

23/2/23

Tienda virtual

Don Pepe



Katherine Bonilla
SOFKAU

1-Construcción del modelo entidad relación

En el análisis realizado al enunciado expuesto, se identifican las siguientes entidades:

- **Proveedor**
- **Producto**
- **Categoría**
- **Cliente**
- **Pedido**
- **Orden de entrega**
- **Domiciliario**
- **Zona Postal**

Las relaciones identificadas son binarias y permiten visualizar la cardinalidad entre las entidades, los verbos identificados para establecer las mismas son: Proveer, contener, solicitar, generar, entregar, repartir.

El diagrama presenta, las siguientes relaciones:

Entidad	cardinalidad	Relación	Entidad
Proveedor	1: N	proveer	producto
Producto	N:1	pertenece	Categoría
Cliente	1: N	solicita	Pedido
Pedido	N:M	Tener	Producto
Pedido	1:1	Genera	Orden de entrega
Orden de entrega	N:1	Entrega	Domiciliario
Cliente	1:1	tiene	Zona Postal
Domiciliario	1:1	reparte	Zona Postal

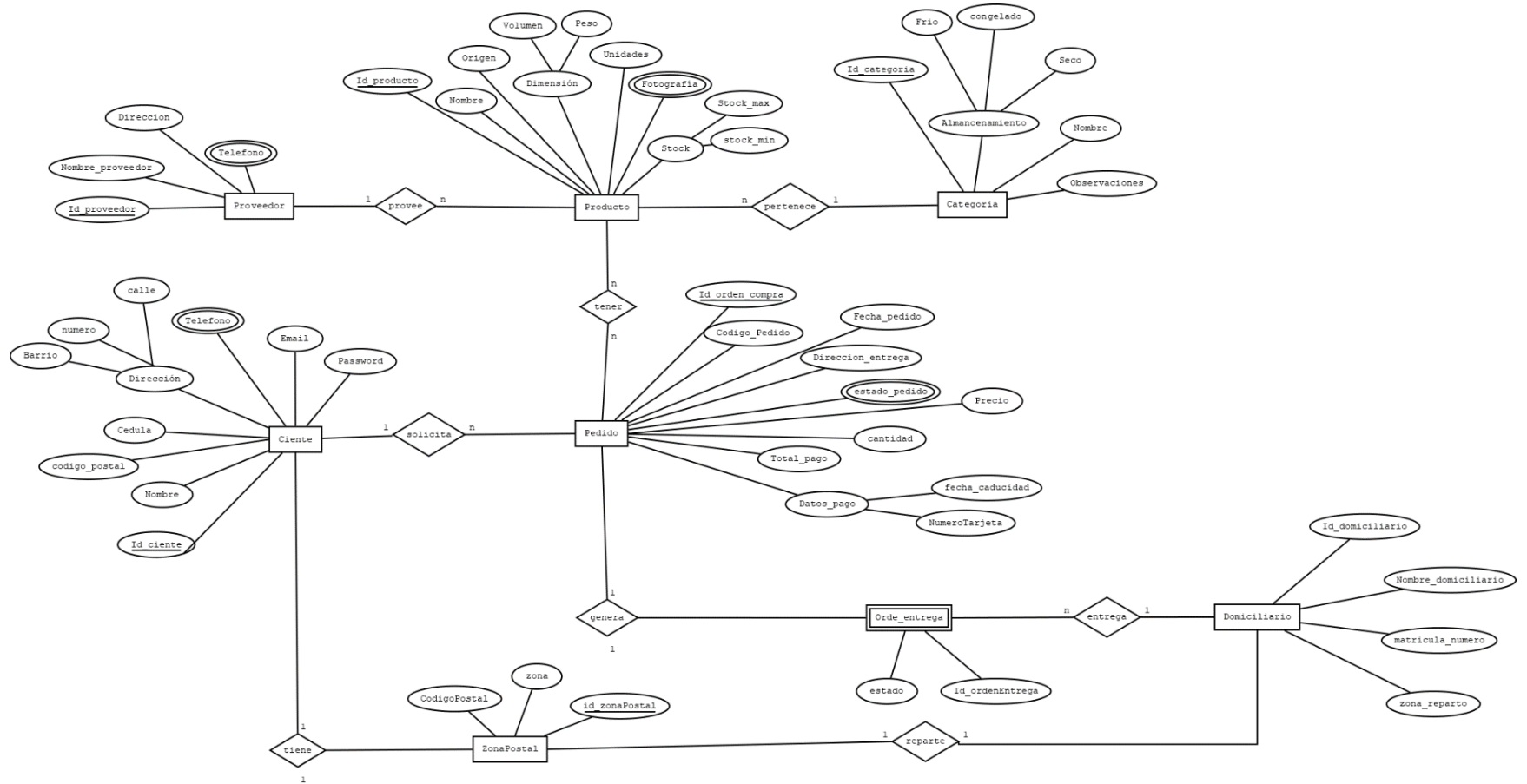
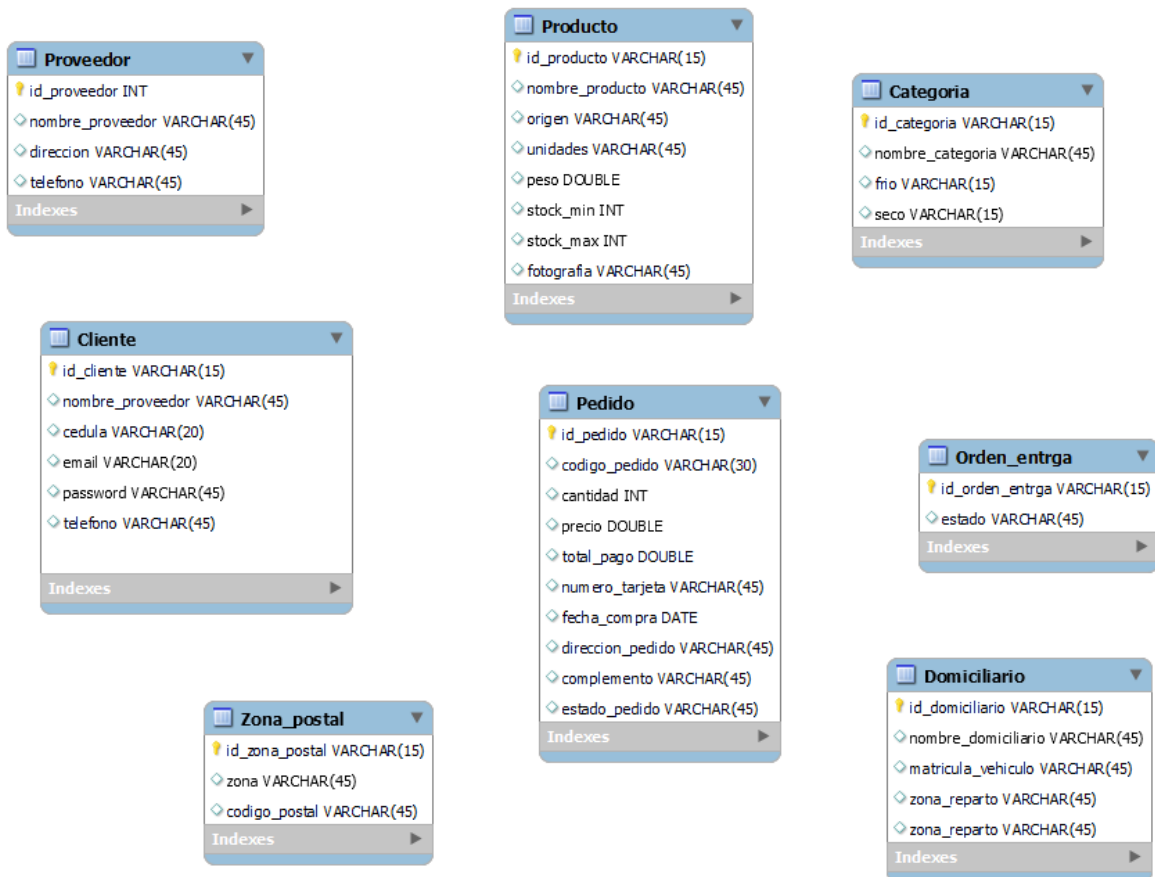


Figura.1 Diagrama Entidad-Relación

2- Modelo Relacional Workbench

Generado el diagrama entidad-relación con las relaciones expuestas y la cardinalidad, se procede a la transformación en un modelo relacional diseñándose a partir de sus atributos simples y multivalor las siguientes tablas:

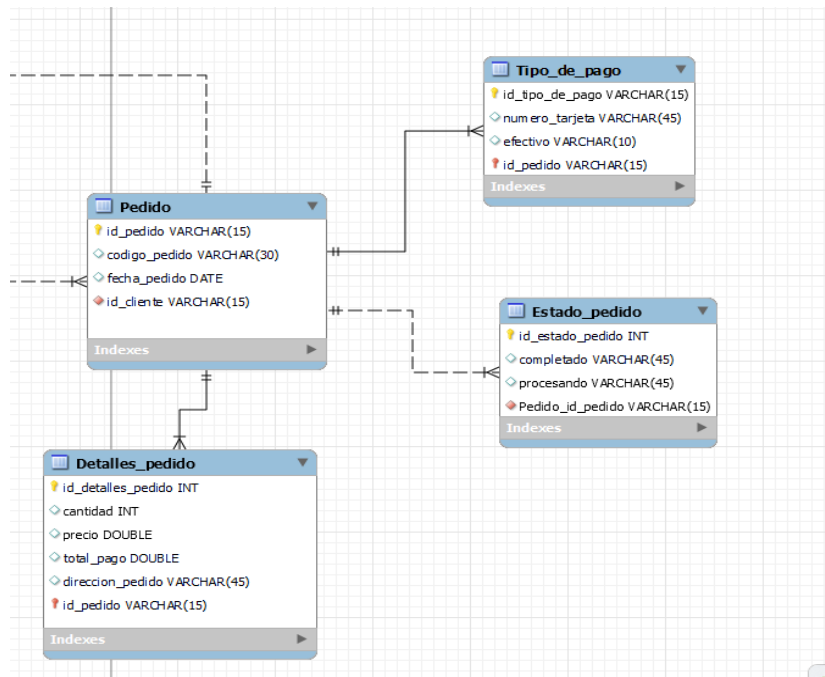


Normalización:

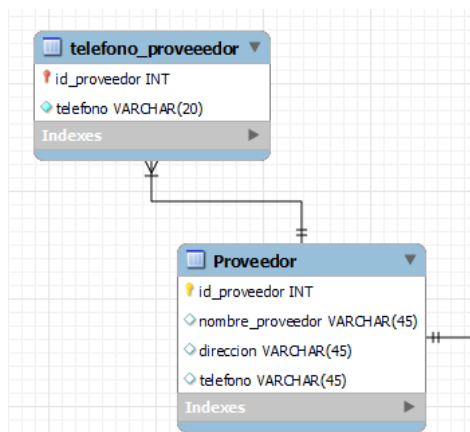
A las tablas, se les normaliza a continuación:

1-N - Aplicación de la primera forma normal, la cual determina evitar la redundancia de los datos e inconsistencias, en grupos de datos en este punto se toma la tabla **pedida** la cual, debe presentar de manera atómica los datos, considerando crear otra tabla que contenga los detalles de un pedido, que es de N:M

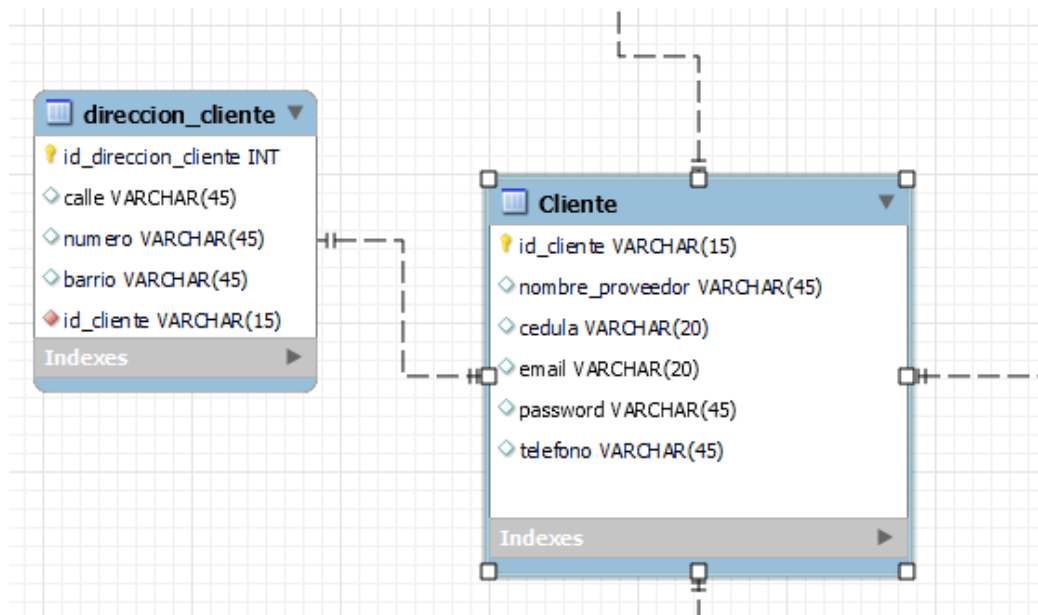
Ya que se encontraría agrupados en pedido de todos los clientes y sus detalles de compra, en este sentido se considera separar para optimizar el tipo de pago y los detalles de un pedido por cliente (N:M), como el estado que presente el pedido.



Por otra parte, se atomiza los datos al subdividir todos los datos multivalor en columnas separadas Ejemplo:



Así mismo, se toma los atributos como dirección que son compuestos para no tener redundancia de datos se crea una nueva tabla, Ejemplo:



2-N - Aplicación de la segunda forma normal, se verifica que todos los atributos dependan de una clave primaria, es decir la eliminación de las dependencias parciales, en nuestro caso al verificar el pedido se pudo establecer que ya este paso se había realizado, al agrupar los datos de pago y detalle de la compra en otra tabla.

3-N - Aplicación de la tercera forma normal, señala que hay que eliminar y separar cualquier dato que no sea clave. El valor de esta columna debe depender de la clave.

Al normalizar y crear las tablas y sus relaciones con las normalizaciones podemos ver el resultado de varias tablas derivadas que nos permite, cubrir la normalización de la base de datos de la Tienda Don Pepe:

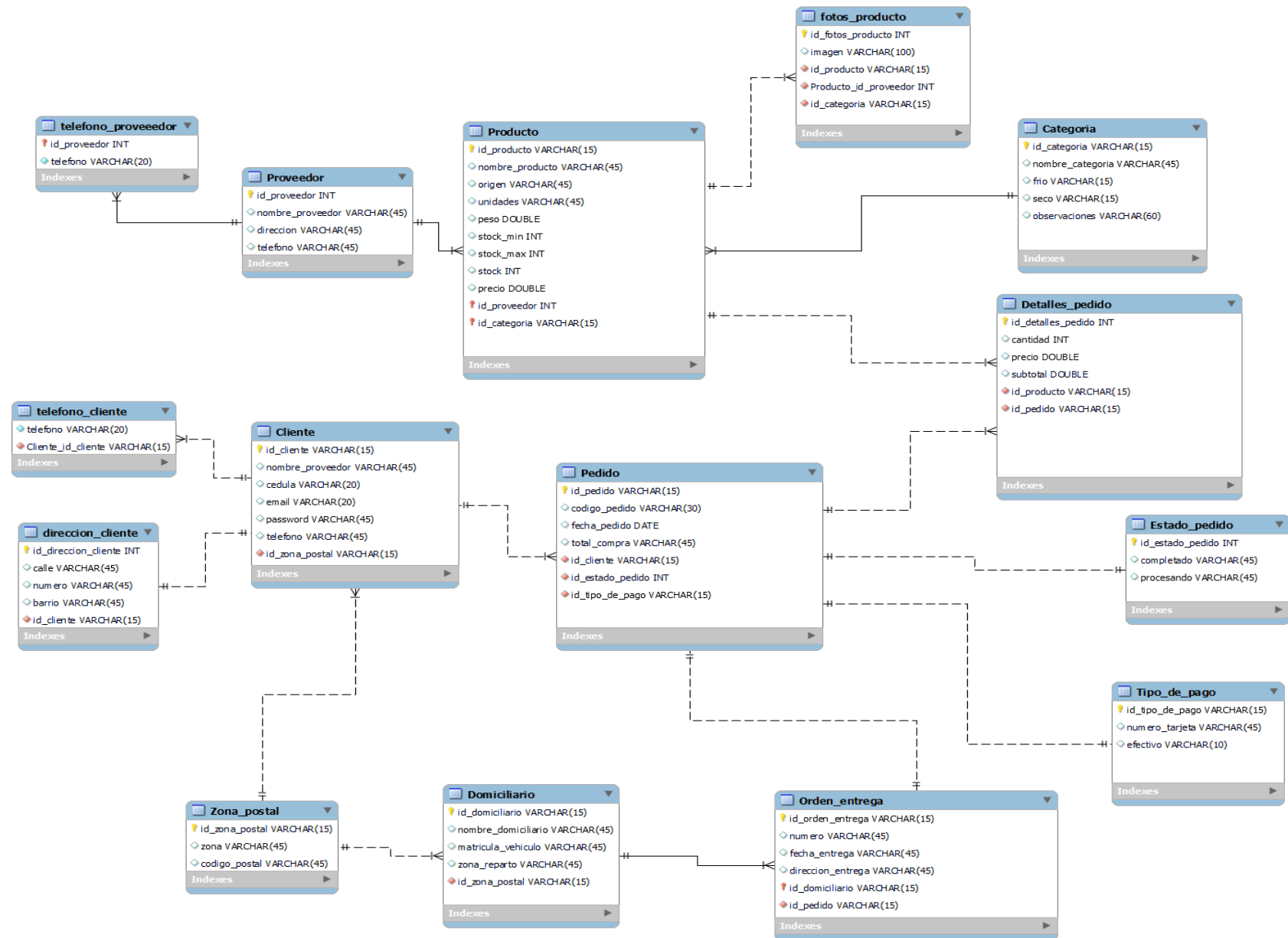


Figura.2 Modelo relacional

En el modelo, se evidencia que todos los atributos no-clave dependen por completo de la clave primaria. Esta dependencia se da si todos los atributos de la clave primaria son necesarios para identificar a los atributos no-clave, como también que las tablas con claves primarias simples se ajustan automáticamente a la 2FN si se cumplen las condiciones para la 1FN.

Por lo cual, se limita a los atributos no-clave que no dependen de la clave primaria a tablas diferentes, como en el caso de estado pedido y tipo de pago.

3- Creación de base de datos

Sentencias manuales:

-- Creación de base de datos Tienda Don Pepe, se crean tablas con llave primaria y foránea

CREATE DATABASE tienda_don_pepe;

USE tienda_don_pepe;

-- Creación tabla proveedor

```
CREATE TABLE Proveedor(  
  id_proveedor INT NOT NULL,  
  nombre_proveedor VARCHAR (45) NULL,  
  direccion VARCHAR (45) NULL,  
  telefono VARCHAR (45) NULL,  
  PRIMARY KEY (id_proveedor) );
```

-- Creación tabla Categoría

```
CREATE TABLE Categoria (  
  id_categoria VARCHAR (15) NOT NULL,  
  nombre_categoria VARCHAR (45) NULL,  
  almacenamiento VARCHAR (15) NULL,  
  observaciones VARCHAR (60) NULL,  
  PRIMARY KEY (id_categoria)  
);
```


-- Creación tabla producto

```
CREATE TABLE Producto (  
  id_producto VARCHAR (15) NOT NULL,  
  nombre_producto VARCHAR (45) NULL,  
  origen VARCHAR (45) NULL,  
  marca VARCHAR (45) NULL,  
  peso DOUBLE NULL,  
  stock INT NULL,  
  precio DOUBLE NULL,  
  id_proveedor INT NOT NULL,  
  id_categoria VARCHAR (15) NOT NULL,  
  PRIMARY KEY (id_producto, id_proveedor, id_categoria),  
  FOREIGN KEY (id_proveedor) REFERENCES Proveedor (id_proveedor),  
  FOREIGN KEY (id_categoria) REFERENCES Categoria (id_categoria)  
);
```

-- Creación tabla zona postal

```
CREATE TABLE Zona_postal (  
  id_zona_postal VARCHAR (15) NOT NULL,  
  zona VARCHAR (45) NULL,  
  codigo_postal VARCHAR (45) NULL,  
  PRIMARY KEY (id_zona_postal)  
);
```

-- Creación tabla tipo de pago

```
CREATE TABLE Tipo_de_pago (  
  id_tipo_de_pago VARCHAR (15) NOT NULL,  
  numero_tarjeta VARCHAR (45) NULL,  
  fecha_caducidad VARCHAR (45) NULL,  
  PRIMARY KEY (`id_tipo_de_pago`)  
);
```

-- Creación tabla estado de pedido

```
CREATE TABLE Estado_pedido (  
  id_estado_pedido INT NOT NULL,  
  estado VARCHAR (45) NULL,  
  dscripcion VARCHAR (60) NULL,  
  PRIMARY KEY (`id_estado_pedido`)
```

```
);
```

-- Creación tabla Pedido

```
CREATE TABLE Pedido (  
  id_pedido VARCHAR (15) NOT NULL,  
  codigo_pedido VARCHAR (30) NULL,  
  fecha_pedido DATE NULL,  
  total VARCHAR (45) NULL,  
  id_cliente VARCHAR (15) NOT NULL,  
  id_estado_pedido INT NOT NULL,  
  id_tipo_de_pago VARCHAR (15) NOT NULL,  
  PRIMARY KEY (id_pedido),  
  FOREIGN KEY (id_cliente) REFERENCES Cliente (id_cliente),  
  FOREIGN KEY (id_estado_pedido) REFERENCES Estado_pedido (id_estado_pedido),  
  FOREIGN KEY (id_tipo_de_pago) REFERENCES Tipo_de_pago (id_tipo_de_pago)  
);
```

-- Creación tabla orden entrega

```
CREATE TABLE Orden_entrega (  
  id_orden_entrega VARCHAR (15) NOT NULL,  
  numero VARCHAR (45) NULL,  
  fecha_entrega VARCHAR (45) NULL,  
  direccion_entrega VARCHAR (45) NULL,  
  id_domiciliario VARCHAR (15) NOT NULL,  
  id_pedido VARCHAR (15) NOT NULL,  
  PRIMARY KEY (id_orden_entrega, id_domiciliario),  
  FOREIGN KEY (id_domiciliario) REFERENCES Domiciliario (id_domiciliario),  
  FOREIGN KEY (id_pedido) REFERENCES Pedido (id_pedido)  
);
```

-- Creación tabla cliente

```
CREATE TABLE Cliente (  
  id_cliente VARCHAR (15) NOT NULL,  
  nombre_cliente VARCHAR (45) NULL,  
  cedula VARCHAR (20) NULL,  
  email VARCHAR (20) NULL,  
  password VARCHAR (45) NULL,  
  telefono VARCHAR (45) NULL,  
  id_zona_postal VARCHAR (15) NOT NULL,  
  PRIMARY KEY (id_cliente),  
  FOREIGN KEY (id_zona_postal) REFERENCES Zona_postal (id_zona_postal)
```

```
);
```

-- Creación tabla direccion cliente

```
CREATE TABLE direccion_cliente (  
    id_direccion_cliente INT NOT NULL,  
    calle VARCHAR (45) NULL,  
    numero VARCHAR (45) NULL,  
    barrio VARCHAR (45) NULL,  
    id_cliente VARCHAR (15) NOT NULL,  
    PRIMARY KEY (id_direccion_cliente, id_cliente),  
    FOREIGN KEY (id_cliente) REFERENCES Cliente (id_cliente)  
);
```

-- Creación tabla domiciliario

```
CREATE TABLE Domiciliario (  
    id_domiciliario VARCHAR (15) NOT NULL,  
    nombre_domiciliario VARCHAR (45) NULL,  
    matricula_vehiculo VARCHAR (45) NULL,  
    zona_reparto VARCHAR (45) NULL,  
    id_zona_postal VARCHAR (15) NOT NULL,  
    PRIMARY KEY (id_domiciliario, id_zona_postal),  
    FOREIGN KEY (id_zona_postal) REFERENCES Zona_postal (id_zona_postal)  
);
```

-- Creación tabla telefono proveedor

```
CREATE TABLE telefono_proveedor (  
    id_proveedor INT NOT NULL,  
    telefono VARCHAR (20) NOT NULL,  
    PRIMARY KEY (id_proveedor),  
    FOREIGN KEY (id_proveedor) REFERENCES Proveedor (id_proveedor)  
);
```

-- Creación tabla direccion cliente

```
CREATE TABLE telefono_cliente (  
    id_cliente VARCHAR (15) NOT NULL,  
    telefono VARCHAR (20) NOT NULL,  
    PRIMARY KEY (id_cliente),  
    FOREIGN KEY (id_cliente) REFERENCES Cliente (id_cliente)  
);
```

-- Creación tabla detalles de pedido

```
CREATE TABLE Detalles_pedido (  
    id_detalle_pedido INT NOT NULL,  
    cantidad INT NULL,  
    precio DOUBLE NULL,  
    id_producto VARCHAR (15) NOT NULL,  
    id_pedido VARCHAR (15) NOT NULL,  
    PRIMARY KEY (id_detalle_pedido),  
    FOREIGN KEY (id_producto) REFERENCES Producto (id_producto),  
    FOREIGN KEY (id_pedido) REFERENCES Producto (id_producto),  
    FOREIGN KEY (id_categoria) REFERENCES Producto (id_categoria)  
);
```

-- Creación tabla direccion_cliente

```
CREATE TABLE direccion_cliente (  
    id_direccion_cliente INT NOT NULL,  
    calle VARCHAR (45) NULL,  
    numero VARCHAR (45) NULL,  
    barrio VARCHAR (45) NULL,  
    id_cliente VARCHAR (15) NOT NULL,  
    PRIMARY KEY (id_direccion_cliente),  
    FOREIGN KEY (id_direccion_cliente) REFERENCES Cliente (id_cliente)  
);
```

-- Creación tabla fotos producto

```
CREATE TABLE fotos_producto (  
    id_fotos_producto INT NOT NULL,  
    imagen VARCHAR (100) NULL,  
    id_producto VARCHAR (15) NOT NULL,  
    id_proveedor INT NOT NULL,  
    id_categoria VARCHAR (15) NOT NULL,  
    PRIMARY KEY (id_fotos_producto),  
    FOREIGN KEY (id_producto , id_proveedor , id_categoria)  
        REFERENCES Producto (id_producto , id_proveedor , id_categoria) );
```

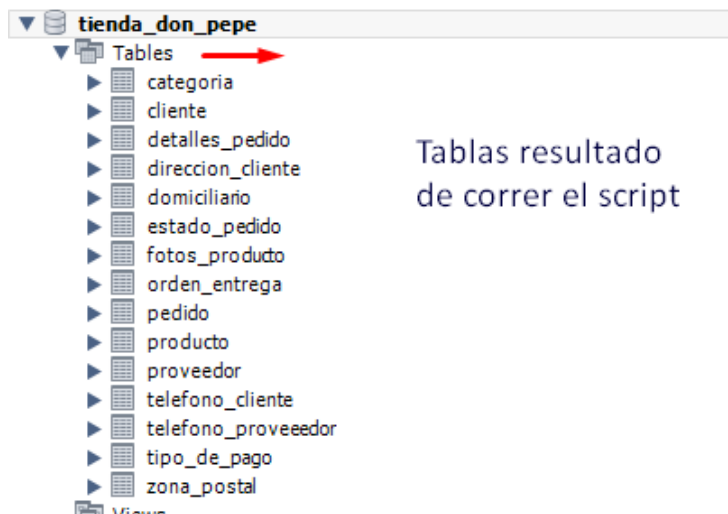


Figura.3 tablas creadas

4-Consultas de la información de cada tabla o de varias tablas

1- consulta: Cliente datos de contacto y su dirección:

```

2
3  -- consulta 1 cliente datos de contacto y su direccion
4  SELECT nombre_cliente, cedula, email, telefono, calle, numero, barrio
5  FROM cliente
6  INNER JOIN direccion_cliente
7  ON cliente.id_cliente = direccion_cliente.id_cliente;
8

```

nombre_cliente	cedula	email	telefono	calle	numero	barrio
Juan Perez	120455	Juan@gmail.com	31156897	45	5	Olarte
Saundra Turbat	660457	turba@gmail.com	3165600	8	36	La Libertad
Luna Carrillo	160457	Luna@gmail.com	3115600	12	56	Ciudad Jardín
Shaughn Sneddon	990457	Sneddon@gmail.com	3115665	102	74	Antonia Sa...
Ransell Filppov	500457	Filppov@gmail.com	3215614	80	1	Chapinero ...
Felipe Bigen	514478	felip@gmail.com	654789	53	15	Villa del Río
Ramiro Diaz	3478996	Diaz@gmail.com	31441064	50	32	Madelena

Result 1 x

Output

#	Time	Action	Message	
1	11:35:24	use tienda_don_pepe	0 row(s) affected	0
2	11:39:19	SELECT * FROM tienda_don_pepe.categoria LIMIT 0, 1000	7 row(s) returned	0
3	14:01:50	USE tienda_don_pepe	0 row(s) affected	0
4	14:01:56	SELECT nombre_cliente, cedula, email, telefono, calle, numero, barrio FROM cliente IN...	7 row(s) returned	1

2-Consulta: mostrar los productos y características que comiencen por la letra A

```
9
10 -- consulta 2 mostrar los productos y características que comiencen por la letra A
11 • SELECT *
12 FROM Producto
13 WHERE nombre_producto LIKE 'A%';
14
```

	id_producto	nombre_producto	origen	marca	peso	stock	precio	id_proveedor	id_categoria
▶	2	Arroz blanco	nacional	Diana	1kg	60	5390	301	50
	5	atún natural	nacional	Alamar	140gr	35	6990	302	60
	7	ajo	nacional	El Rey	60g	40	2490	303	70
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Producto 2 x

Output

Action Output

#	Time	Action	Message
✓ 1	14:04:34	SELECT * FROM Producto WHERE nombre_producto LIKE 'A%' LIMIT 0, 1000	3 row(s) returned

3-Consulta: agrupar el número de domiciliarios por zonas

```
13 WHERE nombre_producto LIKE 'A%';
14
15 -- Consulta 3 agrupar el numero de domiciliarios por zonas
16
17 • SELECT COUNT(nombre_domiciliario), zona_reparto
18 FROM domiciliario
19 GROUP BY domiciliario.zona_reparto;
20
```

	COUNT(nombre_domiciliario)	zona_reparto
▶	2	Barrios Unidos
	1	Bosa
	1	Ciudad Bolívar
	1	Antonio Nariño
	3	Chapinero
	1	Soacha

Result 4 x

Output

Action Output

#	Time	Action	Message
✓ 1	14:07:46	SELECT COUNT(nombre_domiciliario), zona_reparto FROM domiciliario GROUP BY domi...	6 row(s) returned

4-Consulta: listado de productos nombre y precio

The screenshot shows a database interface with a SQL editor and a results pane. The SQL editor contains the following queries:

```
19 GROUP BY domiciliario.zona_reparto;
20
21 -- consulta 4 listado de productos nombre y precio
22
23 • SELECT concat(nombre_producto, ' ', ' ', marca, + ' ', ' ', precio ) lista_producto
24 FROM producto;
25
26 -- Consulta 5 pedidos con sus fechas pedido, direccion de pedido y estatus
```

The results pane shows the output of the first query, which is a list of products and their prices:

lista_producto
Pernil de pollo , Bucanero , 13589
Arroz blanco , Diana , 5390
Shampoo , Savitel , 9442
Queso Mozzarella , Colanta , 7590
atún natural , Alamar , 6990
Mr Tea , Mr. Tea , 4543
ajo , El Rey , 2490

The results pane also shows the output of the second query, which is a list of products and their prices:

lista_producto
Pernil de pollo , Bucanero , 13589
Arroz blanco , Diana , 5390
Shampoo , Savitel , 9442
Queso Mozzarella , Colanta , 7590
atún natural , Alamar , 6990
Mr Tea , Mr. Tea , 4543
ajo , El Rey , 2490

5- Consulta: pedidos con sus fechas pedido, dirección de pedido y estatus

The screenshot shows a database interface with a SQL editor and a results pane. The SQL editor contains the following queries:

```
28 • SELECT fecha_pedido as 'fecha' , nombre_cliente, direccion_cliente.calle, direccion_cliente.numero
29 FROM pedido
30 INNER JOIN estado_pedido
31 ON estado_pedido.id_estado_pedido = pedido.id_estado_pedido
32 INNER JOIN cliente
33 ON pedido.id_cliente = cliente.id_cliente
34 INNER JOIN direccion_cliente
35 ON pedido.id_cliente = direccion_cliente.id_cliente;
36
37 -- Consulta 6 buscar cliente Juan Perez
```

The results pane shows the output of the first query, which is a list of orders with their dates, client names, addresses, and statuses:

fecha	nombre_cliente	calle	numero	barrio	estado
2023-02-11	Juan Perez	45	5	Olarte	solicitado
2023-02-12	Ramiro Diaz	50	32	Madelena	recibido
2023-01-31	Juan Perez	45	5	Olarte	recibido
2023-02-11	Luna Carrillo	12	56	Ciudad Jardín	recibido

The results pane also shows the output of the second query, which is a list of orders with their dates, client names, addresses, and statuses:

fecha	nombre_cliente	calle	numero	barrio	estado
2023-02-11	Juan Perez	45	5	Olarte	solicitado
2023-02-12	Ramiro Diaz	50	32	Madelena	recibido
2023-01-31	Juan Perez	45	5	Olarte	recibido
2023-02-11	Luna Carrillo	12	56	Ciudad Jardín	recibido

6-Consulta: buscar cliente Juan Pérez

```
37 -- Consulta 6 buscar cliente Juan Perez
38
39 • SELECT *
40 FROM cliente
41 WHERE nombre_cliente LIKE '%Juan Perez%';
42
43 -- consulta 7 producto cuyo precio es mayor a 8000 $
44 • SELECT *
45 FROM producto
```

Result Grid

	id_cliente	nombre_cliente	cedula	email	password	telefono	id_zona_postal
▶	100	Juan Perez	120455	Juan@gmail.com	1235a	31156897	3
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

cliente 7 x

Output

Action Output

#	Time	Action	Message
✓ 1	14:13:31	SELECT * FROM cliente WHERE nombre_cliente LIKE '%Juan Perez%' LIMIT 0, 1000	1 row(s) returned

7-Consulta: productos cuyo precio es mayor a 8000 \$

```
42
43 -- consulta 7 producto cuyo precio es mayor a 8000 $
44 • SELECT *
45 FROM producto
46 WHERE producto.precio > 8000;
47
48 -- consulta 8 mostrar los productos cuyo stock sea menor a 30 productos
```

Result Grid

	id_producto	nombre_producto	origen	marca	peso	stock	precio	id_proveedor	id_categoria
▶	1	Pernil de pollo	nacional	Bucanero	750g	26	13589	304	40
3		Shampoo	nacional	Savitel	350ml	30	9442	302	10
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

producto 8 x

Output

Action Output

#	Time	Action	Message
✓ 1	14:13:31	SELECT * FROM cliente WHERE nombre_cliente LIKE '%Juan Perez%' LIMIT 0, 1000	1 row(s) returned
✓ 2	14:14:31	SELECT * FROM producto WHERE producto.precio > 8000 LIMIT 0, 1000	2 row(s) returned

8- Consulta: mostrar los productos cuyo stock sea menor a 30 productos

```
49
50 • SELECT *
51 FROM producto
52 WHERE producto.stock < 30;
53
54
```

	id_producto	nombre_producto	origen	marca	peso	stock	precio	id_proveedor	id_categoria
▶	1	Pernil de pollo	nacional	Bucanero	750g	26	13589	304	40
	4	Queso Mozzarella	nacional	Colanta	250gr	22	7590	305	30
	6	Mr Tea	nacional	Mr. Tea	500ml	25	4543	302	20
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

producto 9 x

Output

Action Output

#	Time	Action	Message
✓ 1	14:13:31	SELECT * FROM cliente WHERE nombre_cliente LIKE "%Juan Perez%" LIMIT 0, 1000	1 row(s) returned
✓ 2	14:14:31	SELECT * FROM producto WHERE producto.precio > 8000 LIMIT 0, 1000	2 row(s) returned
✓ 3	14:16:15	SELECT * FROM producto WHERE producto.stock < 30 LIMIT 0, 1000	3 row(s) returned

9- consulta 9 Ordenar por stock mayor a 30 unidades y precio los productos de manera descendente

```
57
58 • SELECT *
59 FROM producto
60 WHERE stock < 30
61 ORDER BY precio DESC;
62
```

	id_producto	nombre_producto	origen	marca	peso	stock	precio	id_proveedor	id_categoria
▶	1	Pernil de pollo	nacional	Bucanero	750g	26	13589	304	40
	4	Queso Mozzarella	nacional	Colanta	250gr	22	7590	305	30
	6	Mr Tea	nacional	Mr. Tea	500ml	25	4543	302	20
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

producto 11 x

Output

Action Output

#	Time	Action	Message
✓ 1	14:17:30	SELECT * FROM producto WHERE stock < 30 ORDER BY precio DESC LIMIT 0, 1000	3 row(s) returned

10-Consulta: Ver los clientes y domiciliarios que se pueden encontrar en la misma zona

```
64 -- Consulta 10 ver los clientes y domiciliarios que se pueden encontrar en la misma zona
65 • SELECT cliente.nombre_cliente, domiciliario.nombre_domiciliario, zona_postal.zona
66 FROM cliente
67 INNER JOIN domiciliario
68 ON cliente.id_zona_postal = domiciliario.id_zona_postal
69 INNER JOIN zona_postal
70 ON cliente.id_zona_postal = zona_postal.id_zona_postal
71 GROUP BY cliente.id_zona_postal
72
```

nombre_cliente	nombre_domiciliario	zona
Saundra Turbat	Pedro Jimenez	Barrios Unidos
Juan Perez	Juan Lima	Bosa
Ransell Filippov	Ruben Mora	Chapinero
Ramiro Diaz	Liliana Amaro	Ciudad Bolivar
Luna Carrillo	Ana Tinto	Antonio Nariño

Result 35 x

Output

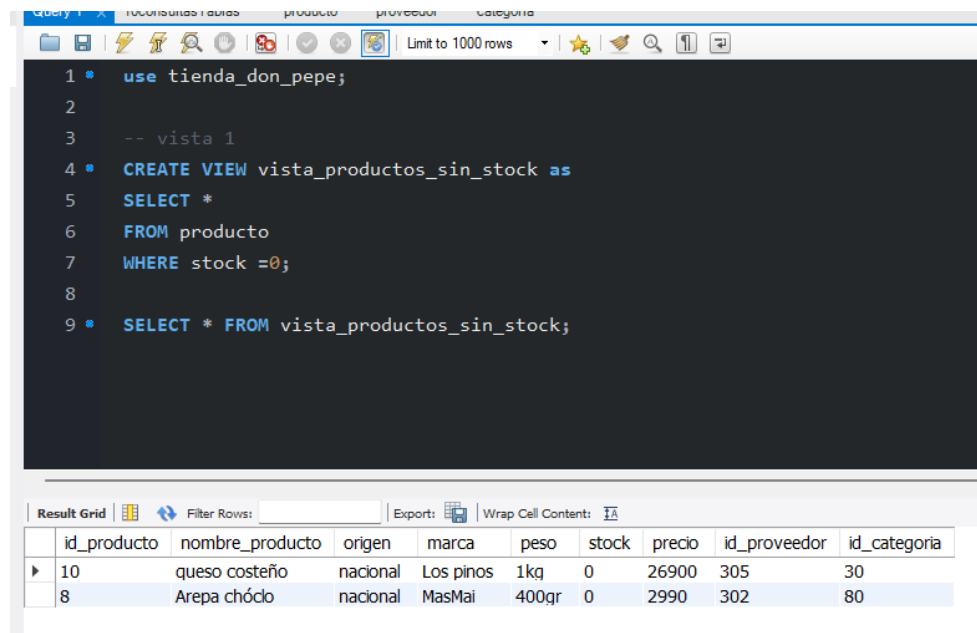
Action Output

#	Time	Action	Message
1	14:26:53	SELECT cliente.nombre_cliente, domiciliario.nombre_domiciliario, zona_postal.zona FRO...	5 row(s) returned

5- Generar vistas

Recordemos que las vistas son una tabla virtual a partir de un conjunto de tablas en una base relacional que, solo almacenan la definición, los datos que se recuperan mediante una consulta y se presentan como una tabla.

1- Vista: lista de productos sin unidades en stock



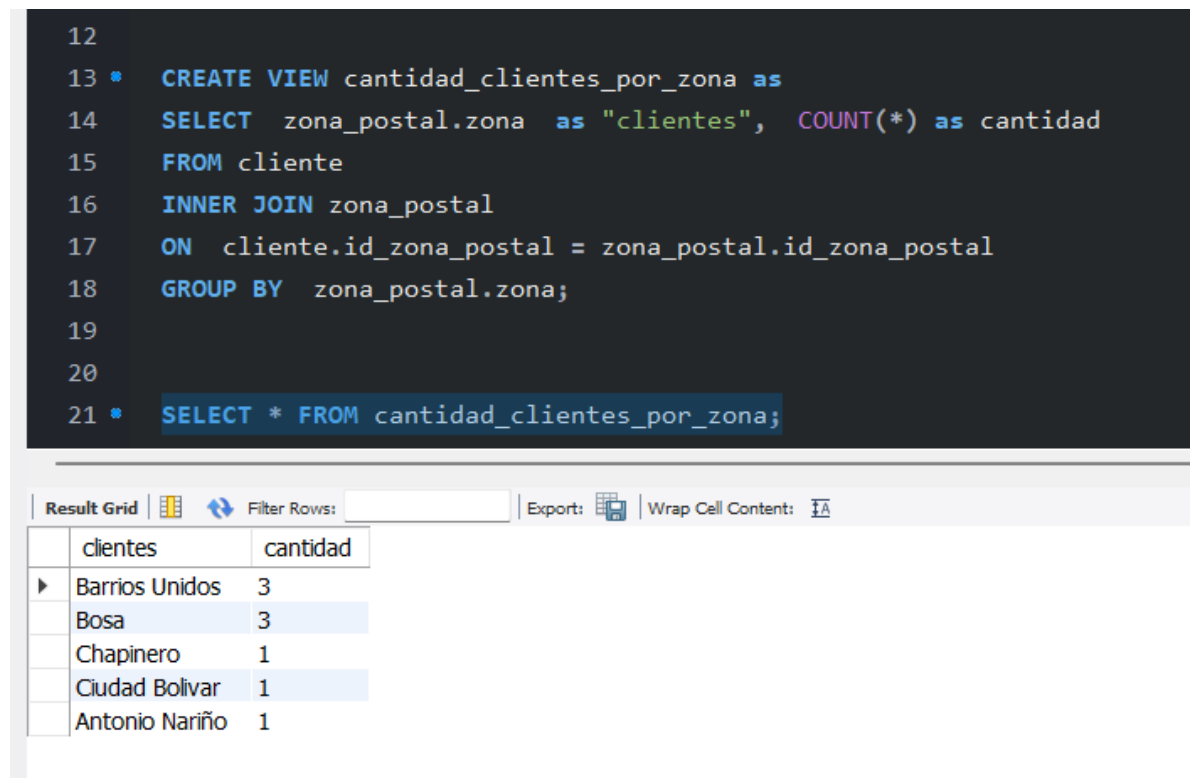
The screenshot shows a SQL IDE window with a query editor and a result grid. The query editor contains the following SQL code:

```
1 • use tienda_don_pepe;
2
3 -- vista 1
4 • CREATE VIEW vista_productos_sin_stock as
5 SELECT *
6 FROM producto
7 WHERE stock =0;
8
9 • SELECT * FROM vista_productos_sin_stock;
```

The result grid displays the following data:

	id_producto	nombre_producto	origen	marca	peso	stock	precio	id_proveedor	id_categoria
▶	10	queso costeño	nacional	Los pinos	1kg	0	26900	305	30
	8	Arepá chóclo	nacional	MasMai	400gr	0	2990	302	80

1- Vista: cantidad de clientes por zonas que cubre la tienda



The screenshot shows a SQL IDE window with a query editor and a result grid. The query editor contains the following SQL code:

```
12
13 • CREATE VIEW cantidad_clientes_por_zona as
14 SELECT zona_postal.zona as "clientes", COUNT(*) as cantidad
15 FROM cliente
16 INNER JOIN zona_postal
17 ON cliente.id_zona_postal = zona_postal.id_zona_postal
18 GROUP BY zona_postal.zona;
19
20
21 • SELECT * FROM cantidad_clientes_por_zona;
```

The result grid displays the following data:

	clientes	cantidad
▶	Barrios Unidos	3
	Bosa	3
	Chapinero	1
	Ciudad Bolívar	1
	Antonio Nariño	1

2- Vista: Datos del cliente, fecha de pedido y estado del pedido

```
23
24 • CREATE VIEW datos_cliente_estatus_pedido as
25 SELECT cliente.nombre_cliente, cliente.email, cliente.cedula, cliente.telefono, pedido.fecha_pedido, estado_pedido.estado
26 FROM cliente
27 INNER JOIN pedido
28 ON cliente.id_cliente = pedido.id_cliente
29 INNER JOIN estado_pedido
30 ON pedido.id_estado_pedido = estado_pedido.id_estado_pedido;
31
32 • SELECT * FROM datos_cliente_estatus_pedido;
33
```

nombre_cliente	email	cedula	telefono	fecha_pedido	estado
Juan Perez	Juan@gmail.com	120455	31156897	2023-02-11	solicitado
Luna Carrillo	Luna@gmail.com	160457	3115600	2023-02-11	recibido
Juan Perez	Juan@gmail.com	120455	31156897	2023-01-31	recibido
Ramiro Diaz	Diaz@gmail.com	3478996	31441064	2023-02-12	recibido
Ramiro Diaz	Diaz@gmail.com	3478996	31441064	2023-02-15	enProceso

3- Vista: Nombre de producto y su tipo de almacenamiento

```
31
32 • SELECT * FROM datos_cliente_estatus_pedido;
33
34 -- vista 4 producto y almacenamiento
35 • CREATE VIEW producto_tipo_almacenamiento as
36 SELECT producto.nombre_producto, categoria.almacenamiento, categoria.nombre_categoria
37 FROM producto
38 INNER JOIN categoria
39 ON producto.id_categoria = categoria.id_categoria
40 ORDER BY categoria.almacenamiento ASC
41
```

nombre_producto	almacenamiento	nombre_categoria
Pernil de pollo	congelado	Carnes
Helado chocolate	congelado	Helado
Mr Tea	frio	Bebidas
queso costeño	frio	Lacteos
Queso Mozzarella	frio	Lacteos
Arepa chόclo	frio	Arepas
Arepa chόclo	frio	Arepas
Shampoo	seco	Aseo Personal
Jabon	seco	Aseo Personal

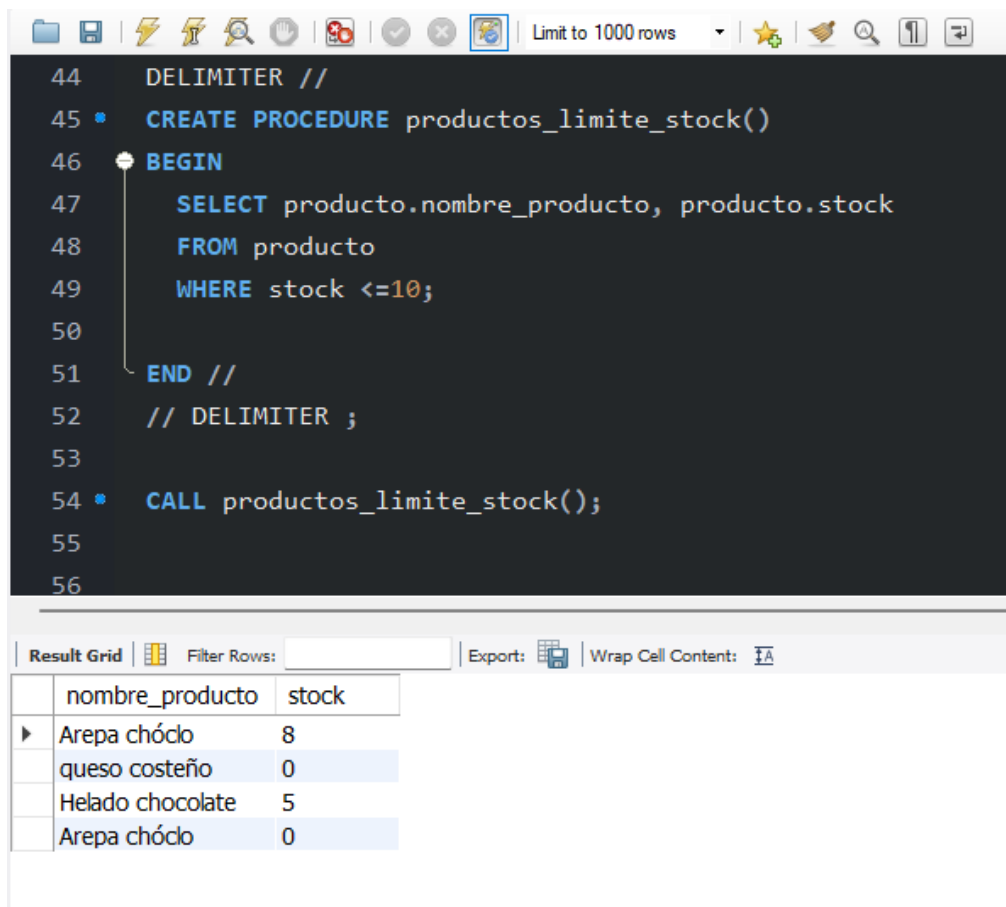
Result 40 x

6- Generar Procesos

Los procedimientos nos permiten agilizar consultas de datos, reutilizar código, como subrutinas creadas que pueden ser invocadas repetidamente.

1- Procedimiento: límite mínimo de stock en almacén por producto.

Este procedimiento permite visualizar los productos que tengan un mínimo de 10 o menos de unidades en stock.



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and search. A dropdown menu indicates 'Limit to 1000 rows'. The main editor displays the following SQL code:

```
44 DELIMITER //  
45 * CREATE PROCEDURE productos_limite_stock()  
46 BEGIN  
47     SELECT producto.nombre_producto, producto.stock  
48     FROM producto  
49     WHERE stock <=10;  
50  
51 END //  
52 // DELIMITER ;  
53  
54 * CALL productos_limite_stock();  
55  
56
```

Below the editor, the 'Result Grid' tab is active, showing a table with the results of the procedure execution:

	nombre_producto	stock
▶	Arepa chóclo	8
	queso costeño	0
	Helado chocolate	5
	Arepa chóclo	0

2- Procedimiento: agregar un cliente (insertar data)

```
60 • CREATE PROCEDURE agregar_cliente(  
61     IN id_cliente VARCHAR(15),  
62     IN nombre_cliente VARCHAR(45),  
63     IN cedula VARCHAR(20),  
64     IN email VARCHAR(60),  
65     IN password VARCHAR(45),  
66     IN telefono VARCHAR(45),  
67     IN id_zona_postal VARCHAR(15)  
68 )  
69  
70  
71 • BEGIN  
72     INSERT INTO cliente(id_cliente, nombre_cliente, cedula, email, password, telefono, id_zona_postal)  
73     VALUES(id_cliente, nombre_cliente, cedula, email, password, telefono, id_zona_postal);  
74  
75 END //  
76 // DELIMITER ;  
77  
78 • CALL agregar_cliente('111', 'Camila Cabello', '2567892', 'cabelloc@gmail.com', 'C546C', '32045878', '1');  
79  
80  
81
```

id_cliente	nombre_cliente	cedula	email	password	telefono	id_zona_postal
100	Juan Perez	120455	Juan@gmail.com	1235a	31156897	3
101	Saundra Turbat	660457	turba@gmail.com	1235TH	3165600	2
102	Luna Carrillo	160457	Luna@gmail.com	1235L	3115600	6
103	Shaughn Sneddon	990457	Sneddon@gmail.com	123PP	3115665	3
104	Ransel Filppov	500457	Filppov@gmail.com	1235KJ	3215614	4
105	Felipe Biquen	514478	felp@gmail.com	1254IFF	654789	3
106	Ramiro Diaz	3478996	Diaz@gmail.com	123RD	31441064	5
107	Raquel Smith	845759	smith@gmail.com	R123	32147866	2
108	Ramon Ortiz	106476	ramo@gmail.com	3698RO	314527852	2
111	Camila Cabello	2567892	cabelloc@gmail.com	C546C	32045878	1

Se agrego nuevo cliente

3- Procedimiento: Ganancias por pedidos en una fecha

```
84 • CREATE PROCEDURE ventas_totales_pedidos_porfecha()  
85  
86 • BEGIN  
87     SELECT pedido.fecha_pedido, SUM(total)  
88     FROM pedido  
89     GROUP BY pedido.fecha_pedido;  
90  
91 END //  
92 // DELIMITER ;  
93  
94  
95 • CALL ventas_totales_pedidos_porfecha();
```

fecha_pedido	SUM(total)
2023-02-11	110678
2023-01-31	235800
2023-02-12	218000
2023-02-15	36000

Suma los ingresos de venta de los pedidos por fecha

4- Procedimiento: Modificar/actualizar el precio de un producto

```
DELIMITER //
```

- **CREATE PROCEDURE** actualizar_precio_producto(
 IN id_producto VARCHAR(15),
 IN nuevo_precio VARCHAR(45)
)

 BEGIN
 UPDATE producto
 SET precio = nuevo_precio
 WHERE producto.id_producto = id_producto;

 END //
 // DELIMITER ;
- **CALL** actualizar_precio_producto('8', '6500');

Antes de aplicar proceso:

```
2
3 • SELECT *
4 FROM producto
5 WHERE id_producto = 8;
```

	id_producto	nombre_producto	origen	marca	peso	stock	precio	id_proveedor	id_categoria
▶	8	Arepá choco	nacional	MasMai	400gr	0	6500	302	80

Después:

```
1 SELECT * FROM tienda_don_pepe.producto;
2
3 SELECT *
4 FROM producto
5 WHERE id_producto =8;
```

	id_producto	nombre_producto	origen	marca	peso	stock	precio	id_proveedor	id_categoria
▶	8 NULL	Arepá chóclo	nacional NULL	MasMai NULL	400qr NULL	0 NULL	6540	302 NULL	80 NULL

7- Generar Tigger

Es un disparador cuya función, es lanzar un evento en cierto momento before (antes), after(después), casi siempre se usan para insertar, actualizar o borrar registros.

1- Tigger: actualizar la cantidad de unidades de un producto

Este Tigger permite disparar una operación cuando se realiza un detalle de pedido, que especifica la cantidad de cada producto que compra el cliente, restando al stock de ese producto:

```
165 -- tigger 4 actualizar stock cuando se haga un pedidos
166 DELIMITER //
167 • CREATE TRIGGER actualizar_stock_productos
168 AFTER INSERT ON detalles_pedido
169 FOR EACH ROW
170 BEGIN
171     UPDATE producto
172     SET stock = producto.stock - NEW.cantidad_productos
173     WHERE producto.id_producto = NEW.id_producto;
174
175 END;
176 // DELIMITER ;
```

Antes de aplicar proceso:

id_producto	nombre_producto	origen	marca	peso	stock	precio	id_proveedor	id_categoria
1	Pernil de pollo	nacional	Bucanero	750g	26	13589	304	40
10	Arepa chóclo	nacional	Sary	300gr	8	3500	302	80
11	queso costeño	nacional	Los pinos	1kg	0	26900	305	30
12	Helado chocolate	nacional	popsy	2375	5	18900	305	90
15	lavalosa	nacional	Ajax	500gr	15	5900	302	11
2	Arroz blanco	nacional	Diana	1kg	40	5300	301	50
3	Shampoo	nacional	Savitel	350ml	12	9600	302	10
4	Queso Mozzarella	nacional	Colanta	250gr	15	7500	305	30
5	atún natural	nacional	Alamar	140gr	35	6990	302	60
6	Mr Tea	nacional	Mr. Tea	500ml	25	4500	302	20
7	ajo	nacional	El Rey	60g	40	2900	303	70
8	Arepa chóclo	nacional	MasMai	400gr	0	6540	302	80
9	Jabon	nacional	Protex	330gr	25	8100	302	10
* NOLE	NOLE	NOLE	NOLE	NOLE	NOLE	NOLE	NOLE	NOLE

Antes de usar el tigger de detalles de pedido que disminuye el stock

Registra un pedido (se solicita una unidad del producto):

	cantidad_productos	precio	id_producto	id_pedido	id_categoria
	2	13598	1	801	40
	4	7590	4	801	30
	1	7590	4	805	30
	3	2490	7	801	70
	NULL	NULL	NULL	NULL	NULL

Después de aplicar proceso, se resta una unidad al stock:

id_producto	nombre_producto	origen	marca	peso	stock	precio	id_proveedor	id_categoria
1	Pernil de pollo	nacional	Bucanero	750g	20	13589	304	40
10	Arepa chόdo	nacional	Sary	300gr	8	3500	302	80
11	queso costeño	nacional	Los pinos	1kg	0	26900	305	30
12	Helado chocolate	nacional	popsy	2375	5	18900	305	90
15	lavalosa	nacional	Ajax	500gr	15	5900	302	11
2	Arroz blanco	nacional	Diana	1kg	40	5300	301	50
3	Shampoo	nacional	Savitel	350ml	12	9600	302	10
4	Queso Mozzarella	nacional	Colanta	250gr	14	7500	305	30
5	atún natural	nacional	Alamar	140gr	35	6990	302	60
6	Mr Tea	nacional	Mr. Tea	500ml	25	4500	302	20
7	ajo	nacional	El Rey	60g	40	2900	303	70
8	Arepa chόdo	nacional	MasMai	400gr	0	6540	302	80
9	Jabon	nacional	Protex	330gr	25	8100	302	10
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

2- Tigger: registrar un log de acciones agregar un nuevo producto

Este Tigger permite disparar diversas acciones que se registran en la siguiente tabla:

```

• CREATE TABLE acciones (
    id INT NOT NULL AUTO_INCREMENT,
    accion VARCHAR(45) NULL,
    fecha DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (id));
  
```

Se crea una tabla acciones para log de cambios

```
-- tigger 2 log de registros cuando se agregar nuevos pedidos mostrando la fecha
DELIMITER //
• CREATE TRIGGER log_productos → EL tigger se dispara cuando se
  AFTER INSERT ON producto          agrega un nuevo producto
  FOR EACH ROW
  BEGIN
    INSERT INTO acciones(accion)
    VALUE ('Se agrego un nuevo producto');
  END;
  // DELIMITER ;
```

Se registra en la tabla acciones que se agregó a la tabla productos un nuevo producto:

Result Grid			
	id	accion	fecha
▶	1	Se agrego un nuevo producto	2023-02-18 22:55:49

3- Tigger: registrar un log de acciones agregar un nuevo pedido

```
51 -- tigger 3 log de registros cuando se agregar nuevos pedidos mostrando la fecha
52 DELIMITER //
53 • CREATE TRIGGER log_pedido
54 AFTER INSERT ON pedido
55 FOR EACH ROW
56 BEGIN
57   INSERT INTO acciones(accion)
58   VALUE ('Se solicito un nuevo pedido');
59
60 END;
61 // DELIMITER ;
62
```

Se registra en la tabla acciones que se agregó a la tabla productos un nuevo registro:

Result Grid			
	id	accion	fecha
▶	1	Se agrego un nuevo producto	2023-02-18 22:55:49
	2	Se solicito un nuevo pedido	2023-02-18 22:58:53
+	NULL	NULL	NULL

4- Tigger: registrar un log de acciones actualizo el registro de un domiciliario

```
-- tigger 5 log de registros actualizacion de un domiciliari
DELIMITER //

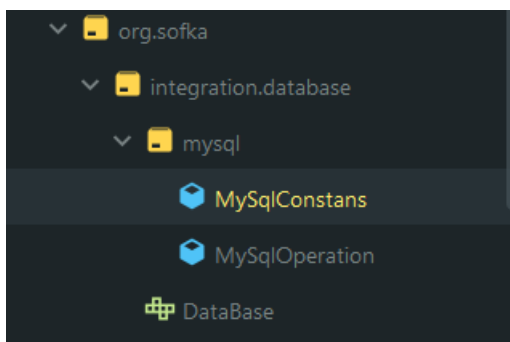
• CREATE TRIGGER agregar_domiciliario
  AFTER DELETE ON domiciliario
  FOR EACH ROW
  BEGIN
    INSERT INTO acciones(accion)
    VALUES ('El domiciliario ha sido actualizado.');
```

DELIMITER ;

8- Generar Data por medio de conexión Java

Al momento de realizar pruebas sobre un sistema de software, se debe generar o incluir datos para probar diferentes funcionalidades. En este sentido, para la tienda Don Pepe, se poblará las tablas existentes con datos simulados de la librería Faker generar datos falsos de forma automática para permite explorar y validar datos.

El primer paso fue crear la integración de MySQL y Java mediante constantes y operaciones en conjunto con una interface a continuación se evidencia el código:



```

1 usage  👤 katherine Bonilla
public class MySQLConstans {

    1 usage
    public static final String MY_SQL_JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";

    1 usage
    public static final String CONNECTION_STRING = "jdbc:mysql://%s/%s?user=%s&password=%s";

    1 usage
    public static final String SERVER="localhost";
    1 usage
    public static final String DATA_BASE_NAME="tienda_don_pepe";
    1 usage
    public static final String USER="root";
    1 usage
    public static final String PASSWORD="password";

}

```

Interface que se implementa en las operaciones:

```

package org.sofka.integration.database;

import java.sql.ResultSet;
import java.sql.SQLException;

2 📖 1 implementation  👤 katherine Bonilla
public interface DataBase {

    1 implementation  👤 katherine Bonilla
    public void configureDatabaseconetion();
    1 usage  1 implementation  👤 katherine Bonilla
    public void executeSqlStatement();
    1 usage  1 implementation  👤 katherine Bonilla
    public void executeSqlStatementVoid();
    4 usages  1 implementation  👤 katherine Bonilla
    public ResultSet getResultset();
    3 usages  1 implementation  👤 katherine Bonilla
    public void close();
    no usages  1 implementation  👤 katherine Bonilla
    public void printResultset() throws SQLException;

}

```

Clase que ejecuta connexion:

```
package org.sofka.integration.database.mysql;

import org.sofka.integration.database.DataBase;

import java.sql.*;

import static org.sofka.integration.database.mysql.MySqlConstans.*;

⚡ katherine Bonilla
public class MySqlOperation implements DataBase {

    4 usages
    private Connection connection=null;
    5 usages
    private Statement statement = null;
    7 usages
    private ResultSet resultSet = null;

    4 usages
    private String sqlStatement ;
    3 usages
    private String server;
    3 usages
    private String dataBaseName;
    3 usages
    private String user;
    3 usages
    private String password;

    public String getSqlStatement() { return sqlStatement; }

    2 usages ⚡ katherine Bonilla
    public void setSqlStatement(String sqlStatement) { this.sqlStatement = sqlStatement; }

    no usages ⚡ katherine Bonilla
    public String getServer() { return server; }

    1 usage ⚡ katherine Bonilla
    public void setServer(String server) { this.server = server; }

    no usages ⚡ katherine Bonilla
    public String getDataBaseName() { return dataBaseName; }

    1 usage ⚡ katherine Bonilla
    public void setDataBaseName(String dataBaseName) { this.dataBaseName = dataBaseName; }

    no usages ⚡ katherine Bonilla
    public String getUser() { return user; }

    1 usage ⚡ katherine Bonilla
    public void setUser(String user) { this.user = user; }

    no usages ⚡ katherine Bonilla
    public String getPassword() { return password; }

    1 usage ⚡ katherine Bonilla
    public void setPassword(String password) { this.password = password; }

    ⚡ katherine Bonilla
```

Métodos sobrescritos de la interface DataBase:

```
1 Katherine Bonilla
@Override
public void configureDatabaseconetion() {
    try {
        Class.forName(MY_SQL_JDBC_DRIVER);

        connection=DriverManager.getConnection(
            String.format(CONNECTION_STRING,
                this.server,
                this.dataBaseName,
                this.user,
                this.password ));
        statement =connection.createStatement();
    }
    catch (Exception e){
        close();
        System.out.println(e.getMessage());
    }
}

1 usage Katherine Bonilla
@Override
public void executeSqlStatement() {
    try {
        configureDatabaseconetion();
        resultSet = statement.executeQuery(sqlStatement);
    }
    catch (Exception e){
        System.out.println(e.getMessage());
    }
}

1 usage Katherine Bonilla
@Override
public void executeSqlStatementVoid() {
    try {
        configureDatabaseconetion();
        statement.execute(sqlStatement);
    }
    catch (Exception e){
        System.out.println(e.getMessage());
    }
}
```

Como se dará el accionar de la conexión para cerrar punteros de cada consulta para cerrar o ejecutar una consulta:

```

3 usages  Katherine Bonilla
public void myOpenConnection(){
    this.setServer(SERVER);
    this.setDataBaseName(DATA_BASE_NAME);
    this.setUser(USER);
    this.setPassword(PASSWORD);
}

1 usage  Katherine Bonilla
public void myExecuteQuery(String sql){
    myOpenConnection();
    this.setSqlStatement(sql);
    this.executeSqlStatement();
};

Katherine Bonilla
public void myExecuteInsertUpdate(String sql){
    myOpenConnection();
    setSqlStatement(sql);
    executeSqlStatementVoid();
    myCloseConnection();
};

2 usages  Katherine Bonilla
public void myCloseConnection(){ this.close(); };

}

```

Se modelan las clases para cada entidad de la base de datos algunos ej:

1. Modelo categoría con setters and getters:

```

19 usages  Katherine Bonilla
public class Categoria {

    3 usages
    private String idCategoria ;
    3 usages
    private String nombreCategoria;
    3 usages
    private String almacenamiento;
    3 usages
    private String observacioes;

    no usages  Katherine Bonilla
    public Categoria() {
    }

    8 usages  Katherine Bonilla
    public Categoria(String idCategoria, String nombreCategoria, String almacenamiento, String observacioes) {
        this.idCategoria = idCategoria;
        this.nombreCategoria = nombreCategoria;
        this.almacenamiento = almacenamiento;
        this.observacioes = observacioes;
    }
}

```

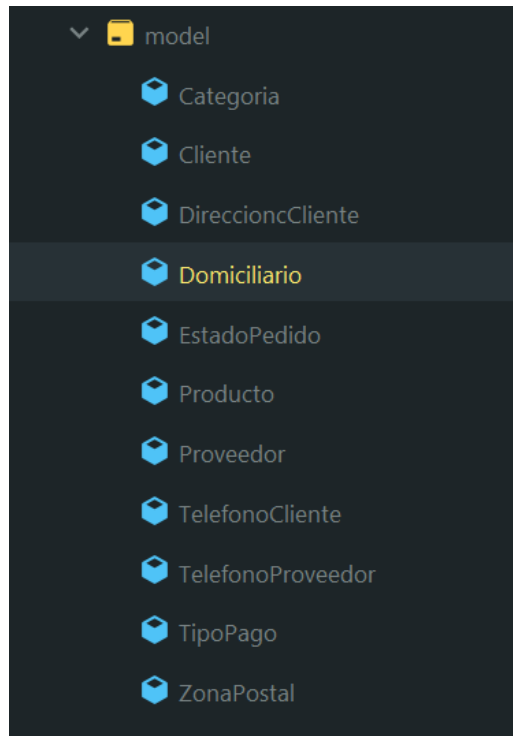
2. Modelo cliente con setters and getters:

```
public class Cliente {  
    3 usages  
    private String id;  
    3 usages  
    private String cedula;  
    3 usages  
    private String nombre;  
    3 usages  
    private String email;  
    3 usages  
    private String password;  
  
    3 usages  
    private String zonaPostal;  
  
    1 usage  Katherine Bonilla  
    public Cliente() {  
    }  
  
    no usages  Katherine Bonilla  
    public Cliente(String id, String cedula, String nombre, String email, String password, String zonaPostal) {  
        this.id = id;  
        this.cedula = cedula;  
        this.nombre = nombre;  
        this.email = email;  
        this.password = password;  
        this.zonaPostal = zonaPostal;  
    }  
}
```

3. Modelo domiciliario con setters and getters:

```
package org.sorka.modelo;  
  
3 usages  Katherine Bonilla  
public class Domiciliario {  
  
    3 usages  
    private String id;  
    3 usages  
    private String nombre;  
  
    3 usages  
    private String matricula;  
  
    3 usages  
    private String zonaReparto;  
    3 usages  
    private String idZonapostal;  
  
    1 usage  Katherine Bonilla  
    public Domiciliario() {  
    }  
  
    no usages  Katherine Bonilla  
    public Domiciliario(String id, String nombre, String matricula, String zonaReparto, String idZonapostal) {  
        this.id = id;  
        this.nombre = nombre;  
        this.matricula = matricula;  
        this.zonaReparto = zonaReparto;  
        this.idZonapostal = idZonapostal;  
    }  
}
```


La carpeta de model en la estructura de proyecto queda de la siguiente manera:



Luego de modelar cada tabla en una clase se crean sus respectivos repository creando la interface, que se presenta a continuación para implantar en cada repository:

```
package org.sofka.repository;

import java.sql.SQLException;

11 implementations  katherine Bonilla
public interface Table {

    11 implementations  katherine Bonilla
    public void generateData() throws SQLException;

}

}
```

Tabla algunos ejemplos:

1. Repository cliente:

```
private Faker faker;

katherine Bonilla
@Override
public void generateData() throws SQLException {
    mysqlOperation = new MySqlOperation();
    mysqlOperation.configureDatabaseconetion();
    Cliente c = new Cliente();
    String sql = "";
    faker = new Faker();
    for (int i = 200; i <= 250; i++) {
        c.setId(String.valueOf(i));
        c.setCedula(String.valueOf(faker.idNumber().valid().replace( target: "-", replacement: "")));
        c.setNombre(faker.name().fullName());
        c.setEmail((faker.funnyName().name()+ "@gmail.com").replace( target: " ", replacement: ""));
        c.setPassword(String.valueOf(faker.number()));
        c.setZonaPostal(String.valueOf(i));

        sql = String.format("INSERT INTO CLIENTE VALUES('%s','%s','%s','%s','%s','%s');",
            c.getId(),
            c.getCedula(),
            c.getNombre(),
            c.getEmail(),
            c.getPassword(),
            c.getZonaPostal());
        System.out.println(sql);
        this.mysqlOperation.myExecuteInsertUpdate(sql);
    }
}
```

2- Repository Proveedor:

```
1 usage katherine Bonilla
public class ProveedorRepository implements Table {
    3 usages
    private MySqlOperation mysqlOperation;
    4 usages
    private Faker faker;

    katherine Bonilla
    @Override
    public void generateData() throws SQLException {
        mysqlOperation = new MySqlOperation();
        mysqlOperation.configureDatabaseconetion();
        Proveedor p = new Proveedor();
        String sql = "";
        faker = new Faker();

        for (int i = 100; i <= 150; i++) {
            p.setId(i);
            p.setNombre(faker.company().name().replace( target: "", replacement: " "));
            p.setDireccion(faker.address().fullAddress().replace( target: "", replacement: " "));
            p.setTelefono(faker.phoneNumber().subscriberNumber( length: 10));

            sql = String.format("INSERT INTO PROVEEDOR VALUES(%s,%s,%s,%s);",
                p.getId(),
                p.getNombre(),
                p.getDireccion(),
                p.getTelefono());
            System.out.println(sql);
            this.mysqlOperation.myExecuteInsertUpdate(sql);
        }
    }
}
```

Los repository permiten insertar los métodos que generan la data a través de la librería Faker, quedando siguiente manera las tablas:

- 1- Tabla categoría que es una excepción pues se realizó mediante un ArrayList :

```
public void generateData() throws SQLException {
    mySqlOperation = new MySqlOperation();
    mySqlOperation.configureDatabaseConnection();

    List<Categoria> categorias = new ArrayList<Categoria>();
    categorias.add(new Categoria( idCategoria: "110", nombreCategoria: "Electronicos", almacenamiento: "seco", observaciones: "cuidado personal"));
    categorias.add(new Categoria( idCategoria: "120", nombreCategoria: "Medicamentos", almacenamiento: "frio", observaciones: "gaseosa"));
    categorias.add(new Categoria( idCategoria: "130", nombreCategoria: "Inflamables", almacenamiento: "frio", observaciones: "Queso "));
    categorias.add(new Categoria( idCategoria: "140", nombreCategoria: "Veganos", almacenamiento: "frio", observaciones: "Pollo"));
    categorias.add(new Categoria( idCategoria: "150", nombreCategoria: "Categoria 150", almacenamiento: "seco", observaciones: "diferentes presentaciones"));
    categorias.add(new Categoria( idCategoria: "160", nombreCategoria: "Categoria 160", almacenamiento: "seco", observaciones: "atún y conservas"));
    categorias.add(new Categoria( idCategoria: "170", nombreCategoria: "Categoria 170", almacenamiento: "seco", observaciones: "despensa"));
    categorias.add(new Categoria( idCategoria: "180", nombreCategoria: "Vegetales y Frutas", almacenamiento: "frio", observaciones: "despensa "));

    String sql = "";

    for (Categoria c:categorias) {
        sql=String.format("INSERT INTO CATEGORIA VALUES('%s','%s','%s','%s');",
            c.getIdCategoria(),
            c.getNombreCategoria(),
            c.getAlmacenamiento(),
            c.getObservaciones());
    }
}
```

	id_categoria	nombre_categoria	almacenamiento	observaciones
▶	10	Aseo Personal	seco	cuidado personal
	11	Aseo	seco	Ase hogar
	110	Electronicos	seco	cuidado personal
	120	Medicamentos	frio	gaseosa
	130	Inflamables	frio	Queso
	140	Veganos	frio	Pollo
	150	Categoria 150	seco	diferentes presentaciones
	160	Categoria 160	seco	atÃn y conservas
	170	Categoria 170	seco	despensa
	180	Vegetales y Frutas	frio	despensa
	20	Bebidas	frio	gaseosa
	30	Lacteos	frio	Queso
	40	Carnes	congelado	Pollo
	50	Arroz y granos	seco	diferentes presentaciones
	60	enlatados	seco	atún y conservas
	70	condimentos	seco	despensa

2- Tabla cliente con data desde Faker con más de 50 registros:

id_cliente	nombre_cliente	cedula	email	password	id_zona_postal
103	Shaughn Sneddon	990457	Sneddon@gmail.com	123PP	3
104	Ransel Filppov	500457	Filppov@gmail.com	1235KJ	4
105	Felipe Bigen	514478	felip@gmail.com	1254IFF	3
106	Ramiro Diaz	3478996	Diaz@gmail.com	123RD	5
107	Raquel Smith	845759	smith@gmail.com	R123	2
108	Ramon Ortiz	106476	ramo@gmail.com	3698RO	2
111	Camila Cabello	2567892	cabelloc@gmail.com	C546C	1
200	193039957	Suzann Fritsch	CandiceB.DePlace@gmail.c...	com.github....	200
201	561206139	Shenna Murphy MD	ValVeeta@gmail.com	com.github....	201
202	491893772	William Rath	MaryGold@gmail.com	com.github....	202
203	430910933	Lucila Veum DVM	MidasWell@gmail.com	com.github....	203
204	307046902	Lyle Von	HammondEqqs@gmail.com	com.github....	204
205	459219804	Kayla Hill	OrenJelow@gmail.com	com.github....	205
206	795516241	Blake Rowe	JasmineRice@gmail.com	com.github....	206
207	392137555	Robbie Kemmer	SueRidge@gmail.com	com.github....	207
208	262647758	Suzi Ferry	NickL.Andime@gmail.com	com.github....	208
209	390022664	Lynn Baumbach	C.Worthy@gmail.com	com.github....	209
210	633520612	Annita Schmeler	JoyAnnaDeLight@gmail.com	com.github....	210
211	739812813	Augustine Wiza	ReneeSance@gmail.com	com.github....	211
212	491047799	Leo Howe	ClaraNett@gmail.com	com.github....	212
213	590312435	Newton Brown	MelbaCrisp@gmail.com	com.github....	213
214	280076347	Ambrose Volkman	MarshaDimes@gmail.com	com.github....	214
215	493736236	Ms. Iola Emard	DouglasFurr@gmail.com	com.github....	215
216	408322750	Glen Beier	MaryOtt@gmail.com	com.github....	216
217	668866140	Walker Little	DouglasS.Halfempty@qma...	com.github....	217
218	282177390	Margarito Wsozk	OliveYew@gmail.com	com.github....	218

3- Tabla Domiciliario con data desde Faker con más de 50 registros:

id_domiciliario	nombre_domiciliario	matricula_vehiculo	zona_reparto	id_zona_postal
13D	Myriam Ramos	L45K	Bosa	3
14D	Jose Martinez	BVFG6	Soacha	1
1D	Pedro Jimenez	RMT452	Barrios Unidos	2
200	Illustrious Callisto Ivy	424153	Runteland	200
201	Captain Blade Lord	487678	North Latoya	201
202	Scorpion	384960	Schimmelshire	202
203	Firestar	111511	West Sterling	203
204	Magnificent Rachel Pirzad	996531	East Benjaminfort	204
205	Ultra Maya Herrera	032593	North Gary	205
206	Sif X	621818	Beermouth	206
207	Hybrid	744704	Shakitaland	207
208	Metron	697653	Beerborough	208
209	Forge	768876	Caseyhaven	209
210	Tinkerer Lord	710924	Blickville	210
211	Green Moonstone Dragon	443043	Randyshire	211
212	Red Ink the Hunter	913815	Port Marlena	212
213	Golden	031330	Reflexion	213

4- Tabla Proveedor con data desde Faker con más de 50 registros:

id_proveedor	nombre_proveedor	direccion	telefono
100	Hilpert-Klocko	Apt. 306 0495 Gerh...	5042343302
101	Klein Inc	6218 Chasity Hill, Ea...	2131558864
102	Prosacco Group	Suite 356 1399 Emil ...	2232049876
103	Lynch-Yundt	Apt. 614 600 Farrell ...	8409517159
104	Rolfson, Leannon and Crist	048 Hayes Street, N...	2303192487
105	Weimann-Considine	0872 Kenneth Points...	4052142537
106	Funk-Gleason	4451 Vandervort Vis...	1581949653
107	Balstreri-Tromp	6555 Antoine Mill, La...	0224105651
108	Gottlieb LLC	6100 Dietrich Knolls, ...	7158588488
109	Smitham Inc	Apt. 755 78961 Jaci...	0242778897
110	Harber and Sons	016 Trantow Road, ...	2405697523
111	Becker, Fay and Dicki	236 Wuckert Greens...	2740000291
112	Emmerich, Ortiz and Schneider	67529 Bernhard Inle...	2071114480
113	Padberg-Corwin	2581 Ronald Station...	7124429656
114	Cummings-Volkman	Apt. 608 568 Lane S...	4692979570
115	Bednar Inc	570 Thompson Light...	6345000413
116	Kemmer, Kuhic and Corkery	Apt. 946 666 Maggie...	8513819521
117	Walsh, Abernathy and Langosh	648 Armstrong Lake...	5483329750
118	Mosciski LLC	Apt. 547 252 Breiten...	0960902439
119	Roberts-Halvorson	Suite 251 265 Micha ...	9830138095
120	Schoen-Thompson	32125 Ozzie Keys, ...	7889165958
121	Turner LLC	Apt. 209 3377 Mitch...	5862279563
122	McDermott, Frami and Sporer	1904 Emilia Park, La...	3098116549
123	Stroman LLC	79167 Miguel Manor	2220636726

5- Tabla teléfono cliente con data desde Faker con más de 50 registros:

id_cliente	telefono
100	31156897
101	3165600
102	3115600
103	3115665
104	3215614
105	654789
106	31441064
200	1493606344
201	3030639868
202	4558324278
203	2922080400
204	5555056291
205	2066117038
206	2442785307
207	0888543903
208	8056103362
209	0410148984
210	3525383650
211	4037681717
212	7664037654
213	0376654737
214	9070180642
215	9150258665
216	7666504507

6- Tabla con data desde Faker con más de 50 registros:

Result Grid			Filter Rows:	Edit:	Export/Import
	id_proveedor	telefono			
▶	100	4659646673			
	101	1841033114			
	102	5643884539			
	103	2006986929			
	104	4969625489			
	105	8933804396			
	106	2271663157			
	107	7782908087			
	108	2024567345			
	109	0832143020			
	110	0172018527			
	111	1157595239			
	112	9267147080			
	113	9680193007			
	114	4180087199			
	115	9942615854			
	116	4963522905			
	117	3493651078			
	118	3305457398			
	119	7263517027			
	120	6894650435			
	121	8571045302			
	122	1300105125			
	123	1905425816			

7- Tabla tipo de pago con data desde Faker con más de 50 registros:

Result Grid				Filter Rows:	Edit:	Export/Import
	id_tipo_de_pago	numero_tarjeta	fecha_caducidad			
▶	1	2766985511996006	23-03-2027			
	10	6386061488537002	23-07-2025			
	11	4781740091909305	13-10-2024			
	12	2597745589996601	24-09-2023			
	13	7283522023304943	22-10-2027			
	14	2415128468426632	14-08-2025			
	15	7686635107452398	25-04-2027			
	16	9968803240087005	08-07-2028			
	17	5515015539452597	09-12-2023			
	18	8423170285991057	12-09-2029			
	19	6258010144449175	26-10-2023			
	2	8535627084431670	09-10-2030			
	20	9555941625587345	09-09-2026			
	21	7384101594770877	09-06-2026			
	22	4946754688686816	19-08-2027			
	23	2008073065090464	19-11-2030			
	24	8598442599795475	06-04-2030			
	25	6624327627692946	01-11-2028			
	26	7013297155371475	15-11-2027			
	27	9821123839520566	12-02-2027			
	28	6868258719095135	09-05-2026			
	29	8436265826702596	29-03-2027			
	3	7464738845283122	16-08-2023			
	30	4925443011803655	16-06-2026			
	31	3201887076078791	02-12-2027			
	32	7949913431451826	12-07-2026			







po_de_pago1 x

Output

Action Output

#	Time	Action
✓ 14	16:38:44	SELECT * FROM tienda_don_pepe.telefono_proveedor LIMIT 0, 1000

8- Tabla zona postal con data desde Faker con más de 50 registros:

Result Grid			 Filter Rows:	<input type="text"/>	Edit:				Export/Import:	
	id_zona_postal	zona	codigo_postal							
▶	1	Soacha	25411							
	2	Barrios Unidos	111211							
	200	Runteland	35306							
	201	North Latoya	67357-7391							
	202	Schimmelshire	70211-3431							
	203	West Sterling	21117							
	204	East Benjaminfort	11913							
	205	North Gary	56192-6018							
	206	Beermouth	91604-6138							
	207	Shakitaland	90878							
	208	Beerborough	14211-4973							
	209	Caseyhaven	28279							
	210	Blickville	87822-0318							
	211	Randyshire	03706							
	212	Port Marlana	39321							
	213	Balistreriport	69661-4039							
	214	East Tomekaton	41407							
	215	North Fidel	34536							
	216	Dreamaport	49222							
	217	Bergehaven	94634							
	218	Cordelland	35147							
	219	Doyleshire	72029							
	220	New Bridgettechester	66454							
	221	Ellsworthshire	98236-6568							
	222	New Larry	99322-0668							
	223	East Teenafurt	26058-1132							

zona postal 1 ×