

Banco FIUBA

Trabajo Práctico de Base de Datos (TA044)

Integrantes:

Nombre	Padrón	DNI
Ariel Leandro Aguirre	100199	38047275
Alejandro Paff	103376	40892509
Tomas Francisco Gonzalez	109717	43892616
Daniel Mamani	109932	44365376
Diego Adrián González Cina	110148	43170166

Fecha de entrega

25 de junio del 2025

Índices

Introducción.....	2
Contexto del problema.....	2
Relevamiento de requerimientos.....	2
Descripción del problema o sistema a modelar.....	2
Requisitos funcionales y no funcionales.....	2
Modelo conceptual.....	3
Diagrama entidad-relación (DER).....	3
Explicación de entidades, relaciones, atributos y claves.....	4
Modelo lógico.....	5
Transformación del modelo ER al modelo relacional.....	5
Normalización (1NF, 2NF y 3NF).....	5
Restricciones de integridad referencial.....	6
Reglas de negocio a nivel de Base de Datos.....	6
Diccionario de datos (nombre de tablas, columnas, tipos de datos).....	6
Tabla: Usuario.....	6
Tabla: Cuenta.....	7
Implementación.....	7
El siguiente script muestra cómo se implementan las tablas en SQL:.....	8
Consultas representativas (SELECTs con joins, subconsultas y agregaciones).....	9
Casos de uso / Consultas avanzadas.....	9
Ejemplos que muestran cómo se responde a los requerimientos con SQL.....	9
Triggers, procedimientos almacenados, vistas, índices.....	10
Interfaz de usuario.....	12
Recorrido de la web app.....	12
Conclusiones.....	16
Reflexión sobre lo aprendido.....	16
Dificultades encontradas.....	16
Posibles mejoras.....	16
Anexos.....	17

Introducción

Contexto del problema

Esta aplicación tiene como objetivo simular las operaciones fundamentales de un banco, permitiendo a los usuarios gestionar sus finanzas de manera eficiente. Su desarrollo surge de la necesidad de un sistema automatizado y confiable que centralice la gestión segura de los datos financieros, eliminando procesos manuales propensos a errores y proporcionando un acceso rápido y eficiente a la información. Entre sus funcionalidades principales se encuentran la creación y eliminación de usuarios, la gestión de cuentas (depósitos, retiros, transferencias) y el registro de transacciones.

Relevamiento de requerimientos

Descripción del problema o sistema a modelar

La necesidad de esta aplicación bancaria surge de la demanda de un sistema automatizado y confiable que permita a los usuarios realizar sus operaciones bancarias de manera digital. El problema principal que resuelve es la gestión centralizada y segura de los datos financieros, eliminando la necesidad de procesos manuales propensos a errores y proporcionando un acceso rápido y eficiente a la información. La aplicación busca ofrecer una plataforma intuitiva para que los usuarios administren sus fondos, realicen pagos y consulten su historial transaccional, mientras que el banco mantiene un registro preciso de todas las actividades.

Requisitos funcionales y no funcionales

Requisitos Funcionales:

- **Gestión de Usuarios:**
 - Crear nuevos usuarios (clientes).
 - Eliminar usuarios existentes.
 - Actualizar información de usuarios (dirección, teléfono, etc.).
- **Gestión de Cuentas:**
 - Abrir nuevas cuentas bancarias para usuarios.
 - Cerrar cuentas bancarias.
 - Consultar saldo de cuentas.
- **Gestión de Transacciones:**
 - Realizar depósitos.
 - Realizar retiros.
 - Realizar transferencias entre cuentas.
 - Consultar historial de transacciones.
- **Seguridad:**
 - Autenticación de usuarios para acceder a sus cuentas.

Requisitos No Funcionales:

- **Rendimiento:** La aplicación debe ser capaz de manejar un alto volumen de transacciones simultáneas sin degradación significativa del rendimiento. Las consultas de saldo y el registro de transacciones deben ser rápidos.
- **Seguridad:** Implementar medidas robustas para proteger los datos sensibles de los clientes (información personal, saldos de cuenta) contra accesos no autorizados y fraudes.
- **Disponibilidad:** El sistema debe estar disponible 24/7 con un mínimo tiempo de inactividad programado.
- **Escalabilidad:** El diseño de la base de datos debe permitir el crecimiento futuro en el número de usuarios y transacciones sin requerir rediseños fundamentales.
- **Integridad de Datos:** Garantizar la consistencia y precisión de los datos, especialmente en operaciones críticas como transferencias de fondos, para evitar inconsistencias.

Modelo conceptual

Diagrama entidad-relación (DER)



Explicación de entidades, relaciones, atributos y claves

A continuación, se describe el diagrama Entidad-Relación (E-R) propuesto para la aplicación bancaria:

- **Entidades:**

- **Usuario:** Representa a los clientes del banco.
 - **Atributos:**
 - `id_usuario` (Clave Primaria - PK)
 - `nombre`
 - `apellido`
 - `dni` (Único)
 - `direccion`
 - `telefono`
 - `email`
 - `fecha_nacimiento`
- **Cuenta:** Representa las cuentas bancarias de los usuarios.
 - **Atributos:**
 - `id_cuenta` (PK)
 - `numero_cuenta` (Único)
 - `tipo_cuenta` (ej. "Ahorro", "Corriente")
 - `saldo`
 - `fecha_apertura`
 - `id_usuario` (Clave Foránea - FK, referencia a `Usuario`)
- **Transaccion:** Representa cada movimiento de dinero realizado en las cuentas.
 - **Atributos:**
 - `id_transaccion` (PK)
 - `monto`
 - `tipo_transaccion` (ej. "Deposito", "Retiro", "Transferencia")
 - `fecha_hora`
 - `id_cuenta_origen` (FK, referencia a `Cuenta`)
 - `id_cuenta_destino` (FK, referencia a `Cuenta`, puede ser nulo para depósitos/retiros)

- **Relaciones:**

- **Usuario tiene Cuenta:** Una relación de uno a muchos (1:N). Un `Usuario` puede tener varias `Cuentas`, pero una `Cuenta` pertenece a un único `Usuario`.
 - **Cardinalidad:** Un Usuario (1) a muchas Cuentas (N).
- **Cuenta realiza Transaccion:** Una relación de uno a muchos (1:N) donde una `Cuenta` puede ser el origen o destino de muchas `Transacciones`.
 - **Cardinalidad:** Una Cuenta (1) a muchas Transacciones (N).

- **Notas:** Una **Transaccion** siempre tiene una **Cuenta** de origen. Si es una transferencia, también tendrá una **Cuenta** de destino.

Modelo lógico

Transformación del modelo ER al modelo relacional

Basado en el modelo E-R, las tablas relacionales serían las siguientes:

- **Usuario:**
 - **id_usuario** (PK)
 - **nombre**
 - **apellido**
 - **dni** (UNIQUE)
 - **direccion**
 - **telefono**
 - **email**
 - **fecha_nacimiento**
- **Cuenta:**
 - **id_cuenta** (PK)
 - **numero_cuenta** (UNIQUE)
 - **tipo_cuenta**
 - **saldo**
 - **fecha_apertura**
 - **id_usuario** (FK) → **Usuario** (**id_usuario**)
- **Transaccion:**
 - **id_transaccion** (PK)
 - **monto**
 - **tipo_transaccion**
 - **fecha_hora**
 - **id_cuenta_origen** (FK) → **Cuenta**(**id_cuenta**)
 - **id_cuenta_destino** (FK, NULLABLE) → **Cuenta**(**id_cuenta**)

Normalización (1NF, 2NF y 3NF)

El diseño de base de datos propuesto cumple con la Primera Forma Normal (1NF) y la Segunda Forma Normal (2NF) al asegurar que todos los atributos son atómicos y que los atributos que no clave dependen completamente de la clave primaria.

Además, el diseño adhiere a la Tercera Forma Normal (3NF). Esto se asegura al eliminar las dependencias transitivas, garantizando que todos los atributos no clave en cada tabla dependen directamente de la clave primaria y no de otros atributos no clave. Por ejemplo, en la tabla Usuario, **nombre**, **apellido**, **direccion**, **telefono**, **email** y **fecha_nacimiento** dependen directamente de **id_usuario**, son dependencias de otros atributos no clave.

Restricciones de integridad referencial

Para mantener la integridad de los datos entre las entidades, se establecen las siguientes restricciones de integridad referencial:

- Cuenta y Usuario: la columna **id_usuario** en la tabla cuenta es una clave foránea que referencia a **id_usuario** en la tabla Usuario. Esto asegura que cada cuenta esté asociada a un usuario existente.
- Transaccion y Cuenta: las columnas **id_cuenta_origen** y **id_cuenta_destino** en la tabla Transaccion son claves foráneas que refieren a **id_cuenta** en la tabla Cuenta. Esto garantiza que todas las transacciones estén vinculadas a cuentas válidas. La columna **id_cuenta_destino** puede ser nula para depósitos o retiros.

Reglas de negocio a nivel de Base de Datos

Para asegurar la coherencia y validez de las operaciones bancarias, el modelo lógico incorpora las siguientes reglas de negocio, que pueden ser aplicadas mediante restricciones en la base de datos:

- El saldo en tabla Cuenta debe ser un valor no negativo.
- El monto de cualquier Transaccion debe ser siempre mayor que cero.
- El tipo_cuenta en la tabla Cuenta sólo puede ser 'Ahorro' o 'Corriente'.
- El tipo_transaccion en la tabla Transaccion solo puede ser 'Deposito', 'Retiro' o 'Transferencia'.

Diccionario de datos (nombre de tablas, columnas, tipos de datos)

Tabla: Usuario

Columna	Tipo de Dato	Descripción
id_usuario	INT	clave primaria, autoincremental
nombre	VARCHAR(50)	Nombre del cliente
apellido	VARCHAR(50)	Apellido del cliente
dni	VARCHAR(15) UNIQUE	Documento nacional de identidad
dirección	VARCHAR(100)	Dirección del cliente
telefono	VARCHAR(20)	Número de teléfono
email	VARCHAR(100)	Correo electrónico
fecha_nacimiento	DATE	Fecha de nacimiento

Tabla: Cuenta

Columna	Tipo de Dato	Descripción
---------	--------------	-------------

id_cuenta	INT	Clave primaria, autoincremental
numero_cuenta	VARCHAR(20) UNIQUE	Número único de cuenta bancaria
tipo_cuenta	VARCHAR(20)	Tipo de cuenta ('Ahorro', 'Corriente', etc)
saldo	DECIMAL(12,2)	Monto actual en cuenta
fecha_apertura	DATE	Fecha de apertura de la cuenta
id_usuario	INT	Clave foránea → Cuenta(id_cuenta)

Tabla: Transaccion

Columna	Tipo de Dato	Descripción
id_transaccion	INT	Clave primaria, autoincremental
monto	DECIMAL(12,2)	Monto de dinero involucrado
tipo_transacción	VARCHAR(20)	'Depósito', 'Retiro', 'Transferencia'
fecha_hora	DATETIME	Fecha y hora de la operación
id_cuenta_origen	INT	Clave foránea → Cuenta(id_cuenta)
id_cuenta_destino	INT (nullable)	Clave foránea → Cuenta(id_cuenta), puede ser nulo

Implementación

Esta sección detalla cómo el modelo lógico de la base de datos se traduce a una implementación concreta utilizando scripts de SQL. Incluyendo la creación de tablas, la población de datos de ejemplo y consultas representativas.

El siguiente script muestra cómo se implementan las tablas en SQL:

```
CREATE TABLE Usuario (  
    id_usuario INT PRIMARY KEY AUTO_INCREMENT,  
    nombre VARCHAR(50) NOT NULL,  
    apellido VARCHAR(50) NOT NULL,  
    dni VARCHAR(15) UNIQUE NOT NULL,  
    direccion VARCHAR(100),  
    telefono VARCHAR(20),  
    email VARCHAR(100),  
    fecha_nacimiento DATE  
);  
  
CREATE TABLE Cuenta (  
    id_cuenta INT PRIMARY KEY AUTO_INCREMENT,  
    numero_cuenta VARCHAR(20) UNIQUE NOT NULL,  
    tipo_cuenta VARCHAR(20) CHECK (tipo_cuenta IN ('Ahorro', 'Corriente')),  
    saldo DECIMAL(12,2) DEFAULT 0,  
    fecha_apertura DATE NOT NULL,  
    id_usuario INT NOT NULL,  
    FOREIGN KEY (id_usuario) REFERENCES Usuario(id_usuario)  
);  
  
CREATE TABLE Transaccion (  
    id_transaccion INT PRIMARY KEY AUTO_INCREMENT,  
    monto DECIMAL(12,2) NOT NULL CHECK (monto > 0),  
    tipo_transaccion VARCHAR(20) CHECK (tipo_transaccion IN ('Deposito', 'Retiro', 'Transferencia',  
    fecha_hora DATETIME NOT NULL,  
    id_cuenta_origen INT NOT NULL,  
    id_cuenta_destino INT,  
    FOREIGN KEY (id_cuenta_origen) REFERENCES Cuenta(id_cuenta),  
    FOREIGN KEY (id_cuenta_destino) REFERENCES Cuenta(id_cuenta)  
);
```

La población de datos (inserts) se vería de esta forma:

```
1  -- Usuarios  
2  v INSERT INTO Usuario (nombre, apellido, dni, direccion, telefono, email, fecha_nacimiento) VALUES  
3  ('Juan', 'Pérez', '12345678', 'Av. Siempreviva 123', '1234567890', 'juan@example.com', '1990-05-10'),  
4  ('Ana', 'López', '87654321', 'Calle Falsa 456', '0987654321', 'ana@example.com', '1985-12-01');  
5  
6  -- Cuentas  
7  v INSERT INTO Cuenta (numero_cuenta, tipo_cuenta, saldo, fecha_apertura, id_usuario) VALUES  
8  ('0001', 'Ahorro', 5000.00, '2024-01-15', 1),  
9  ('0002', 'Corriente', 10000.00, '2024-02-20', 1),  
10 ('0003', 'Ahorro', 2500.00, '2024-03-10', 2);  
11  
12 -- Transacciones  
13 v INSERT INTO Transaccion (monto, tipo_transaccion, fecha_hora, id_cuenta_origen, id_cuenta_destino) VALUES  
14 (1000.00, 'Deposito', '2025-06-01 10:00:00', 1, NULL),  
15 (500.00, 'Retiro', '2025-06-02 11:30:00', 1, NULL),  
16 (750.00, 'Transferencia', '2025-06-03 09:00:00', 2, 3);  
17
```

Consultas representativas (SELECTs con joins, subconsultas y agregaciones)

Estas consultas demuestran cómo la base de datos puede responder a preguntas clave y soportar las funcionalidades de la aplicación.

```
1  -- 1. Listar todos los usuarios con el saldo de todas sus cuentas --
2  SELECT U.nombre, U.apellido, C.numero_cuenta, C.tipo_cuenta, C.saldo
3  FROM Usuario U
4  JOIN Cuenta C ON U.id_usuario = C.id_usuario;
5
6  -- 2. Calcular el saldo total de todas las cuentas de un usuario específico (ej. Juan Cralos) --
7  SELECT U.nombre, U.apellido, SUM(C.saldo) AS saldo_total
8  FROM Usuario U
9  JOIN Cuenta C ON U.id_usuario = C.id_usuario
10 WHERE U.nombre = 'Juan' AND U.apellido = 'Pérez'
11 GROUP BY U.id_usuario;
12
13 -- 3. Obtener todas las transacciones de una cuenta específica (ej. cuenta '0001') --
14 SELECT T.id_transaccion, T.monto, T.tipo_transaccion, T.fecha_hora,
15        COALESCE(C_origen.numero_cuenta, 'N/A') AS cuenta_origen,
16        COALESCE(C_destino.numero_cuenta, 'N/A') AS cuenta_destino
17 FROM Transaccion T
18 LEFT JOIN Cuenta C_origen ON T.id_cuenta_origen = C_origen.id_cuenta
19 LEFT JOIN Cuenta C_destino ON T.id_cuenta_destino = C_destino.id_cuenta
20 WHERE C_origen.numero_cuenta = '0001' OR C_destino.numero_cuenta = '0001';
21
22 -- 4. Encontrar el monto total de depósitos realizados en el mes de junio de 2025 --
23 SELECT SUM(monto) AS total_depositos_junio_2025
24 FROM Transaccion
25 WHERE tipo_transaccion = 'Deposito' AND fecha_hora BETWEEN '2025-06-01 00:00:00' AND '2025-06-30 23:59:59';
26
27 -- 5. Obtener los usuarios que no tienen ninguna cuenta asociada --
28 SELECT U.nombre, U.apellido
29 FROM Usuario U
30 LEFT JOIN Cuenta C ON U.id_usuario = C.id_usuario
31 WHERE C.id_cuenta IS NULL;
32
```

Casos de uso / Consultas avanzadas

Estos ejemplos muestran como la base de datos puede soportar escenarios más específicos o complejos de la aplicación.

```
37 -- 1. Listar las 3 cuentas con mayor saldo --
38 SELECT numero_cuenta, tipo_cuenta, saldo
39 FROM Cuenta
40 ORDER BY saldo DESC
41 LIMIT 3;
42
43 -- 2. Encontrar el número de transacciones realizadas por cada tipo de transacción --
44 SELECT tipo_transaccion, COUNT(id_transaccion) AS numero_de_transacciones
45 FROM Transaccion
46 GROUP BY tipo_transaccion;
47
48 -- 3. Obtener el DNI de los usuarios que han realizado al menos una transferencia como origen --
49 SELECT DISTINCT U.dni, U.nombre, U.apellido
50 FROM Usuario U
51 JOIN Cuenta C ON U.id_usuario = C.id_usuario
52 JOIN Transaccion T ON C.id_cuenta = T.id_cuenta_origen
53 WHERE T.tipo_transaccion = 'Transferencia';
54
```

Ejemplos que muestran cómo se responde a los requerimientos con SQL

Aquí se enlazan directamente las consultas con los requisitos funcionales definidos previamente:

- **Requisito Funcional: Consultar saldo de cuentas.**
 - `SELECT saldo FROM Cuenta WHERE numero_cuenta = '0001';`
- **Requisito Funcional: Consultar historial de transacciones.**
 - `SELECT * FROM Transaccion WHERE id_cuenta_origen = (SELECT id_cuenta FROM Cuenta WHERE numero_cuenta = '0001')`
 - `OR id_cuenta_destino = (SELECT id_cuenta FROM Cuenta WHERE numero_cuenta = '0001')`
 - `ORDER BY fecha_hora DESC;`
- **Requisito Funcional: Realizar depósitos.** (Representado por un INSERT, pero la aplicación maneja la lógica para actualizar el saldo)
Este INSERT representa un depósito. La lógica de la aplicación actualizará el saldo.
 - `INSERT INTO Transaccion (monto, tipo_transaccion, fecha_hora, id_cuenta_origen, id_cuenta_destino) VALUES (250.00, 'Deposito', '2025-06-25 10.30.00', 1, NULL);`
 En un app real, esto iría acompañado de una UPDATE al saldo de la cuenta
 - `UPDATE Cuenta SET saldo = saldo + 250.00 WHERE id_cuenta = 1;`

Triggers, procedimientos almacenados, vistas, índices

Para un sistema bancario es crucial asegurar la consistencia de los datos y optimizar el rendimiento. Aunque la implementación completa de estos elementos es parte del desarrollo de la aplicación, se pueden prever sus usos:

- **Triggers:** Se podría implementar un TRIGGER para actualizar automáticamente el saldo de una Cuenta cada vez que se inserta una nueva Transaccion de tipo 'Deposito' o 'Retiro', asegurando que el saldo siempre refleje los movimientos.
 - *Ejemplo conceptual de trigger (no para implementar aún, sólo para describir su propósito):* Un trigger AFTER INSERT ON Transaccion que sume/reste el monto al saldo de la Cuenta correspondiente.
- **Procedimientos almacenados:** Para operaciones complejas como una "Transferencia", un procedimiento almacenado podría asegurar la tonicidad (ACID) de la operación, debitando de una cuenta y acreditando en otra única transacción lógica. Esto reduce el riesgo de inconsistencias.
 - *Ejemplo conceptual de procedimiento almacenado (propósito):* Un procedimiento llamado RealizarTransferencia(id_origen, id_destino, monto) que realice las actualizaciones de saldo y registre la transacción.
- **Vistas:** Se podrían crear vistas para simplificar consultas comunes o para controlar el acceso a datos sensibles. Por ejemplo, una vista de MovimientosRecientes para un usuario específico o una vista que combine información de Usuario y Cuenta para reportes.

```

56 -- Ejemplo de Vista para obtener un resumen de cuentas y sus dueños --
57 CREATE VIEW VistaCuentaConUsuarios AS
58 SELECT U.nombre AS nombre_usuario, U.apellido AS apellido_usuario, C.numero_cuenta C.tipo_cuenta, C.saldo
59 FROM Usuario U
60 JOIN Cuenta C ON U.id_usuario = C.id_usuario
61

```

- **Índices:** Para mejorar el rendimiento de las consultas, especialmente en tablas grandes, se recomendaría crear índices en columnas frecuentemente utilizadas en cláusulas WHERE, JOIN y ORDER BY.
 - *Ejemplos de columnas para índices:* dni en Usuario, numero_cuenta en Cuenta, fecha_hora y tipo_transaccion en Transaccion.

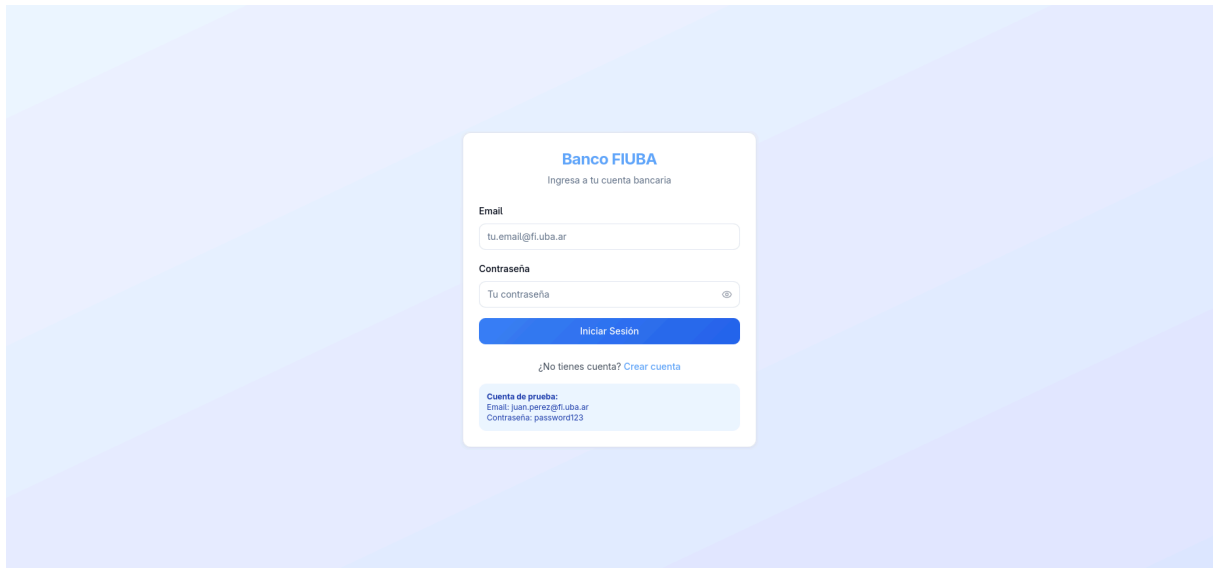
```
64  -- Ejemplo de creación índice --  
65  CREATE INDEX idx_dni_usuario ON Usuario (dni);  
66  CREATE INDEX idx_numero_cuenta ON Cuenta (numero_cuenta);  
67  CREATE INDEX idx_fecha_hora_transaccion ON Transaccion (fecha_hora);  
68
```

Interfaz de usuario

Recorrido de la web app

A continuación haremos un recorrido de las diferentes pantallas que se pueden encontrar en la web app que hace de frontend a la aplicación bancaria.

1. La app inicia en la pantalla de login.
El usuario debe ingresar su **email** y **contraseña** para acceder.
Hay un link para registrarse si aún no tiene cuenta.



The login screen for Banco FIUBA features a white card centered on a light blue background with diagonal stripes. The card has the bank's logo and name at the top, followed by the instruction 'Ingresa a tu cuenta bancaria'. It contains two input fields for 'Email' and 'Contraseña', a blue 'Iniciar Sesión' button, a link to 'Crear cuenta', and a 'Cuenta de prueba' section with test credentials.

Banco FIUBA
Ingresa a tu cuenta bancaria

Email
tu_email@fi.uba.ar

Contraseña
Tu contraseña

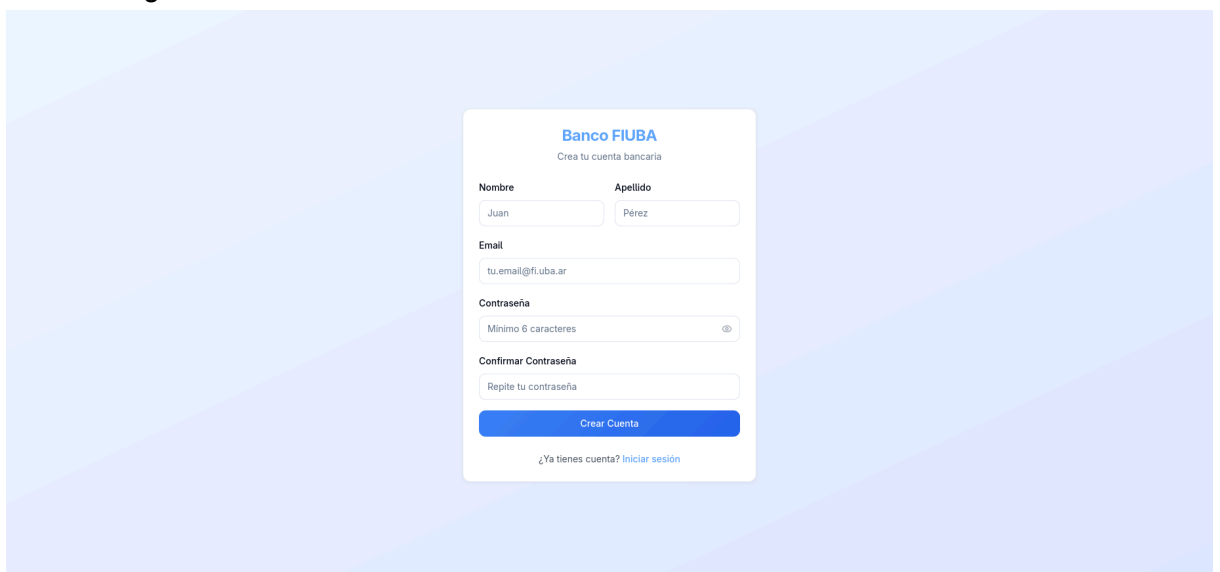
Iniciar Sesión

¿No tienes cuenta? [Crear cuenta](#)

Cuenta de prueba:
Email: juan.perez@fi.uba.ar
Contraseña: password123

2. Si el usuario selecciona registrarse, se le solicita:
 - Nombre
 - Apellido
 - Email
 - Contraseña
 - Confirmar contraseña

Una vez registrado, accede al dashboard.



The registration screen for Banco FIUBA features a white card centered on a light blue background with diagonal stripes. The card has the bank's logo and name at the top, followed by the instruction 'Crea tu cuenta bancaria'. It contains input fields for 'Nombre', 'Apellido', 'Email', 'Contraseña', and 'Confirmar Contraseña', a blue 'Crear Cuenta' button, and a link to 'Iniciar sesión'.

Banco FIUBA
Crea tu cuenta bancaria

Nombre Apellido
Juan Pérez

Email
tu_email@fi.uba.ar

Contraseña
Mínimo 6 caracteres

Confirmar Contraseña
Repite tu contraseña

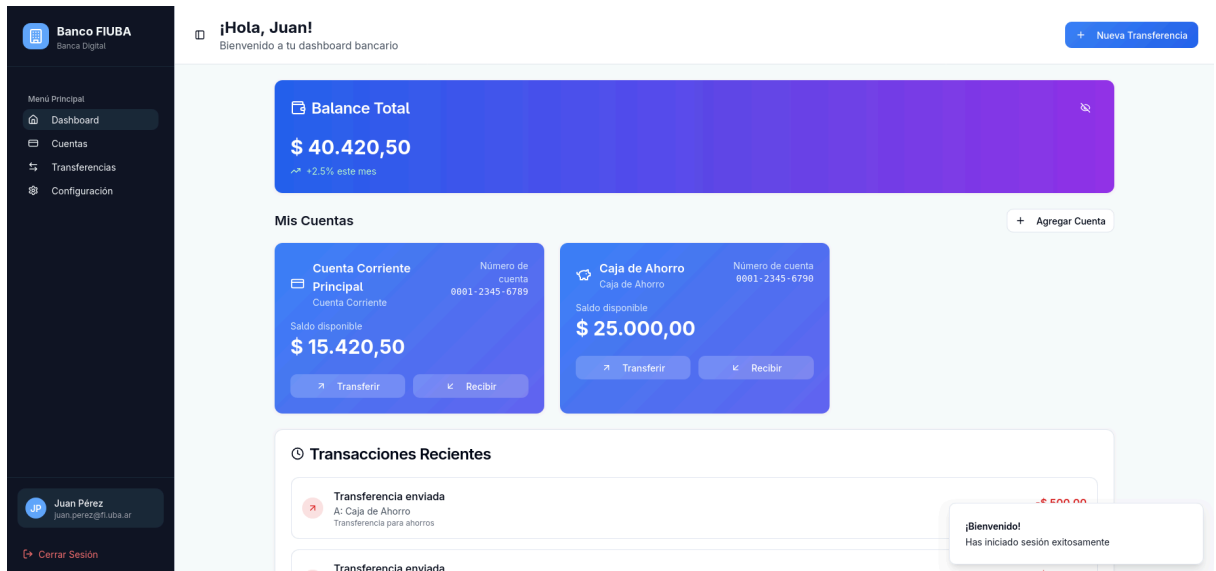
Crear Cuenta

¿Ya tienes cuenta? [Iniciar sesión](#)

3. Es la vista principal después del login.

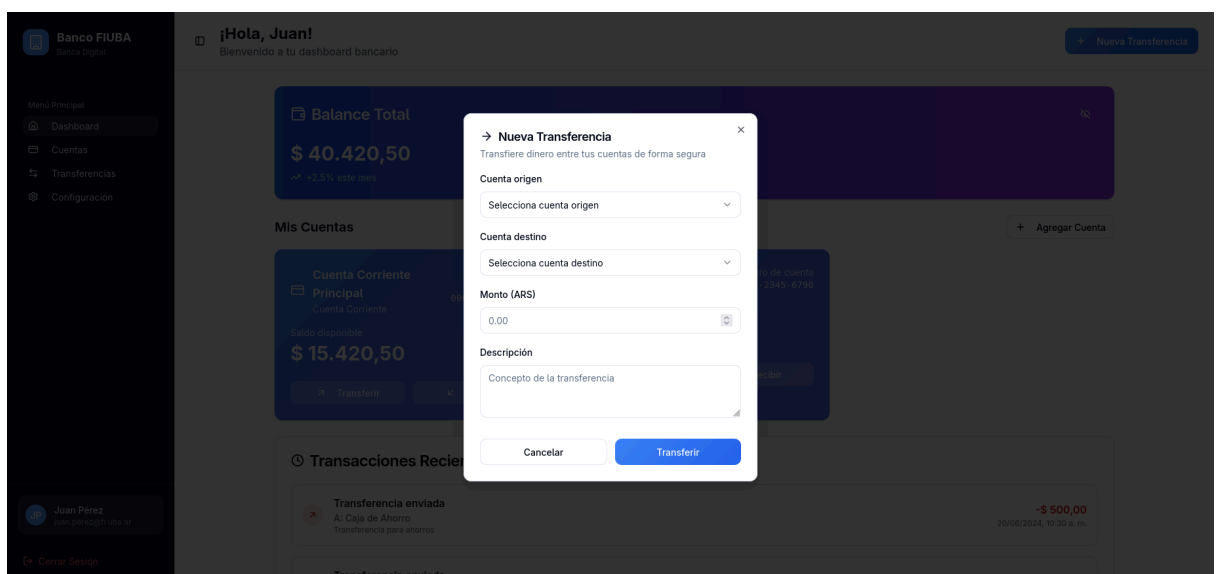
Muestra:

- El **Balance Total** del usuario.
- **Cards** con el detalle de cada cuenta (por ejemplo: cuenta corriente, caja de ahorro).
- Un listado de **Transacciones Recientes**.
- Un botón destacado para iniciar una **Nueva Transferencia**

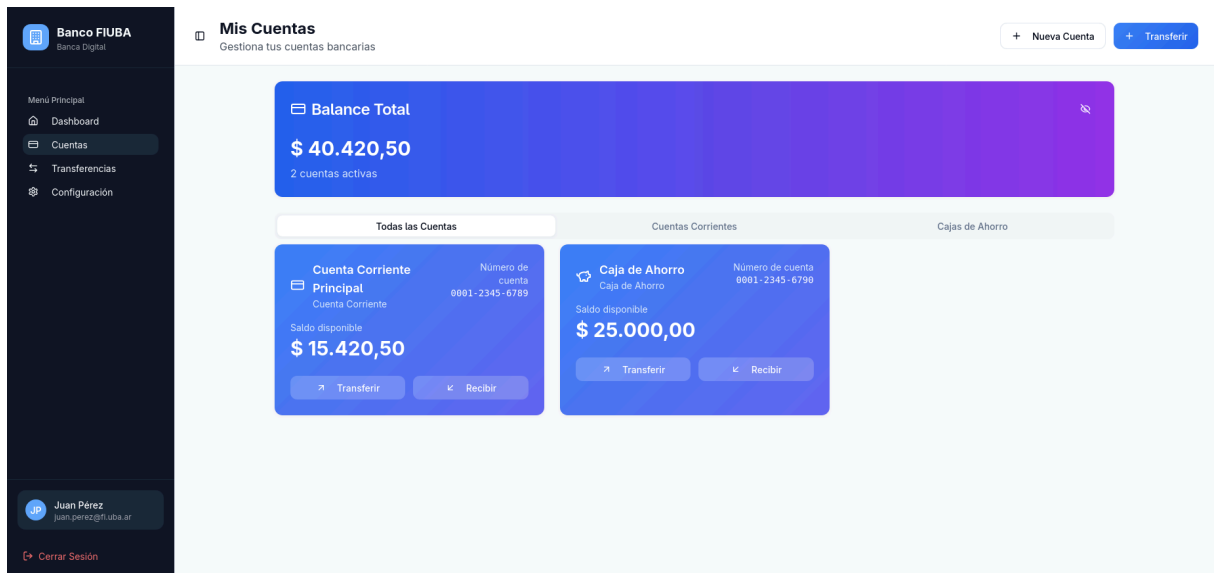


4. Al presionar el botón de Nueva Transferencia se abre un modal con los siguientes campos:

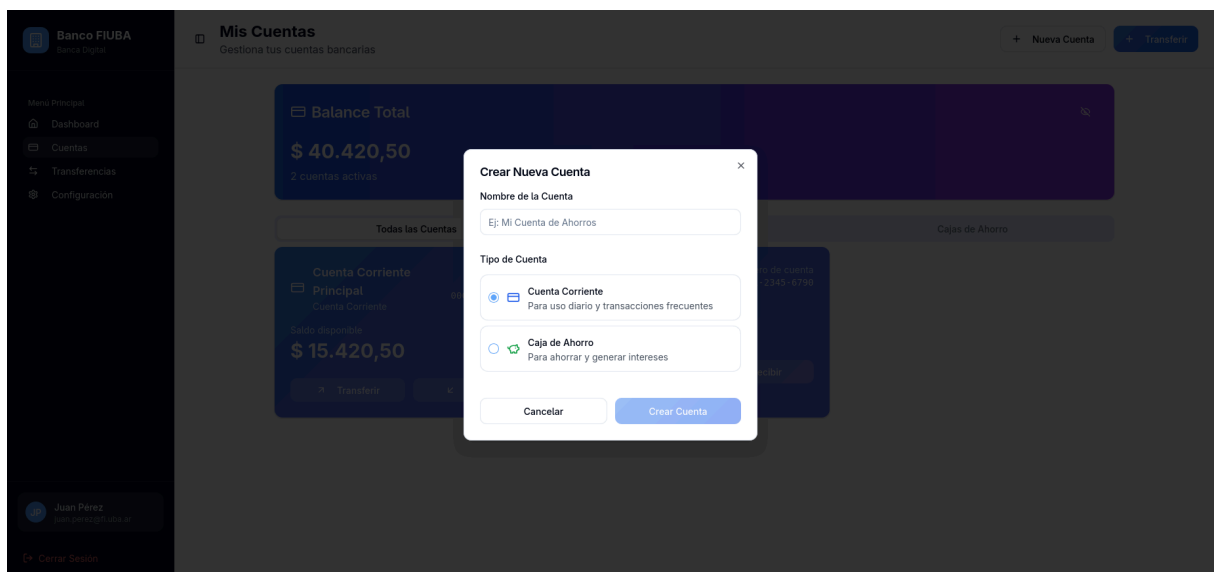
- Cuenta de **origen** (seleccionable entre las cuentas del usuario)
- Cuenta de **destino**
- **Monto** a transferir
- **Descripción** (opcional)



5. Desde el menú, el usuario puede acceder a la sección "Cuentas", donde se muestran:
- El **Balance Total** de las cuentas.
 - Las **cards** de cada cuenta existente
 - Un botón para **"Crear nueva cuenta"**
 - Un botón para **"Transferir"**



6. Al presionar el botón "Crear nueva cuenta", aparece un modal que permite seleccionar el tipo de cuenta a crear:
- **Caja de Ahorro**
 - **Cuenta Corriente**



7. En la sección de Transferencias se presentan:
- Cards con resumen de **transferencias de hoy**
 - **Transferencias del mes**
 - **Estados de las cuentas** (activa, suspendida)
 - Una sección para hacer una **"Transferencia rápida"**

Banco FIUBA
Banca Digital

Menú Principal
Dashboard
Cuentas
Transferencias
Configuración

Transferencias
Administra tus transferencias entre cuentas

+ Nueva Transferencia

Transferencias Hoy
0
Sin transferencias hoy

Este Mes
0
Total: \$ 0,00

Estado
Activo
3 cuentas disponibles

Transferencia Rápida Historial

Transferencia Rápida

Cuenta origen
Selecciona cuenta origen

Cuenta destino
Selecciona cuenta destino

Monto (ARS)
0.00

Transferir Ahora

Juan Pérez
juan.perez@fi.uba.ar

Cerrar Sesión

8. Finalmente, se le ofrece al usuario la opción de que pueda editar sus datos personales y hacer otros cambios desde la sección de Configuración

Banco FIUBA
Banca Digital

Menú Principal
Dashboard
Cuentas
Transferencias
Configuración

Configuración
Administra tu cuenta y preferencias de la aplicación

Perfil Seguridad Notificaciones Privacidad

Información Personal
Actualiza tu información personal y de contacto

Nombre
Juan

Apellido
Pérez

Correo Electrónico
juan.perez@fi.uba.ar

Teléfono
+54 11 1234-5678

Guardar Cambios

Juan Pérez
juan.perez@fi.uba.ar

Cerrar Sesión

Conclusiones

El desarrollo de este trabajo permitió diseñar e implementar un sistema de base de datos robusto para una aplicación bancaria, simulando operaciones clave como gestión de usuarios, cuentas y transacciones. Se logró un modelo lógico y físico que respeta las formas normales, garantiza la integridad referencial y responde a los requerimientos funcionales planteados. La estructura modular y normalizada facilita tanto la escalabilidad del sistema como su mantenimiento.

Reflexión sobre lo aprendido

Durante el desarrollo del trabajo se afianzaron conocimientos teóricos en modelado de datos, normalización y SQL. Además, se incorporaron habilidades prácticas en la implementación de estructuras eficientes, consultas complejas y elementos avanzados como triggers y procedimientos almacenados. La integración con una interfaz de usuario también permitió reflexionar sobre la importancia de una buena arquitectura de base de datos para el funcionamiento fluido de una aplicación real.

Dificultades encontradas

Entre los principales desafíos se destacaron:

- La correcta definición de las claves foráneas y sus implicancias en la integridad referencial.
- El diseño de las transacciones para contemplar todos los posibles casos (depósitos, retiros, transferencias) sin inconsistencias.
- La creación de consultas SQL que contemplaran condiciones específicas sin generar ambigüedades ni errores de duplicación.
- La sincronización entre la lógica de backend y el modelo de datos, especialmente al implementar reglas de negocio.

Posibles mejoras

Algunas mejoras que podrían implementarse en futuras iteraciones del sistema incluyen:

- Implementación efectiva de triggers y procedimientos almacenados para automatizar operaciones críticas.
- Validaciones adicionales desde el lado de la base de datos para mayor seguridad (por ejemplo, límites máximos de transferencia).
- Optimización mediante índices en consultas de uso frecuente.
- Incorporación de logs o auditorías para monitorear actividades sensibles.
- Ampliar la funcionalidad de la interfaz web para incluir reportes financieros personalizados y filtros más avanzados.

Anexos

En esta sección se presentan algunos de los prompts utilizados durante el desarrollo de este proyecto, enfocados en la resolución de desafíos complejos relacionados con el diseño y la implementación de bases de datos.

Prompts de Diseño y Normalización:

1. Prompt: "Estamos diseñando la base de datos para una aplicación bancaria. Necesitamos asegurar la Tercera Forma Normal (3FN) para las entidades Usuario, Cuenta y Transaccion. Explica detalladamente, con ejemplos de nuestro esquema, cómo aplicarías la 3FN, prestando especial atención a la eliminación de dependencias transitivas y justificando por qué es crucial para la integridad de los datos en un sistema bancario."
2. Prompt: "Considerando la entidad Transaccion con atributos como monto, tipo_transaccion, id_cuenta_origen e id_cuenta_destino, y la necesidad de manejar depósitos, retiros y transferencias: ¿Cómo diseñarías las restricciones de integridad referencial para id_cuenta_origen y id_cuenta_destino de modo que id_cuenta_destino pueda ser nulo solo para depósitos/retiros, pero sea obligatorio para transferencias? Detalla la lógica SQL o las reglas de negocio necesarias para implementar esto a nivel de base de datos."
3. Prompt: "Necesitamos una consulta SQL que genere un informe de movimientos mensuales para una cuenta específica, mostrando la fecha, el tipo de transacción, el monto y si fue un crédito o un débito al saldo de la cuenta. Además, la consulta debe calcular el saldo acumulado después de cada transacción dentro de ese mes. Demuestra cómo construirías esta consulta compleja utilizando funciones de ventana o subconsultas correlacionadas, explicando los pasos lógicos."
4. Prompt: "Dada la necesidad de mantener la integridad del saldo de una cuenta durante las transferencias entre usuarios, propón una solución a nivel de base de datos que garantice la atomicidad (ACID) de la operación, es decir, que la debitación y la acreditación ocurran simultáneamente o ninguna ocurra. Describe cómo implementarías esto utilizando un procedimiento almacenado y transacciones explícitas (BEGIN TRAN, COMMIT, ROLLBACK), explicando el flujo de la lógica para manejar posibles errores."

Prompts de Optimización y Rendimiento:

1. Prompt: "Nuestro sistema bancario manejará un alto volumen de transacciones y consultas. Identifica las columnas clave en las tablas Usuario, Cuenta y Transaccion que serían candidatas ideales para la creación de índices, justificando tu elección en términos de tipos de consultas frecuentes (ej., búsquedas por DNI, consultas de saldo, historial de transacciones por fecha) y su impacto en el rendimiento de la aplicación. Además, describe qué tipo de índice (ej. B-tree, Hash) sería más apropiado para cada caso."