**PROFESSIONAL PRACTICES**

**SUBMITTED BY:MUNEEB IQBAL(SP22-BCS-030)**

**PROJECT REPORT**

**Table of Contents**

1. **Introduction**
2. **Objectives**
3. **System Requirements**
4. **System Design**
5. **Implementation**
6. **Testing and Evaluation**
7. **Conclusion**
8. **FUTURE WORK**

**1. Introduction**

**1.1 Project Overview**

The Banking Management System is a comprehensive software solution designed to handle the operations of a bank. It provides an intuitive graphical user interface (GUI) developed using C#, coupled with a robust backend powered by SQL for database management. This system allows for various banking operations such as deposit, withdrawal, loan management, money transfer, and generation of bank statements.

**1.2 Purpose of the Project**

The purpose of this project is to streamline the banking processes, minimize human error, and provide a user-friendly interface for both bank staff and customers. It aims to enhance the efficiency and reliability of banking operations.

**2. Objectives**

**2.1 Primary Objectives**

* To develop a user-friendly GUI for banking operations using C#.
* To design and implement a robust SQL database for managing banking data.
* To facilitate essential banking operations such as deposits, withdrawals, loans, and money transfers.
* To provide functionalities for generating bank statements.

**2.2 Secondary Objectives**

* To ensure data integrity and security.
* To implement error handling and validation checks.
* To create a scalable system that can be extended with additional features.

**3. System Requirements**

**3.1 Hardware Requirements**

* Processor: Intel i5 or higher
* RAM: 8GB or higher
* Hard Disk: 500GB or more
* Display: 1024x768 resolution or higher

**3.2 Software Requirements**

* Operating System: Windows 10 or higher
* Development Environment: Visual Studio 2019 or higher
* Database Server: SQL Server 2019
* .NET Framework: Version 4.7 or higher

**4. System Design**

**4.1 Architecture**

The system follows a client-server architecture where the client application (developed in C#) interacts with the server (SQL database) to perform banking operations.

**4.2 Database Design**

The database consists of the following tables:

* **User**: Stores user information.
* **Account**: Manages account details.
* **Transaction**: Records all transactions including deposits and withdrawals.
* **Loan**: Manages loan details.
* **BankStatement**: Generates and stores bank statements.
* **Transfer**: Manages money transfer records.

**5. Implementation**

**5.1 Development Environment**

The system is developed using Visual Studio 2019 with C# for the front end and SQL Server 2019 for the backend database.

**5.2 Key Features**

**5.2.1 Deposit**

* **Description**: The deposit feature allows users to add money to their bank accounts. Users can enter the amount they wish to deposit, and the system updates the account balance accordingly.
* **Implementation Details**: The deposit function verifies the entered amount, updates the balance in the database, and records the transaction in the Transaction table.
* **Challenges and Solutions**: Ensuring accurate updates to the database and handling concurrent transactions were critical challenges. This was addressed by implementing transaction control in SQL and validating data inputs in C#.

**5.2.2 Withdraw**

* **Description**: The withdrawal feature enables users to withdraw money from their accounts. Users specify the amount to withdraw, and the system checks for sufficient funds before processing the transaction.
* **Implementation Details**: The withdrawal function checks the account balance, processes the transaction if funds are sufficient, updates the balance, and logs the transaction.
* **Challenges and Solutions**: Preventing overdrafts and ensuring real-time balance updates were key challenges. These were managed by implementing checks in the C# code and using SQL transactions to maintain data consistency.

**5.2.3 Loan Management**

* **Description**: This feature manages the entire lifecycle of a loan, from application to approval and repayment. Users can apply for loans, and bank staff can approve or reject applications.
* **Implementation Details**: The system maintains a Loan table to track loan details. It includes functions for applying, approving, and recording repayments.
* **Challenges and Solutions**: Handling complex loan calculations and ensuring timely updates to repayment schedules were significant challenges. These were addressed by implementing detailed business logic in C# and SQL.

**5.2.4 Money Transfer**

* **Description**: The money transfer feature allows users to transfer funds between accounts. Users specify the source and destination accounts and the amount to transfer.
* **Implementation Details**: The transfer function ensures both accounts exist, verifies sufficient funds in the source account, updates balances for both accounts, and records the transaction.
* **Challenges and Solutions**: Ensuring the atomicity of the transfer operation (both accounts are updated successfully or none) was a major challenge. This was achieved using SQL transactions.

**5.2.5 Bank Statement**

* **Description**: The bank statement feature generates detailed statements for users, showing all transactions over a specified period.
* **Implementation Details**: The system queries the Transaction table to retrieve and format the transaction history for the user.
* **Challenges and Solutions**: Ensuring the accuracy and readability of the generated statements was crucial. This was managed by designing clear SQL queries and formatting the output in C#.

**5.3 Implementation Challenges and Solutions**

* **Database Management**: Ensuring data integrity and preventing data loss were critical. The use of SQL transactions and careful database schema design helped address these concerns.
* **Concurrency Handling**: Managing multiple users accessing the system simultaneously posed challenges. Implementing proper locking mechanisms and transaction isolation levels in SQL helped mitigate these issues.
* **User Interface Design**: Creating a user-friendly and intuitive interface was essential. Iterative design and user feedback were used to refine the GUI.
* **Security**: Protecting sensitive user data was paramount. Security measures such as password hashing, input validation, and secure communication protocols were implemented.

**6. Testing and Evaluation**

**6.1 Testing Methodologies**

* **Unit Testing**: Individual components tested for functionality.
* **Integration Testing**: Combined components tested to ensure they work together.
* **System Testing**: Complete system tested in an environment similar to production.

**6.2 Evaluation**

The system was evaluated based on performance, usability, and security. It successfully met the defined objectives and requirements.

**7. Conclusion**

The Banking Management System developed in this project effectively automates various banking operations. It provides a user-friendly interface and robust backend, ensuring data integrity and security. The system can be further extended with additional features to enhance its functionality.

**8. Future Work**

* Implementation of online banking features.
* Integration with mobile banking applications.
* Enhanced security measures such as multi-factor authentication.
* Real-time transaction notifications.

**PROJECT CODE**

**C-SHARP CODE**

using MySql.Data.MySqlClient;

using System;

using System.Windows.Forms;

namespace WindowsFormsApplication2

{

public partial class Form1 : Form

{

string connectionString = "Server=localhost;Database=final\_projectt;Uid=root;Pwd=123456;";

public Form1()

{

InitializeComponent();

}

private void btnSignUp\_Click(object sender, EventArgs e)

{

try

{

string username = txtUsername.Text;

string firstName = txtFirstName.Text;

string lastName = txtLastName.Text;

string phone = txtPhone.Text;

string cnic = txtCNIC.Text;

string pin = txtPIN.Text; // Plain PIN text entered by user

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

// Start a transaction

using (MySqlTransaction transaction = connection.BeginTransaction())

{

try

{

// Insert user data into the User table

string insertUserQuery = "INSERT INTO User (Username, PIN, FirstName, LastName, Phone, CNIC) VALUES (@Username, @PIN, @FirstName, @LastName, @Phone, @CNIC)";

MySqlCommand cmd = new MySqlCommand(insertUserQuery, connection, transaction);

cmd.Parameters.AddWithValue("@Username", username);

cmd.Parameters.AddWithValue("@PIN", pin); // Store the PIN in plain text

cmd.Parameters.AddWithValue("@FirstName", firstName);

cmd.Parameters.AddWithValue("@LastName", lastName);

cmd.Parameters.AddWithValue("@Phone", phone);

cmd.Parameters.AddWithValue("@CNIC", cnic);

int rowsAffected = cmd.ExecuteNonQuery();

if (rowsAffected > 0)

{

// Insert account data into the Account table

string getInsertedUserIDQuery = "SELECT LAST\_INSERT\_ID()";

cmd = new MySqlCommand(getInsertedUserIDQuery, connection, transaction);

int userID = Convert.ToInt32(cmd.ExecuteScalar());

string insertAccountQuery = "INSERT INTO Account (UserID, Balance) VALUES (@UserID, @Balance)";

cmd = new MySqlCommand(insertAccountQuery, connection, transaction);

cmd.Parameters.AddWithValue("@UserID", userID);

cmd.Parameters.AddWithValue("@Balance", 0.0); // Initial balance

rowsAffected = cmd.ExecuteNonQuery();

if (rowsAffected > 0)

{

// Commit the transaction if both user and account records are successfully inserted

transaction.Commit();

MessageBox.Show("You have successfully created an account!");

login loginForm = new login();

loginForm.Show();

this.Hide();

}

else

{

// Rollback the transaction if there's an issue with inserting the account record

transaction.Rollback();

MessageBox.Show("Error: Unable to create account.");

}

}

else

{

MessageBox.Show("Error: Unable to create account.");

}

}

catch (MySqlException ex)

{

// Rollback the transaction if there's an exception

transaction.Rollback();

MessageBox.Show("MySQL Exception: " + ex.Message);

}

catch (Exception ex)

{

// Rollback the transaction if there's an exception

transaction.Rollback();

MessageBox.Show("An error occurred: " + ex.Message);

}

}

}

}

catch (MySqlException ex)

{

MessageBox.Show("MySQL Exception: " + ex.Message);

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message);

}

}

}

}

using System;

using System.Windows.Forms;

namespace WindowsFormsApplication2

{

public partial class MainForm : Form

{

private int UserID; // Assuming you have access to the UserID

public MainForm(int userID)

{

InitializeComponent();

UserID = userID; // Set the UserID

}

private void btnDeposit\_Click(object sender, EventArgs e)

{

// Open a new form for deposit functionality

DepositForm depositForm = new DepositForm(UserID); // Pass UserID to the constructor

depositForm.Show();

}

private void btnWithdraw\_Click(object sender, EventArgs e)

{

// Open a new form for withdrawal functionality

WithdrawForm withdrawForm = new WithdrawForm(UserID); // Pass UserID to the constructor

withdrawForm.Show();

}

private void btnTransfer\_Click(object sender, EventArgs e)

{

// Open a new form for transfer money functionality

TransferForm transferForm = new TransferForm(UserID); // Pass UserID to the constructor

transferForm.Show();

}

private void btnTakeLoan\_Click(object sender, EventArgs e)

{

// Open a new form for taking a loan

LoanForm loanForm = new LoanForm(UserID); // Pass UserID to the constructor

loanForm.Show();

}

private void btnBankStatement\_Click(object sender, EventArgs e)

{

// Open a new form for displaying bank statement

BankStatementForm bankStatementForm = new BankStatementForm(UserID); // Pass UserID to the constructor

bankStatementForm.Show();

}

}

}

using MySql.Data.MySqlClient;

using System;

using System.Windows.Forms;

namespace WindowsFormsApplication2

{

public partial class login : Form

{

string connectionString = "Server=localhost;Database=final\_projectt;Uid=root;Pwd=123456;";

public login()

{

InitializeComponent();

}

private void btnLogin\_Click(object sender, EventArgs e)

{

try

{

string username = txtUsername.Text;

string pin = txtPIN.Text;

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

// Query to check if the entered username and PIN match any record in the User table

string query = "SELECT UserID FROM User WHERE Username = @Username AND PIN = @PIN";

MySqlCommand cmd = new MySqlCommand(query, connection);

cmd.Parameters.AddWithValue("@Username", username);

cmd.Parameters.AddWithValue("@PIN", pin);

object result = cmd.ExecuteScalar();

if (result != null)

{

int userID = Convert.ToInt32(result);

MessageBox.Show("Login successful!");

// Open the main application form

MainForm mainForm = new MainForm(userID);

mainForm.Show();

this.Hide();

}

else

{

MessageBox.Show("Invalid username or PIN.");

}

}

}

catch (MySqlException ex)

{

MessageBox.Show("MySQL Exception: " + ex.Message);

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message);

}

}

}

}

using System;

using System.Windows.Forms;

using MySql.Data.MySqlClient;

namespace WindowsFormsApplication2

{

public partial class DepositForm : Form

{

private int UserID;

public DepositForm(int userID)

{

InitializeComponent();

UserID = userID;

}

private void btnGo\_Click(object sender, EventArgs e)

{

decimal depositAmount;

if (!string.IsNullOrEmpty(txtAmount.Text) && decimal.TryParse(txtAmount.Text, out depositAmount))

{

string connectionString = "Server=localhost;Database=final\_projectt;Uid=root;Pwd=123456;";

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

try

{

connection.Open();

string updateQuery = "UPDATE Account SET Balance = Balance + @DepositAmount WHERE UserID = @UserID";

MySqlCommand cmd = new MySqlCommand(updateQuery, connection);

cmd.Parameters.AddWithValue("@DepositAmount", depositAmount);

cmd.Parameters.AddWithValue("@UserID", UserID);

int rowsAffected = cmd.ExecuteNonQuery();

if (rowsAffected > 0)

{

// Log deposit in BankStatement table

string insertStatementQuery = @"

INSERT INTO BankStatement (AccountID, Date, Details)

VALUES ((SELECT AccountID FROM Account WHERE UserID = @UserID), NOW(), CONCAT('Deposit: ', @DepositAmount))";

MySqlCommand logCmd = new MySqlCommand(insertStatementQuery, connection);

logCmd.Parameters.AddWithValue("@DepositAmount", depositAmount);

logCmd.Parameters.AddWithValue("@UserID", UserID);

logCmd.ExecuteNonQuery();

MessageBox.Show("Deposit successful!");

this.DialogResult = DialogResult.OK;

this.Close();

}

else

{

MessageBox.Show("Error: Unable to deposit money.");

}

}

catch (MySqlException ex)

{

MessageBox.Show("MySQL Exception: " + ex.Message);

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message);

}

}

}

else

{

MessageBox.Show("Please enter a valid deposit amount.");

}

}

private void txtAmount\_TextChanged(object sender, EventArgs e)

{

}

}

}

using System;

using System.Windows.Forms;

using MySql.Data.MySqlClient;

namespace WindowsFormsApplication2

{

public partial class WithdrawForm : Form

{

private int UserID;

public WithdrawForm(int userID)

{

InitializeComponent();

UserID = userID;

}

private void btnGo\_Click(object sender, EventArgs e)

{

decimal withdrawAmount;

if (!string.IsNullOrEmpty(txtAmount.Text) && decimal.TryParse(txtAmount.Text, out withdrawAmount))

{

string connectionString = "Server=localhost;Database=final\_projectt;Uid=root;Pwd=123456;";

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

try

{

connection.Open();

string checkBalanceQuery = "SELECT Balance FROM Account WHERE UserID = @UserID";

MySqlCommand checkBalanceCmd = new MySqlCommand(checkBalanceQuery, connection);

checkBalanceCmd.Parameters.AddWithValue("@UserID", UserID);

decimal currentBalance = Convert.ToDecimal(checkBalanceCmd.ExecuteScalar());

if (withdrawAmount > currentBalance)

{

MessageBox.Show("Insufficient balance. Cannot withdraw more than available.");

return;

}

string updateQuery = "UPDATE Account SET Balance = Balance - @WithdrawAmount WHERE UserID = @UserID";

MySqlCommand cmd = new MySqlCommand(updateQuery, connection);

cmd.Parameters.AddWithValue("@WithdrawAmount", withdrawAmount);

cmd.Parameters.AddWithValue("@UserID", UserID);

int rowsAffected = cmd.ExecuteNonQuery();

if (rowsAffected > 0)

{

// Log withdrawal in BankStatement table

string insertStatementQuery = @"

INSERT INTO BankStatement (AccountID, Date, Details)

VALUES ((SELECT AccountID FROM Account WHERE UserID = @UserID), NOW(), CONCAT('Withdraw: ', @WithdrawAmount))";

MySqlCommand logCmd = new MySqlCommand(insertStatementQuery, connection);

logCmd.Parameters.AddWithValue("@WithdrawAmount", withdrawAmount);

logCmd.Parameters.AddWithValue("@UserID", UserID);

logCmd.ExecuteNonQuery();

MessageBox.Show("Withdrawal successful!");

this.DialogResult = DialogResult.OK;

this.Close();

}

else

{

MessageBox.Show("Error: Unable to withdraw money.");

}

}

catch (MySqlException ex)

{

MessageBox.Show("MySQL Exception: " + ex.Message);

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message);

}

}

}

else

{

MessageBox.Show("Please enter a valid withdrawal amount.");

}

}

}

}

using System;

using System.Windows.Forms;

using MySql.Data.MySqlClient;

namespace WindowsFormsApplication2

{

public partial class TransferForm : Form

{

private int UserID;

public TransferForm(int userID)

{

InitializeComponent();

UserID = userID;

}

private void btnGo\_Click(object sender, EventArgs e)

{

decimal transferAmount;

int toAccountID;

if (!string.IsNullOrEmpty(txtAmount.Text) && decimal.TryParse(txtAmount.Text, out transferAmount) &&

!string.IsNullOrEmpty(txtToAccountID.Text) && int.TryParse(txtToAccountID.Text, out toAccountID))

{

string connectionString = "Server=localhost;Database=final\_projectt;Uid=root;Pwd=123456;";

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

try

{

connection.Open();

// Check if the transfer amount is greater than the current balance

string checkBalanceQuery = "SELECT Balance FROM Account WHERE UserID = @UserID";

MySqlCommand checkBalanceCmd = new MySqlCommand(checkBalanceQuery, connection);

checkBalanceCmd.Parameters.AddWithValue("@UserID", UserID);

decimal currentBalance = Convert.ToDecimal(checkBalanceCmd.ExecuteScalar());

if (transferAmount > currentBalance)

{

MessageBox.Show("Insufficient balance. Cannot transfer more than available.");

return;

}

// Update the balance for the sender's account

string updateSenderQuery = "UPDATE Account SET Balance = Balance - @TransferAmount WHERE UserID = @UserID";

MySqlCommand updateSenderCmd = new MySqlCommand(updateSenderQuery, connection);

updateSenderCmd.Parameters.AddWithValue("@TransferAmount", transferAmount);

updateSenderCmd.Parameters.AddWithValue("@UserID", UserID);

updateSenderCmd.ExecuteNonQuery();

// Update the balance for the recipient's account

string updateRecipientQuery = "UPDATE Account SET Balance = Balance + @TransferAmount WHERE AccountID = @ToAccountID";

MySqlCommand updateRecipientCmd = new MySqlCommand(updateRecipientQuery, connection);

updateRecipientCmd.Parameters.AddWithValue("@TransferAmount", transferAmount);

updateRecipientCmd.Parameters.AddWithValue("@ToAccountID", toAccountID);

updateRecipientCmd.ExecuteNonQuery();

// Log transfer in Transaction table

string insertTransactionQuery = @"

INSERT INTO Transaction (FromAccountID, ToAccountID, Amount, Date)

VALUES ((SELECT AccountID FROM Account WHERE UserID = @UserID), @ToAccountID, @TransferAmount, NOW())";

MySqlCommand insertTransactionCmd = new MySqlCommand(insertTransactionQuery, connection);

insertTransactionCmd.Parameters.AddWithValue("@TransferAmount", transferAmount);

insertTransactionCmd.Parameters.AddWithValue("@UserID", UserID);

insertTransactionCmd.Parameters.AddWithValue("@ToAccountID", toAccountID);

insertTransactionCmd.ExecuteNonQuery();

// Log the transfer details in the BankStatement table for both accounts

string insertStatementQuery = @"

INSERT INTO BankStatement (AccountID, Date, Details)

VALUES

((SELECT AccountID FROM Account WHERE UserID = @UserID), NOW(), CONCAT('Transferred Amount: ', @TransferAmount, ' to AccountID: ', @ToAccountID)),

(@ToAccountID, NOW(), CONCAT('Received Amount: ', @TransferAmount, ' from AccountID: ', (SELECT AccountID FROM Account WHERE UserID = @UserID)))";

MySqlCommand insertStatementCmd = new MySqlCommand(insertStatementQuery, connection);

insertStatementCmd.Parameters.AddWithValue("@TransferAmount", transferAmount);

insertStatementCmd.Parameters.AddWithValue("@UserID", UserID);

insertStatementCmd.Parameters.AddWithValue("@ToAccountID", toAccountID);

insertStatementCmd.ExecuteNonQuery();

MessageBox.Show("Transfer successful!");

this.DialogResult = DialogResult.OK;

this.Close();

}

catch (MySqlException ex)

{

MessageBox.Show("MySQL Exception: " + ex.Message);

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message);

}

}

}

else

{

MessageBox.Show("Please enter valid transfer details.");

}

}

}

}

using MySql.Data.MySqlClient;

using System;

using System.Data;

using System.Windows.Forms;

namespace WindowsFormsApplication2

{

public partial class BankStatementForm : Form

{

private int UserID;

public BankStatementForm(int userID)

{

InitializeComponent();

UserID = userID;

LoadBankStatement();

}

private void LoadBankStatement()

{

try

{

string connectionString = "Server=localhost;Database=final\_projectt;Uid=root;Pwd=123456;";

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

string selectStatementQuery = @"

SELECT Date, Details

FROM BankStatement

WHERE AccountID = (SELECT AccountID FROM Account WHERE UserID = @UserID)

ORDER BY Date";

MySqlCommand cmd = new MySqlCommand(selectStatementQuery, connection);

cmd.Parameters.AddWithValue("@UserID", UserID);

DataTable dt = new DataTable();

dt.Load(cmd.ExecuteReader());

listViewBankStatement.View = View.Details;

listViewBankStatement.Columns.Add("Date", 150);

listViewBankStatement.Columns.Add("Details", 300);

foreach (DataRow row in dt.Rows)

{

ListViewItem item = new ListViewItem(row["Date"].ToString());

item.SubItems.Add(row["Details"].ToString());

listViewBankStatement.Items.Add(item);

}

}

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message);

}

}

}

}

using MySql.Data.MySqlClient;

using System;

using System.Windows.Forms;

namespace WindowsFormsApplication2

{

public partial class LoanForm : Form

{

private int UserID; // User's ID

private int AccountID; // Account ID associated with the user

private decimal loanAmount; // Amount of loan taken by the user

private decimal outstandingLoanBalance;

string connectionString = "Server=localhost;Database=final\_projectt;Uid=root;Pwd=123456;";

public LoanForm(int userID)

{

InitializeComponent();

UserID = userID;

GetAccountID();

GetOutstandingLoanBalance();

}

private void GetAccountID()

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

try

{

connection.Open();

string query = "SELECT AccountID FROM Account WHERE UserID = @UserID";

MySqlCommand cmd = new MySqlCommand(query, connection);

cmd.Parameters.AddWithValue("@UserID", UserID);

AccountID = Convert.ToInt32(cmd.ExecuteScalar());

}

catch (Exception ex)

{

MessageBox.Show("Error getting Account ID: " + ex.Message);

}

}

}

private void GetOutstandingLoanBalance()

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

try

{

connection.Open();

string query = "SELECT SUM(LoanAmount) FROM Loan WHERE AccountID = @AccountID AND Repaid = 0";

MySqlCommand cmd = new MySqlCommand(query, connection);

cmd.Parameters.AddWithValue("@AccountID", AccountID);

var result = cmd.ExecuteScalar();

if (result != null && result != DBNull.Value)

{

outstandingLoanBalance = Convert.ToDecimal(result);

}

else

{

outstandingLoanBalance = 0;

}

}

catch (Exception ex)

{

MessageBox.Show("Error getting outstanding loan balance: " + ex.Message);

}

}

}

private void Go1\_Click(object sender, EventArgs e)

{

if (decimal.TryParse(amount1.Text, out loanAmount) && loanAmount <= 5000)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

using (MySqlTransaction transaction = connection.BeginTransaction())

{

try

{

// Insert loan record into Loan table

string insertLoanQuery = "INSERT INTO Loan (AccountID, LoanAmount, InterestRate, DueDate, Repaid) VALUES (@AccountID, @LoanAmount, @InterestRate, @DueDate, 0)";

MySqlCommand cmd = new MySqlCommand(insertLoanQuery, connection, transaction);

cmd.Parameters.AddWithValue("@AccountID", AccountID);

cmd.Parameters.AddWithValue("@LoanAmount", loanAmount);

cmd.Parameters.AddWithValue("@InterestRate", 5.0); // Example interest rate

cmd.Parameters.AddWithValue("@DueDate", DateTime.Now.AddMonths(1)); // Example due date

cmd.ExecuteNonQuery();

// Update the balance in the Account table

string updateBalanceQuery = "UPDATE Account SET Balance = Balance + @LoanAmount WHERE AccountID = @AccountID";

cmd = new MySqlCommand(updateBalanceQuery, connection, transaction);

cmd.Parameters.AddWithValue("@LoanAmount", loanAmount);

cmd.Parameters.AddWithValue("@AccountID", AccountID);

cmd.ExecuteNonQuery();

// Insert record into BankStatement

string insertStatementQuery = "INSERT INTO BankStatement (AccountID, Date, Details) VALUES (@AccountID, @Date, @Details)";

cmd = new MySqlCommand(insertStatementQuery, connection, transaction);

cmd.Parameters.AddWithValue("@AccountID", AccountID);

cmd.Parameters.AddWithValue("@Date", DateTime.Now);

cmd.Parameters.AddWithValue("@Details", $"Loan Granted: {loanAmount}");

cmd.ExecuteNonQuery();

// Update the outstanding loan balance

outstandingLoanBalance += loanAmount;

transaction.Commit();

MessageBox.Show("Loan granted successfully!");

}

catch (MySqlException ex)

{

transaction.Rollback();

MessageBox.Show("MySQL Exception: " + ex.Message);

}

catch (Exception ex)

{

transaction.Rollback();

MessageBox.Show("An error occurred: " + ex.Message);

}

}

}

}

else

{

MessageBox.Show("Invalid loan amount. It should be less than or equal to 5000.");

}

}

private void Go2\_Click(object sender, EventArgs e)

{

decimal repaymentAmount;

if (decimal.TryParse(amount2.Text, out repaymentAmount) && repaymentAmount <= outstandingLoanBalance)

{

using (MySqlConnection connection = new MySqlConnection(connectionString))

{

connection.Open();

using (MySqlTransaction transaction = connection.BeginTransaction())

{

try

{

// Get the current account balance

string getBalanceQuery = "SELECT Balance FROM Account WHERE AccountID = @AccountID";

MySqlCommand cmd = new MySqlCommand(getBalanceQuery, connection, transaction);

cmd.Parameters.AddWithValue("@AccountID", AccountID);

decimal currentBalance = Convert.ToDecimal(cmd.ExecuteScalar());

// Check if the account has sufficient balance for the repayment

if (repaymentAmount > currentBalance)

{

MessageBox.Show("Insufficient balance for the repayment.");

return;

}

// Update the balance in the Account table

string updateBalanceQuery = "UPDATE Account SET Balance = Balance - @RepaymentAmount WHERE AccountID = @AccountID";

cmd = new MySqlCommand(updateBalanceQuery, connection, transaction);

cmd.Parameters.AddWithValue("@RepaymentAmount", repaymentAmount);

cmd.Parameters.AddWithValue("@AccountID", AccountID);

cmd.ExecuteNonQuery();

// Mark loan as repaid if the entire loan amount is repaid

string repayLoanQuery = "UPDATE Loan SET Repaid = 1 WHERE AccountID = @AccountID AND LoanAmount = @RepaymentAmount AND Repaid = 0 LIMIT 1";

cmd = new MySqlCommand(repayLoanQuery, connection, transaction);

cmd.Parameters.AddWithValue("@AccountID", AccountID);

cmd.Parameters.AddWithValue("@RepaymentAmount", repaymentAmount);

cmd.ExecuteNonQuery();

// Insert record into BankStatement

string insertStatementQuery = "INSERT INTO BankStatement (AccountID, Date, Details) VALUES (@AccountID, @Date, @Details)";

cmd = new MySqlCommand(insertStatementQuery, connection, transaction);

cmd.Parameters.AddWithValue("@AccountID", AccountID);

cmd.Parameters.AddWithValue("@Date", DateTime.Now);

cmd.Parameters.AddWithValue("@Details", $"Loan Repaid: {repaymentAmount}");

cmd.ExecuteNonQuery();

// Update the outstanding loan balance

outstandingLoanBalance -= repaymentAmount;

transaction.Commit();

MessageBox.Show("Loan repaid successfully!");

}

catch (MySqlException ex)

{

transaction.Rollback();

MessageBox.Show("MySQL Exception: " + ex.Message);

}

catch (Exception ex)

{

transaction.Rollback();

MessageBox.Show("An error occurred: " + ex.Message);

}

}

}

}

else

{

MessageBox.Show("Invalid repayment amount. It should be less than or equal to the outstanding loan balance.");

}

}

}

}

**SQL CODE**

CREATE DATABASE final\_projectt;

use final\_projectt;

CREATE TABLE User (

UserID INT AUTO\_INCREMENT PRIMARY KEY,

Username VARCHAR(50) UNIQUE NOT NULL CHECK (Username REGEXP '^[A-Za-z0-9]+$'),

PIN VARCHAR(60) NOT NULL, -- Use hashed PINs for security

FirstName VARCHAR(50) NOT NULL CHECK (FirstName REGEXP '^[A-Za-z]+$'),

LastName VARCHAR(50) NOT NULL CHECK (LastName REGEXP '^[A-Za-z]+$'),

Phone VARCHAR(15) NOT NULL CHECK (Phone REGEXP '^[0-9]{11}$'),

CNIC VARCHAR(15) UNIQUE NOT NULL CHECK (CNIC REGEXP '^[0-9]{5}-[0-9]{7}-[0-9]$')

);

CREATE TABLE Account (

AccountID INT AUTO\_INCREMENT PRIMARY KEY,

UserID INT NOT NULL,

Balance DECIMAL(10, 2) DEFAULT 0.0,

FOREIGN KEY (UserID) REFERENCES User(UserID)

);

CREATE TABLE Transaction (

TransactionID INT AUTO\_INCREMENT PRIMARY KEY,

FromAccountID INT NOT NULL,

ToAccountID INT NOT NULL,

Amount DECIMAL(10, 2) NOT NULL,

Date DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (FromAccountID) REFERENCES Account(AccountID),

FOREIGN KEY (ToAccountID) REFERENCES Account(AccountID)

);

CREATE TABLE Loan (

LoanID INT AUTO\_INCREMENT PRIMARY KEY,

AccountID INT NOT NULL,

LoanAmount DECIMAL(10, 2) NOT NULL,

InterestRate FLOAT NOT NULL,

DueDate DATE NOT NULL,

FOREIGN KEY (AccountID) REFERENCES Account(AccountID)

);

CREATE TABLE BankStatement (

StatementID INT AUTO\_INCREMENT PRIMARY KEY,

AccountID INT NOT NULL,

Date DATE NOT NULL,

Details VARCHAR(255) NOT NULL,

FOREIGN KEY (AccountID) REFERENCES Account(AccountID)

);

ALTER TABLE Loan ADD COLUMN Repaid TINYINT(1) NOT NULL DEFAULT 0;