATE PROCEDURE sp_eliminar_producto_con_pedidos(
340      IN producto_id INT
341  )
342  ● BEGIN
343      -- Eliminar los registros en Pedido_Producto que corresponden al producto
344      DELETE FROM Pedido_Producto
345      WHERE Producto_ID = producto_id;
346
347      -- Eliminar el producto de la tabla Producto
348      DELETE FROM Producto
349      WHERE ID = producto_id;
350  END //
351  DELIMITER ;
352  ● CALL sp_eliminar_producto_con_pedidos(4);
---
```

El procedimiento acepta un parámetro de entrada llamado "producto_id".

El procedimiento realiza dos operaciones. Primero, elimina todas las filas de la tabla "Pedido_Producto" que corresponden al producto con el ID especificado en el parámetro "producto_id". La instrucción DELETE se utiliza para eliminar las filas de la tabla "Pedido_Producto" donde el valor de la columna "Producto_ID" coincide con el valor del parámetro "producto_id".

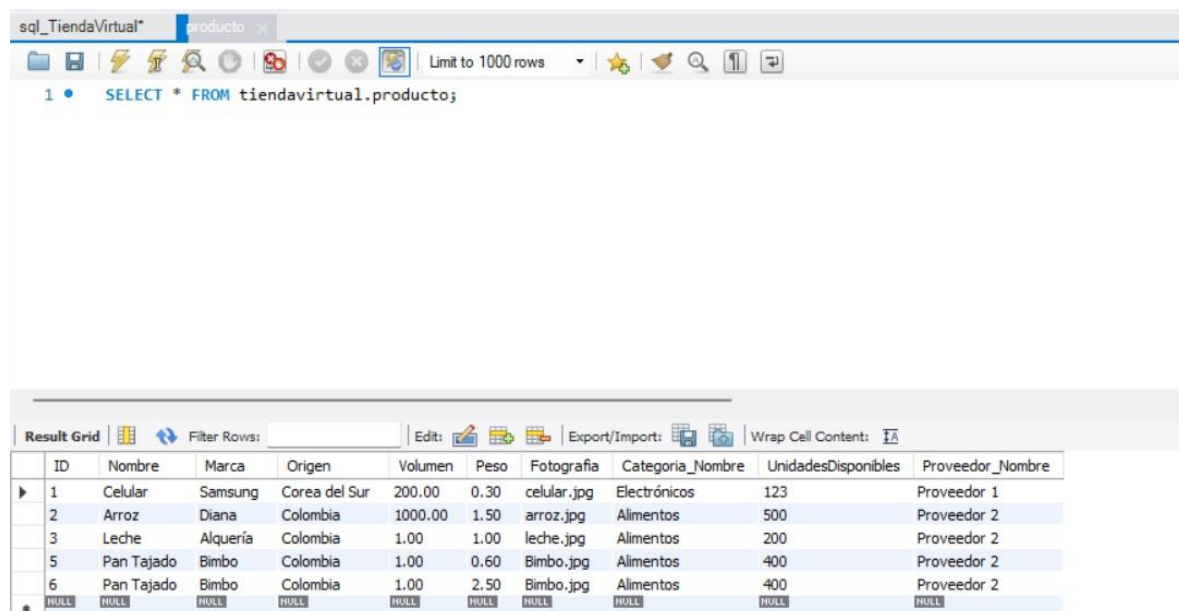
A continuación, el procedimiento elimina la fila correspondiente al producto con el ID especificado en el parámetro "producto_id" de la tabla "Producto". La instrucción DELETE se utiliza para eliminar la fila de la tabla "Producto" donde el valor de la columna "ID" coincide con el valor del parámetro "producto_id".

La cláusula BEGIN y END se utilizan para agrupar el cuerpo del procedimiento almacenado. La cláusula DELIMITER se utiliza para cambiar el delimitador de instrucciones para que el código pueda ser analizado correctamente por el motor de base de datos.

Una vez que el procedimiento almacenado se ha definido, se puede llamar utilizando la instrucción CALL, proporcionando el valor del parámetro "producto_id" para eliminar el producto y sus referencias en la tabla "Pedido_Producto".

Producto eliminado:

En la siguiente imagen vemos que el producto 4 ya no se encuentra en la tabla de productos.



ID	Nombre	Marca	Origen	Volumen	Peso	Fotografia	Categoria_Nombre	UnidadesDisponibles	Proveedor_Nombre
1	Celular	Samsung	Corea del Sur	200.00	0.30	celular.jpg	Electrónicos	123	Proveedor 1
2	Arroz	Diana	Colombia	1000.00	1.50	arroz.jpg	Alimentos	500	Proveedor 2
3	Leche	Alquería	Colombia	1.00	1.00	leche.jpg	Alimentos	200	Proveedor 2
5	Pan Tajado	Bimbo	Colombia	1.00	0.60	Bimbo.jpg	Alimentos	400	Proveedor 2
6	Pan Tajado	Bimbo	Colombia	1.00	2.50	Bimbo.jpg	Alimentos	400	Proveedor 2
	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

5. Procedimiento para AGREGAR DOMICILIARIO:

```
354  -----
355  -- ----##Procedimiento para AGREGAR DOMICILIARIO-----
356  -----
357  DELIMITER //
358  ● CREATE PROCEDURE sp_agregar_domiciliario(
359      IN domiciliario_id INT,
360      IN nombre_domiciliario VARCHAR(100),
361      IN matricula_furgoneta VARCHAR(10),
362      IN zona_codigopostal INT
363  )
364  ● BEGIN
365      INSERT INTO Domiciliario (ID, Nombre, NumeroMatriculaFurgoneta, Zona_CodigoPostal)
366      VALUES (domiciliario_id, nombre_domiciliario, matricula_furgoneta, zona_codigopostal);
367  END //
368  DELIMITER ;
369  ● CALL sp_agregar_domiciliario(3, 'Ana López', 'XYZ987', 1030);
370  ● CALL sp_agregar_domiciliario(4, 'luis', 'XYZ980', 1040);
371
```

El procedimiento acepta cuatro parámetros de entrada: "domiciliario_id", "nombre_domiciliario", "matricula_furgoneta" y "zona_codigopostal".

El procedimiento inserta una nueva fila en la tabla "Domiciliario". La instrucción INSERT INTO se utiliza para insertar un nuevo registro en la tabla "Domiciliario" con los valores proporcionados en los parámetros de entrada.

La lista de columnas especificadas en la instrucción INSERT INTO ("ID", "Nombre", "NumeroMatriculaFurgoneta", "Zona_CodigoPostal") se corresponde con las columnas de la tabla "Domiciliario". Los valores proporcionados en los parámetros de entrada se corresponden con los valores que se insertan en la tabla.

La cláusula BEGIN y END se utilizan para agrupar el cuerpo del procedimiento almacenado. La cláusula DELIMITER se utiliza para cambiar el delimitador de instrucciones para que el código pueda ser analizado correctamente por el motor de base de datos.

Una vez que el procedimiento almacenado se ha definido, se puede llamar utilizando la instrucción CALL, proporcionando los valores para los parámetros de entrada definidos en el procedimiento. En este caso, el procedimiento se llama para agregar un nuevo domiciliario a la tabla "Domiciliario".

TRIGGERS:

Trigger para verificar si hay domiciliario disponible en esa zona que ingresa el cliente antes de registrar un cliente:

```
378  -----
379  -- ----#Trigger para verificar si hay domiciliario disponible en esa zona que ingresa el cliente antes de registrar un cliente: ----
380  -----
381
382  DELIMITER //
383  • CREATE TRIGGER tr_verificar_domiciliarios
384  BEFORE INSERT ON Cliente
385  FOR EACH ROW
386  BEGIN
387      DECLARE num_domiciliarios INT;
388
389      SELECT COUNT(*) INTO num_domiciliarios
390      FROM Domiciliario
391      WHERE Zona_CodigoPostal = NEW.Zona_CodigoPostal;
392
393      IF num_domiciliarios = 0 THEN
394          SIGNAL SQLSTATE '45000'
395          SET MESSAGE_TEXT = 'No existen domiciliarios para la zona del cliente';
396      END IF;
397  END//
398  DELIMITER ;
399
```

Este es un trigger que se ejecuta automáticamente antes de insertar un nuevo registro en la tabla "Cliente" en MySQL. El trigger comprueba si hay domiciliarios disponibles para la zona del cliente que se está insertando y si no hay, envía una señal de error que impide la inserción.

Más específicamente, este trigger realiza lo siguiente:

Declara una variable "num_domiciliarios" para almacenar el número de domiciliarios disponibles para la zona del cliente.

Utiliza la sentencia "SELECT COUNT(*) INTO num_domiciliarios" para contar el número de registros en la tabla "Domiciliario" que tienen el mismo código postal de zona que el cliente que se está insertando. Esta información se almacena en la variable "num_domiciliarios".

Comprueba si el número de domiciliarios disponibles es igual a cero. Si es así, utiliza la sentencia "SIGNAL" para enviar una señal de error con el mensaje "No existen domiciliarios para la zona del cliente". Si el número de domiciliarios disponibles no es igual a cero, la inserción de datos en la tabla "Cliente" continúa normalmente.

En resumen, este trigger se utiliza para garantizar que siempre haya al menos un domiciliario disponible para cada zona de clientes. Si no hay domiciliarios disponibles para una zona en particular, la inserción de datos en la tabla "Cliente" se detiene y se envía un mensaje de error indicando que no se pueden insertar registros en la tabla "Cliente".

Con este trigger se valida el primer requerimiento para poder enviar un pedido para ser más específico: "El cliente deberá pertenecer a una zona (Código Postal) donde existan domiciliarios. Un domiciliario se identifica mediante un nombre, número de matrícula de la furgoneta y zona donde reparte."

Trigger para actualizar la cantidad de unidades disponibles de un producto:

```
-- -----  
-- ----#Trigger para actualizar la cantidad de unidades disponibles de un producto: -----  
-- -----  
  
DELIMITER //  
CREATE TRIGGER tr_actualizar_unidades_disponibles  
AFTER INSERT ON Pedido_Producto  
FOR EACH ROW  
BEGIN  
    UPDATE Producto  
    SET UnidadesDisponibles = UnidadesDisponibles - NEW.Cantidad  
    WHERE ID = NEW.Producto_ID;  
END//  
DELIMITER ;
```

Este trigger se activa automáticamente después de que se inserta una nueva fila en la tabla "Pedido_Producto".

Lo que hace es actualizar la cantidad de unidades disponibles en la tabla "Producto" basándose en la cantidad que se agregó en la tabla "Pedido_Producto". Para hacerlo, resta la cantidad de la columna "UnidadesDisponibles" de la fila correspondiente en la tabla "Producto" con la cantidad insertada en la tabla "Pedido_Producto" (accesible a través de la variable "NEW").

En resumen, este trigger actualiza la cantidad de unidades disponibles de un producto automáticamente cada vez que se hace un nuevo pedido.

Trigger para validar la cantidad de unidades disponibles antes de insertar un nuevo registro en la tabla "Pedido_Producto":

```
414  -- -----
415  -- ----#Trigger para validar la cantidad de unidades disponibles antes de insertar un nuevo registro en la tabla "Pedido_Producto"; ----
416  -- -----
417  DELIMITER //
418  • CREATE TRIGGER tr_validar_unidades_disponibles
419  BEFORE INSERT ON Pedido_Producto
420  FOR EACH ROW
421  BEGIN
422      DECLARE unidades_disponibles INT;
423      SELECT UnidadesDisponibles INTO unidades_disponibles
424      FROM Producto
425      WHERE ID = NEW.Producto_ID;
426
427      IF unidades_disponibles < NEW.Cantidad THEN
428          SIGNAL SQLSTATE '45000'
429          SET MESSAGE_TEXT = 'La cantidad solicitada no está disponible en el inventario';
430      END IF;
431  END//
432  DELIMITER ;
```

se activa automáticamente antes de que se inserte una nueva fila en la tabla "Pedido_Producto".

Lo que hace es verificar si la cantidad solicitada del producto en la nueva fila a insertar está disponible en la tabla "Producto", basándose en la cantidad de unidades disponibles en la columna "UnidadesDisponibles". Para hacerlo, primero asigna el valor de la columna "UnidadesDisponibles" de la fila correspondiente en la tabla "Producto" a la variable "unidades_disponibles".

Luego, si la cantidad solicitada en la nueva fila a insertar (accesible a través de la variable "NEW") es mayor que la cantidad de unidades disponibles, el trigger genera un error con el mensaje "La cantidad solicitada no está disponible en el inventario" mediante la instrucción "SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La cantidad solicitada no está disponible en el inventario'".

En resumen, este trigger asegura que no se puedan insertar filas en la tabla "Pedido_Producto" que soliciten más unidades de un producto de las que están disponibles en la tabla "Producto" y con este trigger se concluye las dos validaciones para poder enviar un pedido para ser más específico esta validación: "Debe haber unidades suficientes por cada producto para satisfacer las demandas de cada pedido."

Trigger que se activará automáticamente cada vez que se inserte un nuevo registro en la tabla "Domiciliario" y actualizará la columna "FechaModificacion" del registro correspondiente:

```
433 -----
434 -- ----#Trigger se activará automáticamente cada vez que se inserte un nuevo registro en la tabla "Domiciliario" y actualizará la columna "F
435 -----
436
437
438 DELIMITER //
439
440 • CREATE TRIGGER tr_actualizar_registro_domiciliario
441   AFTER INSERT ON Domiciliario
442   FOR EACH ROW
443   BEGIN
444       UPDATE Domiciliario
445       SET FechaModificacion = CURRENT_TIMESTAMP
446       WHERE ID = NEW.ID;
447   END //
448
449 DELIMITER ;
450
```

Este trigger se dispara después de una inserción en la tabla "Domiciliario" y ejecuta la instrucción UPDATE para actualizar el valor de la columna "FechaModificacion" de la fila correspondiente al nuevo registro insertado en la tabla. La cláusula NEW se utiliza para hacer referencia a la fila recién insertada.

La cláusula FOR EACH ROW especifica que el trigger se debe activar una vez para cada fila insertada en la tabla "Domiciliario".

La cláusula DELIMITER se utiliza para cambiar el delimitador de instrucciones para que el código pueda ser analizado correctamente por el motor de base de datos.

Una vez que el trigger se ha definido, se activará automáticamente cada vez que se inserte un nuevo registro en la tabla "Domiciliario" y actualizará la columna "FechaModificacion" del registro correspondiente.

INGRESANDO REGISTROS DESDE JAVA

Después de terminar los triggers mi siguiente paso fue hacer la configuración en java para poder ingresar los registros en las tablas y de una manera mas automática.

Para ello lo primero que hice fue crear un proyecto en java que utiliza gradle, después de esto lo que hice fue configurar mi base de datos.

Adjunto algunas imágenes de esta configuración:

```
1 package com.sofkau.integration.database;
2
3 import java.sql.*;
4
5 6 usages
6 public class MySqlConnection {
7     2 usages
8     private String connectionString;
9     2 usages
10    private String dbName = "";
11    2 usages
12    private String dbUser = "";
13    2 usages
14    private String dbPassword = "";
15
16    8 usages
17    Connection connection;
18
19    1 usage
20    public MySqlConnection (String dbName, String dbUser, String dbPassword){
21        this.dbName = dbName;
22        this.dbUser = dbUser;
23        this.dbPassword = dbPassword;
24    }
25
26    1 usage
27    public void connect(){
28        try{
29            if(this.connection == null){
30                Class.forName( className: "com.mysql.cj.jdbc.Driver");
31                System.out.println("Conectándose a mysql");
32            }
33        }
34    }
35 }
```

```

23         System.out.println("Conectándose a mysql");
24         this.connectionString = "jdbc:mysql://localhost:3306/" + this.dbName;
25         this.connection = DriverManager.getConnection(
26             this.connectionString,
27             this.dbUser,
28             this.dbPassword
29         );
30         if (this.connection == null){
31             System.out.println("Error de conexión");
32         } else {
33             System.out.println("Conexión exitosa");
34         }
35     }
36     }catch(ClassNotFoundException e){
37         e.printStackTrace();
38     }catch(SQLException e){
39         e.printStackTrace();
40     }
41 }
42
43 1 usage
44 public void closeConnection (){
45     try {
46         if(this.connection != null){
47             this.connection.close();
48         }
49     }catch (SQLException e){
50         e.printStackTrace();
51     }
52 }

```

11 usages

```
54 public PreparedStatement getStatement(String sql){  
55     try{  
56         return this.connection.prepareStatement(sql);  
57     }catch (SQLException e){  
58         e.printStackTrace();  
59         return null;  
60     }  
61 }
```

no usages

```
64 public CallableStatement getCallable (String sql){  
65     try{  
66         return this.connection.prepareCall(sql);  
67     }catch (SQLException e){  
68         e.printStackTrace();  
69         return null;  
70     }  
71 }
```

no usages

```
73 @ public boolean insert(PreparedStatement statement){  
74     try{  
75         int rowsInserted = statement.executeUpdate();  
76         return rowsInserted > 0;  
77     }catch (SQLException e){  
78         e.printStackTrace();
```

```

79         return false;
80     }
81 }
82
no usages
83 @ public boolean insert(CallableStatement procedure){
84     try{
85         int rowsInserted = procedure.executeUpdate();
86         return rowsInserted > 0;
87     }catch (SQLException e){
88         e.printStackTrace();
89         return false;
90     }
91 }
92
no usages
93 public Connection getConnector() { return this.connection; }
96 }
97

```

Esta fue la configuración básica que hice para mi base de datos.

En la siguiente imagen ingrese los datos para poder acceder a la base de datos y también llamar mis métodos de la clase tienda. Donde puedo ejecutar cada uno para poder ingresar registros a mi base de datos.

```

6 public class Main {
7     no usages
8     public static void main(String[] args) {
9         MySQLConnector connector = new MySQLConnector(
10             dbName: "TiendaVirtual",
11             dbUser: "root",
12             dbPassword: "123456789"
13         );
14         connector.connect();
15
16         Tienda tienda = new Tienda(connector);
17
18         //tienda.insertarZona();
19         //tienda.insertarDomiciliario();
20         //tienda.insertarCliente();
21         //tienda.insertarTelefonoCliente();
22         //tienda.insertarCategoria();
23         //tienda.insertarProveedor();
24         //tienda.insertarProducto();
25         //tienda.insertarPedido();
26         //tienda.insertarPedidoProducto();
27         //tienda.insertarCarritoCompras();
28         //tienda.insertarCarcomprasPedido();
29
30         connector.closeConnection();
31     }
32 }
33

```

En la imagen siguiente voy a explicar como ingrese los registros en dos de mis tablas:

```
12 public class Tienda {
13     12 usages
14     MySqlConnection mySqlConnection;
15
16     1 usage
17     public Tienda(MySqlConnection mySqlConnection) { this.mySqlConnection = mySqlConnection; }
18     no usages
19     public void insertarZona() {
20         try {
21             Faker faker = new Faker();
22             String query = "INSERT INTO zona (CodigoPostal)" +
23                 "VALUES (?)";
24             PreparedStatement statement = this.mySqlConnection.getStatement(query);
25             int CodigoPostal = 1050;
26             for (int i = 0; i <= 50; i++) {
27                 statement.setString( parameterIndex: 1,  x: (CodigoPostal + i) + "");
28                 statement.execute();
29             }
30             statement.close();
31         } catch (SQLException e) {
32             e.printStackTrace();
33         }
34     }
35 }
```

```
15     public void insertarDomiciliario() {
16         try {
17             Faker faker = new Faker();
18             String query = "INSERT INTO domiciliario (ID, Nombre, NumeroMatriculaFurgoneta, Zona_CodigoPostal)" +
19                 "VALUES (?, ?, ?, ?)";
20             PreparedStatement statement = this.mySqlConnection.getStatement(query);
21             int ID= 10;
22             String matricula="ARX";
23             int CodigoPostal = 1050;
24             for (int i = 0; i <= 50; i++) {
25                 statement.setString( parameterIndex: 1,  x: (ID + i) + "");
26                 statement.setString( parameterIndex: 2,  faker.name().firstName());
27                 statement.setString( parameterIndex: 3,  (matricula+ i));
28                 statement.setString( parameterIndex: 4,  x: (CodigoPostal + i) + "");
29             }
30             statement.execute();
31             statement.close();
32         } catch (SQLException e) {
33             e.printStackTrace();
34         }
35     }
36 }
```

En las dos imágenes anteriores se define la clase "Tienda", donde se ven dos de los métodos implementados: "insertarZona" e "insertarDomiciliario".

El método "insertarZona" genera códigos postales ficticios utilizando la biblioteca Faker, y los inserta en una tabla llamada "zona" en una base de datos TiendaVirtual de MySQL. El código de postal inicial es 1050, y se generan 50 códigos postales adicionales incrementando en 1 el valor del código postal anterior.

El método "insertarDomiciliario" también utiliza la biblioteca Faker para generar nombres ficticios de personas y matrículas de furgonetas. Los datos generados se insertan en una tabla llamada "domiciliario" en la base de datos TiendaVirtual de MySQL, junto con el ID y el código postal de la zona en la que operan.

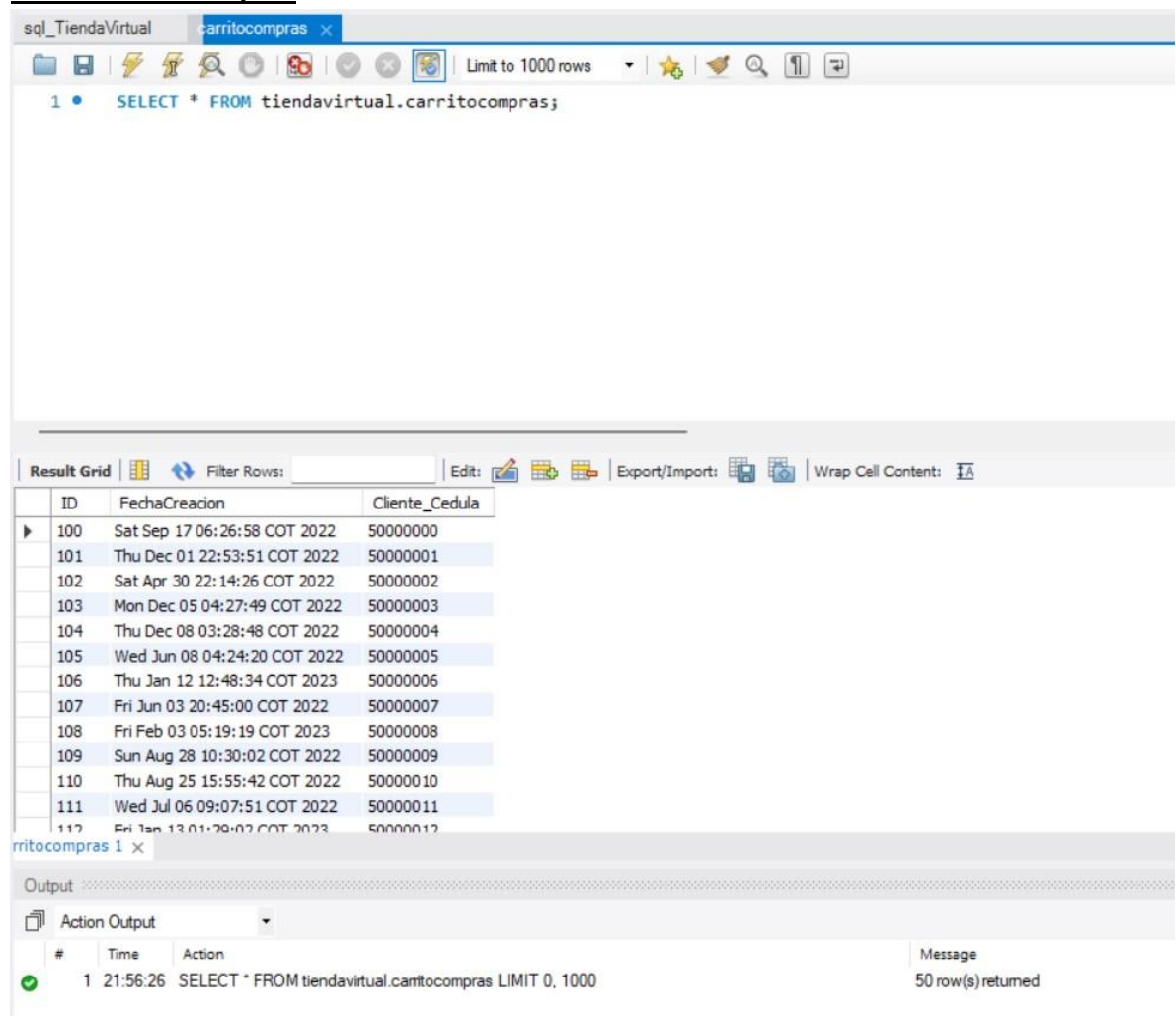
La conexión a la base de datos se realiza a través de una clase llamada "MySQLConnector", que se espera que se le pase como argumento al constructor de la clase Tienda. La variable "mysqlConnector" guarda esta referencia para su uso posterior en los métodos "insertarZona" e "insertarDomiciliario".

Los métodos utilizan objetos de tipo "PreparedStatement" para preparar y ejecutar las consultas SQL. Cada consulta utiliza un parámetro de signo de interrogación (?) para indicar un valor que se pasará posteriormente a través del método "setString".

En caso de que se produzca una excepción de tipo SQLException durante la ejecución de cualquiera de los métodos, se imprimen los errores mediante el método "printStackTrace".

Para finalizar el reto dejo las imágenes donde muestro los registros que ingrese en cada tabla:

1. Tabla CarritoCompras



ID	FechaCreacion	Cliente_Cedula
100	Sat Sep 17 06:26:58 COT 2022	50000000
101	Thu Dec 01 22:53:51 COT 2022	50000001
102	Sat Apr 30 22:14:26 COT 2022	50000002
103	Mon Dec 05 04:27:49 COT 2022	50000003
104	Thu Dec 08 03:28:48 COT 2022	50000004
105	Wed Jun 08 04:24:20 COT 2022	50000005
106	Thu Jan 12 12:48:34 COT 2023	50000006
107	Fri Jun 03 20:45:00 COT 2022	50000007
108	Fri Feb 03 05:19:19 COT 2023	50000008
109	Sun Aug 28 10:30:02 COT 2022	50000009
110	Thu Aug 25 15:55:42 COT 2022	50000010
111	Wed Jul 06 09:07:51 COT 2022	50000011
112	Fri Jan 13 01:29:02 COT 2023	50000012

#	Time	Action	Message
1	21:56:26	SELECT * FROM tiendavirtual.carritocompras LIMIT 0, 1000	50 row(s) returned

2. Tabla CarritoCompras Pedido

```
sql_TiendaVirtual  carritocompras  carritocompras_pedido x
Limit to 1000 rows
1 • SELECT * FROM tiendavirtual.carritocompras_pedido;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

	CarritoCompras_ID	CodigoPedido_FK	Cantidad
▶	100	1000	71
	101	1001	31
	102	1002	42
	103	1003	23
	104	1004	13
	105	1005	63
	106	1006	77
	107	1007	45
	108	1008	60
	109	1009	41
	110	1010	90
	111	1011	53
	112	1012	14

carrito_pedido 1 x

Output

3. Categoria

```
sql_TiendaVirtual  carritocompras  carritocompras_pedido  categoria x
Limit to 1000 rows
1 • SELECT * FROM tiendavirtual.categoria;
```

Result Grid

Filter Rows:

Edit: Export/Import:

	Nombre	CondicionesAlmacenamiento	Observaciones
►	Categoria0	36°C	Mostly cloudy
	Categoria1	38°C	Clear skies
	Categoria10	-4°C	Thunderstorms
	Categoria11	2°C	Hail
	Categoria12	0°C	Rain
	Categoria13	-20°C	Hail
	Categoria14	3°C	Thunderstorms
	Categoria15	-3°C	Partly cloudy
	Categoria16	22°C	Cloudy periods
	Categoria17	-11°C	Showers
	Categoria18	4°C	Drizzle
	Categoria19	-15°C	Sleet
	Categoria2	31°C	Sleet

categoria 1 x

Output

4. Cliente

sql_TiendaVirtual carritocompras carritocompras_pedido categoria **cliente** x

Limit to 1000 rows

```
1 • SELECT * FROM tiendavirtual.cliente;
```

Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:

	Cedula	Nombre	Direccion	Email	Contrasena	Zona_CodigoPostal
▶	50000000	Demarcus	146	chase.wiegand@hotmail.com	g8tmi6o65hm8	1050
	50000001	Clinton	382	wilmer.jacobi@yahoo.com	pflc6sjg81	1051
	50000002	Cordelia	874	javier.schmidt@gmail.com	gt54dizx	1052
	50000003	Modesto	225	sheree.rippin@gmail.com	kpdptyuyk4x	1053
	50000004	Lashawna	578	preston.cronin@yahoo.com	0kcpcwqw9sw0i	1054
	50000005	Susannah	634	makeda.quigley@gmail.com	zeg81x95j0vyh	1055
	50000006	Alia	289	melani.friesen@hotmail.com	qhl1yqt0akwy8	1056
	50000007	Trevor	735	darnell.larson@yahoo.com	y5jwxhn38nsh	1057
	50000008	Dorian	869	josefina.hills@hotmail.com	kmlfq07fpq	1058
	50000009	Nu	355	rachell.mclaughlin@hotmail.com	s24jgplt8krst	1059
	50000010	Ronnie	10	colin.beahan@hotmail.com	miyt4j4hyy60s3	1060
	50000011	Temika	471	myles.brakus@gmail.com	37x4yzny18mlf	1061
	50000012	Teider	308	luanna.okuneva@hotmail.com	m4hlefd4u	1062

cliente 1 x

Output

5. Domiciliario

sql_TiendaVirtual carritocompras carritocompras_pedido categoria cliente domiciliario x

Limit to 1000 rows

```
1 • SELECT * FROM tiendavirtual.domiciliario;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: IA

	ID	Nombre	NumeroMatriculaFurgoneta	Zona_CodigoPostal
▶	10	Mellie	ARX0	1050
	11	Randall	ARX1	1051
	12	Adele	ARX2	1052
	13	Eliana	ARX3	1053
	14	Merlin	ARX4	1054
	15	Danuta	ARX5	1055
	16	Raymon	ARX6	1056
	17	Robby	ARX7	1057
	18	Edmond	ARX8	1058
	19	Norris	ARX9	1059
	20	Clayton	ARX10	1060
	21	Brendan	ARX11	1061
	22	Lavonne	APV12	1062

domiciliario 1 x

6. Pedido

sql_TiendaVirtual **pedido** x

Limit to 1000 rows

1 • SELECT * FROM tiendavirtual.pedido;

Aut
disal
mai
curr
to

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	CodigoPedido	FechaPedido	Cliente_Cedula	Domiciliario_ID	DireccionEntrega	ImporteTotal	NumeroTarjeta	FechaCaducidad
▶	1000	Sun Dec 04 14:58:16 COT 2022	50000000	10	xx	753136.00	1234-2121-1221-1211	Mon Feb 06 16:03:11 COT 2023
	1001	Fri Nov 04 18:06:20 COT 2022	50000001	11	xx	857788.00	1212-1221-1121-1234	Fri Feb 17 00:16:37 COT 2023
	1002	Wed Oct 12 21:52:39 COT 2022	50000002	12	xx	306185.00	1228-1221-1221-1431	Sun Jan 15 23:32:27 COT 2023
	1003	Wed May 25 01:25:56 COT 2022	50000003	13	xx	539375.00	1211-1221-1234-2201	Tue Feb 07 18:26:46 COT 2023
	1004	Wed Oct 26 06:49:59 COT 2022	50000004	14	xx	152277.00	1211-1221-1234-2201	Mon Feb 13 05:16:07 COT 2023
	1005	Fri Sep 23 00:26:06 COT 2022	50000005	15	xx	416341.00	1211-1221-1234-2201	Tue Feb 14 09:04:47 COT 2023
	1006	Sat Dec 24 23:12:14 COT 2022	50000006	16	xx	470835.00	1211-1221-1234-2201	Wed Feb 08 22:54:28 COT 2023
	1007	Thu Oct 06 21:40:39 COT 2022	50000007	17	xx	632977.00	1228-1221-1221-1431	Thu Jan 26 18:20:14 COT 2023
	1008	Thu Jan 05 00:21:02 COT 2023	50000008	18	xx	465105.00	1228-1221-1221-1431	Fri Jan 13 21:28:57 COT 2023
	1009	Tue Nov 29 07:33:00 COT 2022	50000009	19	xx	680351.00	1212-1221-1121-1234	Thu Feb 09 09:56:36 COT 2023
	1010	Thu Dec 29 15:44:06 COT 2022	50000010	20	xx	849387.00	1228-1221-1221-1431	Wed Jan 11 21:20:06 COT 2023
	1011	Fri Sep 02 03:35:45 COT 2022	50000011	21	xx	70707.00	1212-1221-1121-1234	Thu Feb 14 20:01:34 COT 2023

pedido 1 x

Apply Revert Context

Output

7. Pedido Producto

sql_TiendaVirtual **pedido** **pedido_producto** x

Limit to 1000 rows

1 • SELECT * FROM tiendavirtual.pedido_producto;

Result Grid | Filter Rows: | Edit: |

	Pedido_CodigoPedido	Producto_ID	Cantidad
▶	1000	1	51
	1001	2	81
	1002	3	75
	1003	4	45
	1004	5	19
	1005	6	10
	1006	7	77
	1007	8	3
	1008	9	24
	1009	10	61
	1010	11	4
	1011	12	65
	1012	13	37

pedido_producto 1 x

Output

8. Producto

sql_TiendaVirtual pedido pedido_producto producto

Limit to 1000 rows

1 • SELECT * FROM tiendavirtual.producto;

Result Grid

	ID	Nombre	Marca	Origen	Volumen	Peso	Fotografia	Categoria_Nombre
▶	1	Tacos	Ritchie Group	Germany	0.00	67.00	com.github.javafaker.Artist@13cc0b90	Categoria0
	2	Poutine	Kohler-Howell	Saint Martin	0.00	71.00	com.github.javafaker.Artist@13cc0b90	Categoria1
	3	Ricotta Stuffed Ravioli	Guskowski Inc	Georgia	51.00	51.00	com.github.javafaker.Artist@13cc0b90	Categoria2
	4	Pork Sausage Roll	Harvey, Ward and Maggio	Turkey	86.00	90.00	com.github.javafaker.Artist@13cc0b90	Categoria3
	5	Fish and Chips	Luetngen Group	Brunei Darussalam	67.00	37.00	com.github.javafaker.Artist@13cc0b90	Categoria4
	6	Pappardelle alla Bolognese	Welch, Lynch and Wuckert	American Samoa	55.00	74.00	com.github.javafaker.Artist@13cc0b90	Categoria5
	7	Chicken Wings	Schmidt, Hirthe and Ebert	Taiwan	20.00	38.00	com.github.javafaker.Artist@13cc0b90	Categoria6
	8	Bruschette with Tomato	Turcotte-Boyer	Chile	99.00	7.00	com.github.javafaker.Artist@13cc0b90	Categoria7
	9	Fish and Chips	Christiansen Group	French Southern Territories	25.00	83.00	com.github.javafaker.Artist@13cc0b90	Categoria8
	10	Massaman Curry	Towne, Abshire and Nolan	Trinidad and Tobago	99.00	85.00	com.github.javafaker.Artist@13cc0b90	Categoria9
	11	Cheeseburger	Bailey-Macejkovic	Saint Helena	45.00	17.00	com.github.javafaker.Artist@13cc0b90	Categoria10
	12	Dark Rolly Bun	Marvin, Karlika and Grimes	United States of America	30.00	82.00	com.github.javafaker.Artist@13cc0b90	Categoria11

producto 1 x

Apply Revert Context

9. Proveedor

sql_TiendaVirtual pedido pedido_producto producto proveedor

Limit to 1000 rows

1 • SELECT * FROM tiendavirtual.proveedor;

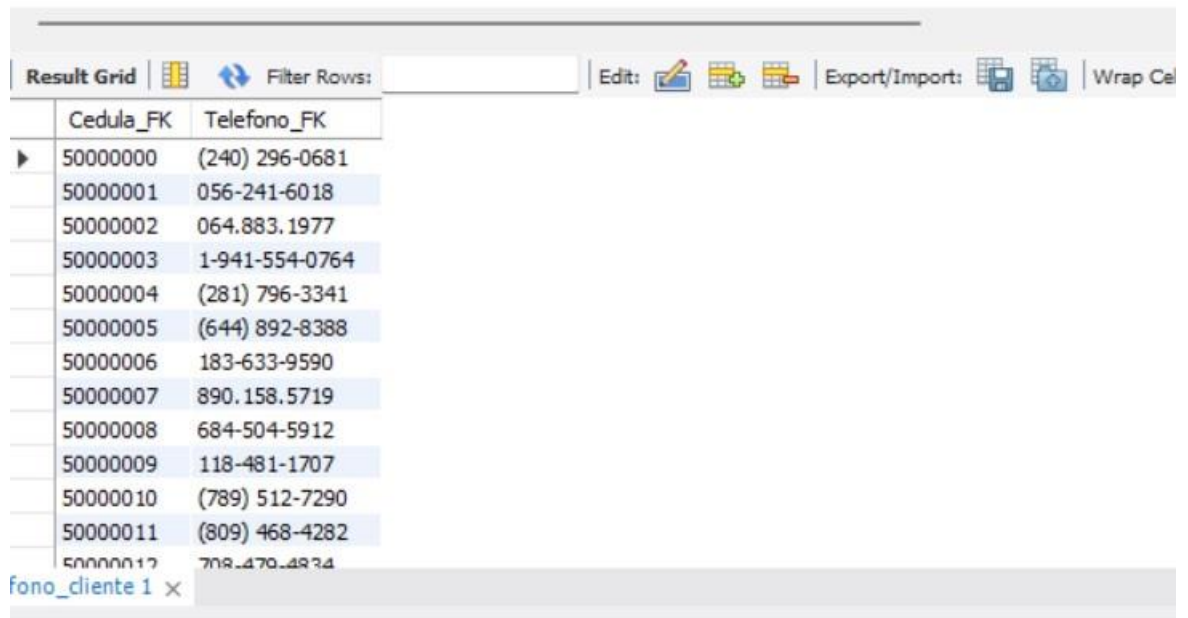
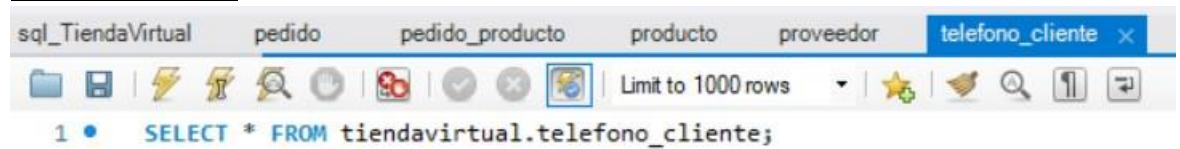
Result Grid

	Nombre	Direccion	Telefono	Email
▶	Proveedor-0	North Mireilleton	1-381-107-9077	freddie.wintheiser@gmail.com
	Proveedor-1	Lancetown	1-752-459-0632	terisa.littel@yahoo.com
	Proveedor-10	Blandaburgh	177.609.8198	mike.haag@gmail.com
	Proveedor-11	Lake Marxmouth	331-798-9276	shirley.robel@hotmail.com
	Proveedor-12	Tillmantown	1-159-113-4216	slyvia.abbott@gmail.com
	Proveedor-13	Port Grace	375.733.6940	billie.doyle@gmail.com
	Proveedor-14	Lake Tobymouth	(276) 255-8821	brant.runte@gmail.com
	Proveedor-15	West Milfordfort	(379) 146-9448	beatrice.kovacek@hotmail.com
	Proveedor-16	Rossieborough	1-768-418-7926	cherilyn.schuppe@gmail.com
	Proveedor-17	Yoshikochester	326-490-3927	wilfredo.tromp@hotmail.com
	Proveedor-18	Edisonbury	(976) 333-3242	le.dare@yahoo.com
	Proveedor-19	North Simacheater	771.791.5837	brendon.schimmel@hotmail.com
	Proveedor-2	North Trinehaven	1-848-470-8588	terrell.yost@ushah.com

proveedor 1 x

Output

10. Teléfono Cliente



The screenshot shows a SQL query result grid with a toolbar at the top. The toolbar includes icons for 'Result Grid', 'Filter Rows', 'Edit', 'Export/Import', and 'Wrap Cell'. The data is displayed in a table with two columns: 'Cedula_FK' and 'Telefono_FK'.

	Cedula_FK	Telefono_FK
▶	50000000	(240) 296-0681
	50000001	056-241-6018
	50000002	064.883.1977
	50000003	1-941-554-0764
	50000004	(281) 796-3341
	50000005	(644) 892-8388
	50000006	183-633-9590
	50000007	890.158.5719
	50000008	684-504-5912
	50000009	118-481-1707
	50000010	(789) 512-7290
	50000011	(809) 468-4282
	50000012	708.470.4834

fono_cliente 1 x

11. Zona

sql_TiendaVirtual pedido pedido_producto producto p

Limit to 1000 rows

```
1 • SELECT * FROM tiendavirtual.zona;
```

Result Grid | Filter Rows: | Edit:

	CodigoPostal
▶	1050
	1051
	1052
	1053
	1054
	1055
	1056
	1057
	1058
	1059
	1060
	1061
	1062

zona 1 x

Output

¿Está conforme con el resultado obtenido según el contexto o cree que hubiera obtenido un mejor resultado con una base de datos no relacional?

En lo personal no he tenido tanta experiencia manejando bases de datos no relacionales, pero por lo que llegué a consultar en internet sobre ellas entendí lo que son aquellas que no utilizan un esquema predefinido y no tienen una estructura fija. En lugar de tablas, utilizan diferentes estructuras de datos como documentos, grafos, claves-valor o columnas. No utilizan SQL para consultar y manipular datos, y son más flexibles que las bases de datos relacionales. Son ideales para almacenar grandes cantidades de información no estructurada y para escenarios en los que se necesita alta escalabilidad y disponibilidad.

Por otro lado, las bases de datos relacionales son aquellas que almacenan la información en tablas estructuradas y utilizan un esquema definido para relacionar las tablas entre sí. Utilizan un lenguaje de consulta estructurado (SQL) para insertar, actualizar, eliminar y consultar datos. El modelo relacional está diseñado para asegurar la integridad de los datos y garantizar que no se repita la información en la base de datos. Estas bases de datos son ideales para almacenar grandes cantidades de información estructurada y para garantizar la integridad de los datos.

Lo que me lleva a concluir que, para el ejercicio propuesto en el reto, una base de datos relacional podría ser la mejor opción, ya que se trata de una aplicación que maneja datos estructurados con relaciones entre ellos. Se requiere almacenar información de los clientes, productos, categorías, pedidos y domiciliarios, y es importante asegurar la integridad de los datos para garantizar la precisión en los procesos de la aplicación. Además, una base de datos relacional permite hacer consultas complejas y filtrar datos de acuerdo con distintos criterios de cada persona y esto es muy conveniente para una tienda virtual.