Universidad Nacional de San Agustín de Arequipa
Facultada de Ingeniería de Producción y Servicios
Departamento Académico de Ingeniería de Sistemas e Informática
Escuela Profesional de Ingeniería de Sistemas

DESARROLLO DE SOFTWARE ORIENTADO A OBJETOS

TEMA:

Interfaces Gráficas y Base de Datos

Escuela: Escuela Profesional de Ingeniería de Sistemas

Docente: Saire Peralta, Edwar Abril

Integrantes:

- Miranda Lopinta, Ana Pamela
- Paredes Chalco Emerson Jair
- Quispe Hacha Aleyda Luz
- Mamani Gutierrez Jonahtan Joaquin Mael

ÍNDICE

1. INTRODUCCIÓN

El presente trabajo de investigación formativa aborda la problemática de la volatilidad de la información en las versiones previas del sistema bancario "Los Andes", donde los datos se perdían al finalizar la ejecución del programa. Para solucionar este inconveniente crítico, el proyecto se centra en la transición de un almacenamiento en estructuras de datos temporales hacia un modelo de persistencia robusto.

Es fundamental integrar Java, Interfaz Gráfica (GUI) y Base de Datos porque simula la arquitectura real de las aplicaciones empresariales modernas. Mientras que Java gestiona la lógica de negocio y la GUI facilita la interacción con el usuario, la base de datos garantiza la integridad y disponibilidad de la información a lo largo del tiempo. Esta integración permite al estudiante comprender el flujo completo de desarrollo de software, desde el frontend visual hasta el backend de datos.

El objetivo principal del trabajo es implementar una solución tecnológica capaz de realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre una base de datos MySQL, conectada mediante JDBC a una aplicación de escritorio. Finalmente, en este informe se presentará el marco teórico necesario (SQL, JDBC), la metodología de diseño de la base de datos "banco", la implementación del código fuente y las evidencias funcionales que demuestran la correcta persistencia de las transacciones financieras.

2. PLANTEAMIENTO DEL PROBLEMA

En las etapas iniciales del desarrollo del software para el "Banco Los Andes", el sistema operaba utilizando estructuras dinámicas en memoria volátil (como ArrayList), lo que representaba una limitación severa: toda la información de clientes, cuentas y transacciones se eliminaba automáticamente al cerrar la aplicación o ante fallos de energía. Esta situación hacía inviable la operatividad real del banco, ya que no existía un registro histórico confiable ni seguridad sobre los saldos de los usuarios.

Por consiguiente, surge la necesidad de implementar persistencia de datos mediante un Sistema Gestor de Base de Datos (SGBD). El nuevo sistema debe satisfacer los requerimientos del "Banco Los Andes", garantizando que el registro de clientes (con DNI y dirección), la gestión de empleados, la creación de cuentas y las operaciones monetarias (depósitos y retiros) queden almacenados permanentemente. Esto permitirá realizar consultas históricas y asegurar que la información financiera se mantenga íntegra e inalterable independientemente del ciclo de vida del programa Java

3. OBJETIVOS DEL TRABAJO

3.1. Objetivo General

Implementar un sistema bancario en Java con interfaz gráfica que se conecte a una base de datos MySQL para realizar operaciones CRUD y garantizar la persistencia de datos.

3.2. Objetivo Específicos

- Crear la base de datos del sistema bancario con sus tablas y relaciones.
- Implementar la conexión Java ↔ MySQL mediante JDBC.
- Migrar el almacenamiento en memoria hacia almacenamiento persistente.
- Ejecutar las operaciones CRUD desde la GUI.
- Validar datos y asegurar integridad de las transacciones.

4. MARCO TEÓRICO

4.1. Base de Datos Relacional

El sistema de información del banco se sustenta en un modelo relacional, el cual organiza los datos en estructuras lógicas que permiten su almacenamiento, recuperación y gestión eficiente.

Tabla: Es la estructura fundamental donde se almacena la información. Representa una entidad del sistema (por ejemplo, Cliente o Cuenta) y está compuesta por filas donde se encuentran las instancias y columnas donde están los atributos (Silberschatz et al., 2020).

		id_cliente	nombre	direccion	contrasena
☐	Editar Copiar Borrar	C-001	x	x	123
☐	Editar Copiar Borrar	C-002	Ximena	Jiron Cusco	1234
☐	Editar Copiar Borrar	C-003	Emerson	Calle Unsa	1234
☐	Editar Copiar Borrar	C-004	Jonathan	Calle Sistemas	1234
☐	Editar Copiar Borrar	C-005	Pepe	Calle Bugambilias	1234
☐	Editar Copiar Borrar	C-006	Juancito	Calle Perú	1234

Campo: Corresponde a cada columna de una tabla y representa un atributo específico de la entidad (ej. DNI, Nombre, Saldo). Cada campo tiene definido un tipo de dato como Varchar, Int, Float o Date (Silberschatz et al., 2020).

id_cliente	nombre	direccion	contrasena
C-001	x	x	123
C-002	Ximena	Jiron Cusco	1234
C-003	Emerson	Calle Unsa	1234
C-004	Jonathan	Calle Sistemas	1234
C-005	Pepe	Calle Bugambilias	1234
C-006	Juancito	Calle Perú	1234

Registro: Es cada fila individual dentro de una tabla. Representa una instancia única de los datos, es decir, un objeto específico. Ej. un cliente en particular con todos sus datos (Silberschatz et al., 2020).

id_cliente	nombre	direccion	contrasena
C-001	x	x	123
C-002	Ximena	Jiron Cusco	1234
C-003	Emerson	Calle Unsa	1234
C-004	Jonathan	Calle Sistemas	1234
C-005	Pepe	Calle Bugambilias	1234
C-006	Juancito	Calle Perú	1234

Clave Primaria (Primary Key): Es un campo (o conjunto de campos) que identifica de manera única a cada registro en una tabla, impidiendo la duplicidad. En el sistema bancario, el DNI o un NumeroCuenta actúan como identificadores únicos (Connolly & Begg, 2015).

id_cliente	nombre	direccion	contrasena
C-001	x	x	123
C-002	Ximena	Jiron Cusco	1234
C-003	Emerson	Calle Unsa	1234
C-004	Jonathan	Calle Sistemas	1234
C-005	Pepe	Calle Bugambilias	1234
C-006	Juancito	Calle Perú	1234

Clave Foránea (Foreign Key): Es un campo que establece una relación con la clave primaria de otra tabla, garantizando la integridad referencial entre las entidades. Ej: el campo DNI_Cliente en la tabla Cuenta vincula la cuenta con su propietario (Connolly & Begg, 2015).

id_cliente	nombre	direccion	contrasena
C-001	x	x	123
C-002	Ximena	Jiron Cusco	1234
C-003	Emerson	Calle Unsa	1234
C-004	Jonathan	Calle Sistemas	1234
C-005	Pepe	Calle Bugambilias	1234
C-006	Juancito	Calle Perú	1234

Relaciones: Definen cómo interactúan las tablas entre sí. El sistema utiliza relaciones para conectar clientes con sus cuentas y cuentas con sus transacciones, permitiendo consultas cruzadas (Elmasri & Navathe, 2017).

4.2. Lenguaje SQL

SQL (Structured Query Language) es el lenguaje estándar utilizado para definir, manipular y controlar datos en sistemas de gestión de bases de datos relacionales (RDBMS). Se divide principalmente en Lenguaje de Definición de Datos (DDL) y Lenguaje de Manipulación de Datos (DML) (Beaulieu, 2020).

CREATE, INSERT, SELECT, UPDATE, DELETE: La sentencia CREATE pertenece al DDL y se utiliza para definir la estructura de nuevas tablas y restricciones. Las sentencias DML permiten la gestión de la información: INSERT agrega nuevos registros, UPDATE modifica datos existentes y DELETE elimina filas específicas. La sentencia SELECT es la más utilizada para la recuperación de datos, permitiendo proyecciones y selecciones basadas en criterios lógicos (Silberschatz et al., 2020).

Creación de tablas:

```
CREATE TABLE `administrador` (
  `id_admin` varchar(20) NOT NULL,
  `nombre` varchar(100) NOT NULL,
  `contrasena` varchar(10) NOT NULL,
  PRIMARY KEY (`id_admin`)
)
```

Creación de DB:

```
CREATE DATABASE `banco`;
```

Insert:

```
INSERT INTO `administrador` (`id_admin`, `nombre`, `contrasena`) VALUES
('A-001', 'admin', '1234'),
('A-002', 'admin2', '12345');
```

Select:

```
SELECT nombre, direccion FROM cliente;
```

Update:

```
UPDATE cliente
SET email = `nuevo@email.com`
WHERE id = 1;
```

JOIN básico: La cláusula JOIN permite combinar registros de dos o más tablas en una base de datos relacional, basándose en un campo común entre ellas. Es la implementación técnica del producto cartesiano filtrado, esencial para reconstruir la información distribuida en tablas normalizadas (Beaulieu, 2020).

4.3. JDBC

JDBC es una API (Application Programming Interface) estándar de Java que proporciona conectividad independiente de la base de datos entre el lenguaje de programación Java y una amplia gama de bases de datos SQL, permitiendo enviar sentencias SQL y procesar los resultados (Oracle, 2023).

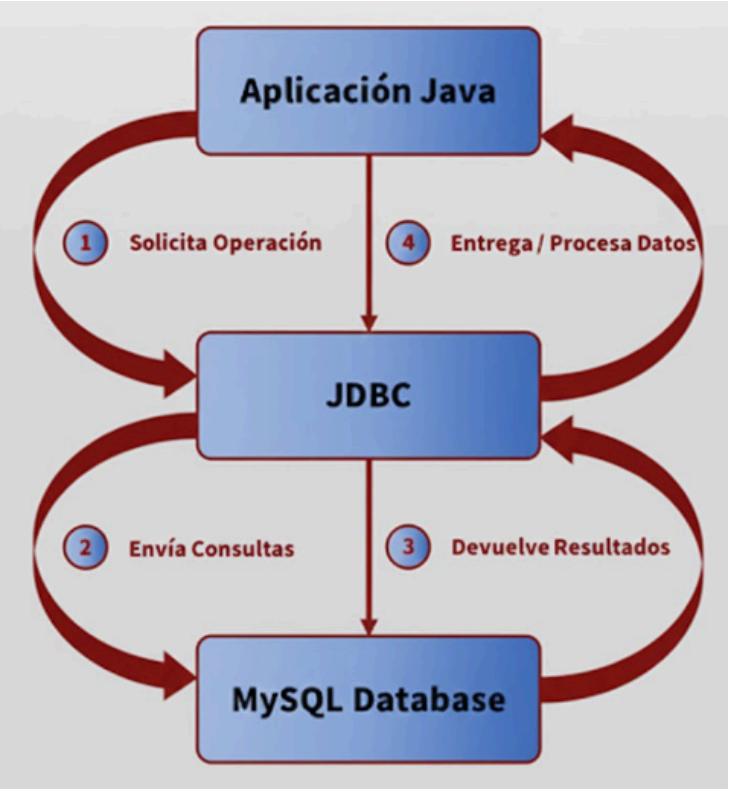
Driver: Es el componente de software específico del proveedor (como mysql-connector-java) que implementa las interfaces definidas por la API JDBC. Actúa como un traductor que convierte las llamadas de métodos Java en el protocolo de red nativo del sistema gestor de base de datos (Horstmann, 2019).

Connection: Representa una sesión física con la base de datos específica. A través de este objeto se gestionan las transacciones y se crean los objetos necesarios para ejecutar sentencias SQL. Sin una instancia activa de Connection, la aplicación no puede interactuar con el servidor de datos (Oracle, 2023).

PreparedStatement: Es una subinterfaz de Statement que representa una sentencia SQL precompilada. Su uso es una buena práctica de seguridad y rendimiento, ya que previene ataques de inyección SQL mediante el uso de parámetros (?) y permite que la base de datos reutilice el plan de ejecución de la consulta (Horstmann, 2019).

ResultSet: Es un objeto que encapsula los resultados devueltos por una consulta SQL (SELECT). Actúa como un cursor o iterador que permite recorrer las filas resultantes una a una para extraer los datos y presentarlos en la interfaz gráfica de la aplicación (Oracle, 2023).

4.4. Integración GUI + Base de Datos



5. METODOLOGÍA

5.1. Diseño de Base de Datos

a) Nombre de la Base de Datos

- Base de datos: banco

b) Tablas utilizadas

Se definieron las siguientes tablas principales:

- cliente
- empleado
- cuenta
- transacción

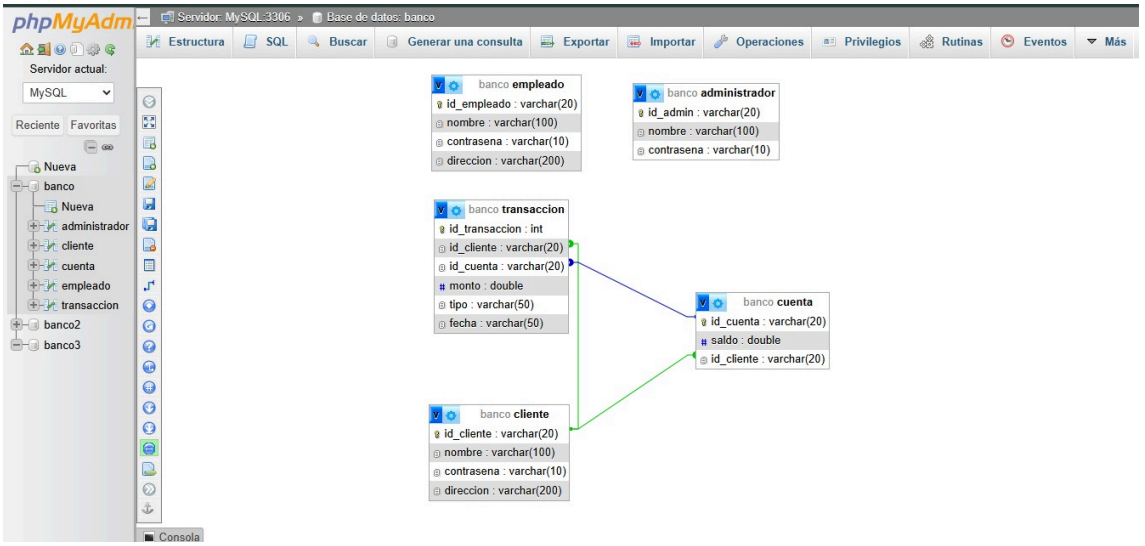
c) Explicación breve del modelo y relación entre tablas

Las tablas se conectan mediante los ids, esto permite relacionar una tabla con otra usando las claves foráneas.

Este diseño permite:

- Mantener el saldo de la cuenta actualizado.
- Guardar un historial de movimientos consultable.
- Evitar registros inválidos gracias a claves foráneas.

d) Diagrama ER



5.2. Implementación del Sistema

a) Descripción de las ventanas usadas (GUI)

La aplicación se implementó con ventanas como:

- Menú de admin
- Menú de clientes
- Menú de empleados
- Menú de registro de clientes y empleados
- Menú de transacciones
- Menú para mostrar información

b) Migración del código del laboratorio previo

Se tomó como base el sistema del laboratorio anterior, que trabajaba con almacenamiento en memoria, y se realizó la migración a persistencia:

- Se reemplazaron estructuras en memoria por consultas a BD.
- Las operaciones que antes modificaban listas ahora ejecutan:
  - INSERT para registros nuevos
  - SELECT para consultas/listados
  - UPDATE para cambios
  - DELETE para eliminaciones (cuando corresponda)

Esto permitió mantener la lógica principal, pero cambiando la fuente de datos a MySQL.

c) Cómo se integró JDBC

La integración con MySQL se realizó con JDBC siguiendo el flujo:

1. Configuración del driver MySQL Connector/J en el proyecto.
2. Implementación de una clase de conexión que:
   - Define URL, usuario y contraseña.
   - Retorna un objeto Connection.
3. Uso de PreparedStatement para ejecutar sentencias SQL con parámetros, evitando concatenaciones y mejorando seguridad y estabilidad.
4. Recuperación de resultados con ResultSet para poblar tablas y campos en la GUI.

d) Validaciones aplicadas

- **Campos vacíos:** no permitir registrar clientes/cuentas/transacciones si faltan datos obligatorios.
- **Tipos numéricos:** validar que montos
- **Monto válido:** no permitir montos menores o iguales a cero.

- **Mensajes de error/éxito:** retroalimentación clara al usuario cuando una operación falla o se completa.

6. RESULTADOS

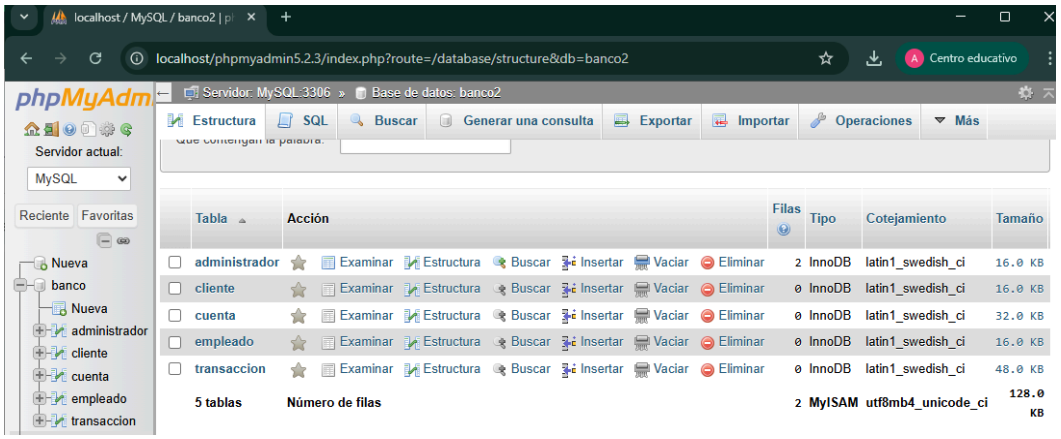
En esta sección se presentan los resultados obtenidos tras la implementación del sistema bancario desarrollado en Java con interfaz gráfica y conexión a una base de datos MySQL. Se evidencia el correcto funcionamiento del sistema mediante capturas de pantalla y pruebas realizadas sobre operaciones reales de la base de datos, demostrando la persistencia de la información y la correcta integración entre la GUI y la base de datos.

6.1. Capturas de pantalla

A continuación, se muestran las principales capturas que evidencian el funcionamiento del sistema bancario:

- **Creación de la base de datos**

  A partir del Script SQL que se denota en el **Anexo A**, se presenta la base de datos denominada banco, creada en el gestor MySQL a través de phpMyAdmin, donde se observan las tablas definidas para el sistema, tales como cliente, empleado, cuenta y transacción.



- **Tablas llenas con datos**

  Se evidencian las tablas con registros reales insertados desde el sistema Java, mostrando que la información ya no se almacena en memoria volátil, sino de forma persistente en la base de datos:

Tabla administrador

Tabla cliente

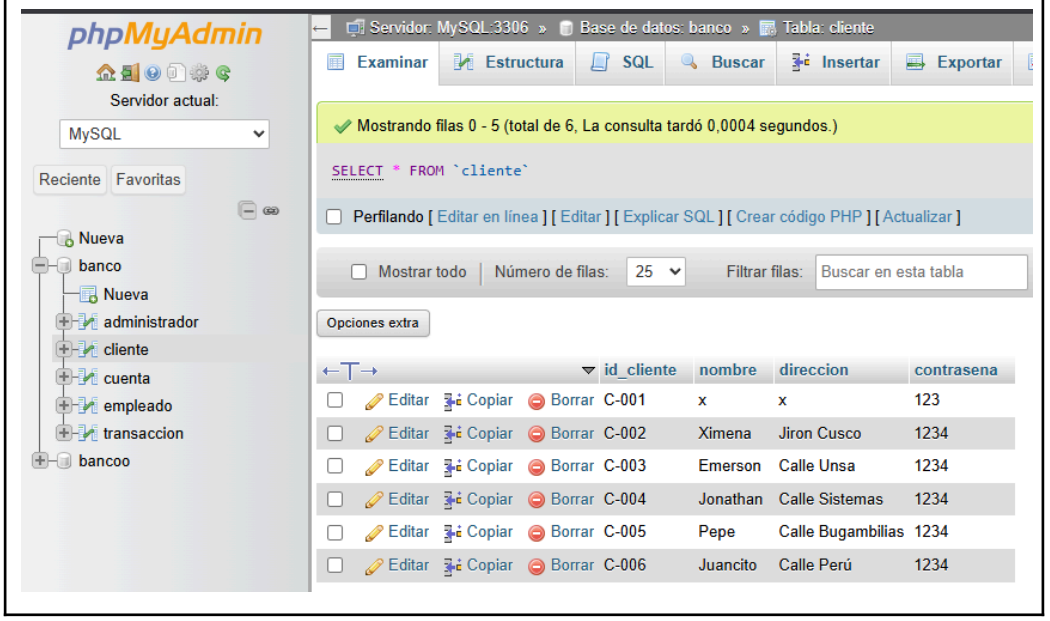
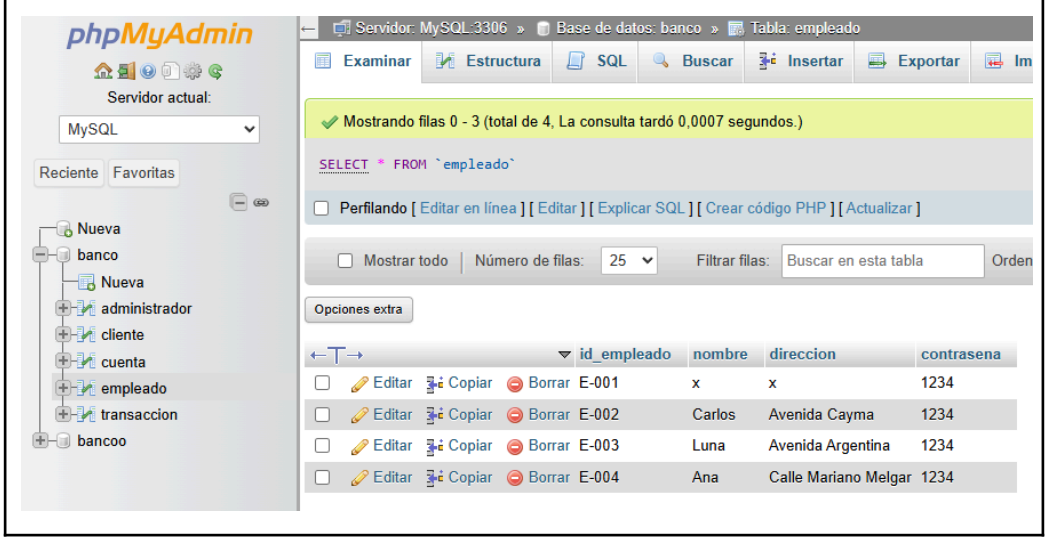


Tabla cuenta

Tabla empleado

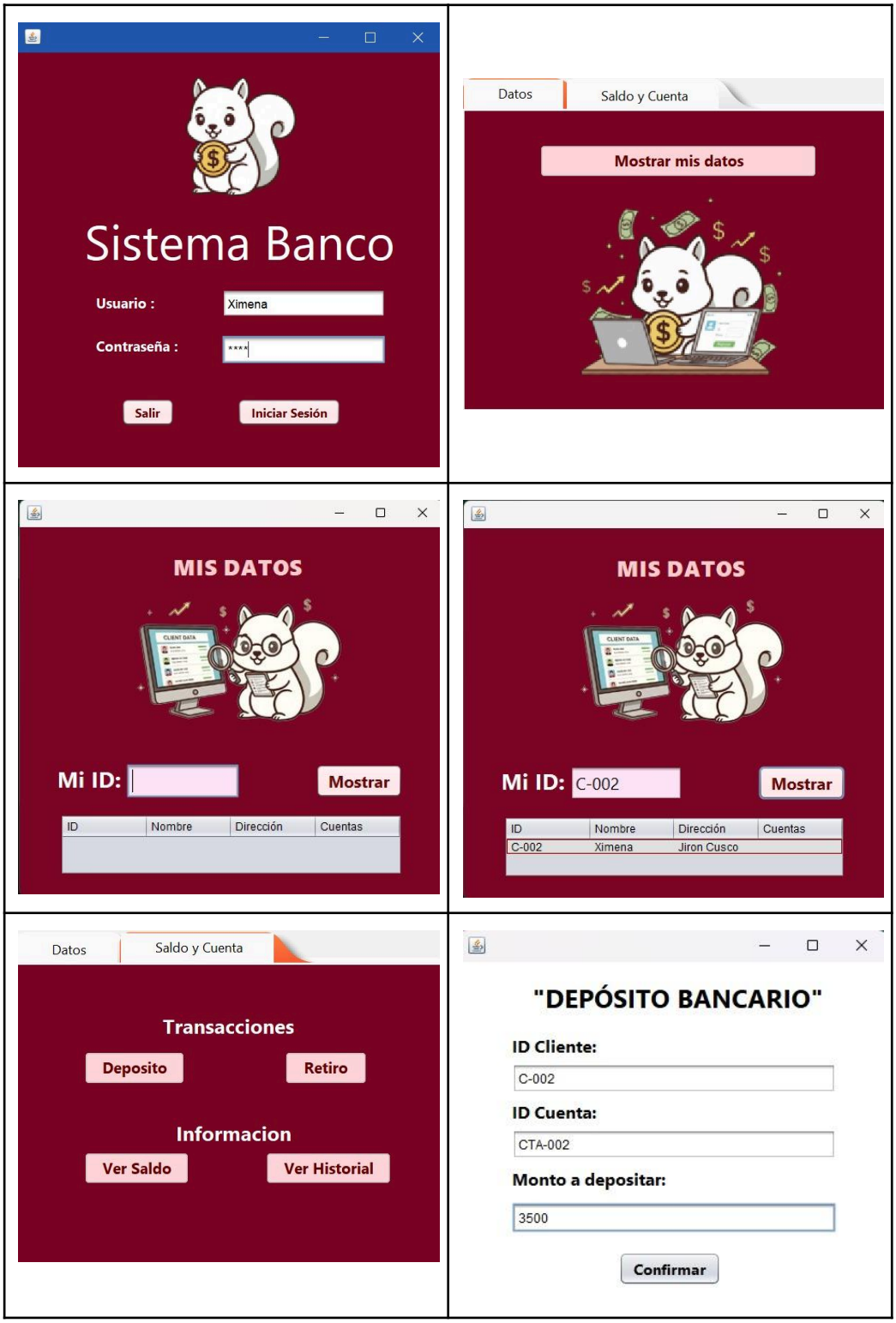


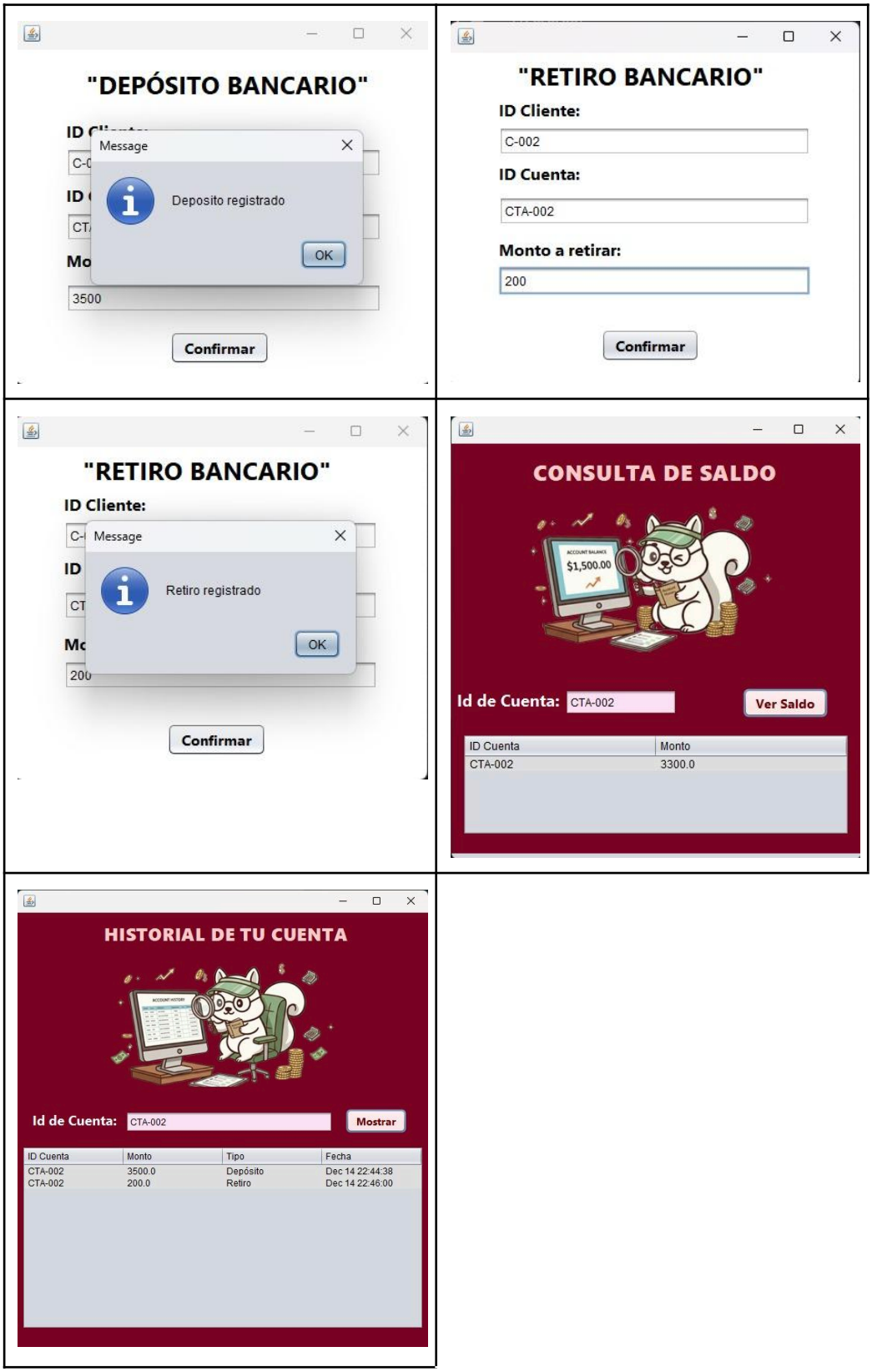
- **Ventanas Java funcionando con datos reales**

  Se incluyen capturas de las interfaces gráficas desarrolladas en Java, como el registro de clientes, visualización de datos del cliente y listado de cuentas, donde se observa que los datos mostrados provienen directamente de la base de datos.

  A continuación se presentan las Ventanas Java funcionando según el Login es decir el tipo de usuario que ingrese al sistema, así como el sql que quedará al final después de los respectivos cambios.

CLIENTE:


Sistema Banco
Usuario :
Ximena
Contraseña :
****
Salir
Iniciar Sesión
Datos
Saldo y Cuenta
Mostrar mis datos
MIS DATOS
Mi ID:
Mostrar
ID
Nombre
Dirección
Cuentas
MIS DATOS
Mi ID:
C-002
Mostrar
ID
Nombre
Dirección
Cuentas
C-002
Ximena
Jiron Cusco
Datos
Saldo y Cuenta
Transacciones
Deposito
Retiro
Informacion
Ver Saldo
Ver Historial
"DEPÓSITO BANCARIO"
ID Cliente:
C-002
ID Cuenta:
CTA-002
Monto a depositar:
3500
Confirmar


"DEPÓSITO BANCARIO"
Message
Deposito registrado
OK
3500
Confirmar
"RETIRO BANCARIO"
ID Cliente:
C-002
ID Cuenta:
CTA-002
Monto a retirar:
200
Confirmar
"RETIRO BANCARIO"
ID Cliente:
Message
Retiro registrado
OK
Confirmar
CONSULTA DE SALDO
$1,500.00
Id de Cuenta:
CTA-002
Ver Saldo
ID Cuenta
Monto
CTA-002
3300.0
HISTORIAL DE TU CUENTA
Id de Cuenta:
CTA-002
Mostrar
ID Cuenta
Monto
Tipo
Fecha
CTA-002
3500.0
Depósito
Dec 14 22:44:38
CTA-002
200.0
Retiro
Dec 14 22:46:00

EMPLEADO:


Sistema Banco
Usuario :
Luna
Contraseña :
****
Salir
Iniciar Sesión

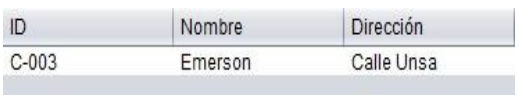

Gestion
Transaccion
Varios
Registar Cliente
Modificar cliente
Eliminar cliente
Registrar cuenta a cliente

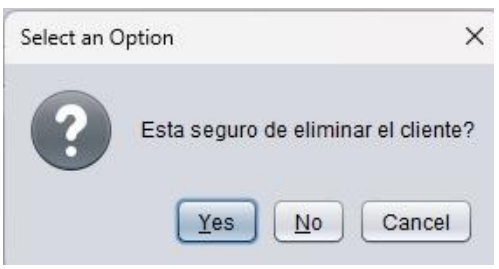

REGISTRAR CLIENTE
Nombre :
Fernanda
ID :
C-007
Clave :
1502
Dirección :
Calle Villanueva
Registrar


REGISTRAR CLIENTE
Message
Cliente registrado
OK
Fernanda
C-007
1502
Calle Villanueva
Registrar


Modificar Cliente por ID
ID de cliente :
C-001
Nuevo Nombre :
Ana
Nueva Dirección :
Av. el Sol
Nueva Clave :
4321
Aceptar


Eliminar Cliente
Id de Cliente:
C-003
ID
Nombre
Dirección
Buscar
Aceptar


ID
Nombre
Dirección
C-003
Emerson
Calle Unsa


Select an Option
Esta seguro de eliminar el cliente?
Yes
No
Cancel


REGISTRAR CUENTA A CLIENTE
ID del cliente:
C-004
ID de la cuenta:
CTA-005
Registrar


Registro Completado
CUENTA REGISTRADA EXITOSAMENTE
ID Cliente: C-004
ID Cuenta: CTA-005
Saldo Inicial: $0.00
La cuenta ha sido asociada al cliente.
OK


Gestion
Transaccion
Varios
Procesar depósito
Procesar retiro

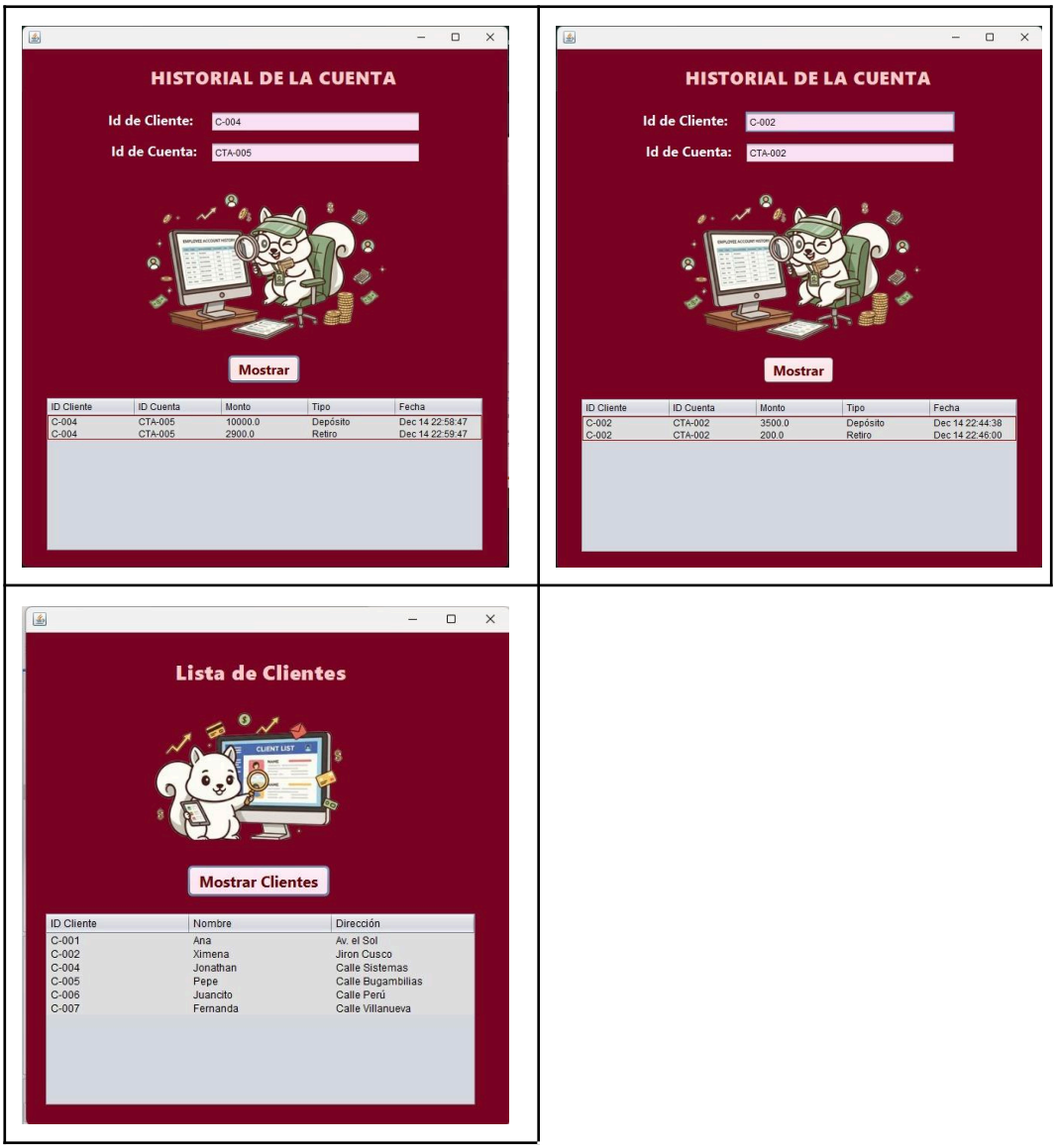

"DEPÓSITO BANCARIO"
ID Cliente:
C-004
ID Cuenta:
CTA-005
Monto a depositar:
10000
Confirmar


"DEPÓSITO BANCARIO"
Message
Deposito registrado
OK
10000
Confirmar

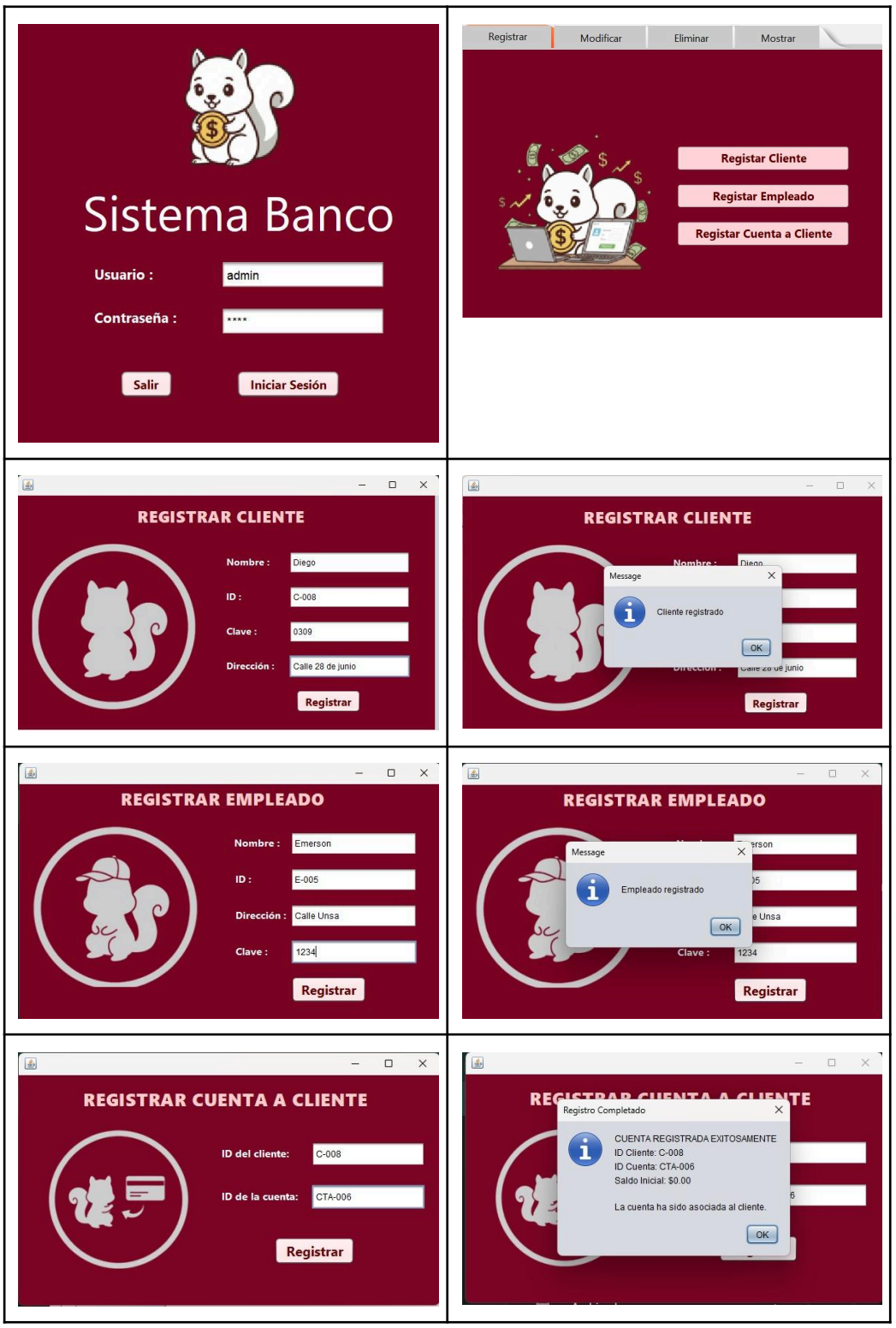

"RETIRO BANCARIO"
ID Cliente:
C-004
ID Cuenta:
CTA-005
Monto a retirar:
2900
Confirmar

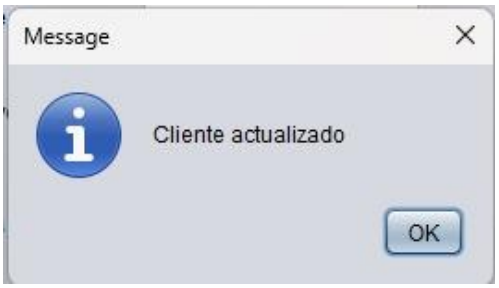

"RETIRO BANCARIO"
ID Cliente:
Message
Retiro registrado
OK
Confirmar


Gestion
Transaccion
Varios
Ver historial de una cuenta
Ver lista de clientes

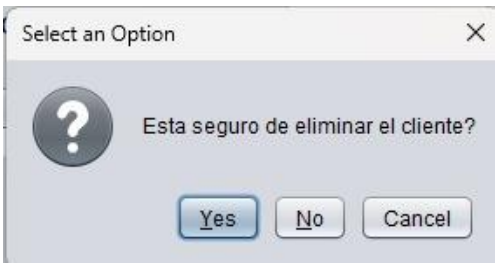

HISTORIAL DE LA CUENTA
Id de Cliente: C-004
Id de Cuenta: CTA-005
Mostrar
ID Cliente | ID Cuenta | Monto | Tipo | Fecha
C-004 | CTA-005 | 10000.0 | Depósito | Dec 14 22:58:47
C-004 | CTA-005 | 2900.0 | Retiro | Dec 14 22:59:47
HISTORIAL DE LA CUENTA
Id de Cliente: C-002
Id de Cuenta: CTA-002
Mostrar
ID Cliente | ID Cuenta | Monto | Tipo | Fecha
C-002 | CTA-002 | 3500.0 | Depósito | Dec 14 22:44:38
C-002 | CTA-002 | 200.0 | Retiro | Dec 14 22:46:00
Lista de Clientes
Mostrar Clientes
ID Cliente | Nombre | Dirección
C-001 | Ana | Av. el Sol
C-002 | Ximena | Jiron Cusco
C-004 | Jonathan | Calle Sistemas
C-005 | Pepe | Calle Bugambilias
C-006 | Juancito | Calle Perú
C-007 | Fernanda | Calle Villanueva

ADMINISTRADOR:

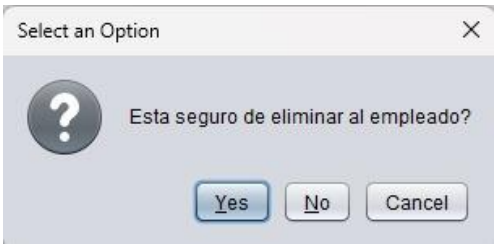


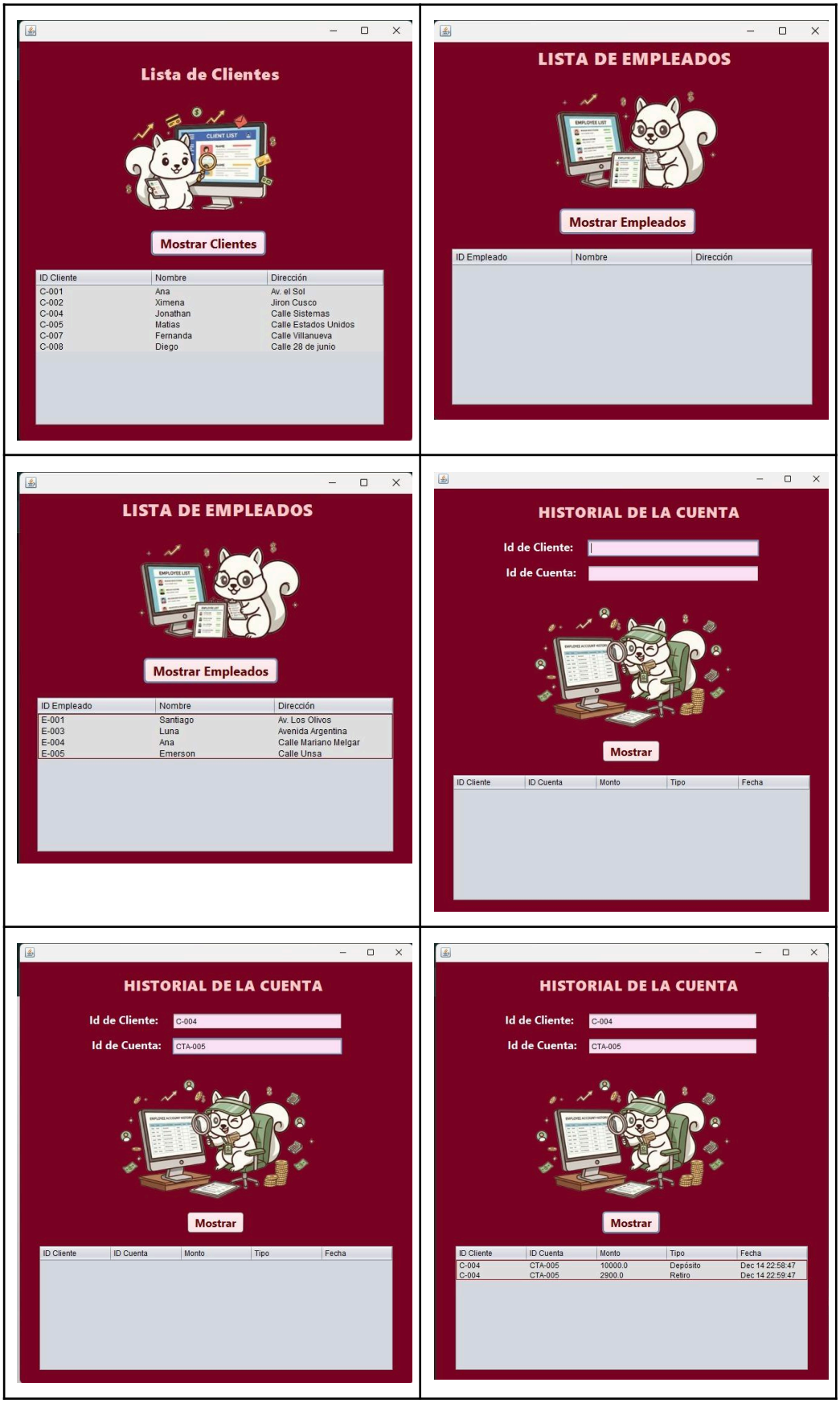


ID	Nombre	Dirección
C-006	Juancito	Calle Perú



ID	Nombre	Dirección
E-005	Emerson	Calle Unsa
E-002	Carlos	Avenida Cayma




Lista de Clientes
Mostrar Clientes
ID Cliente Nombre Dirección
C-001 Ana Av. el Sol
C-002 Ximena Jiron Cusco
C-004 Jonathan Calle Sistemas
C-005 Matias Calle Estados Unidos
C-007 Fernanda Calle Villanueva
C-008 Diego Calle 28 de junio
LISTA DE EMPLEADOS
Mostrar Empleados
ID Empleado Nombre Dirección
LISTA DE EMPLEADOS
Mostrar Empleados
ID Empleado Nombre Dirección
E-001 Santiago Av. Los Olivos
E-003 Luna Avenida Argentina
E-004 Ana Calle Mariano Melgar
E-005 Emerson Calle Unsa
HISTORIAL DE LA CUENTA
Id de Cliente:
Id de Cuenta:
Mostrar
ID Cliente ID Cuenta Monto Tipo Fecha
HISTORIAL DE LA CUENTA
Id de Cliente: C-004
Id de Cuenta: CTA-005
Mostrar
ID Cliente ID Cuenta Monto Tipo Fecha
HISTORIAL DE LA CUENTA
Id de Cliente: C-004
Id de Cuenta: CTA-005
Mostrar
ID Cliente ID Cuenta Monto Tipo Fecha
C-004 CTA-005 10000.0 Depósito Dec 14 22:58:47
C-004 CTA-005 2900.0 Retiro Dec 14 22:59:47

SQL FINAL:

Tabla administrador

Tabla cliente

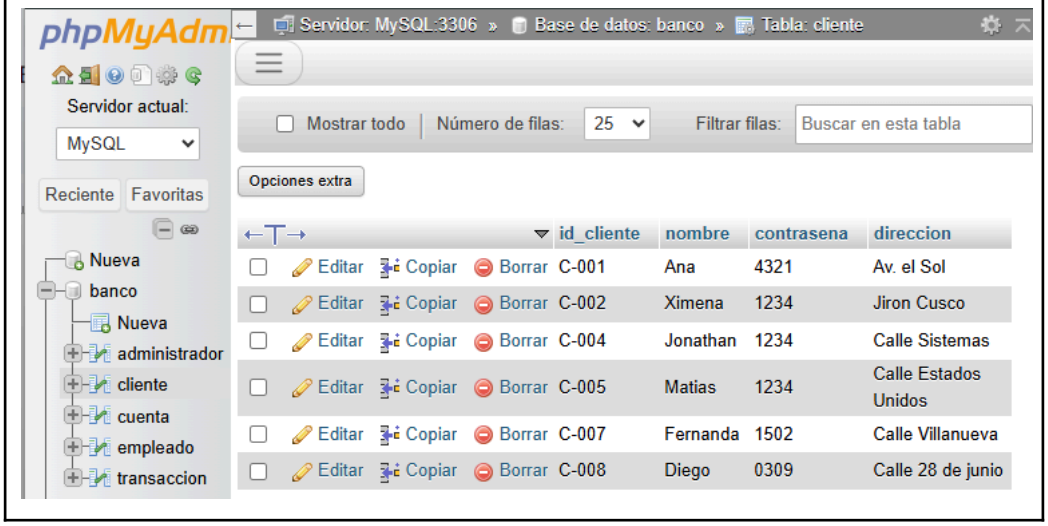


Tabla cuenta

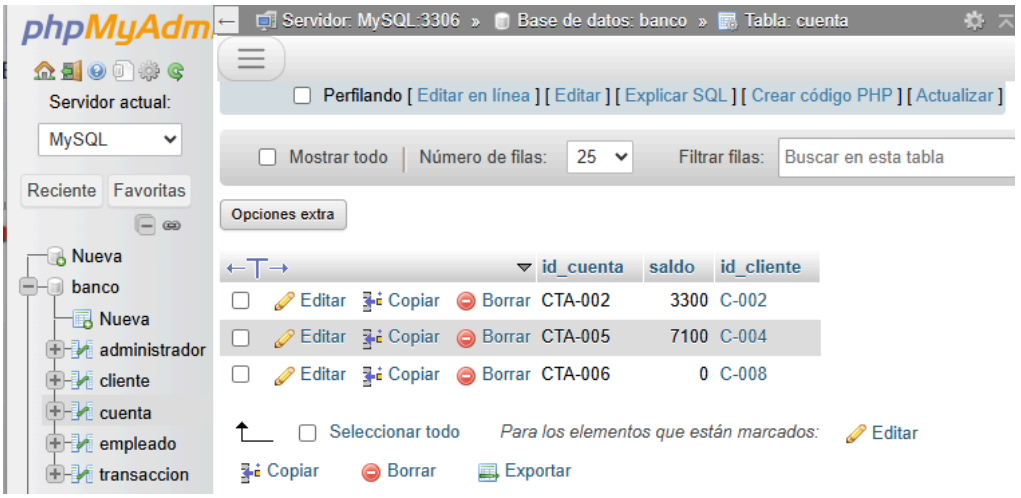


Tabla empleado

Tabla Transacciones

- **Resultados de consultas**

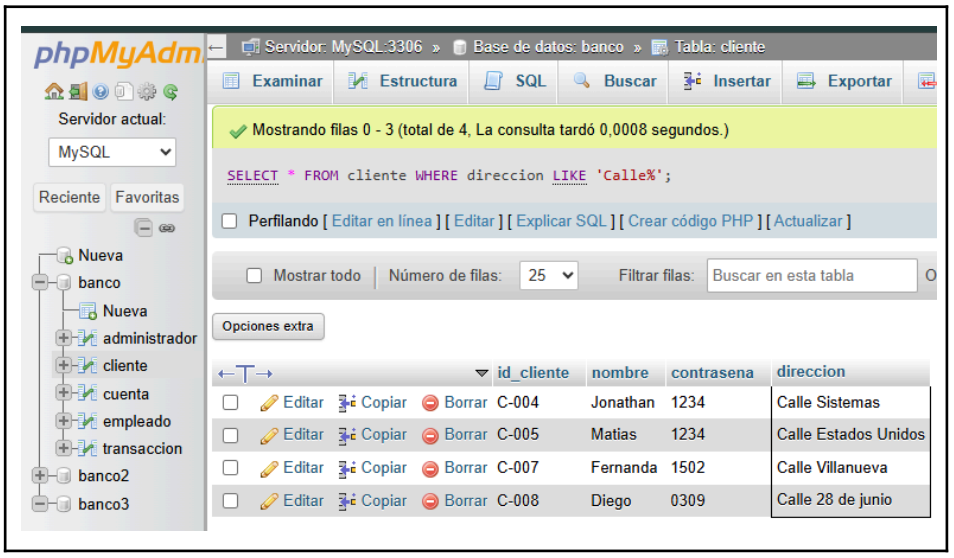
  Se muestra la ejecución de consultas SQL (SELECT) reflejadas en las ventanas del sistema, permitiendo listar clientes, empleados y cuentas registradas.

  1. **Select en tabla cliente:**

     Se realizó la ejecución de consultas SQL directamente sobre la base de datos del sistema bancario con el fin de validar el correcto almacenamiento y recuperación de la información. En este caso, se ejecutó una consulta para listar todos los clientes cuya dirección inicia con la palabra **"Calle"**, demostrando el uso de filtros mediante la cláusula `WHERE` y el operador `LIKE`.

```sql
SELECT *
FROM cliente
WHERE direccion LIKE 'Calle%';
```

     El resultado de esta consulta se refleja tanto en el gestor de base de datos como en las ventanas del sistema Java, confirmando la correcta integración entre la base de datos y la interfaz gráfica.

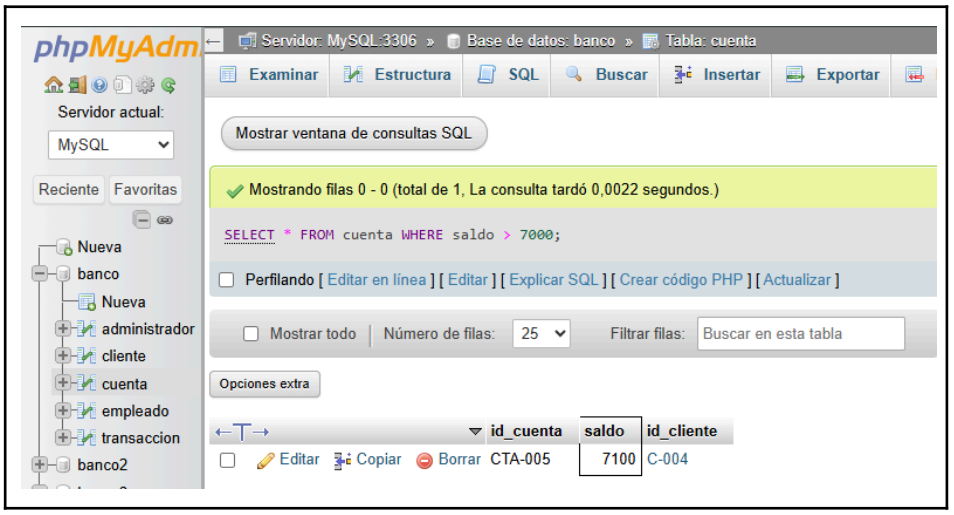


2. Select en tabla Cuenta

Se ejecutó una consulta SQL sobre la tabla cuenta con el objetivo de identificar aquellas cuentas bancarias que poseen un saldo mayor a 7000. Esta consulta permite validar el correcto registro y actualización de los saldos en la base de datos, así como la capacidad del sistema para filtrar información mediante condiciones numéricas.

```sql
SELECT *
FROM cuenta
WHERE saldo > 7000;
```

Los resultados obtenidos se reflejan en las ventanas del sistema bancario, demostrando que la información mostrada corresponde fielmente a los datos almacenados en la base de datos.



3. Select en tabla Empleado

Se realizó una consulta SQL sobre la tabla empleado con el objetivo de listar a todos los empleados cuyo nombre termina con la letra "a". Esta consulta permite demostrar el uso del operador LIKE para realizar búsquedas por patrones de texto, así como la correcta recuperación de información desde la base de datos del sistema bancario.

```
1 SELECT *
2 FROM empleado
3 WHERE nombre LIKE '%a';
```

Los resultados obtenidos confirman que el sistema puede filtrar y mostrar datos específicos de manera correcta desde la base de datos hacia la interfaz gráfica.

4. Select en tabla transacción

Se ejecutó una consulta SQL sobre la tabla transacción con la finalidad de identificar aquellas operaciones bancarias cuyo monto es menor a 3000. Esta consulta permite verificar el correcto registro de los movimientos realizados por los clientes, así como la posibilidad de filtrar transacciones según el monto involucrado.

```sql
SELECT *
FROM transaccion
WHERE monto < 3000;
```

Los resultados obtenidos se visualizan tanto en el gestor de base de datos como en las ventanas del sistema bancario, evidenciando la correcta integración entre la base de datos y la interfaz gráfica.

- **Movimientos registrados**

Se evidencian los depósitos y retiros realizados, los cuales quedan correctamente almacenados en la tabla de transacciones.

EL SQL FINAL DESPUES DE RETIROS Y DEPÓSITOS:


phpMyAdm
Servidor: MySQL:3306 » Base de datos: banco » Tabla: transaccion
Servidor actual:
MySQL
Reciente
Favoritas
Nueva
banco
Nueva
administrador
cliente
cuenta
empleado
transaccion
Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]
Mostrar todo
Número de filas: 25
Filtrar filas: Buscar en esta tabla
Ordenar según la clave:
Opciones extra
id_transaccion id_cliente id_cuenta monto tipo fecha
Editar Copiar Borrar 1 C-002 CTA-002 3500 Depósito Dec 14 22:44:38
Editar Copiar Borrar 2 C-002 CTA-002 200 Retiro Dec 14 22:46:00
Editar Copiar Borrar 3 C-004 CTA-005 10000 Depósito Dec 14 22:58:47
Editar Copiar Borrar 4 C-004 CTA-005 2900 Retiro Dec 14 22:59:47
Seleccionar todo
Para los elementos que están marcados:
Editar
Copiar
Borrar

7. DISCUSIÓN

7.1 Diferencias entre almacenamiento en memoria vs. base de datos

En el sistema anterior, la información se almacenaba en memoria volátil, es decir, en estructuras como listas o arreglos dentro del programa. Este enfoque es rápido y sencillo para pruebas, pero tiene una desventaja, al cerrar la aplicación o ante una falla del equipo, los datos se pierden. Además, el acceso concurrente es difícil de controlar y no existe un mecanismo sólido para asegurar integridad en operaciones sensibles.

En cambio, al utilizar una base de datos MySQL, los datos pasan a ser persistentes, quedando almacenados de manera permanente y centralizada. Esto permite mantener el historial de transacciones, consultar información en cualquier momento y sostener múltiples operaciones sin perder consistencia. Adicionalmente, la base de datos facilita la integridad referencial, el control de duplicados mediante claves primarias/únicas, y el uso de transacciones para asegurar que las operaciones bancarias se registren correctamente.

7.2 Problemas encontrados al conectar Java con MySQL

Durante la integración tuvimos dificultades técnicas en la conectividad y el funcionamiento como:

- Errores de conexión: credenciales incorrectas, base de datos inexistente y desactualizada.
- Consultas SQL y tipos de datos: incongruencias entre tipos Java (String, int, double) y tipos MySQL (VARCHAR, INT, DECIMAL), generaron fallas al insertar y actualizar.
- Manejo de excepciones: no lo manejamos adecuadamente al inicio, nuestro sistema se volvió inestable.

Estas dificultades nos retrasaron, sin embargo, nos ayudaron a implementar buenas prácticas como validación de datos, pruebas por partes y mensajes claros de error para el usuario.

7.3 Cómo afectaría este tipo de sistema en un banco real

Un sistema bancario real requiere alta confiabilidad, seguridad y trazabilidad. El paso de almacenamiento en memoria a una base de datos representa un cambio fundamental, ya que permite:

- Registrar permanentemente clientes, cuentas y transacciones.
- Garantizar que operaciones como depósitos/retiros se realicen de forma consistente mediante transacciones.
- Generar auditoría: historial de movimientos, reportes, seguimiento ante reclamos y cumplimiento normativo.
- Soportar multiusuario: varios cajeros o procesos trabajando simultáneamente sin perder información.

Sin embargo, en un banco real también se exigen aspectos adicionales como autenticación, cifrado, control de accesos, monitoreo, respaldos, tolerancia a fallos y escalabilidad.

7.4 Limitaciones del sistema desarrollado

A pesar de lograr la integración, el sistema desarrollado presenta limitaciones típicas de un proyecto formativo:

- Seguridad básica: no incluye un gran cifrado de contraseñas.
- Validaciones incompletas: aunque se validan campos vacíos o tipos numéricos, podrían faltar validaciones más estrictas.
- Escalabilidad: está diseñado para un entorno pequeño, no para altos volúmenes de clientes o transacciones.
- Interfaz y experiencia de usuario: no tiene una accesibilidad o diseño profesional.
- Seguridad: no tiene implementado métodos para evitar posibles manipulaciones.

En síntesis, el sistema cumple el objetivo principal, pero aún requiere mejoras para acercarse a los estándares de un entorno bancario real.

8. CONCLUSIONES

- Se logró integrar correctamente Java, una interfaz gráfica y la base de datos, permitiendo que la aplicación interactúe con datos reales almacenados de forma centralizada.
- La migración del almacenamiento en memoria hacia un almacenamiento persistente en MySQL permitió que la información de clientes, cuentas y transacciones se conserve aun cuando la aplicación se cierre, incrementando la confiabilidad del sistema.
- Las operaciones se implementaron y ejecutaron desde la GUI, comprobando que el sistema puede gestionar registros y reflejar los cambios directamente en la base de datos.
- Se incorporaron validaciones en partes requerirán lo que contribuyó a mejorar la calidad de los datos ingresados y a reducir errores.
- Se evidenció que el uso de una base de datos relacional aporta mejor organización, integridad y capacidad de consulta frente a un enfoque basado únicamente en estructuras internas del programa, facilitando el seguimiento de movimientos y el control de saldos.
- La experiencia de conexión y pruebas nos permitió identificar la importancia de una configuración adecuada y del manejo de excepciones para garantizar un funcionamiento estable de la aplicación.

9. RECOMENDACIONES

- Mejorar la seguridad del sistema, incorporando una mejor autenticación de usuarios y control de acceso por.
- Fortalecer las validaciones, incluyendo formatos, restricciones de negocio y mensajes de error claros para el usuario.
- Optimizar el diseño de la base de datos, agregando índices en campos de búsqueda frecuente.
- Mejorar la interfaz gráfica, haciendo la navegación más intuitiva, incorporando confirmaciones antes de eliminar, notificaciones de éxito y error, y tablas con opciones de filtrado y búsqueda para facilitar la gestión de registros.
- Agregar módulos de reportes y auditoría, como estados de cuenta, historial de movimientos por rango de fechas, y un registro de acciones relevantes para facilitar el control y la trazabilidad.
- Aplicar buenas prácticas de desarrollo, separando la lógica en capas y manteniendo una estructura clara del proyecto.
- Realizar pruebas adicionales, incluyendo casos límite y pruebas de concurrencia básica, para asegurar estabilidad y consistencia del sistema.

10.BIBLIOGRAFÍA

- Beaulieu, A. (2020). *Learning SQL: Generate, Manipulate, and Retrieve Data*. O'Reilly Media.
- Connolly, T., & Begg, C. (2015). *Database Systems: A Practical Approach to Design, Implementation, and Management* (6.ª ed.). Pearson.
- Elmasri, R., & Navathe, S. B. (2017). *Fundamentals of Database Systems* (7.ª ed.). Pearson.
- Horstmann, C. S. (2019). *Core Java Volume II - Advanced Features* (11.ª ed.). Pearson.
- Oracle. (2023). *Java SE Technologies - Database / JDBC*. Oracle Official Documentation. Recuperado de https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7.ª ed.). McGraw-Hill Education.
-

11. ANEXOS

A. Script SQL

```sql
CREATE TABLE IF NOT EXISTS `administrador` (
    `id_admin` varchar(20) PRIMARY KEY,
    `nombre` varchar(100) NOT NULL,
    `contrasena` varchar(10) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

insert into administrador values ("A-001", "admin", "1234");
insert into administrador values ("A-002", "admin2", "12345");

CREATE TABLE IF NOT EXISTS `cliente` (
  `id_cliente` varchar(20) NOT NULL,
  `nombre` varchar(100) NOT NULL,
  `contrasena` varchar(10) NOT NULL,
  `direccion` varchar(200) NOT NULL,
  PRIMARY KEY (`id_cliente`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS `cuenta` (
  `id_cuenta` varchar(20) NOT NULL,
  `saldo` double NOT NULL,
  `id_cliente` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id_cuenta`),
  KEY `id_cliente` (`id_cliente`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS `empleado` (
  `id_empleado` varchar(20) NOT NULL,
  `nombre` varchar(100) NOT NULL,
  `contrasena` varchar(10) NOT NULL,
  `direccion` varchar(200) NOT NULL,
  PRIMARY KEY (`id_empleado`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS `transaccion` (
  `id_transaccion` int(11) NOT NULL AUTO_INCREMENT,
  `id_cliente` varchar(20) DEFAULT NULL,
  `id_cuenta` varchar(20) DEFAULT NULL,
  `monto` double NOT NULL,
  `tipo` varchar(50) DEFAULT NULL,
  `fecha` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`id_transaccion`),
  KEY `id_cliente` (`id_cliente`),
  KEY `id_cuenta` (`id_cuenta`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

ALTER TABLE `cuenta`
    ADD CONSTRAINT cuenta_ibfk_1
    FOREIGN KEY (id_cliente) REFERENCES cliente (id_cliente)
ON DELETE CASCADE;

ALTER TABLE `transaccion`
  ADD CONSTRAINT `transaccion_ibfk_1` FOREIGN KEY (`id_cliente`) REFERENCES `cliente` (`id_cliente`),
  ADD CONSTRAINT `transaccion_ibfk_2` FOREIGN KEY (`id_cuenta`) REFERENCES `cuenta` (`id_cuenta`);
```

B. Código de la clase ConexionBD

```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to ed
4   */
5  import java.sql.*;
6  /**
7   *
8   * @author User
9   */
10 public class Conexion {
11     static String url="jdbc:mysql://localhost:3306/banco";
12     static String user = "root";
13     static String password = "";
14 
15         public static Connection contectar(){
16         Connection con = null;
17         try {
18             con = DriverManager.getConnection(url,user,password);
19             System.out.println("Conexión exitosa");
20         } catch (SQLException e){
22             e.printStackTrace();
23         }
24         return con;
25     }
}
```