

Reto BD Tienda Don Pepe

Se pide:

- Indicar que ejercicio fue asignado
- Realizar el modelo E-R
- Realizar el modelo relacional
- Normalizar correctamente
- Escribir con sentencias SQL toda la definición de la base de datos.
- Escribir consultas que me permitan ver la información de cada tabla o de varias tablas (10).
- Generar de 4 a 6 vistas donde se evidencie lo más importante de cada ejercicio (haga una selección muy responsable de la información realmente importante según el contexto).
- Generar al menos 4 procedimientos almacenados.
- Generar al menos 4 triggers
- Poblar la base de datos (50 registros por tabla) utilizando una conexión desde Java.
- Al terminar el ejercicio responda ¿Está conforme con el resultado obtenido según el contexto o cree que hubiera obtenido un mejor resultado con una base de datos no relacional?
- documente muy bien su proceso (paso a paso) en un archivo PDF escriba todas las aclaraciones o especificaciones necesarias para realizar el ejercicio.

CONTEXTO

Tienda Virtual Don pepe (Ejercicio C)

Don pepe quiere que sus clientes puedan realizar compras desde sus casas. El junto a su esposa tienen una cantidad de domiciliarios conocidos que se encargan de llevar los pedidos a los clientes.

A continuación, se muestra la conversación que se tuvo con don pepe:

- ¡Veee mijo! yo quiero que más gente me compre los producticos, cuando llega un vecino nuevo a la cuadra yo lo apunto en un cuadernito. ¿Entiendo don pepe, y no le gustaría que le comprarán por internet?
- Eh hh mijo pues no es mala idea y que hago con mi clientela?
- Pues don pepe hacemos un video tutorial para usar la aplicación, y le pedimos una información a sus clientes indicando sus datos personales (ID, cedula, Nombre, Dirección, Teléfono, email y password) a través de un formulario de registro. Una vez registrado podrá acceder a la realización de pedidos con su email y su password.
- ¡ eeeee yo no te creo! Asi de fácil? ¿Como motilando calvos?
- Don pepe ojala fuera así de sencillo déjeme le cuento mejor, Los productos que oferta el supermercado deben estar divididos en diversas categorías. Los datos necesarios para cada categoría son: nombre de la categoría, condiciones de almacenamiento (frío, congelado, seco) y observaciones. También debemos detallar la información de los productos (nombre, marca, origen, dimensiones (volumen y peso), una fotografía, la categoría y unidades disponibles). ¡no mijo eso me va salir muy caro con tanto detalle!
- don pepe todo lo contrario va aumentar mucho sus ganancias Espéreme le cuento algo mas, la aplicación permitirá visualizar un listado de productos ordenado por

categoría, permitiendo seleccionar los productos que desee comprar mediante una caja de texto donde se indicará el número de unidades seleccionadas. La aplicación llevará la cuenta (cesta de la compra) de los productos que el cliente ha ido seleccionando. La aplicación permitirá también efectuar un pedido con todos los productos que lleve almacenados en su cesta de la compra. Los datos del pedido son: 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)®.

Para poder generar un pedido se deberán dar dos situaciones:

- 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.
- Debe haber unidades suficientes por cada producto para satisfacer las demandas de cada pedido.

Una vez generado el pedido se mostrará al usuario una página con los datos de su pedido, se restarán del stock las unidades pedidas y se emitirá una nota de entrega a los responsables de almacén para que sirvan ese pedido.

Paso a paso de la solución

Paso 1: Lectura del contexto y reconocimiento de las entidades y sus atributos según la historia de usuario narrada

Cliente

Inicialmente se identifica la entidad **Cliente** cuyos atributos serán:

- Un **identificador único** de cada cliente en el momento en el que se registra
- **Nombre** que se va a componer de un **Nombre** y un **Apellido**
- **Cedula Cliente**
- **Dirección** que se va a componer de un **barrio** y un **numero de casa**
- Un **teléfono** como atributo multivaluado por si se desean ingresar múltiples teléfonos de un mismo usuario
- Por último como credenciales de acceso un **Email** y **Password**

Domiciliario

Se identificó la entidad **domiciliario** al cual le será asignado un pedido según el código postal al que pertenece, se reconocieron los siguientes atributos

- **Nombre** que estará compuesto por **Nombre y Apellido** ,
- **Identificador único** que le asignará la aplicación al momento de su registro
- Por último, un **numero de matricula**

Proveedor

Se identificó la entidad **proveedor** quien será la encargada de proveer los productos de la **Tienda Don Pepe**, se reconocieron los siguientes atributos.

- **Identificador único de proveedor**
- **Nombre del proveedor** (bien puede ser el nombre de la empresa o razón social)
- **Teléfono proveedor**

Producto

Se identificó la entidad **producto**, la cual tendrá la información de los productos que ofrece la **tienda Don Pepe**, se reconocieron los siguientes atributos.

- **Fotografía** para identificar el producto visualmente
- **Unidades disponibles:** Esta es la cantidad de unidades que se encuentra en el stock
- **Observaciones:** Para dar información que se considere relevante como la Fecha de caducidad
- **Dimensiones** que se compone del **volumen** y el **peso** del producto
- **Marca:** La marca del producto en cuestión
- **Nombre:** El nombre exacto del producto
- **Origen:** el lugar de procedencia, es decir donde fue fabricado
- **Precio:** Indica el precio del producto

Categoría del producto

Se identificó la entidad **categoría producto**, la cual contiene el **tipo de almacenamiento**, y una **descripción del almacenamiento**, se reconocieron los siguientes atributos.

- **Tipo de almacenamiento como una conexión a la tabla Tipo Almacenamiento:** este contiene la llave foránea del tipo de almacenamiento, bien sea Frio, a temperatura ambiente, seco, congelado , etc.
- **Descripción:** La descripción más detallada de cómo debe ser su almacenamiento
- **Identificador único categoría producto**

Tipo de almacenamiento producto

- **Id Almacenamiento:** Este es el Id del tipo de almacenamiento que requiere el producto
- **Tipo de almacenamiento:** Este es el tipo de almacenamiento

Aclaración: Esto se hace por si se desea agregar un tipo de almacenamiento más adelante y pensando también en el momento en que se van a agregar el tipo de almacenamiento

Pedido

Se identificó la entidad **Pedido**, la cual va a contener información importante de cada compra, la cual contiene los siguientes atributos.

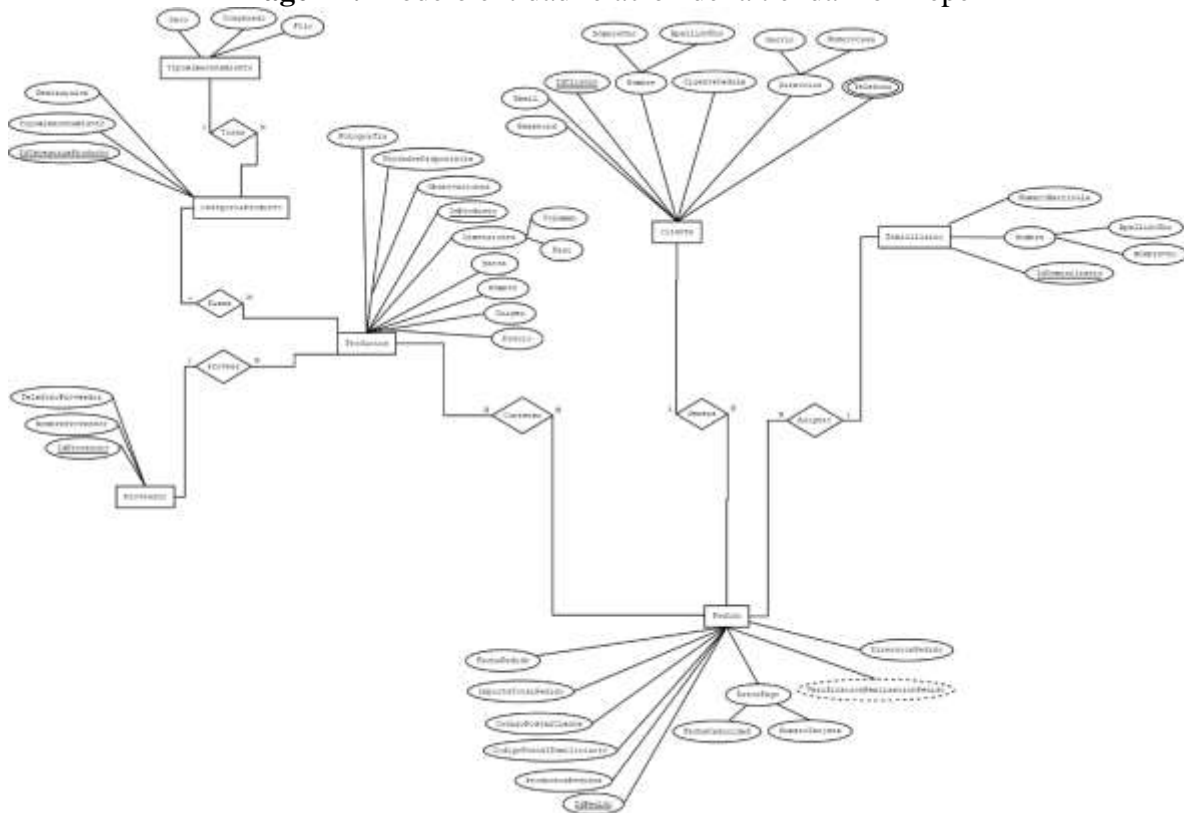
- **Fecha de pedido:** Fecha en la cual se realizan los pedidos
- **Importe total pedido:** Este es el valor total del pedido
- **Productos pedidos:** Estos son los productos pedidos
- **Datos de pago:** Está compuesto por **Fecha de caducidad** y **Numero de tarjeta**
- **Código Postal Cliente:** Código postal del cliente para realizar la verificación
- **Código Postal Domiciliario:** Código postal del domiciliario para realizar la verificación
- **Verificación Realización Pedido:** Este atributo tiene como valor un True o un False como resultado de una comparación entre El Código Postal Cliente y Código Postal Domiciliario, si son iguales retorna un True y se realiza el pedido y si retorna un False se cancela el pedido. (Esta lógica no fue implementada).
- **Dirección del pedido**

Paso 2: Relaciones entre entidades

Ahora realizamos las relaciones entre las entidades teniendo en cuenta el orden de los procedimientos.

- Un Proveedor **proveer** muchos productos, mientras que un producto puede ser provisto por un solo proveedor. Relación (1,N).
- Un producto **tiene** una categoría, mientras que una categoría puede tener muchos productos. Relación (1,N).
- Un tipo de almacenamiento puede **tener** muchas categorías del Producto, mientras que una categoría del producto puede tener un solo Tipo de almacenamiento. Relación (1,N).
- Un Producto puede ser **contenido** en muchos pedidos, mientras que un pedido puede contener muchos productos. Relación (1,N).
- Un cliente puede **generar** muchos pedidos, mientras que un pedido puede ser generado por un único cliente. (Si tenemos en cuenta que un pedido le va a pertenecer a un único cliente por cada proceso). Relación (1,N).
- Un domiciliario puede ser **asignado** a un muchos pedidos , mientras que un pedido solo se puede asignar a un único domiciliario (1,N)

Imagen 1: Modelo entidad relación de la tienda Don Pepe

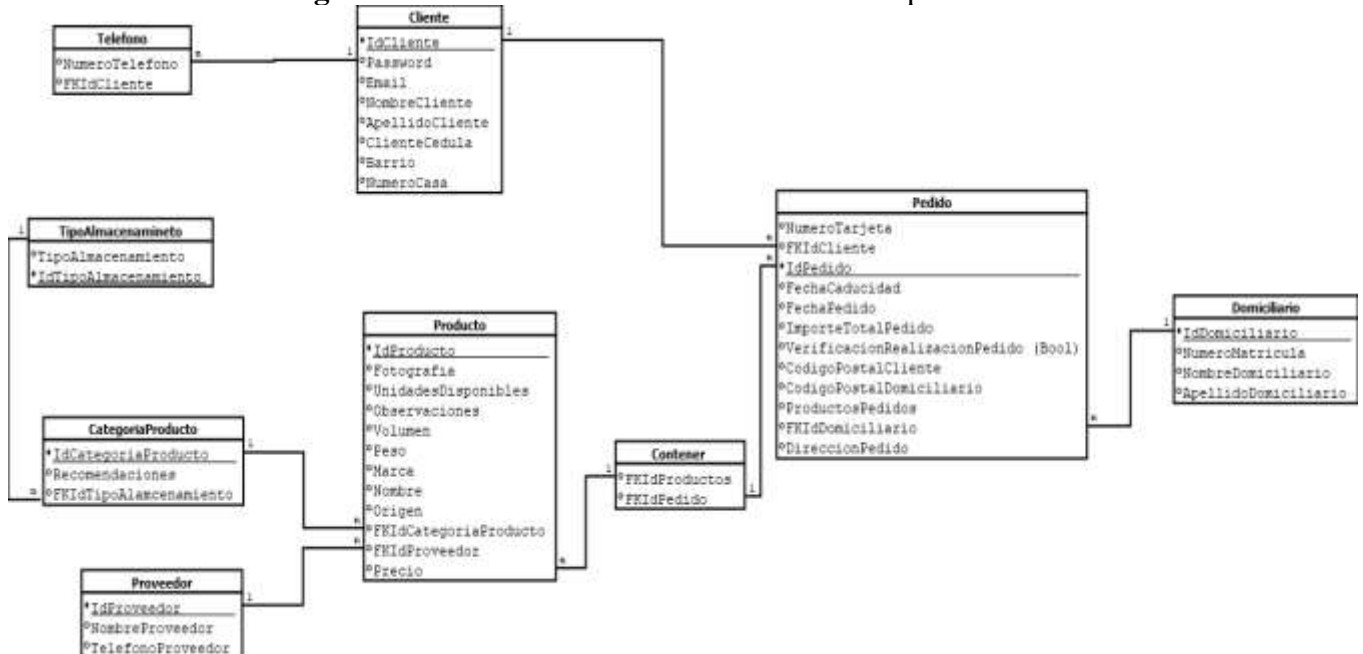


Paso 3: Creación del modelo relacional

- Ahora se crean las tablas en el modelo relacional, y se asignan las llaves foráneas en las tablas que tienen una cardinalidad (1,N), asignando la llave primaria de la entidad cuyo Cardinalidad es 1, como llave foránea a la tabla cuya cardinalidad es N.

- Después se crea la tabla para la relación (N,M) que va a contener las llaves primarias de las dos entidades que se relacionan , en este caso específico de las tablas Producto y pedido

Imagen 2: Modelo Relacional de la tienda Don Pepe



Normalización

La base de datos de la tienda Don Pepe cumple con la normalización de bases de datos ya que cada tabla cuenta con una clave primaria única y las relaciones entre ellas establecidas a través de claves foráneas. Además, las tablas cumplen con las siguientes formas normales:

- Primera forma normal (1FN): Cada columna contiene un solo valor, evitando la repetición de información.
- Segunda forma normal (2FN): Todos los atributos dependen completamente de la clave primaria, no hay dependencias parciales.
- Tercera forma normal (3FN): No hay dependencias transitivas, es decir, no existen atributos que dependan de otros que no sean clave primaria.

Esto permite que la información se encuentre organizada, sin duplicidades y de manera coherente en la base de datos. Además, se facilita la actualización, inserción y eliminación de datos, y se mejora el rendimiento y la seguridad del sistema

Paso 4

Para este paso se crea el código SQL que genere la base de datos anterior

```
CREATE DATABASE IF NOT EXISTS TiendaDonPepeSanty;
USE TiendaDonPepeSanty;

-- Tabla cliente
CREATE TABLE cliente (
    IdCliente VARCHAR(100) PRIMARY KEY,
    PasswordCliente VARCHAR(100),
    Email VARCHAR(100),
    NombreCliente VARCHAR(100),
    ApellidoCliente VARCHAR(100),
    ClienteCedula VARCHAR(100),
    Barrio VARCHAR(100),
    NumeroCasa VARCHAR(100)
);

-- CAMBIAR TIPO A ENTERO
-- Tabla TelefonoCLIENTE
CREATE TABLE TelefonoCliente (
    IdTelefononoCliente VARCHAR (100) PRIMARY KEY,
    FKIdCliente VARCHAR(100),
    NumeroTelefono INT (100) ,
    CONSTRAINT FKIdCliente FOREIGN KEY (FKIdCliente)
        REFERENCES Cliente (IdCliente)
);

-- Tabla Domiciliario
CREATE TABLE Domiciliario (
    IdDomiciliario VARCHAR(100) PRIMARY KEY,
    NumeroMatricula VARCHAR(100),
    NombreDomiciliario VARCHAR(100),
    ApellidoDomiciliario VARCHAR(100)
);

-- Tabla Proveedor
CREATE TABLE Proveedor (
    IdProveedor VARCHAR(100) PRIMARY KEY,
    NombreProveedor VARCHAR(100),
    TelefonoProveedor VARCHAR(100)
);

-- Tabla Tipo Almacenamiento
CREATE TABLE TipoAlmacenamiento (
    IdTipoAlmacenamiento VARCHAR(100) PRIMARY KEY,
    TipoDeAlmacenamiento VARCHAR(100)
);

-- Tabla Categoria Producto
CREATE TABLE CategoriaProducto (
    IdCategoriaProducto VARCHAR(100) PRIMARY KEY,
    FKIdTipoAlmacenamiento VARCHAR(100),
```

```

    Recomendaciones VARCHAR(500),
    CONSTRAINT FKCategoriaProductoTipoAlmacenamiento FOREIGN KEY
(FKIdTipoAlmacenamiento)
    REFERENCES TipoAlmacenamiento (IdTipoAlmacenamiento)
);

```

-- Tabla Producto

```

CREATE TABLE Producto (
    IdProducto VARCHAR(100) PRIMARY KEY,
    FKIdCategoriaProducto VARCHAR(100),
    FKIdProveedor VARCHAR(100),
    Fotografia VARCHAR(100),
    UnidadesDisponibles INT,
    Observaciones VARCHAR(100),
    Volumen VARCHAR(100),
    Peso VARCHAR(100),
    Nombre VARCHAR(100),
    PrecioProductoUnidad VARCHAR(100),
    CONSTRAINT FKProductoCategoriaProducto FOREIGN KEY
(FKIdCategoriaProducto)
    REFERENCES CategoriaProducto (IdCategoriaProducto),
    CONSTRAINT FKProductoProveedor FOREIGN KEY (FKIdProveedor)
    REFERENCES Proveedor (IdProveedor)
);

```

-- Tabla Pedido

```

CREATE TABLE Pedido (
    IdPedido VARCHAR(100) PRIMARY KEY,
    FKIdCliente VARCHAR(100),
    FKIdDomiciliario VARCHAR(100),
    FechaPedido VARCHAR(100),
    ImporteTotalPedido VARCHAR(100),
    VerificacionRealizacionPedido VARCHAR(100),
    CodigoPostalCliente VARCHAR(100),
    CodigoPostalDomiciliario VARCHAR(100),
    TajetaCredito VARCHAR(100),
    FechaCaducidadTarjeta VARCHAR(100),
    DireccionPedido VARCHAR(100),
    UnidadesPedidas INT,
    CONSTRAINT FKPedidoCliente FOREIGN KEY (FKIdCliente)
    REFERENCES Cliente (IdCliente),
    CONSTRAINT FKPedidoDomiciliario FOREIGN KEY (FKIdDomiciliario)
    REFERENCES Domiciliario (IdDomiciliario)
);

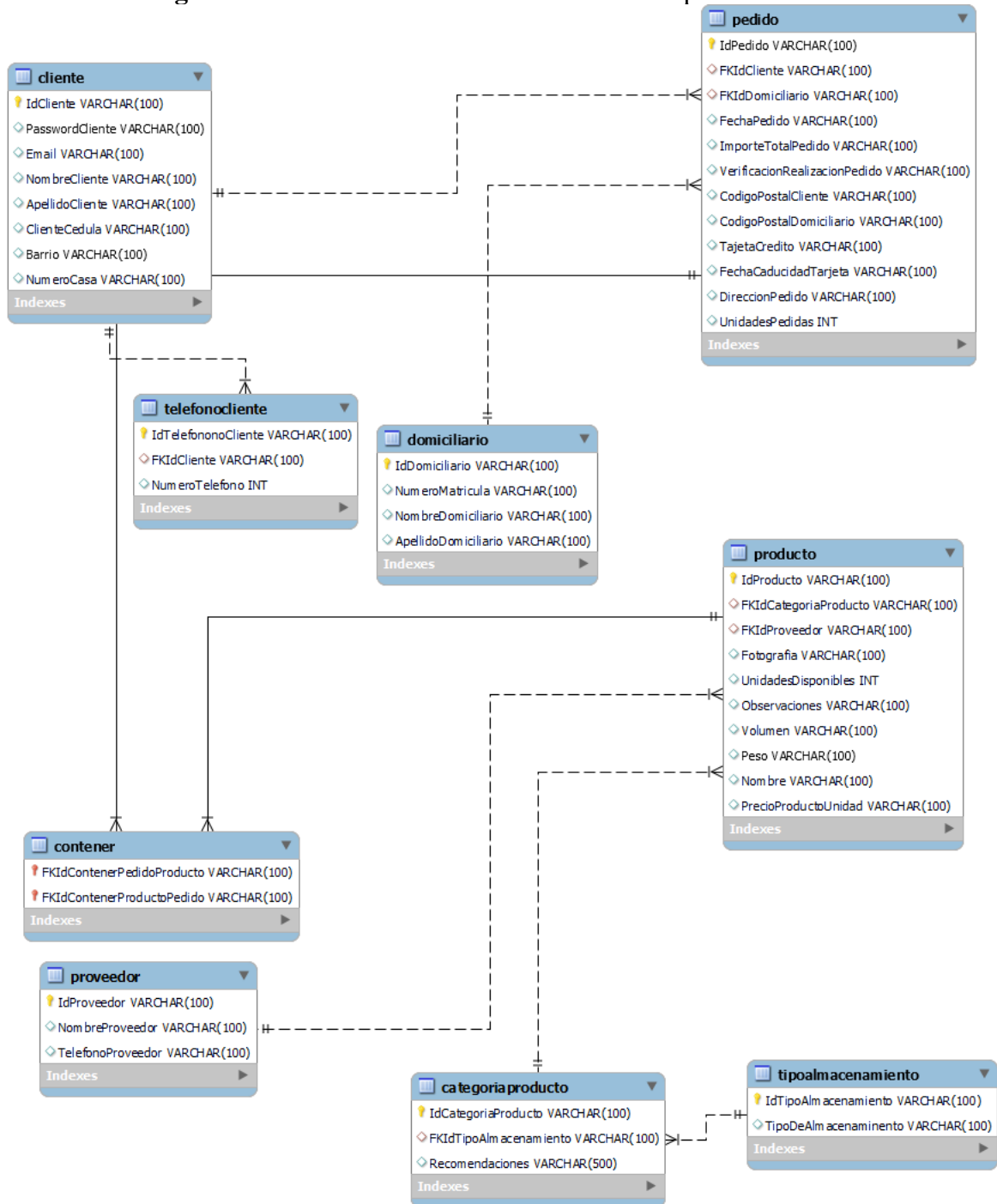
```

```
-- Tabla Contener
```

```
CREATE TABLE Contener (  
    FKIdContenerPedidoProducto VARCHAR(100),  
    FKIdContenerProductoPedido VARCHAR(100),  
    PRIMARY KEY (FKIdContenerPedidoProducto,FKIdContenerProductoPedido),  
    foreign key (FKIdContenerPedidoProducto) references Pedido (IdPedido),  
    foreign key (FKIdContenerProductoPedido) references Producto (IdProducto)  
);
```


Aplicando ingeniería inversa tenemos el siguiente diagrama.

Imagen 3: Modelo Relacional de la tienda Don Pepe en WorkBench



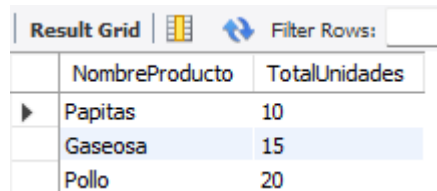
Consultas SQL

Para este ejemplo los datos fueron insertados manualmente

-- Mostrar la cantidad total de unidades disponibles de cada producto:

```
SELECT p.Nombre AS NombreProducto, SUM(p.UnidadesDisponibles) AS
TotalUnidades
FROM Producto p
GROUP BY p.Nombre;
```

Imagen 3: Mostrar la cantidad total de unidades disponibles de cada producto:

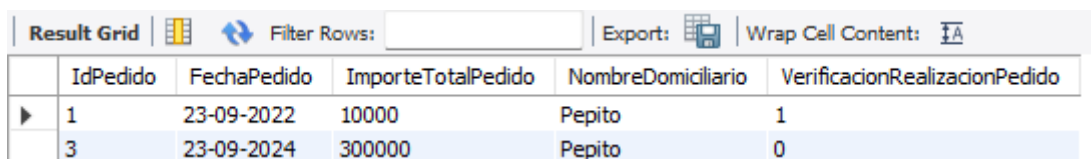


	NombreProducto	TotalUnidades
▶	Papitas	10
	Gaseosa	15
	Pollo	20

-- Mostrar todos los pedidos realizados por un cliente específico:

```
SELECT p.IdPedido, p.FechaPedido, p.ImporteTotalPedido,
d.NombreDomiciliario, p.VerificacionRealizacionPedido
FROM Pedido p
INNER JOIN Domiciliario d ON p.FKIdDomiciliario = d.IdDomiciliario
WHERE p.FKIdCliente = '1';
```

Imagen 4: Mostrar todos los pedidos realizados por un cliente específico

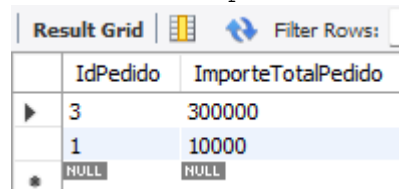


	IdPedido	FechaPedido	ImporteTotalPedido	NombreDomiciliario	VerificacionRealizacionPedido
▶	1	23-09-2022	10000	Pepito	1
	3	23-09-2024	300000	Pepito	0

----- Mostrar el número total de pedidos realizados por cada cliente:

```
SELECT c.NombreCliente, COUNT(p.IdPedido) AS TotalPedidos
FROM Cliente c
LEFT JOIN Pedido p ON c.IdCliente = p.FKIdCliente
GROUP BY c.NombreCliente;
```

Imagen 5: Mostrar el número total de pedidos realizados por cada cliente



	IdPedido	ImporteTotalPedido
▶	3	300000
	1	10000
*	NULL	NULL

-- Mostrar la lista de todos los domiciliarios con su número de matrícula y el número de pedidos que han entregado:

```
SELECT d.NombreDomiciliario, d.NumeroMatricula, COUNT(p.IdPedido) AS
TotalPedidos
FROM Domiciliario d
```

```
LEFT JOIN Pedido p ON d.IdDomiciliario = p.FKIdDomiciliario
GROUP BY d.NombreDomiciliario, d.NumeroMatricula;
```

Imagen 6: Mostrar la lista de todos los domiciliarios con su número de matrícula y el número de pedidos que han entregado

Result Grid	Filter Rows:	E
NombreDomiciliario	NumeroMatricula	TotalPedidos
Pepito	23-45	2
Jamit	29-23	1
Carlitos	032-02	0

```
-- Mostrar la cantidad total de productos suministrados por cada
proveedor:
```

```
SELECT pr.NombreProveedor, COUNT(p.IdProducto) AS TotalProductos
FROM Proveedor pr
LEFT JOIN Producto p ON pr.IdProveedor = p.FKIdProveedor
GROUP BY pr.NombreProveedor;
```

Imagen 7: Mostrar la cantidad total de productos suministrados por cada proveedor

Result Grid	Filter Rows:
NombreProveedor	TotalProductos
Saltina	2
Yupi	1

```
-- Mostrar el nombre y el apellido de los clientes que han realizado
algún pedido:
```

```
SELECT NombreCliente, ApellidoCliente
FROM Cliente
WHERE IdCliente IN (SELECT FKIdCliente FROM Pedido)
```




Imagen 8: Mostrar el nombre y el apellido de los clientes que han realizado algún pedido:

Result Grid	Filter Rows:
NombreCliente	ApellidoCliente
Santy	Tamirez
Pepe	Arenas

```
-- Mostrar la cantidad de pedidos realizados por cada domiciliario, ordenados de forma
descendente:
```

```
SELECT d.NombreDomiciliario, d.ApellidoDomiciliario, COUNT(*) AS
CantidadPedidos
FROM Domiciliario d
INNER JOIN Pedido pe ON d.IdDomiciliario = pe.FKIdDomiciliario
GROUP BY d.IdDomiciliario
ORDER BY CantidadPedidos DESC
```


Imagen 9: Mostrar la cantidad de pedidos realizados por cada domiciliario, ordenados de forma descendente

Result Grid   Filter Rows: <input type="text"/> Export: 			
	NombreDomiciliario	ApellidoDomiciliario	CantidadPedidos
▶	Pepito	Perez	2
	Jamit	Gonzales	1

-- Mostrar el nombre y el importe total de los pedidos realizados por el cliente con IdCliente = '1', ordenados por fecha de forma descendente:

```
SELECT pe.IdPedido, pe.ImporteTotalPedido
FROM Pedido pe
WHERE pe.FKIdCliente = '1'
ORDER BY pe.fechaPedido DESC;
```



Imagen 10: Mostrar el nombre y el importe total de los pedidos realizados por el cliente con IdCliente = '1', ordenados por fecha de forma descendente

Result Grid   Filter Rows: <input type="text"/>		
	IdPedido	ImporteTotalPedido
▶	3	300000
	1	10000
•	NULL	NULL

-- Mostrar el nombre y el teléfono del proveedor del producto con IdProducto = '1':

```
SELECT p.NombreProveedor, p.TelefonoProveedor
FROM Proveedor p
INNER JOIN Producto pr ON p.IdProveedor = pr.FKIdProveedor
WHERE pr.IdProducto = '1'
```

Imagen 11: Mostrar el nombre y el teléfono del proveedor del producto con IdProducto = '1':

Result Grid   Filter Rows: <input type="text"/>		
	NombreProveedor	TelefonoProveedor
▶	Saltina	1234

Vistas importantes

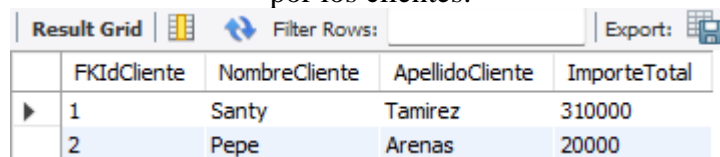
- **Primer vista:** Esta vista es importante para tener un historial de compra que tiene cada cliente y ofrecer posibles rebajas según la

inversión de los clientes en el negocio(Según la fidelidad del cliente)

-- -- Vista que muestra Vista que muestra el nombre y el importe total de todos los pedidos realizados por los clientes:

```
CREATE VIEW ImporteTotalPedidos AS
SELECT pe.FKIdCliente, c.NombreCliente, c.ApellidoCliente,
SUM(pe.ImporteTotalPedido) AS ImporteTotal
FROM Pedido pe
INNER JOIN Cliente c ON pe.FKIdCliente = c.IdCliente
GROUP BY pe.FKIdCliente, c.NombreCliente, c.ApellidoCliente
```

Imagen 12: Vista que muestra el nombre y el importe total de todos los pedidos realizados por los clientes:

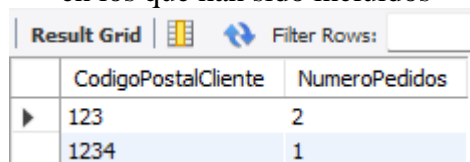


	FKIdCliente	NombreCliente	ApellidoCliente	ImporteTotal
▶	1	Santy	Tamirez	310000
	2	Pepe	Arenas	20000

- **Segunda vista:** Vista que nos muestra el total de pedidos por el código Postal, esto nos puede ayudar a saber que tantos pedidos realizan los clientes de una determinada área de la ciudad, y dar posibles rebajas o planear mejores rutas de acceso desde la tienda hasta el área en la que se piden los pedidos

```
-- Vista que nos muestra el total de pedidos por el codigo Postal
CREATE VIEW PedidosPorCodigoPostal AS
SELECT pe.CodigoPostalCliente, COUNT(pe.IdPedido) AS NumeroPedidos
FROM Pedido pe
GROUP BY pe.CodigoPostalCliente;
```

Imagen 13: Vista que muestra el nombre de los productos y el número de pedidos en los que han sido incluidos





	CodigoPostalCliente	NumeroPedidos
▶	123	2
	1234	1

Tercera vista: Esta vista nos puede ayudar que tan activos son los domiciliarios, e incluso podrían darse incentivos a los domiciliarios más activos, (Podría compararse con la opción que tiene Uber de total de viajes realizados por cada conductor)

```
-- Vista que muestra el numero total de pedidos por domiciliario:
CREATE VIEW PedidosPorDomiciliario AS
SELECT pe.FKIdDomiciliario, COUNT(pe.IdPedido) AS NumeroPedidos
```

```
FROM Pedido pe
GROUP BY pe.FKIdDomiciliario
```

Imagen 14: Vista que muestra el numero total de pedidos por domiciliario:

Result Grid  Filter Rows: 		
	FKIdDomiciliario	NumeroPedidos
▶	1	3
	2	2

TRIGGERS

- Trigger para descontar el stock

En este caso, el trigger se llama **"descontar_unidades"** y se activa después de que se inserta una nueva fila en la tabla "contener". El trigger se ejecuta una vez por cada fila insertada (gracias al uso de "FOR EACH ROW").

El cuerpo del trigger es una instrucción UPDATE que cambia los valores en la tabla "Producto". La instrucción UPDATE utiliza las tablas "Producto", "contener" y "Pedido" para determinar qué valores cambiar.

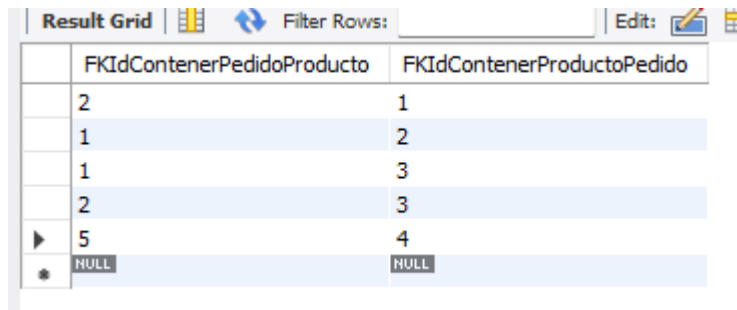
La cláusula JOIN se utiliza para combinar varias tablas en una sola consulta. En este caso, hay tres tablas: "Producto", "contener" y "Pedido". La primera JOIN combina la tabla "Producto" con la tabla "contener" utilizando el campo **"IdProducto"** de la primera tabla y **"FKIdContenerProductoPedido"** de la segunda tabla. La segunda JOIN combina la tabla "contener" con la tabla "Pedido" utilizando el campo **"FKIdContenerPedidoProducto"** de la primera tabla y **"IdPedido"** de la segunda tabla.

La cláusula SET actualiza el valor de **"UnidadesDisponibles"** en la tabla "Producto". Resta el valor de **"UnidadesPedidas"** de la tabla "Pedido" al valor actual de **"UnidadesDisponibles"** en la tabla "Producto".

Finalmente, la cláusula WHERE filtra las filas de la tabla "contener" para incluir solo las que acaban de ser insertadas (utilizando la variable NEW) y actualiza solo los valores en la tabla "Producto" que corresponden a esas filas.

```
DELIMITER //
CREATE TRIGGER descontar_unidades
AFTER INSERT ON contener
FOR EACH ROW
UPDATE Producto p
JOIN contener c ON p.IdProducto = c.FKIdContenerProductoPedido
JOIN Pedido ped ON ped.IdPedido = c.FKIdContenerPedidoProducto
SET p.UnidadesDisponibles = p.UnidadesDisponibles - ped.UnidadesPedidas
WHERE c.FKIdContenerProductoPedido = NEW.FKIdContenerProductoPedido AND
c.FKIdContenerPedidoProducto = NEW.FKIdContenerPedidoProducto;
//DELIMITER ;
```

Imagen 15: Visualización de como se realiza la reducción del stock



	FKIdContenerPedidoProducto	FKIdContenerProductoPedido
	2	1
	1	2
	1	3
	2	3
▶	5	4
*	NULL	NULL

Explicación del funcionamiento: Ya que la entidad Producto tiene un atributo llamado Unidades disponibles y la entidad Pedido tiene un atributo llamado unidades pedido, en la tabla intermediaria que salió como resultado de la relación M,N entre ambas entidades se le indica qué tipo de producto se desea elegir con el ID y el número de unidades pedidas que se obtienen desde la entidad pedido. Después de ingresar esta opción se restan las unidades pedidas a las unidades disponibles que están en la tabla producto

- Trigger de notificación de creación de pedido
-

Primero creamos una tabla para guardar la información de ID de la notificación, un mensaje para mostrar el ID del pedido que fue creado y finalmente una fecha de creación, así se tendrá un registro para el personal encargado de la gestión de los pedidos

```
CREATE TABLE Notificacion (  
    IdNotificacion AUTO_INCREMENT PRIMARY KEY,  
    Mensaje VARCHAR(100),  
    Fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);  
  
DELIMITER //  
CREATE TRIGGER tr_notificacion_nuevo_pedido  
AFTER INSERT ON Pedido  
FOR EACH ROW  
BEGIN  
    DECLARE mensaje VARCHAR(255);  
    SET mensaje = CONCAT('Se ha registrado un nuevo pedido con ID ',  
NEW.IdPedido);  
    INSERT INTO Notificacion (Mensaje) VALUES (mensaje);  
END  
//DELIMITER ;
```

Explicación del funcionamiento el trigger se llama **tr_notificacion_nuevo_pedido** y se ejecuta después de insertar un nuevo registro en la tabla **Pedido**.

La cláusula **FOR EACH ROW** indica que el trigger se ejecutará para cada registro insertado en la tabla. Dentro del trigger, se utiliza la sentencia **DECLARE** para crear una variable

llamada **mensaje** de tipo **VARCHAR(255)** que se utilizará para almacenar el mensaje de la notificación.

Luego se asigna a la variable **mensaje** el valor del mensaje que se desea almacenar, utilizando la función **CONCAT()** para concatenar una cadena de texto con el valor de la columna **IdPedido** del nuevo registro insertado. Finalmente, se utiliza la sentencia **INSERT INTO** para insertar el mensaje de la notificación en la tabla **Notificacion**.

Imagen 16: Trigger notificación creación de pedido

Result Grid	Filter Rows:	Edit:	Export/Imp
	IdNotificacion	Mensaje	Fecha
▶		Se ha registrado un nuevo pedido con ID 7	2023-02-19 06:34:33
✱	NULL	NULL	NULL

- Trigger para saber que usuario agregó y eliminó registros en la tabla Proveedor
Creamos la tabla para registrar los cambios

```
CREATE TABLE IF NOT EXISTS ControlCambiosProveedor (
  usuario VARCHAR(45) NOT NULL,
  accion VARCHAR(10) NOT NULL,
  fecha DATETIME NOT NULL,
  PRIMARY KEY (usuario, accion, fecha)
);
```

```
-- Trigger para saber quien ingreso nuevos Proveedores
DELIMITER //
CREATE TRIGGER TriggerIngresarProveedor
AFTER INSERT ON Proveedor
FOR EACH ROW
BEGIN
  INSERT INTO ControlCambiosProveedor (usuario, accion,
fecha)
  VALUES (USER(), "Agrego", NOW());
END//
DELIMITER ;
```

```
-- Trigger para saber quien elimino datos en la tabla de Proveedores

DELIMITER //
CREATE TRIGGER TriggerEliminarProveedor
AFTER DELETE ON Proveedor
FOR EACH ROW
BEGIN
  INSERT INTO ControlCambiosProveedor (usuario, accion,
fecha)
  VALUES (USER(), "Elimino", NOW());
```



```
END//
DELIMITER ;
```

Imagen 17: Trigger registro de usuario y fecha sobre modificaciones en la tabla proveedor

Result Grid			
Filter Rows:			
	usuario	accion	fecha
▶	root@localhost	Agrego	2023-02-19 07:14:37
	root@localhost	Elimino	2023-02-19 07:14:45
*	NULL	NULL	NULL

Procedimientos

Procedimiento para agregar un nuevo proveedor

```
-- Procedimiento para agregar Proveedor
DELIMITER //
CREATE PROCEDURE AgregarProveedor (IN idProveedorLocal VARCHAR(30),
IN NombreProveedorLocal VARCHAR(30), TelefonoProveedorLocal VARCHAR(30) )
BEGIN
    INSERT INTO Proveedor (IdProveedor, NombreProveedor, TelefonoProveedor)
VALUES
(idProveedorLocal, NombreProveedorLocal, TelefonoProveedorLocal);
END//
DELIMITER ;
CALL AgregarProveedor("10", "Agregar Santy", "1111");
```

Procedimiento para actualizar un proveedor por ID

```
-- Procedimiento actualizar Proveedor
DELIMITER //
CREATE PROCEDURE ActualizarProveedor (IN idProveedorLocal VARCHAR(30),
IN NombreProveedorLocal VARCHAR(30), TelefonoProveedorLocal VARCHAR(30))
BEGIN
    UPDATE Proveedor
    SET NombreProveedor = NombreProveedorLocal,
TelefonoProveedor = TelefonoProveedorLocal

    WHERE IdProveedor = idProveedorLocal ;
END//
DELIMITER ;
CALL ActualizarProveedor('1', 'atualizar', '02');
```

Procedimiento para borrar un proveedor por ID

```
DELIMITER //
CREATE PROCEDURE BorrarProveedor (IN idProveedorLocal VARCHAR(30))
BEGIN
    DELETE FROM Proveedor WHERE IdProveedor =
idProveedorLocal ;
END//
DELIMITER ;
CALL BorrarProveedor('10');
```

Procedimiento para consultar un proveedor por ID

```
-- Procedimiento consultar Proveedor por ID
DELIMITER //
CREATE PROCEDURE ConsultarProveedor (IN idProveedorLocal VARCHAR(30))
SELECT * FROM Proveedor WHERE IdProveedor = idProveedorLocal ;
END//
DELIMITER ;
CALL ConsultarProveedor('1');
```

Ingreso de registros automatizados

Población de la base de datos utilizando Java Faker, para esto voy a explicar como se ingresan los datos a la tabla Cliente, ya que el proceso es muy similar para las otras tablas, solo que teniendo en cuenta los atributos que contiene cada tabla en específico

```
/*String para inserción */
```

```
private static final String INSERT_CLIENTE = String.format("INSERT INTO
%s.Cliente VALUES (%s, %s, %s, %s, %s, %s, %s, %s)", DATA_BASE_NAME,
"Cliente%d", "%s", "%s", "%s", "%s", "%s", "%s", "%s");
```

```
/*Insercion de clientes*/
```

```
public static void insertClientes() {
    Faker faker = new Faker(new Locale("es"));
    for (int i = 1; i <= 50; i++) {
        int IdCliente = i;

        String PasswordCliente = faker.internet().password();
        String Email = faker.internet().emailAddress();
        String NombreCliente = faker.name().firstName();
        String ApellidoCliente = faker.name().lastName();
        String ClienteCedula = faker.idNumber().valid();
        String Barrio = faker.address().cityName();
        String NumeroCasa =
Integer.toString(faker.number().numberBetween(1, 1000));
        String query = String.format(INSERT_CLIENTE, IdCliente,
PasswordCliente, Email, NombreCliente, ApellidoCliente, ClienteCedula,
Barrio, NumeroCasa);
        mySqlOperation.setSqlStatement(query);
        System.out.println(query);
        mySqlOperation.executeSqlStatementVoid();
    }
}
```

```
}
```

Explicación

El código representa una función llamada "insertClientes()" que inserta 50 registros de clientes aleatorios en una base de datos MySQL.

La función comienza creando una instancia de la clase Faker con localización en español, que se utilizará para generar datos aleatorios de clientes.

Luego, se realiza un bucle for que se ejecuta 50 veces, en cada iteración se genera un conjunto aleatorio de datos del cliente utilizando la instancia Faker, y se utiliza una consulta SQL predefinida (INSERT_CLIENTE) para insertar el registro del cliente en la base de datos.

La consulta INSERT_CLIENTE es una constante definida en el código y utiliza la función "String.format()" para reemplazar los valores de los parámetros en la consulta. Los parámetros de la consulta representan las columnas de la tabla "Cliente" en la base de datos y se definen como cadenas de texto con formato.

La función utiliza el objeto "mysqlOperation" para ejecutar la consulta INSERT y registrar los resultados de la ejecución. La función también imprime la consulta SQL que se ejecutará en la consola antes de la ejecución.

Nota: Este mismo diseño se hizo para cada una de las tablas

Al terminar el ejercicio responda ¿ Esté conforme con el resultado obtenido según el contexto o cree que hubiera obtenido un mejor resultado con una base de datos no relacional?

Realmente tengo muy poca experiencia en referencia a Bases de datos no relacionales, por lo que puedo dar una opinión solo basándome en la teoría que hay en internet bajo mi interpretación. Las bases de datos relacionales son una buena opción cuando se necesita un alto nivel de integridad y consistencia de los datos y se requiere un acceso estructurado y eficiente a los mismos. Por otro lado, las bases de datos no relacionales son una buena opción cuando se trabaja con grandes volúmenes de datos no estructurados y se necesita una mayor escalabilidad y flexibilidad en el manejo de los datos, por lo tanto una base de datos relacional para el ejercicio que estamos realizando me parece que es la indicada ya que necesitamos los datos estructurados para llevar un registro de los movimientos de la tienda Don Pepe, y realmente al ser un negocio pequeño no se espera que maneje demasiado datos