“Año de la recuperación y consolidación de la economía peruana”

UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

APLICATIVO - BANCO DE LA NACION

CURSO: BASE DE DATOS

DOCENTE: MAYENKA FERNANDEZ CHAMBI

PRESENTADO POR:	**CODIGO**
Arizaca Machaca Erick Abel	231449
Ñaupa Valeriano Hector Pedro	230347
Quispe Tapia Alex Daniel	231522
Ponce Quilca Mishael Fernando	230452

INDICES

1 Resumen del negocio

La aplicación tiene como objetivo principal brindar a los usuarios una plataforma digital donde puedan gestionar de forma segura y eficiente sus operaciones bancarias. Esto incluye el acceso a saldos, transferencias, pagos, recargas, entre otros. Está orientada a personas naturales y jurídicas que poseen cuentas en el Banco de la Nación y que buscan una alternativa rápida, remota y confiable a las operaciones presenciales.

Ofrecer servicios bancarios a distancia:

El aplicativo permite realizar consultas de saldos, movimientos, transferencias, pagos y otras operaciones sin necesidad de acudir a una agencia física.

Fomentar la inclusión financiera:

Al llegar a más usuarios a través de la tecnología móvil, el Banco de la Nación democratiza el acceso a servicios financieros, especialmente en zonas rurales o con dificultades para acceder a agencias.

Mejorar la experiencia del cliente:

El aplicativo ofrece una interfaz intuitiva y segura, con la posibilidad de realizar transacciones rápidas y seguras desde cualquier lugar y en cualquier momento.

Impulsar la innovación digital:

El Banco de la Nación se adapta a las nuevas tecnologías y tendencias del mercado financiero, incorporando funcionalidades innovadoras al aplicativo para satisfacer las necesidades de sus clientes.

Funcionalidades clave del aplicativo:

- Consultar saldos y movimientos de cuentas.
- Realizar transferencias entre cuentas del Banco de la Nación y a otros bancos.
- Ubicación de agencias, cajeros automáticos y agentes MultiRed.
- Notificaciones por operaciones realizadas.
- Gestionar perfiles y datos personales.
- Pago de servicios.
- Acceso a información sobre productos y servicios.

Beneficios para el usuario:

- Ahorro de tiempo y esfuerzo al realizar operaciones bancarias desde cualquier lugar.
- Mayor comodidad y flexibilidad para gestionar sus finanzas.
- Mayor seguridad en las transacciones gracias a la encriptación y otros mecanismos de seguridad.
- Acceso a información y herramientas útiles para la gestión financiera.

2 Análisis

2.1 Funcionalidades de la aplicación con los datos (por ejemplo ingreso de datos, edición de datos, eliminación de datos, recuperación de datos, tipos de usuarios)

2.1.1 Operaciones CRUD básicas

Ingreso de datos:

- Registro completo de usuarios (procedimiento registrar_usuario_completo)
- Creación automática de cuentas (trigger crear_cuenta_con_cci)
- Registro de transferencias, giros, recargas y pagos de servicios

Edición de datos:

- Actualización de saldos mediante operaciones financieras
- Modificación de datos de contacto/dirección de usuarios

Eliminación de datos:

- Implementa eliminación lógica (no física) mediante campos de estado por requisitos de auditoría

Recuperación de datos:

- Consulta de saldos (ver_saldo)
- Historial de operaciones
- Reportes de transacciones por periodo

Tipos de usuarios

- Clientes, operaciones basicas
- Administradores, generación de reportes y todo ello.
- Cajeros y asesores, operaciones especiales

2.2 Análisis de operaciones especiales de negocio para identificar y argumentar la programación de funciones personalizadas, disparadores, y procedimientos almacenados.

2.2.1 Funciones personalizadas críticas

calcular_comision_transferencia

- Justificación: Implementa lógica compleja de comisiones interbancarias variables
- Beneficio: Centraliza reglas de negocio evitando redundancia

tiene_saldo_suficiente

- Justificación: Verificación crítica usada en múltiples procedimientos
- Beneficio: Garantiza consistencia en validaciones de fondos

2.2.2 Procedimientos almacenados esenciales

[Inicio] → [Validaciones] → [Operación principal] → [Registro auditoría] → [Confirmación]

2.2.3 Disparadores clave

- tr_historial_: Garantizan auditoría automática de operaciones
- crear_cuenta_con_cci: Automatiza generación de cuentas con formato estándar

2.3 Análisis de necesidades de lecturas intensivas para argumentar la creación de índices.

Consultas frecuentes:

- Saldo de la cuenta principal (alta demanda).
- Historial de operaciones (consultas por fechas).
- Índice por dni_usuario y fecha en la tabla de transacciones.
- Índice en numero_celular para transferencias.

2.4 Análisis de operaciones que deben ser controladas a nivel de transacciones para argumentar la incorporación de transacciones.

Operaciones críticas:

- Transferencias entre bancos (deben validarse completamente antes de finalizar).
- Giros con DNI (se requiere que tanto el envío como la recepción estén correctamente registrados).
- Pagos de servicios y recargas (se debe asegurar que el pago fue aceptado antes de descontar el saldo).

Justificación:

- Estas operaciones deben ser atómicas, ya que un fallo parcial puede generar pérdida de dinero o duplicación de transacciones.
- Se requiere garantizar consistencia e integridad de datos, por lo cual deben estar encapsuladas en transacciones SQL con control de errores.

3 Diseño

3.1 Modelo Relacional de la Base de Datos

El modelo relacional representa la estructura lógica de la base de datos del sistema del Banco de la Nación. Cada entidad del sistema ha sido transformada en una relación (tabla), definiendo adecuadamente sus atributos, claves primarias y claves foráneas. A continuación, se detalla el conjunto de relaciones que conforman la base de datos:

3.1.1 Relaciones principales

- usuarios(dni[PK], nombre, apellidos, clave_digital, serie_tarjeta)
  Representa a los usuarios registrados en el sistema. El dni es la clave primaria que los identifica de forma única.
- ocupaciones_usuario(id_ocupacion[PK], dni[FK], ocupacion)
  Permite registrar una o más ocupaciones por usuario. Se asocia al usuario mediante su dni.
- contacto(id_contacto[PK], dni[FK], correo, celular)
  Registra los datos de contacto de cada usuario, como correo electrónico y número de celular.
- direccion_usuario(id_direccion[PK], dni[FK], departamento, provincia, distrito, direccion)
  Guarda la dirección física de los usuarios, segmentada por nivel geográfico.

3.1.2 Relaciones financieras

- cuentas(num_cuenta[PK], dni[FK], cci[UNIQUE], saldo)
  Define las cuentas bancarias vinculadas a cada usuario. Se incluye un código CCI único para transferencias interbancarias.
- historial(id_historial[PK], num_cuenta[FK],tipo_operacion, descripcion, monto, fecha)
  Almacena el historial de operaciones financieras realizadas desde cada cuenta.
- deposito(id_deposito[PK],num_cuenta[FK],monto,saldo_anterior,saldo_nuevo, fecha_hora)
  Registra las operaciones de depósito en una cuenta, incluyendo el saldo anterior y el nuevo

3.1.3 Relaciones con otras entidades bancarias

- bancos(id_banco[PK], nombre_banco)
  Contiene el listado de bancos a nivel nacional con los que se puede interactuar.
- banco_celular(id[PK], celular, id_banco[FK])
  Asocia un número de celular a una entidad bancaria, permitiendo operaciones como transferencias por celular.
- cci_afiliados(id[PK], cci[UNIQUE], id_banco[FK])
  Guarda códigos CCI afiliados a otras entidades financieras.

3.1.4 Transferencias y giros

- transferencias_celular(id_transferencia[PK],num_cuenta_origen[FK],celular_destino, id_banco_destino[FK], monto, fecha)
  Gestiona transferencias entre cuentas mediante el número de celular del destinatario.
- giros(id_giro[PK], num_cuenta_origen[FK],dni_destino, id_banco_destino[FK], monto, fecha)
  Registra giros bancarios a otros usuarios identificados por su DNI.
- transferencias_cuentas(id_transferencia[PK], num_cuenta_origen[FK], numero_cuenta_destino, id_banco_destino[FK], descripcion, monto, fecha)
  Representa transferencias internas entre cuentas dentro de la misma institución o hacia cuentas específicas de otros bancos.

- transferencias_cci(id_transferencia[PK], num_cuenta_origen[FK], destino, id_banco_destino[FK], descripcion, monto, fecha)
  Permite realizar transferencias interbancarias utilizando códigos CCI.

3.1.5 Servicios y pagos

- empresas_servicios(id_empresa[PK], nombre_empresa, tipo_servicio)
  Lista las empresas proveedoras de servicios como agua, luz o telefonía.
- deudas_servicios(id_deuda[PK], tipo_servicio, id_empresa[FK], codigo_pago, monto)
  Representa las deudas registradas por los usuarios hacia las empresas de servicios.
- pago_servicios(id_pago[PK], num_cuenta[FK], id_deuda[FK], fecha_pago)
  Registra el pago de deudas de servicios realizados desde una cuenta bancaria.
- empresas_recargas(id_empresa[PK], nombre_empresa)
  Guarda información sobre las empresas que ofrecen servicios de recarga móvil.
- recargas(id_recarga[PK], num_cuenta[FK], id_empresa[FK], numero, monto, fecha_recarga)
  Maneja las operaciones de recarga telefónica desde una cuenta hacia un número celular.

3.1.6 Ubicación

- ubicaciones_banco(id_ubicacion[PK], departamento, provincia, direccion)
  Define las ubicaciones físicas de las agencias del banco.
- ubicacion_usuario(id[PK], dni[FK], id_ubicacion[FK])
  Relaciona a los usuarios con una ubicación específica del banco, por ejemplo, para gestión territorial o servicios asociados.

3.2 Validación Detallada de Tercera Forma Normal (3FN)

La Tercera Forma Normal (3FN) es una etapa del proceso de normalización en el diseño de bases de datos relacionales, cuyo objetivo es eliminar redundancias y dependencias no deseadas. Una tabla está en 3FN si cumple con los requisitos de la Segunda Forma Normal (2FN) y, además, no contiene dependencias transitivas entre atributos no clave y la clave primaria. Es decir, todos los atributos no clave deben depender únicamente de la clave primaria, y no de otros atributos no clave. Aplicar la 3FN contribuye a mejorar la integridad de los datos, facilita el mantenimiento y reduce la posibilidad de anomalías en las operaciones de inserción, actualización y eliminación.

3.2.1 Tabla usuarios(dni [PK], nombre, apellidos, clave_digital, serie_tarjeta)

- 1FN (Primera Forma Normal)
  - Todos los atributos son atómicos:
    - dni: cadena de 8 caracteres
    - nombre: cadena de texto (sin listas ni multivalores)
    - apellidos: cadena de texto
    - clave_digital: carácter fijo de 6
    - serie_tarjeta: cadena de texto
  - No hay grupos repetitivos ni atributos con múltiples valores en una misma columna.
    ⇒ Cumple 1FN.
- 2FN (Segunda Forma Normal)
  - La clave primaria es un único atributo (dni).

- Cualquier atributo no clave (nombre, apellidos, clave_digital, serie_tarjeta) depende directamente del dni.
- No existe una clave compuesta, por lo que no pueden darse dependencias parciales.
  ⇒ Cumple 2FN.

- 3FN (Tercera Forma Normal)
  - Verificamos que, además de los requisitos de 2FN, no haya dependencias transitivas:
    - Ningún atributo no clave depende de otro atributo no clave.
    - Por ejemplo, no existe un atributo "Tarjeta" que dependa de "serie_tarjeta" u otro atributo; la única dependencia existente es desde dni hacia cada uno de los demás.
      ⇒ Cumple 3FN.

3.2.2 Tabla ocupaciones_usuario(id_ocupacion [PK], dni [FK], ocupacion)

- 1FN
  - Atributos atómicos:
    - id_ocupacion: entero autoincremental
    - dni: cadena de 8 caracteres (clave foránea)
    - ocupacion: texto (por ejemplo, "Ingeniero", "Profesor", etc.)
  - No hay listas ni grupos repetitivos: cada fila enlaza un único dni con una única ocupacion.
    ⇒ Cumple 1FN.
- 2FN
  - La clave primaria es un solo atributo (id_ocupacion).
  - Los atributos no clave son dni y ocupacion. Ambos dependen de manera plena de id_ocupacion (sin división sobre una parte de clave, porque no es compuesta).
    ⇒ Cumple 2FN.
- 3FN
  - No existen dependencias transitivas:
    - Ni dni ni ocupacion dependen entre sí.
    - La única dependencia es id_ocupacion → (dni, ocupacion).
      ⇒ Cumple 3FN.

3.2.3 Tabla contacto(id_contacto [PK], dni [FK], correo, celular)

- 1FN
  - Atributos atómicos:
    - id_contacto: entero autoincremental
    - dni: cadena de 8 caracteres (clave foránea)
    - correo: texto (email)
    - celular: 9 caracteres
  - Ningún atributo almacena listas o valores múltiples.
    ⇒ Cumple 1FN.
- 2FN

- Clave primaria simple: id_contacto.
- Los atributos no clave (dni, correo, celular) dependen plenamente de id_contacto. ⇒ Cumple 2FN.

- 3FN
  - No hay dependencias transitivas:
    - Ni correo ni celular dependen uno del otro; ambos dependen únicamente de id_contacto.
    - El atributo dni es clave foránea, pero no actúa como atributo intermedio que creara dependencia entre otros no clave. ⇒ Cumple 3FN.

3.2.4 Tabla direccion_usuario(id_direccion [PK], dni [FK], departamento, provincia, distrito, direccion)

- 1FN
  - Atributos atómicos:
    - id_direccion: entero autoincremental
    - dni: 8 caracteres (clave foránea)
    - departamento, provincia, distrito, direccion: cada uno texto o varchar (sin listas internas).
  - No existen grupos repetitivos en ninguna columna. ⇒ Cumple 1FN.
- 2FN
  - Clave primaria simple: id_direccion.
  - Cada atributo no clave (dni, departamento, provincia, distrito, direccion) depende en su totalidad de id_direccion. ⇒ Cumple 2FN.
- 3FN
  - Verificar posibles dependencias transitivas:
    - Podría argumentarse que “provincia” depende de “departamento” o “distrito” depende de “provincia”, pero en este modelo cada una de esas columnas es parte de la misma entidad “dirección”. No existe un atributo no clave que dependa de otro atributo no clave en esta tabla, puesto que estamos almacenando la estructura completa de la dirección como un bloque que corresponde a un único registro de id_direccion.
    - En otras palabras, no hay un atributo como “id_provincia” que creara una dependencia adicional dentro de esta misma tabla; simplemente se tipifican las cuatro columnas textuales. ⇒ Cumple 3FN.

3.2.5 Tabla cuentas(num_cuenta [PK], dni [FK], cci [UNIQUE], saldo)

- 1FN
  - Atributos atómicos:

        - num_cuenta: cadena de 11 caracteres
        - dni: 8 caracteres (clave foránea)
        - cci: cadena de 20 caracteres (único)
        - saldo: decimal DECIMAL(12,2)
    - No hay listas ni repetición de valores en un solo campo.
      ⇒ Cumple 1FN.
- 2FN
    - Clave primaria simple: num_cuenta.
    - Los atributos no clave (dni, cci, saldo) dependen directamente de num_cuenta en su totalidad.
      ⇒ Cumple 2FN.
- 3FN
    - Revisar posibles dependencias transitivas:
        - dni es clave foránea, pero no introduce dependencia de un no clave hacia otro.
        - cci y saldo dependen únicamente de num_cuenta, sin depender entre sí.
        - No existe, por ejemplo, un atributo como "tipo de cuenta" que dependiera de "saldo" o similar.
          ⇒ Cumple 3FN.

3.2.6 Tabla historial(id_historial [PK], num_cuenta [FK], tipo_operacion, descripcion, monto, fecha)

- 1FN
    - Atributos atómicos:
        - id_historial: entero autoincremental
        - num_cuenta: varchar (11)
        - tipo_operacion: cadena (ej. "DEPÓSITO", "RETIRO", etc.)
        - descripcion: texto libre
        - monto: decimal(12,2)
        - fecha: timestamp
    - Ningún campo almacena múltiples valores ni conjuntos.
      ⇒ Cumple 1FN.
- 2FN
    - Clave primaria simple: id_historial.
    - Los atributos no clave (num_cuenta, tipo_operacion, descripcion, monto, fecha) dependen enteramente de id_historial.
      ⇒ Cumple 2FN.
- 3FN
    - No hay dependencias transitivas:
        - Ni monto ni fecha ni descripcion dependen unos de otros; todos dependen únicamente de id_historial.
        - num_cuenta es clave foránea, pero no genera una dependencia entre atributos no clave.
          ⇒ Cumple 3FN.

3.2.7 Tabla deposito(id_deposito [PK], num_cuenta [FK], monto, saldo_anterior, saldo_nuevo, fecha_hora)

- 1FN
  - Atributos atómicos:
    - id_deposito: entero autoincremental
    - num_cuenta: varchar(11)
    - monto: decimal(12,2)
    - saldo_anterior: decimal(12,2)
    - saldo_nuevo: decimal(12,2)
    - fecha_hora: datetime
  - Ningún campo guarda listas ni valores repetidos. ⇒ Cumple 1FN.
- 2FN
  - Clave primaria simple: id_deposito.
  - Todos los demás atributos dependen en su totalidad de id_deposito. ⇒ Cumple 2FN.
- 3FN
  - Verificar posibles dependencias transitivas:
    - Podría pensarse que saldo_nuevo = saldo_anterior + monto, pero esto es una regla de negocio que no implica dependencia funcional dentro de la tabla, sino un cálculo que se almacena explícitamente.
    - Ningún atributo no clave depende de otro no clave:
      - Por ejemplo, saldo_anterior no "determina" monto; ambos vienen de la misma operación.
    - Por ende, no hay transiciones de dependencia que violen 3FN. ⇒ Cumple 3FN.

3.2.8 Tabla bancos(id_banco [PK], nombre_banco)

- 1FN
  - Atributos atómicos:
    - id_banco: entero autoincremental
    - nombre_banco: texto (ej. "BBVA", "Banco de la Nación", etc.)
  - No hay listas ni campos multivaluados. ⇒ Cumple 1FN.
- 2FN
  - Clave primaria simple: id_banco.
  - nombre_banco depende únicamente de id_banco. ⇒ Cumple 2FN.
- 3FN
  - Con solo dos atributos, no pueden existir dependencias transitivas. ⇒ Cumple 3FN.

3.2.9 Tabla banco_celular(id [PK], celular, id_banco [FK])

- 1FN
  - Atributos atómicos:
    - id: entero autoincremental
    - celular: 9 caracteres (número de teléfono)
    - id_banco: entero (clave foránea)
  - No hay campos con múltiples valores.
    ⇒ Cumple 1FN.
- 2FN
  - Clave primaria simple: id.
  - Tanto celular como id_banco dependen completamente de id.
    ⇒ Cumple 2FN.
- 3FN
  - No existen dependencias transitivas:
    - celular no depende de id_banco y viceversa; ambos atributos dependen directamente de la clave id.
      ⇒ Cumple 3FN.

3.2.10 Tabla cci_afiliados(id [PK], cci [UNIQUE], id_banco [FK])

- 1FN
  - Atributos atómicos:
    - id: entero autoincremental
    - cci: varchar(20) (único)
    - id_banco: entero (clave foránea)
  - No hay colecciones ni campos repetitivos.
    ⇒ Cumple 1FN.
- 2FN
  - Clave primaria simple: id.
  - cci e id_banco dependen plenamente de id.
    ⇒ Cumple 2FN.
- 3FN
  - No hay dependencias transitivas:
    - cci no es función de id_banco y viceversa; ambos dependen sólo de id.
      ⇒ Cumple 3FN.

3.2.11 Tabla transferencias_celular(id_transferencia [PK], num_cuenta_origen [FK], celular_destino, id_banco_destino [FK], monto, fecha)

- 1FN
  - Atributos atómicos:
    - id_transferencia: entero autoincremental
    - num_cuenta_origen: varchar(11)
    - celular_destino: char(9)
    - id_banco_destino: entero
    - monto: decimal(12,2)

        - fecha: timestamp
    - No hay listas ni datos multivaluados. ⇒ Cumple 1FN.
- 2FN
    - Clave primaria simple: id_transferencia.
    - Todos los atributos no clave (num_cuenta_origen, celular_destino, id_banco_destino, monto, fecha) dependen en su totalidad de id_transferencia. ⇒ Cumple 2FN.
- 3FN
    - No existe dependencia transitiva:
        - Ni monto ni fecha dependen uno de otro; ambos dependen directamente de id_transferencia.
        - celular_destino tampoco depende de id_banco_destino; se almacenan en la misma fila sin jerarquía de dependencia entre ellos. ⇒ Cumple 3FN.

3.2.12 Tabla giros(id_giro [PK], num_cuenta_origen [FK], dni_destino, id_banco_destino [FK], monto, fecha)

- 1FN
    - Atributos atómicos:
        - id_giro: entero autoincremental
        - num_cuenta_origen: varchar(11)
        - dni_destino: char(8)
        - id_banco_destino: entero
        - monto: decimal(12,2)
        - fecha: timestamp
    - Ningún campo almacena colecciones ni múltiples valores. ⇒ Cumple 1FN.
- 2FN
    - Clave primaria simple: id_giro.
    - Los atributos no clave dependen íntegramente de id_giro. ⇒ Cumple 2FN.
- 3FN
    - No hay dependencias transitivas:
        - dni_destino y id_banco_destino no están relacionados funcionalmente entre sí; dependen directamente de la clave. ⇒ Cumple 3FN.

3.2.13 Tabla transferencias_cuentas(id_transferencia [PK], num_cuenta_origen [FK], numero_cuenta_destino, id_banco_destino [FK], descripcion, monto, fecha)

- 1FN
    - Atributos atómicos:
        - id_transferencia: entero autoincremental
        - num_cuenta_origen: varchar(11)
        - numero_cuenta_destino: varchar(13)

        - id_banco_destino: entero
        - descripcion: texto libre
        - monto: decimal(12,2)
        - fecha: timestamp
    - No se almacenan listas ni conjuntos en ningún atributo.
      ⇒ Cumple 1FN.
- 2FN
    - Clave primaria simple: id_transferencia.
    - Los atributos no clave (num_cuenta_origen, numero_cuenta_destino, id_banco_destino, descripcion, monto, fecha) dependen enteramente de id_transferencia.
      ⇒ Cumple 2FN.
- 3FN
    - No existen dependencias transitivas:
        - numero_cuenta_destino no depende de id_banco_destino o de descripcion, y viceversa.
        - Todos los atributos no clave se relacionan directamente con la clave primaria.
          ⇒ Cumple 3FN.

3.2.14 Tabla transferencias_cci(id_transferencia [PK], num_cuenta_origen [FK], destino, id_banco_destino [FK], descripcion, monto, fecha)

- 1FN
    - Atributos atómicos:
        - id_transferencia: entero autoincremental
        - num_cuenta_origen: varchar(11)
        - destino: varchar(20) (CCI del destinatario)
        - id_banco_destino: entero
        - descripcion: texto libre
        - monto: decimal(12,2)
        - fecha: timestamp
    - No hay atributos con múltiples valores.
      ⇒ Cumple 1FN.
- 2FN
    - Clave primaria simple: id_transferencia.
    - Cada uno de los demás campos depende únicamente de id_transferencia.
      ⇒ Cumple 2FN.
- 3FN
    - No hay dependencias transitivas:
        - destino no depende de id_banco_destino y viceversa.
        - Todos los atributos no clave dependen directamente de la clave.
          ⇒ Cumple 3FN.

3.2.15 Tabla empresas_servicios(id_empresa [PK], nombre_empresa, tipo_servicio)

- 1FN
  - Atributos atómicos:
    - id_empresa: entero autoincremental
    - nombre_empresa: texto (ej. “Sedapal”, “Enel”, etc.)
    - tipo_servicio: ENUM('AGUA','LUZ','TELEFONIA')
  - Ningún atributo almacena listas ni valores múltiples. ⇒ Cumple 1FN.
- 2FN
  - Clave primaria simple: id_empresa.
  - Los atributos nombre_empresa y tipo_servicio dependen en su totalidad de id_empresa. ⇒ Cumple 2FN.
- 3FN
  - No existen atributos no clave que dependan de otro atributo no clave:
    - tipo_servicio no depende de nombre_empresa, simplemente ambos dependen de la clave. ⇒ Cumple 3FN.

3.2.16 Tabla deudas_servicios(id_deuda [PK], tipo_servicio, id_empresa [FK], codigo_pago, monto)

- 1FN (Primera Forma Normal)
  - Atributos atómicos:
    - id_deuda: entero autoincremental
    - tipo_servicio: ENUM('AGUA','LUZ','TELEFONIA')
    - id_empresa: entero (clave foránea)
    - codigo_pago: VARCHAR(50)
    - monto: DECIMAL(12,2)
  - No hay valores multivaluados ni listas en ninguna columna. ⇒ Cumple 1FN.
- 2FN (Segunda Forma Normal)
  - Clave primaria simple: id_deuda.
  - Todos los atributos no clave (tipo_servicio, id_empresa, codigo_pago, monto) dependen en su totalidad de id_deuda. ⇒ Cumple 2FN.
- 3FN (Tercera Forma Normal)
  - No existen dependencias transitivas:
    - tipo_servicio no depende de id_empresa ni de otro atributo no clave; todos los atributos no clave dependen directamente de id_deuda. ⇒ Cumple 3FN.

3.2.17 Tabla pago_servicios(id_pago [PK], num_cuenta [FK], id_deuda [FK], fecha_pago)

- 1FN
  - Atributos atómicos:
    - id_pago: entero autoincremental
    - num_cuenta: varchar(11)

        - id_deuda: entero
        - fecha_pago: timestamp
    - No hay colecciones ni repetición de valores en un mismo campo. ⇒ Cumple 1FN.
- 2FN
    - Clave primaria simple: id_pago.
    - Cada atributo no clave (num_cuenta, id_deuda, fecha_pago) depende directamente de id_pago.
      ⇒ Cumple 2FN.
- 3FN
    - No hay dependencias transitivas:
        - num_cuenta no determina id_deuda ni viceversa; ambos se refieren a tablas externas (cuentas y deudas_servicios).
        - fecha_pago solo depende de la transacción (clave primaria). ⇒ Cumple 3FN.

3.2.18 Tabla empresas_recargas(id_empresa [PK], nombre_empresa)

- 1FN
    - Atributos atómicos:
        - id_empresa: entero autoincremental
        - nombre_empresa: texto (por ejemplo, “Claro”, “Movistar”, etc.)
    - Ningún valor repetitivo o lista interna. ⇒ Cumple 1FN.
- 2FN
    - Clave primaria simple: id_empresa.
    - nombre_empresa depende únicamente de id_empresa. ⇒ Cumple 2FN.
- 3FN
    - Con solo dos atributos, no hay espacio para dependencia transitiva. ⇒ Cumple 3FN.

Tabla recargas(id_recarga [PK], num_cuenta [FK], id_empresa [FK], numero, monto, fecha_recarga)

- 1FN
    - Atributos atómicos:
        - id_recarga: entero autoincremental
        - num_cuenta: varchar(11)
        - id_empresa: entero
        - numero: char(9) (número de celular a recargar)
        - monto: decimal(12,2)
        - fecha_recarga: timestamp

    - No existen valores multivaluados ni repetitivos en un mismo campo. ⇒ Cumple 1FN.
- 2FN
    - Clave primaria simple: id_recarga.
    - Los demás atributos dependen íntegramente de id_recarga. ⇒ Cumple 2FN.
- 3FN
    - No se presentan dependencias transitivas:
        - numero no depende de id_empresa ni de monto; todos dependen directamente de la clave. ⇒ Cumple 3FN.

3.2.19 Tabla ubicaciones_banco(id_ubicacion [PK], departamento, provincia, direccion)

- 1FN
    - Atributos atómicos:
        - id_ubicacion: entero autoincremental
        - departamento: texto
        - provincia: texto
        - direccion: texto libre (calle, número)
    - No hay listas ni valores repetidos dentro de una sola columna. ⇒ Cumple 1FN.
- 2FN
    - Clave primaria simple: id_ubicacion.
    - Los atributos departamento, provincia y direccion dependen completamente de id_ubicacion. ⇒ Cumple 2FN.
- 3FN
    - No hay un atributo no clave que dependa de otro no clave dentro de esta misma tabla. ⇒ Cumple 3FN.

3.2.20 Tabla ubicacion_usuario(id [PK], dni [FK], id_ubicacion [FK])

- 1FN
    - Atributos atómicos:
        - id: entero autoincremental
        - dni: varchar(8)
        - id_ubicacion: entero
    - Ningún campo tiene múltiples valores o listas. ⇒ Cumple 1FN.
- 2FN
    - Clave primaria simple: id.
    - dni e id_ubicacion dependen enteramente de id. ⇒ Cumple 2FN.

- 3FN
    - No hay dependencias transitivas:
        - dni no determina id_ubicacion ni viceversa.
        - Ambos dependen directamente de la clave primaria. ⇒ Cumple 3FN.

Síntesis de la validación 3FN: Todas las tablas cumplen con la validacion de la 3FN

3.3 Diagrama de Entidad Relacion:

Disponible en el Github: https://github.com/Therick75/BD_BN.git

4 Implementación

4.1 Esquema de la Base de Datos propuesta (SQL-LDD), usuarios y privilegios (create tables, create views, create users, grants

```
-- Base de datos

DROP DATABASE IF EXISTS banco_nacion;
CREATE DATABASE banco_nacion;
USE banco_nacion;

-- 1. Usuarios
CREATE TABLE usuarios (
   dni VARCHAR(8) PRIMARY KEY,
   nombre VARCHAR(100),
   apellidos varchar(50),
   clave_digital CHAR(6),
   serie_tarjeta VARCHAR(50)
);
CREATE TABLE ocupaciones_usuario (
   id_ocupacion INT PRIMARY KEY AUTO_INCREMENT,
   dni VARCHAR(8),
   ocupacion VARCHAR(100),
   FOREIGN KEY (dni) REFERENCES usuarios(dni)
);
CREATE TABLE contacto (
   id_contacto INT PRIMARY KEY AUTO_INCREMENT,
   dni VARCHAR(8),
   correo VARCHAR(100),
   celular CHAR(9),
   FOREIGN KEY (dni) REFERENCES usuarios(dni)
);
CREATE TABLE direccion_usuario (
   id_direccion INT PRIMARY KEY AUTO_INCREMENT,
   dni VARCHAR(8),
   departamento VARCHAR(100),
```

```
    provincia VARCHAR(100),
    distrito VARCHAR(100),
    direccion VARCHAR(255),
    FOREIGN KEY (dni) REFERENCES usuarios(dni)
);
-- 2. Cuentas
CREATE TABLE cuentas (
    num_cuenta VARCHAR(11) PRIMARY KEY ,
    dni VARCHAR(8),
    cci VARCHAR(20) UNIQUE,
    saldo DECIMAL(12,2) DEFAULT 0.00,
    FOREIGN KEY (dni) REFERENCES usuarios(dni)
);

-- 3. Historial de operaciones
CREATE TABLE historial (
    id_historial INT PRIMARY KEY AUTO_INCREMENT,
    num_cuenta VARCHAR(11),
    tipo_operacion varchar(30),
    descripcion TEXT,
    monto DECIMAL(12,2),
    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (num_cuenta) REFERENCES cuentas(num_cuenta)
);

-- 4. Bancos
CREATE TABLE bancos (
    id_banco INT PRIMARY KEY AUTO_INCREMENT,
    nombre_banco VARCHAR(100) NOT NULL
);

-- 5. Banco-Celular
CREATE TABLE banco_celular (
    id INT PRIMARY KEY AUTO_INCREMENT,
    celular CHAR(9),
    id_banco INT,
    FOREIGN KEY (id_banco) REFERENCES bancos(id_banco)
);

-- 6. CCI afiliados
CREATE TABLE cci_afiliados (
    id INT PRIMARY KEY AUTO_INCREMENT,
    cci VARCHAR(20) UNIQUE,
    id_banco INT,
    FOREIGN KEY (id_banco) REFERENCES bancos(id_banco)
);

-- 7. Transferencias por celular
```

```
CREATE TABLE transferencias_celular (
   id_transferencia INT PRIMARY KEY AUTO_INCREMENT,
   num_cuenta_origen VARCHAR(11),
   celular_destino CHAR(9),
   id_banco_destino INT,
   monto DECIMAL(12,2),
   fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (num_cuenta_origen) REFERENCES cuentas(num_cuenta),
   FOREIGN KEY (id_banco_destino) REFERENCES bancos(id_banco)
);

-- 8. Giros
CREATE TABLE giros (
   id_giro INT PRIMARY KEY AUTO_INCREMENT,
   num_cuenta_origen VARCHAR(11),
   dni_destino CHAR(8),
   id_banco_destino INT,
   monto DECIMAL(12,2),
   fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (num_cuenta_origen) REFERENCES cuentas(num_cuenta),
   FOREIGN KEY (id_banco_destino) REFERENCES bancos(id_banco)
);

-- 9. Transferencias por cuenta
CREATE TABLE transferencias_cuentas (
   id_transferencia INT PRIMARY KEY AUTO_INCREMENT,
   num_cuenta_origen VARCHAR(11),
   numero_cuenta_destino VARCHAR(13),  -- solo para cuentas internas
   id_banco_destino INT,
   descripcion text,
   monto DECIMAL(12,2),
   fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (num_cuenta_origen) REFERENCES cuentas(num_cuenta),
   FOREIGN KEY (id_banco_destino) REFERENCES bancos(id_banco)
);

CREATE TABLE transferencias_cci (
   id_transferencia INT PRIMARY KEY AUTO_INCREMENT,
   num_cuenta_origen VARCHAR(11),
   destino VARCHAR(20),
   id_banco_destino INT,
   descripcion text,
   monto DECIMAL(12,2),
   fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (num_cuenta_origen) REFERENCES cuentas(num_cuenta),
   FOREIGN KEY (id_banco_destino) REFERENCES bancos(id_banco)
);
```

```
-- 10. Empresas de servicios (agua, luz, telefonía)
CREATE TABLE empresas_servicios (
   id_empresa INT PRIMARY KEY AUTO_INCREMENT,
   nombre_empresa VARCHAR(100) NOT NULL,
   tipo_servicio ENUM('AGUA', 'LUZ', 'TELEFONIA') NOT NULL
);

-- 11. Deudas de servicios unificadas
CREATE TABLE deudas_servicios (
   id_deuda INT PRIMARY KEY AUTO_INCREMENT,
   id_empresa INT NOT NULL,
   monto DECIMAL(12,2),
   FOREIGN KEY (id_empresa) REFERENCES empresas_servicios(id_empresa)
);

-- 12. Pagos de servicios
CREATE TABLE pago_servicios (
   id_pago INT PRIMARY KEY AUTO_INCREMENT,
   num_cuenta VARCHAR(11) NOT NULL,
   id_deuda INT NOT NULL,
   fecha_pago TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (num_cuenta) REFERENCES cuentas(num_cuenta),
   FOREIGN KEY (id_deuda) REFERENCES deudas_servicios(id_deuda)
);
CREATE TABLE empresas_recargas (
   id_empresa INT PRIMARY KEY AUTO_INCREMENT,
   nombre_empresa VARCHAR(100) NOT NULL
);
-- 14. Recargas
CREATE TABLE recargas (
   id_recarga INT PRIMARY KEY AUTO_INCREMENT,
   num_cuenta VARCHAR(11),
   id_empresa INT,
   numero CHAR(9),
   monto DECIMAL(12,2),
   fecha_recarga TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (num_cuenta) REFERENCES cuentas(num_cuenta),
   FOREIGN KEY (id_empresa) REFERENCES empresas_recargas(id_empresa)
);

-- 15. Ubicación de bancos
CREATE TABLE ubicaciones_banco (
   id_ubicacion INT PRIMARY KEY AUTO_INCREMENT,
   departamento varchar(20),
   distrito VARCHAR(100),
   direccion VARCHAR(255)
);
```

```
-- 16. Ubicación por usuario
CREATE TABLE ubicacion_usuario (
   id INT PRIMARY KEY AUTO_INCREMENT,
   dni VARCHAR(8),
   id_ubicacion INT,
   FOREIGN KEY (dni) REFERENCES usuarios(dni),
   FOREIGN KEY (id_ubicacion) REFERENCES ubicaciones_banco(id_ubicacion)
);
CREATE TABLE IF NOT EXISTS deposito (
   id_deposito INT PRIMARY KEY AUTO_INCREMENT,
   num_cuenta VARCHAR(11),
   monto DECIMAL(12,2),
   saldo_anterior DECIMAL(12,2),
   saldo_nuevo DECIMAL(12,2),
   fecha_hora DATETIME
);
CREATE TABLE retiro (
   id_retiro INT PRIMARY KEY AUTO_INCREMENT,
   num_cuenta VARCHAR(11),
   monto DECIMAL(12,2),
   fecha_hora DATETIME DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (num_cuenta) REFERENCES cuentas(num_cuenta)
);
```

4.2 Creación de objetos de seguridad: usuarios y privilegios

```
-- 4.2. Creación de usuarios y asignación de privilegios

-- CREATE USER 'cliente_app'@'localhost' IDENTIFIED BY 'cliente123';

-- CREATE USER 'admin_app'@'localhost' IDENTIFIED BY 'admin123';

-- Privilegios

GRANT SELECT, INSERT, UPDATE ON banco.* TO 'cliente_app'@'localhost';

GRANT ALL PRIVILEGES ON banco.* TO 'admin_app'@'localhost';
```

4.3 Creación de índices

```
-- 4.3. Creación de índices

CREATE INDEX idx_bancos_name ON bancos(nombre_banco);

CREATE INDEX idx_empresas_tipo ON empresas_servicios(nombre_empresa, tipo_servicio);

CREATE INDEX idx_ubicacion ON ubicaciones_banco(distrito, direccion);
```

4.4 Creación de objetos programados: funciones, disparadores y procedimientos almacenados.

```
-- funciones
delimiter //
CREATE FUNCTION calcular_comision_transferencia(
   p_id_banco_origen INT,
   p_id_banco_destino INT,
   p_monto DECIMAL(12,2)
)
RETURNS DECIMAL(12,2)
DETERMINISTIC
BEGIN
   DECLARE tarifa DECIMAL(12,2);
   IF p_id_banco_origen = p_id_banco_destino THEN
      SET tarifa = 0.00;
   ELSE
      SET tarifa = ROUND(p_monto * 0.005, 2);
      IF tarifa < 1.00 THEN
         SET tarifa = 1.00;
      ELSEIF tarifa > 25.00 THEN
         SET tarifa = 25.00;
      END IF;
   END IF;
   RETURN tarifa;
END//
delimiter ;
delimiter //
CREATE FUNCTION calcular_nuevo_saldo(
   p_num_cuenta VARCHAR(11),
   p_monto DECIMAL(12,2)
)
RETURNS DECIMAL(12,2)
DETERMINISTIC
BEGIN
   DECLARE saldo_actual DECIMAL(12,2);
   SELECT saldo INTO saldo_actual
     FROM cuentas
    WHERE num_cuenta = p_num_cuenta;
   IF saldo_actual IS NULL THEN
      RETURN NULL; -- Cuenta no existe
   END IF;
   RETURN ROUND(saldo_actual + p_monto, 2);
END//
delimiter ;
delimiter //
CREATE FUNCTION obtener_saldo_actual(p_num_cuenta VARCHAR(11))
RETURNS DECIMAL(12,2)
```

```
DETERMINISTIC
BEGIN
   DECLARE s DECIMAL(12,2);
   SELECT saldo INTO s
     FROM cuentas
    WHERE num_cuenta = p_num_cuenta;
   RETURN s;
END//
delimiter ;
delimiter //
CREATE FUNCTION formatear_numero_cuenta(p_num_cuenta VARCHAR(11))
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
   DECLARE prefijo CHAR(2);
   DECLARE medio CHAR(3);
   DECLARE sufijo CHAR(6);
   IF LENGTH(p_num_cuenta) <> 11 THEN
      RETURN NULL; -- Formato incorrecto
   END IF;
   SET prefijo = LEFT(p_num_cuenta, 2);
   SET medio   = MID(p_num_cuenta, 3, 3);
   SET sufijo  = RIGHT(p_num_cuenta, 6);
   RETURN CONCAT(prefijo, '-', medio, '-', sufijo);
END//
delimiter ;
delimiter //
CREATE FUNCTION tiene_saldo_suficiente(
   p_num_cuenta VARCHAR(11),
   p_id_banco_origen INT,
   p_id_banco_destino INT,
   p_monto DECIMAL(12,2)
)
RETURNS bool
DETERMINISTIC
BEGIN
   DECLARE saldo_actual DECIMAL(12,2);
   DECLARE comision DECIMAL(12,2);
   -- Obtener el saldo actual
   SELECT saldo INTO saldo_actual
     FROM cuentas
    WHERE num_cuenta = p_num_cuenta;
   IF saldo_actual IS NULL THEN
      RETURN NULL; /* cuenta no existe */
   END IF;
   -- Calcular comisión según banco origen/destino
   SET comision = calcular_comision_transferencia(p_id_banco_origen, p_id_banco_destino,
p_monto);
```

```
    IF saldo_actual >= (p_monto + comision) THEN
        RETURN 1;
    ELSE
        RETURN 0;
    END IF;
END//
delimiter ;
DELIMITER //

CREATE FUNCTION existe_cuenta(p_num_cuenta VARCHAR(11))
RETURNS BOOLEAN
DETERMINISTIC
BEGIN
    DECLARE v_existe INT;

    SELECT COUNT(*) INTO v_existe
    FROM cuentas
    WHERE num_cuenta = p_num_cuenta;

    RETURN v_existe > 0;
END//

DELIMITER //

CREATE FUNCTION obtener_banco_por_cci(p_cci VARCHAR(20))
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE v_id_banco INT;
    SELECT id_banco
    INTO v_id_banco
    FROM cci_afiliados
    WHERE cci = p_cci
    LIMIT 1;

    RETURN v_id_banco;
END//
DELIMITER ;
DELIMITER //
CREATE FUNCTION obtener_banco_por_celular(p_celular CHAR(9))
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE v_id_banco INT;
    SELECT id_banco
    INTO v_id_banco
    FROM banco_celular
    WHERE celular = p_celular
```

```
    LIMIT 1;
    RETURN v_id_banco;
END//
DELIMITER ;
-- ------------------------------------------------------
-- procedimientos
DELIMITER //
CREATE PROCEDURE login_usuario(
    IN p_dni VARCHAR(8),
    IN p_clave CHAR(6)
)
BEGIN
    IF EXISTS (
        SELECT 1 FROM usuarios
        WHERE dni = p_dni AND clave_digital = p_clave
    ) THEN
        SELECT 'Login exitoso' AS mensaje;
    ELSE
        SELECT 'Credenciales incorrectas' AS mensaje;
    END IF;
END//
DELIMITER ;
--  ------------------------------------------------------------------------------------------------------------------------
--------------
DELIMITER //
CREATE PROCEDURE ver_saldo(
    IN p_dni VARCHAR(8)
)
BEGIN
    SELECT num_cuenta, saldo
    FROM cuentas
    WHERE dni = p_dni;
END//
DELIMITER ;
-- ------------------------------------------------------------------------
DELIMITER //

CREATE PROCEDURE transferir_por_celular(
  IN p_cuenta_origen VARCHAR(11),
  IN p_celular_destino CHAR(9),
  IN p_monto DECIMAL(12,2)
)
BEGIN

  DECLARE id_banco_destino INT;
  -- Manejo de errores
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
```

```
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error en la transferencia por celular';
  END;
  START TRANSACTION;
  SET id_banco_destino = obtener_banco_por_celular(p_celular_destino);
  IF id_banco_destino IS NULL THEN
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El banco del celular destino no está afiliado.';
  END IF;
  IF tiene_saldo_suficiente(p_cuenta_origen, 1, id_banco_destino, p_monto) THEN
    INSERT INTO transferencias_celular (num_cuenta_origen, celular_destino, id_banco_destino, monto)
    VALUES (p_cuenta_origen, p_celular_destino, id_banco_destino, p_monto);
    -- Descontar saldo
    UPDATE cuentas SET saldo = saldo - p_monto
    WHERE num_cuenta = p_cuenta_origen;
    COMMIT;
  ELSE
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para transferencia por celular';
  END IF;
END//
DELIMITER ;
-- ------------------------------------------------------------------------
DELIMITER //

CREATE PROCEDURE transferir_cuenta_a_cuenta(
  IN p_cuenta_origen VARCHAR(11),
  IN p_cuenta_destino VARCHAR(11),
  IN p_monto DECIMAL(12,2),
  IN p_descripcion TEXT
)
BEGIN
  DECLARE id_banco_origen INT;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error en la transferencia cuenta a cuenta';
  END;
  START TRANSACTION;
  IF tiene_saldo_suficiente(p_cuenta_origen, 1, 1, p_monto) THEN
    INSERT INTO transferencias_cuentas (num_cuenta_origen, numero_cuenta_destino, id_banco_destino, descripcion, monto)
    VALUES (p_cuenta_origen, p_cuenta_destino, 1, p_descripcion, p_monto);
    UPDATE cuentas SET saldo = saldo - p_monto WHERE num_cuenta = p_cuenta_origen;
    UPDATE cuentas SET saldo = saldo + p_monto WHERE num_cuenta = p_cuenta_destino;
```

```
    COMMIT;
  ELSE
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para transferencia cuenta a cuenta';
  END IF;
END//
DELIMITER //

CREATE PROCEDURE transferir_cci(
  IN p_cuenta_origen VARCHAR(11),
  IN p_destino_cci VARCHAR(20),
  IN p_monto DECIMAL(12,2),
  IN p_descripcion TEXT
)
BEGIN
  DECLARE id_banco_destino INT;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error en la transferencia CCI';
  END;
  START TRANSACTION;
  -- Obtener banco de destino usando la función
  SET id_banco_destino = obtener_banco_por_cci(p_destino_cci);
  IF id_banco_destino IS NULL THEN
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'CCI de destino inválido';
  END IF;
  -- Verificar si tiene saldo suficiente
  IF tiene_saldo_suficiente(p_cuenta_origen, 1, id_banco_destino, p_monto) THEN
    INSERT INTO transferencias_cci (num_cuenta_origen, destino, id_banco_destino, descripcion, monto)
    VALUES (p_cuenta_origen, p_destino_cci, id_banco_destino, p_descripcion, p_monto);
    UPDATE cuentas
    SET saldo = saldo - (p_monto + calcular_comision_transferencia(1, id_banco_destino, p_monto))
    WHERE num_cuenta = p_cuenta_origen;
    COMMIT;
  ELSE
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para transferencia CCI';
  END IF;
END//
DELIMITER ;

-- ------------------------------------------------------------------------------------
DELIMITER //
CREATE PROCEDURE hacer_recarga(
```

```
    IN p_cuenta VARCHAR(11),
    IN p_id_empresa INT,
    IN p_numero CHAR(9),
    IN p_monto DECIMAL(12,2)
)
BEGIN
    DECLARE saldo_actual DECIMAL(12,2);
    SELECT saldo INTO saldo_actual FROM cuentas WHERE num_cuenta = p_cuenta;
    IF saldo_actual >= p_monto THEN
        -- Insertar recarga
        INSERT INTO recargas (num_cuenta, id_empresa, numero, monto)
        VALUES (p_cuenta, p_id_empresa, p_numero, p_monto);
        -- Actualizar saldo
        UPDATE cuentas SET saldo = saldo - p_monto WHERE num_cuenta = p_cuenta;
        -- Insertar en historial
        INSERT INTO historial (num_cuenta, tipo_operacion, descripcion, monto)
        VALUES (p_cuenta, 'Recarga', CONCAT('Recarga a ', p_numero), p_monto);
    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para recarga';
    END IF;
END//
-- ------------------------------------------------------------------------------------
DELIMITER //

CREATE PROCEDURE pagar_servicio(
    IN p_cuenta VARCHAR(11),
    IN p_id_deuda INT
)
BEGIN
    DECLARE monto_a_pagar DECIMAL(12,2);
    SELECT monto INTO monto_a_pagar FROM deudas_servicios WHERE id_deuda = p_id_deuda;
    IF (SELECT saldo FROM cuentas WHERE num_cuenta = p_cuenta) >= monto_a_pagar THEN
        -- Registrar pago
        INSERT INTO pago_servicios (num_cuenta, id_deuda)
        VALUES (p_cuenta, p_id_deuda);
        -- Actualizar saldo
        UPDATE cuentas SET saldo = saldo - monto_a_pagar WHERE num_cuenta = p_cuenta;
    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para pago de servicio';
    END IF;
END//
DELIMITER ;

DELIMITER //

CREATE PROCEDURE realizar_giro(
  IN p_cuenta_origen VARCHAR(11),
  IN p_dni_destino CHAR(8),
```

```
  IN p_monto DECIMAL(12,2)
)
BEGIN
  DECLARE id_banco_origen INT;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error al realizar el giro';
  END;
  START TRANSACTION;
  IF tiene_saldo_suficiente(p_cuenta_origen, 1, 1, p_monto) THEN
    -- Insertamos el giro
    INSERT INTO giros (num_cuenta_origen, dni_destino, monto)
    VALUES (p_cuenta_origen, p_dni_destino, p_monto);
    UPDATE cuentas SET saldo = saldo - p_monto WHERE num_cuenta = p_cuenta_origen;
    COMMIT;
  ELSE
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para realizar el giro';
  END IF;
END//
DELIMITER ;
DELIMITER //
CREATE PROCEDURE registrar_usuario_completo(
    IN p_dni VARCHAR(8),
    IN p_nombre VARCHAR(100),
    IN p_apellidos VARCHAR(50),
    IN p_clave_digital CHAR(6),
    IN p_serie_tarjeta VARCHAR(50),
    IN p_ocupacion VARCHAR(100),
    IN p_correo VARCHAR(100),
    IN p_celular CHAR(9),
    IN p_departamento VARCHAR(100),
    IN p_provincia VARCHAR(100),
    IN p_distrito VARCHAR(100),
    IN p_direccion VARCHAR(255)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error al registrar el usuario';
    END;
    START TRANSACTION;
    -- Insertar en la tabla usuarios
    INSERT INTO usuarios (dni, nombre, apellidos, clave_digital, serie_tarjeta)
    VALUES (p_dni, p_nombre, p_apellidos, p_clave_digital, p_serie_tarjeta);
    -- Insertar ocupación
```

```
    INSERT INTO ocupaciones_usuario (dni, ocupacion)
    VALUES (p_dni, p_ocupacion);
    -- Insertar contacto
    INSERT INTO contacto (dni, correo, celular)
    VALUES (p_dni, p_correo, p_celular);
    -- Insertar dirección
    INSERT INTO direccion_usuario (dni, departamento, provincia, distrito, direccion)
    VALUES (p_dni, p_departamento, p_provincia, p_distrito, p_direccion);
    COMMIT;
END//
DELIMITER ;
DELIMITER //
CREATE PROCEDURE sp_deposito (
    IN p_num_cuenta VARCHAR(11),
    IN p_monto DECIMAL(12,2)
)
BEGIN
    DECLARE v_saldo_anterior DECIMAL(12,2);
    DECLARE v_saldo_nuevo DECIMAL(12,2);
    IF NOT EXISTS (SELECT 1 FROM cuentas WHERE num_cuenta = p_num_cuenta) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cuenta no existe';
    END IF;
    SELECT saldo INTO v_saldo_anterior FROM cuentas WHERE num_cuenta = p_num_cuenta;
    SET v_saldo_nuevo = v_saldo_anterior + p_monto;
    UPDATE cuentas SET saldo = v_saldo_nuevo WHERE num_cuenta = p_num_cuenta;
END//
DELIMITER ;
DELIMITER //

CREATE PROCEDURE realizar_retiro(
    IN p_num_cuenta VARCHAR(11),
    IN p_monto DECIMAL(12,2)
)
BEGIN
    DECLARE v_saldo_actual DECIMAL(12,2);
    -- Manejo de errores
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error al realizar el retiro';
    END;
    START TRANSACTION;
    -- Verificar si la cuenta existe
    IF NOT existe_cuenta(p_num_cuenta) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La cuenta no existe';
    END IF;
    -- Obtener saldo actual
    SELECT saldo INTO v_saldo_actual FROM cuentas WHERE num_cuenta = p_num_cuenta;
```

```
    -- Verificar saldo suficiente
    IF v_saldo_actual < p_monto THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para realizar el retiro';
    END IF;
    -- Actualizar saldo en la cuenta
    UPDATE cuentas SET saldo = saldo - p_monto WHERE num_cuenta = p_num_cuenta;
    -- Registrar el retiro (el trigger se encargará del historial)
    INSERT INTO retiro (num_cuenta, monto)
    VALUES (p_num_cuenta, p_monto);
    COMMIT;
END//
DELIMITER ;
-- ----------------------------------------------------------------------------
-- trigger
DELIMITER //
CREATE TRIGGER crear_cuenta_con_cci
AFTER INSERT ON usuarios
FOR EACH ROW
BEGIN
    DECLARE ultimos_6_dni CHAR(6);
    DECLARE numero_cuenta VARCHAR(20);
    DECLARE cci_generado CHAR(20);
    DECLARE codigo_entidad CHAR(3) DEFAULT '009';  -- BBVA Perú (ejemplo)
    DECLARE codigo_oficina CHAR(3) DEFAULT '661';  -- Sucursal (ejemplo)
    DECLARE parte_cuenta CHAR(12);  -- Se llenará con dni y ceros
    DECLARE digitos_control CHAR(2) DEFAULT '01'; -- Simulado
    -- Tomar últimos 6 dígitos del DNI
    SET ultimos_6_dni = RIGHT(NEW.dni, 6);
    -- Crear número de cuenta en formato: 04-000-XXXXXX
    SET numero_cuenta = CONCAT('04000', ultimos_6_dni);
    -- Parte media del CCI: aquí se usa el DNI rellenado a 12 dígitos
    SET parte_cuenta = LPAD(NEW.dni, 12, '0');
    -- Armar CCI completo (20 dígitos)
    SET cci_generado = CONCAT(codigo_entidad, codigo_oficina, parte_cuenta, digitos_control);
    -- Insertar en cuentas
    INSERT INTO cuentas (num_cuenta, dni, cci, saldo)
    VALUES (
        numero_cuenta,
        new.dni,
        cci_generado,
        0.00
    );
END//
DELIMITER ;

DELIMITER //
CREATE TRIGGER tr_historial_transferencia_celular
AFTER INSERT ON transferencias_celular
```

```
FOR EACH ROW
BEGIN
  INSERT INTO historial (num_cuenta, tipo_operacion, descripcion, monto)
  VALUES (
    NEW.num_cuenta_origen,
    'Transferencia por Celular',
    CONCAT('Transferencia a celular ', NEW.celular_destino),
    NEW.monto
  );
END//

CREATE TRIGGER tr_historial_transferencia_cuentas
AFTER INSERT ON transferencias_cuentas
FOR EACH ROW
BEGIN
  INSERT INTO historial (num_cuenta, tipo_operacion, descripcion, monto)
  VALUES (
    NEW.num_cuenta_origen,
    'Transferencia Cuenta a Cuenta',
    NEW.descripcion,
    NEW.monto
  );
END//

CREATE TRIGGER tr_historial_transferencia_cci
AFTER INSERT ON transferencias_cci
FOR EACH ROW
BEGIN
  INSERT INTO historial (num_cuenta, tipo_operacion, descripcion, monto)
  VALUES (
    NEW.num_cuenta_origen,
    'Transferencia CCI',
    NEW.descripcion,
    NEW.monto
  );
END//

CREATE TRIGGER tr_historial_recarga
AFTER INSERT ON recargas
FOR EACH ROW
BEGIN
  INSERT INTO historial (num_cuenta, tipo_operacion, descripcion, monto)
  VALUES (
    NEW.num_cuenta,
    'Recarga',
    CONCAT('Recarga a ', NEW.numero),
    NEW.monto
  );
```

```
END//

CREATE TRIGGER tr_historial_pago_servicio
AFTER INSERT ON pago_servicios
FOR EACH ROW
BEGIN
  DECLARE monto_pago DECIMAL(12,2);
  SELECT monto INTO monto_pago FROM deudas_servicios WHERE id_deuda = NEW.id_deuda;
  INSERT INTO historial (num_cuenta, tipo_operacion, descripcion, monto)
  VALUES (
    NEW.num_cuenta,
    'Pago de Servicio',
    CONCAT('Pago deuda ID ', NEW.id_deuda),
    monto_pago
  );
END//

CREATE TRIGGER tr_historial_giros
AFTER INSERT ON giros
FOR EACH ROW
BEGIN
  DECLARE monto_pago DECIMAL(12,2);
  SELECT monto INTO monto_pago FROM giros WHERE id_giro = NEW.id_giro;
  INSERT INTO historial (num_cuenta, tipo_operacion, descripcion, monto)
  VALUES (
    NEW.num_cuenta_origen,
    'Giro',
    CONCAT('Se hizo un giro a', NEW.id_giro),
    monto_pago
  );
END//
DELIMITER ;
DELIMITER //

CREATE TRIGGER trg_after_update_cuentas
AFTER UPDATE ON cuentas
FOR EACH ROW
BEGIN
    -- Solo si el saldo aumentó (depósito)
    IF NEW.saldo > OLD.saldo THEN
        INSERT INTO deposito (num_cuenta, monto, saldo_anterior, saldo_nuevo, fecha_hora)
        VALUES (NEW.num_cuenta, NEW.saldo - OLD.saldo, OLD.saldo, NEW.saldo, NOW());

        INSERT INTO historial (num_cuenta, tipo_operacion, descripcion, monto, fecha)
        VALUES (NEW.num_cuenta, 'Depósito',"CORRECTO", NEW.saldo, NOW());
    END IF;
END//
CREATE TRIGGER trg_insertar_ubicaciones_usuario
```

```
AFTER INSERT ON direccion_usuario
FOR EACH ROW
BEGIN
  INSERT INTO ubicacion_usuario (dni, id_ubicacion)
  SELECT NEW.dni, ub.id_ubicacion
  FROM ubicaciones_banco ub
  WHERE ub.departamento = NEW.departamento;
END//
DELIMITER ;
DELIMITER //
CREATE TRIGGER tr_historial_retiro
AFTER INSERT ON retiro
FOR EACH ROW
BEGIN
   -- Registrar en el historial
   INSERT INTO historial (num_cuenta, tipo_operacion, descripcion, monto)
   VALUES (
      NEW.num_cuenta,
      'Retiro',
      'Retiro',
      NEW.monto
   );
END//
DELIMITER ;
```

4.5 Operaciones realizadas bajo el esquema de transacciones

Todos los procedimientos se realizan bajo esquema de transacciones

4.6 Conjunto de sentencias SQL utilizadas para las funcionalidades con datos de la aplicación (inserts, updates, deletes, selects).

```
-- procedimientos
DELIMITER //
CREATE PROCEDURE login_usuario(
   IN p_dni VARCHAR(8),
   IN p_clave CHAR(6)
)
BEGIN
   IF EXISTS (
      SELECT 1 FROM usuarios
      WHERE dni = p_dni AND clave_digital = p_clave
   ) THEN
      SELECT 'Login exitoso' AS mensaje;
   ELSE
      SELECT 'Credenciales incorrectas' AS mensaje;
```

```
   END IF;
END//
DELIMITER ;
--  ------------------------------------------------------------------------------------------------------------
-----------------------
DELIMITER //
CREATE PROCEDURE ver_saldo(
   IN p_dni VARCHAR(8)
)
BEGIN
   SELECT num_cuenta, saldo
   FROM cuentas
   WHERE dni = p_dni;
END//
DELIMITER ;
-- ----------------------------------------------------------------------------
DELIMITER //

CREATE PROCEDURE transferir_por_celular(
  IN p_cuenta_origen VARCHAR(11),
  IN p_celular_destino CHAR(9),
  IN p_monto DECIMAL(12,2)
)
BEGIN

  DECLARE id_banco_destino INT;
  -- Manejo de errores
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error en la transferencia por
celular';
  END;
  START TRANSACTION;
  SET id_banco_destino = obtener_banco_por_celular(p_celular_destino);
  IF id_banco_destino IS NULL THEN
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El banco del celular destino no
está afiliado.';
  END IF;
  IF tiene_saldo_suficiente(p_cuenta_origen, 1, id_banco_destino, p_monto) THEN
    INSERT INTO transferencias_celular (num_cuenta_origen, celular_destino,
id_banco_destino, monto)
    VALUES (p_cuenta_origen, p_celular_destino, id_banco_destino, p_monto);
    -- Descontar saldo
    UPDATE cuentas SET saldo = saldo - p_monto
    WHERE num_cuenta = p_cuenta_origen;
    COMMIT;
```

```
  ELSE
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para transferencia por celular';
  END IF;
END//
DELIMITER ;
-- ----------------------------------------------------------------------------
DELIMITER //

CREATE PROCEDURE transferir_cuenta_a_cuenta(
  IN p_cuenta_origen VARCHAR(11),
  IN p_cuenta_destino VARCHAR(11),
  IN p_monto DECIMAL(12,2),
  IN p_descripcion TEXT
)
BEGIN
  DECLARE id_banco_origen INT;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error en la transferencia cuenta a cuenta';
  END;
  START TRANSACTION;
  IF tiene_saldo_suficiente(p_cuenta_origen, 1, 1, p_monto) THEN
    INSERT INTO transferencias_cuentas (num_cuenta_origen, numero_cuenta_destino, id_banco_destino, descripcion, monto)
    VALUES (p_cuenta_origen, p_cuenta_destino, 1, p_descripcion, p_monto);
    UPDATE cuentas SET saldo = saldo - p_monto WHERE num_cuenta = p_cuenta_origen;
    UPDATE cuentas SET saldo = saldo + p_monto WHERE num_cuenta = p_cuenta_destino;
    COMMIT;
  ELSE
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para transferencia cuenta a cuenta';
  END IF;
END//
DELIMITER //

CREATE PROCEDURE transferir_cci(
  IN p_cuenta_origen VARCHAR(11),
  IN p_destino_cci VARCHAR(20),
  IN p_monto DECIMAL(12,2),
  IN p_descripcion TEXT
)
BEGIN
  DECLARE id_banco_destino INT;
```

```
 DECLARE EXIT HANDLER FOR SQLEXCEPTION
 BEGIN
  ROLLBACK;
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error en la transferencia CCI';
 END;
 START TRANSACTION;
 -- Obtener banco de destino usando la función
 SET id_banco_destino = obtener_banco_por_cci(p_destino_cci);
 IF id_banco_destino IS NULL THEN
  ROLLBACK;
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'CCI de destino inválido';
 END IF;
 -- Verificar si tiene saldo suficiente
 IF tiene_saldo_suficiente(p_cuenta_origen, 1, id_banco_destino, p_monto) THEN
  INSERT INTO transferencias_cci (num_cuenta_origen, destino, id_banco_destino,
descripcion, monto)
  VALUES (p_cuenta_origen, p_destino_cci, id_banco_destino, p_descripcion, p_monto);
  UPDATE cuentas
  SET saldo = saldo - (p_monto + calcular_comision_transferencia(1, id_banco_destino,
p_monto))
  WHERE num_cuenta = p_cuenta_origen;
  COMMIT;
 ELSE
  ROLLBACK;
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para
transferencia CCI';
 END IF;
END//
DELIMITER ;

-- ------------------------------------------------------------------------------------
DELIMITER //
CREATE PROCEDURE hacer_recarga(
  IN p_cuenta VARCHAR(11),
  IN p_id_empresa INT,
  IN p_numero CHAR(9),
  IN p_monto DECIMAL(12,2)
)
BEGIN
  DECLARE saldo_actual DECIMAL(12,2);
  SELECT saldo INTO saldo_actual FROM cuentas WHERE num_cuenta = p_cuenta;
  IF saldo_actual >= p_monto THEN
    -- Insertar recarga
    INSERT INTO recargas (num_cuenta, id_empresa, numero, monto)
    VALUES (p_cuenta, p_id_empresa, p_numero, p_monto);
    -- Actualizar saldo
    UPDATE cuentas SET saldo = saldo - p_monto WHERE num_cuenta = p_cuenta;
    -- Insertar en historial
```

```
      INSERT INTO historial (num_cuenta, tipo_operacion, descripcion, monto)
      VALUES (p_cuenta, 'Recarga', CONCAT('Recarga a ', p_numero), p_monto);
   ELSE
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para recarga';
   END IF;
END//
-- ----------------------------------------------------------------------------------------
DELIMITER //

CREATE PROCEDURE pagar_servicio(
   IN p_cuenta VARCHAR(11),
   IN p_id_deuda INT
)
BEGIN
   DECLARE monto_a_pagar DECIMAL(12,2);
   SELECT monto INTO monto_a_pagar FROM deudas_servicios WHERE id_deuda =
p_id_deuda;
   IF (SELECT saldo FROM cuentas WHERE num_cuenta = p_cuenta) >= monto_a_pagar
THEN
      -- Registrar pago
      INSERT INTO pago_servicios (num_cuenta, id_deuda)
      VALUES (p_cuenta, p_id_deuda);
      -- Actualizar saldo
      UPDATE cuentas SET saldo = saldo - monto_a_pagar WHERE num_cuenta = p_cuenta;
   ELSE
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para pago de
servicio';
   END IF;
END//
DELIMITER ;

DELIMITER //

CREATE PROCEDURE realizar_giro(
  IN p_cuenta_origen VARCHAR(11),
  IN p_dni_destino CHAR(8),
  IN p_monto DECIMAL(12,2)
)
BEGIN
  DECLARE id_banco_origen INT;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error al realizar el giro';
  END;
  START TRANSACTION;
  IF tiene_saldo_suficiente(p_cuenta_origen, 1, 1, p_monto) THEN
    -- Insertamos el giro
```

```
    INSERT INTO giros (num_cuenta_origen, dni_destino, monto)
    VALUES (p_cuenta_origen, p_dni_destino, p_monto);
    UPDATE cuentas SET saldo = saldo - p_monto WHERE num_cuenta = p_cuenta_origen;
    COMMIT;
  ELSE
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para realizar el
giro';
  END IF;
END//
DELIMITER ;
DELIMITER //
CREATE PROCEDURE registrar_usuario_completo(
    IN p_dni VARCHAR(8),
    IN p_nombre VARCHAR(100),
    IN p_apellidos VARCHAR(50),
    IN p_clave_digital CHAR(6),
    IN p_serie_tarjeta VARCHAR(50),
    IN p_ocupacion VARCHAR(100),
    IN p_correo VARCHAR(100),
    IN p_celular CHAR(9),
    IN p_departamento VARCHAR(100),
    IN p_provincia VARCHAR(100),
    IN p_distrito VARCHAR(100),
    IN p_direccion VARCHAR(255)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error al registrar el usuario';
    END;
    START TRANSACTION;
    -- Insertar en la tabla usuarios
    INSERT INTO usuarios (dni, nombre, apellidos, clave_digital, serie_tarjeta)
    VALUES (p_dni, p_nombre, p_apellidos, p_clave_digital, p_serie_tarjeta);
    -- Insertar ocupación
    INSERT INTO ocupaciones_usuario (dni, ocupacion)
    VALUES (p_dni, p_ocupacion);
    -- Insertar contacto
    INSERT INTO contacto (dni, correo, celular)
    VALUES (p_dni, p_correo, p_celular);
    -- Insertar dirección
    INSERT INTO direccion_usuario (dni, departamento, provincia, distrito, direccion)
    VALUES (p_dni, p_departamento, p_provincia, p_distrito, p_direccion);
    COMMIT;
END//
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE sp_deposito (
   IN p_num_cuenta VARCHAR(11),
   IN p_monto DECIMAL(12,2)
)
BEGIN
   DECLARE v_saldo_anterior DECIMAL(12,2);
   DECLARE v_saldo_nuevo DECIMAL(12,2);
   IF NOT EXISTS (SELECT 1 FROM cuentas WHERE num_cuenta = p_num_cuenta) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cuenta no existe';
   END IF;
   SELECT saldo INTO v_saldo_anterior FROM cuentas WHERE num_cuenta = p_num_cuenta;
   SET v_saldo_nuevo = v_saldo_anterior + p_monto;
   UPDATE cuentas SET saldo = v_saldo_nuevo WHERE num_cuenta = p_num_cuenta;
END//
DELIMITER ;
DELIMITER //

CREATE PROCEDURE realizar_retiro(
   IN p_num_cuenta VARCHAR(11),
   IN p_monto DECIMAL(12,2)
)
BEGIN
   DECLARE v_saldo_actual DECIMAL(12,2);
   -- Manejo de errores
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error al realizar el retiro';
   END;
   START TRANSACTION;
   -- Verificar si la cuenta existe
   IF NOT existe_cuenta(p_num_cuenta) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La cuenta no existe';
   END IF;
   -- Obtener saldo actual
   SELECT saldo INTO v_saldo_actual FROM cuentas WHERE num_cuenta = p_num_cuenta;
   -- Verificar saldo suficiente
   IF v_saldo_actual < p_monto THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo insuficiente para realizar el retiro';
   END IF;
   -- Actualizar saldo en la cuenta
   UPDATE cuentas SET saldo = saldo - p_monto WHERE num_cuenta = p_num_cuenta;
   -- Registrar el retiro (el trigger se encargará del historial)
   INSERT INTO retiro (num_cuenta, monto)
```

```
    VALUES (p_num_cuenta, p_monto);
    COMMIT;
END//
DELIMITER ;
```

4.7 Conjunto de sentencias SQL que utilizan índices, transacciones, funciones, disparadores y procedimientos almacenados.

```
explain select * from ubicaciones_banco where distrito="AREQUIPA" and direccion like "%aviacion%";

explain select * from empresas_servicios where nombre_empresa="Electro Puno S.A." and tipo_servicio="AGUA";

explain select * from bancos where nombre_banco="BANCO DE LA NACION";

call transferir_cuenta_a_cuenta("04000654321","04000345678", 100, "Para tu pollo");

call transferir_cci("04000654321","60000123456789000002",100,"no se");

CALL realizar_retiro("04000654321",100);

call sp_deposito("04000654321",500.00);

CALL pagar_servicio('04000654321', 1);
```

5 Conclusiones

Se diseñó una base de datos relacional completa que simula el funcionamiento del aplicativo del Banco de la Nación, integrando correctamente las operaciones esenciales como pagos de servicios, transferencias, giros, manejo de cuentas, afiliaciones por CCI y celular, así como un historial detallado de movimientos. Se implementaron restricciones de integridad referencial, usuarios con privilegios específicos, índices para mejorar el rendimiento, y objetos programados como funciones, procedimientos almacenados y triggers para automatizar operaciones clave y garantizar la consistencia de los datos mediante transacciones seguras. Todo ello permite una estructura sólida, segura, escalable y alineada con las funcionalidades reales del sistema bancario.

6 URL donde se encuentren los scripts (1 punto)

https://github.com/Therick75/BD_BN.git