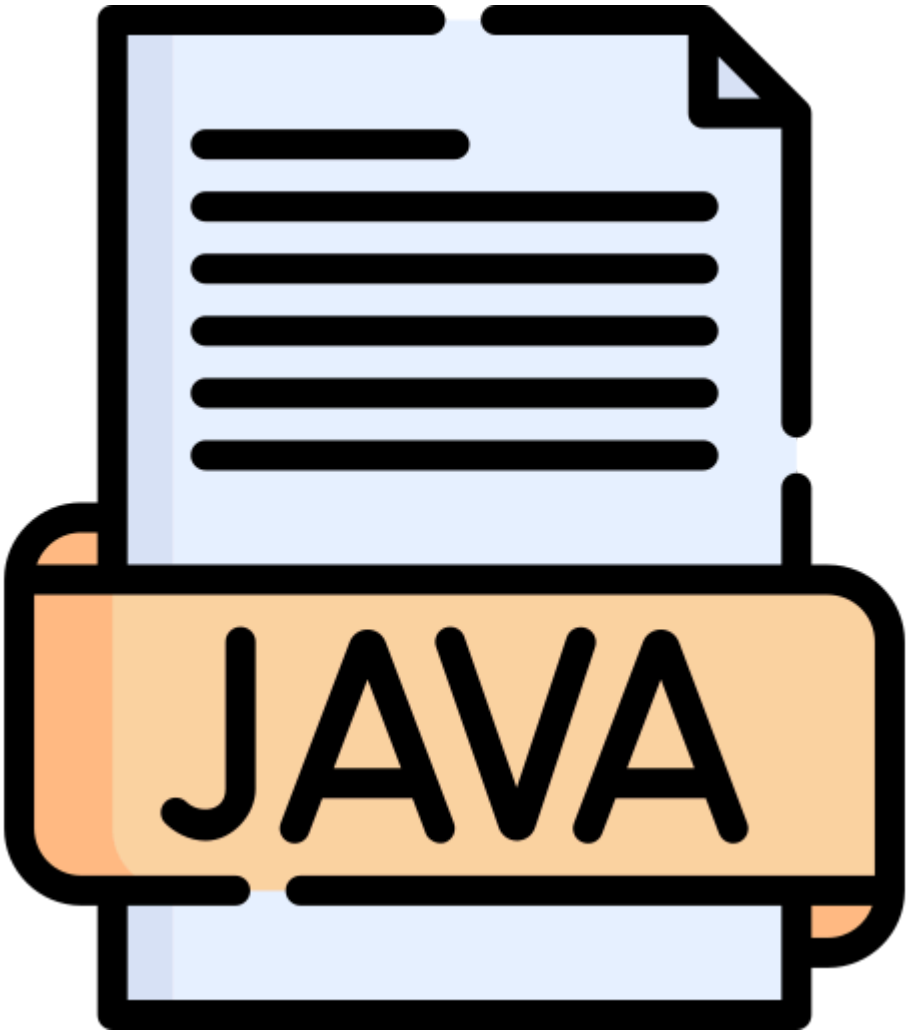
Rafael Martín Mayor

Actividad 3 UT 2 - Ampliando Aplicación Bancaria



Índice

Enunciado

Actividad 3 UT 2 - Ampliando aplicación bancaria
En esta práctica se ampliará la aplicación bancaria de la Actividad 2, añadiendo nuevas funcionalidades que hagan uso de consultas SQL con JOIN entre varias tablas.
Nuevas Tablas
Además de usuarios y cuentas, se añaden las siguientes tablas:
1. movimientos
○ idmovimiento (PK)
○ idcuenta (FK → cuentas.id)
○ fecha (TEXT o DATE)
○ tipo (TEXT: "ingreso", "retiro", "transferencia")
○ cantidad (REAL)
2. sucursales
○ idsucursal (PK)
○ nombre (TEXT)
○ direccion (TEXT)
Una cuenta puede estar asociada a una sucursal (añadir campo idsucursal en cuentas). Nuevas Funcionalidades
Dependiendo del rol, el menú se amplía con consultas que impliquen JOINs.
Menú para Empleado
Además de las opciones de la Actividad 2, se añaden:
1. Listar todos los usuarios junto con sus cuentas (JOIN usuarios–cuentas).
2. Listar todas las cuentas con su sucursal asociada (JOIN cuentas–sucursales).
3. Consultar todos los movimientos de un usuario (JOIN usuarios–cuentas–movimientos).
4. Consultar metadatos de la base de datos.
- Mostrar el nombre de las tablas existentes.
- Listar las columnas de cada tabla con su tipo de dato. (Usar DatabaseMetaData o ResultSetMetaData de JDBC.)
Ejemplo:

```
DatabaseMetaData meta = connection.getMetaData();
ResultSet tablas = meta.getTables(null, null, "%", new String[]{"TABLE"});
```

Menú para Cliente
Además de las opciones de la Actividad 2, se añaden:
1. Consultar los movimientos de una de sus cuentas.
2. Transferir dinero entre dos de sus cuentas (con transacción).
- Se deben insertar dos movimientos: uno de tipo "retiro" y otro de tipo "ingreso".
- Actualizar el saldo de ambas cuentas. Usar transacciones para garantizar la atomicidad:
- Si una operación falla, ningún cambio debe aplicarse. Utilizar setAutoCommit(false), commit() y rollback().

```
connection.setAutoCommit(false);
try {
    // retirar dinero
    // ingresar dinero
    // registrar movimientos
    connection.commit();
} catch (Exception e) {
    connection.rollback();
}
```

3. Consultar mis cuentas mostrando también la sucursal a la que pertenecen.

4. El cliente podrá usar una opción del menú para ver la estructura de sus tablas, es decir, qué columnas tiene y de qué tipo son.
Por ejemplo, después de consultar sus movimientos o sus cuentas, el programa mostrará también información como:
Columna 1: fecha (TEXT)
Columna 2: tipo (TEXT)
Columna 3: cantidad (REAL)
Para hacerlo, deberán usar los métodos getCoumnName() y getColumnTypeName() de la clase ResultSetMetaData de JDBC, que permite obtener información sobre las columnas devueltas por una consulta (nombre, tipo, número de columnas, etc.).
Requisitos Técnicos
● La base de datos tendrá al menos 4 tablas:
- usuarios(id, nombre, password, rol)
- cuentas(id, idUsuario, saldo, tipoCuenta, idSucursal)
- movimientos(id, idCuenta, fecha, tipo, cantidad)
- sucursales(id, nombre, direccion)
● Uso de JOIN en las consultas que lo requieran (no se permite resolver con varias consultas separadas).
● Uso de DAO:
- UsuarioDAO
- CuentaDAO
- MovimientoDAO
- SucursalDAO
Ejemplos de Consultas con JOIN
Usuarios con sus cuentas:
SELECT u.nombre, c.id, c.saldo, c.tipoCuenta FROM usuarios u JOIN cuentas c ON u.id = c.idUsuario;
Cuentas y sucursales:
SELECT c.id, c.saldo, c.tipoCuenta, s.nombre AS sucursal FROM cuentas c JOIN sucursales s ON c.idSucursal = s.id;
Movimientos de un usuario:
SELECT u.nombre, c.id AS cuenta, m.fecha, m.tipo, m.cantidad FROM usuarios u JOIN cuentas c ON u.id = c.idUsuario JOIN movimientos m ON c.id = m.idCuenta WHERE u.id = ?;

Explicación de las secciones del código

Clase Sucursal (Sucursal.java)

```java
public class Sucursal {
    private int idsucursal;
    private String nombre;
    private String direccion;

    public Sucursal(String nombre, String direccion) {
        this.nombre = nombre;
        this.direccion = direccion;
    }

    public Sucursal(int idsucursal, String nombre, String direccion) {
        this.idsucursal = idsucursal;
        this.nombre = nombre;
        this.direccion = direccion;
    }

    public int getIdsucursal() { return idsucursal; }
    public String getNombre() { return nombre; }
    public String getDireccion() { return direccion; }

    @Override
    public String toString() {
        return String.format("Sucursal{id=%d, nombre='%s',
direccion='%s'}",
                idsucursal, nombre, direccion);
    }
}
```

Propósito:
Representa una sucursal bancaria dentro del sistema. Es una clase modelo que sirve para mapear objetos Java con registros de la tabla “sucursales” en la base de datos.

Atributos:

- idsucursal: identificador único de la sucursal (clave primaria).
- nombre: nombre de la sucursal (por ejemplo, “Sucursal Centro”).
- direccion: dirección física (por ejemplo, “Av. España 23”).

Constructores:

- public Sucursal(String nombre, String direccion): se utiliza antes de insertar la sucursal en la base de datos (aún no tiene ID).
- public Sucursal(int idsucursal, String nombre, String direccion): se usa cuando se lee una sucursal existente de la base de datos (ya tiene ID asignado).

Getters:
Permiten acceder a los atributos privados de forma controlada: getIdsucursal(), getNombre(), getDireccion().

Método toString():
Devuelve una representación legible del objeto, útil para mostrar la información por consola al listar sucursales.

Clase SucursalDAO (SucursalDAO.java)

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class SucursalDAO {
    private final String url = "jdbc:sqlite:banco.db";

    // Crear tabla si no existe
    public SucursalDAO() {
        try (Connection conn = DriverManager.getConnection(url);
             Statement st = conn.createStatement()) {
            st.execute("""
                CREATE TABLE IF NOT EXISTS sucursales (
                    idsucursal INTEGER PRIMARY KEY AUTOINCREMENT,
                    nombre TEXT NOT NULL,
                    direccion TEXT
                )
            """);
        } catch (SQLException e) {
            System.out.println("Error creando tabla sucursales: " +
e.getMessage());
        }
    }

    // Crear sucursal
    public void crearSucursal(Sucursal s) {
```

```
        String sql = "INSERT INTO sucursales (nombre, direccion) VALUES
(?, ?)";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setString(1, s.getNombre());
            ps.setString(2, s.getDireccion());
            ps.executeUpdate();
            System.out.println("Sucursal creada correctamente.");
        } catch (SQLException e) {
            System.out.println("Error creando sucursal: " +
e.getMessage());
        }
    }

    // Listar sucursales
    public List<Sucursal> listarSucursales() {
        List<Sucursal> lista = new ArrayList<>();
        String sql = "SELECT * FROM sucursales";
        try (Connection conn = DriverManager.getConnection(url);
             Statement st = conn.createStatement();
             ResultSet rs = st.executeQuery(sql)) {
            while (rs.next()) {
                lista.add(new Sucursal(
                        rs.getInt("idsucursal"),
                        rs.getString("nombre"),
                        rs.getString("direccion")
                ));
            }
        } catch (SQLException e) {
            System.out.println("Error listando sucursales: " +
e.getMessage());
        }
        return lista;
    }
}
```

Propósito:
Se encarga del acceso a datos de la tabla “sucursales”. Aplica el patrón DAO (Data Access Object) para separar la lógica de negocio de la lógica de persistencia.

Atributo de conexión:

- private final String url = "jdbc:sqlite:banco.db";
  Define la ruta de conexión a la base de datos SQLite. Cada método abre su propia conexión con esta URL.

Constructor: crear tabla si no existe
Crea la tabla “sucursales” al inicializar el DAO si no existe. El bloque establece la conexión, ejecuta un CREATE TABLE IF NOT EXISTS y maneja excepciones SQL.

Método crearSucursal(Sucursal s):
Inserta una nueva sucursal en la tabla. Utiliza PreparedStatement para evitar inyecciones SQL y muestra un mensaje de confirmación si se ejecuta correctamente.

Método listarSucursales():
Ejecuta un SELECT * FROM sucursales, recorre el ResultSet y crea objetos Sucursal con los datos obtenidos. Devuelve una lista de todas las sucursales registradas.

Clase Movimiento (Movimiento.java)

```
public class Movimiento {
    private int idmovimiento;
    private int idcuenta;
    private String tipo;
    private double cantidad;
    private String fecha;

    public Movimiento(int idcuenta, String tipo, double cantidad) {
        this.idcuenta = idcuenta;
        this.tipo = tipo;
        this.cantidad = cantidad;
    }

    public Movimiento(int idmovimiento, int idcuenta, String tipo,
double cantidad, String fecha) {
        this.idmovimiento = idmovimiento;
        this.idcuenta = idcuenta;
        this.tipo = tipo;
        this.cantidad = cantidad;
        this.fecha = fecha;
    }

    public int getIdmovimiento() { return idmovimiento; }
    public int getIdcuenta() { return idcuenta; }
    public String getTipo() { return tipo; }
```

```
    public double getCantidad() { return cantidad; }
    public String getFecha() { return fecha; }

    @Override
    public String toString() {
        return String.format("Movimiento{id=%d, cuenta=%d, tipo='%s',
cantidad=%.2f, fecha='%s'}",
                idmovimiento, idcuenta, tipo, cantidad, fecha);
    }
}
```

Propósito:
Representa un movimiento bancario (operación sobre una cuenta): ingreso, retiro o transferencia.

Atributos:

- idmovimiento: identificador único del movimiento.
- idcuenta: referencia a la cuenta asociada (clave foránea).
- tipo: tipo de operación (“ingreso”, “retiro” o “transferencia”).
- cantidad: importe del movimiento (positivo o negativo).
- fecha: marca temporal (CURRENT_TIMESTAMP en la base de datos).

Constructores:

- public Movimiento(int idcuenta, String tipo, double cantidad): se usa al crear un nuevo movimiento antes de insertarlo.
- public Movimiento(int idmovimiento, int idcuenta, String tipo, double cantidad, String fecha): se usa al leer movimientos desde la base de datos.

Método toString():
Devuelve una representación textual legible del movimiento, mostrando id, cuenta, tipo, cantidad y fecha.

Clase MovimientoDAO (MovimientoDAO.java)

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class MovimientoDAO {
    private final String url = "jdbc:sqlite:banco.db";
```

```
    // Crear tabla si no existe
    public MovimientoDAO() {
        try (Connection conn = DriverManager.getConnection(url);
             Statement st = conn.createStatement()) {
            st.execute("""
                CREATE TABLE IF NOT EXISTS movimientos (
                    idmovimiento INTEGER PRIMARY KEY AUTOINCREMENT,
                    idcuenta INTEGER,
                    tipo TEXT,
                    cantidad REAL,
                    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                    FOREIGN KEY(idcuenta) REFERENCES cuentas(idcuenta)
                )
            """);
        } catch (SQLException e) {
            System.out.println("Error creando tabla movimientos: " +
e.getMessage());
        }
    }

    // Registrar movimiento
    public void registrarMovimiento(Movimiento m) {
        String sql = "INSERT INTO movimientos (idcuenta, tipo,
cantidad) VALUES (?, ?, ?)";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, m.getIdcuenta());
            ps.setString(2, m.getTipo());
            ps.setDouble(3, m.getCantidad());
            ps.executeUpdate();
            System.out.println("Movimiento registrado: " + m.getTipo()
+ " (" + m.getCantidad() + "€)");
        } catch (SQLException e) {
            System.out.println("Error registrando movimiento: " +
e.getMessage());
        }
    }

    // Listar todos los movimientos
    public List<Movimiento> listarMovimientos() {
        List<Movimiento> lista = new ArrayList<>();
        String sql = "SELECT * FROM movimientos ORDER BY fecha DESC";
```

```
        try (Connection conn = DriverManager.getConnection(url);
             Statement st = conn.createStatement();
             ResultSet rs = st.executeQuery(sql)) {
            while (rs.next()) {
                lista.add(new Movimiento(
                        rs.getInt("idmovimiento"),
                        rs.getInt("idcuenta"),
                        rs.getString("tipo"),
                        rs.getDouble("cantidad"),
                        rs.getString("fecha")
                ));
            }
        } catch (SQLException e) {
            System.out.println("Error listando movimientos: " +
e.getMessage());
        }
        return lista;
    }

    // Listar movimientos de un usuario (JOIN)
    public List<Movimiento> movimientosDeUsuario(int idUsuario) {
        List<Movimiento> lista = new ArrayList<>();
        String sql = """
            SELECT m.idmovimiento, m.idcuenta, m.tipo, m.cantidad,
m.fecha
            FROM movimientos m
            JOIN cuentas c ON m.idcuenta = c.idcuenta
            WHERE c.idpropietario = ?
            ORDER BY m.fecha DESC
        """;
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, idUsuario);
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                lista.add(new Movimiento(
                        rs.getInt("idmovimiento"),
                        rs.getInt("idcuenta"),
                        rs.getString("tipo"),
                        rs.getDouble("cantidad"),
                        rs.getString("fecha")
                ));
            }
```

```
        } catch (SQLException e) {
            System.out.println("Error obteniendo movimientos del
usuario: " + e.getMessage());
        }
        return lista;
    }
}
```

Propósito:
Gestiona la tabla “movimientos” en la base de datos. Aplica el patrón DAO para registrar y consultar movimientos.

Atributo de conexión:

- private final String url = "jdbc:sqlite:banco.db";
  Indica la ruta de conexión a la base de datos SQLite.

Constructor: crear tabla si no existe
Crea la tabla “movimientos” si no existe. Define las columnas idmovimiento, idcuenta, tipo, cantidad, fecha y la clave foránea hacia la tabla cuentas.

Método registrarMovimiento(Movimiento m):
Inserta un nuevo registro en la tabla movimientos con los campos idcuenta, tipo y cantidad.
La columna fecha se rellena automáticamente por SQLite con el valor actual.
Muestra un mensaje por consola confirmando el registro.

Método listarMovimientos():
Ejecuta un SELECT * FROM movimientos ORDER BY fecha DESC.
Crea un objeto Movimiento por cada fila del resultado y devuelve una lista de todos los movimientos.

Método movimientosDeUsuario(int idUsuario):
Ejecuta una consulta SQL con JOIN entre movimientos y cuentas para obtener los movimientos de las cuentas pertenecientes a un usuario específico.
Filtra con WHERE c.idpropietario = ?, usa un PreparedStatement y devuelve una lista de movimientos ordenada por fecha descendente.

Clase Principal (Main.java)

```
import java.sql.*;
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        UsuarioDAO usuarioDAO = new UsuarioDAO();
        CuentaDAO cuentaDAO = new CuentaDAO();
        MovimientoDAO movimientoDAO = new MovimientoDAO();
        SucursalDAO sucursalDAO = new SucursalDAO();

        System.out.println("Bienvenido al Banco JDBC");
        System.out.print("Usuario: ");
        String nombre = sc.nextLine();
        System.out.print("Contraseña: ");
        String pass = sc.nextLine();

        Usuario user = usuarioDAO.autenticar(nombre, pass);

        if (user == null) {
            System.out.println("Credenciales incorrectas.");
            return;
        }

        System.out.println("Bienvenido, " + user.getNombre() + " (" +
user.getRol() + ")");

        if (user.getRol().equalsIgnoreCase("empleado")) {
            menuEmpleado(sc, usuarioDAO, cuentaDAO, movimientoDAO,
sucursalDAO);
        } else {
            menuCliente(sc, cuentaDAO, movimientoDAO, user);
        }
    }

    // MENÚ EMPLEADO
    private static void menuEmpleado(Scanner sc, UsuarioDAO usuarioDAO,
                                     CuentaDAO cuentaDAO, MovimientoDAO
movDAO,
                                     SucursalDAO sucDAO) {
        int opcion;
```

```
        do {
            System.out.println("""
            \nMENÚ EMPLEADO
            1. Crear nuevo usuario
            2. Listar usuarios
            3. Crear cuenta bancaria
            4. Listar cuentas
            5. Consultar cuentas de un usuario
            6. Eliminar usuario
            7. Listar usuarios con sus cuentas
            8. Listar cuentas con su sucursal
            9. Consultar movimientos de un usuario
            10. Ver metadatos (tablas y columnas)
            11. Crear nueva sucursal
            12. Listar sucursales
            13. Salir
            """);
            System.out.print("Opción: ");
            opcion = Integer.parseInt(sc.nextLine());

            switch (opcion) {
                case 1 -> {
                    System.out.print("Nombre: ");
                    String nombre = sc.nextLine();
                    System.out.print("Contraseña: ");
                    String pass = sc.nextLine();
                    System.out.print("Rol (empleado/cliente): ");
                    String rol = sc.nextLine();
                    usuarioDAO.crearUsuario(new Usuario(nombre, pass,
rol));
                }
                case 2 ->
usuarioDAO.listarUsuarios().forEach(System.out::println);
                case 3 -> {
                    System.out.print("ID del propietario: ");
                    int id = Integer.parseInt(sc.nextLine());
                    System.out.print("Tipo de cuenta: ");
                    String tipo = sc.nextLine();
                    System.out.print("ID de sucursal: ");
                    int idsuc = Integer.parseInt(sc.nextLine());
                    cuentaDAO.crearCuenta(new Cuenta(id, 0.0, tipo,
idsuc));
                }
```

```
                case 4 ->
cuentaDAO.listarCuentas().forEach(System.out::println);
                case 5 -> {
                    System.out.print("ID de usuario: ");
                    int id = Integer.parseInt(sc.nextLine());

cuentaDAO.obtenerCuentasUsuario(id).forEach(System.out::println);
                }
                case 6 -> {
                    System.out.print("ID de usuario a eliminar: ");
                    int id = Integer.parseInt(sc.nextLine());
                    usuarioDAO.eliminarUsuario(id);
                }
                case 7 -> cuentaDAO.listarUsuariosConCuentas();
                case 8 -> cuentaDAO.listarCuentasConSucursal();
                case 9 -> {
                    System.out.print("ID de usuario: ");
                    int id = Integer.parseInt(sc.nextLine());

movDAO.movimientosDeUsuario(id).forEach(System.out::println);
                }
                case 10 -> mostrarMetadatos();
                case 11 -> {
                    System.out.print("Nombre de sucursal: ");
                    String n = sc.nextLine();
                    System.out.print("Dirección: ");
                    String d = sc.nextLine();
                    sucDAO.crearSucursal(new Sucursal(n, d));
                }
                case 12 ->
sucDAO.listarSucursales().forEach(System.out::println);
                case 13 -> System.out.println("Saliendo del menú
empleado...");
                default -> System.out.println("Opción no válida.");
            }
        } while (opcion != 13);
    }

    // MENÚ CLIENTE
    private static void menuCliente(Scanner sc, CuentaDAO cuentaDAO,
MovimientoDAO movDAO, Usuario user) {
        int opcion;
        do {
```

```
            System.out.println("""
            \nMENÚ CLIENTE
            1. Consultar mis cuentas (con sucursal)
            2. Consultar saldo
            3. Ingresar dinero
            4. Retirar dinero
            5. Ver mis movimientos
            6. Transferir entre mis cuentas
            7. Ver estructura de tabla (movimientos o cuentas)
            8. Salir
            """);
            System.out.print("Opción: ");
            opcion = Integer.parseInt(sc.nextLine());

            switch (opcion) {
                case 1 ->
cuentaDAO.listarCuentasConSucursalDeUsuario(user.getIdusuario());
                case 2 -> {
                    System.out.print("ID de cuenta: ");
                    int id = Integer.parseInt(sc.nextLine());
                    double saldo = cuentaDAO.consultarSaldo(id);
                    System.out.printf("Saldo actual: %.2f€%n", saldo);
                }
                case 3 -> {
                    System.out.print("ID de cuenta: ");
                    int id = Integer.parseInt(sc.nextLine());
                    System.out.print("Cantidad a ingresar: ");
                    double cant = Double.parseDouble(sc.nextLine());
                    cuentaDAO.ingresar(id, cant);
                    movDAO.registrarMovimiento(new Movimiento(id,
"ingreso", cant));
                }
                case 4 -> {
                    System.out.print("ID de cuenta: ");
                    int id = Integer.parseInt(sc.nextLine());
                    System.out.print("Cantidad a retirar: ");
                    double cant = Double.parseDouble(sc.nextLine());
                    cuentaDAO.retirar(id, cant);
                    movDAO.registrarMovimiento(new Movimiento(id,
"retiro", cant));
                }
```

```
                case 5 ->
movDAO.movimientosDeUsuario(user.getIdusuario()).forEach(System.out::pr
intln);
                case 6 -> transferir(sc, user);
                case 7 -> mostrarMetadatos();
                case 8 -> System.out.println("Saliendo del menú
cliente...");
                default -> System.out.println("Opción no válida.");
            }
        } while (opcion != 8);
    }

    // Transferencia con transacción
    private static void transferir(Scanner sc, Usuario user) {
        System.out.print("Cuenta origen: ");
        int origen = Integer.parseInt(sc.nextLine());
        System.out.print("Cuenta destino: ");
        int destino = Integer.parseInt(sc.nextLine());
        System.out.print("Cantidad a transferir: ");
        double cantidad = Double.parseDouble(sc.nextLine());

        String url = "jdbc:sqlite:banco.db";
        try (Connection conn = DriverManager.getConnection(url)) {
            conn.setAutoCommit(false);

            PreparedStatement retirar = conn.prepareStatement("UPDATE
cuentas SET saldo = saldo - ? WHERE idcuenta = ?");
            PreparedStatement ingresar = conn.prepareStatement("UPDATE
cuentas SET saldo = saldo + ? WHERE idcuenta = ?");
            PreparedStatement mov = conn.prepareStatement("INSERT INTO
movimientos (idcuenta, tipo, cantidad) VALUES (?, ?, ?)");

            retirar.setDouble(1, cantidad);
            retirar.setInt(2, origen);
            ingresar.setDouble(1, cantidad);
            ingresar.setInt(2, destino);

            retirar.executeUpdate();
            ingresar.executeUpdate();

            mov.setInt(1, origen);
            mov.setString(2, "transferencia");
            mov.setDouble(3, -cantidad);
```

```
            mov.executeUpdate();

            mov.setInt(1, destino);
            mov.setString(2, "transferencia");
            mov.setDouble(3, cantidad);
            mov.executeUpdate();

            conn.commit();
            System.out.println("Transferencia realizada
correctamente.");
        } catch (Exception e) {
            System.out.println("Error en la transferencia: " +
e.getMessage());
        }
    }

    // Metadatos (tablas y columnas)
    private static void mostrarMetadatos() {
        String url = "jdbc:sqlite:banco.db";
        try (Connection conn = DriverManager.getConnection(url)) {
            DatabaseMetaData meta = conn.getMetaData();
            ResultSet tablas = meta.getTables(null, null, "%", new
String[]{"TABLE"});
            while (tablas.next()) {
                String nombreTabla = tablas.getString("TABLE_NAME");
                System.out.println("\nTabla: " + nombreTabla);
                ResultSet columnas = meta.getColumns(null, null,
nombreTabla, "%");
                while (columnas.next()) {
                    System.out.printf(" - %s (%s)%n",
                        columnas.getString("COLUMN_NAME"),
                        columnas.getString("TYPE_NAME"));
                }
            }
        } catch (SQLException e) {
            System.out.println("Error leyendo metadatos: " +
e.getMessage());
        }
    }
}
```

El nuevo archivo Main.java amplía y mejora significativamente la funcionalidad de la aplicación bancaria, incorporando nuevas clases, menús y operaciones que permiten una gestión más completa y profesional del sistema.

1. Nuevos DAO y entidades

Además de UsuarioDAO y CuentaDAO, se agregan MovimientoDAO y SucursalDAO, junto con las clases Movimiento y Sucursal.
Esto permite registrar movimientos bancarios (ingresos, retiros, transferencias) y asociar las cuentas a una sucursal. De esta forma, la aplicación pasa a tener un modelo de datos más realista y estructurado.

2. Menú del empleado

El menú para empleados aumenta de 7 a 13 opciones, incluyendo:

Listar usuarios con sus cuentas.

Listar cuentas junto con la sucursal.

Consultar movimientos de un usuario.

Crear y listar sucursales.

Visualizar metadatos de la base de datos (tablas y columnas).

Estas funciones otorgan al empleado un control total del sistema, facilitando tareas administrativas y de mantenimiento.

3. Menú del cliente

El menú del cliente también se amplía, pasando de 5 a 8 opciones.
Ahora, el cliente puede:

Ver sus cuentas junto con la sucursal.

Consultar sus movimientos.

Realizar transferencias entre sus propias cuentas.

Explorar la estructura de las tablas mediante metadatos.

Además, los ingresos y retiros registran automáticamente un movimiento en la base de datos, mejorando la trazabilidad.

4. Nuevas operaciones JDBC

Se implementa el método transferir() usando transacciones JDBC. Esto asegura que las operaciones de retiro e ingreso se ejecuten de forma atómica, manteniendo la integridad de los datos.
También se añade mostrarMetadatos(), que utiliza DatabaseMetaData para mostrar las tablas y columnas de la base de datos, ayudando en la depuración y análisis del sistema.

Clase Usuario (Usuario.java)

```
public class Usuario {
    private int idusuario;
    private String nombre;
    private String password;
    private String rol; // "empleado" o "cliente"

    public Usuario() {}

    public Usuario(String nombre, String password, String rol) {
        this.nombre = nombre;
        this.password = password;
        this.rol = rol;
    }

    public Usuario(int idusuario, String nombre, String password,
String rol) {
        this(nombre, password, rol);
        this.idusuario = idusuario;
    }

    public int getIdusuario() { return idusuario; }
    public void setIdusuario(int idusuario) { this.idusuario =
idusuario; }

    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }

    public String getPassword() { return password; }
    public void setPassword(String password) { this.password =
password; }

    public String getRol() { return rol; }
    public void setRol(String rol) { this.rol = rol; }

    @Override
```

```
    public String toString() {
        return String.format("[%d] %s (%s)", idusuario, nombre, rol);
    }
}
```

No han habido cambios de ningún tipo en Usuario.java. El fichero permanece como en la anterior práctica.

Clase UsuarioDAO (UsuarioDAO.java)

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class UsuarioDAO {
    private final String url = "jdbc:sqlite:banco.db";

    public UsuarioDAO() {
        crearTablas();
        crearAdminPorDefecto();
    }

    private void crearTablas() {
        String sqlUsuarios = """
            CREATE TABLE IF NOT EXISTS usuarios (
                idusuario INTEGER PRIMARY KEY AUTOINCREMENT,
                nombre TEXT NOT NULL UNIQUE,
                password TEXT NOT NULL,
                rol TEXT CHECK(rol IN ('empleado','cliente')) NOT NULL
            );
        """;

        String sqlSucursales = """
            CREATE TABLE IF NOT EXISTS sucursales (
                idsucursal INTEGER PRIMARY KEY AUTOINCREMENT,
                nombre TEXT NOT NULL,
                direccion TEXT
            );
        """;

        String sqlCuentas = """
            CREATE TABLE IF NOT EXISTS cuentas (
```

```
                idcuenta INTEGER PRIMARY KEY AUTOINCREMENT,
                idpropietario INTEGER NOT NULL,
                saldo REAL DEFAULT 0,
                tipoCuenta TEXT,
                idsucursal INTEGER,
                FOREIGN KEY (idpropietario) REFERENCES
usuarios(idusuario) ON DELETE CASCADE,
                FOREIGN KEY (idsucursal) REFERENCES
sucursales(idsucursal)
            );
        """;

        String sqlMovimientos = """
            CREATE TABLE IF NOT EXISTS movimientos (
                idmovimiento INTEGER PRIMARY KEY AUTOINCREMENT,
                idcuenta INTEGER,
                tipo TEXT,
                cantidad REAL,
                fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                FOREIGN KEY(idcuenta) REFERENCES cuentas(idcuenta)
            );
        """;

        try (Connection conn = DriverManager.getConnection(url);
            Statement stmt = conn.createStatement()) {
            stmt.execute(sqlUsuarios);
            stmt.execute(sqlSucursales);
            stmt.execute(sqlCuentas);
            stmt.execute(sqlMovimientos);
        } catch (SQLException e) {
            System.out.println("Error creando tablas: " +
e.getMessage());
        }
    }


    private void crearAdminPorDefecto() {
        String check = "SELECT COUNT(*) FROM usuarios WHERE nombre =
'admin'";
        try (Connection conn = DriverManager.getConnection(url);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(check)) {
            if (rs.next() && rs.getInt(1) == 0) {
```

```
                String insert = "INSERT INTO usuarios (nombre,
password, rol) VALUES ('admin', '1234', 'empleado')";
                stmt.executeUpdate(insert);
                System.out.println("Usuario administrador creado
automáticamente (admin / 1234).");
            }
        } catch (SQLException e) {
            System.out.println("Error creando admin: " +
e.getMessage());
        }
    }

    public void crearUsuario(Usuario u) {
        String sql = "INSERT INTO usuarios (nombre, password, rol)
VALUES (?, ?, ?)";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, u.getNombre());
            pstmt.setString(2, u.getPassword());
            pstmt.setString(3, u.getRol());
            pstmt.executeUpdate();
            System.out.println("Usuario creado correctamente.");
        } catch (SQLException e) {
            System.out.println("Error creando usuario: " +
e.getMessage());
        }
    }

    public List<Usuario> listarUsuarios() {
        List<Usuario> lista = new ArrayList<>();
        String sql = "SELECT * FROM usuarios";
        try (Connection conn = DriverManager.getConnection(url);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {
            while (rs.next()) {
                lista.add(new Usuario(
                    rs.getInt("idusuario"),
                    rs.getString("nombre"),
                    rs.getString("password"),
                    rs.getString("rol")
                ));
            }
        } catch (SQLException e) {
```

```
            System.out.println("Error listando usuarios: " +
e.getMessage());
        }
        return lista;
    }

    public Usuario autenticar(String nombre, String password) {
        String sql = "SELECT * FROM usuarios WHERE nombre = ? AND
password = ?";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, nombre);
            pstmt.setString(2, password);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                return new Usuario(
                    rs.getInt("idusuario"),
                    rs.getString("nombre"),
                    rs.getString("password"),
                    rs.getString("rol")
                );
            }
        } catch (SQLException e) {
            System.out.println("Error autenticando: " +
e.getMessage());
        }
        return null;
    }

    public void eliminarUsuario(int idusuario) {
        String sql1 = "DELETE FROM cuentas WHERE idpropietario = ?";
        String sql2 = "DELETE FROM usuarios WHERE idusuario = ?";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement pstmt1 = conn.prepareStatement(sql1);
             PreparedStatement pstmt2 = conn.prepareStatement(sql2)) {
            pstmt1.setInt(1, idusuario);
            pstmt1.executeUpdate();
            pstmt2.setInt(1, idusuario);
            pstmt2.executeUpdate();
            System.out.println("Usuario eliminado (y sus cuentas
asociadas).");
        } catch (SQLException e) {
```

```
            System.out.println("Error eliminando usuario: " +
e.getMessage());
        }
    }
}
```

Cambios en UsuarioDAO.java:

Nuevas tablas: sucursales y movimientos, en cuentas se añadió la columan idsucursal y relación con sucursales.

Clase Cuenta (Cuenta.java)

```
public class Cuenta {
    private int idcuenta;
    private int idpropietario;
    private double saldo;
    private String tipoCuenta;
    private int idsucursal;

    public Cuenta() {}

    public Cuenta(int idpropietario, double saldo, String tipoCuenta,
int idsucursal) {
        this.idpropietario = idpropietario;
        this.saldo = saldo;
        this.tipoCuenta = tipoCuenta;
        this.idsucursal = idsucursal;
    }

    // Sobrecarga sin sucursal (por compatibilidad con versiones
previas)
    public Cuenta(int idpropietario, double saldo, String tipoCuenta) {
        this(idpropietario, saldo, tipoCuenta, 0);
    }

    public Cuenta(int idcuenta, int idpropietario, double saldo, String
tipoCuenta, int idsucursal) {
        this(idpropietario, saldo, tipoCuenta, idsucursal);
        this.idcuenta = idcuenta;
    }
```

```
    public int getIdcuenta() { return idcuenta; }
    public void setIdcuenta(int idcuenta) { this.idcuenta = idcuenta; }

    public int getIdpropietario() { return idpropietario; }
    public void setIdpropietario(int idpropietario) {
this.idpropietario = idpropietario; }

    public double getSaldo() { return saldo; }
    public void setSaldo(double saldo) { this.saldo = saldo; }

    public String getTipoCuenta() { return tipoCuenta; }
    public void setTipoCuenta(String tipoCuenta) { this.tipoCuenta =
tipoCuenta; }

    public int getIdsucursal() { return idsucursal; }
    public void setIdsucursal(int idsucursal) { this.idsucursal =
idsucursal; }

    @Override
    public String toString() {
        return String.format("Cuenta #%d | Propietario ID: %d | Tipo:
%s | Saldo: %.2f€ | Sucursal ID: %d",
                idcuenta, idpropietario, tipoCuenta, saldo,
idsucursal);
    }
}
```

Cambios en Cuenta.java:

1. Nueva propiedad idsucursal

Se añadió un atributo idsucursal para asociar la cuenta a una sucursal concreta:

private int idsucursal;

2. Nuevos constructores para manejar sucursales

Constructor principal:

public Cuenta(int idpropietario, double saldo, String tipoCuenta, int idsucursal)

Permite crear una cuenta asociada a una sucursal desde el inicio.

Sobrecarga para compatibilidad:

public Cuenta(int idpropietario, double saldo, String tipoCuenta)

Llama al constructor principal con idsucursal = 0 para mantener compatibilidad con versiones anteriores donde no existía la sucursal.

Constructor completo con id de cuenta y sucursal:

public Cuenta(int idcuenta, int idpropietario, double saldo, String tipoCuenta, int idsucursal)

Sustituye al constructor anterior que no incluía idsucursal.

3. Nuevos getters y setters

Se agregaron métodos para acceder y modificar idsucursal:

public int getIdsucursal() { return idsucursal; }
public void setIdsucursal(int idsucursal) { this.idsucursal = idsucursal; }

4. Modificación del toString()

Ahora incluye información de la sucursal:

return String.format("Cuenta #%d | Propietario ID: %d | Tipo: %s | Saldo: %.2f€ | Sucursal ID: %d",
idcuenta, idpropietario, tipoCuenta, saldo, idsucursal);

Clase CuentaDAO (CuentaDAO.java)

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class CuentaDAO {
    private final String url = "jdbc:sqlite:banco.db";

    public void crearCuenta(Cuenta c) {
        String sql = "INSERT INTO cuentas (idpropietario, saldo,
tipoCuenta, idsucursal) VALUES (?, ?, ?, ?)";
        try (Connection conn = DriverManager.getConnection(url);
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
```

```
            pstmt.setInt(1, c.getIdpropietario());
            pstmt.setDouble(2, c.getSaldo());
            pstmt.setString(3, c.getTipoCuenta());
            pstmt.setInt(4, c.getIdsucursal());
            pstmt.executeUpdate();
            System.out.println("Cuenta creada correctamente.");
        } catch (SQLException e) {
            System.out.println("Error creando cuenta: " +
e.getMessage());
        }
    }

    public List<Cuenta> listarCuentas() {
        return obtenerLista("SELECT * FROM cuentas");
    }

    public List<Cuenta> obtenerCuentasUsuario(int idUsuario) {
        return obtenerLista("SELECT * FROM cuentas WHERE idpropietario
= " + idUsuario);
    }

    public double consultarSaldo(int idcuenta) {
        String sql = "SELECT saldo FROM cuentas WHERE idcuenta = ?";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, idcuenta);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) return rs.getDouble("saldo");
        } catch (SQLException e) {
            System.out.println("Error consultando saldo: " +
e.getMessage());
        }
        return -1;
    }

    public void ingresar(int idcuenta, double cantidad) {
        actualizarSaldo(idcuenta, cantidad, true);
    }

    public void retirar(int idcuenta, double cantidad) {
        actualizarSaldo(idcuenta, cantidad, false);
    }
```

```
    private void actualizarSaldo(int idcuenta, double cantidad, boolean
ingresar) {
        String operacion = ingresar ? "Ingreso" : "Retiro";
        double signo = ingresar ? cantidad : -cantidad;

        String sql = "UPDATE cuentas SET saldo = saldo + ? WHERE
idcuenta = ?";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setDouble(1, signo);
            pstmt.setInt(2, idcuenta);
            pstmt.executeUpdate();
            System.out.printf("%s de %.2f€ realizado correctamente.%n",
operacion, cantidad);
        } catch (SQLException e) {
            System.out.println("Error en " + operacion.toLowerCase() +
": " + e.getMessage());
        }
    }

    private List<Cuenta> obtenerLista(String sql) {
        List<Cuenta> lista = new ArrayList<>();
        try (Connection conn = DriverManager.getConnection(url);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {
            while (rs.next()) {
                lista.add(new Cuenta(
                    rs.getInt("idcuenta"),
                    rs.getInt("idpropietario"),
                    rs.getDouble("saldo"),
                    rs.getString("tipoCuenta"),
                    rs.getInt("idsucursal")
                ));
            }
        } catch (SQLException e) {
            System.out.println("Error obteniendo cuentas: " +
e.getMessage());
        }
        return lista;
    }

    public void listarUsuariosConCuentas() {
        String sql = """
```

```
            SELECT u.nombre, c.idcuenta, c.saldo, c.tipoCuenta
            FROM usuarios u
            JOIN cuentas c ON u.idusuario = c.idpropietario
        """;
        try (Connection conn = DriverManager.getConnection(url);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql)) {
            while (rs.next()) {
                System.out.printf("%s -> Cuenta %d (%.2f€, %s)%n",
                    rs.getString("nombre"), rs.getInt("idcuenta"),
                    rs.getDouble("saldo"), rs.getString("tipoCuenta"));
            }
        } catch (SQLException e) {
            System.out.println("Error listando usuarios con cuentas: "
+ e.getMessage());
        }
    }

    public void listarCuentasConSucursal() {
        String sql = """
            SELECT c.idcuenta, c.saldo, c.tipoCuenta, s.nombre AS
sucursal
            FROM cuentas c
            LEFT JOIN sucursales s ON c.idsucursal = s.idsucursal
        """;
        try (Connection conn = DriverManager.getConnection(url);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql)) {
            while (rs.next()) {
                System.out.printf("Cuenta %d | %.2f€ | %s | Sucursal:
%s%n",
                    rs.getInt("idcuenta"),
                    rs.getDouble("saldo"),
                    rs.getString("tipoCuenta"),
                    rs.getString("sucursal"));
            }
        } catch (SQLException e) {
            System.out.println("Error listando cuentas con sucursal: "
+ e.getMessage());
        }
    }

    public void listarCuentasConSucursalDeUsuario(int idUsuario) {
```

```
        String sql = """
            SELECT c.idcuenta, c.saldo, c.tipoCuenta, s.nombre AS
sucursal
            FROM cuentas c
            LEFT JOIN sucursales s ON c.idsucursal = s.idsucursal
            WHERE c.idpropietario = ?
        """;
        try (Connection conn = DriverManager.getConnection(url);
            PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, idUsuario);
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                System.out.printf("Cuenta %d | %.2f€ | %s | Sucursal:
%s%n",
                    rs.getInt("idcuenta"),
                    rs.getDouble("saldo"),
                    rs.getString("tipoCuenta"),
                    rs.getString("sucursal"));
            }
        } catch (SQLException e) {
            System.out.println("Error listando cuentas del usuario: " +
e.getMessage());
        }
    }
}
```

Cambios en CuentaDAO.java:

1. Soporte para sucursales

Se añade soporte para el campo idsucursal en todos los métodos que crean o leen cuentas:

String sql = "INSERT INTO cuentas (idpropietario, saldo, tipoCuenta, idsucursal) VALUES (?, ?, ?, ?)";
pstmt.setInt(4, c.getIdsucursal());

El método obtenerLista también incorpora idsucursal al instanciar objetos Cuenta:

lista.add(new Cuenta(
   rs.getInt("idcuenta"),
   rs.getInt("idpropietario"),
   rs.getDouble("saldo"),
   rs.getString("tipoCuenta"),

```
    rs.getInt("idsucursal")
));
```

2. Nuevos métodos para listar cuentas con sucursal

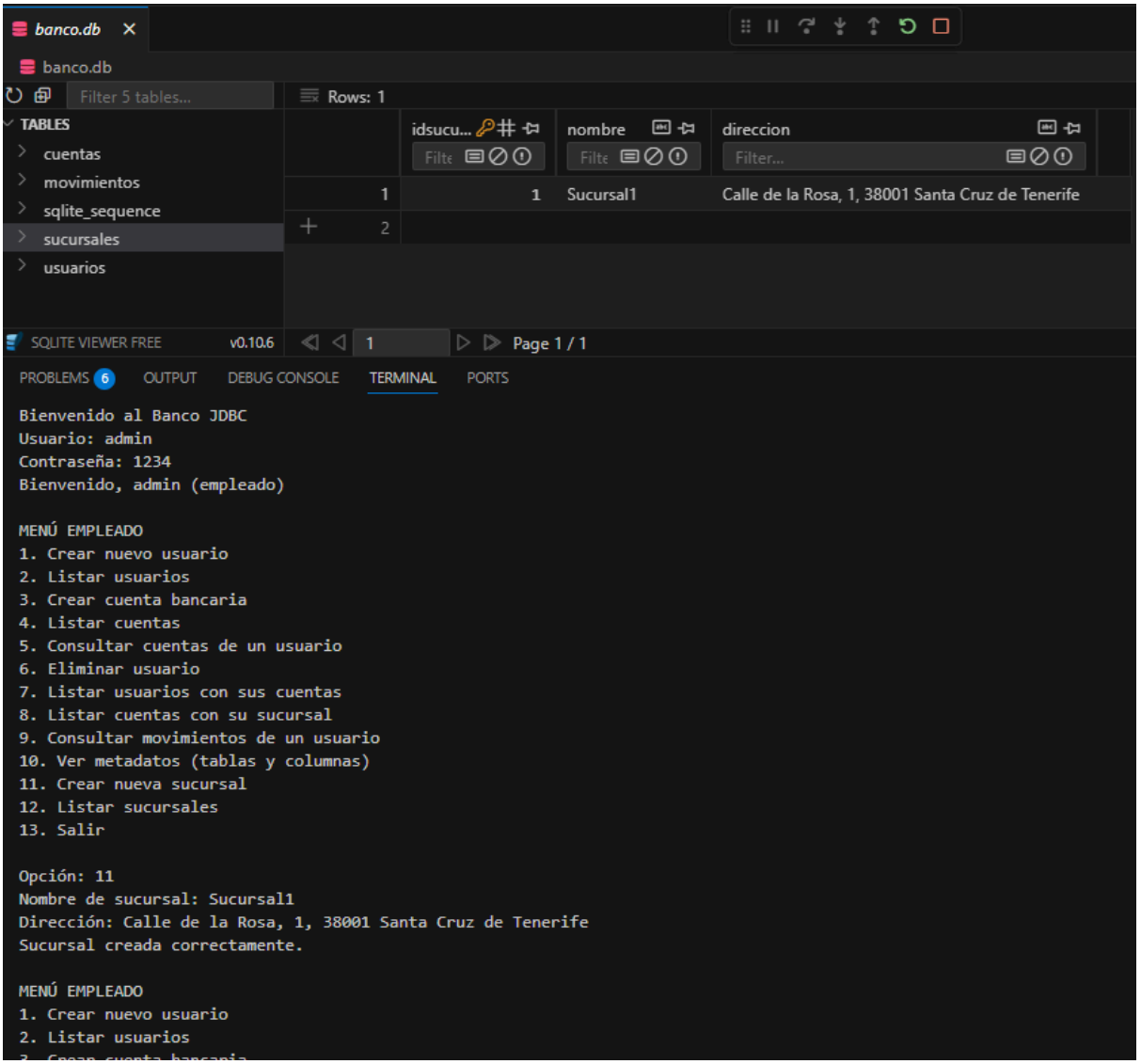
Se agregaron varios métodos que muestran cuentas junto con la información de sucursal:

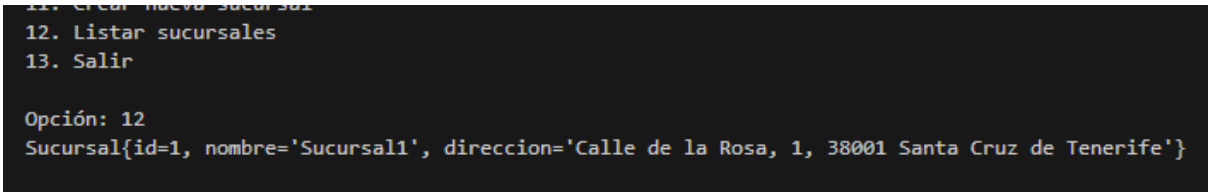
listarCuentasConSucursal() → Lista todas las cuentas mostrando la sucursal asociada (LEFT JOIN).

listarCuentasConSucursalDeUsuario(int idUsuario) → Lista solo las cuentas de un usuario con la sucursal correspondiente (LEFT JOIN con WHERE).

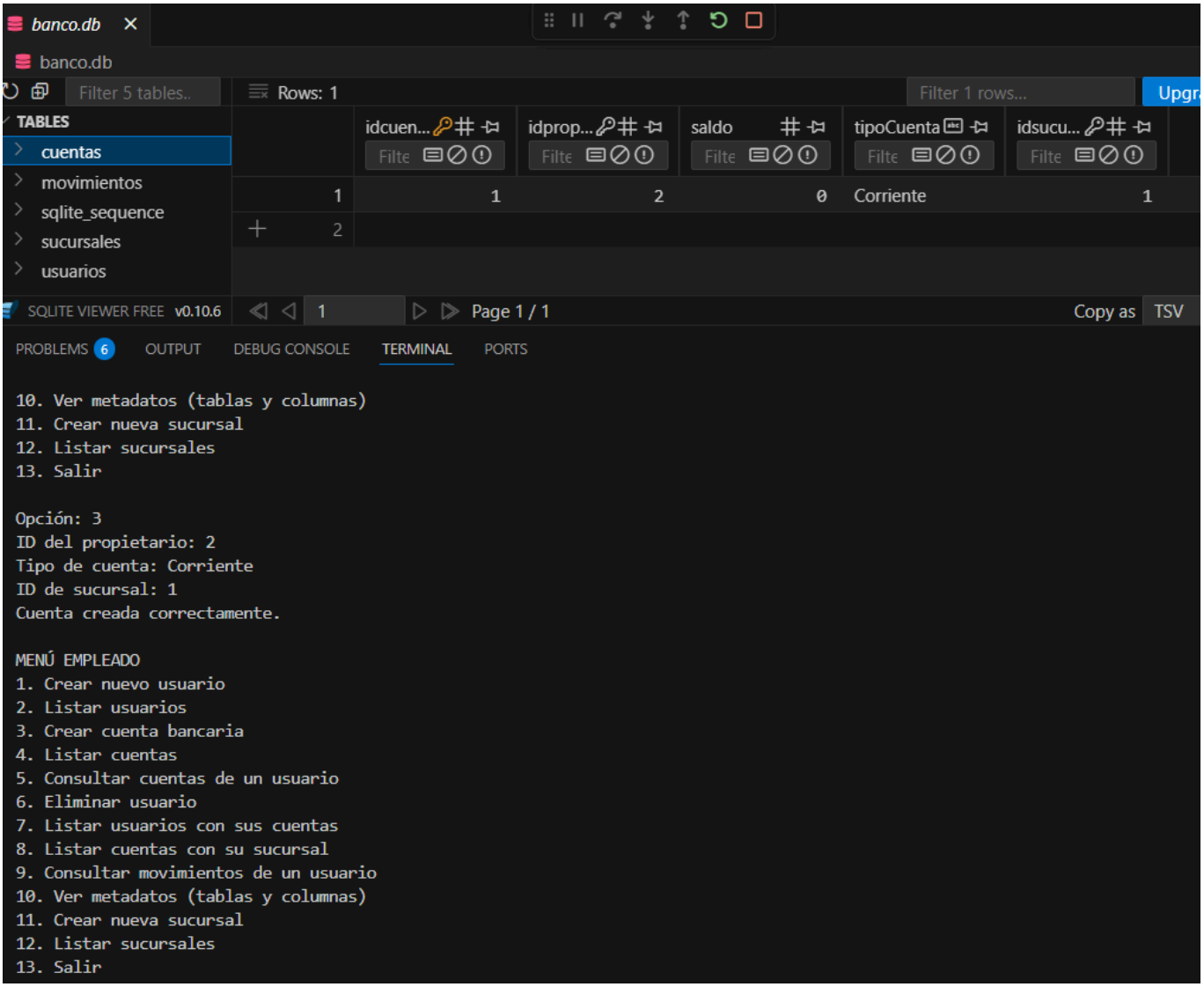
Se añadió listarUsuariosConCuentas() que hace un JOIN entre usuarios y cuentas para mostrar qué usuario posee qué cuentas.

Capturas de pantalla





```
10. Ver metadatos (tablas y columnas)
11. Crear nueva sucursal
12. Listar sucursales
13. Salir

Opción: 10

Tabla: cuentas
 - idcuenta (INTEGER)
 - idpropietario (INTEGER)
 - saldo (REAL)
 - tipoCuenta (TEXT)
 - idsucursal (INTEGER)

Tabla: movimientos
 - idmovimiento (INTEGER)
 - idcuenta (INTEGER)
 - tipo (TEXT)
 - cantidad (REAL)
 - fecha (TIMESTAMP)

Tabla: sucursales
 - idsucursal (INTEGER)
 - nombre (TEXT)
 - direccion (TEXT)

Tabla: usuarios
 - idusuario (INTEGER)
 - nombre (TEXT)
 - password (TEXT)
 - rol (TEXT)

MENÚ EMPLEADO
1. Crear nuevo usuario
```


banco.db
banco.db
Filter 5 tables..
Rows: 1
Filter 1 rows...
TABLES
cuentas
movimientos
sqlite_sequence
sucursales
usuarios
idcuen...
idprop...
saldo
tipoCuenta
idsucu...
1 1 2 0 Corriente 1
2
SQLITE VIEWER FREE v0.10.6
Page 1 / 1
Copy as TSV
PROBLEMS 6
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
10. Ver metadatos (tablas y columnas)
11. Crear nueva sucursal
12. Listar sucursales
13. Salir
Opción: 3
ID del propietario: 2
Tipo de cuenta: Corriente
ID de sucursal: 1
Cuenta creada correctamente.
MENÚ EMPLEADO
1. Crear nuevo usuario
2. Listar usuarios
3. Crear cuenta bancaria
4. Listar cuentas
5. Consultar cuentas de un usuario
6. Eliminar usuario
7. Listar usuarios con sus cuentas
8. Listar cuentas con su sucursal
9. Consultar movimientos de un usuario
10. Ver metadatos (tablas y columnas)
11. Crear nueva sucursal
12. Listar sucursales
13. Salir

```
Opción: 4
Cuenta #1 | Propietario ID: 2 | Tipo: Corriente | Saldo: 0,00? | Sucursal ID: 1
Cuenta #2 | Propietario ID: 3 | Tipo: Ahorro | Saldo: 0,00? | Sucursal ID: 1

MENÚ EMPLEADO
1. Crear nuevo usuario
2. Listar usuarios
3. Crear cuenta bancaria
4. Listar cuentas
5. Consultar cuentas de un usuario
6. Eliminar usuario
7. Listar usuarios con sus cuentas
8. Listar cuentas con su sucursal
9. Consultar movimientos de un usuario
10. Ver metadatos (tablas y columnas)
11. Crear nueva sucursal
12. Listar sucursales
13. Salir

Opción: 5
ID de usuario: 2
Cuenta #1 | Propietario ID: 2 | Tipo: Corriente | Saldo: 0,00? | Sucursal ID: 1
```

```
Opción: 7
Walid -> Cuenta 1 (0,00?, Corriente)
Marcos -> Cuenta 2 (0,00?, Ahorro)

MENÚ EMPLEADO
1. Crear nuevo usuario
2. Listar usuarios
3. Crear cuenta bancaria
4. Listar cuentas
5. Consultar cuentas de un usuario
6. Eliminar usuario
7. Listar usuarios con sus cuentas
8. Listar cuentas con su sucursal
9. Consultar movimientos de un usuario
10. Ver metadatos (tablas y columnas)
11. Crear nueva sucursal
12. Listar sucursales
13. Salir

Opción: 8
Cuenta 1 | 0,00? | Corriente | Sucursal: Sucursal1
Cuenta 2 | 0,00? | Ahorro | Sucursal: Sucursal1
```

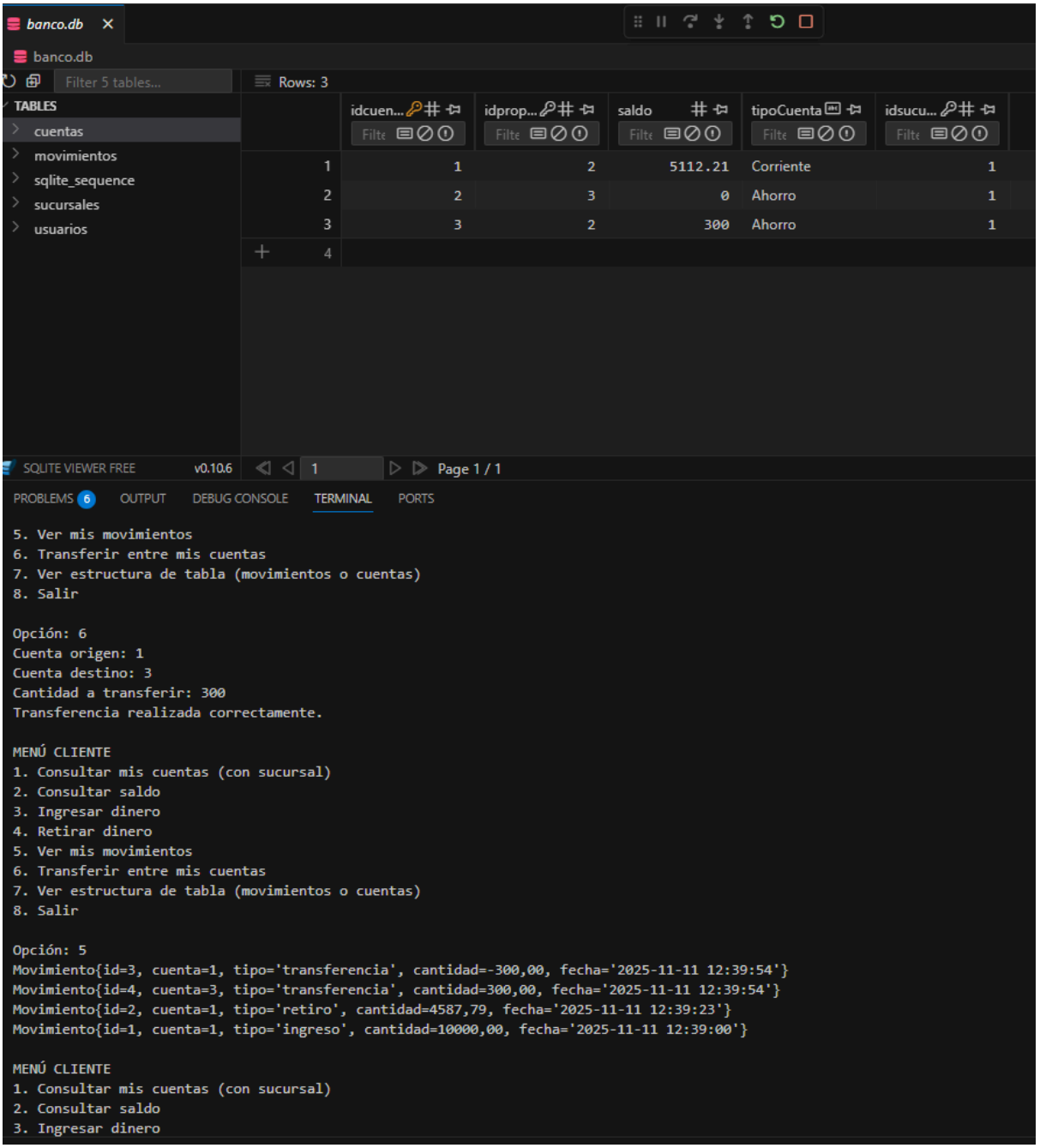
```
Bienvenido al Banco JDBC
Usuario: Walid
Contraseña: 1234
Bienvenido, Walid (cliente)

MENÚ CLIENTE
1. Consultar mis cuentas (con sucursal)
2. Consultar saldo
3. Ingresar dinero
4. Retirar dinero
5. Ver mis movimientos
6. Transferir entre mis cuentas
7. Ver estructura de tabla (movimientos o cuentas)
8. Salir

Opción: 1
Cuenta 1 | 0,00? | Corriente | Sucursal: Sucursal1

MENÚ CLIENTE
1. Consultar mis cuentas (con sucursal)
2. Consultar saldo
3. Ingresar dinero
4. Retirar dinero
5. Ver mis movimientos
6. Transferir entre mis cuentas
7. Ver estructura de tabla (movimientos o cuentas)
8. Salir

Opción: 2
ID de cuenta: 1
Saldo actual: 0,00?
```

```
Opción: 3
ID de cuenta: 1
Cantidad a ingresar: 10000
Ingreso de 10000,00? realizado correctamente.
Movimiento registrado: ingreso (10000.0?)

MENÚ CLIENTE
1. Consultar mis cuentas (con sucursal)
2. Consultar saldo
3. Ingresar dinero
4. Retirar dinero
5. Ver mis movimientos
6. Transferir entre mis cuentas
7. Ver estructura de tabla (movimientos o cuentas)
8. Salir

Opción: 4
ID de cuenta: 1
Cantidad a retirar: 4587.79
Retiro de 4587,79? realizado correctamente.
Movimiento registrado: retiro (4587.79?)

MENÚ CLIENTE
1. Consultar mis cuentas (con sucursal)
2. Consultar saldo
3. Ingresar dinero
4. Retirar dinero
5. Ver mis movimientos
6. Transferir entre mis cuentas
7. Ver estructura de tabla (movimientos o cuentas)
8. Salir

Opción: 5
Movimiento{id=2, cuenta=1, tipo='retiro', cantidad=4587,79, fecha='2025-11-11 12:19:04'}
Movimiento{id=1, cuenta=1, tipo='ingreso', cantidad=10000,00, fecha='2025-11-11 12:18:02'}
```

```
Opción: 5
Movimiento{id=2, cuenta=1, tipo='retiro', cantidad=4587,79, fecha='2025-11-11 12:19:04'}
Movimiento{id=1, cuenta=1, tipo='ingreso', cantidad=10000,00, fecha='2025-11-11 12:18:02'}

MENÚ CLIENTE
1. Consultar mis cuentas (con sucursal)
2. Consultar saldo
3. Ingresar dinero
4. Retirar dinero
5. Ver mis movimientos
6. Transferir entre mis cuentas
7. Ver estructura de tabla (movimientos o cuentas)
8. Salir

Opción: 2
ID de cuenta: 1
Saldo actual: 5412,21?
```

```
MENÚ CLIENTE
1. Consultar mis cuentas (con sucursal)
2. Consultar saldo
3. Ingresar dinero
4. Retirar dinero
5. Ver mis movimientos
6. Transferir entre mis cuentas
7. Ver estructura de tabla (movimientos o cuentas)
8. Salir

Opción: 7

Tabla: cuentas
 - idcuenta (INTEGER)
 - idpropietario (INTEGER)
 - saldo (REAL)
 - tipoCuenta (TEXT)
 - idsucursal (INTEGER)

Tabla: movimientos
 - idmovimiento (INTEGER)
 - idcuenta (INTEGER)
 - tipo (TEXT)
 - cantidad (REAL)
 - fecha (TIMESTAMP)

Tabla: sucursales
 - idsucursal (INTEGER)
 - nombre (TEXT)
 - direccion (TEXT)

Tabla: usuarios
 - idusuario (INTEGER)
 - nombre (TEXT)
 - password (TEXT)
 - rol (TEXT)
```

```
Opción: 3
ID del propietario: 2
Tipo de cuenta: Ahorro
ID de sucursal: 1
Cuenta creada correctamente.

MENÚ EMPLEADO
1. Crear nuevo usuario
2. Listar usuarios
3. Crear cuenta bancaria
4. Listar cuentas
5. Consultar cuentas de un usuario
6. Eliminar usuario
7. Listar usuarios con sus cuentas
8. Listar cuentas con su sucursal
9. Consultar movimientos de un usuario
10. Ver metadatos (tablas y columnas)
11. Crear nueva sucursal
12. Listar sucursales
13. Salir

Opción:
```


banco.db
banco.db
Filter 5 tables...
TABLES
cuentas
movimientos
sqlite_sequence
sucursales
usuarios
Rows: 3
idcuen...
idprop...
saldo
tipoCuenta
idsucu...
1 1 2 5112.21 Corriente 1
2 2 3 0 Ahorro 1
3 3 2 300 Ahorro 1
4
SQLITE VIEWER FREE v0.10.6
Page 1 / 1
PROBLEMS 6
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
5. Ver mis movimientos
6. Transferir entre mis cuentas
7. Ver estructura de tabla (movimientos o cuentas)
8. Salir
Opción: 6
Cuenta origen: 1
Cuenta destino: 3
Cantidad a transferir: 300
Transferencia realizada correctamente.
MENÚ CLIENTE
1. Consultar mis cuentas (con sucursal)
2. Consultar saldo
3. Ingresar dinero
4. Retirar dinero
5. Ver mis movimientos
6. Transferir entre mis cuentas
7. Ver estructura de tabla (movimientos o cuentas)
8. Salir
Opción: 5
Movimiento{id=3, cuenta=1, tipo='transferencia', cantidad=-300,00, fecha='2025-11-11 12:39:54'}
Movimiento{id=4, cuenta=3, tipo='transferencia', cantidad=300,00, fecha='2025-11-11 12:39:54'}
Movimiento{id=2, cuenta=1, tipo='retiro', cantidad=4587,79, fecha='2025-11-11 12:39:23'}
Movimiento{id=1, cuenta=1, tipo='ingreso', cantidad=10000,00, fecha='2025-11-11 12:39:00'}
MENÚ CLIENTE
1. Consultar mis cuentas (con sucursal)
2. Consultar saldo
3. Ingresar dinero

```
Opción: 1
Cuenta 1 | 5112,21? | Corriente | Sucursal: Sucursal1
Cuenta 3 | 300,00? | Ahorro | Sucursal: Sucursal1

MENÚ CLIENTE
1. Consultar mis cuentas (con sucursal)
2. Consultar saldo
3. Ingresar dinero
4. Retirar dinero
5. Ver mis movimientos
6. Transferir entre mis cuentas
7. Ver estructura de tabla (movimientos o cuentas)
8. Salir

Opción: 2
ID de cuenta: 1
Saldo actual: 5112,21?
```

```
Bienvenido al Banco JDBC
Usuario: admin
Contraseña: 1234
Bienvenido, admin (empleado)

MENÚ EMPLEADO
1. Crear nuevo usuario
2. Listar usuarios
3. Crear cuenta bancaria
4. Listar cuentas
5. Consultar cuentas de un usuario
6. Eliminar usuario
7. Listar usuarios con sus cuentas
8. Listar cuentas con su sucursal
9. Consultar movimientos de un usuario
10. Ver metadatos (tablas y columnas)
11. Crear nueva sucursal
12. Listar sucursales
13. Salir

Opción: 9
ID de usuario: 2
Movimiento{id=3, cuenta=1, tipo='transferencia', cantidad=-300,00, fecha='2025-11-11 12:39:54'}
Movimiento{id=4, cuenta=3, tipo='transferencia', cantidad=300,00, fecha='2025-11-11 12:39:54'}
Movimiento{id=2, cuenta=1, tipo='retiro', cantidad=4587,79, fecha='2025-11-11 12:39:23'}
Movimiento{id=1, cuenta=1, tipo='ingreso', cantidad=10000,00, fecha='2025-11-11 12:39:00'}
```

Código completo

Main.java:

```
import java.sql.*;
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        UsuarioDAO usuarioDAO = new UsuarioDAO();
        CuentaDAO cuentaDAO = new CuentaDAO();
        MovimientoDAO movimientoDAO = new MovimientoDAO();
        SucursalDAO sucursalDAO = new SucursalDAO();

        System.out.println("Bienvenido al Banco JDBC");
        System.out.print("Usuario: ");
        String nombre = sc.nextLine();
        System.out.print("Contraseña: ");
        String pass = sc.nextLine();

        Usuario user = usuarioDAO.autenticar(nombre, pass);

        if (user == null) {
            System.out.println("Credenciales incorrectas.");
            return;
        }

        System.out.println("Bienvenido, " + user.getNombre() + " (" +
user.getRol() + ")");

        if (user.getRol().equalsIgnoreCase("empleado")) {
            menuEmpleado(sc, usuarioDAO, cuentaDAO, movimientoDAO,
sucursalDAO);
        } else {
            menuCliente(sc, cuentaDAO, movimientoDAO, user);
        }
    }

    // MENÚ EMPLEADO
    private static void menuEmpleado(Scanner sc, UsuarioDAO usuarioDAO,
                                     CuentaDAO cuentaDAO, MovimientoDAO
movDAO,
```

```
                                    SucursalDAO sucDAO) {
        int opcion;
        do {
            System.out.println("""
            \nMENÚ EMPLEADO
            1. Crear nuevo usuario
            2. Listar usuarios
            3. Crear cuenta bancaria
            4. Listar cuentas
            5. Consultar cuentas de un usuario
            6. Eliminar usuario
            7. Listar usuarios con sus cuentas
            8. Listar cuentas con su sucursal
            9. Consultar movimientos de un usuario
            10. Ver metadatos (tablas y columnas)
            11. Crear nueva sucursal
            12. Listar sucursales
            13. Salir
            """);
            System.out.print("Opción: ");
            opcion = Integer.parseInt(sc.nextLine());

            switch (opcion) {
                case 1 -> {
                    System.out.print("Nombre: ");
                    String nombre = sc.nextLine();
                    System.out.print("Contraseña: ");
                    String pass = sc.nextLine();
                    System.out.print("Rol (empleado/cliente): ");
                    String rol = sc.nextLine();
                    usuarioDAO.crearUsuario(new Usuario(nombre, pass,
rol));
                }
                case 2 ->
usuarioDAO.listarUsuarios().forEach(System.out::println);
                case 3 -> {
                    System.out.print("ID del propietario: ");
                    int id = Integer.parseInt(sc.nextLine());
                    System.out.print("Tipo de cuenta: ");
                    String tipo = sc.nextLine();
                    System.out.print("ID de sucursal: ");
                    int idsuc = Integer.parseInt(sc.nextLine());
```

```
                    cuentaDAO.crearCuenta(new Cuenta(id, 0.0, tipo,
idsuc));
                }
                case 4 ->
cuentaDAO.listarCuentas().forEach(System.out::println);
                case 5 -> {
                    System.out.print("ID de usuario: ");
                    int id = Integer.parseInt(sc.nextLine());

cuentaDAO.obtenerCuentasUsuario(id).forEach(System.out::println);
                }
                case 6 -> {
                    System.out.print("ID de usuario a eliminar: ");
                    int id = Integer.parseInt(sc.nextLine());
                    usuarioDAO.eliminarUsuario(id);
                }
                case 7 -> cuentaDAO.listarUsuariosConCuentas();
                case 8 -> cuentaDAO.listarCuentasConSucursal();
                case 9 -> {
                    System.out.print("ID de usuario: ");
                    int id = Integer.parseInt(sc.nextLine());

movDAO.movimientosDeUsuario(id).forEach(System.out::println);
                }
                case 10 -> mostrarMetadatos();
                case 11 -> {
                    System.out.print("Nombre de sucursal: ");
                    String n = sc.nextLine();
                    System.out.print("Dirección: ");
                    String d = sc.nextLine();
                    sucDAO.crearSucursal(new Sucursal(n, d));
                }
                case 12 ->
sucDAO.listarSucursales().forEach(System.out::println);
                case 13 -> System.out.println("Saliendo del menú
empleado...");
                default -> System.out.println("Opción no válida.");
            }
        } while (opcion != 13);
    }

    // MENÚ CLIENTE
```

```
    private static void menuCliente(Scanner sc, CuentaDAO cuentaDAO,
MovimientoDAO movDAO, Usuario user) {
        int opcion;
        do {
            System.out.println("""
            \nMENÚ CLIENTE
            1. Consultar mis cuentas (con sucursal)
            2. Consultar saldo
            3. Ingresar dinero
            4. Retirar dinero
            5. Ver mis movimientos
            6. Transferir entre mis cuentas
            7. Ver estructura de tabla (movimientos o cuentas)
            8. Salir
            """);
            System.out.print("Opción: ");
            opcion = Integer.parseInt(sc.nextLine());

            switch (opcion) {
                case 1 ->
cuentaDAO.listarCuentasConSucursalDeUsuario(user.getIdusuario());
                case 2 -> {
                    System.out.print("ID de cuenta: ");
                    int id = Integer.parseInt(sc.nextLine());
                    double saldo = cuentaDAO.consultarSaldo(id);
                    System.out.printf("Saldo actual: %.2f€%n", saldo);
                }
                case 3 -> {
                    System.out.print("ID de cuenta: ");
                    int id = Integer.parseInt(sc.nextLine());
                    System.out.print("Cantidad a ingresar: ");
                    double cant = Double.parseDouble(sc.nextLine());
                    cuentaDAO.ingresar(id, cant);
                    movDAO.registrarMovimiento(new Movimiento(id,
"ingreso", cant));
                }
                case 4 -> {
                    System.out.print("ID de cuenta: ");
                    int id = Integer.parseInt(sc.nextLine());
                    System.out.print("Cantidad a retirar: ");
                    double cant = Double.parseDouble(sc.nextLine());
                    cuentaDAO.retirar(id, cant);
```

```
                    movDAO.registrarMovimiento(new Movimiento(id,
"retiro", cant));
                }
                case 5 ->
movDAO.movimientosDeUsuario(user.getIdusuario()).forEach(System.out::pr
intln);
                case 6 -> transferir(sc, user);
                case 7 -> mostrarMetadatos();
                case 8 -> System.out.println("Saliendo del menú
cliente...");
                default -> System.out.println("Opción no válida.");
            }
        } while (opcion != 8);
    }

    // Transferencia con transacción
    private static void transferir(Scanner sc, Usuario user) {
        System.out.print("Cuenta origen: ");
        int origen = Integer.parseInt(sc.nextLine());
        System.out.print("Cuenta destino: ");
        int destino = Integer.parseInt(sc.nextLine());
        System.out.print("Cantidad a transferir: ");
        double cantidad = Double.parseDouble(sc.nextLine());

        String url = "jdbc:sqlite:banco.db";
        try (Connection conn = DriverManager.getConnection(url)) {
            conn.setAutoCommit(false);

            PreparedStatement retirar = conn.prepareStatement("UPDATE
cuentas SET saldo = saldo - ? WHERE idcuenta = ?");
            PreparedStatement ingresar = conn.prepareStatement("UPDATE
cuentas SET saldo = saldo + ? WHERE idcuenta = ?");
            PreparedStatement mov = conn.prepareStatement("INSERT INTO
movimientos (idcuenta, tipo, cantidad) VALUES (?, ?, ?)");

            retirar.setDouble(1, cantidad);
            retirar.setInt(2, origen);
            ingresar.setDouble(1, cantidad);
            ingresar.setInt(2, destino);

            retirar.executeUpdate();
            ingresar.executeUpdate();
```

```
            mov.setInt(1, origen);
            mov.setString(2, "transferencia");
            mov.setDouble(3, -cantidad);
            mov.executeUpdate();

            mov.setInt(1, destino);
            mov.setString(2, "transferencia");
            mov.setDouble(3, cantidad);
            mov.executeUpdate();

            conn.commit();
            System.out.println("Transferencia realizada
correctamente.");
        } catch (Exception e) {
            System.out.println("Error en la transferencia: " +
e.getMessage());
        }
    }

    // Metadatos (tablas y columnas)
    private static void mostrarMetadatos() {
        String url = "jdbc:sqlite:banco.db";
        try (Connection conn = DriverManager.getConnection(url)) {
            DatabaseMetaData meta = conn.getMetaData();
            ResultSet tablas = meta.getTables(null, null, "%", new
String[]{"TABLE"});
            while (tablas.next()) {
                String nombreTabla = tablas.getString("TABLE_NAME");
                System.out.println("\nTabla: " + nombreTabla);
                ResultSet columnas = meta.getColumns(null, null,
nombreTabla, "%");
                while (columnas.next()) {
                    System.out.printf(" - %s (%s)%n",
                        columnas.getString("COLUMN_NAME"),
                        columnas.getString("TYPE_NAME"));
                }
            }
        } catch (SQLException e) {
            System.out.println("Error leyendo metadatos: " +
e.getMessage());
        }
    }
}
```

Usuario.java:

```
public class Usuario {
    private int idusuario;
    private String nombre;
    private String password;
    private String rol; // "empleado" o "cliente"

    public Usuario() {}

    public Usuario(String nombre, String password, String rol) {
        this.nombre = nombre;
        this.password = password;
        this.rol = rol;
    }

    public Usuario(int idusuario, String nombre, String password,
String rol) {
        this(nombre, password, rol);
        this.idusuario = idusuario;
    }

    public int getIdusuario() { return idusuario; }
    public void setIdusuario(int idusuario) { this.idusuario =
idusuario; }

    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }

    public String getPassword() { return password; }
    public void setPassword(String password) { this.password =
password; }

    public String getRol() { return rol; }
    public void setRol(String rol) { this.rol = rol; }

    @Override
    public String toString() {
        return String.format("[%d] %s (%s)", idusuario, nombre, rol);
    }
```

```
}
```

UsuarioDAO.java:

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class UsuarioDAO {
    private final String url = "jdbc:sqlite:banco.db";

    public UsuarioDAO() {
        crearTablas();
        crearAdminPorDefecto();
    }

    private void crearTablas() {
        String sqlUsuarios = """
            CREATE TABLE IF NOT EXISTS usuarios (
                idusuario INTEGER PRIMARY KEY AUTOINCREMENT,
                nombre TEXT NOT NULL UNIQUE,
                password TEXT NOT NULL,
                rol TEXT CHECK(rol IN ('empleado','cliente')) NOT NULL
            );
        """;

        String sqlSucursales = """
            CREATE TABLE IF NOT EXISTS sucursales (
                idsucursal INTEGER PRIMARY KEY AUTOINCREMENT,
                nombre TEXT NOT NULL,
                direccion TEXT
            );
        """;

        String sqlCuentas = """
            CREATE TABLE IF NOT EXISTS cuentas (
                idcuenta INTEGER PRIMARY KEY AUTOINCREMENT,
                idpropietario INTEGER NOT NULL,
                saldo REAL DEFAULT 0,
                tipoCuenta TEXT,
                idsucursal INTEGER,
```

```
                FOREIGN KEY (idpropietario) REFERENCES
usuarios(idusuario) ON DELETE CASCADE,
                FOREIGN KEY (idsucursal) REFERENCES
sucursales(idsucursal)
            );
        """;

        String sqlMovimientos = """
            CREATE TABLE IF NOT EXISTS movimientos (
                idmovimiento INTEGER PRIMARY KEY AUTOINCREMENT,
                idcuenta INTEGER,
                tipo TEXT,
                cantidad REAL,
                fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                FOREIGN KEY(idcuenta) REFERENCES cuentas(idcuenta)
            );
        """;

        try (Connection conn = DriverManager.getConnection(url);
            Statement stmt = conn.createStatement()) {
            stmt.execute(sqlUsuarios);
            stmt.execute(sqlSucursales);
            stmt.execute(sqlCuentas);
            stmt.execute(sqlMovimientos);
        } catch (SQLException e) {
            System.out.println("Error creando tablas: " +
e.getMessage());
        }
    }


    private void crearAdminPorDefecto() {
        String check = "SELECT COUNT(*) FROM usuarios WHERE nombre =
'admin'";
        try (Connection conn = DriverManager.getConnection(url);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(check)) {
            if (rs.next() && rs.getInt(1) == 0) {
                String insert = "INSERT INTO usuarios (nombre,
password, rol) VALUES ('admin', '1234', 'empleado')";
                stmt.executeUpdate(insert);
                System.out.println("Usuario administrador creado
automáticamente (admin / 1234).");
```

```
            }
        } catch (SQLException e) {
            System.out.println("Error creando admin: " +
e.getMessage());
        }
    }

    public void crearUsuario(Usuario u) {
        String sql = "INSERT INTO usuarios (nombre, password, rol)
VALUES (?, ?, ?)";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, u.getNombre());
            pstmt.setString(2, u.getPassword());
            pstmt.setString(3, u.getRol());
            pstmt.executeUpdate();
            System.out.println("Usuario creado correctamente.");
        } catch (SQLException e) {
            System.out.println("Error creando usuario: " +
e.getMessage());
        }
    }

    public List<Usuario> listarUsuarios() {
        List<Usuario> lista = new ArrayList<>();
        String sql = "SELECT * FROM usuarios";
        try (Connection conn = DriverManager.getConnection(url);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {
            while (rs.next()) {
                lista.add(new Usuario(
                    rs.getInt("idusuario"),
                    rs.getString("nombre"),
                    rs.getString("password"),
                    rs.getString("rol")
                ));
            }
        } catch (SQLException e) {
            System.out.println("Error listando usuarios: " +
e.getMessage());
        }
        return lista;
    }
```

```
    public Usuario autenticar(String nombre, String password) {
        String sql = "SELECT * FROM usuarios WHERE nombre = ? AND
password = ?";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, nombre);
            pstmt.setString(2, password);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                return new Usuario(
                    rs.getInt("idusuario"),
                    rs.getString("nombre"),
                    rs.getString("password"),
                    rs.getString("rol")
                );
            }
        } catch (SQLException e) {
            System.out.println("Error autenticando: " +
e.getMessage());
        }
        return null;
    }

    public void eliminarUsuario(int idusuario) {
        String sql1 = "DELETE FROM cuentas WHERE idpropietario = ?";
        String sql2 = "DELETE FROM usuarios WHERE idusuario = ?";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement pstmt1 = conn.prepareStatement(sql1);
             PreparedStatement pstmt2 = conn.prepareStatement(sql2)) {
            pstmt1.setInt(1, idusuario);
            pstmt1.executeUpdate();
            pstmt2.setInt(1, idusuario);
            pstmt2.executeUpdate();
            System.out.println("Usuario eliminado (y sus cuentas
asociadas).");
        } catch (SQLException e) {
            System.out.println("Error eliminando usuario: " +
e.getMessage());
        }
    }
}
```

Cuenta.java:

```
public class Cuenta {
    private int idcuenta;
    private int idpropietario;
    private double saldo;
    private String tipoCuenta;
    private int idsucursal;

    public Cuenta() {}

    public Cuenta(int idpropietario, double saldo, String tipoCuenta,
int idsucursal) {
        this.idpropietario = idpropietario;
        this.saldo = saldo;
        this.tipoCuenta = tipoCuenta;
        this.idsucursal = idsucursal;
    }

    // Sobrecarga sin sucursal (por compatibilidad con versiones
previas)
    public Cuenta(int idpropietario, double saldo, String tipoCuenta) {
        this(idpropietario, saldo, tipoCuenta, 0);
    }

    public Cuenta(int idcuenta, int idpropietario, double saldo, String
tipoCuenta, int idsucursal) {
        this(idpropietario, saldo, tipoCuenta, idsucursal);
        this.idcuenta = idcuenta;
    }

    public int getIdcuenta() { return idcuenta; }
    public void setIdcuenta(int idcuenta) { this.idcuenta = idcuenta; }

    public int getIdpropietario() { return idpropietario; }
    public void setIdpropietario(int idpropietario) {
this.idpropietario = idpropietario; }

    public double getSaldo() { return saldo; }
    public void setSaldo(double saldo) { this.saldo = saldo; }
```

```
    public String getTipoCuenta() { return tipoCuenta; }
    public void setTipoCuenta(String tipoCuenta) { this.tipoCuenta =
tipoCuenta; }

    public int getIdsucursal() { return idsucursal; }
    public void setIdsucursal(int idsucursal) { this.idsucursal =
idsucursal; }

    @Override
    public String toString() {
        return String.format("Cuenta #%d | Propietario ID: %d | Tipo:
%s | Saldo: %.2f€ | Sucursal ID: %d",
                idcuenta, idpropietario, tipoCuenta, saldo,
idsucursal);
    }
}
```

CuentaDAO.java:

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class CuentaDAO {
    private final String url = "jdbc:sqlite:banco.db";

    public void crearCuenta(Cuenta c) {
        String sql = "INSERT INTO cuentas (idpropietario, saldo,
tipoCuenta, idsucursal) VALUES (?, ?, ?, ?)";
        try (Connection conn = DriverManager.getConnection(url);
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, c.getIdpropietario());
            pstmt.setDouble(2, c.getSaldo());
            pstmt.setString(3, c.getTipoCuenta());
            pstmt.setInt(4, c.getIdsucursal());
            pstmt.executeUpdate();
            System.out.println("Cuenta creada correctamente.");
        } catch (SQLException e) {
            System.out.println("Error creando cuenta: " +
e.getMessage());
        }
```

```
    }

    public List<Cuenta> listarCuentas() {
        return obtenerLista("SELECT * FROM cuentas");
    }

    public List<Cuenta> obtenerCuentasUsuario(int idUsuario) {
        return obtenerLista("SELECT * FROM cuentas WHERE idpropietario
= " + idUsuario);
    }

    public double consultarSaldo(int idcuenta) {
        String sql = "SELECT saldo FROM cuentas WHERE idcuenta = ?";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, idcuenta);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) return rs.getDouble("saldo");
        } catch (SQLException e) {
            System.out.println("Error consultando saldo: " +
e.getMessage());
        }
        return -1;
    }

    public void ingresar(int idcuenta, double cantidad) {
        actualizarSaldo(idcuenta, cantidad, true);
    }

    public void retirar(int idcuenta, double cantidad) {
        actualizarSaldo(idcuenta, cantidad, false);
    }

    private void actualizarSaldo(int idcuenta, double cantidad, boolean
ingresar) {
        String operacion = ingresar ? "Ingreso" : "Retiro";
        double signo = ingresar ? cantidad : -cantidad;

        String sql = "UPDATE cuentas SET saldo = saldo + ? WHERE
idcuenta = ?";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setDouble(1, signo);
```

```
            pstmt.setInt(2, idcuenta);
            pstmt.executeUpdate();
            System.out.printf("%s de %.2f€ realizado correctamente.%n",
operacion, cantidad);
        } catch (SQLException e) {
            System.out.println("Error en " + operacion.toLowerCase() +
": " + e.getMessage());
        }
    }

    private List<Cuenta> obtenerLista(String sql) {
        List<Cuenta> lista = new ArrayList<>();
        try (Connection conn = DriverManager.getConnection(url);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {
            while (rs.next()) {
                lista.add(new Cuenta(
                    rs.getInt("idcuenta"),
                    rs.getInt("idpropietario"),
                    rs.getDouble("saldo"),
                    rs.getString("tipoCuenta"),
                    rs.getInt("idsucursal")
                ));
            }
        } catch (SQLException e) {
            System.out.println("Error obteniendo cuentas: " +
e.getMessage());
        }
        return lista;
    }

    public void listarUsuariosConCuentas() {
        String sql = """
            SELECT u.nombre, c.idcuenta, c.saldo, c.tipoCuenta
            FROM usuarios u
            JOIN cuentas c ON u.idusuario = c.idpropietario
        """;
        try (Connection conn = DriverManager.getConnection(url);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql)) {
            while (rs.next()) {
                System.out.printf("%s → Cuenta %d (%.2f€, %s)%n",
                    rs.getString("nombre"), rs.getInt("idcuenta"),
```

```
                    rs.getDouble("saldo"), rs.getString("tipoCuenta"));
            }
        } catch (SQLException e) {
            System.out.println("Error listando usuarios con cuentas: "
+ e.getMessage());
        }
    }

    public void listarCuentasConSucursal() {
        String sql = """
            SELECT c.idcuenta, c.saldo, c.tipoCuenta, s.nombre AS
sucursal
            FROM cuentas c
            LEFT JOIN sucursales s ON c.idsucursal = s.idsucursal
        """;
        try (Connection conn = DriverManager.getConnection(url);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql)) {
            while (rs.next()) {
                System.out.printf("Cuenta %d | %.2f€ | %s | Sucursal:
%s%n",
                    rs.getInt("idcuenta"),
                    rs.getDouble("saldo"),
                    rs.getString("tipoCuenta"),
                    rs.getString("sucursal"));
            }
        } catch (SQLException e) {
            System.out.println("Error listando cuentas con sucursal: "
+ e.getMessage());
        }
    }

    public void listarCuentasConSucursalDeUsuario(int idUsuario) {
        String sql = """
            SELECT c.idcuenta, c.saldo, c.tipoCuenta, s.nombre AS
sucursal
            FROM cuentas c
            LEFT JOIN sucursales s ON c.idsucursal = s.idsucursal
            WHERE c.idpropietario = ?
        """;
        try (Connection conn = DriverManager.getConnection(url);
            PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, idUsuario);
```

```
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                System.out.printf("Cuenta %d | %.2f€ | %s | Sucursal:
%s%n",
                    rs.getInt("idcuenta"),
                    rs.getDouble("saldo"),
                    rs.getString("tipoCuenta"),
                    rs.getString("sucursal"));
            }
        } catch (SQLException e) {
            System.out.println("Error listando cuentas del usuario: " +
e.getMessage());
        }
    }
}
```

Sucursal.java:

```
public class Sucursal {
    private int idsucursal;
    private String nombre;
    private String direccion;

    public Sucursal(String nombre, String direccion) {
        this.nombre = nombre;
        this.direccion = direccion;
    }

    public Sucursal(int idsucursal, String nombre, String direccion) {
        this.idsucursal = idsucursal;
        this.nombre = nombre;
        this.direccion = direccion;
    }

    public int getIdsucursal() { return idsucursal; }
    public String getNombre() { return nombre; }
    public String getDireccion() { return direccion; }

    @Override
    public String toString() {
```

```
        return String.format("Sucursal{id=%d, nombre='%s',
direccion='%s'}",
                idsucursal, nombre, direccion);
    }
}
```

SucursalDAO.java:

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class SucursalDAO {
    private final String url = "jdbc:sqlite:banco.db";

    // Crear tabla si no existe
    public SucursalDAO() {
        try (Connection conn = DriverManager.getConnection(url);
             Statement st = conn.createStatement()) {
            st.execute("""
                CREATE TABLE IF NOT EXISTS sucursales (
                    idsucursal INTEGER PRIMARY KEY AUTOINCREMENT,
                    nombre TEXT NOT NULL,
                    direccion TEXT
                )
            """);
        } catch (SQLException e) {
            System.out.println("Error creando tabla sucursales: " +
e.getMessage());
        }
    }

    // Crear sucursal
    public void crearSucursal(Sucursal s) {
        String sql = "INSERT INTO sucursales (nombre, direccion) VALUES
(?, ?)";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setString(1, s.getNombre());
            ps.setString(2, s.getDireccion());
            ps.executeUpdate();
```

```
            System.out.println("Sucursal creada correctamente.");
        } catch (SQLException e) {
            System.out.println("Error creando sucursal: " +
e.getMessage());
        }
    }

    // Listar sucursales
    public List<Sucursal> listarSucursales() {
        List<Sucursal> lista = new ArrayList<>();
        String sql = "SELECT * FROM sucursales";
        try (Connection conn = DriverManager.getConnection(url);
             Statement st = conn.createStatement();
             ResultSet rs = st.executeQuery(sql)) {
            while (rs.next()) {
                lista.add(new Sucursal(
                        rs.getInt("idsucursal"),
                        rs.getString("nombre"),
                        rs.getString("direccion")
                ));
            }
        } catch (SQLException e) {
            System.out.println("Error listando sucursales: " +
e.getMessage());
        }
        return lista;
    }
}
```

Movimiento.java:

```
public class Movimiento {
    private int idmovimiento;
    private int idcuenta;
    private String tipo;
    private double cantidad;
    private String fecha;

    public Movimiento(int idcuenta, String tipo, double cantidad) {
        this.idcuenta = idcuenta;
        this.tipo = tipo;
```

```
        this.cantidad = cantidad;
    }

    public Movimiento(int idmovimiento, int idcuenta, String tipo,
double cantidad, String fecha) {
        this.idmovimiento = idmovimiento;
        this.idcuenta = idcuenta;
        this.tipo = tipo;
        this.cantidad = cantidad;
        this.fecha = fecha;
    }

    public int getIdmovimiento() { return idmovimiento; }
    public int getIdcuenta() { return idcuenta; }
    public String getTipo() { return tipo; }
    public double getCantidad() { return cantidad; }
    public String getFecha() { return fecha; }

    @Override
    public String toString() {
        return String.format("Movimiento{id=%d, cuenta=%d, tipo='%s',
cantidad=%.2f, fecha='%s'}",
                idmovimiento, idcuenta, tipo, cantidad, fecha);
    }
}
```

MovimientoDAO.java:

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class MovimientoDAO {
    private final String url = "jdbc:sqlite:banco.db";

    // Crear tabla si no existe
    public MovimientoDAO() {
        try (Connection conn = DriverManager.getConnection(url);
             Statement st = conn.createStatement()) {
            st.execute("""
                CREATE TABLE IF NOT EXISTS movimientos (
```

```
                    idmovimiento INTEGER PRIMARY KEY AUTOINCREMENT,
                    idcuenta INTEGER,
                    tipo TEXT,
                    cantidad REAL,
                    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                    FOREIGN KEY(idcuenta) REFERENCES cuentas(idcuenta)
                )
            """);
        } catch (SQLException e) {
            System.out.println("Error creando tabla movimientos: " +
e.getMessage());
        }
    }

    // Registrar movimiento
    public void registrarMovimiento(Movimiento m) {
        String sql = "INSERT INTO movimientos (idcuenta, tipo,
cantidad) VALUES (?, ?, ?)";
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, m.getIdcuenta());
            ps.setString(2, m.getTipo());
            ps.setDouble(3, m.getCantidad());
            ps.executeUpdate();
            System.out.println("Movimiento registrado: " + m.getTipo()
+ " (" + m.getCantidad() + "€)");
        } catch (SQLException e) {
            System.out.println("Error registrando movimiento: " +
e.getMessage());
        }
    }

    // Listar todos los movimientos
    public List<Movimiento> listarMovimientos() {
        List<Movimiento> lista = new ArrayList<>();
        String sql = "SELECT * FROM movimientos ORDER BY fecha DESC";
        try (Connection conn = DriverManager.getConnection(url);
             Statement st = conn.createStatement();
             ResultSet rs = st.executeQuery(sql)) {
            while (rs.next()) {
                lista.add(new Movimiento(
                        rs.getInt("idmovimiento"),
                        rs.getInt("idcuenta"),
```

```
                        rs.getString("tipo"),
                        rs.getDouble("cantidad"),
                        rs.getString("fecha")
                ));
            }
        } catch (SQLException e) {
            System.out.println("Error listando movimientos: " +
e.getMessage());
        }
        return lista;
    }

    // Listar movimientos de un usuario (JOIN)
    public List<Movimiento> movimientosDeUsuario(int idUsuario) {
        List<Movimiento> lista = new ArrayList<>();
        String sql = """
            SELECT m.idmovimiento, m.idcuenta, m.tipo, m.cantidad,
m.fecha
            FROM movimientos m
            JOIN cuentas c ON m.idcuenta = c.idcuenta
            WHERE c.idpropietario = ?
            ORDER BY m.fecha DESC
        """;
        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, idUsuario);
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                lista.add(new Movimiento(
                        rs.getInt("idmovimiento"),
                        rs.getInt("idcuenta"),
                        rs.getString("tipo"),
                        rs.getDouble("cantidad"),
                        rs.getString("fecha")
                ));
            }
        } catch (SQLException e) {
            System.out.println("Error obteniendo movimientos del
usuario: " + e.getMessage());
        }
        return lista;
    }
}
```