Final Project
(Banking System)

Name: Utkarsh Tripathi **Student Id:** CT_CSI_SQ_5569

Domain: SQL **College Name:** Chandigarh University

1. Introduction

The Banking System Database is a centralized, robust, and scalable solution designed to manage core banking operations in a structured and efficient manner. In modern banking, the need for digital transformation and data-driven decision-making is more important than ever. Traditional banking operations often rely on manual processes or scattered data storage, leading to redundancy, inconsistency, and limited reporting capabilities. To address these challenges, this project provides a relational database system that streamlines the entire workflow of a banking environment.

This database has been architected to cover all essential functional areas of a typical banking system:

- Customer Information Management: Storing and retrieving customer data such as personal details, contact information, and unique identifiers. Each customer can be linked to multiple accounts, ensuring a one-to-many relationship between customers and their accounts.

- Account Management: Managing different types of accounts (Savings, Current) along with their balances, opening dates, and associated branches. The system ensures minimum balance constraints, prevents invalid data entry using CHECK constraints, and maintains data integrity using foreign key relationships.

- Transaction Handling: Recording every transaction in detail, including deposits, withdrawals, and inter-account transfers. Each transaction is logged with the amount, type, timestamp, and remarks, ensuring a complete financial history for audit and reporting purposes. Triggers have been implemented to monitor unusual activities like low balance alerts and automatic withdrawal auditing.

- Loan Management: Allowing the tracking of loans issued to customers, along with loan amounts, interest rates, and approval statuses. This enables the bank to manage pending, approved, and closed loans efficiently and generate branch-wise or customer-wise loan reports.

- Branch-Level Operations: Supporting multiple branches and allowing each branch to maintain its own customers, accounts, employees, and loans. The system also generates branch-level summaries such as the total deposits, number of accounts, and total loan amounts for managerial decision-making.

- Employee Management: Although this project primarily focuses on customer-centric operations, it also supports employee details such as their assigned branch, role (Teller, Manager, Admin), and contact details. This helps in extending the system to role-based access in future enhancements.

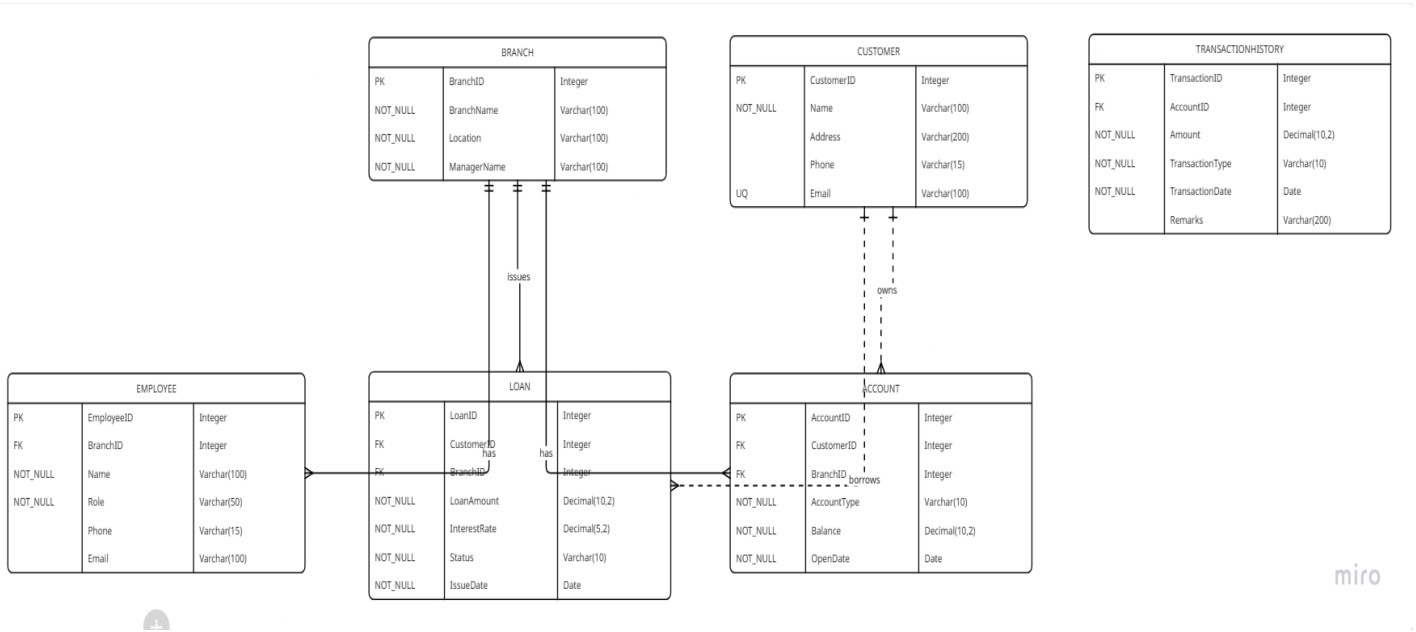
2. Problem Statement

In the modern financial sector, banks face significant challenges in managing large volumes of data while ensuring accuracy, security, and efficiency. Traditional manual systems or outdated databases lead to redundant data, lack of consistency, slow retrieval times, and limited reporting capabilities. This creates operational bottlenecks that impact both customer experience and managerial decision-making.

Some of the key problems in a traditional banking setup include:

- Fragmented customer records, making it difficult to link customers with their multiple accounts or loans.
- Manual account management, which increases the chances of errors in balance tracking and transaction history.
- Unstructured transaction records, leading to challenges in auditing and reconciliation.
- Difficulty in tracking loans and their repayment statuses, causing delays in approvals and monitoring.
- Limited branch-level insights, making it hard for management to analyze branch performance, total deposits, or loan disbursement statistics.
- No automation for alerts, such as low balance notifications or withdrawal audit logs.

3. ER-Diagram



4. Stored Procedure and Triggers

Core Stored Procedures

- CreateCustomer → Creates a new customer

SQLQuery1.sql - AS...LAPTOP\utkar (55))*

```
-- See Loan Status Report
EXEC GetLoanStatusReport;

-- See Low Balance Trigger in action
UPDATE Account SET Balance = 500 WHERE AccountID = 1;
GO


EXEC CreateCustomer
    @Name = 'Pravriti',
    @Address = 'Mumbai',
    @Phone = '9876543210',
    @Email = 'Prav.@email.com';

SELECT * FROM Customer WHERE Email='prav.@email.com';
```

110 %

Results | Messages

	CustomerID	Name	Address	Phone	Email
1	3	Pravriti	Mumbai	9876543210	Prav.@email.com

- OpenAccount → Opens a new account with initial deposit
- DepositMoney → Deposits money into an account

```
SELECT TOP 5 * FROM Account;
EXEC DepositMoney
    @AccountID = 1,
    @Amount = 2000;
SELECT * FROM Account WHERE AccountID=1;
SELECT TOP 5 * FROM TransactionHistory WHERE AccountID=1 ORDER BY TransactionDate DESC;
```

110 %

Results | Messages

	AccountID	CustomerID	BranchID	AccountType	Balance	OpenDate
1	1	1	1	Savings	500.00	2025-07-22
2	2	2	2	Current	18000.00	2025-07-22

	AccountID	CustomerID	BranchID	AccountType	Balance	OpenDate
1	1	1	1	Savings	2500.00	2025-07-22

	TransactionID	AccountID	Amount	TransactionType	TransactionDate	Remarks
1	7	1	2000.00	Deposit	2025-07-22 00:21:29.103	Money Deposited
2	5	1	3000.00	Transfer	2025-07-22 00:05:45.413	Transferred to AccountID 2
3	4	1	500.00	Withdraw	2025-07-22 00:05:45.407	Money Withdrawn
4	3	1	2000.00	Deposit	2025-07-22 00:05:45.403	Money Deposited
5	1	1	10000.00	Deposit	2025-07-22 00:05:45.387	Initial Deposit

- WithdrawMoney → Withdraws money (checks balance before withdrawal)

- TransferAmount → Transfers money between accounts (with transaction safety)
- ViewTransactionHistory → Displays all transactions for a given account

Reporting Stored Procedures

- GetBranchSummary → Shows branch-wise accounts, total deposits & loans
- GetLoanStatusReport → Shows total loans grouped by status

Triggers

- trg_LowBalance → Prints ⚠️ Low Balance Alert when balance < ₹1000
- trg_AuditWithdraw → Logs all withdrawals into a separate WithdrawalAudit table

```
EXEC WithdrawMoney @AccountID=1, @Amount=500;
EXEC TransferAmount @FromAccountID=1, @ToAccountID=2, @Amount=1000;
EXEC ViewTransactionHistory @AccountID=1;
```

110 %

Results | Messages

	TransactionID	AccountID	Amount	TransactionType	TransactionDate	Remarks
1	9	1	1000.00	Transfer	2025-07-22 00:23:18.270	Transferred to AccountID 2
2	8	1	500.00	Withdraw	2025-07-22 00:23:04.470	Money Withdrawn
3	7	1	2000.00	Deposit	2025-07-22 00:21:29.103	Money Deposited
4	5	1	3000.00	Transfer	2025-07-22 00:05:45.413	Transferred to AccountID 2
5	4	1	500.00	Withdraw	2025-07-22 00:05:45.407	Money Withdrawn
6	3	1	2000.00	Deposit	2025-07-22 00:05:45.403	Money Deposited
7	1	1	10000.00	Deposit	2025-07-22 00:05:45.387	Initial Deposit

5. Security Measures

The Banking System Database implements multiple layers of security and integrity checks to ensure accurate, consistent, and reliable banking operations.

1. Data Integrity with Constraints

- Primary Keys uniquely identify each record in all tables.
- Foreign Keys maintain referential integrity (e.g., an Account must belong to a valid Customer and Branch).
- CHECK Constraints ensure valid entries (e.g., only Savings or Current allowed as AccountType, no negative balances).
- Unique Constraints on customer emails prevent duplicate registrations.

2. Transaction Safety

- Money transfer operations use BEGIN TRANSACTION, COMMIT, and ROLLBACK to maintain consistency.
- If any step of a transfer fails, the entire operation is rolled back, avoiding partial updates.

3. Triggers for Automated Monitoring

- Low Balance Trigger: Alerts whenever an account balance goes below ₹1000.

- Withdrawal Audit Trigger: Logs every withdrawal into an WithdrawalAudit table for fraud monitoring.

4. Restricted Access (Future Ready)

- Supports role-based access control (e.g., Admin, Teller, Viewer).
- Future enhancement: Only stored procedures will handle data modifications, preventing direct table edits.

5. Scalability & Modularity

- Each module (Customer, Account, Transaction, Branch, Loan) is independent but linked, ensuring easier maintenance and extension.

6. Sample Queries and Outputs

- **View Top 5 Customers (By Balance)**
  - SELECT * FROM TopCustomers;

```
SELECT * FROM TopCustomers;
```

110 %

Results | Messages

	CustomerID	Name	TotalBalance
1	2	Arundhati Singh	19000.00
2	1	Utkarsh Tripathi	1000.00

- **Monthly Transaction Summary**
  - SELECT * FROM MonthlyTransactionSummary;

```
SELECT * FROM MonthlyTransactionSummary;
```

110 %

Results | Messages

	AccountID	Month	TotalTransactions	TotalDeposited	TotalWithdrawn
1	1	2025-07	12	18000.00	2000.00
2	2	2025-07	4	20000.00	0.00
3	3	2025-07	1	10000.00	0.00
4	4	2025-07	1	15000.00	0.00

- **Branch Summary Report**
  - EXEC GetBranchSummary;

```
EXEC GetBranchSummary;
```

110 %

Results | Messages

	BranchID	BranchName	TotalAccounts	TotalDeposits	TotalLoansIssued
1	1	Lucknow Main	2	22000.00	2
2	2	Delhi Central	2	70000.00	2
3	3	Lucknow Main	0	NULL	0
4	4	Delhi Central	0	NULL	0

- **Loan Status Report**
  - EXEC GetLoanStatusReport;

```
EXEC GetLoanStatusReport;
```

110 %

Results | Messages

	Status	TotalLoans	TotalAmount
1	Approved	2	100000.00
2	Pending	2	150000.00

7. Future Enhancements

While the current Banking System Database efficiently manages core banking operations, it can be further enhanced to provide more automation, security, and analytical capabilities.

1. Role-Based Access Control

- Implement a user authentication system with roles like Admin, Teller, and Viewer.
- Restrict direct table access and allow only stored procedure-based interactions for better security.

2. Automated Interest Calculations

- Add scheduled jobs or triggers to calculate monthly interest on Savings accounts.
- Automate loan EMI calculations and due date reminders.

3. Credit/Debit Card Module

- Extend the schema to manage card details, payments, and card transactions linked to accounts.

4. Integration with Web & Mobile Applications

- Expose REST APIs to connect this database with a frontend dashboard or a mobile banking app.

5. Notification System

- Implement SMS/Email alerts for low balance, loan approval, or unusual transactions.

6. **Analytics & AI Reports**

- Integrate with Power BI or AI models to predict loan defaults, customer trends, or branch performance.
- Generate customer behaviour insights to improve banking services.

7. **Enhanced Audit & Logging**

- Create a more detailed audit trail for every transaction, including who performed it and from where.
- Enable fraud detection rules for unusual withdrawal or transfer patterns.

8. Conclusion

The Banking System Database successfully fulfils the requirements of a core banking solution by providing a centralized, secure, and efficient way to manage customers, accounts, transactions, loans, branches, and employees.

Through the use of normalized relational tables, foreign key constraints, and stored procedures, the system ensures data integrity, consistency, and reliability. Operations such as creating customers, opening accounts, making deposits, withdrawals, and fund transfers are handled seamlessly while maintaining audit trails through transaction history.

Additional features like triggers for low balance alerts and withdrawal audits, along with reporting stored procedures and views, provide enhanced monitoring and analytics capabilities. This makes it easier for management to analyse branch performance, loan statuses, and top customers.

The database is modular and scalable, making it easy to extend with future enhancements such as role-based access control, credit/debit card modules, automated interest calculations, and integration with web or mobile banking applications.

In summary, this project lays a strong foundation for a complete Core Banking System (CBS) by addressing the key challenges of data management, security, and reporting in a banking environment, while remaining flexible enough for further upgrades.

GitHub Project link: https://github.com/Utkarshxtripathi/Banking_System