Planteamiento del problema

Considérese dos bancos separados que deciden fusionarse. Asúmase que ambos bancos usan exactamente el mismo esquema de bases de datos E-R, el de la **figura 1**. Si la fusión del banco tiene una única base de datos, hay varios problemas potenciales:

- La posibilidad de que los dos bancos originales tengan sucursales con el mismo nombre.
- La posibilidad de que algunos clientes sean clientes de ambos bancos originales.
- La posibilidad de que algunos números de préstamo o de cuenta fueran usados en ambos bancos originales (para diferentes préstamos o cuentas, por supuesto).

Luego de analizar la situación planteado, realizar las siguientes actividades:

1. Para cada uno de estos problemas potenciales descríbase por qué existen de hecho dificultades potenciales. Propóngase una solución a este problema. Explíquese cualquier cambio que se tendría que hacer para la solución y descríbase cómo afecta al esquema y a los datos.
2. Plantear el Modelo relacional, una vez se ha llevado a cabo la fusión de los bancos.
3. Crear la base de datos utilizando sentencias SQL.
4. Implementar 2 vistas que se consideren importantes.
5. Implementar 2 procedimientos almacenados.
6. Implementar por lo menos un trigger.
7. Poblar las tablas de la base de datos, utilizando una conexión desde Java.

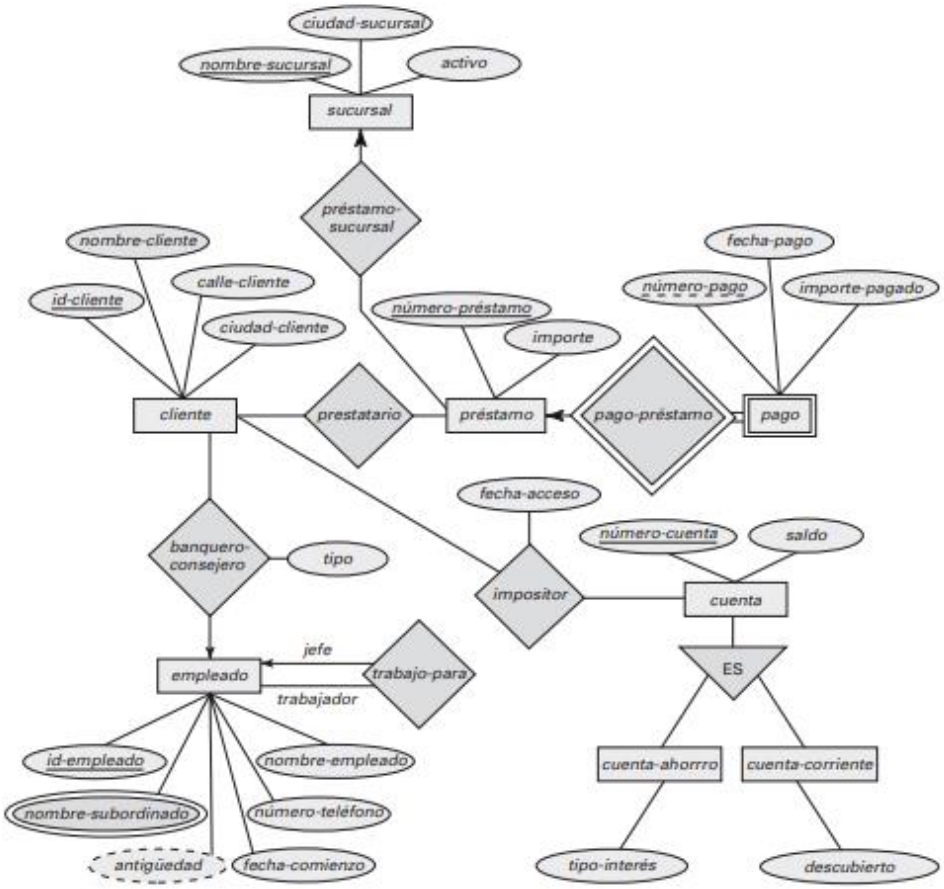


Figura 1: Modelo Entidad Relación compartido por ambos bancos

Solución

1. En la fusión de los bancos descritos en el planteamiento, se pueden presentar los siguientes problemas potenciales:

- En la figura 1 se observa que la entidad Sucursal tiene como clave primaria el atributo nombre-sucursal. En caso de que se tengan sucursales con el mismo nombre en ambos bancos, se estaría presentando una perdida de información de una de las sucursales originales de los bancos a la hora de realizar la fusión. La solución que se propone para este inconveniente es adicionar un atributo nuevo que cumpla la función de identificador único para la entidad Sucursal. De esta manera se podrían tener sucursales con el mismo nombre, pero serían diferenciadas una de la otra mediante su id único.
- Al realizar la fusión de los bancos, se podrían encontrar registros en las bases de datos de ambos bancos originales, generando de esta forma un problema

potencial de información duplicada. Sin embargo, si se eliminan los registros de clientes duplicados antes de realizar la fusión, las llaves foranes de las relaciones que la entidad Cliente tiene con las entidades Consejero, Préstamo y Cuenta, no se verían afectadas. Este proceso se llevaría a cabo a la hora de normalizar la base de datos fusionada.

- En cuanto a las entidades Préstamo y Cuenta que tengan mismo número de identificación en ambos bancos; pueden generar un problema de perdida de información. Esto puede resolverse actualizando los números de cuenta y préstamo en uno de los bancos. Posteriormente, se deberían actualizar las llaves foráneas de las relaciones que presentan estas entidades. De esta manera podría realizarse la fusión sin correr el riesgo de pérdida de información.

De acuerdo a lo anterior, al modelo Entidad Relación de la figura 1 solo sería necesario realizarle una modificación; agregar en la entidad Sucursal un nuevo atributo id, que haga las veces de identificador único en lugar del nombre de la sucursal.

Ahora se procede a definir la cardinalidad de cada una de las relaciones entre entidades del Modelo Entidad Relación:

- **Prestatario (1: N):** Un cliente puede ser prestatario de ninguno o varios Prestamos. Un préstamo puede estar asociado a un único cliente.
- **Banquero-consejero (N: M):** Un cliente puede ser atendido por uno o varios Consejeros. Un Consejero puede asesorar a uno o varios clientes.
- **Impositor (1: N):** Un cliente puede ser impositor de una o varias cuentas. Una cuenta está asociada a un único Cliente. Las sub entidades Cuenta-corriente y Cuenta-ahorro tienen esta misma relación con el Cliente a través de la super entidad Cuenta.
- **Préstamo-sucursal (1: N):** Una sucursal puede tener registrados uno o varios préstamos. Un préstamo puede estar asociado a una sucursal.
- **Pago-préstamo (1: N):** Está es una relación débil, ya que la existencia de la entidad Pago depende de la existencia de la entidad Préstamo. En cuanto a la cardinalidad para este tipo de relaciones se tiene que, un Préstamo puede tener asociados uno o varios pagos, y un Pago está asociado a un Préstamo.
- **Trabajo-para (1: N):** Se trata de una relación reflexiva, es decir una relación de la entidad Empleado con ella misma. En este caso, la cardinalidad se presenta como que un Empleado puede trabajar para un Jefe, y un Jefe tiene a uno o varios Empleados a su cargo.

Al no encontrarse relaciones uno a uno, no se considera importante el análisis de la participación de las entidades. Esta cobra relevancia únicamente en el caso de la entidad débil Pago, cuya existencia depende de la entidad Préstamo.

2. Partiendo del Modelo Entidad Relación de la figura 1, se procede a realizar las correspondientes transformaciones para obtener el Modelo Relacional de la figura 2. El archivo con el esquema se puede encontrar en **diagramas/MR-fusion-bancos.**

   El primer paso para realizar la transformación en convertir cada entidad en una tabla, cuyas columnas son los atributos propios de cada entidad. Posteriormente se analiza el tipo de relación entre las entidades de la siguiente manera:

   - **Relaciones uno a muchos**: Este es el caso de las relaciones prestatario e impositor. En este tipo de relaciones, la tabla asociada a la entidad con la cardinalidad N hereda la llave primaria de la entidad con cardinalidad 1 como llave foránea.
   - **Relaciones débiles**: Este es el caso de la relación pago-préstamo. Para realizar la transformación, se concatena la llave primaria de la tabla Préstamo con el atributo diferenciador número de pago de la tabla Pago. Esta concatenación forma la llave primaria de la tabla Pago.
   - **Relaciones reflexivas/atributos multivaluados**: Este es el caso de la entidad Empleado, que cuenta con el atributo multivaluado nombre_subordinado. Para realizar la transformación se genera una tabla adicional, que lleva el nombre del atributo multivaluado. La llave primaria de esta tabla adicional será la concatenación de la llave primaria de la tabla Empleado, más el valor del atributo nombre_empleado.
   - **Especialización disyuntiva parcial:** Este es el caso de las entidades Cuenta-corriente y Cuenta-ahorro, las cuales heredan de la entidad Cuenta. Para realizar la transformación, las sub entidades heredan como llave foránea la llave primaria de la entidad Cuenta.
   - **Relaciones 1: N con atributos propios:** Este es el caso de la relación Impositor. Adicional a lo mencionado en la transformación para relaciones 1: N; se debe agregar el atributo fecha acceso a la tabla Cuenta.
   - **Relaciones N: M:** Este es el caso de la relación banquero-consejero. En esta situación se genera una tabla intermedia que hereda las llaves primarias de las tablas Empleado y Cliente.
   - **Relaciones N: M con atributos propios:** Este es el caso de la relación banquero-consejero. Adicional a los pasos que se deben

seguir en la transformación de relaciones N: M, se debe agregar en la tabla intermedia generada el atributo propio de la relación (tipo).

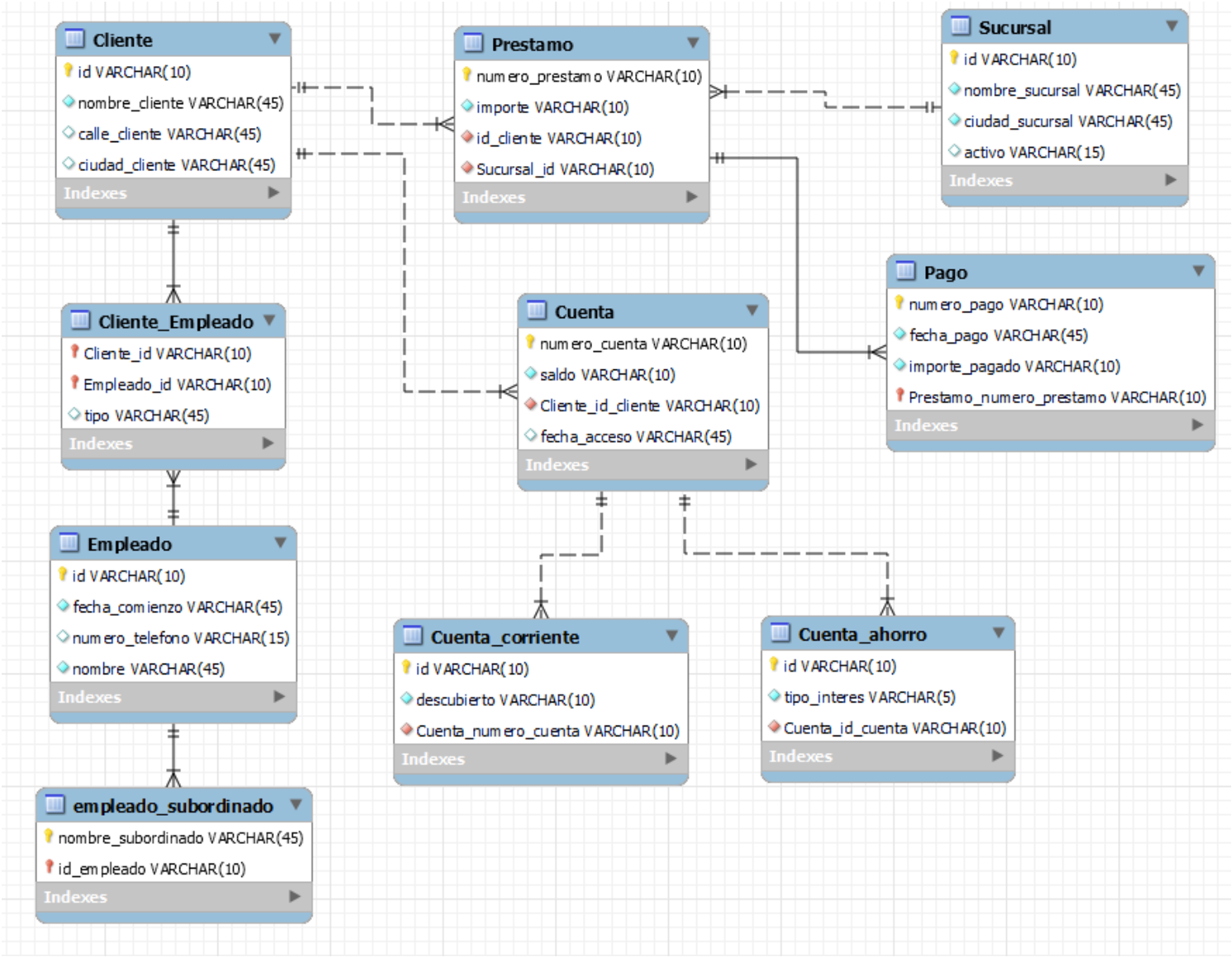


Figura 2: Modelo Relacional fusión bancos

El proceso de normalización para la fusión de los bancos se presenta en las tablas 1, 2 y 3.

Tabla 1. Primera forma Normal

Cada atributo tiene valores atómicos	cumple
No hay atributos multivaluados	cumple
No hay registros duplicados	En este paso se eliminan los registros duplicados de la tabla cliente
No hay columnas duplicadas	Cumple
Cada tabla tiene definida una clave principal	Cumple

Tabla 2: Segunda forma Normal

Se cumple la 1FN	Cumple
Los valores de las tablas dependen solo de la llave primaria	Cumple
Las tablas tienen una única llave primaria que las identifica	cumple

Tabla 3: Tercera forma Normal

Se cumple la 2FN	Cumple
Los atributos no incluidos en la clave primaria no dependen transitivamente de la llave primaria	cumple

3. Ahora se procede a generar la base de datos para la fusión de los bancos utilizando sentencias SQL. El archivo con el script para crear las tablas se encuentra en **scripts/créate-tables.**

```sql
use bancos_fusion;

-- ----------------------------------
-- Se crea la tabla Cliente
-- ----------------------------------

create table cliente (
    id varchar(10) not null,
    nombre_cliente varchar(45) not null,
    calle_cliente varchar(45) null,
    ciudad_cliente varchar(45) null,
    primary key (id));

-- -------------------------------------
-- se crea la tabla Empleado
-- -------------------------------------

create table empleado (
    id varchar(10) not null,
    fecha_comienzo varchar(45) not null,
    numero_telefono varchar(15) null,
    nombre varchar(45) not null,
    primary key (id));
```

Figura 3: Sentencias para crear tablas cliente y empleado

```
25      -- -------------------------------
26      -- se crea la tabla Sucursal
27      -- -------------------------------
28
29  •   create table sucursal (
30          id varchar(10) not null,
31          nombre_sucursal varchar(45) not null,
32          ciudad_sucursal varchar(45) not null,
33          activo varchar(15) null,
34          primary key (id));
35
36      -- ----------------------------
37      -- Se crea la tabla Prestamo
38      -- ----------------------------
39
40  •   create table prestamo (
41          numero_prestamo varchar(10) not null,
42          importe varchar(10) not null,
43          id_cliente varchar(10) not null,
44          sucursal_id varchar(10) not null,
45          primary key(numero_prestamo),
46          foreign key (id_cliente) references cliente (id),
47          foreign key (sucursal_id) references sucursal (id));
```

Figura 4: Sentencias para crear tablas sucursal y prestamo

```sql
create table pago (
    numero_pago varchar(10) not null,
    fecha_pago varchar(45) not null,
    importe_pagado varchar(10) not null,
    Prestamo_numero_prestamo varchar(10) not null,
    primary key (numero_pago, Prestamo_numero_prestamo),
    foreign key (Prestamo_numero_prestamo) references prestamo (numero_prestamo));

-- -----------------------------------------
-- se crea la tabla Cuenta
-- -----------------------------------------

create table Cuenta (
    numero_cuenta varchar(10) not null,
    saldo varchar(10) not null,
    fecha_acceso varchar(45) null,
    Cliente_id_cliente varchar(10) not null,
    primary key (numero_cuenta),
    foreign key (Cliente_id_cliente) references cliente (id));
```

Figura 5: Sentencias para crear tablas pago y cuenta

```
73      -- ---------------------------------------
74      -- se crea la tabla Cuenta_corriente
75      -- ---------------------------------------
76
77  ●   create table cuenta_corriente (
78          id varchar(10) not null,
79          descubierto varchar(10) not null,
80          Cuenta_numero_cuenta varchar(10) not null,
81          primary key (id),
82          foreign key (Cuenta_numero_cuenta) references Cuenta (numero_cuenta));
83
84      -- -------------------------------
85      -- se crea la tabla cuenta_ahorro
86      -- -------------------------------
87
88  ●   create table cuenta_ahorro (
89          id varchar(10) not null,
90          tipo_interes varchar(5) not null,
91          Cuenta_id_cuenta varchar(10),
92          primary key (id),
93          foreign key (Cuenta_id_cuenta) references Cuenta (numero_cuenta));
```

Figura 6: Sentencias para crear tablas cuenta_corriente y cuenta_ahorro

```
95      -- ---------------------------------------
96      -- se crea la tabla empleado_subordinado
97      -- ---------------------------------------
98
99  •   create table empleado_subordinado (
100         nombre_subordinado varchar(45) not null,
101         id_empleado varchar(10) not null,
102         primary key (id_empleado, nombre_subordinado),
103         foreign key (id_empleado) references Empleado (id));
104
105     -- ---------------------------------------------
106     -- se crea la tabla cliente_empleado
107     -- ---------------------------------------------
108
109 •   create table cliente_empleado (
110         Cliente_id varchar(10) not null,
111         empleado_id varchar(10) not null,
112         tipo varchar(45) null,
113         primary key (Cliente_id, empleado_id),
114         foreign key (Cliente_id) references Cliente (id),
115         foreign key (empleado_id) references Empleado (id));
```

Figura 7: Sentencias para crear tablas empleado_subordinado y cliente_empleado

4. Ahora se procede a implementar dos vistas que aporten información valiosa de la base de datos. El script con las sentencias para crear las vistas se encuentra en **scripts/script-views**

    - Vista 1 (productos_cliente): Esta vista se implementa para visualizar los productos que un cliente tiene activos con el banco. Estos productos pueden ser una cuenta, un préstamo o ambos.
    - Vista 2 (pagos_prestamo): Esta vista permite visualizar los pagos realizados por un cliente, asociados a un préstamo.

```sql
use bancos_fusion;


-- ---------------------------------
-- se crea vista productos_cliente
-- ---------------------------------
create view productos_cliente as
select cliente.nombre_cliente, prestamo.importe as importe_prestamo, cuenta.numero_cuenta, cuenta.saldo as saldo_cuenta
from cliente
inner join prestamo on cliente.id = prestamo.id_cliente
inner join cuenta on cliente.id = cuenta.Cliente_id_cliente;
```

Figura 8: sentencia para crear la vista productos_cliente

```sql
-- ------------------------------------
-- se crea la vista pagos_prestamo
-- ------------------------------------
use bancos_fusion;
create view pagos_prestamo as
select cliente.nombre_cliente, prestamo.importe as importe_prestamo, Pago.fecha_pago, Pago.importe_pagado
from cliente
inner join prestamo on cliente.id = prestamo.id_cliente
inner join pago on prestamo.numero_prestamo = pago.Prestamo_numero_prestamo;
```

Figura 9: Sentencia para crear la vista pagos_prestamo

5. Ahora se crean 2 procedimientos almacenados, uno para agregar un cliente nuevo a la tabla Cliente; y otro para actualizar los datos de un cliente existente. El script con las sentencias para crear los procedimientos se encuentra en **scripts/script-procedures**.

```
use bancos_fusion;
DELIMITER //
create procedure agregar_cliente(in id_cliente varchar(10),
    nombre varchar(45),
    calle varchar(45),
    ciudad varchar(45))
    begin
        insert into cliente values (id_cliente, nombre, calle, ciudad);
    end //
    DELIMITER ;
```

Figura 10: Sentencia para crear procedimiento agregar_cliente

```
DELIMITER //
create procedure actualizar_cliente(in id_cliente varchar(10),
    nombre varchar(10),
    calle varchar(45),
    ciudad varchar(45))
begin
    update cliente
    set cliente.nombre_cliente = nombre,
        cliente.calle_cliente = calle,
        cliente.ciudad_cliente = ciudad
    where cliente.id = id_cliente;
end //
DELIMITER ;
```

Figura 11: Sentencia para crear procedimiento actualizar_cliente

6. Ahora se implementará un trigger que se dispare cuando se agregue un nuevo registro en la tabla cliente. El primer paso para esto es crear una tabla llamada control_cambios_bancos con los siguientes campos: usuario (para llevar un registro del usuario que inserta el nuevo registro en la tabla cliente), acción y la fecha en la cual se registró el cliente nuevo (figura 12).

```
1     -- ----------------------------------------
2     -- se crea la tabla control_cambios_banco
3     -- ----------------------------------------
4
5     use bancos_fusion;
6
7     create table control_cambios_banco (
8         usuario varchar(45) not null,
9         accion varchar(45) not null,
10        fecha datetime default current_timestamp
11    );
```

Figura 12: Sentencia para crear tabla control_cambios_banco

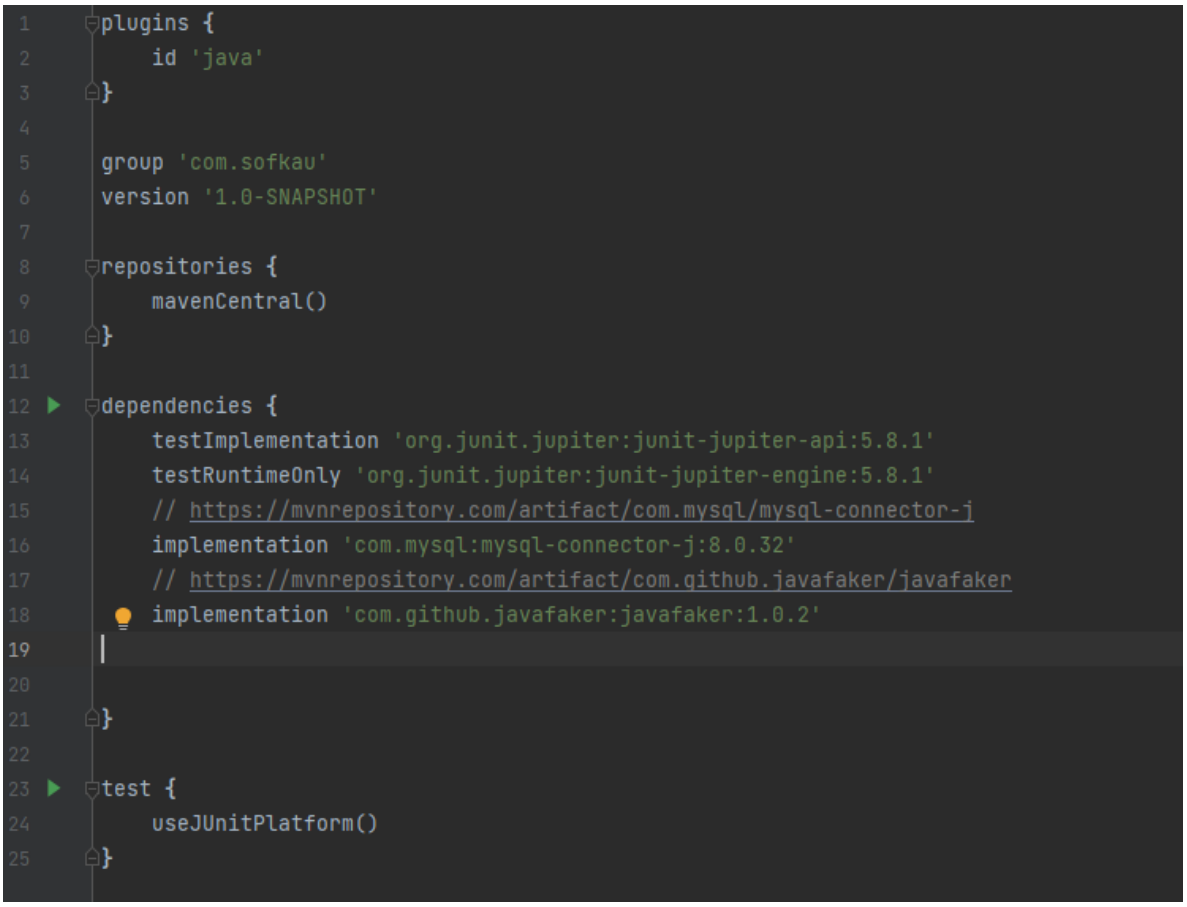
Luego, se crea el trigger on_insert_cliente con la sentencia que se presenta en la figura 13.

```
18    DELIMITER //
19    create trigger on_insert_cliente after insert on bancos_fusion.cliente
20    for each row
21    begin
22        insert into control_cambios_banco
23        values (user(), "Agregó cliente", now());
24    end //
25    DELIMITER ;
```

Figura 13: Sentencia para crear trigger insert_cliente

7. Por último, se procede a poblar cada una de las tablas de la base de datos, utilizando una conexión desde Java.

   Para esto se utiliza como gestor de dependencias Gradle y como editor de código IntelliJ Idea. Las dependencias utilizadas son Java Faker, para generar datos aleatorios para ingresar a las tablas, y el conector de MySql; el cual permite realizar la conexión entre Java y el motor de base de datos. En la figura 14 se presenta una imagen del archivo build.gradle para mostrar las dependencias del proyecto.



```
plugins {
    id 'java'
}

group 'com.sofkau'
version '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

dependencies {
    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'
    // https://mvnrepository.com/artifact/com.github.javafaker/javafaker
    implementation 'com.github.javafaker:javafaker:1.0.2'

}

test {
    useJUnitPlatform()
}
```

Figura 14: Dependencias proyecto fusión_bancos

El proyecto implementado en IntelliJ Idea se puede encontrar en **fusión_bancos/**

La estructura de carpetas y las clases e interfaces contenidas en ellas se presentan en la figura 15.

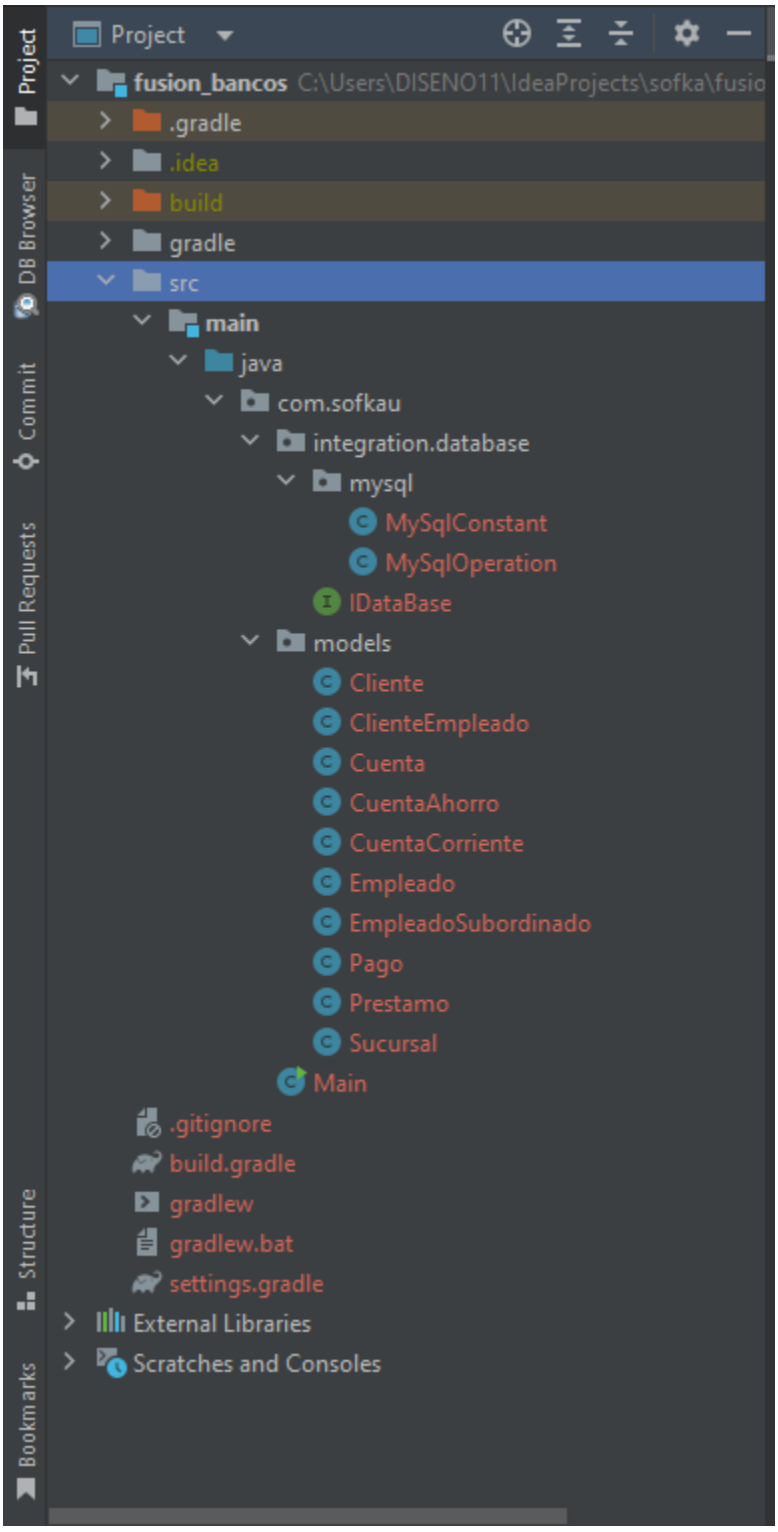


Figura 15: Estructura carpetas proyecto fusion_bancos

- **Paquete intergration**: Allí se puede encontrar el paquete **mysql** y la interfaz IDataBase. En esta interfaz se definen los métodos mostrados en la figura 16. Estos métodos permiten, respectivamente, configurar la conexión con la base de datos, ejecutar una sentencia para insertar datos en una tabla de la base de datos y cerrar una conexión abierta con la base de datos.

```
package com.sofkau.integration.database;

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

2 usages  1 implementation
public interface IDataBase {
    2 usages  1 implementation
    public void configureDataBaseConnection();

    no usages  1 implementation
    public void executeSqlStatement();

    10 usages  1 implementation
    public void executeInsertStatement();

    no usages  1 implementation
    public ResultSet getResultset();

    2 usages  1 implementation
    public void close();

    no usages  1 implementation
    public void printResultset() throws SQLException;
}
```

Figura 16: Interfaz IDataBase

- **Paquete integration.mysql:** Allí se tienen dos clases. En **MySqlConstant** se define una constante para definir el driver a utilizar para la conexión con

MySql, y la otra para definir la url con la cual se realiza la conexión a la base de datos. La otra clase es **MySqlOperation**, en donde se implementan los métodos definidos en la interfaz **IDataBase**

- **Paquete models:** en este paquete se encuentran las clases Cliente, Empleado, Sucursal, Cuenta, CuentaCorriente, CuentaAhorro, Prestamo, Pago, ClienteEmpleado y EmpleadoSubordinado. Cada una de estas clases contiene los mismos atributos que aquellos definidos en cada una de las entidades de la base de datos.
- **Clase main.** En esta clase se asignan los valores a las constantes definidas en la clase MySqlConstant (servidor, nombre base de datos, usuario y contraseña). Además, se implementan todos los métodos que permiten crear instancias de las clases del paquete models, asignar valores a cada uno de sus atributos, y guardar esta información en las tablas de la base de datos.

Por último, en las figuras 17 a 27, se muestra cada una de las tablas de la base de datos, luego de ser pobladas con datos desde Java:

id	nombre_cliente	calle_cliente	ciudad_cliente
100258	Andrés Rios	30A	Envigado
123456	Paco	calle12	Medellín
654321	Fulano	30	Medellin
cliente0	KyraDickens	Garfield Mount	MedellÃ-n
cliente1	BradlyMann	Deandre Mall	MedellÃ-n
cliente2	CarolyneKunde	Emmerich Locks	MedellÃ-n
cliente3	ClementBarrows	Maggio Path	MedellÃ-n
cliente4	RosalynKeebler	Brady Forge	MedellÃ-n
NULL	NULL	NULL	NULL

Figura 17: Tabla Cliente

Cliente_id	empleado_id	tipo
cliente0	EMP0	NULL
cliente1	EMP1	NULL
cliente2	EMP2	NULL
cliente3	EMP3	NULL
cliente4	EMP4	NULL
NULL	NULL	NULL

Figura 18: Tabla Cliente_empleado

	usuario	accion	fecha
▶	root@localhost	Agregó cliente	2023-02-26 09:57:30
	root@localhost	Agregó cliente	2023-02-26 14:31:31
	root@localhost	Agregó cliente	2023-02-26 14:31:31
	root@localhost	Agregó cliente	2023-02-26 14:31:31
	root@localhost	Agregó cliente	2023-02-26 14:31:31
	root@localhost	Agregó cliente	2023-02-26 14:31:31

Figura 19: Tabla control_cambios_banco

	numero_cuenta	saldo	fecha_acceso	Cliente_id_cliente
▶	0771	1524223	Tue Apr 05 07:01:15 COT 2022	cliente1
	1591	2317406	Sun Nov 06 14:33:54 COT 2022	cliente0
	2937	2063847	Fri Feb 04 13:32:00 COT 2022	cliente4
	300	1000000	25/02/2023	123456
	3738	9101825	Sun Feb 20 05:19:37 COT 2022	cliente1
	4160	7756956	Tue Jul 26 06:43:44 COT 2022	cliente2
	6598	4664585	Tue Dec 06 12:20:25 COT 2022	cliente3
	8493	6425890	Tue Oct 04 10:42:05 COT 2022	cliente4
	8494	5237118	Tue Sep 06 22:54:17 COT 2022	cliente3
	9766	2145372	Thu Sep 22 12:21:25 COT 2022	cliente2
	9897	9319977	Sun Jun 26 04:53:43 COT 2022	cliente0
*	NULL	NULL	NULL	NULL

Figura 20: Tabla Cuenta

	id	tipo_interes	Cuenta_id_cuenta
▶	CA#0	1.6	9897
	CA#1	6.8	3738
	CA#2	3.7	4160
	CA#3	8.8	6598
	CA#4	7.4	8493
*	NULL	NULL	NULL

Figura 21: Tabla cuenta_ahorro

	id	descubierto	Cuenta_numero_cuenta
▶	CC#0	3093482	9897
	CC#1	1471967	3738
	CC#2	8948206	4160
	CC#3	0961012	6598
	CC#4	1504534	8493
*	NULL	NULL	NULL

Figura 22: tabla cuenta_corriente

	id	fecha_comienzo	numero_telefono	nombre
▶	EMP0	Tue Nov 08 07:40:10 COT 2022	60496828	RubenKihn
	EMP1	Wed Jan 11 12:16:26 COT 2023	60480734	RossHettinger
	EMP2	Wed Jan 26 04:13:05 COT 2022	60408923	VernonRitchie
	EMP3	Fri Mar 18 04:39:28 COT 2022	60419009	AbelFeil
	EMP4	Thu Jan 20 13:00:06 COT 2022	60449319	AikoHuel
*	NULL	NULL	NULL	NULL

Figura 23: tabla Empleado

	nombre_subordinado	id_empleado
▶	MalikGerlach	EMP0
	ValentinaAbbott	EMP0
	ChongFeest	EMP1
	ClarisaAbbott	EMP1
	JessieWalker	EMP2
	TaunyaHessel	EMP2
	ArethaHalvorson	EMP3
	VonBarrows	EMP3
	LeeJohns	EMP4
	VandaCorkery	EMP4
*	NULL	NULL

Figura 24: Tabla empleado_subordinado

	numero_pago	fecha_pago	importe_pagado	Prestamo_numero_prestamo
▶	1	25/02/2023	100000	100
	Pago#0	Mon May 16 06:55:01 ...	489156	PREST-0
	Pago#1	Mon Dec 19 22:18:08 ...	032043	PREST-1
	Pago#2	Sat Nov 26 07:08:52 ...	897558	PREST-2
	Pago#3	Sat Jan 14 20:04:37 C...	482506	PREST-3
	Pago#4	Sun Jun 26 02:39:19 ...	437057	PREST-4
*	NULL	NULL	NULL	NULL

Figura 25: Tabla pago

	numero_prestamo	importe	id_cliente	sucursal_id
▶	100	10000000	123456	500
	PREST-0	7300392	cliente0	SUC-0
	PREST-1	3786395	cliente1	SUC-1
	PREST-2	1439751	cliente2	SUC-2
	PREST-3	9838631	cliente3	SUC-3
	PREST-4	7688494	cliente4	SUC-4
*	NULL	NULL	NULL	NULL

Figura 26: Tabla préstamo

	id	nombre_sucursal	ciudad_sucursal	activo
▶	500	parque Belén	Medellín	NULL
	SUC-0	Hettinger Bridge	North Pedro	257971
	SUC-1	Durgan Mountains	Wesleyton	117291
	SUC-2	Quigley Bypass	West Darren	275111
	SUC-3	Barrows Path	Ortizview	204838
	SUC-4	Minta Squares	Janellfort	546856
*	NULL	NULL	NULL	NULL

Figura 27: Tabla sucursal