Project Report

On

Development Of Banking System Database

Project Submitted By Mantu Pal

Table of Contents

Project Overview

Project Title: **Development of Banking System Database**

Description: The banking industry is increasingly reliant on sophisticated data management systems to handle customer information, manage accounts, and track transactions. This project aims to design and implement a robust and scalable database for a banking system. The goal is to ensure that the database can efficiently manage customer details, account information, and transaction records while maintaining data integrity, security, and performance.

The project involves creating a relational database schema that supports the operations of a banking system, including account creation, deposits, withdrawals, transfers, and transaction history tracking. The database will be implemented using SQL, and stored procedures will be developed to handle the various operations required by the banking system. The end product will be a comprehensive set of SQL scripts for table creation and stored procedures, accompanied by detailed documentation.

Project Objectives

The primary objectives of this project are as follows:

1. **Design a Robust Database Schema**:
    - Develop a relational database schema tailored to the needs of a banking system.
    - Ensure the schema includes tables for storing customer information, account details, and transaction records.
    - Implement foreign key constraints to maintain referential integrity between tables.
2. **Implement Key Functionalities Through Stored Procedures**:
    - Create stored procedures for adding new customers, opening new accounts, and recording transactions.
    - Ensure stored procedures handle operations such as deposits, withdrawals, and transfers accurately and efficiently.
    - Implement error handling within stored procedures to manage scenarios like insufficient funds.
3. **Maintain Data Integrity and Security**:
    - Use appropriate data types and constraints to ensure data accuracy and consistency.
    - Implement security measures to protect sensitive customer information.
    - Ensure transactional operations are atomic, consistent, isolated, and durable (ACID properties).
4. **Optimize Database Performance**:
    - Ensure the database design supports efficient query processing and transaction handling.
    - Implement indexing strategies to enhance performance for common queries and operations.
5. **Provide Comprehensive Documentation**:
    - Document the database schema, including detailed descriptions of tables and their relationships.
    - Provide clear and concise documentation for each stored procedure, explaining its purpose, inputs, and outputs.
    - Include sample queries and usage examples to demonstrate how to interact with the database.
6. **Facilitate Testing and Validation**:
    - Develop test cases to validate the functionality of the stored procedures.
    - Ensure that the database operations meet the specified requirements through thorough testing.
    - Provide instructions for setting up and initializing the database for testing purposes.

By achieving these objectives, the project aims to deliver a fully functional and well-documented banking system database that can serve as a reliable foundation for further development and integration with banking applications.

.Scope

The scope of this project encompasses the following activities:

1. **Database Design**:
    - Define the relational database schema, including all necessary tables and relationships.
    - Ensure that the database design supports all required operations and maintains data integrity.
2. **Table Creation**:
    - Implement SQL scripts to create tables for storing customer information, account details, and transaction records.
    - Include appropriate data types, primary keys, and foreign keys in table definitions.
3. **Stored Procedures**:
    - Develop stored procedures to handle the core functionalities of the banking system.
    - Ensure stored procedures cover customer creation, account opening, deposits, withdrawals, transfers, and transaction history retrieval.
4. **Documentation**:
    - Provide detailed documentation of the database schema and stored procedures.
    - Include explanations of each table and stored procedure, along with sample usage.
5. **Testing and Validation**:
    - Create test cases to validate the functionality and performance of the database.
    - Conduct thorough testing to ensure that all operations work as expected.

    - Example queries and instructions for testing the stored procedures.

Resources

To successfully complete this project, the following resources are required:

1. **Software Tools**:
    - SQL Server Management Studio (SSMS) or any other SQL development tool.
    - Access to a database server where the schema and procedures can be implemented and tested.
2. **Documentation**:
    - Guidelines and best practices for database design and SQL development.

- Access to relevant documentation for SQL Server or the specific database management system being used.

3. **Test Data**:
    - Sample data for testing the functionality of the database tables and stored procedures.
    - Test cases to validate the correctness and performance of the stored procedures.

Database Schema

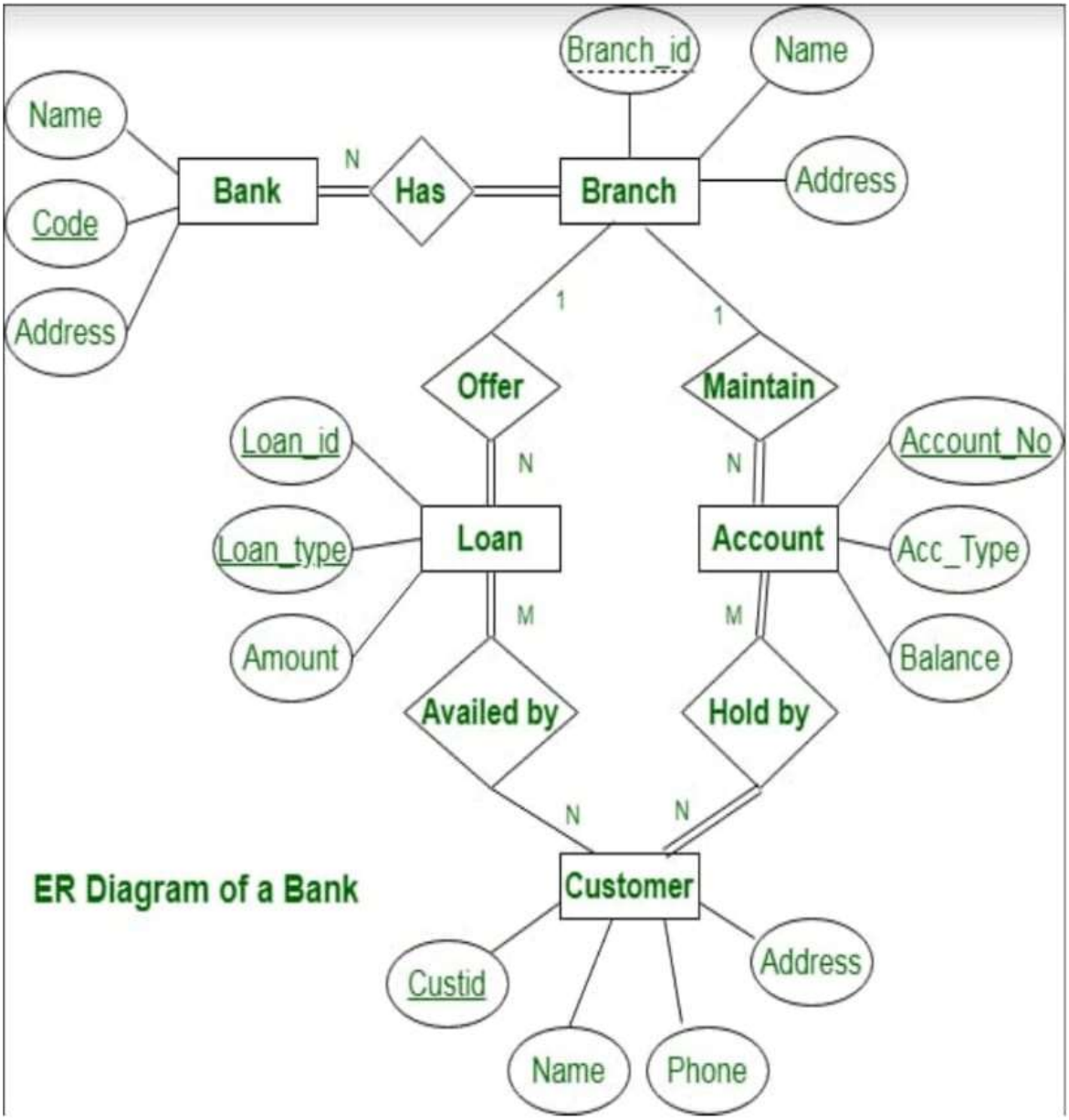


Table Definitions

Customers Table

Stores customer details.

```
CREATE TABLE Customers (
   CustomerID INT PRIMARY KEY IDENTITY,
   FirstName NVARCHAR(50),
   LastName NVARCHAR(50),
   DateOfBirth DATE,
   Address NVARCHAR(255),
   PhoneNumber NVARCHAR(15),
   Email NVARCHAR(50)
);
```

Columns:

- **CustomerID**: Unique identifier for each customer (Primary Key, Identity).
- **FirstName**: First name of the customer.
- **LastName**: Last name of the customer.
- **DateOfBirth**: Date of birth of the customer.
- **Address**: Address of the customer.
- **PhoneNumber**: Phone number of the customer.
- **Email**: Email address of the customer.

Account table

Store Account details linked to customer

```
CREATE TABLE Accounts (
   AccountID INT PRIMARY KEY IDENTITY,
   CustomerID INT FOREIGN KEY REFERENCES Customers(CustomerID),
   AccountType NVARCHAR(20),
   Balance DECIMAL(18, 2),
   CreatedDate DATE
);
```

Columns:

- **AccountID**: Unique identifier for each account (Primary Key, Identity).
- **CustomerID**: Reference to the CustomerID in the Customers table (Foreign Key).
- **AccountType**: Type of the account (e.g., Checking, Savings).
- **Balance**: Current balance of the account.
- **CreatedDate**: Date when the account was created.

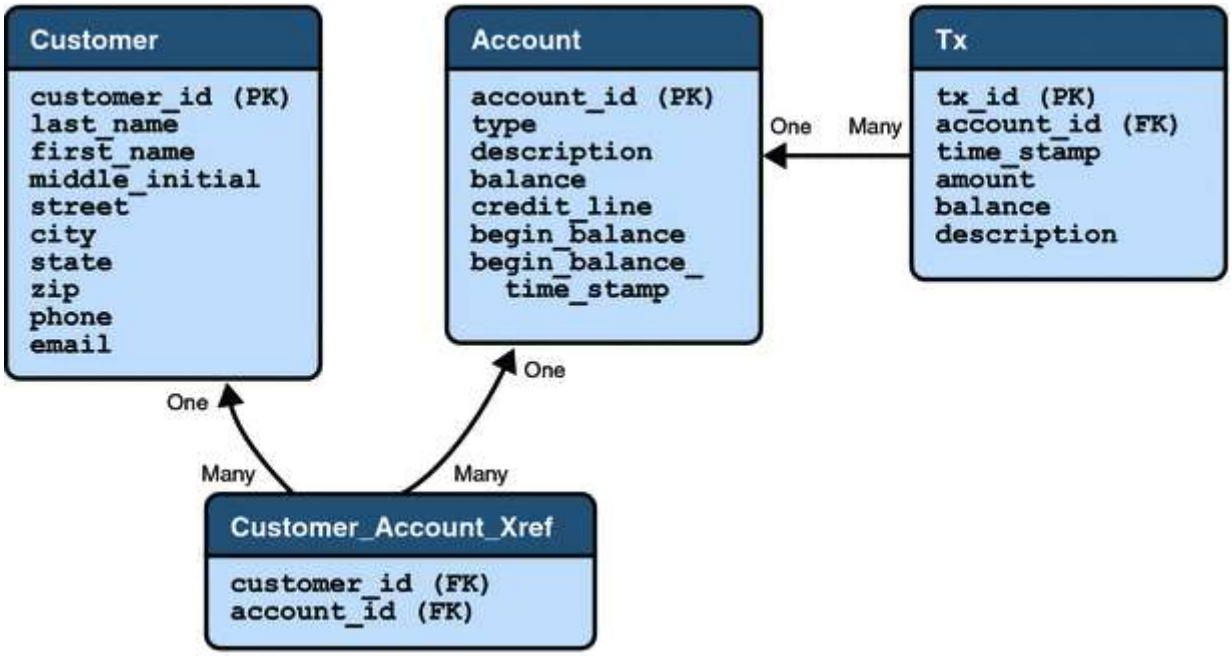
Transaction table

Store transaction record for deposits, withdrawal and transfers

```
CREATE TABLE Transactions (
   TransactionID INT PRIMARY KEY IDENTITY,
   AccountID INT FOREIGN KEY REFERENCES Accounts(AccountID),
   TransactionType NVARCHAR(20),
   Amount DECIMAL(18, 2),
   TransactionDate DATE,
   Description NVARCHAR(255)
);
```

Columns:

- **TransactionID**: Unique identifier for each transaction (Primary Key, Identity).
- **AccountID**: Reference to the AccountID in the Accounts table (Foreign Key).
- **TransactionType**: Type of transaction (e.g., Deposit, Withdrawal, Transfer).
- **Amount**: Amount of money involved in the transaction.
- **TransactionDate**: Date of the transaction.
- **Description**: Description of the transaction.



- **Stored Procedures**

Customer Management

Adds a new customer to the database:

```
CREATE PROCEDURE AddCustomer
    @FirstName NVARCHAR(50),
    @LastName NVARCHAR(50),
    @DateOfBirth DATE,
    @Email NVARCHAR(100),
    @Phone NVARCHAR(15),
    @Address NVARCHAR(255),
    @City NVARCHAR(50),
    @State NVARCHAR(50),
    @ZipCode NVARCHAR(10)
AS
BEGIN
    INSERT INTO Customers (FirstName, LastName, DateOfBirth, Email, Phone, Address, City, State, ZipCode)
    VALUES (@FirstName, @LastName, @DateOfBirth, @Email, @Phone, @Address, @City, @State, @ZipCode);
END;
```

Update Customer:

```
CREATE PROCEDURE UpdateCustomer
    @CustomerID INT,
    @FirstName NVARCHAR(50),
    @LastName NVARCHAR(50),
    @DateOfBirth DATE,
    @Email NVARCHAR(100),
    @Phone NVARCHAR(15),
    @Address NVARCHAR(255),
    @City NVARCHAR(50),
    @State NVARCHAR(50),
    @ZipCode NVARCHAR(10)
AS
BEGIN
    UPDATE Customers
    SET FirstName = @FirstName,
        LastName = @LastName,
```

```
    DateOfBirth = @DateOfBirth,
    Email = @Email,
    Phone = @Phone,
    Address = @Address,
    City = @City,
    State = @State,
    ZipCode = @ZipCode
  WHERE CustomerID = @CustomerID;
END;
```

Account Management

Create Account:

```
CREATE PROCEDURE CreateAccount
  @CustomerID INT,
  @AccountType NVARCHAR(50),
  @InitialDeposit DECIMAL(18, 2)
AS
BEGIN
  INSERT INTO Accounts (CustomerID, AccountType, Balance, DateOpened)
  VALUES (@CustomerID, @AccountType, @InitialDeposit, GETDATE());
END;
```

Deposit Money

```
CREATE PROCEDURE DepositMoney
  @AccountID INT,
  @Amount DECIMAL(18, 2)
AS
BEGIN
  UPDATE Accounts
  SET Balance = Balance + @Amount
  WHERE AccountID = @AccountID;

  INSERT INTO Transactions (AccountID, TransactionType, Amount, TransactionDate, Description)
  VALUES (@AccountID, 'Deposit', @Amount, GETDATE(), 'Deposit to account');
END;
```

Withdraw Money

```
CREATE PROCEDURE WithdrawMoney
```

```
    @AccountID INT,
    @Amount DECIMAL(18, 2)
AS
BEGIN
    DECLARE @CurrentBalance DECIMAL(18, 2);
    SELECT @CurrentBalance = Balance FROM Accounts WHERE AccountID = @AccountID;

    IF @CurrentBalance >= @Amount
    BEGIN
        UPDATE Accounts
        SET Balance = Balance - @Amount
        WHERE AccountID = @AccountID;

        INSERT INTO Transactions (AccountID, TransactionType, Amount, TransactionDate, Description)
        VALUES (@AccountID, 'Withdrawal', @Amount, GETDATE(), 'Withdrawal from account');
    END
    ELSE
    BEGIN
        RAISERROR('Insufficient funds', 16, 1);
    END
END;
```

Transfer Money

```
CREATE PROCEDURE TransferMoney
    @FromAccountID INT,
    @ToAccountID INT,
    @Amount DECIMAL(18, 2)
AS
BEGIN
    BEGIN TRANSACTION;

    DECLARE @CurrentBalance DECIMAL(18, 2);
    SELECT @CurrentBalance = Balance FROM Accounts WHERE AccountID = @FromAccountID;

    IF @CurrentBalance >= @Amount
    BEGIN
```

```
    UPDATE Accounts
    SET Balance = Balance - @Amount
    WHERE AccountID = @FromAccountID;

    UPDATE Accounts
    SET Balance = Balance + @Amount
    WHERE AccountID = @ToAccountID;

    INSERT INTO Transactions (AccountID, TransactionType, Amount, TransactionDate, Description)
    VALUES (@FromAccountID, 'Transfer Out', @Amount, GETDATE(), 'Transfer to account ' +
CAST(@ToAccountID AS NVARCHAR(10)));

    INSERT INTO Transactions (AccountID, TransactionType, Amount, TransactionDate, Description)
    VALUES (@ToAccountID, 'Transfer In', @Amount, GETDATE(), 'Transfer from account ' +
CAST(@FromAccountID AS NVARCHAR(10)));
  END
  ELSE
  BEGIN
    ROLLBACK TRANSACTION;
    RAISERROR('Insufficient funds', 16, 1);
  END

  COMMIT TRANSACTION;
END;
```

View Transaction History

```
CREATE PROCEDURE ViewTransactionHistory
  @AccountID INT
AS
BEGIN
  SELECT * FROM Transactions
  WHERE AccountID = @AccountID
  ORDER BY TransactionDate DESC;
END;
```

Update Account Balance

```
CREATE PROCEDURE UpdateAccountBalance
```

```
    @AccountID INT,
    @Amount DECIMAL(18, 2),
    @TransactionType NVARCHAR(50)
AS
BEGIN
    IF @TransactionType = 'Deposit'
    BEGIN
        UPDATE Accounts
        SET Balance = Balance + @Amount
        WHERE AccountID = @AccountID;
    END
    ELSE IF @TransactionType = 'Withdrawal'
    BEGIN
        DECLARE @Balance DECIMAL(18, 2);
        SELECT @Balance = Balance FROM Accounts WHERE AccountID = @AccountID;

        IF @Balance >= @Amount
        BEGIN
            UPDATE Accounts
            SET Balance = Balance - @Amount
            WHERE AccountID = @AccountID;
        END
        ELSE
        BEGIN
            RAISERROR('Insufficient funds', 16, 1);
        END
    END
END;
```

Transaction Management

Add Transaction:

```
CREATE PROCEDURE AddTransaction
    @AccountID INT,
    @TransactionType NVARCHAR(50),
    @Amount DECIMAL(18, 2),
    @Description NVARCHAR(255)
```

```
AS
BEGIN
    INSERT INTO Transactions (AccountID, TransactionType, Amount, TransactionDate, Description)
    VALUES (@AccountID, @TransactionType, @Amount, GETDATE(), @Description);

    EXEC UpdateAccountBalance @AccountID, @Amount, @TransactionType;
END;
```

View Trasaction

```
CREATE PROCEDURE ViewTransactions
    @AccountID INT
AS
BEGIN
    SELECT * FROM Transactions
    WHERE AccountID = @AccountID
    ORDER BY TransactionDate DESC;
END;
```

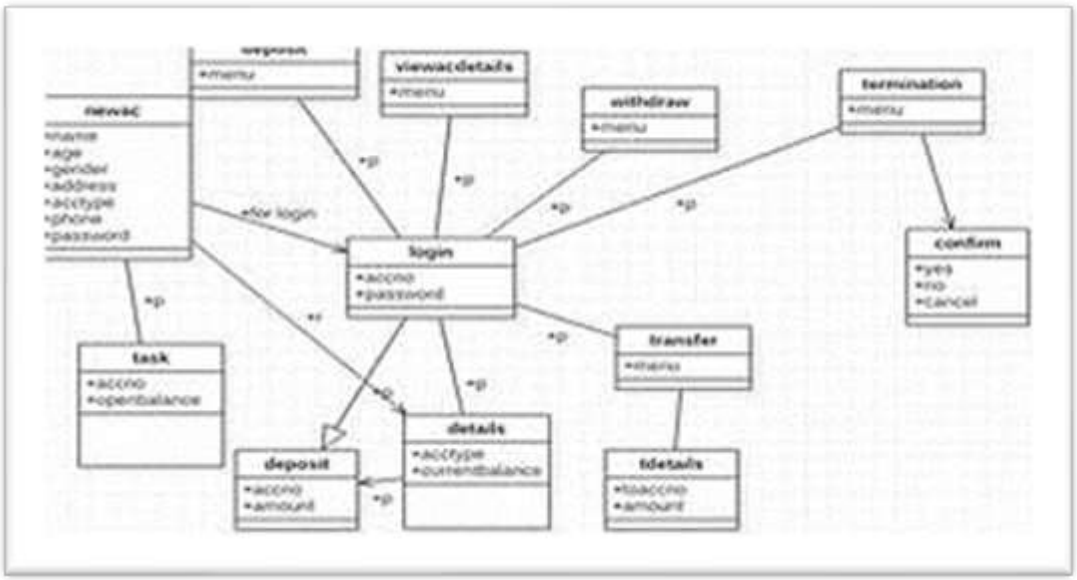
Loan Management

Tables

```
CREATE TABLE Loans (
    LoanID INT PRIMARY KEY IDENTITY,
    CustomerID INT,
    LoanAmount DECIMAL(18, 2),
    InterestRate DECIMAL(5, 2),
    StartDate DATE,
    EndDate DATE,
    Status NVARCHAR(20),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

CREATE TABLE LoanRepayments (
    RepaymentID INT PRIMARY KEY IDENTITY,
    LoanID INT,
    RepaymentAmount DECIMAL(18, 2),
    RepaymentDate DATE,
    FOREIGN KEY (LoanID) REFERENCES Loans(LoanID)
```

);



Stored Procedure

```
CREATE PROCEDURE ApplyForLoan
   @CustomerID INT,
   @LoanAmount DECIMAL(18, 2),
   @InterestRate DECIMAL(5, 2),
   @StartDate DATE,
   @EndDate DATE
AS
BEGIN
   INSERT INTO Loans (CustomerID, LoanAmount, InterestRate, StartDate, EndDate, Status)
   VALUES (@CustomerID, @LoanAmount, @InterestRate, @StartDate, @EndDate, 'Pending');
END;

CREATE PROCEDURE ApproveLoan
   @LoanID INT
AS
BEGIN
   UPDATE Loans
   SET Status = 'Approved'
   WHERE LoanID = @LoanID;
END;
```

```
CREATE PROCEDURE RepayLoan
    @LoanID INT,
    @RepaymentAmount DECIMAL(18, 2)
AS
BEGIN
    INSERT INTO LoanRepayments (LoanID, RepaymentAmount, RepaymentDate)
    VALUES (@LoanID, @RepaymentAmount, GETDATE());

    DECLARE @TotalRepayment DECIMAL(18, 2);
    SELECT @TotalRepayment = SUM(RepaymentAmount) FROM LoanRepayments WHERE LoanID =
@LoanID;

    DECLARE @LoanAmount DECIMAL(18, 2);
    SELECT @LoanAmount = LoanAmount FROM Loans WHERE LoanID = @LoanID;

    IF @TotalRepayment >= @LoanAmount
    BEGIN
        UPDATE Loans
        SET Status = 'Repaid'
        WHERE LoanID = @LoanID;
    END;
END;
```

Mobile And Online Banking Integration

Tables

```
CREATE TABLE OnlineBankingUsers (
    UserID INT PRIMARY KEY IDENTITY,
    CustomerID INT,
    Username NVARCHAR(50) UNIQUE,
    PasswordHash NVARCHAR(255),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

Stored Procedure

```
CREATE PROCEDURE RegisterOnlineUser
    @CustomerID INT,
```

```
    @Username NVARCHAR(50),
    @PasswordHash NVARCHAR(255)
AS
BEGIN
    INSERT INTO OnlineBankingUsers (CustomerID, Username, PasswordHash)
    VALUES (@CustomerID, @Username, @PasswordHash);
END;

CREATE PROCEDURE AuthenticateOnlineUser
    @Username NVARCHAR(50),
    @PasswordHash NVARCHAR(255)
AS
BEGIN
    SELECT UserID FROM OnlineBankingUsers
    WHERE Username = @Username AND PasswordHash = @PasswordHash;
END;
```

ATM Transaction

Tables

```
CREATE TABLE ATMTransactions (
    ATMTransactionID INT PRIMARY KEY IDENTITY,
    AccountID INT,
    TransactionType NVARCHAR(50),
    Amount DECIMAL(18, 2),
    TransactionDate DATETIME,
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);
```

Stored Procedure

```
CREATE PROCEDURE ATMWithdraw
    @AccountID INT,
    @Amount DECIMAL(18, 2)
AS
BEGIN
    DECLARE @Balance DECIMAL(18, 2);
    SELECT @Balance = Balance FROM Accounts WHERE AccountID = @AccountID;
```

```
    IF @Balance >= @Amount
    BEGIN
        UPDATE Accounts
        SET Balance = Balance - @Amount
        WHERE AccountID = @AccountID;

        INSERT INTO ATMTransactions (AccountID, TransactionType, Amount, TransactionDate)
        VALUES (@AccountID, 'Withdrawal', @Amount, GETDATE());
    END
    ELSE
    BEGIN
        RAISERROR('Insufficient funds', 16, 1);
    END
END;

CREATE PROCEDURE ATMDeposit
    @AccountID INT,
    @Amount DECIMAL(18, 2)
AS
BEGIN
    UPDATE Accounts
    SET Balance = Balance + @Amount
    WHERE AccountID = @AccountID;

    INSERT INTO ATMTransactions (AccountID, TransactionType, Amount, TransactionDate)
    VALUES (@AccountID, 'Deposit', @Amount, GETDATE());
END;

CREATE PROCEDURE ATMBalanceInquiry
    @AccountID INT
AS
BEGIN
    SELECT Balance FROM Accounts
    WHERE AccountID = @AccountID;
END;
```

```
CREATE PROCEDURE ATMMiniStatement
   @AccountID INT
AS
BEGIN
   SELECT TOP 5 * FROM ATMTransactions
   WHERE AccountID = @AccountID
   ORDER BY TransactionDate DESC;
END;
```

Customer Support System

Tables

```
CREATE TABLE SupportTickets (
   TicketID INT PRIMARY KEY IDENTITY,
   CustomerID INT,
   IssueDescription NVARCHAR(255),
   Status NVARCHAR(20),
   CreatedDate DATETIME,
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

Stored Procedure

```
CREATE PROCEDURE CreateSupportTicket
   @CustomerID INT,
   @IssueDescription NVARCHAR(255)
AS
BEGIN
   INSERT INTO SupportTickets (CustomerID, IssueDescription, Status, CreatedDate)
   VALUES (@CustomerID, @IssueDescription, 'Open', GETDATE());
END;

CREATE PROCEDURE UpdateSupportTicketStatus
   @TicketID INT,
   @Status NVARCHAR(20)
AS
BEGIN
   UPDATE SupportTickets
```

```
    SET Status = @Status
    WHERE TicketID = @TicketID;
END;

CREATE PROCEDURE ViewSupportTickets
    @CustomerID INT
AS
BEGIN
    SELECT * FROM SupportTickets
    WHERE CustomerID = @CustomerID
    ORDER BY CreatedDate DESC;
END;
```

Testing And Validation

Test Cases

1. **CreateCustomer:**

```
EXEC CreateCustomer 'John', 'Doe', '1980-01-01', '123 Main St', '555-1234', 'john.doe@example.com';
```

- **Expected Result**: A new customer record should be added to the Customers table.

2.OpenAccount:

```
EXEC OpenAccount 1, 'Checking', 1000.00;
```

- **Expected Result**: A new account with an initial deposit should be created for the customer with CustomerID = 1.

3.DepositMoney:

```
EXEC DepositMoney 1, 500.00;
```

Expected Result: The balance of the account with AccountID = 1 should increase by 500.00, and a corresponding transaction record should be added.

4.WithdrawMoney:

```
EXEC WithdrawMoney 1, 200.00;
```

- **Expected Result**: The balance of the account with AccountID = 1 should decrease by 200.00 if sufficient funds are available, and a corresponding transaction record should be added. If insufficient funds, an error should be raised.

5.TransferMoney:

```
EXEC TransferMoney 1, 2, 50.00;
```

- **Expected Result**: 50.00 should be transferred from the account with AccountID = 1 to the account with AccountID = 2, and corresponding transaction records should be added for both accounts. If insufficient funds, an error should be raised, and the transaction should be rolled back.

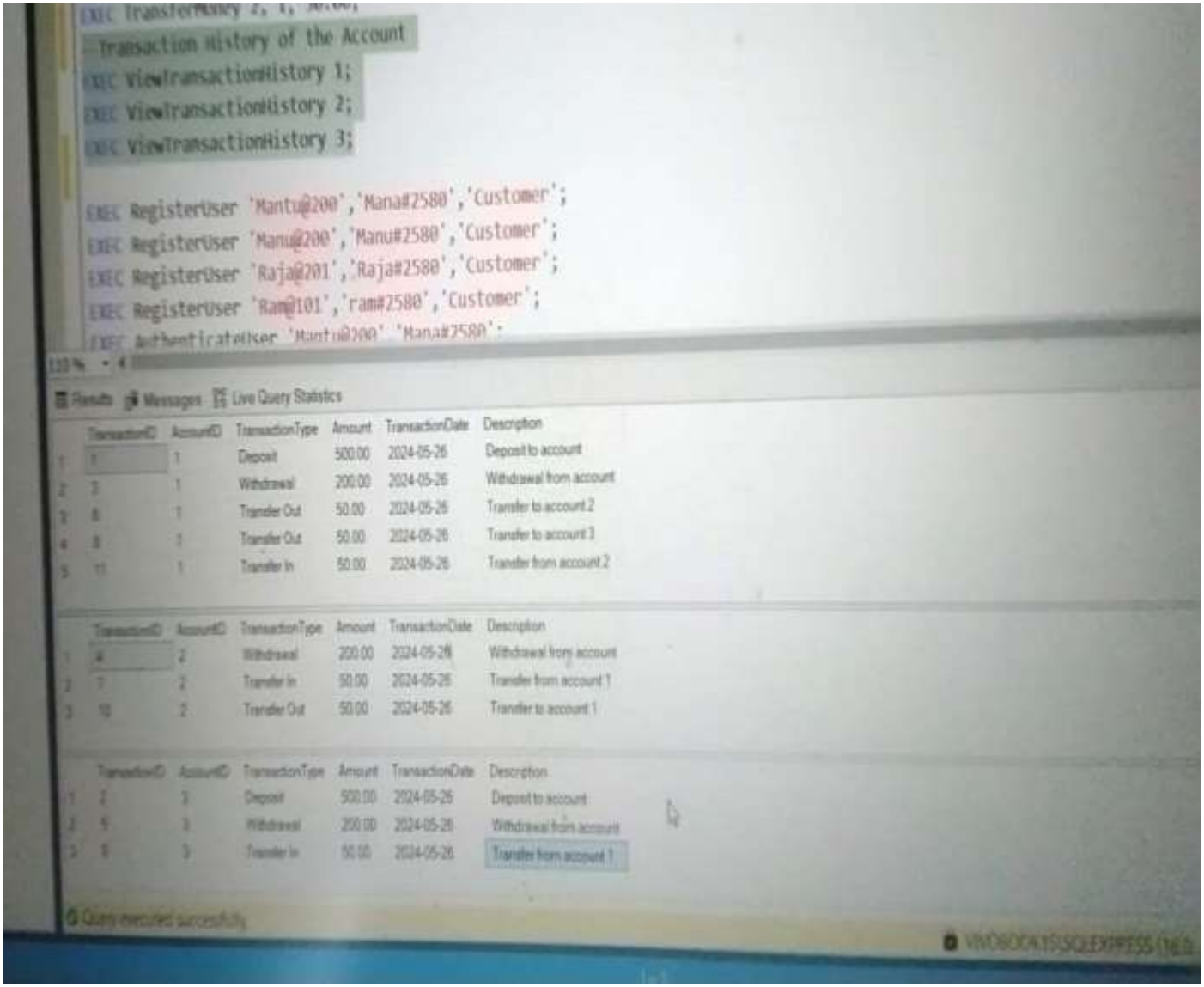
6.ViewTransactionHistory:

EXEC ViewTransactionHistory 1;

- **Expected Result**: The transaction history for the account with AccountID = 1 should be displayed, ordered by TransactionDate in descending order.

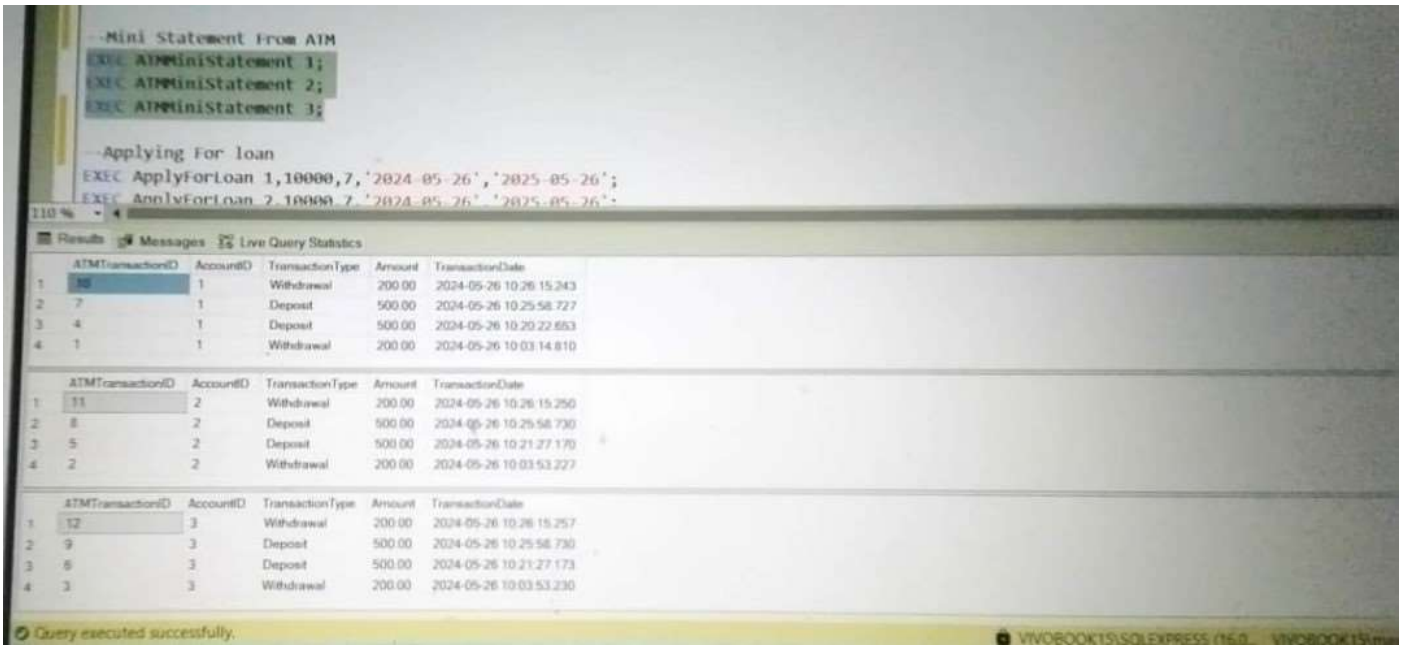
Testing And ValiDation:

Viewing Transaction History:

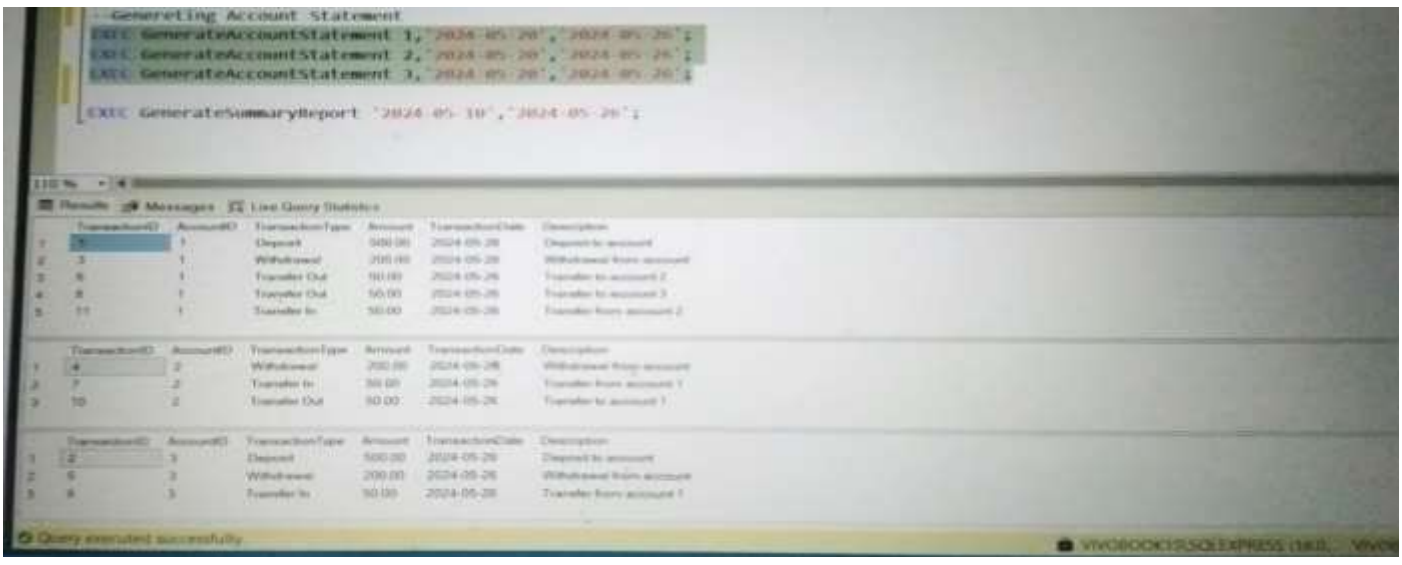


ATM Balance Enquiry:

ATM Mini Statement:



Generating Account Statement:



Conclusion:

This project is developed to nurture the needs of a user in a banking sector by embedding all the tasks of transactions taking place in a bank. Future version of this project will still be much enhanced than the current version. Writing and depositing checks are perhaps the most fundamental ways to move money in and out of a checking account, but advancements in technology have added ATM and debit card transactions. All banks have rules about how long it takes to access your deposits, how many debit card transactions you're allowed in a day, and how much cash you can withdraw from an ATM. Access to the balance in your checking account can also be limited by businesses that place holds on your funds. Banks are providing internet banking services also so that the customers can be attracted. By asking the bank employs we came to know that maximum numbers of internet bank account holders are youth and business man. Online banking is an innovative tool that is fast becoming a necessity. It is a successful strategic weapon for banks to remain profitable in a volatile and competitive marketplace of today. If proper training should be given to customer by the bank employs to open an account will be beneficial secondly the website should be made friendlier from where the first time customers can directly make and access their accounts.

Thus the Bank Management System it is developed and executed successfully.