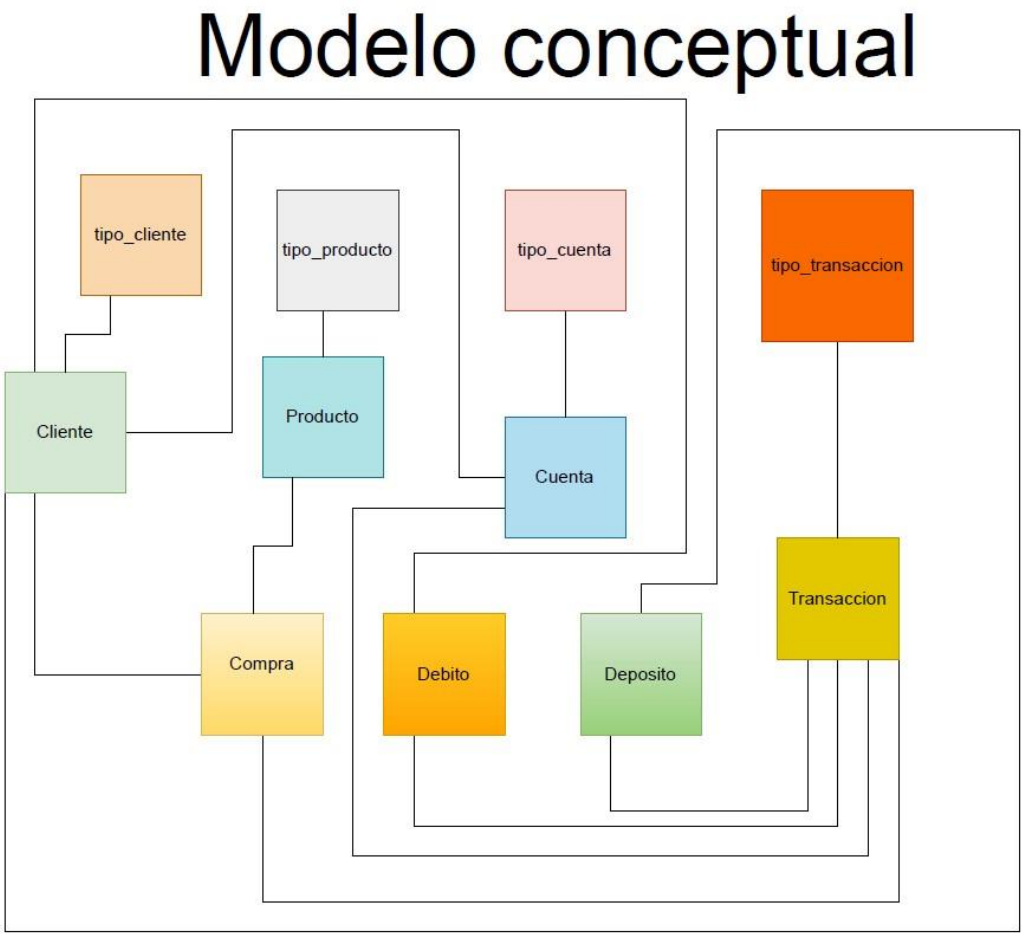
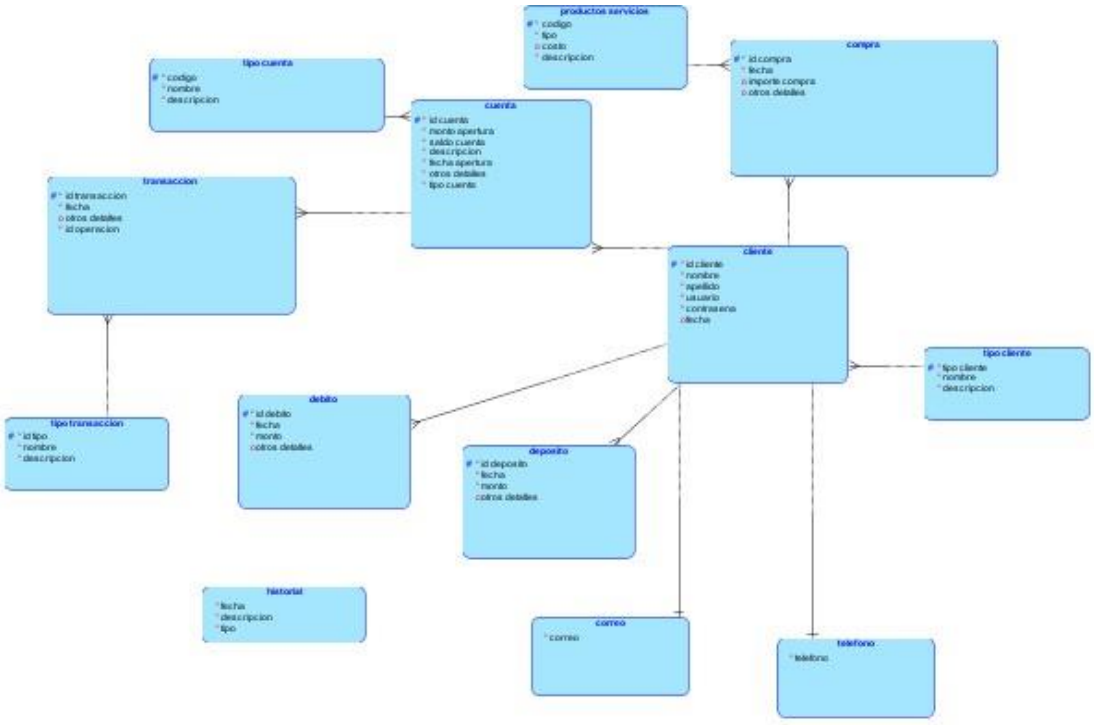
# Manual Técnico

201122881

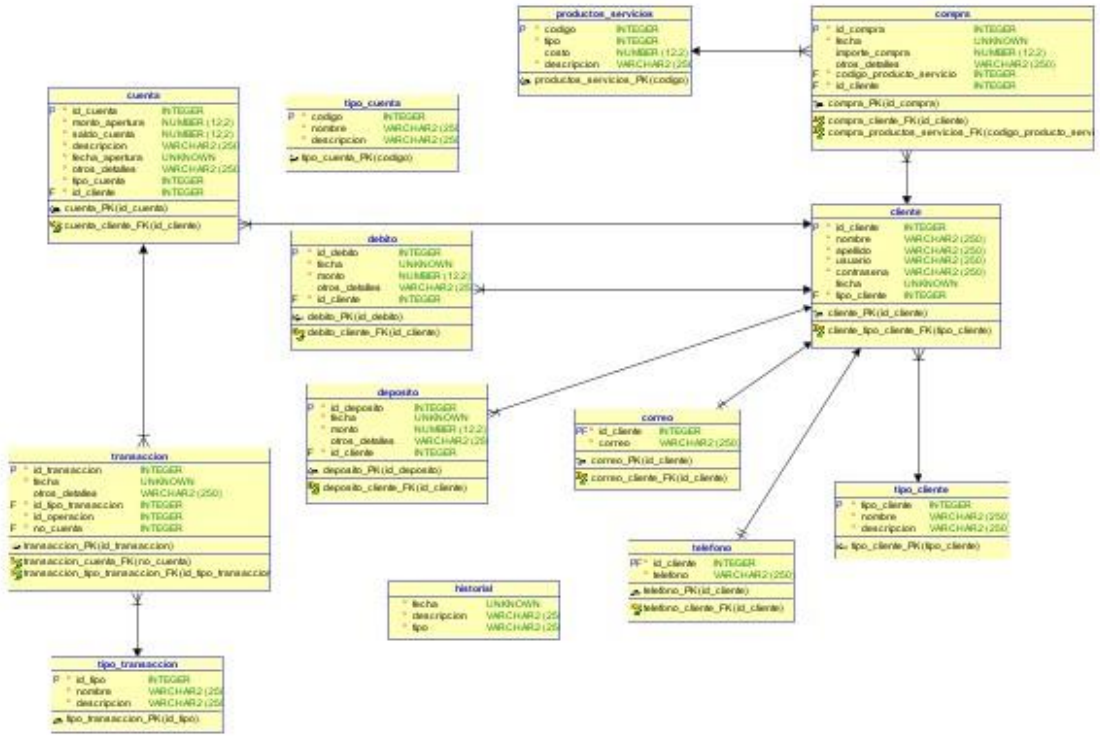
Modelo conceptual:



Modelo lógico:




Modelo físico:



Script de la base de datos:

```sql
CREATE DATABASE proyecto2bd1;


CREATE TABLE historial (
    fecha DATETIME NOT NULL,
    descripcion VARCHAR(250) NOT NULL,
    tipo VARCHAR(250) NOT NULL
);

CREATE TABLE tipo_cuenta (
    codigo INTEGER PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(250) NOT NULL,
    descripcion VARCHAR(250) NOT NULL
) AUTO_INCREMENT=1;

CREATE TABLE tipo_cliente (
    tipo_cliente INTEGER PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(250) NOT NULL,
    descripcion VARCHAR(250) NOT NULL
) AUTO_INCREMENT=1;

CREATE TABLE tipo_transaccion (
    id_tipo INTEGER PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(250) NOT NULL,
    descripcion VARCHAR(250) NOT NULL
) AUTO_INCREMENT=1;

CREATE TABLE cliente (
    id_cliente INTEGER PRIMARY KEY,
    nombre VARCHAR(250) NOT NULL,
    apellido VARCHAR(250) NOT NULL,
    usuario VARCHAR(250) NOT NULL,
    contrasena VARCHAR(250) NOT NULL,
    fecha DATETIME,
    tipo_cliente INTEGER NOT NULL,
    FOREIGN KEY (tipo_cliente) REFERENCES tipo_cliente(tipo_cliente)
);

CREATE TABLE telefono (
    id_cliente INTEGER PRIMARY KEY,
    telefono VARCHAR(250) NOT NULL,
    FOREIGN KEY (id_cliente) REFERENCES cliente(id_cliente)
);
```

```sql
CREATE TABLE correo (
    id_cliente INTEGER PRIMARY KEY,
    correo VARCHAR(250) NOT NULL,
    FOREIGN KEY (id_cliente) REFERENCES cliente(id_cliente)
);

CREATE TABLE cuenta (
    id_cuenta INTEGER PRIMARY KEY,
    monto_apertura DECIMAL(12,2) NOT NULL,
    saldo_cuenta DECIMAL(12,2) NOT NULL,
    descripcion VARCHAR(250) NOT NULL,
    fecha_apertura DATETIME NOT NULL,
    otros_detalles VARCHAR(250) NOT NULL,
    tipo_cuenta INTEGER NOT NULL,
    id_cliente INTEGER NOT NULL,
    FOREIGN KEY (id_cliente) REFERENCES cliente(id_cliente)
);

CREATE TABLE productos_servicios (
    codigo INTEGER PRIMARY KEY,
    tipo INTEGER NOT NULL,
    costo DECIMAL(12,2),
    descripcion VARCHAR(250) NOT NULL
);

CREATE TABLE compra (
    id_compra INTEGER PRIMARY KEY,
    fecha DATETIME NOT NULL,
    importe_compra DECIMAL(12,2),
    otros_detalles VARCHAR(250),
    codigo_producto_servicio INTEGER NOT NULL,
    id_cliente INTEGER NOT NULL,
    FOREIGN KEY (codigo_producto_servicio) REFERENCES
productos_servicios(codigo),
    FOREIGN KEY (id_cliente) REFERENCES cliente(id_cliente)
);

CREATE TABLE debito (
    id_debito INTEGER PRIMARY KEY,
    fecha DATETIME NOT NULL,
    monto DECIMAL(12,2) NOT NULL,
    otros_detalles VARCHAR(250),
    id_cliente INTEGER NOT NULL,
    FOREIGN KEY (id_cliente) REFERENCES cliente(id_cliente)
);
```

```sql
CREATE TABLE deposito (
    id_deposito INTEGER PRIMARY KEY,
    fecha DATETIME NOT NULL,
    monto DECIMAL(12,2) NOT NULL,
    otros_detalles VARCHAR(250),
    id_cliente INTEGER NOT NULL,
    FOREIGN KEY (id_cliente) REFERENCES cliente(id_cliente)
);

CREATE TABLE transaccion (
    id_transaccion INTEGER PRIMARY KEY AUTO_INCREMENT,
    fecha DATETIME NOT NULL,
    otros_detalles VARCHAR(250),
    id_tipo_transaccion INTEGER NOT NULL,
    id_operacion INTEGER NOT NULL,
    no_cuenta INTEGER NOT NULL,
    FOREIGN KEY (id_tipo_transaccion) REFERENCES tipo_transaccion(id_tipo),
    FOREIGN KEY (no_cuenta) REFERENCES cuenta(id_cuenta)
);
```

# Procedimientos almacenados:

**registrarTipoCliente**: Este procedimiento almacenado sirve para insertar los tipos de cliente

Realiza las validaciones correspondientes para verificar si es posible realizar la inserccion.

```sql
-- tabla tipo cliente

DELIMITER $$
CREATE PROCEDURE registrarTipoCliente(
    IN parametroIgnorado INT,
    IN nombreTipoCliente VARCHAR(250),
    IN descripcionTipoCliente VARCHAR(250)
)
BEGIN
    DECLARE nombre_existente INT DEFAULT 0;
    DECLARE descripcion_invalida INT DEFAULT 0;

    IF LENGTH(TRIM(nombreTipoCliente)) = 0 OR
LENGTH(TRIM(descripcionTipoCliente)) = 0 THEN
```

```
        SELECT 'Error: Tanto el nombre como la descripción deben tener una
longitud mayor a 0.' AS Mensaje;
    ELSE
        SELECT COUNT(*) INTO nombre_existente
        FROM tipo_cliente
        WHERE nombre = nombreTipoCliente;

        IF descripcionTipoCliente REGEXP '^[a-zA-Z ]+$' THEN
            IF nombre_existente > 0 THEN
                SELECT 'Error: El tipo de cliente ya existe.' AS Mensaje;
            ELSE
                INSERT INTO tipo_cliente (nombre, descripcion)
                VALUES (nombreTipoCliente, descripcionTipoCliente);
                SELECT 'Tipo de cliente registrado correctamente.' AS
Mensaje;
            END IF;
        ELSE
            SELECT 'Error: La descripción solo debe contener letras.' AS
Mensaje;
        END IF;
    END IF;
END$$
DELIMITER ;
```

**registrarTipoCuenta**: este procedimiento almacenado sirve para almacenar los tipos de cuenta.

Realiza las validaciones correspondientes para verificar si es posible realizar la inserccion.

```
-- tabla tipo_cliente

DELIMITER $$
CREATE PROCEDURE registrarTipoCuenta(
    IN parametroIgnorado INT,
    IN nombreTipoCuenta VARCHAR(250),
    IN descripcionTipoCuenta VARCHAR(250)
)
BEGIN
    DECLARE nombre_existente INT DEFAULT 0;

    IF LENGTH(TRIM(nombreTipoCuenta)) = 0 OR
LENGTH(TRIM(descripcionTipoCuenta)) = 0 THEN
```

```
        SELECT 'Error: Tanto el nombre como la descripción deben tener una
longitud mayor a 0.' AS Mensaje;
    ELSE
        SELECT COUNT(*) INTO nombre_existente
        FROM tipo_cuenta
        WHERE nombre = nombreTipoCuenta;

        IF nombre_existente > 0 THEN
            SELECT 'Error: El tipo de cuenta ya existe.' AS Mensaje;
        ELSE
            INSERT INTO tipo_cuenta (nombre, descripcion)
            VALUES (nombreTipoCuenta, descripcionTipoCuenta);
            SELECT 'Tipo de cuenta registrado correctamente.' AS Mensaje;
        END IF;
    END IF;
END$$
DELIMITER ;
```

**registrarTipoTransaccion**: este procedimiento almacenado sirve para almacenar el tipo de transacciones.

Realiza las validaciones correspondientes para verificar si es posible realizar la inserccion.

```
-- tabla tipo_transaccion

DELIMITER $$
CREATE PROCEDURE registrarTipoTransaccion(
    IN parametroIgnorado INT,
    IN nombreTipoTransaccion VARCHAR(250),
    IN descripcionTipoTransaccion VARCHAR(250)
)
BEGIN
    DECLARE nombre_existente INT DEFAULT 0;

    IF LENGTH(TRIM(nombreTipoTransaccion)) = 0 OR
LENGTH(TRIM(descripcionTipoTransaccion)) = 0 THEN
        SELECT 'Error: Tanto el nombre como la descripción deben tener una
longitud mayor a 0.' AS Mensaje;
    ELSE
        SELECT COUNT(*) INTO nombre_existente
        FROM tipo_transaccion
        WHERE nombre = nombreTipoTransaccion;
```

```
        IF nombre_existente > 0 THEN
            SELECT 'Error: El tipo de transacción ya existe.' AS Mensaje;
        ELSE
            INSERT INTO tipo_transaccion (nombre, descripcion)
            VALUES (nombreTipoTransaccion, descripcionTipoTransaccion);
            SELECT 'Tipo de transacción registrado correctamente.' AS
Mensaje;
        END IF;
    END IF;
END$$
DELIMITER ;
```

**registrarCliente**: Este procedimiento almacenado sirve para insertar los datos de un cliente nuevo, verifica si ya esta disponible el id de cliente, si el correo cumple con un formato valido adicional a eso, permite la llamada a un segundo procedimiento almacenado que se encarga de realizar la inserción de los teléfonos de cada cliente.

También permite que la contraseña del cliente sea guardada de forma encriptada.

```
-- tabla cliente

DELIMITER $$
CREATE PROCEDURE registrarCliente(
    IN id_cliente_sp VARCHAR(250),
    IN nombre_sp VARCHAR(250),
    IN apellido_sp VARCHAR(250),
    IN telefonos_sp VARCHAR(250),
    IN correo_electronico_sp VARCHAR(250),
    IN usuario_sp VARCHAR(250),
    IN contrasena_sp VARCHAR(250),
    IN tipo_cliente_sp INT
)
verificaciones: BEGIN
    DECLARE id_cliente_existente INT DEFAULT 0;
    DECLARE usuario_existente INT DEFAULT 0;
    DECLARE tipo_cliente_valido INT DEFAULT 0;
    DECLARE contrasena_encriptada VARCHAR(250);
    DECLARE correo_valido BOOLEAN DEFAULT FALSE;


     -- Verificar si id_cliente_sp contiene solo números

    DECLARE variable_varchar VARCHAR(250);
```

```
    SET variable_varchar = CAST(id_cliente_sp AS CHAR(250));

    IF  variable_varchar REGEXP '[a-zA-Z]+' THEN
        SELECT 'Error: El id_cliente debe contener solo números.' AS
Mensaje;
        LEAVE verificaciones;
    END IF;

    -- Validar si algún parámetro tiene longitud 0
    IF LENGTH(TRIM(nombre_sp)) = 0 OR
        LENGTH(TRIM(id_cliente_sp)) = 0 OR
        LENGTH(TRIM(tipo_cliente_sp)) = 0 OR
       LENGTH(TRIM(apellido_sp)) = 0 OR
       LENGTH(TRIM(telefonos_sp)) = 0 OR
       LENGTH(TRIM(correo_electronico_sp)) = 0 OR
       LENGTH(TRIM(usuario_sp)) = 0 OR
       LENGTH(TRIM(contrasena_sp)) = 0 THEN
        SELECT 'Error: Todos los parámetros deben tener una longitud mayor a
0.' AS Mensaje;
        LEAVE verificaciones;
    END IF;

    -- Validar si id_cliente ya existe
    SELECT COUNT(*) INTO id_cliente_existente FROM cliente WHERE id_cliente
= id_cliente_sp;
    IF id_cliente_existente > 0 THEN
        SELECT 'Error: El id_cliente ya existe.' AS Mensaje;
        LEAVE verificaciones;
    END IF;

    -- Validar si el nombre y apellido contienen solo letras
    IF NOT nombre_sp REGEXP '^[a-zA-Z]+$' OR NOT apellido_sp REGEXP '^[a-zA-
Z]+$' THEN
        SELECT 'Error: El nombre y apellido deben contener solo letras.' AS
Mensaje;
        LEAVE verificaciones;
    END IF;

    -- Validar si el usuario ya existe
    SELECT COUNT(*) INTO usuario_existente FROM cliente WHERE usuario =
usuario_sp;
    IF usuario_existente > 0 THEN
        SELECT 'Error: El usuario ya existe.' AS Mensaje;
        LEAVE verificaciones;
    END IF;
```

```
    -- Validar si el tipo_cliente es válido
    SELECT COUNT(*) INTO tipo_cliente_valido FROM tipo_cliente WHERE
tipo_cliente = tipo_cliente_sp;
    IF tipo_cliente_valido = 0 THEN
        SELECT 'Error: El tipo de cliente no es válido.' AS Mensaje;
        LEAVE verificaciones;
    END IF;

    -- Validar formato de correo electrónico
    IF correo_electronico_sp REGEXP '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-
zA-Z]{2,}$' THEN
        SET correo_valido = TRUE;
    ELSE
        SELECT 'Error: El formato de correo electrónico no es válido.' AS
Mensaje;
        LEAVE verificaciones;
    END IF;

    -- Encriptar la contraseña
    SET contrasena_encriptada = SHA2(CONCAT('rivera', contrasena_sp), 256);

    -- Insertar en la tabla cliente sólo si el correo es válido
    IF correo_valido THEN
        INSERT INTO cliente (id_cliente, nombre, apellido, usuario,
contrasena, tipo_cliente, fecha)
        VALUES (id_cliente_sp, nombre_sp, apellido_sp, usuario_sp,
contrasena_encriptada, tipo_cliente_sp, NOW());

        -- Insertar en la tabla telefono
        CALL insertarTelefonos(id_cliente_sp, telefonos_sp);

        INSERT INTO correo (id_cliente, correo) VALUES (id_cliente_sp,
correo_electronico_sp);

        SELECT 'Cliente registrado correctamente.' AS Mensaje;
    END IF;
END$$
DELIMITER ;
```

**insertarTelefonos**: este procedimiento almacenado se encarga de separar los teléfonos y almacenarlos en la tabla teléfono para cada cliente nuevo.

```
DELIMITER $$
CREATE PROCEDURE insertarTelefonos(
    IN id_cliente_sp INT,
    IN telefonos_sp VARCHAR(100)
)
BEGIN
    DECLARE telefono_start INT DEFAULT 1;
    DECLARE telefono_end INT;
    DECLARE telefono_length INT;
    DECLARE telefono VARCHAR(12);

    WHILE telefono_start <= LENGTH(telefonos_sp) DO
        SET telefono_end = IF(LOCATE('-', telefonos_sp, telefono_start) > 0,
                              LOCATE('-', telefonos_sp, telefono_start) - 1,
                              LENGTH(telefonos_sp));
        SET telefono_length = telefono_end - telefono_start + 1;
        SET telefono = SUBSTRING(telefonos_sp, telefono_start,
telefono_length);

        IF LENGTH(TRIM(telefono)) > 0 THEN
            INSERT INTO telefono (id_cliente, telefono)
            VALUES (id_cliente_sp, IF(length(telefono) > 8,
SUBSTRING(telefono, 4), telefono));
        END IF;

        SET telefono_start = telefono_end + 2; -- Avanzar al siguiente
número después del '-'
    END WHILE;
END$$
DELIMITER ;
```

**registrarCuenta**: Este procedimiento almacenado se encarga de insertar cuentas nuevas basadas en los tipos de cuentas ya previamente establecidos para diversos clientes. Realiza diferentes validaciones para poder permitir la inserción.

```
-- tabla cuenta

DELIMITER $$

CREATE PROCEDURE registrarCuenta(
    IN p_id_cuenta INTEGER,
    IN p_monto_apertura DECIMAL(12,2),
    IN p_saldo_cuenta DECIMAL(12,2),
    IN p_descripcion VARCHAR(250),
    IN p_fecha_apertura VARCHAR(250),
    IN p_otros_detalles VARCHAR(250),
    IN p_tipo_cuenta INTEGER,
    IN p_id_cliente INTEGER
)
BEGIN
    DECLARE error_message VARCHAR(250);

    IF EXISTS (SELECT 1 FROM cuenta WHERE id_cuenta = p_id_cuenta) THEN
        SELECT 'Error: El numero de cuenta ya esta registrado.' AS
Message;
    END IF;

    -- Verificar que todos los parámetros tengan una longitud mayor a cero
excepto el quinto y sexto parámetro
    IF LENGTH(p_descripcion) = 0 THEN
        SELECT 'Error: Los parámetros de descripción y otros detalles deben
tener una longitud mayor a cero.' AS Message;
    END IF;

    -- Si el quinto parámetro viene vacío, asignarle el valor de la función
NOW() para guardar la fecha actual
    IF p_fecha_apertura = '' THEN
        SET p_fecha_apertura = NOW();
    ELSE
        SET p_fecha_apertura = STR_TO_DATE(p_fecha_apertura, '%d/%m/%Y
%H:%i:%s');
    END IF;

    -- Verificar que el monto de apertura y el saldo de cuenta sean iguales
y mayores a 0
    IF p_monto_apertura <> p_saldo_cuenta OR p_monto_apertura <= 0 THEN
```

```
        SELECT 'Error: El monto de apertura y el saldo de cuenta deben ser
iguales y mayores a 0.' AS  Message;
    END IF;

    -- Validar que el tipo de cuenta exista en la tabla tipo_cuenta
    IF NOT EXISTS (SELECT 1 FROM tipo_cuenta WHERE codigo = p_tipo_cuenta)
THEN
        SELECT 'Error: El tipo de cuenta especificado no existe en la tabla
tipo_cuenta.' AS Message;
    END IF;

    -- Validar que el id_cliente exista en la tabla cliente
    IF NOT EXISTS (SELECT 1 FROM cliente WHERE id_cliente = p_id_cliente)
THEN
        SELECT 'Error: El ID de cliente especificado no existe en la tabla
cliente.' AS Message;
    END IF;

    -- Insertar en la tabla cuenta si todas las validaciones son exitosas
    INSERT INTO cuenta (id_cuenta, monto_apertura, saldo_cuenta,
descripcion, fecha_apertura, otros_detalles, tipo_cuenta, id_cliente)
    VALUES (p_id_cuenta, p_monto_apertura, p_saldo_cuenta, p_descripcion,
p_fecha_apertura, p_otros_detalles, p_tipo_cuenta, p_id_cliente);

    -- Mostrar mensaje de éxito
    SELECT 'Se ha registrado la cuenta exitosamente.' AS Message;
END $$

DELIMITER ;
```

**crearProductoServicio**: permite insertar productos o servicios diferenciándolos en 1 para servicios y 2 para productos.

Realiza distintas validaciones así como también que tipo puede incluir el precio y cual no.

```
    DELIMITER //

    CREATE PROCEDURE crearProductoServicio(
        IN p_codigo INTEGER,
        IN p_tipo INTEGER,
        IN p_costo DECIMAL(12,2),
        IN p_descripcion VARCHAR(250)
```

```
    )
    BEGIN
        DECLARE error_message VARCHAR(250);

        -- Verificar que los parámetros no sean nulos
        IF p_codigo IS NULL OR p_tipo IS NULL OR p_costo IS NULL OR
p_descripcion IS NULL THEN
            SET error_message = 'Error: Todos los parámetros deben tener
valores no nulos.';
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;

        -- Verificar que el primer parámetro no esté ya insertado en la
tabla productos_servicios
        IF EXISTS (SELECT 1 FROM productos_servicios WHERE codigo =
p_codigo) THEN
            SET error_message = 'Error: El código proporcionado ya está
siendo utilizado en la tabla productos_servicios.';
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;

        -- Verificar que el segundo parámetro tenga el valor 1 o 2
únicamente
        IF p_tipo NOT IN (1, 2) THEN
            SET error_message = 'Error: El tipo debe ser 1 o 2 únicamente.';
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;

        -- Verificar que el tercer parámetro sea mayor a cero si el valor
del segundo parámetro es 1, de lo contrario debe ser cero
        IF (p_tipo = 1 AND p_costo <= 0) OR (p_tipo = 2 AND p_costo <> 0)
THEN
            SET error_message = 'Error: El costo debe ser mayor a cero si el
tipo es 1, de lo contrario debe ser cero.';
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;

        -- Insertar en la tabla productos_servicios si todas las
validaciones son exitosas
        INSERT INTO productos_servicios (codigo, tipo, costo, descripcion)
        VALUES (p_codigo, p_tipo, p_costo, p_descripcion);

        -- Mostrar mensaje de éxito
        SELECT 'Se ha creado el producto o servicio exitosamente.' AS
Message;
```

```
    END //

    DELIMITER ;
```

**realizarCompra**: procedimiento almacenado que registra todas las compras, realiza distintas validaciones precio a dejar insertar.

```
-- tabla compras

    DELIMITER //

    CREATE PROCEDURE realizarCompra(
        IN p_id_compra INTEGER,
        IN p_fecha VARCHAR(50),
        IN p_importe_compra DECIMAL(12,2),
        IN p_otros_detalles VARCHAR(250),
        IN p_codigo_producto_servicio INTEGER,
        IN p_id_cliente INTEGER
    )
    BEGIN
        DECLARE error_message VARCHAR(250);
        DECLARE tipo_producto INTEGER;

        -- Validar que el primer parámetro no esté siendo utilizado ya en la
tabla compras
        IF EXISTS (SELECT 1 FROM compra WHERE id_compra = p_id_compra) THEN
            SET error_message = 'Error: El ID de compra proporcionado ya
está siendo utilizado en la tabla compra.';
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;

        -- Insertar la fecha en formato adecuado
        SET p_fecha = STR_TO_DATE(p_fecha, '%d/%m/%Y %H:%i:%s');

        -- Validar que el valor del quinto parámetro exista en la tabla
productos_servicios
        IF NOT EXISTS (SELECT 1 FROM productos_servicios WHERE codigo =
p_codigo_producto_servicio) THEN
            SET error_message = 'Error: El código de producto/servicio
especificado no existe en la tabla productos_servicios.';
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        ELSE
            -- Si el producto/servicio existe, verificar la validez del
importe_compra si el tipo es 2
```

```
            SELECT tipo INTO tipo_producto FROM productos_servicios WHERE
codigo = p_codigo_producto_servicio;
            IF tipo_producto = 2 AND p_importe_compra <= 0 THEN
                SET error_message = 'Error: El importe de la compra debe ser
mayor a cero para productos de tipo servicio.';
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
            END IF;
        END IF;

        -- Validar que el id_cliente exista en la tabla cliente
        IF NOT EXISTS (SELECT 1 FROM cliente WHERE id_cliente =
p_id_cliente) THEN
            SET error_message = 'Error: El ID de cliente especificado no
existe en la tabla cliente.';
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;

        -- Insertar en la tabla compra si todas las validaciones son
exitosas
        INSERT INTO compra (id_compra, fecha, importe_compra,
otros_detalles, codigo_producto_servicio, id_cliente)
        VALUES (p_id_compra, p_fecha, p_importe_compra, p_otros_detalles,
p_codigo_producto_servicio, p_id_cliente);

        -- Mostrar mensaje de éxito
        SELECT 'Se ha realizado la compra exitosamente.' AS Message;
    END //

    DELIMITER ;
```

**asignarTransaccion**: permite crear la información para las transacciones, valida que tipo de transacción se esta realizando, y si hay dinero de por medio se encarga de realizar las operaciones monetarias.

```
    DELIMITER //

CREATE PROCEDURE asignarTransaccion(
    IN p_ignore_param INTEGER,
    IN p_fecha VARCHAR(250),
    IN p_otros_detalles VARCHAR(250),
    IN p_id_tipo_transaccion INTEGER,
```

```
    IN p_id_operacion INTEGER,
    IN p_no_cuenta INTEGER
)
BEGIN
    DECLARE error_message VARCHAR(250);
    DECLARE monto_operacion DECIMAL(12,2);
    DECLARE saldo DECIMAL(12,2);
    DECLARE aux_operation INTEGER;



    -- Validar que el segundo parámetro tenga longitud mayor a 0
    IF LENGTH(p_fecha) = 0 THEN
        SET error_message = 'Error: La fecha no puede estar vacía.';
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
    END IF;

    -- Insertar la fecha en formato adecuado
    SET p_fecha = STR_TO_DATE(p_fecha, '%d/%m/%Y %H:%i:%s');

    -- Validar que el sexto parámetro exista en la columna id_cuenta de la
tabla cuenta
    IF NOT EXISTS (SELECT 1 FROM cuenta WHERE id_cuenta = p_no_cuenta) THEN
        SET error_message = 'Error: El ID de cuenta especificado no existe
en la tabla cuenta.';
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
    END IF;

    -- Validar que el valor del quinto parámetro exista en la tabla
correspondiente según el valor del cuarto parámetro

    IF p_id_tipo_transaccion = 1 THEN -- Transacción de compra
        -- Verificar que el quinto parámetro exista en la tabla compra y
guardar el valor de la columna importe_compra
        SELECT 1 into aux_operation;
        SELECT importe_compra INTO monto_operacion FROM compra WHERE
id_compra = p_id_operacion;
        IF monto_operacion IS NULL THEN
            SET error_message = 'Error: La compra especificada no existe en
la tabla compra.';
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    ELSEIF p_id_tipo_transaccion = 2 THEN -- Transacción de depósito
        -- Verificar que el quinto parámetro exista en la tabla deposito
        SELECT 2 into aux_operation;
```

```
        SELECT monto INTO monto_operacion FROM deposito WHERE id_deposito =
p_id_operacion;
        IF NOT EXISTS (SELECT 1 FROM deposito WHERE id_deposito =
p_id_operacion) THEN
            SET error_message = 'Error: El depósito especificado no existe
en la tabla deposito.';
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    ELSEIF p_id_tipo_transaccion = 3 THEN -- Transacción de débito
        -- Verificar que el quinto parámetro exista en la tabla debito y
guardar el valor de la columna monto
        SELECT 3 into aux_operation;
        SELECT monto INTO monto_operacion FROM debito WHERE id_debito =
p_id_operacion;
        IF monto_operacion IS NULL THEN
            SET error_message = 'Error: El débito especificado no existe en
la tabla debito.';
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    ELSE
        SET error_message = 'Error: El tipo de transacción especificado no
es válido.';
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
    END IF;

    -- Verificar que el saldo de la cuenta sea mayor al monto de la
operación

    SELECT saldo_cuenta INTO saldo FROM cuenta WHERE id_cuenta =
p_no_cuenta;
    IF saldo < monto_operacion THEN
        SET error_message = 'Error: El saldo de la cuenta no es suficiente
para realizar la transacción.';
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
    END IF;

    -- Insertar en la tabla transaccion si todas las validaciones son
exitosas
    INSERT INTO transaccion (fecha, otros_detalles, id_tipo_transaccion,
id_operacion, no_cuenta)
    VALUES (p_fecha, p_otros_detalles, p_id_tipo_transaccion,
p_id_operacion, p_no_cuenta);

    -- Actualizando en la tabla cuenta
       IF aux_operation = 2 THEN
```

```
            UPDATE cuenta SET saldo_cuenta = (saldo_cuenta +
monto_operacion)
            WHERE id_cuenta = p_no_cuenta;
       ELSE
            UPDATE cuenta SET saldo_cuenta = (saldo_cuenta -
monto_operacion)
            WHERE id_cuenta = p_no_cuenta;
       END IF;


    SELECT 'Se ha asignado la transacción exitosamente.' AS Message;
END //

DELIMITER ;
```

# Triggers

**Trigers para la tabla tipo_cuenta**: se agrego un trigger after_insert y otro after update para tener registro de las actividades en la tabla tipo_cuenta.

```
-- tabla tipo_cuenta

DELIMITER //
CREATE TRIGGER tipo_cuenta_after_insert
AFTER INSERT ON tipo_cuenta
FOR EACH ROW
BEGIN
        INSERT INTO historial (fecha, descripcion, tipo)
        VALUES (now(), "Actividad en tabla tipo_cuenta", "INSERT");
END //


CREATE TRIGGER tipo_cuenta_after_update
AFTER UPDATE  ON tipo_cuenta
FOR EACH ROW

BEGIN
        INSERT INTO historial (fecha, descripcion, tipo)
        VALUES (now(), 'Actividad en tabla tipo_cuenta', 'UPDATE');
END //
```

**Trigers para la tabla tipo_cliente**: se agrego un trigger after_insert y otro after update para tener registro de las actividades en la tabla tipo_cliente.

```
-- tipo_cliente

DELIMITER //

CREATE TRIGGER tipo_cliente_after_insert
AFTER INSERT ON tipo_cliente
FOR EACH ROW

BEGIN
	INSERT INTO historial (fecha, descripcion, tipo)
	VALUES (now(), 'Actividad en tabla tipo_cliente', 'INSERT');
END//


CREATE TRIGGER tipo_cliente_after_update
AFTER UPDATE  ON tipo_cliente
FOR EACH ROW

BEGIN
	INSERT INTO historial (fecha, descripcion, tipo)
	VALUES (now(), 'Actividad en tabla tipo_cliente', 'UPDATE');
END//
```

**Trigers para la tabla tipo_transaccion**: se agrego un trigger after_insert y otro after update para tener registro de las actividades en la tabla tipo_transaccion.

```
-- tabla tipo_transaccion

DELIMITER //

CREATE TRIGGER tipo_transaccion_after_insert
AFTER INSERT ON tipo_transaccion
FOR EACH ROW

BEGIN
	INSERT INTO historial (fecha, descripcion, tipo)
	VALUES (now(), 'Actividad en tipo_transaccion', 'INSERT');
END//

CREATE TRIGGER tipo_transaccion_after_update
```

```
AFTER UPDATE  ON tipo_transaccion
FOR EACH ROW

BEGIN
        INSERT INTO historial (fecha, descripcion, tipo)
        VALUES (now(), 'Actividad en tabla tipo_transaccion', 'UPDATE');
END//
```

**Trigers para la tabla cliente**: se agrego un trigger after_insert y otro after update para tener registro de las actividades en la tabla cliente.

```
-- tabla cliente

DELIMITER //

CREATE TRIGGER cliente_after_insert
AFTER INSERT ON cliente
FOR EACH ROW

BEGIN
        INSERT INTO historial (fecha, descripcion, tipo)
        VALUES (now(), 'Actividad en cliente', 'INSERT');
END//

CREATE TRIGGER cliente_after_update
AFTER UPDATE  ON cliente
FOR EACH ROW

BEGIN
        INSERT INTO historial (fecha, descripcion, tipo)
        VALUES (now(), 'Actividad en tabla cliente', 'UPDATE');
END//
```

**Trigers para la tabla cuenta**: se agrego un trigger after_insert y otro after update para tener registro de las actividades en la tabla cuenta.

```
-- tabla cuenta

DELIMITER //

CREATE TRIGGER cuenta_after_insert
AFTER INSERT ON cuenta
```

```
FOR EACH ROW

BEGIN
        INSERT INTO historial (fecha, descripcion, tipo)
        VALUES (now(), 'Actividad en cuenta', 'INSERT');
END//

CREATE TRIGGER cuenta_after_update
AFTER UPDATE  ON cuenta
FOR EACH ROW

BEGIN
        INSERT INTO historial (fecha, descripcion, tipo)
        VALUES (now(), 'Actividad en tabla cuenta', 'UPDATE');
END//
```

**Trigers para la tabla transaccion**: se agrego un trigger after_insert y otro after update para tener registro de las actividades en la tabla transaccion.

```
-- tabla transaccion

DELIMITER //

CREATE TRIGGER transaccion_after_insert
AFTER INSERT ON transaccion
FOR EACH ROW

BEGIN
        INSERT INTO historial (fecha, descripcion, tipo)
        VALUES (now(), 'Actividad en transaccion', 'INSERT');
END//

CREATE TRIGGER transaccion_after_update
AFTER UPDATE  ON transaccion
FOR EACH ROW

BEGIN
        INSERT INTO historial (fecha, descripcion, tipo)
        VALUES (now(), 'Actividad en tabla transaccion', 'UPDATE');
END//
```