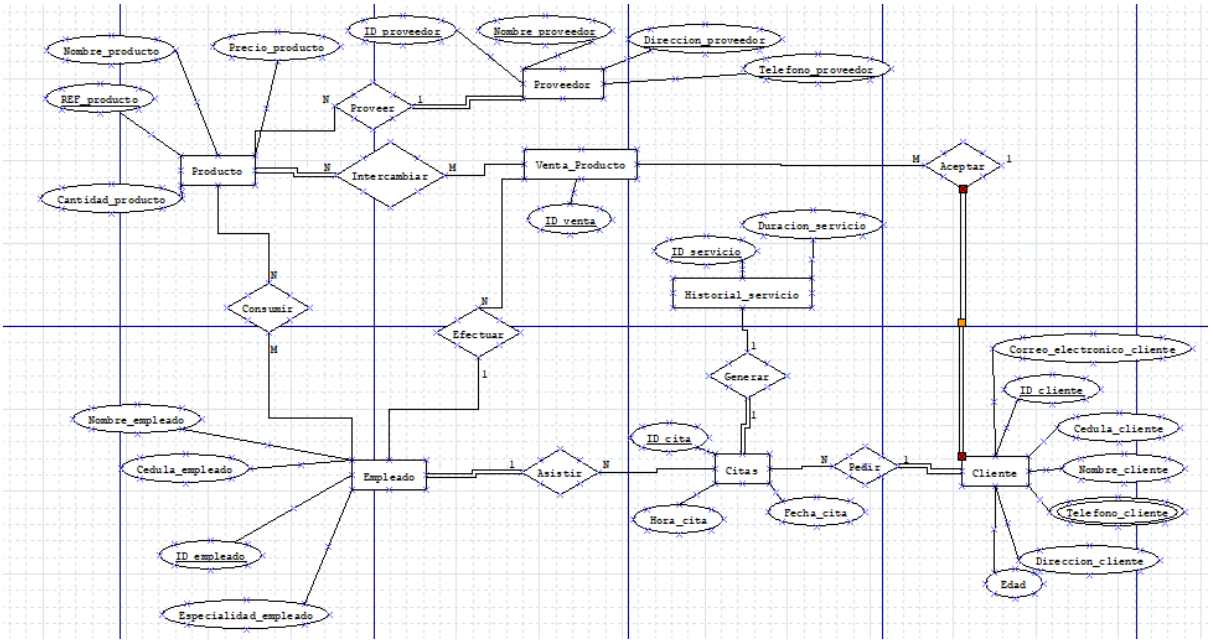
El ejercicio asignado fue el A (Barberia)

1. Se realiza el modelo E/R y queda de esta forma:

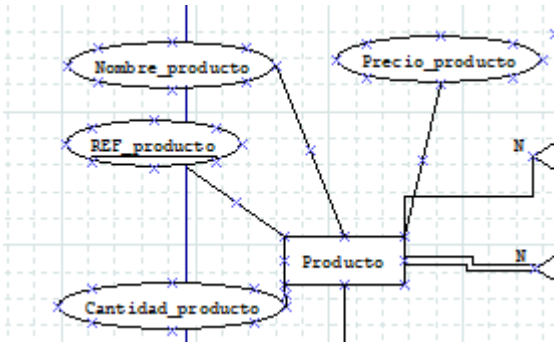
   En el análisis para la creación del modelo, se observan dos condiciones muy importantes:

   - Los clientes serán atendidos por medio de una cita, no importa si llegan directamente al local y piden un servicio, se les da la cita dependiendo la disponibilidad de los empleados.
   - Cuando el empleado atiende una cita, consume x cantidad de productos que están tomados en cuenta en el costo del servicio.

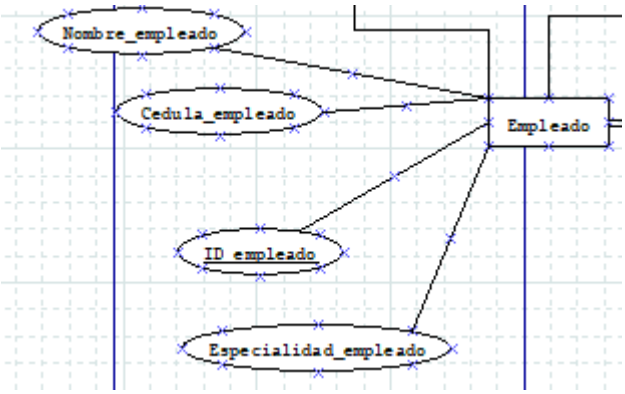


Las entidades involucradas son:

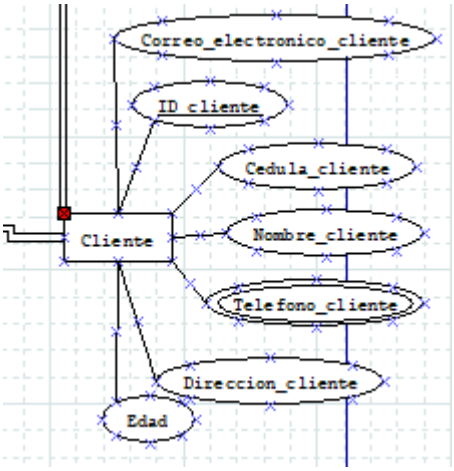
- Producto



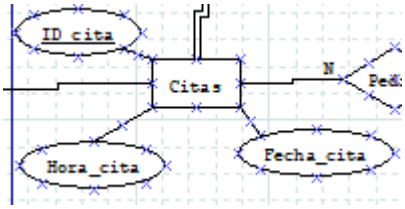
- Empleado



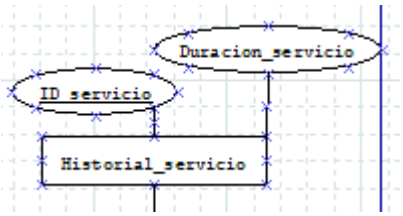
- Cliente



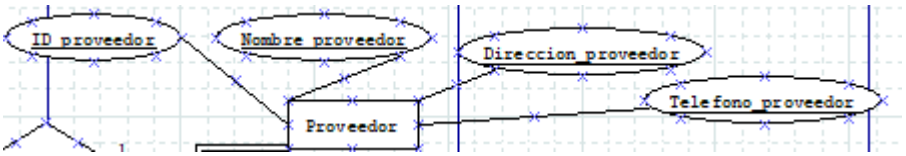
- Cita



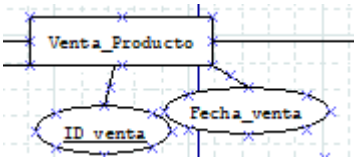
- Historial servicio



- Proveedor

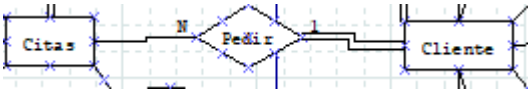


- Venta producto

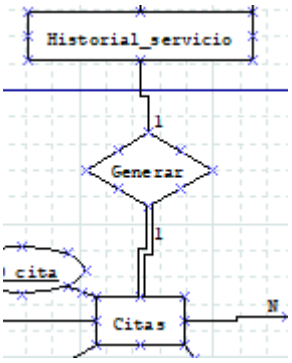


Las relaciones que se pueden evidenciar en el modelo E/R son:

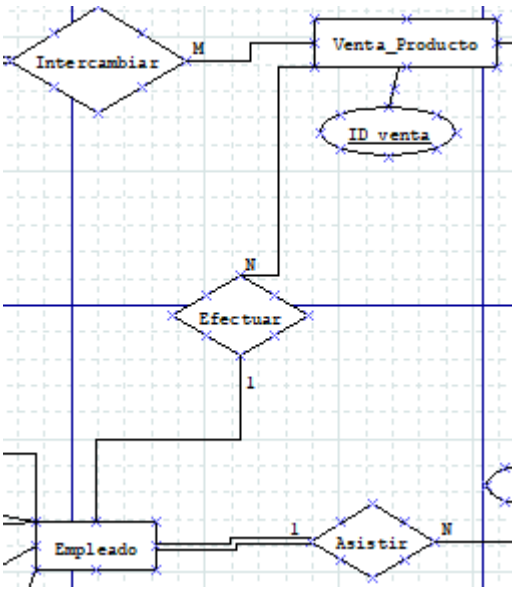
- Un empleado puede asistir a una o varias citas.
- Una cita puede ser asistida por uno empleado o varios empleados.



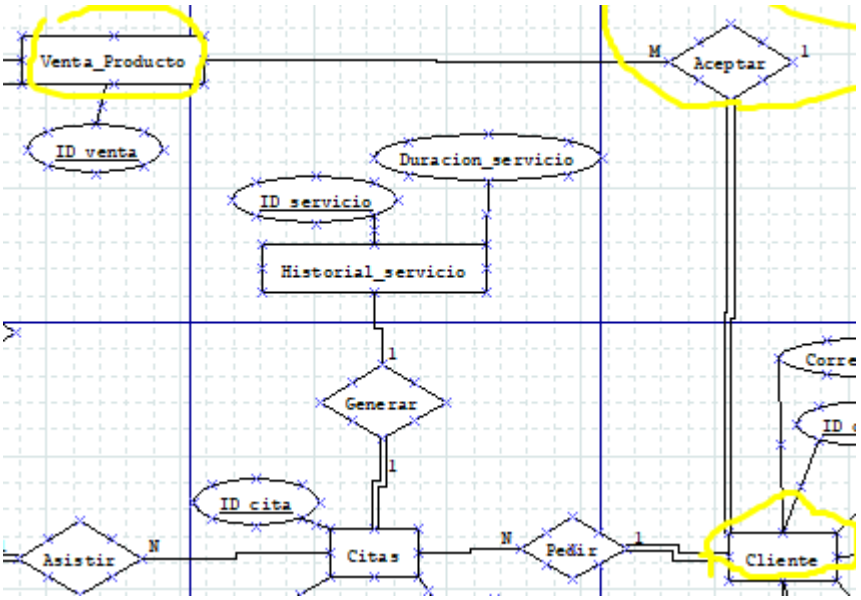
- Una cita puede ser pedida por un cliente.
- Un cliente puede pedir una o varias citas.



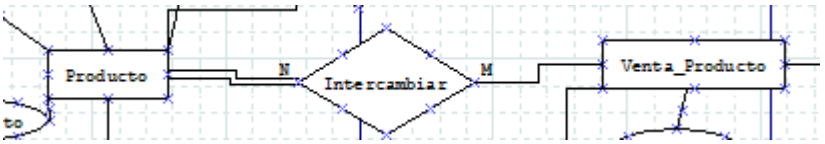
- Una cita puede generar un historial.
- Un historial puede ser generado por una cita.



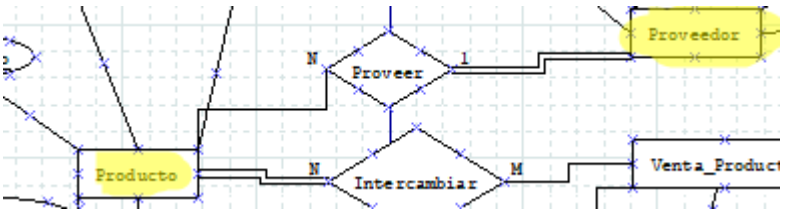
- Un empleado puede efectuar una o varias ventas.
- Una venta puede ser efectuada por un empleado.



- Un cliente puede aceptar una o varias ventas.
- Una venta puede ser aceptada por un cliente.

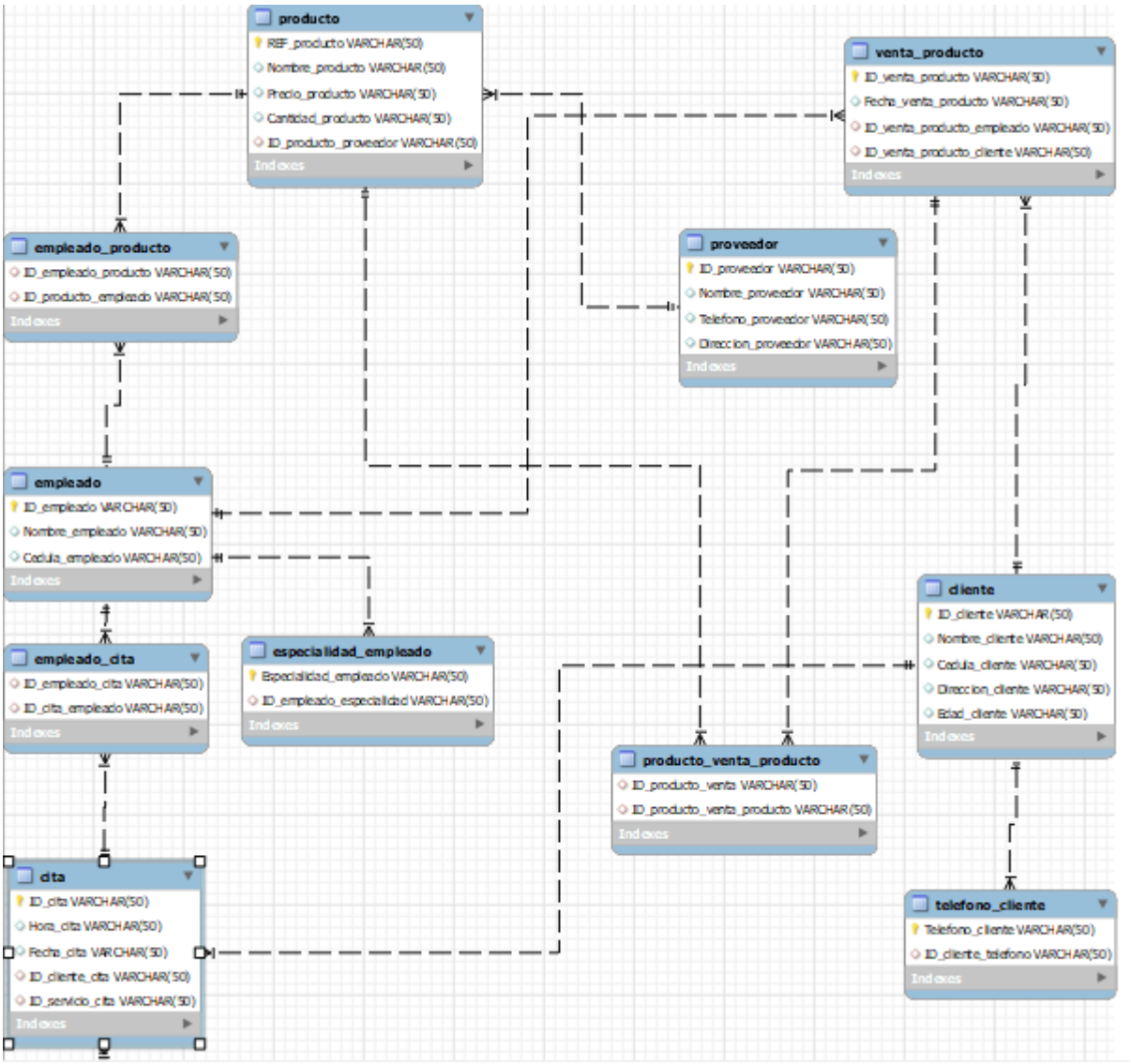


- Un producto puede ser intercambiado en una o varias ventas.
- Una venta puede intercambiar uno o varios productos.



- Un producto puede ser proveído por un proveedor.
- Un proveedor puede proveer uno o varios productos.

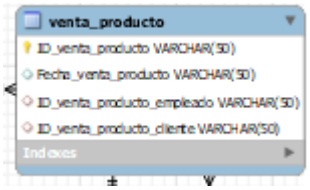
2. Se crea modelo relacional y queda de esta forma:



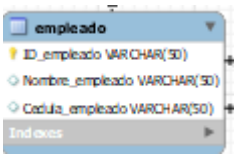
Las tablas del modelo relacional son:

- Producto

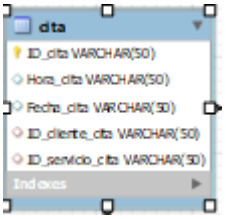
- Venta producto



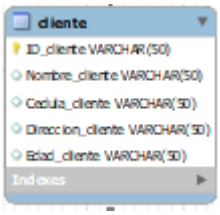
- Empleado



- Cita



- Cliente

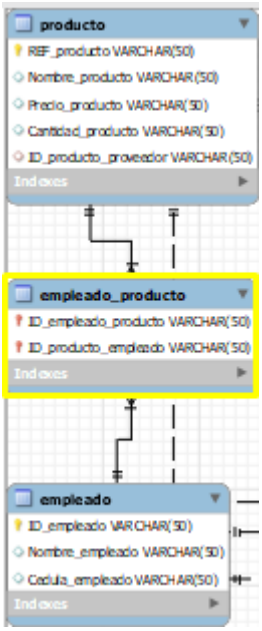


- Proveedor

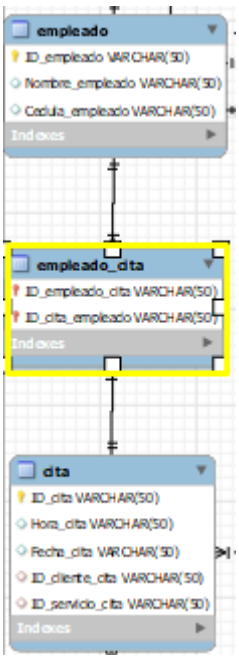
- Historial servicio

Tomando en cuenta las relaciones entre entidades y los atributos multivaluados se crean nuevas tablas y nuevos atributos en las tablas:

- Tenemos la entidad empleado y producto con una cardinalidad de m/n. Por esta razón se crea otra tabla con sus respectivas llaves primarias:

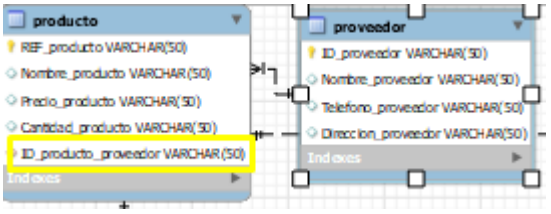


- Tenemos la entidad empleado y cita con una cardinalidad de n/m. Por esta razón se crea otra tabla con las llaves primarias de cada una:

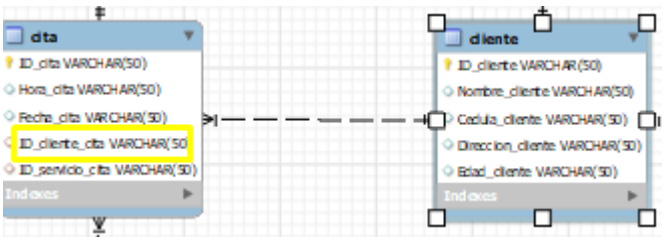


- Tenemos las entidades producto y venta_producto con una cardinalidad n/m. Por esta razón se crea otra tabla con las llaves primarias de cada entidad:

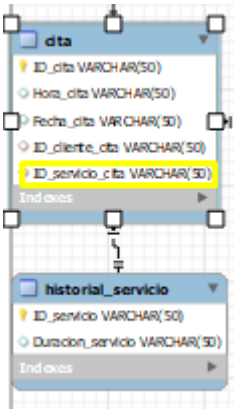
- Tenemos la entidad proveedor y producto con una cardinalidad 1/n. Por esta razón se crea un llave foránea en la entidad producto, referenciada a la llave primaria de la entidad proveedor:



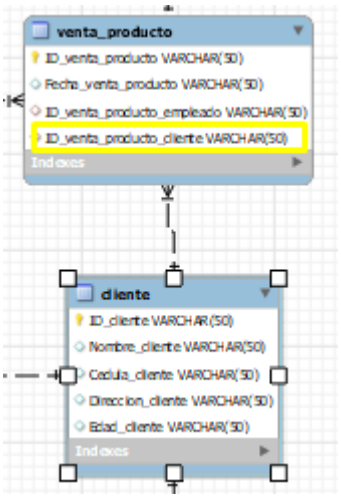
- Tenemos las entidades cita y cliente con una cardinalidad de n/1. Por esta razón se crea una llave foránea en la entidad producto, referenciada a la llave primaria de la entidad cliente:



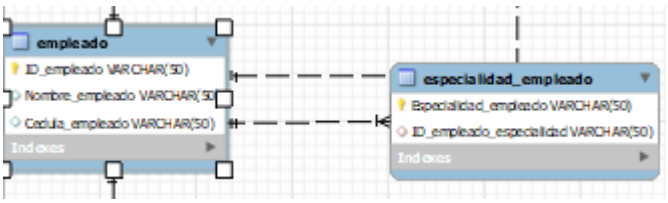
- Tenemos las entidades cita e historial servicio con una cardinalidad de 1/n. Por esta razón se crea una llave foránea en la entidad historial_servicio, referenciada a la llave primaria de la entidad cita:



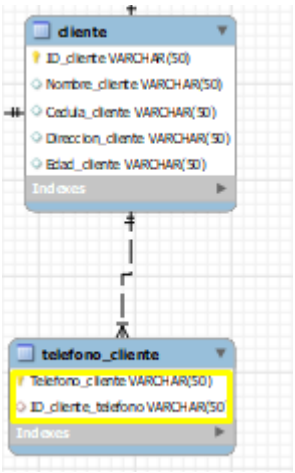
- Tenemos las entidades cliente y venta producto con una cardinalidad de 1/n. Por esta razón crea una llave foránea en la entidad venta producto, referenciada a la llave primaria de la entidad cliente servicio:



- Tenemos la entidad empleado con un atributo multivaluado. Por esta razón se crea otra tabla con una llave foránea referenciada a la llave primaria de la entidad empleado y se crea el atributo multivaluado como llave primaria:



- Tenemos una entidad cliente con un atributo multivaluado. Por esta razón se crea otra tabla con una llave foránea referenciada a la llave primaria de la entidad cliente y se crea el atributo multivaluado como llave primaria:



Normalización:

1. NF:
    - Todos los atributos tienen valores atómicos.
    - No hay atributos multivaluados.
    - Se eliminaron los registros y columnas duplicadas .
    - Se definen claves primarias.

2. NF:
    - Está en 1NF.
    - Todos los valores de las columnas dependen de la llave primaria de la tabla.
    - Las tablas dependen de una única llave primaria.

3. NF:
    - Está en 2NF.
    - Los atributos que no están incluidos en la llave primaria no dependen de la clave.

Se crea BD(Barbería) con sentencias SQL y queda de esta forma:

- Tabla producto:

```sql
create table producto(
REF_producto varchar(50) primary key,
Nombre_producto varchar(50),
Precio_producto varchar(50),
Cantidad_producto varchar(50),
ID_producto_proveedor varchar(50),
foreign key (ID_producto_proveedor) references proveedor(ID_proveedor)
);
```

- Tabla empleado:

```sql
create table empleado(
ID_empleado varchar(50) primary key,
Nombre_empleado varchar(50),
Cedula_empleado varchar(50)
);
```

- Tabla producto_venta_producto: Se crea tabla por la cardinalidad n/m entre la entidad producto y venta_producto.

```sql
create table producto_venta_producto(
ID_producto_venta varchar(50),
ID_producto_venta_producto varchar(50),
foreign key (ID_producto_venta) references producto(REF_producto),
foreign key (ID_producto_venta_producto) references venta_producto(ID_venta_producto)
);
```

- Tabla empleado_producto: Se crea tabla por la cardinalidad n/m entre la entidad producto y empleado.

```
create table empleado_producto(
ID_empleado_producto varchar(50),
ID_producto_empleado varchar(50),
foreign key (ID_empleado_producto) references empleado(ID_empleado),
foreign key (ID_producto_empleado) references producto(REF_producto)
);
```

- Tabla especialidad_cliente : Se crea por el atributo multivaluado especialidad en la entidad cliente.

```
create table especialidad_empleado(
Especialidad_empleado varchar(50) primary key,
ID_empleado_especialidad varchar(50),
foreign key (ID_empleado_especialidad) references empleado(ID_empleado)
);
```

- Tabla cliente:

```
create table cliente(
ID_cliente varchar(50) primary key,
Nombre_cliente varchar(50),
Cedula_cliente varchar(50),
Direccion_cliente varchar(50),
Edad_cliente varchar(50)
);
```

- Tabla telefono_cliente: Se crea tabla por el atributo multivaluado teléfono en la entidad cliente.

```
create table telefono_cliente(
Telefono_cliente varchar(50) primary key,
ID_cliente_telefono varchar(50),
foreign key (ID_cliente_telefono) references cliente(ID_cliente)
);
```

- Tabla cita:

```
create table cita(
ID_cita varchar(50) primary key,
Hora_cita varchar(50),
Fecha_cita varchar(50),
ID_cliente_cita varchar(50),
foreign key (ID_cliente_cita) references cliente(ID_cliente)
);
```

- Tabla empleado_cita: Se crea tabla por la cardinalidad de n/m de la entidad empleado y cita:

```
create table empleado_cita(
ID_empleado_cita varchar(50),
ID_cita_empleado varchar(50),
foreign key (ID_empleado_cita) references empleado(ID_empleado),
foreign key (ID_cita_empleado) references cita(ID_cita)
);
```

- Tabla historial_servicio:

```
create table historial_servicio(
ID_servicio varchar(50) primary key,
Duracion_servicio varchar(50),
ID_servicio_cita varchar(50),
foreign key (ID_servicio_cita) references cita(ID_cita)
);
```

- Tabla proveedor:

```
create table proveedor(
ID_proveedor varchar(50) primary key,
Nombre_proveedor varchar(50),
Telefono_proveedor varchar(50),
Direccion_proveedor varchar(50)
);
```

- Tabla venta producto:

```
create table venta_producto(
ID_venta_producto varchar(50) primary key,
Fecha_venta_producto varchar(50),
ID_venta_producto_empleado varchar(50),
ID_venta_producto_cliente varchar(50),
foreign key (ID_venta_producto_empleado) references empleado(ID_empleado),
foreign key (ID_venta_producto_cliente) references cliente(ID_cliente)
);
```

Se realizan 10 consultan a la BD barbería:

- Esta consulta me muestra los clientes de la barbería:

```
3 •  select * from cliente;
4
5
```

sult Grid | Filter Rows: | Edit: | Export/

ID_cliente	Nombre_cliente	Cedula_cliente	Direccion_cliente	Edad_cliente
1	Efrain	1144046696	cr50	31
NULL	NULL	NULL	NULL	NULL

- Esta consulta me muestra los empleados de la barbería:

```
9    #2
10 •  select * from empleado;
11
```

Result Grid | Filter Rows: | Edit:

ID_empleado	Nombre_empleado	Cedula_empleado
1	Andres	12345667
NULL	NULL	NULL

- Esta consulta me muestra los proveedores:

```
12    #3
13 •  select * from proveedor;
```

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

ID_proveedor	Nombre_proveedor	Telefono_proveedor	Direccion_proveedor
1	Ego	4325454	cr50
NULL	NULL	NULL	NULL

- Esta consulta me muestra las citas que estas pendientes:

```
14    #4
15 •  select * from cita;
```

Result Grid | Filter Rows: | E

ID_cita	Hora_cita	Fecha_cita	ID_cliente_cita
1	2:00	20/02/2023	1
NULL	NULL	NULL	NULL

- Esta consulta me muestra los productos disponibles:

```
17 •  select * from producto;
```

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

REF_producto	Nombre_producto	Precio_producto	Cantidad_producto	ID_producto_proveedor
1	gel	20000	50	1
NULL	NULL	NULL	NULL	NULL

- En esta consulta me muestra la especialidad de los empleados:

```
18    #6
19 •  select * from especialidad_empleado;
```

Result Grid | Filter Rows: | Edit:

Especialidad_empleado	ID_empleado_especialidad
Barbero, peluquero	1
NULL	NULL

- En esta consulta me muestra los datos de la cita y el nombre del empleado asignado para la cita:

```
#7
select ID_cita,Hora_cita,Fecha_cita,ID_cliente_cita, Nombre_empleado from cita
join empleado_cita on ID_cita_empleado = ID_cita
join empleado on ID_empleado = ID_empleado_cita;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ID_cita	Hora_cita	Fecha_cita	ID_cliente_cita	Nombre_empleado
1	2:00	20/02/2023	1	Andres

- En esta consulta me muestra el teléfono y el cliente que le pertenece:

```
#8
select Telefono_cliente, Nombre_cliente from telefono_cliente
join cliente on ID_cliente_telefono = ID_cliente;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Telefono_cliente	Nombre_cliente
23242453	Efrain

- En esta consulta me muestra el nombre del producto y la fecha en que se vendió:

```
#9
select Nombre_producto, Fecha_venta_producto from producto
join producto_venta_producto on ID_producto_venta = REF_producto
join venta_producto on  ID_venta_producto = ID_producto_venta_producto;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Nombre_producto	Fecha_venta_producto
gel	20/2/2023

- En esta consulta me muestra el historial de servicio:

```
#10
select * from historial_servicio;
```

esult Grid | Filter Rows: | Ec

ID_servicio	Duracion_servicio	ID_servicio_cita
1	2:00	1
2	2:00	1
NULL	NULL	NULL

Se realizan las 4 vistas en la BD barbería:

- En esta vista se visualiza el nombre del empleado, el producto que vendió y el cliente que compró el producto.

```
#1
create view produtos_vendidos_empleado as
select Nombre_empleado, Nombre_cliente, Nombre_producto  from venta_producto
join empleado on ID_empleado = ID_venta_producto
join cliente on ID_cliente = ID_venta_producto_cliente
join producto on ID_venta_producto = REF_producto;
select * from produtos_vendidos_empleado;
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

Nombre_empleado	Nombre_cliente	Nombre_producto
Andres	Efrain	gel

- En esta vista se visualiza una tabla con el nombre del cliente, nombre del empleado, fecha y hora de la cita.

```
#2
create view cita_servicio as
select nombre_cliente, Nombre_empleado, Fecha_cita, Hora_cita from cliente
join cita on ID_cliente_cita = ID_cliente
join empleado_cita on ID_cita_empleado = ID_cita
join empleado on ID_empleado_cita = ID_empleado;
select * from cita_servicio;
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

nombre_cliente	Nombre_empleado	Fecha_cita	Hora_cita
Efrain	Andres	20/02/2023	2:00

- En esta vista se visualiza el nombre del proveedor y el producto que vende.

```
#3
create view Proveedor_productoo as
select Nombre_proveedor, Nombre_producto from proveedor
join producto on ID_producto_proveedor = ID_proveedor;
select * from Proveedor_productoo;
```

sult Grid | Filter Rows: | Export: | Wrap Cell Content:

Nombre_proveedor	Nombre_producto
Ego	gel

- En esta vista se visualiza el nombre del empleado y el producto que gatos en una cita.

```
#4
create view empleado_producto_utilizado as
select Nombre_empleado, Nombre_producto from empleado
join empleado_producto on ID_empleado_producto = ID_empleado
join producto on REF_producto = ID_producto_empleado;
select * from empleado_producto_utilizado;
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

Nombre_empleado	Nombre_producto
Andres	gel

Se realizan 4 procedimientos almacenados:

- Este procedimiento permite crear un registro de un cliente en la tabla cliente con todos sus atributos. Se hace un llamado del procedimiento y se agregan los datos pertinentes para agregar un nuevo cliente a la tabla.

```
#1
delimiter //
create procedure agregar_registro_cliente(in ID_cliente_nuevo varchar(15),in Nombre_cliente_nuevo  varchar(
in Cedula_cliente_nuevo  varchar(50),
in Direccion_cliente_nuevo  varchar(50),
in Edad_cliente_nuevo  varchar(50))
begin
select * from cliente;
insert into cliente  values (ID_cliente_nuevo, Nombre_cliente_nuevo,Cedula_cliente_nuevo,
Direccion_cliente_nuevo,
Edad_cliente_nuevo);
end//
delimiter ;
```

```
15 •  call barberia.agregar_registro_cliente(5, 'Juna', '123456', 'cr60', '45');
16    #2
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ID_cliente	Nombre_cliente	Cedula_cliente	Direccion_cliente	Edad_cliente
1	cristian	1234567	cr67	45

- Este procedimiento permite eliminar un registro de la tabla cliente. Se hace un llamado del procedimiento pasandole un id para eliminar el registro.

```
8 •  create procedure eliminar_registro_cliente(in ID_cliente_nuevo varchar(15))
9    begin
0    select * from cliente;
1    delete from  cliente where ID_cliente = ID_cliente_nuevo;
2    end//
3    delimiter ;
4    call eliminar_registro_cliente(5)
5    #3
```

sult Grid | Filter Rows: | Export: | Wrap Cell Content:

ID_cliente	Nombre_cliente	Cedula_cliente	Direccion_cliente	Edad_cliente
1	cristian	1234567	cr67	45
5	Juna	123456	cr60	45

- Este procedimiento permite actualizar un registro de la tabla cliente. Se hace un llama del procedimiento. Se le pasa un id y los datos que necesita para hacer la actualización.

```
#3
delimiter //
create procedure actualizar_registro_cliente(in ID_cliente_nuevo varchar(15),in Nombre_cliente_nuevo  varch
in Cedula_cliente_nuevo  varchar(50),
in Direccion_cliente_nuevo  varchar(50),
in Edad_cliente_nuevo  varchar(50))
begin
select * from cliente;
update cliente set Nombre_cliente = Nombre_cliente_nuevo,
Cedula_cliente = Cedula_cliente_nuevo,
Direccion_cliente =  Direccion_cliente_nuevo,
Edad_cliente = Edad_cliente_nuevo  where ID_cliente = ID_cliente_nuevo;
end//
delimiter ;
```

```
call actualizar_registro_cliente(1,'Roman', 1234567, 'cr67', 45);
#4
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

ID_cliente	Nombre_cliente	Cedula_cliente	Direccion_cliente	Edad_cliente
1	cristian	1234567	cr67	45

- Este procedimiento permite consultar la información de un registro. Haciendo el llamado del procedimiento se le pasa un id para que nos muestre el registro.

```
delimiter //
create procedure consultar_registro_cliente(in ID_cliete_nuevo varchar(15))
begin
select * from cliente where ID_cliente = ID_cliete_nuevo;
end//
delimiter ;
call consultar_registro_cliente(1);
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

ID_cliente	Nombre_cliente	Cedula_cliente	Direccion_cliente	Edad_cliente
1	Roman	1234567	cr67	45

Se realizan los 4 trigger en la BD barbería pero antes que todo, se crea una tabla llamada registro_cambio donde se visualiza el nombre del usuario, un mensaje y la fecha en que se realizó un cambio, como agregar, eliminar o actualizar información en la tabla cliente.

```
create table registro_cambios(
Nombre_usuario varchar(50),
Respuesta varchar(50),
Fecha datetime default current_timestamp
)
```

- Este trigger hace una alerta cuando un usuario crea un registro en la tabla cliente.

```
delimiter //
create trigger registro_cliente after insert on cliente
for each row
begin
    insert into registro_cambios values (user(),'se agrego un registro registro',now());
end//
delimiter ;
```

El registro se ve de esta forma.

	Nombre_usuario	Respuesta	Fecha
▶	root@localhost	se agrego un registro registro	2023-02-16 23:30:31

- Este trigger hace una alerta cuando un usuario elimina un registro en la tabla cliente.

```
delimiter //
create trigger registro_cliente_eliminado after delete on cliente
for each row
begin
    insert into registro_cambios values (user(),'se elimino un  registro',now());
end//
delimiter //
```

El registro se ve de esta forma:

Nombre_usuario	Respuesta	Fecha
root@localhost	se agrego un registro registro	2023-02-16 23:30:31
root@localhost	se elimino un registro	2023-02-16 23:32:28

- Este trigger hace una alerta cuando un usuario actualiza un registro en la tabla cliente.

```
create trigger registro_cliente_actualizado after update on cliente
for each row
begin
    insert into registro_cambios values (user(),'se actualizo un  registro',now());
end//
delimiter ;
```

El registro se ve de esta forma:

Nombre_usuario	Respuesta	Fecha
root@localhost	se agrego un registro registro	2023-02-16 23:30:31
root@localhost	se elimino un registro	2023-02-16 23:32:28
root@localhost	se actualizo un registro	2023-02-16 23:34:40
root@localhost	se actualizo un registro	2023-02-16 23:34:46

- Este trigger hace una alerta cuando un usuario crea un registro en la tabla empleado.

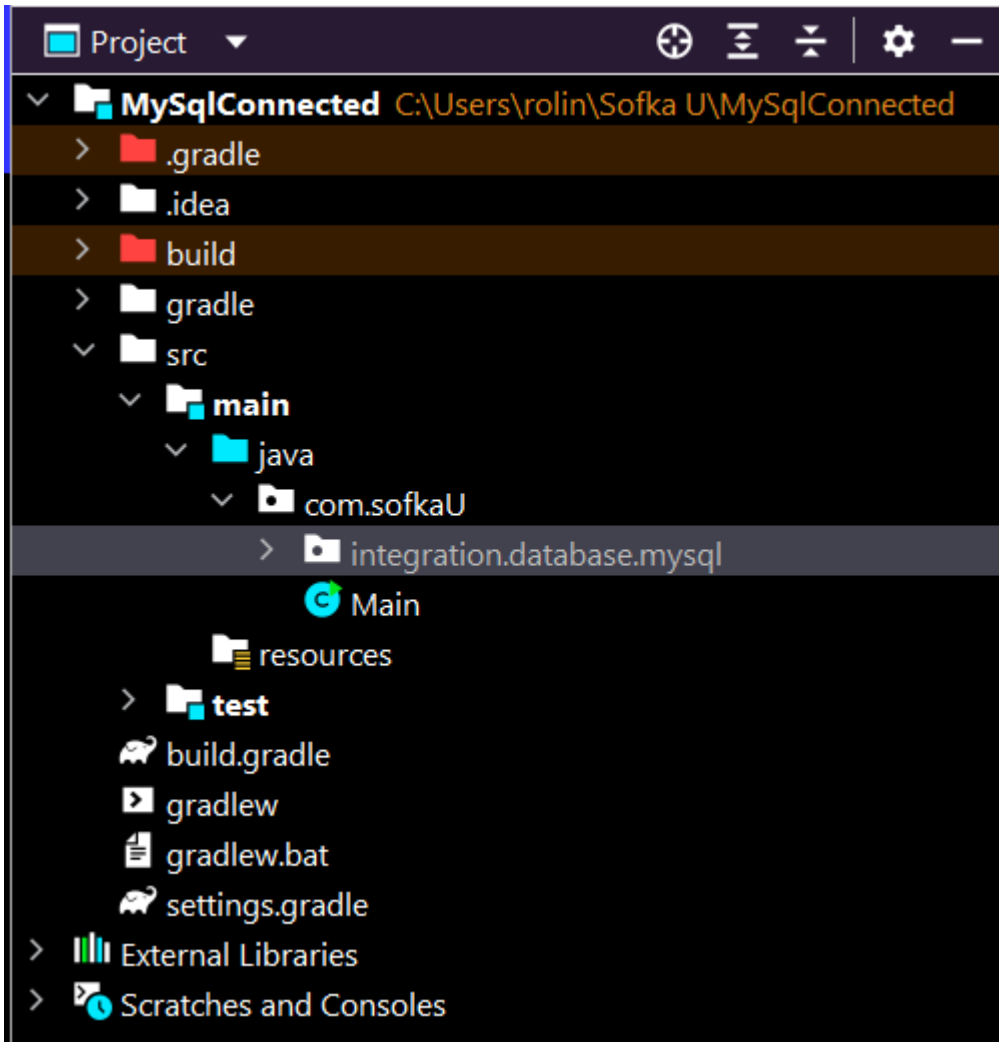
```
delimiter //
create trigger registro_empleado after insert on empleado
for each row
begin
    insert into registro_cambios values (user(),'se agrego un registro a empleado',now());
end//
delimiter ;
```

El registro se ve de este forma:

Nombre_usuario	Respuesta	Fecha
root@localhost	se agrego un registro registro	2023-02-16 23:30:31
root@localhost	se elimino un registro	2023-02-16 23:32:28
root@localhost	se actualizo un registro	2023-02-16 23:34:40
root@localhost	se actualizo un registro	2023-02-16 23:34:46
root@localhost	se agrego un registro a empleado	2023-02-16 23:42:22

Se hace la población de BD por medio de JAVA:

- Lo primero que se hace, es crear un proyecto con GRADLE y JAVA 11 en intelliy que se ve de esta forma:



- Se descargan las dependencias de Mysql en el proyecto:

```
dependencies {
    implementation 'com.lambdaisland:faker:0.2.8'
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.8.1'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.8.1'
    // https://mvnrepository.com/artifact/com.mysql/mysql-connector-j
    implementation 'com.mysql:mysql-connector-j:8.0.32'
}
```

- Se crea la clase connection para hacer conexión a la BD barbería, en donde se establecen los atributos estáticos para el username, URL Y Password que no van a cambiar en el ciclo de vida de la aplicación:

```
public class Connetion {   Complexity is 3 Everything is cool!
    1 usage
    private static final String URL = "jdbc:mysql://localhost:3306/barberia";
    1 usage
    private static final String username = "root";
    1 usage
    private static final String password = "onpjd9bf";
```

- En la clase connection también se crean dos atributos estáticos muy importantes que son connection que va a utilizar los atributos antes mencionados para hacer la conexión. y statement que recibe la sentencia SQL para hacer las consultas:

```
private Connection connection;

3 usages
private Statement statement;
```

- Se crea el getter de statement para poder utilizarlo en otra clases y pasarle las sentencias SQL:

```
public Statement getStatement() {
    return statement;
}
```

- Se crea el método sqlOperation, que es el encargado de hacer la conexión cada vez que se haga un petición a la BD barbería por medio de una sentencia o statement:

```
public void sqlOperation() throws SQLException {
    connection = DriverManager.getConnection(URL, username, password);
    statement = connection.createStatement();
}
```

- Se crea el método close, que permite cerrar la conexión en cada petición:

```
public void close() throws SQLException {
    connection.close();
    statement.close();
}
```

- Se crea la clase Customer para ingresar los datos a la tabla cliente en BD. Se hace un instancia de la clase connection en la clase Customer para poder utilizar los métodos para hacer la conexión. Se crea un método createCustomer qué va hacer 50 iteraciones por medio de un For, pasándole datos aleatorios utilizando la clase Faker y las variables locales que guardan esta información por cada iteración. Se llama a la clase connection para hacer la conexión con el método sqlOperation y también se llama el método getter para poder pasar la sentencia sql con las variables locales por medio de un format. Y por último se cierra la conexión con el método close.

```
public class Customer {  Complexity is 4 Everything is cool!
    3 usages
    Connetion connetion = new Connetion();


    no usages
    public void createCustomer() throws SQLException {  Complexity is 3 Everything is cool!
        Faker faker = new Faker();

        for (int i = 1; i <= 50; i++) {
            String customerid = String.valueOf(i);
            String nombre = faker.name().firstName();
            String cedula = faker.idNumber().valid();
            String direccion = faker.address().city();
            String edad = faker.number().digit();
            connetion.sqlOperation();
            connetion.getStatement().executeUpdate(String.format("insert into cliente VALUES ('%s', '%s', '%s', '%s', '%s')", customerid, nombre, cedula
        }
        connetion.close();
    }
}
```

- Se crea la clase Employee para ingresar los datos a la tabla empleado en BD. Se hace un instancia de la clase connection en la clase Employee para poder utilizar los métodos para hacer la conexión. Se crea un método createEmployee qué va hacer 50 iteraciones por medio de un For, pasándole datos aleatorios utilizando la clase Faker y las variables locales que guardan esta información por cada iteración. Se llama a la clase connection para hacer la conexión con el método sqlOperation y también se llama el método getter para poder pasar la sentencia sql con las variables locales por medio de un format. Y por último se cierra la conexión con el método close:

```java
public class Employee {  Complexity is 4 Everything is cool!
    3 usages
    Connetion connetion = new Connetion();

    no usages
    public void createEmployee() throws SQLException {  Complexity is 3 Everything is cool!
        Faker faker = new Faker();

        for (int i = 0; i <= 50; i++) {
            String ID = String.valueOf(i);
            String nombre = faker.name().firstName();
            String cedula = faker.idNumber().valid();
            connetion.sqlOperation();
            connetion.getStatement().executeUpdate(String.format("insert into empleado VALUES ('%s', '%s', '%s')", ID, nombre, cedula))
        }
        connetion.close();
    }
}
```

- Se crea la clase Product para ingresar los datos a la tabla producto en BD. Se hace un instancia de la clase connection en la clase Product para poder utilizar los métodos para hacer la conexión. Se crea un método createProduct qué va hacer 50 iteraciones por medio de un For, pasándole datos aleatorios utilizando la clase Faker y las variables locales que guardan esta información por cada iteración. Se llama a la clase connection para hacer la conexión con el método sqlOperation y también se llama el método getter para poder pasar la sentencia sql con las variables locales por medio de un format. Y por último se cierra la conexión con el método close.

```java
public class Product {  Complexity is 4 Everything is cool!

    3 usages
    Connetion connetion = new Connetion();

    no usages
    public void createProduct() throws SQLException {  Complexity is 3 Everything is cool!
        Faker faker = new Faker();

        for (int i = 1; i <= 50; i++) {
            String id = String.valueOf(i);
            String nombre = faker.beer().name();
            int precioProducto = faker.number().numberBetween(1000,100000);
            int cantidadProducto = faker.number().randomDigit();
            connetion.sqlOperation();
            connetion.getStatement().executeUpdate(String.format("insert into producto VALUES ('%s', '%s', '%s', '%s', '%s')", id, nombre, precioPro
        }
        connetion.close();
    }
}
```

- Se crea la clase Appointment para ingresar los datos a la tabla cita en la BD. Se hace un instancia de la clase connection en la clase Appointment para poder utilizar los métodos para hacer la conexión. Se crea un método createAppointment qué va hacer 50 iteraciones por medio de un For, pasándole datos aleatorios utilizando la clase Faker y las variables locales que guardan esta información por cada iteración. Se llama a la clase connection para hacer la conexión con el método sqlOperation y también se llama el método getter para poder pasar la sentencia sql con las variables locales por medio de un format. Y por último se cierra la conexión con el método close:

```java
public class Appointment {  Complexity is 4 Everything is cool!
    3 usages
    Connetion connetion = new Connetion();

    no usages
    public void createAppointment() throws SQLException {  Complexity is 3 Everything is cool!

        Faker faker = new Faker();

        for (int i = 1; i <= 50; i++) {
            String ID = String.valueOf(i);
            String hora = faker.number().digit();
            String fecha = faker.number().digit();
            connetion.sqlOperation();
            connetion.getStatement().executeUpdate(String.format("insert into cita VALUES ('%s', '%s', '%s','%s')", ID, hora, fecha, ID));
        }
        connetion.close();
    }
}
```

- Se crea la clase Employee_Appointment para ingresar los datos a la tabla cita_empleado en la BD. Se hace un instancia de la clase connection en la clase Employee_Appointment para poder utilizar los métodos para hacer la conexión. Se crea un método create_Employee_Appointment qué va hacer 50 iteraciones por medio de un For, pasándole datos aleatorios utilizando la clase Faker

y las variables locales que guardan esta información por cada iteración. Se llama a la clase connection para hacer la conexión con el método sqlOperation y también se llama el método getter para poder pasar la sentencia sql con las variables locales por medio de un format. Y por último se cierra la conexión con el método close:

```
public class Employee_Appointment {  Complexity is 4 Everything is cool!

    3 usages
    Connetion connetion = new Connetion();

    no usages
    public void createEmployee_Appointment() throws SQLException {  Complexity is 3 Everything is cool!

        Faker faker = new Faker();

        for (int i = 1; i <= 50; i++) {
            String ID = String.valueOf(i);
            connetion.sqlOperation();
            connetion.getStatement().executeUpdate(String.format("insert into empleado_cita      VALUES ('%s', '%s')", ID, ID));
        }
        connetion.close();
    }
}
```

- Se crea la clase Supplier para ingresar los datos a la tabla proveedor en la BD. Se hace un instancia de la clase connection en la clase Supplier para poder utilizar los métodos para hacer la conexión. Se crea un método create_Supplier qué va hacer 50 iteraciones por medio de un For, pasándole datos aleatorios utilizando la clase Faker y las variables locales que guardan esta información por cada iteración. Se llama a la clase connection para hacer la conexión con el método sqlOperation y también se llama el método getter para poder pasar la sentencia sql con las variables locales por medio de un format. Y por último se cierra la conexión con el método close:

```
public class Supplier {  Complexity is 4 Everything is cool!

    3 usages
    Connetion connetion = new Connetion();

    no usages
    public void createSupplier() throws SQLException {  Complexity is 3 Everything is cool!
        Faker faker = new Faker();

        for (int i = 1; i <= 50; i++) {
            String id = String.valueOf(i);
            String nombre = faker.name().firstName();
            String telefono = faker.phoneNumber().phoneNumber();
            String direccion = faker.address().city();
            connetion.sqlOperation();
            connetion.getStatement().executeUpdate(String.format("insert into proveedor VALUES ('%s', '%s', '%s', '%s')", id, nombre, telefono , di
        }
        connetion.close();
    }
}
```

- Se crea la clase Service_History para ingresar los datos a la tabla historial-servicio en la BD. Se hace un instancia de la clase connection en la clase Service_History para poder utilizar los métodos para hacer la conexión. Se crea un método create_Service_Histrory qué va hacer 50 iteraciones por medio de un For, pasándole datos aleatorios utilizando la clase Faker y las variables locales que guardan esta información por cada iteración. Se llama a la clase connection para hacer la conexión con el método sqlOperation y también se llama el método getter para poder pasar la sentencia sql con las variables locales por medio de un format. Y por último se cierra la conexión con el método close:

```
no usages
public class Service_history {  Complexity is 4 Everything is cool!
    3 usages
    Connetion connetion = new Connetion();

    no usages
    public void createService() throws SQLException {  Complexity is 3 Everything is cool!

        Faker faker = new Faker();

        for (int i = 1; i <= 50; i++) {
            String ID = String.valueOf(i);
            String hora = String.valueOf(faker.number().randomDigit());
            connetion.sqlOperation();
            connetion.getStatement().executeUpdate(String.format("insert into historial_servicio VALUES ('%s', '%s', '%s')", ID, hora, ID));
        }
        connetion.close();
    }
}
```

- Se crea la clase Sale_Product para ingresar los datos a la tabla venta_producto en la BD. Se hace un instancia de la clase connection en la clase Sale_Product para poder utilizar los métodos para hacer la conexión. Se crea un método create_Sale_Product qué va hacer 50 iteraciones por medio de un For, pasándole datos aleatorios utilizando la clase Faker y las variables locales que guardan esta información por cada iteración. Se llama a la clase connection para hacer la conexión con el método sqlOperation y también se llama el método getter para poder pasar la sentencia sql con las variables locales por medio de un format. Y por último se cierra la conexión con el método close:

```java
public class Sale_Product {  Complexity is 4 Everything is cool!

    3 usages
    Connetion connetion = new Connetion();


    no usages
    public void createSale_Product() throws SQLException {  Complexity is 3 Everything is cool!
        Faker faker = new Faker();

        for (int i = 1; i <= 50; i++) {
            String id = String.valueOf(i);
            String nombre = faker.beer().name();
            DateAndTime fecha_venta = faker.date();
            connetion.sqlOperation();
            connetion.getStatement().executeUpdate(String.format("insert into venta_producto VALUES ( '%s', '%s', '%s', '%s')", id, fecha_venta , id, id
        }
        connetion.close();
    }
}
```

- Se crea la clase Phone_Customer para ingresar los datos a la tabla telefono_cliente en la BD. Se hace un instancia de la clase connection en la clase Phone_Customer para poder utilizar los métodos para hacer la conexión. Se crea un método create_Phone_Customer qué va hacer 50 iteraciones por medio de un For, pasándole datos aleatorios utilizando la clase Faker y las variables locales que guardan esta información por cada iteración. Se llama a la clase connection para hacer la conexión con el método sqlOperation y también se llama el método getter para poder pasar la sentencia sql con las variables locales por medio de un format. Y por último se cierra la conexión con el método close:

```java
public class Phone_Customer {  Complexity is 4 Everything is cool!
    3 usages
    Connetion connetion = new Connetion();


    no usages
    public void create_Telefono_cliente() throws SQLException {  Complexity is 3 Everything is cool!
        Faker faker = new Faker();

        for (int i = 1; i <= 50; i++) {
            String id = String.valueOf(i);
            String telefono_cliente = faker.phoneNumber().cellPhone();
            connetion.sqlOperation();
            connetion.getStatement().executeUpdate(String.format("insert into telefono_cliente VALUES ( '%s', '%s')", telefono_cliente ,id));
        }
        connetion.close();
    }
}
```

- Para implementar todos las clases y que se hagan los registros en la tablas, se hace una instancia de cada clase en el main y se utiliza el método create asignado a cada clase por medio de la instancia y los registros se hacen automatico corriendo la aplicación:

```java
public static void main(String[] args) throws SQLException {
    Customer customer = new Customer();
    customer.createCustomer();

    Employee employee = new Employee();
    employee.createEmployee();

    Appointment appointment = new Appointment();
    appointment.createAppointment();

    Service_history serviceHistory = new Service_history();
    serviceHistory.createService();

    Supplier supplier = new Supplier();
    supplier.createSupplier();

    Product product = new Product();
    product.createProduct();

    Employee_Appointment employee_appointment = new Employee_Appointment();
    employee_appointment.createEmployee_Appointment();

    Sale_Product sale_product = new Sale_Product();
    sale_product.createSale_Product();
```

Se responde la pregunta propuesta en el reto:

- Estoy conforme con el resultado obtenido utilizando una BD relacional, ya que esta nos permite tener un mayor control en la información que ingresa en ella. nos permite una mejor estructura y relación entre tablas, evita la duplicidad de registro y en su mayoría, las BD relacionales son más utilizadas en este tipo de proyectos.Las BD no relacionales se utilizan más que todo para big data y otro tipo de archivos.