ASSIGNMENT – 4

NAME : HEMANTH KUMAR.B

REG NO : 192324238

COURSE : DATABASE MANAGEMENT SYSTEM

COURSE CODE : CSA0593

**SCENERIO:**

Develop a system for an Online Banking Application. -Model tables for accounts, customers, transactions, and transfers. Write stored procedures to handle,Deposit and Withdrawal transactions. -Money transfers between accounts, ensuring proper handling of concurrency and transaction isolation. -Implement ACID (Atomicity, Consistency, Isolation, Durability) principles in transactions.

**Database Design for Online Banking Application**

**1. Customers Table**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| customer\_id | INT | Primary Key, unique identifier for each customer |
| customer\_name | VARCHAR(100) | Name of the customer |
| customer\_address | VARCHAR(255) | Address of the customer |
| customer\_email | VARCHAR(100) | Email address of the customer |
| customer\_phone | VARCHAR(20) | Phone number of the customer |

**2. Accounts Table**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| account\_id | INT | Primary Key, unique identifier for each account |
| customer\_id | INT | Foreign key referencing the customer ID |
| account\_type | VARCHAR(50) | Type of account (e.g., savings, checking) |
| account\_balance | DECIMAL(10,2) | Current balance of the account |
| account\_creation\_date | DATE | Date the account was created |

**3. Transactions Table**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| transaction\_id | INT | Primary Key, unique identifier for each transaction |
| account\_id | INT | Foreign key referencing the account ID |
| transaction\_type | VARCHAR(50) | Type of transaction (e.g., deposit, withdrawal, transfer) |
| transaction\_amount | DECIMAL(10,2) | Amount of the transaction |
| transaction\_date | DATE | Date of the transaction |

**Stored Procedures**

**1. Deposit**

SQL

CREATE PROCEDURE Deposit

@account\_id INT,

@amount DECIMAL(10,2)

AS

BEGIN

BEGIN TRANSACTION;

UPDATE Accounts

SET account\_balance = account\_balance + @amount

WHERE account\_id = @account\_id;

INSERT INTO Transactions (account\_id, transaction\_type, transaction\_amount, transaction\_date)

VALUES (@account\_id, 'Deposit', @amount, GETDATE());

COMMIT TRANSACTION;

END

**2. Withdrawal**

SQL

CREATE PROCEDURE Withdraw

@account\_id INT,

@amount DECIMAL(10,2)

AS

BEGIN

BEGIN TRANSACTION;

DECLARE @current\_balance DECIMAL(10,2);

SELECT @current\_balance = account\_balance FROM Accounts WHERE account\_id = @account\_id;

IF @current\_balance >= @amount

BEGIN

UPDATE Accounts

SET account\_balance = account\_balance - @amount

WHERE account\_id = @account\_id;

INSERT INTO Transactions (account\_id, transaction\_type, transaction\_amount, transaction\_date)

VALUES (@account\_id, 'Withdrawal', @amount, GETDATE());

END

ELSE

BEGIN

ROLLBACK TRANSACTION;

RAISERROR('Insufficient funds', 16, 1);

END

COMMIT TRANSACTION;

END

**3. Transfer**

SQL

CREATE PROCEDURE Transfer

@from\_account\_id INT,

@to\_account\_id INT,

@amount DECIMAL(10,2)

AS

BEGIN

BEGIN TRANSACTION;

DECLARE @from\_balance DECIMAL(10,2);

SELECT @from\_balance = account\_balance FROM Accounts WHERE account\_id = @from\_account\_id;

IF @from\_balance >= @amount

BEGIN

UPDATE Accounts

SET account\_balance = account\_balance - @amount

WHERE account\_id = @from\_account\_id;

UPDATE Accounts

SET account\_balance = account\_balance + @amount

WHERE account\_id = @to\_account\_id;

INSERT INTO Transactions (account\_id, transaction\_type, transaction\_amount, transaction\_date)

VALUES (@from\_account\_id, 'Transfer', @amount, GETDATE());

INSERT INTO Transactions (account\_id, transaction\_type, transaction\_amount, transaction\_date)

VALUES (@to\_account\_id, 'Transfer', @amount, GETDATE());

END

ELSE

BEGIN

ROLLBACK TRANSACTION;

RAISERROR('Insufficient funds', 16, 1);

END

COMMIT TRANSACTION;

END

**ACID Principles Implementation**

* **Atomicity:** Each transaction is treated as an indivisible unit. Either all operations within the transaction are completed successfully, or none of them are.
* **Consistency:** Transactions must preserve the database's integrity constraints.
* **Isolation:** Transactions are executed independently, as if they were the only transactions running.
* **Durability:** Once a transaction is committed, its effects are permanent and will survive system failures.

By using transactions and appropriate error handling, we can ensure the consistency and reliability of the online banking system.