



RETO TECNICO BARBERIA

PRESENTADO POR:

ESTIVEN TAPASCO RAMIREZ

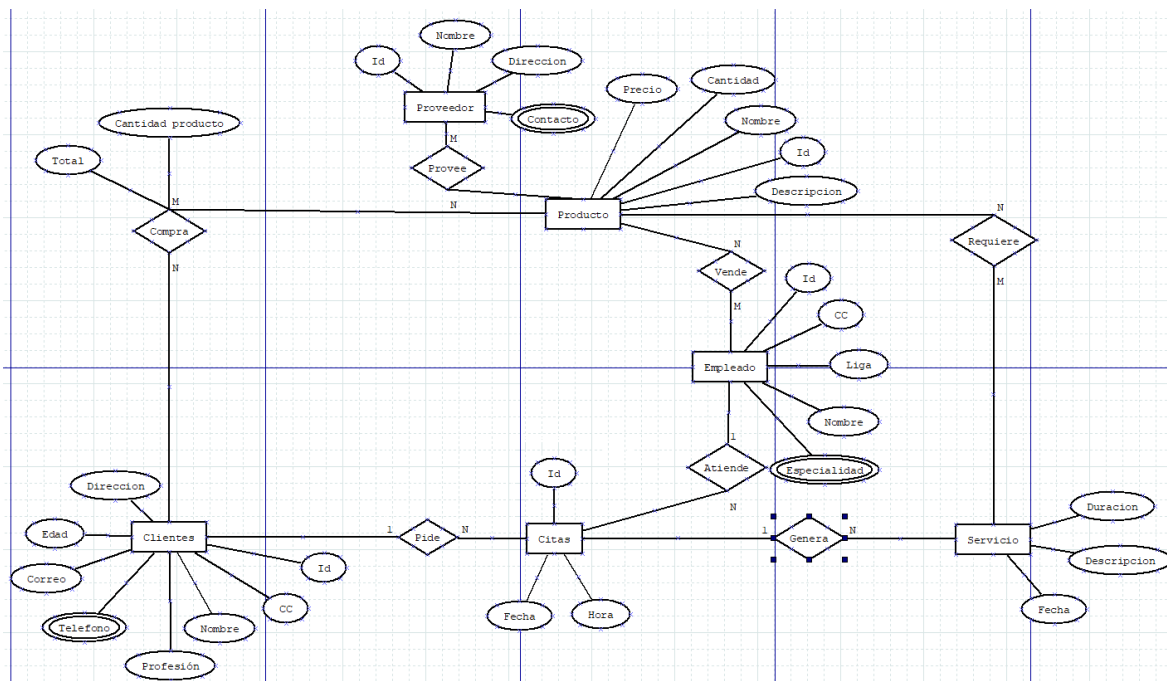
COACH:

JUAN ESTEBAN PINEDA

17/02/2023

Reto técnico bases de datos (Barbería)

Modelo entidad relación:



Algunos de los registros y requisitos de la barbería se trabajarán después como vistas.

Cardinalidades:

1-1:

- No se encontró este tipo de cardinalidades en el modelo.

1-N:

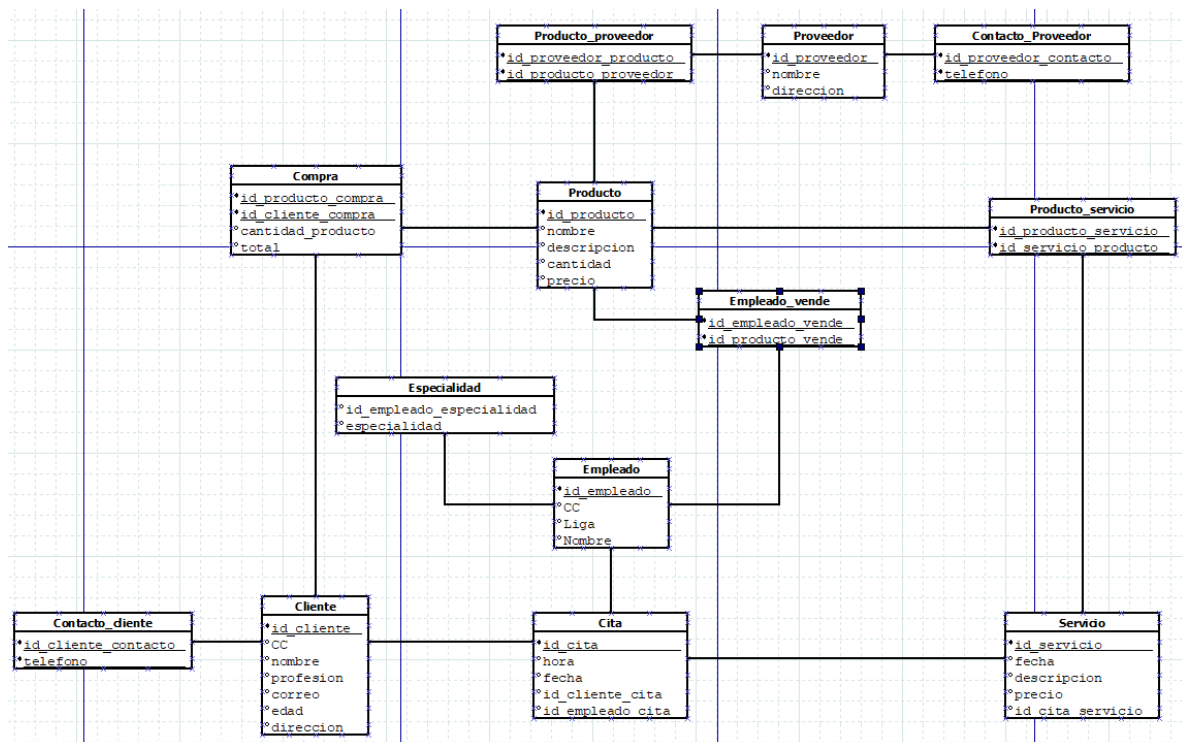
- (Cita - Genera - Servicio): En este modelo se considera que una cita puede generar 1 o varios servicios, y que un servicio solo puede ser generado por una cita, es decir, un cliente puede pedir cita para que le corten el pelo y le arreglen las cejas, pero esos dos servicios se verán como únicos para ese cliente.
- (Empleado - Atiende - Cita): En este modelo se considera que un empleado puede atender una o varias citas, pero una cita solo será atendida por un empleado, es decir, si un cliente pide una cita que requiera de varios servicios, deberá esperar la disponibilidad de un empleado que contenga las especialidades necesarias, o en su defecto, pedir varias citas.
- (Cliente - Pide - Cita): En este modelo se considera que un cliente podrá pedir una o varias citas, pero una cita solo podrá ser pedida por un cliente, es decir, la cita será un producto único para cada cliente.

N-M:

- (Cliente - Compra - Producto): Se considera que un cliente puede comprar muchos productos, y un producto puede ser comprado por uno o varios clientes.

- (Proveedor - Provee - Producto): Un proveedor puede proveer uno o varios productos a la barbería, y un producto lo puede proveer uno o varios proveedores, es decir, un shampoo lo pueden tener varias empresas, no es un producto único.
- (Empleado - Vende - Producto): Un empleado puede vender uno o varios productos, y un producto puede ser vendido por uno o varios empleados, es decir, estos productos al no ser únicos con los clientes, tampoco lo serán con los empleados.
- (Servicio - Requiere - Producto): Un servicio podrá requerir de uno o varios productos, y un producto será necesario para uno o varios servicios.

Modelo relacional



Basándose en el modelo entidad relación se generó este modelo relacional.

Transformación:

Relación N:M

- Se crea la tabla "compra" generada por la relación de N:M entre cliente y producto, que guardara la relación entre estas dos tablas.
- Se crea la tabla "producto_proveedor" generada por la relación de N:M entre producto y proveedor, que guardara la relación entre estas dos tablas.
- Se crea la tabla "empleado_vende" generada por la relación de N:M entre producto y empleado, que guardara la relación entre estas dos tablas.
- Se crea la tabla "producto_servicio" generada por la relación de N:M entre producto y servicio, que guardara la relación entre estas dos tablas.

Relación 1:N

- Se crea una llave foránea en la tabla cita, que cree la relación con la tabla cliente.
- Se crea una llave foránea en la tabla cita, que cree la relación con la tabla empleado.
- Se crea una llave foránea en la tabla servicio, que cree la relación con la tabla cita.

Atributos multivaluados

- Se crea la tabla contacto_proveedor para guardar los diferentes números que pueda tener un proveedor.
- Se crea la tabla contacto_cliente para guarda los diferentes números que puede tener un cliente.
- Se crea la tabla especialidad para guardar las diferentes especialidades que pueda tener un empleado.

Normalización:

Primera forma normal:

No se encontró redundancia en el modelo, cada tabla tiene una llave principal y se crearon las tablas necesarias para que cada tabla guarde solo datos que sean de su área, por ende, se llega a la consideración de que el modelo ya está en primera forma normal.

Segunda forma normal:

Todas las tablas están conectadas y cada tabla tiene una llave que almacena la información que le corresponde sin depender de otra tabla, por ende, se llega a la consideración que el modelo ya se encuentra en segunda forma normal.

Tercera forma normal:

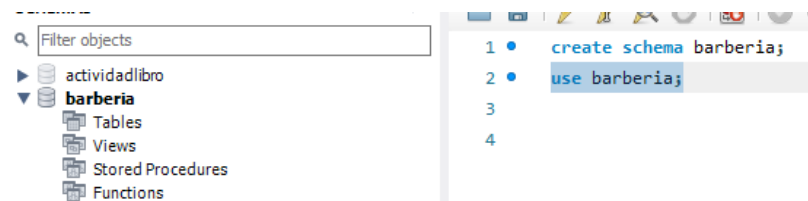
No fue necesario eliminar ningún atributo, ninguno de los atributos que se consideraron incumplen las condiciones para pertenecer a su tabla, por ende, se llega a la conclusión de que el modelo ya está en tercera forma normal.

Creación del modelo mediante comandos SQL

Para tener un orden claro de como se deben crear las tablas, se analiza el modelo relacional y se llega a las siguientes conclusiones:

- Las tablas cliente, empleado, producto y proveedor no poseen llaves foráneas, por ende, deberían ser las tablas candidatas para crear primero.
- La tabla cita debe ser creada después de la tabla cliente y empleado.
- La tabla servicio debe ser creada después de la tabla cita.
- La tabla compra debe ser creada después de la tabla cliente y producto.
- La tabla empleado_vende debe ser creada después de la tabla producto y empleado.
- La tabla producto_proveedor debe ser creada después de la tabla producto y proveedor.
- La tabla especialidad debe ser creada después de la tabla empleado.
- La tabla contacto_proveedor debe ser creada después de la tabla proveedor.
- La tabla contacto_cliente debe ser creada después de la tabla cliente.

Creación del schema “barberia”



Se crea el schema “barberia” y se selecciona para seguir trabajando sobre el.

Tabla cliente y su contacto:

```
2 • use barberia;
3 /*-----Tabla cliente y contacto_cliente -----*/
4 • create table cliente(
5     id_cliente varchar(20) primary key,
6     CC varchar(20),
7     nombre varchar(60),
8     profesion varchar(40),
9     correo varchar(40),
10    edad varchar(3),
11    direccion varchar(40)
12 );
13 • create table contacto_cliente(
14     id_cliente_contacto varchar(20),
15     telefono varchar(20),
16     primary key(id_cliente_contacto,telefono),
17     foreign key(id_cliente_contacto) references cliente(id_cliente)
18 );
```

Tabla empleado y su especialidad:

```
31 /*-----Tabla empleado y especialidad -----*/
32 • create table empleado(
33     id_empleado varchar(20) primary key,
34     CC varchar(60),
35     nombre varchar(40),
36     liga double
37 );
38 • create table especialidad(
39     id_empleado_especialidad varchar(20),
40     especialidad varchar(20),
41     primary key(id_empleado_especialidad,especialidad),
42     foreign key(id_empleado_especialidad) references empleado(id_empleado)
43 );
```

Tabla proveedor y su contacto:

```
18 );
19 /*-----Tabla proveedor y contacto_proveedor -----*/
20 • create table proveedor(
21     id_proveedor varchar(20) primary key,
22     nombre varchar(60),
23     direccion varchar(40)
24 );
25 • create table contacto_proveedor(
26     id_proveedor_contacto varchar(20),
27     telefono varchar(20),
28     primary key(id_proveedor_contacto,telefono),
29     foreign key(id_proveedor_contacto) references proveedor(id_proveedor)
30 );
31
```

Tabla producto:

```
44 /*-----Tabla producto -----*/
45 • create table producto(
46     id_producto varchar(20) primary key,
47     nombre varchar(20),
48     descripcion varchar(60),
49     cantidad varchar(40),
50     precio varchar(40)
51 );
```

Tabla cita

```
--  
52  /*-----Tabla cita -----*/  
53  create table cita(  
54      id_cita varchar(20) primary key,  
55      hora varchar(20),  
56      fecha varchar(60),  
57      id_cliente_cita varchar(20),  
58      id_empleado_cita varchar(20),  
59      foreign key(id_cliente_cita) references cliente(id_cliente),  
60      foreign key(id_empleado_cita) references empleado(id_empleado)  
61  );
```

Tabla servicio

```
62  /*-----Tabla servicio -----*/  
63  create table servicio(  
64      id_servicio varchar(20) primary key,  
65      fecha varchar(20),  
66      descripcion varchar(20),  
67      precio varchar(20),  
68      id_cita_servicio varchar(20),  
69      foreign key(id_cita_servicio) references cita(id_cita)  
70  );
```

Tabla producto_servicio

```
71  /*-----Tabla producto_servicio -----*/  
72  create table producto_servicio(  
73      id_producto_servicio varchar(20),  
74      id_servicio_producto varchar(20),  
75      primary key(id_producto_servicio,id_servicio_producto),  
76      foreign key(id_producto_servicio) references producto(id_producto),  
77      foreign key(id_servicio_producto) references servicio(id_servicio)  
78  );
```

Tabla producto_proveedor

```
71  /*-----Tabla producto_servicio -----*/  
72  create table producto_servicio(  
73      id_producto_servicio varchar(20),  
74      id_servicio_producto varchar(20),  
75      primary key(id_producto_servicio,id_servicio_producto),  
76      foreign key(id_producto_servicio) references producto(id_producto),  
77      foreign key(id_servicio_producto) references servicio(id_servicio)  
78  );
```

Tabla empleado_vende

```
78  );  
79  /*-----Tabla empleado_vende -----*/  
80  create table empleado_vende(  
81      id_empleado_vende varchar(20),  
82      id_producto_vende varchar(20),  
83      primary key(id_empleado_vende,id_producto_vende),  
84      foreign key(id_empleado_vende) references empleado(id_empleado),  
85      foreign key(id_producto_vende) references producto(id_producto)  
86  );
```

Tabla compra

```
86  );  
87  /*-----Tabla compra -----*/  
88  create table compra(  
89      id_producto_compra varchar(20),  
90      id_cliente_compra varchar(20),  
91      cantidad_producto varchar(5),  
92      total varchar(10),  
93      primary key(id_producto_compra,id_cliente_compra),  
94      foreign key(id_producto_compra) references producto(id_producto),  
95      foreign key(id_cliente_compra) references cliente(id_cliente)  
96  );
```

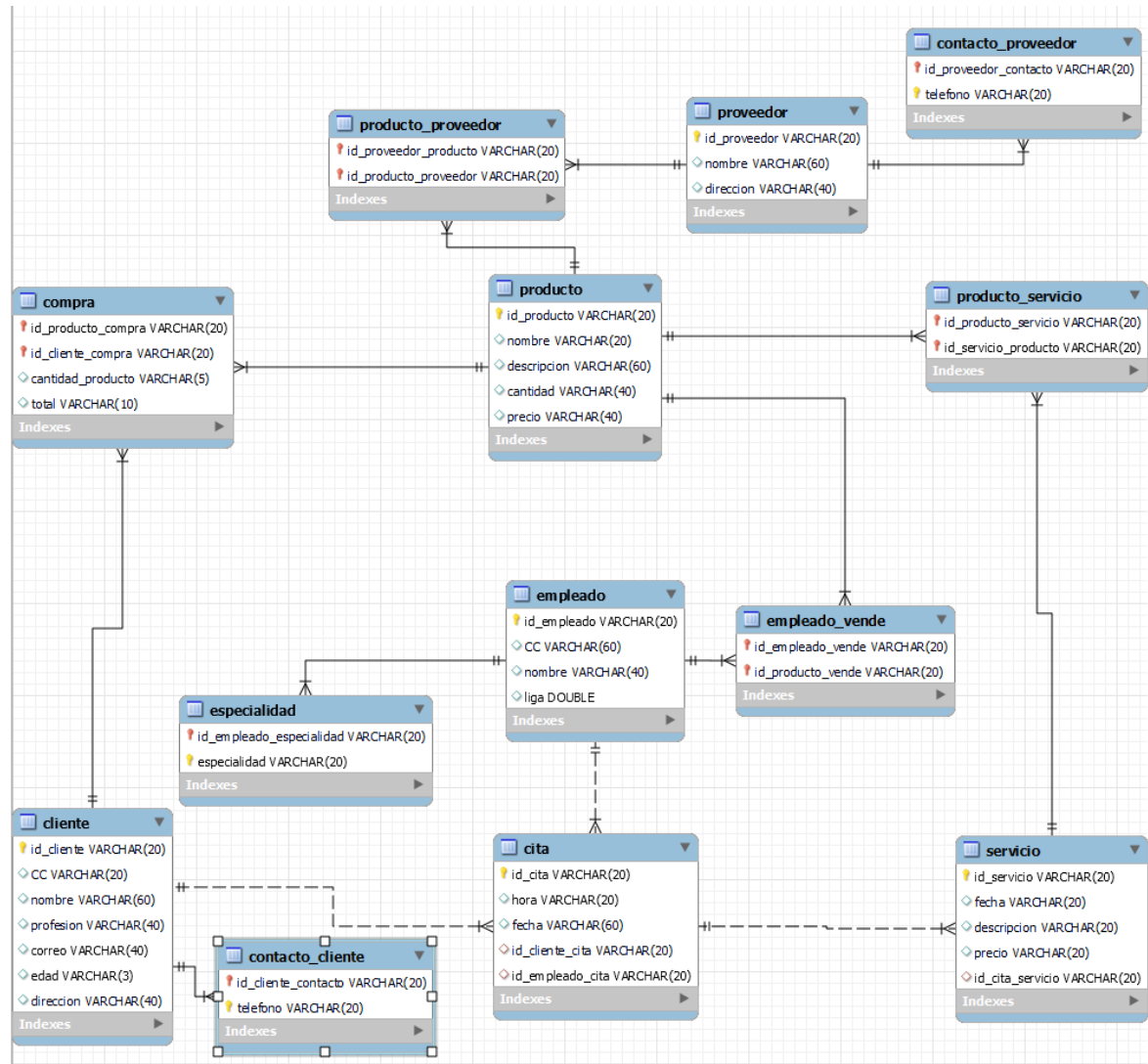
Tabla producto_proveedor

```

97  /*-----Tabla producto_proveedor -----*/
98  create table producto_proveedor(
99      id_proveedor_producto varchar(20),
100     id_producto_proveedor varchar(20),
101     primary key(id_proveedor_producto,id_producto_proveedor),
102     foreign key(id_proveedor_producto) references proveedor(id_proveedor),
103     foreign key(id_producto_proveedor) references producto(id_producto)
104 );

```

Aplicando la opción de reverse engineer de MySQL, se puede observar que el diagrama generado es exactamente igual al modelo relacional planteado inicialmente.



Ahora se crearán registros mediante comandos SQL para realizar pruebas.

Registros

Tabla cliente

```
13 • insert into cliente values
14 ("001","1115456123","Ash Ketchup","Entrenador pokemon","Ash@hotmail.com","12","Calle pueblo # paleta"),
15 ("002","1115456124","Goku Armando","Peleador de MMA","Goku@hotmail.com","27","Namekusei"),
16 ("003","1115456125","Mario Bros","Fontanero","Mario@hotmail.com","20","Reino champion");
17 • select * from cliente;
```

	id_cliente	CC	nombre	profesion	correo	edad	direccion
▶	001	1115456123	Ash Ketchup	Entrenador pokemon	Ash@hotmail.com	12	Calle pueblo # paleta
	002	1115456124	Goku Armando	Peleador de MMA	Goku@hotmail.com	27	Namekusei
	003	1115456125	Mario Bros	Fontanero	Mario@hotmail.com	20	Reino champion
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabla contacto_cliente

```
25 • insert into contacto_cliente values
26 ("001","315789123"),
27 ("002","315789124"),
28 ("003","315789125");
29 • select * from contacto_cliente;
```

	id_cliente_contacto	telefono
▶	001	315789123
	002	315789124
	003	315789125
*	NULL	NULL

Tabla proveedor

```
36 • insert into proveedor values
37 ("101","Proveedor1","Calle falsa 123"),
38 ("102","Proveedor2","Calle falsa 124"),
39 ("103","Proveedor3","Calle falsa 125");
40 • select * from proveedor;
41
```

	id_proveedor	nombre	direccion
▶	101	Proveedor1	Calle falsa 123
	102	Proveedor2	Calle falsa 124
	103	Proveedor3	Calle falsa 125
*	NULL	NULL	NULL

Tabla contacto_proveedor

```
--
48 • insert into contacto_proveedor values
49   ("101","315789129"),
50   ("102","315789128"),
51   ("103","315789127");
52 • select * from contacto_proveedor;
53 /*-----
```

Result Grid

	id_proveedor_contacto	telefono
▶	101	315789129
	102	315789128
	103	315789127
*	NULL	NULL

Tabla empleado

```
60 • insert into empleado values
61   ("1001","1115456111","Empleado1",1000),
62   ("1002","1115456112","Empleado2",2000),
63   ("1003","1115456113","Empleado3",3000);
64 • select * from empleado;
```

Result Grid

	id_empleado	CC	nombre	liga
▶	1001	1115456111	Empleado1	1000
	1002	1115456112	Empleado2	2000
	1003	1115456113	Empleado3	3000
*	NULL	NULL	NULL	NULL

Tabla especialidad

```
72 • insert into especialidad values
73   ("1001","cejas"),
74   ("1002","cabello"),
75   ("1003","piel");
76 • select * from especialidad;
77 /*-----
```

Result Grid

	id_empleado_especialidad	especialidad
▶	1001	cejas
	1002	cabello
	1003	piel
*	NULL	NULL

Tabla producto

```

85 • insert into producto values
86     ("2001","Shampoo anti caspa","Descripcion1","100","50000"),
87     ("2002","Tinte","Descripcion2","100","5000"),
88     ("2003","Prestobarba","Descripcion13","100","6000");
89 • select * from producto;

```

	id_producto	nombre	descripcion	cantidad	precio
▶	2001	Shampoo anti caspa	Descripcion1	100	50000
	2002	Tinte	Descripcion2	100	5000
	2003	Prestobarba	Descripcion13	100	6000
*	NULL	NULL	NULL	NULL	NULL

Tabla cita

```

100 • insert into cita values
101     ("3001","08:30","01/01/91","001","1001"),
102     ("3002","09:30","01/01/92","002","1002"),
103     ("3003","10:30","01/01/93","003","1003");
104 • select * from cita;

```

	id_cita	hora	fecha	id_cliente_cita	id_empleado_cita
▶	3001	08:30	01/01/91	001	1001
	3002	09:30	01/01/92	002	1002
	3003	10:30	01/01/93	003	1003
*	NULL	NULL	NULL	NULL	NULL

Tabla servicio

```

114 • insert into servicio values
115     ("4001","02/02/92","descripcion1","30000","3001"),
116     ("4002","02/02/93","descripcion2","20000","3002"),
117     ("4003","02/02/94","descripcion3","10000","3003");
118 • select * from servicio;

```

	id_servicio	fecha	descripcion	precio	id_cita_servicio
▶	4001	02/02/92	descripcion1	30000	3001
	4002	02/02/93	descripcion2	20000	3002
	4003	02/02/94	descripcion3	10000	3003
*	NULL	NULL	NULL	NULL	NULL

Tabla producto_servicio

```
127 • insert into producto_servicio values
128     ("2001", "4001"),
129     ("2002", "4002"),
130     ("2003", "4003");
131 • select * from producto_servicio;
```

Result Grid | Filter Rows: | Edit:

	id_producto_servicio	id_servicio_producto
▶	2001	4001
	2002	4002
	2003	4003
*	NULL	NULL

Tabla empleado_vende

```
140 • insert into empleado_vende values
141     ("1001", "2001"),
142     ("1002", "2002"),
143     ("1003", "2003");
144 • select * from empleado_vende;
```

Result Grid | Filter Rows: | Edit:

	id_empleado_vende	id_producto_vende
▶	1001	2001
	1002	2002
	1003	2003
*	NULL	NULL

Tabla compra

```
155 • insert into compra values
156     ("2001", "001", "5", "250000"),
157     ("2002", "002", "2", "10000"),
158     ("2003", "003", "5", "30000");
159 • select * from compra;
```

Result Grid | Filter Rows: | Edit:

	id_producto_compra	id_cliente_compra	cantidad_producto	total
▶	2001	001	5	250000
	2002	002	2	10000
	2003	003	5	30000
*	NULL	NULL	NULL	NULL

Tabla producto_proveedor

```
168 • insert into producto_proveedor values
169     ("101", "2001"),
170     ("102", "2002"),
171     ("103", "2003");
172 • select * from producto_proveedor;
```

id_proveedor_producto	id_producto_proveedor
101	2001
102	2002
103	2003
NULL	NULL

Consultas y ejemplos

- 1) Esta consulta trae todos los empleados que tengan una ganancia en "liga" mayor a 2000.

```
174 /*-----Consulta 1-----*/
175 • select nombre from empleado
176     where liga >2000;
```

nombre
Empleado3

- 2) Esta consulta trae el nombre y la especialidad del empleado, de los empleados que se especialicen en cabello.

```
177 /*-----Consulta 2-----*/
178 • select empleado.nombre as "empleado", especialidad.especialidad as "especialidad"
179     from empleado inner join especialidad on id_empleado_especialidad=id_empleado
180     where especialidad.especialidad = "cabello";
```

empleado	especialidad
Empleado2	cabello
Empleado3	cabello

- 3) Esta consulta trae a todos los nombres de los clientes con sus respectivos números de teléfonos concatenados en una sola columna.

```

100 where especialidad.especialidad = 'cabello' ;
181 /*-----Consulta 3-----*/
182 • select cliente.nombre, group_concat(contacto_cliente.telefono SEPARATOR ', ') as telefonos
183 from cliente inner join contacto_cliente on id_cliente_contacto=id_cliente
184 group by cliente.id_cliente;

```

nombre	telefonos
Ash Ketchup	315789123
Goku Armando	315789124
Mario Bros	315789125, 315789126

- 4) Esta consulta trae toda la información de las citas que fueron agendadas a las 08:30

```

184 group by cliente.id_cliente;
185 /*-----Consulta 4-----*/
186 • select * from cita
187 where hora="08:30";

```

id_cita	hora	fecha	id_cliente_cita	id_empleado_cita
3001	08:30	01/01/91	001	1001
NULL	NULL	NULL	NULL	NULL

- 5) Esta consulta trae información mas completa sobre la cita, trae su fecha, hora, nombre del cliente que la pidió y el empleado que la realizara.

```

188 /*-----Consulta 5-----*/
189 • select cita.fecha, cita.hora, empleado.nombre as "empleado", cliente.nombre as "cliente"
190 from cliente inner join cita on id_cliente_cita=cliente.id_cliente
191 inner join empleado on id_empleado=cita.id_empleado_cita;

```

fecha	hora	empleado	cliente
01/01/91	08:30	Empleado1	Ash Ketchup
01/01/92	09:30	Empleado2	Goku Armando
01/01/93	10:30	Empleado3	Mario Bros

- 6) Esta consulta trae el nombre de todos los empleados junto con sus respectivas especialidades, concatenadas en una sola columna.

```

192 /*-----Consulta 6-----*/
193 • select empleado.nombre, group_concat(especialidad.especialidad SEPARATOR ', ') as especialidades
194 from empleado inner join especialidad on id_empleado_especialidad=id_empleado
195 group by empleado.id_empleado;

```

nombre	especialidades
Empleado1	cejas
Empleado2	cabello
Empleado3	cabello, piel

- 7) Esta consulta trae toda la información requerida para generar una factura, el cliente, el producto, la cantidad del producto y el valor total que se pagó.

```
197 • select cliente.CC as "cedula", cliente.nombre as "cliente", producto.nombre as "producto", producto.precio as "precio", compra.cantidad_producto as "cantidad", compra.total as
198 from cliente inner join compra on id_cliente_compra=cliente.id_cliente
199 inner join producto on id_producto=compra.id_producto_compra;
```

cedula	cliente	producto	precio	cantidad	total
1115456123	Ash Ketchup	Shampoo anti caspa	50000	5	250000
1115456124	Goku Armando	Tinte	5000	2	10000
1115456125	Mario Bros	Shampoo anti caspa	50000	1	50000
1115456125	Mario Bros	Prestobarba	6000	5	30000

- 8) Esta consulta trae la información (producto vendido por el empleado, cliente que lo compro y el empleado que hizo la venta) de los productos que han vendido los empleados.

```
200 /*-----Consulta 8-----*/
201 • select producto.nombre as "producto", cliente.nombre as "cliente", empleado.nombre as "empleado"
202 from empleado inner join empleado_vende on id_empleado_vende=empleado.id_empleado
203 inner join producto on id_producto=empleado_vende.id_producto_vende
204 inner join compra on id_producto_compra=producto.id_producto
205 inner join cliente on id_cliente=compra.id_cliente_compra;
```

producto	cliente	empleado
Shampoo anti caspa	Ash Ketchup	Empleado1
Shampoo anti caspa	Mario Bros	Empleado1
Tinte	Goku Armando	Empleado2
Prestobarba	Mario Bros	Empleado3

- 9) Esta consulta trae los ingresos totales de la barbería.

```
206 /*-----Consulta 9-----*/
207 • select sum(total) as "ingresos totales"
208 from compra;
```

ingresos totales
340000

- 10) Esta consulta trae el producto mas vendido de la barbería con su respectiva cantidad total.

```
209 /*-----Consulta 10-----*/
210 • select sum(compra.cantidad_producto) as "mayor producto vendido", producto.nombre as "producto"
211 from compra inner join producto on id_producto=compra.id_producto_compra
212 group by compra.id_producto_compra
213 limit 1;
```

mayor producto vendido	producto
6	Shampoo anti caspa

Vistas

- 1) En este punto quise crear una vista que guardara el registro de facturas, y me di cuenta de que me hacía falta un atributo muy importante (fecha), es por eso por lo que decidí

agregarle la columna fecha a la tabla compra.

```
155 • alter table compra
156 add fecha varchar(20);
```

Result Grid

	id_producto_compra	id_cliente_compra	cantidad_producto	total	fecha
▶	2001	001	5	250000	NULL
	2001	003	1	50000	NULL
	2002	002	2	10000	NULL
	2003	003	5	30000	NULL
*	NULL	NULL	NULL	NULL	NULL

Ya con este atributo puedo crear la vista facturas con los datos correspondientes.

```
219 • create view facturas as
220 select cliente.CC as "cedula", cliente.nombre as "cliente", producto.nombre as "producto", producto.precio as "precio",
221 compra.cantidad_producto as "cantidad", compra.total as "total", compra.fecha as "fecha compra"
222 from cliente inner join compra on id_cliente_compra=cliente.id_cliente
223 inner join producto on id_producto=compra.id_producto_compra;
224 • select * from facturas;
```

Result Grid

	cedula	cliente	producto	precio	cantidad	total	fecha compra
▶	1115456123	Ash Ketchup	Shampoo anti caspa	50000	5	250000	NULL
	1115456125	Mario Bros	Shampoo anti caspa	50000	1	50000	NULL
	1115456124	Goku Armando	Tinte	5000	2	10000	NULL
	1115456125	Mario Bros	Prestobarba	6000	5	30000	NULL

En este caso se deicio llevar el registro de factuas mediante vistas.

2) Se crea la vista historial_de_servicios.

```
227 /*-----Vista2-----*/
228 • create view historial_de_servicios as
229 select cliente.nombre as "cliente", empleado.nombre as "empleado", group_concat(producto.nombre SEPARATOR ', ') as productos_consumidos,
230 servicio.descripcion as "servicio", servicio.duracion as "duracion", servicio.fecha as "fecha"
231 from cliente inner join cita on id_cliente_cita=cliente.id_cliente
232 inner join empleado on id_empleado=cita.id_empleado_cita
233 inner join servicio on id_cita_servicio=cita.id_cita
234 inner join producto_servicio on id_servicio_producto=servicio.id_servicio
235 inner join producto on id_producto=producto_servicio.id_producto_servicio
236 group by servicio.id_servicio;
237 • select * from historial_de_servicios;
```

Result Grid

	cliente	empleado	productos_consumidos	servicio	duracion	fecha
▶	Ash Ketchup	Empleado1	Shampoo anti caspa, Acondicionador	descripcion1	NULL	02/02/92
	Goku Armando	Empleado2	Tinte	descripcion2	NULL	02/02/93
	Mario Bros	Empleado3	Prestobarba	descripcion3	NULL	02/02/94

Se crea con el fin de cumplir con unos de los requerimientos del reto, como se puede observas guarda toda la información necesaria de los servicios prestados por la barbería, el nombre del cliente que recibió el servicio, el empleado que realizo el servicio, la fecha y duración del servicio y los productos que se necesitaron para hacer el servicio.

```

113 );
114 • alter table servicio
115     add duracion varchar(10);

```

	id_servicio	fecha	descripcion	precio	id_cita_servicio	duracion
▶	4001	02/02/92	descripcion1	30000	3001	NULL
	4002	02/02/93	descripcion2	20000	3002	NULL
	4003	02/02/94	descripcion3	10000	3003	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL

Aunque antes de generar la viste tuve que agregar la columna duracion a la tabla servicio, ya que era uno de los requerimientos.

- 3) Se crea la vista registro_de_ventas, la cual contiene información de los empleados que han vendido productos y los cuales deberían recibir la liguita extra.

Esta vista es uno de los requerimientos del reto.

```

238 /*-----Vista3-----*/
239 • create view registro_de_ventas as
240 select producto.nombre as "producto", cliente.nombre as "cliente", empleado.nombre as "empleado"
241 from empleado inner join empleado_vende on id_empleado_vende=empleado.id_empleado
242             inner join producto on id_producto=empleado_vende.id_producto_vende
243             inner join compra on id_producto_compra=producto.id_producto
244             inner join cliente on id_cliente=compra.id_cliente_compra;
245 • select * from registro_de_ventas;

```

	producto	cliente	empleado
▶	Shampoo anti caspa	Ash Ketchup	Empleado1
	Shampoo anti caspa	Mario Bros	Empleado1
	Tinte	Goku Armando	Empleado2
	Prestobarba	Mario Bros	Empleado3

- 4) Se crea la vista información_proveedores con la intención de ver el proveedor y los diferentes productos que vende.

```

247 • create view informacion_proveedores as
248 select proveedor.nombre as "proveedor", group_concat(producto.nombre SEPARATOR ', ') as productos
249 from producto inner join producto_proveedor on id_producto_proveedor=producto.id_producto
250             inner join proveedor on id_proveedor=producto_proveedor.id_proveedor_producto
251 group by proveedor.id_proveedor;
252 • select * from informacion_proveedores;

```

	proveedor	productos
▶	Proveedor1	Shampoo anti caspa
	Proveedor2	Tinte
	Proveedor3	Prestobarba, Acondicionador

- 5) Se crea la vista mejores_clientes, la cual lleva el registro de cuanto dinero han gastado sus clientes comprando productos de la barbería y los muestra en orden descendiente para

saber quienes son los mejores clientes.

```
254 • create view mejores_clientes as
255 select sum(compra.total) as "total", cliente.nombre as "cliente"
256 from compra inner join producto on id_producto=compra.id_producto_compra
257 inner join cliente on id_cliente=compra.id_cliente_compra
258 group by compra.id_cliente_compra
259 order by total desc;
260 • select * from mejores_clientes;
```

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

	total	cliente
▶	250000	Ash Ketchup
	80000	Mario Bros
	10000	Goku Armando

Procedimientos

- 1) Procedimiento creado con el fin de que se puedan generar citas de maneras mas fácil.

```
263 delimiter //
264 • create procedure generar_cita(in id varchar(20),in hora varchar(20),in fecha varchar(20),in cliente varchar(20),in empleado varchar(20))
265 begin
266 insert into cita values
267 (id,hora,fecha,cliente,empleado);
268 end
269 //
270 delimiter ;
271 • call generar_cita("3004","11:30","01/01/94","002","1001");
272 • select * from cita;
```

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

id_cita	hora	fecha	id_cliente_cita	id_empleado_cita
▶ 3001	08:30	01/01/91	001	1001
3002	09:30	01/01/92	002	1002
3003	10:30	01/01/93	003	1003
3004	11:30	01/01/94	002	1001
*	NULL	NULL	NULL	NULL

- 2) Procedimiento creado con el fin de modificar la fecha y hora de una cita según el identificador de la cita que reciba.

```
274 delimiter //
275 • create procedure modificar_cita(in id varchar(20),in fecha_nueva varchar(20),in hora_nueva varchar(20))
276 begin
277 update cita
278 set fecha=fecha_nueva, hora=hora_nueva
279 where id_cita=id;
280 end
281 //
282 delimiter ;
283 • call modificar_cita("3004","01/02/94","13:30");
284 • select * from cita;
```

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

id_cita	hora	fecha	id_cliente_cita	id_empleado_cita
▶ 3001	08:30	01/01/91	001	1001
3002	09:30	01/01/92	002	1002
3003	10:30	01/01/93	003	1003
3004	13:30	01/02/94	002	1001
*	NULL	NULL	NULL	NULL

- 3) Procedimiento creado con el fin de cancelar una cita según el identificador de la cita que reciba.

```
286 delimiter //
287 • create procedure cancelar_cita(in id varchar(20)
288 begin
289 delete from cita
290 where id_cita=id;
291 end
292 //
293 delimiter ;
294 • call cancelar_cita("3004");
295 • select * from cita;
```

Result Grid

	id_cita	hora	fecha	id_cliente_cita	id_empleado_cita
▶	3001	08:30	01/01/91	001	1001
	3002	09:30	01/01/92	002	1002
	3003	10:30	01/01/93	003	1003
*	NULL	NULL	NULL	NULL	NULL

- 4) Procedimiento creado con el fin de buscar en la vista creada anteriormente (información_proveedor) con el fin de traer la información de un solo proveedor según el nombre que se le pase al procedimiento.

```
296 /*-----Procedimiento4-----
297 delimiter //
298 • create procedure buscar_proveedor(in nombre varchar(40))
299 begin
300 select * from informacion_proveedores
301 where proveedor=nombre;
302 end
303 //
304 delimiter ;
305 • call buscar_proveedor("Proveedor3");
```

Result Grid

	idProveedor	proveedor	productos
▶	103	Proveedor3	Prestobarba, Acondicionador

Triggers

- 1) Creo una tabla nueva donde se guardará la información de los usuarios que cancelen las citas junto con la fecha en que cancelo.

```

309 • create table cita_cancelada(
310     usuario varchar(30),
311     accion varchar(30),
312     fecha datetime default current_timestamp
313 );

```

Ahora procedo a crear el trigger, el cual me generara un registro en la tabla creada anteriormente.

```

309 • create table cita_cancelada(
310     usuario varchar(30),
311     fecha datetime default current_timestamp
312 );
313 delimiter //
314 • create trigger del_cita after delete on cita
315     for each row
316     begin
317         insert into cita_cancelada values(user(),now());
318     end;
319 //
320 delimiter ;
321 • call cancelar_cita("3004");
322 • select * from cita_cancelada;

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
usuario	fecha			
root@localhost	2023-02-17 02:05:19			

- 2) Creo la tabla control_cambios_cita, para guardar los registros de usuario que modifiquen o creen nuevas citas.

```

324 • create table control_cambios_cita(
325     usuario varchar(30),
326     descripcion varchar(30),
327     fecha datetime default current_timestamp
328 );
329 delimiter //

```

Utilizo el procedimiento crear_cita, para generar una nueva cita y se puede observar como

genera el registro.

```
328      },
329      delimiter //
330 • create trigger ins_cita after insert on cita
331     for each row
332     begin
333         insert into control_cambios_cita values(user(),"creo cita",now());
334     end;
335 //
336 delimiter ;
337
338
339 • call generar_cita("3006","11:30","01/01/94","002","1002");
340 • select * from control_cambios_cita;
```

<

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

	usuario	descripcion	fecha
▶	root@localhost	creo cita	2023-02-17 02:17:15

- 3) Utilizo el procedimiento modificar_cita para modificar una cita y se puede observar que se genera un nuevo registro en la tabla control_cambios_cita.

```
339 //
340 delimiter //
341 • create trigger mod_cita after update on cita
342     for each row
343     begin
344         insert into control_cambios_cita values(user(),"modifico",now());
345     end;
346 //
347 delimiter ;
348 • call modificar_cita("3006","01/02/94","13:30");
349 • select * from control_cambios_cita;
```

<

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

	usuario	descripcion	fecha
▶	root@localhost	creo cita	2023-02-17 02:17:15
	root@localhost	modifico	2023-02-17 02:21:29

- 4) Se crea trigger que genere un registro de la persona que ingreso un producto nuevo al inventario de la barberia.

```
350  /*-----Trigger4-----*/
351  • select * from producto;
352  delimiter //
353  • create trigger ins_producto after insert on producto
354      for each row
355      begin
356          insert into control_cambios_cita values(user(),"creo producto",now());
357      end;
358  //
359  delimiter ;
360  • insert into producto values
361  ("2005","Gel","Descripcion5","100","1000");
362  • select * from control_cambios_cita;
```

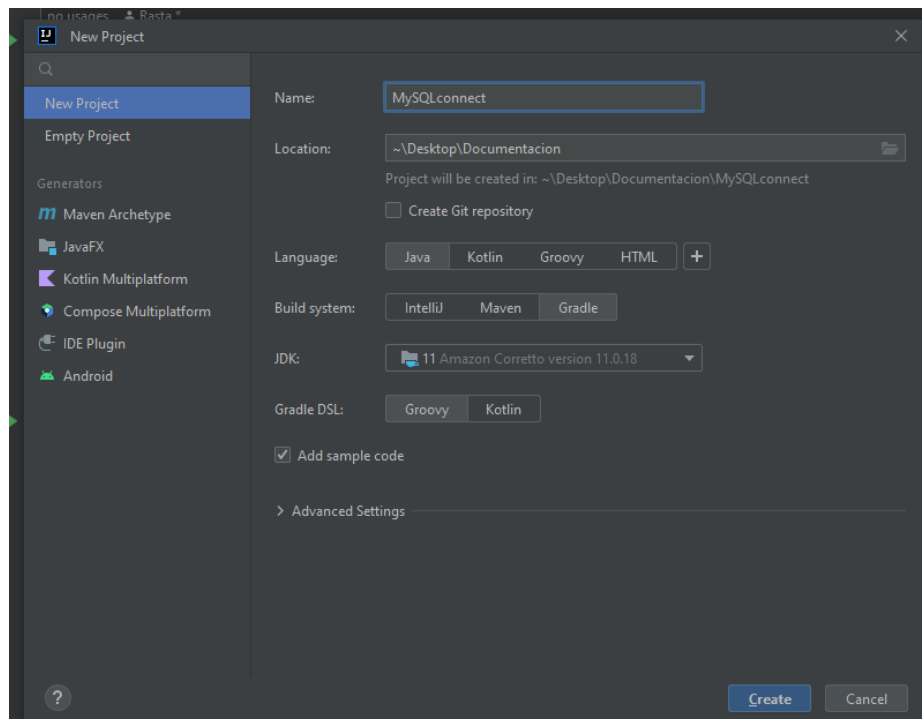
Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	usuario	descripcion	fecha
▶	root@localhost	creo cita	2023-02-17 02:17:15
	root@localhost	modifico	2023-02-17 02:21:29
	root@localhost	creo producto	2023-02-17 02:29:00

Generación de registros en la base de datos

Conexión de java con la base de datos

- 1) Empiezo con la creación de un nuevo proyecto.



- 2) Agrego la dependencia de MySQL.

```
11
12 dependencies {
13     testImplementation 'org.junit.jupiter:junit-jupiter-api:5.8.1'
14     testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.8.1'
15     // https://mvnrepository.com/artifact/com.mysql/mysql-connector-j
16     implementation 'com.mysql:mysql-connector-j:8.0.32'
17
18 }
19
20 test {
21     useJUnitPlatform()
22 }
```

- 3) Creo una clase para guardar las constantes que serán utilizadas a la hora de conectar con la base de datos de MySQL, la primera variable guarda el driver de conexión y la segunda variable guarda la dirección, que después será completada mediante el método `String.format()`, y se le agregarán los datos de la conexión.

```
Main.java x build.gradle (MySQLconnect) x Prueba.java x DataBase.java x MySQLOperation.java x MySQLConstants.java x
package com.sofkau.integration.mysql;

2 usages 2 usages
public class MySQLConstants {
    2 usages
    public static final String MY_SQL_JDBC_DRIVER="com.mysql.cj.jdbc.Driver";
    2 usages
    public static final String CONNECTION_STRING="jdbc:mysql://%s/%s?user=%s&password=%s";
}

}
```

- 4) Creo una interface con los métodos necesarios para generar la conexión.

```
DataBase x executeSqlStatementVoid
Main.java x build.gradle (MySQLconnect) x Prueba.java x DataBase.java x MySQLOperation.java x MySQLConstants.java x
package com.sofkau.integration.mysql;

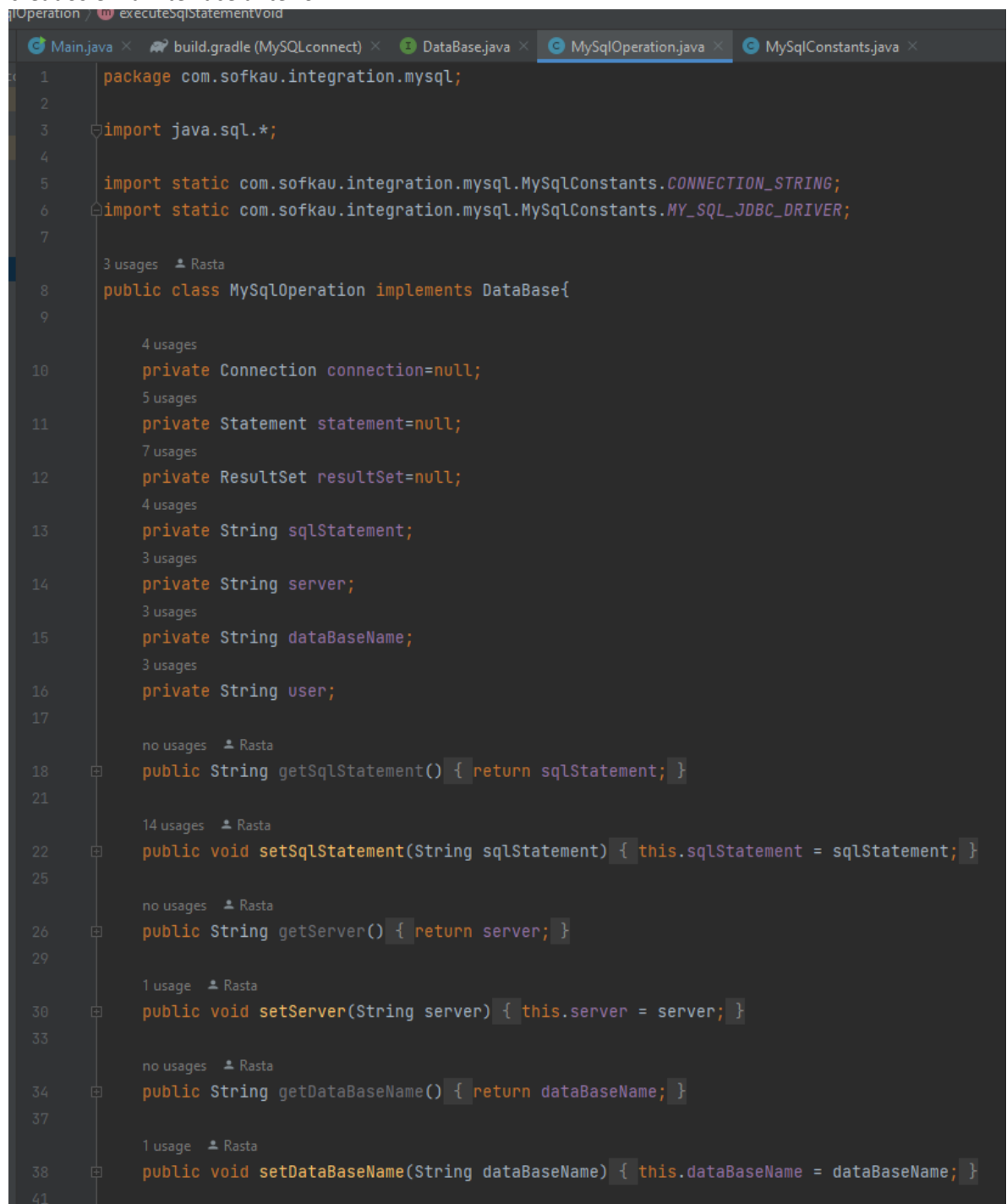
import java.sql.ResultSet;
import java.sql.SQLException;

1 usage 1 implementation 2 usages 1 implementation 1 usage 1 implementation 1 usage 1 implementation 1 usage 1 implementation 13 usages 1 implementation
public interface DataBase {

    2 usages 1 implementation 1 usage 1 implementation no usages 1 implementation 1 usage 1 implementation 1 usage 1 implementation
    public void configureDataBaseConnection();
    public void executeSqlStatement();
    public ResultSet getResultSet();
    public void close();
    public void printResultSet() throws SQLException;
    public void executeSqlStatementVoid();
}

}
```

- 5) Creo una clase llamada MySqlOperation que se encargara de implementar los métodos creados en la interface anterior.



The screenshot shows an IDE with several open files: Main.java, build.gradle (MySQLconnect), DataBase.java, MySqlOperation.java (selected), and MySqlConstants.java. The code in MySqlOperation.java implements the DataBase interface. It includes package declarations, imports for java.sql.*, and static imports for MySQL constants. The class MySqlOperation implements DataBase and contains private fields for connection, statement, resultSet, sqlStatement, server, dbName, and user. It also implements methods getSqlStatement(), setSqlStatement(), getServer(), setServer(), getDataBaseName(), and setDataBaseName().

```
1 package com.sofkau.integration.mysql;
2
3 import java.sql.*;
4
5 import static com.sofkau.integration.mysql.MySqlConstants.CONNECTION_STRING;
6 import static com.sofkau.integration.mysql.MySqlConstants.MY_SQL_JDBC_DRIVER;
7
8 public class MySqlOperation implements DataBase{
9
10     private Connection connection=null;
11     private Statement statement=null;
12     private ResultSet resultSet=null;
13     private String sqlStatement;
14     private String server;
15     private String dataBaseName;
16     private String user;
17
18     public String getSqlStatement() { return sqlStatement; }
19
20     public void setSqlStatement(String sqlStatement) { this.sqlStatement = sqlStatement; }
21
22     public String getServer() { return server; }
23
24     public void setServer(String server) { this.server = server; }
25
26     public String getDataBaseName() { return dataBaseName; }
27
28     public void setDataBaseName(String dataBaseName) { this.dataBaseName = dataBaseName; }
```

6) Implemento todos los métodos necesarios para la conexión.

```
@Override
public void configureDataBaseConnection() {
    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){
        System.out.println(e.getMessage());
    }
}

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

no usages  ▲ Rasta
@Override
public ResultSet getResultSet() {
    return resultSet;
}

@Override
public void close() {
    try{
        if(resultSet != null){
            resultSet.close();
        }
        if(statement != null){
            statement.close();
        }
        if(connection != null){
            connection.close();
        }
    }
    catch (Exception e){
        System.out.println(e.getMessage());
    }
}

1 usage  ▲ Rasta
@Override
public void printResultSet() throws SQLException {
    ResultSetMetaData resultSetMetaData=resultSet.getMetaData();
    int totalColumnNumber=resultSetMetaData.getColumnCount();
    while(resultSet.next()){
        for(int columnIndex=1;columnIndex<=totalColumnNumber;columnIndex++){
            String columnValue=resultSet.getString(columnIndex);
            System.out.print(resultSetMetaData.getColumnName(columnIndex)+" "+columnValue+" ");
        }
        System.out.println();
    }
}

13 usages  ▲ Rasta
@Override
public void executeSqlStatementVoid() {
    try{
        configureDataBaseConnection();
        statement.execute(sqlStatement);
    }catch (Exception e){
        System.out.println(e.getMessage());
    }
}
```

El metodo configureDataBasesConnection se encarga de configurar la conexión utilizando las dos variables estaticas que se crearon anteriormente.

El metodo executeSqlStatement se encarga ejecutar el comando SQL en java.

El metodo close se encarga de cerrar la conexión.

El metodo printResultSet se encarga de mostrar la informacion de una tabla en java.

El metodo executeSqlStatementVoid es el encargado de generar registros en la base de datos.

Poblando la base de datos

```
//long 4 byte
public static void insertRegistros() {
    //DATOS CLIENTE
    int id_cliente=005;
    String cc_cliente="1115789456";
    String nombre_cliente="Pablo Emilio";
    String profesion="Jugador";
    String correo="p.com";
    String edad="37";
    String direccion="calle falsa 123";

    //DATOS CONTACTO_CLIENTE
    String telefono_cliente="315687954";

    //DATOS PROVEEDOR
    int id_proveedor=104;
    String nombre_proveedor="Proveedor";
    String proveedor_direccion="Calle falsa 123456";

    //DATOS CONTACTO_PROVEEDOR
    String telefono_proveedor="315645987";

    //DATOS EMPLEADO
    int id_empleado=1005;
    String cc_empleado="1116987132";
    String nombre_empleado="Nombre";
    double liga=2000;

    //DATOS ESPECIALIDAD
    String especialidad="cabello";

    //DATOS PRODUCTO
    int id_producto = 2000;
    String nombre_producto="Gel ega";
    String descripcion="descripcion";
    String cantidad="100";
    String precio="2000";

    for(int i=0;i<50;i++){
        //Tabla cliente
        MySQLOperation.setSqlStatement("insert into cliente values("+String.valueOf(id_cliente)+","+cc_cliente+","+nombre_cliente+","+profesion+","+
            +correo+","+edad+","+direccion+")");
        MySQLOperation.executeUpdateStatementVoid();

        //Tabla contacto_cliente
        MySQLOperation.setSqlStatement("insert into contacto_cliente values("+String.valueOf(id_cliente)+","+telefono_cliente+")");
        MySQLOperation.executeUpdateStatementVoid();

        //Tabla proveedor
        MySQLOperation.setSqlStatement("insert into proveedor values("+String.valueOf(id_proveedor)+","+nombre_proveedor+","+proveedor_direccion+")");
        MySQLOperation.executeUpdateStatementVoid();

        //Tabla contacto_proveedor
        MySQLOperation.setSqlStatement("insert into contacto_proveedor values("+String.valueOf(id_proveedor)+","+telefono_proveedor+")");
        MySQLOperation.executeUpdateStatementVoid();

        //TABLA EMPLEADO
        MySQLOperation.setSqlStatement("insert into empleado values("+String.valueOf(id_empleado)+","+cc_empleado+","+nombre_empleado+","+liga+")");
        MySQLOperation.executeUpdateStatementVoid();

        //TABLA ESPECIALIDAD
        MySQLOperation.setSqlStatement("insert into especialidad values("+String.valueOf(id_empleado)+","+especialidad+")");
        MySQLOperation.executeUpdateStatementVoid();

        //TABLA PRODUCTO
        MySQLOperation.setSqlStatement("insert into producto values("+String.valueOf(id_producto)+","+nombre_producto+","+descripcion+","+cantidad+","+
            +precio+")");
        MySQLOperation.executeUpdateStatementVoid();

        //TABLA CITA
        MySQLOperation.setSqlStatement("insert into cita values("+String.valueOf(id_cita)+","+hora_cita+","+fecha_cita+","+String.valueOf(id_cliente)+
            +String.valueOf(id_proveedor)+")");
        MySQLOperation.executeUpdateStatementVoid();

        //TABLA SERVICIO
        MySQLOperation.setSqlStatement("insert into servicio values("+String.valueOf(id_servicio)+","+fecha_servicio+","+servicio_descripcion+","+serv
            +String.valueOf(id_cita)+","+servicio_duracion+")");
        MySQLOperation.executeUpdateStatementVoid();

        //TABLA PRODUCTO_SERVICIO
        MySQLOperation.setSqlStatement("insert into producto_servicio values("+String.valueOf(id_producto)+","+String.valueOf(id_servicio)+")");
        MySQLOperation.executeUpdateStatementVoid();

        //TABLA EMPLEADO_VENDE
        MySQLOperation.setSqlStatement("insert into empleado_vende values("+String.valueOf(id_empleado)+","+String.valueOf(id_producto)+")");
        MySQLOperation.executeUpdateStatementVoid();

        //TABLA COMPRA
        MySQLOperation.setSqlStatement("insert into compra values("+String.valueOf(id_producto)+","+String.valueOf(id_cliente)+","+cantidad_producto+",");
    }
}
```

Creo las variables para cada tabla, teniendo en cuenta que las llaves primarias serán de tipo int para que el identificador de cada tabla sea incremental, y al final de cada ciclo aumentarlos en 1 y evitar errores con las conexiones.

```
id_cliente=id_cliente+1;
id_proveedor=id_proveedor+1;
id_empleado=id_empleado+1;
id_producto=id_producto+1;
id_cita=id_cita+1;
id_servicio=id_servicio+1;
```

Como se puede observar en cada ciclo se añade un registro a cada tabla, en el que solo irán cambiando los identificadores, lo hice de esta manera ya que me evita problemas con la llaves foráneas, me hubiera gustado hacerlo mas organizado pero el tiempo me jugaba en contra.

Pregunta final

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

R\\ Si, creo que la manera en la que se enfocó la solución cumple con todos los requisitos que pedían en el enunciado, por ende, creo que el haber trabajado tanto con bases de datos me desarrollo la suficiente lógica y habilidades para haber desarrollado esta actividad sin muchas trabas, aunque debo decir que con más tiempo podría haber poblado las tablas de una manera mucho más organizada.

No tengo mucho conocimiento con NoSQL, pero tengo entendido que se utiliza para el manejo masivo de datos, y como esto es un proyecto pequeño, creo los resultados habrían sido peores.