DO $$

BEGIN

IF EXISTS (SELECT FROM information\_schema.tables WHERE table\_name = 'transactions') THEN

EXECUTE 'DROP TABLE transactions CASCADE';

END IF;

IF EXISTS (SELECT FROM information\_schema.tables WHERE table\_name = 'accounts') THEN

EXECUTE 'DROP TABLE accounts CASCADE';

END IF;

IF EXISTS (SELECT FROM information\_schema.tables WHERE table\_name = 'customers') THEN

EXECUTE 'DROP TABLE customers CASCADE';

END IF;

IF EXISTS (SELECT FROM information\_schema.tables WHERE table\_name = 'auditlog') THEN

EXECUTE 'DROP TABLE auditlog CASCADE';

END IF;

END

$$;

CREATE TABLE Customers (

CustomerID INTEGER PRIMARY KEY,

Name VARCHAR(100),

DOB DATE,

Balance NUMERIC,

LastModified DATE

);

INSERT INTO Customers VALUES

(1, 'Martina', DATE '1985-05-15', 1000, CURRENT\_DATE),

(2, 'Monika', DATE '1990-07-20', 15000, CURRENT\_DATE);

CREATE TABLE Accounts (

AccountID INTEGER PRIMARY KEY,

CustomerID INTEGER REFERENCES Customers(CustomerID),

AccountType VARCHAR(20),

Balance NUMERIC,

LastModified DATE

);

INSERT INTO Accounts VALUES

(1, 1, 'Savings', 5000, CURRENT\_DATE),

(2, 2, 'Checking', 12000, CURRENT\_DATE);

CREATE TABLE Transactions (

TransactionID SERIAL PRIMARY KEY,

AccountID INTEGER REFERENCES Accounts(AccountID),

TransactionDate DATE DEFAULT CURRENT\_DATE,

Amount NUMERIC,

TransactionType VARCHAR(10)

);

CREATE TABLE AuditLog (

AuditID SERIAL PRIMARY KEY,

TransactionID INTEGER,

AccountID INTEGER,

LogTime TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

Message TEXT

);

CREATE OR REPLACE FUNCTION SetCustomerLastModified()

RETURNS TRIGGER AS $$

BEGIN

NEW.LastModified := CURRENT\_DATE;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

EXECUTE FUNCTION SetCustomerLastModified();

CREATE OR REPLACE FUNCTION LogTransactionAudit()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO AuditLog(TransactionID, AccountID, Message)

VALUES (NEW.TransactionID, NEW.AccountID,

'Transaction logged: ' || NEW.TransactionType || ' of amount ' || NEW.Amount);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

EXECUTE FUNCTION LogTransactionAudit();

CREATE OR REPLACE FUNCTION EnforceTransactionRules()

RETURNS TRIGGER AS $$

DECLARE

v\_balance NUMERIC;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = NEW.AccountID;

IF NEW.TransactionType = 'Withdrawal' THEN

IF v\_balance < NEW.Amount THEN

RAISE EXCEPTION 'Insufficient funds for withdrawal.';

END IF;

ELSIF NEW.TransactionType = 'Deposit' THEN

IF NEW.Amount <= 0 THEN

RAISE EXCEPTION 'Deposit amount must be positive.';

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

EXECUTE FUNCTION EnforceTransactionRules();

UPDATE Customers SET Balance = Balance + 500 WHERE CustomerID = 1;

INSERT INTO Transactions (AccountID, Amount, TransactionType)

VALUES (1, 300, 'Deposit');

INSERT INTO Transactions (AccountID, Amount, TransactionType)

VALUES (2, 500, 'Withdrawal');

SELECT \* FROM Customers;

SELECT \* FROM Transactions;

SELECT \* FROM AuditLog;