Bank Management System (SQL Project Documentation)

Developer: Kritika Lohomi

CSI ID: CT\_CSI\_SQ\_4357

# Table of Contents

1. 1. Project Overview
2. 2. ER Diagram
3. 3. Database Schema
4. 4. Stored Procedures
5. 5. Data Integrity and Constraints
6. 6. Sample Data & Execution
7. 7. Conclusion

# 1. Project Overview

This project demonstrates the core backend logic of a banking management system using SQL Server. The project covers:  
- Schema design using normalized relational tables  
- Implementation of business logic via stored procedures  
- Enforcement of data integrity using constraints and validations  
- Features such as account creation, money deposit/withdrawal, fund transfer, employee management, and transaction history tracking

# 2. ER Diagram

The ER Diagram outlines the relationships between core entities such as Person, Customer, Employee, Account, Branch, and Transaction. Each Customer or Employee is a Person. A Customer owns accounts which are linked to a Branch and tracked via Transactions.

# 3. Database Schema

## Person

PersonID INT PRIMARY KEY  
FirstName, LastName, DOB, Email, PhoneNumber, Address  
TaxIdentifier, AadhaarNumber, PANNumber

## Customer

CustomerID INT PRIMARY KEY  
PersonID (FK), CustomerType  
AadhaarNumber (masked), PANNumber (masked)

## Employee

EmployeeID INT PRIMARY KEY  
PersonID (FK), Position, Salary

## Branch

BranchID INT PRIMARY KEY  
BranchName, BranchCode, Address, PhoneNumber

## Account

AccountID INT PRIMARY KEY  
CustomerID (FK), BranchID (FK)  
AccountNumber (unique), AccountType  
CurrentBalance, DateOpened, DateClosed, AccountStatus

## Transaction

TransactionID INT PRIMARY KEY  
AccountID (FK), TransactionType, Amount, TransactionDate

# 4. Stored Procedures

## proc\_CreateNewAccount

Creates a new customer and account with validation and returns generated IDs.

## proc\_DepositAmount

Deposits funds into a given account with balance update and transaction logging.

## proc\_WithdrawalAmount

Withdraws funds after checking balance, updates account and logs transaction.

## proc\_TransferAmount

Transfers funds between two accounts, with full transaction logging.

## proc\_CloseAccount

Marks an account as closed and logs closure.

## proc\_CreateNewEmployee

Registers a new employee with PAN/Aadhaar validation.

## proc\_ViewTransactionHistory

Displays customer info and last 10 transactions.

# 5. Data Integrity and Constraints

- Primary & Foreign Keys: Ensure relational integrity.  
- NOT NULL Constraints: Mandatory fields.  
- CHECK Constraints: Balance ≥ 0, valid email, transaction amount ≥ 0.  
- UNIQUE Constraints: Aadhaar, PAN, AccountNumber.  
- DEFAULT Values: AccountStatus ('Active'), TransactionDate (GETDATE()).  
- Dynamic Data Masking: Aadhaar → XXXX-XXXX-1234, PAN → XXXX123456.

# 6. Sample Data & Execution

Sample Account Creation:  
EXEC proc\_createnewaccount @FirstName = 'Hema', ..., @AccountType = 'Savings';  
  
Deposit:  
EXEC proc\_DepositAmount @Amount=1200, @AccountID=2;  
  
Withdrawal:  
EXEC proc\_withdrawalAmount @Amount=1000, @AccountID=3;  
  
Transfer:  
EXEC proc\_TransferAmount @FromAccountID=1, @ToAccountID=2, @Amount=500;  
  
View Transactions:  
EXEC proc\_ViewTransactionHistory @AccountID=1;

# 7. Conclusion

This banking system simulation demonstrates a real-world relational design, following best practices of:  
- Modularity via stored procedures  
- Security-first design (masking, validations)  
- Scalable and normalized schema  
- Focus on data consistency and auditability  
  
By deploying this system to SQL Server or Azure SQL, it can be further enhanced with triggers, backups, and integration with frontends.

