

PROJECT REPORT

ON

**Banking and Finance System Database**

*Submitted to*

**NMAM INSTITUTE OF TECHNOLOGY, NITTE**

(Off-Campus Centre, Nitte Deemed to be University, Nitte - 574 110, Karnataka, India)

In partial fulfilment of the requirements for the award of the

Degree of Bachelor of Technology

in

**INFORMATION SCIENCE AND ENGINEERING**

by

Aashna Mathias NNM23IS001

Akshatha Kini NNM23IS009

Ankitha NNM23IS015

Bhagya P Shetty NNM23IS032

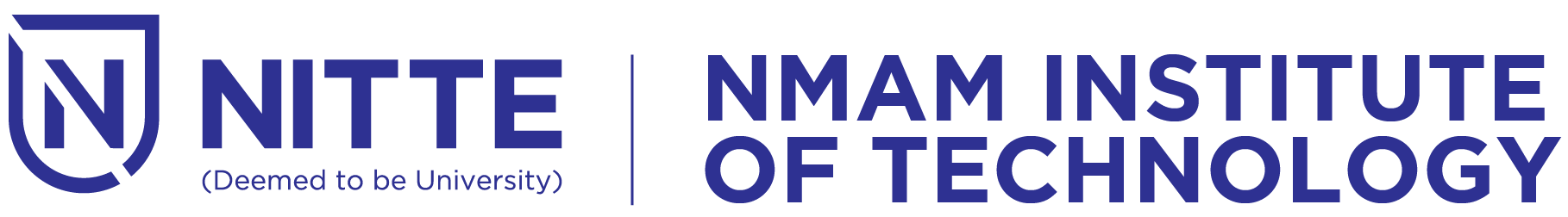
Under the guidance of

Dr.Santhosh S

Assistant Professor Gd.III

Department of Information Science and Technology,

NMAM Institute of Technology,Nitte,Karnataka,India



**Department of Information Science & Engineering**

### CERTIFICATE

This is to certify that Ms. Aashna Mathias (NNM23IS001), Ms. Akshatha Kini (NNM23IS009), Ms. Ankitha (NNM23IS015), Ms. Bhagya P Shetty (NNM23IS032) of II-year B.Tech., a bonafide students of NMAM Institute of Technology, Nitte, has carried out project on “**Banking and Finance System Database**” as part of the **Database Management System (CS2102-1)** course during 2024-25, fulfilling the partial requirements for the award of degree of Bachelor of Technology in Information Science and Engineering at NMAM Institute of Technology, Nitte.

…..………………………...........

Name and Signature of Course Instructor

**PROJECT EVALUATION MARKS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Evaluation Component** | Implementation  (8 Marks) | Presentation  (4 Marks) | Question and Answer  (4 Marks) | Organization of the report  (4 Marks) | Total  (20 Marks) |
| **Marks** |  |  |  |  |  |

..………………………...........

Signature of Course Instructor

PROJECT EVALUATION RUBRICS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation Component** | **Evaluation Levels** | | | |
| **Implementation** | **Excellent**  **(16-20)** | **Good**  **(10 – 15)** | **Satisfactory**  **(1-9)** | **Poor**  **0** |
| The project exceeds expectations with outstanding performance, creativity, and attention to detail. | The project meets expectations and demonstrates a solid understanding of the criteria. | The project partially meets expectations but requires improvement in certain areas. | The project does not meet expectations and requires significant improvement |
| **Presentation**  **(10 Marks)** | **Excellent**  **(9-10)** | **Good**  **(5 – 8)** | **Satisfactory**  **(1-4)** | **Poor**  **0** |
| Uses graphics that explain and reinforce text and presentation.  Refers to slides to make points, engaged with audience  Length and Pace of presentation is Appropriate and well-paced throughout | Uses graphics that explain text and presentation. Refers to slides to make points, eye contact majority of time  Length and Pace of presentation is Adequate and most of the seminar well-paced | Uses graphics that relate to text and presentation. Refers to slides to make points; occasional eye contact  Length and Pace of presentation is Short OR long. Rushed OR dragging in parts | Uses graphics that rarely support text and  Presentation. Reads most slides. No or just occasional eye  Very Short.  Length and Pace of presentation is Rushed OR dragging |
| **Question and Answer**  **(10 Marks)** | Demonstrated full knowledge.  Sufficient for understanding and exceptionally answered for all the questions | Demonstrated knowledge  Sufficient for understanding and answered for all the questions | Demonstrated some knowledge.  insufficient for understanding and answered for few questions | Does not have grasp of  Information and not answered for even one question |
| **Organization of the report**  **(10 Marks**) | Well documented content and followed all specified formats | Documented content and followed almost all specified formats | Fair document. Specified formats are not followed | Poor document content. Specified formats are not followed, submitted after deadline |

**ABSTRACT**

The Banking and Finance Database System developed in this project is designed to automate and simplify the core functionalities of a financial institution. It manages operations such as customer registration, account maintenance, loan processing, EMI (Equated Monthly Instalment) tracking, and transaction logging through a centralized platform that ensures efficient data handling, faster processing, and secure access to records.

The system utilizes a structured relational database model using SQL to store, manage, and retrieve data with precision. It incorporates data validation mechanisms and enforces constraints to maintain integrity while using stored procedures, triggers, and indexing to automate routine tasks and optimize performance.

The frontend, built with HTML, CSS, and JavaScript, offers a responsive user interface, while the backend, powered by Node.js, integrates seamlessly with the database to manage server-side operations. Together, they form a dynamic, real-time platform for efficient banking operations.

The system also emphasizes data security, user authentication, and error handling to ensure safe multi-user interactions. The use of ER and schema diagrams has supported clear data modelling, and the structured logic ensures scalability and maintainability.

This project demonstrates a comprehensive, real-world solution for managing banking and financial data with operational accuracy and reliability.

**ACKNOWLEDGEMENT**

It is with profound gratitude that we acknowledge the invaluable support and guidance received during the course of this project.

We are deeply indebted to the faculty and staff of the Department of Information Science and Engineering at NMAM Institute of Technology, Nitte, for providing us with the opportunity to undertake this project and for offering the resources and academic environment necessary for its successful completion.

We extend our sincere thanks to our project guide, whose expertise, consistent encouragement, and insightful feedback were instrumental throughout the development of this project. Their mentorship significantly contributed to the structure, technical depth, and overall quality of our work.

We also express our appreciation to all teaching and non-teaching staff for their indirect contributions and timely assistance. Our heartfelt thanks are due to our peers for their constant support and constructive suggestions.

Lastly, we are grateful to our families for their patience, moral support, and encouragement, which enabled us to stay motivated and focused throughout this endeavor.

**Table of Contents**

1. **Introduction**   
    1.1 Purpose  
    1.2 Scope  
    1.3 Overview
2. **Requirements Specification**  
    2.1 Hardware Requirements  
    2.2 Software Requirements
3. **System Design**  
    3.1 ER Diagram  
    3.2 Mapping from ER Diagram to Schema Diagram  
    3.3 Assumptions  
    3.4 Schema Diagram
4. **Implementation**  
    4.1 Pseudocode Used  
    4.2 Tables Used  
    4.3 Frontend Development  
    4.4 Backend Development  
    4.5 Database Design and SQL Implementation  
    4.6 Stored Procedures  
    4.7 Trigger Implementation  
    4.8 Integration Flow
5. **Results and Discussions**
6. **Conclusion and Future Work**
7. **References**

**List of Figures**

1. ER Diagram
2. Schema Diagram

**List of Tables**

1. BANK Table
2. BRANCH Table
3. CUSTOMER Table
4. ACCOUNT Table
5. ACCOUNT\_TRANSACTION Table
6. LOAN Table
7. EMI Table

**1.INTRODUCTION**

#### **1.1 Purpose**

The primary purpose of this project is to develop a robust and efficient Banking and Finance Database Management System that can support essential banking operations. These include customer management, account handling, loan tracking, transaction monitoring, and EMI scheduling. The system is intended to replace traditional, manual banking methods with an automated, secure, and scalable digital solution that ensures data consistency, accuracy, and integrity.

#### **1.2 Scope**

This system is designed for use in banking environments where managing a large volume of financial data is crucial. It provides tools for adding, retrieving, and updating customer and account details, processing financial transactions, managing loan data, and scheduling EMIs. The system also supports back-end automation using stored procedures and triggers, and ensures real-time interaction between the database and the user through a connected frontend.

#### **1.3 Overview**

The Banking and Finance System is built using a client-server architecture. The frontend is developed using HTML, CSS, and JavaScript to provide a user-friendly interface. Node.js is used to manage backend logic and serve as a bridge between the frontend and the SQL database, which stores all transactional and operational data. The design includes an ER diagram and schema diagram for efficient data modelling and normalization. The implementation also involves the use of pseudocode for key modules, along with validation through sample inputs and output screenshots.

**2.Requirements Specification**

**2.1 Hardware Requirements**  
To ensure smooth operation of the system, the following minimum hardware configuration is recommended:

* **Processor**: Intel Core i3 or higher (i5 or above preferred for performance)
* **RAM**: Minimum 4 GB (8 GB recommended)
* **Storage**: At least 100 GB HDD or 256 GB SSD
* **Display**: Standard monitor with a minimum resolution of 1366×768
* **Input Devices**: Keyboard and mouse
* **Network**: LAN or Wi-Fi capability for database and server connectivity

**2.2 Software Requirements**  
The project utilizes a combination of client-side, server-side, and database technologies. The required software stack includes:

* **Operating System**: Windows 10 (preferred)
* **Frontend Development**: HTML5, CSS3, JavaScript (for user interface and interactivity)
* **Backend Development**:
  + Node.js (server-side JavaScript environment)
  + Express.js (for route handling)
* **Database Management System**:
  + MySQL or Microsoft SQL Server
  + MySQL Workbench or SQL Server Management Studio (SSMS)
* **Text/Code Editor**: Visual Studio Code
* **Browser**: Chrome / Firefox for testing frontend interfaces
* **Additional Tools**:
  + Postman (for testing API endpoints)
  + XAMPP or Node runtime environment for server hosting.

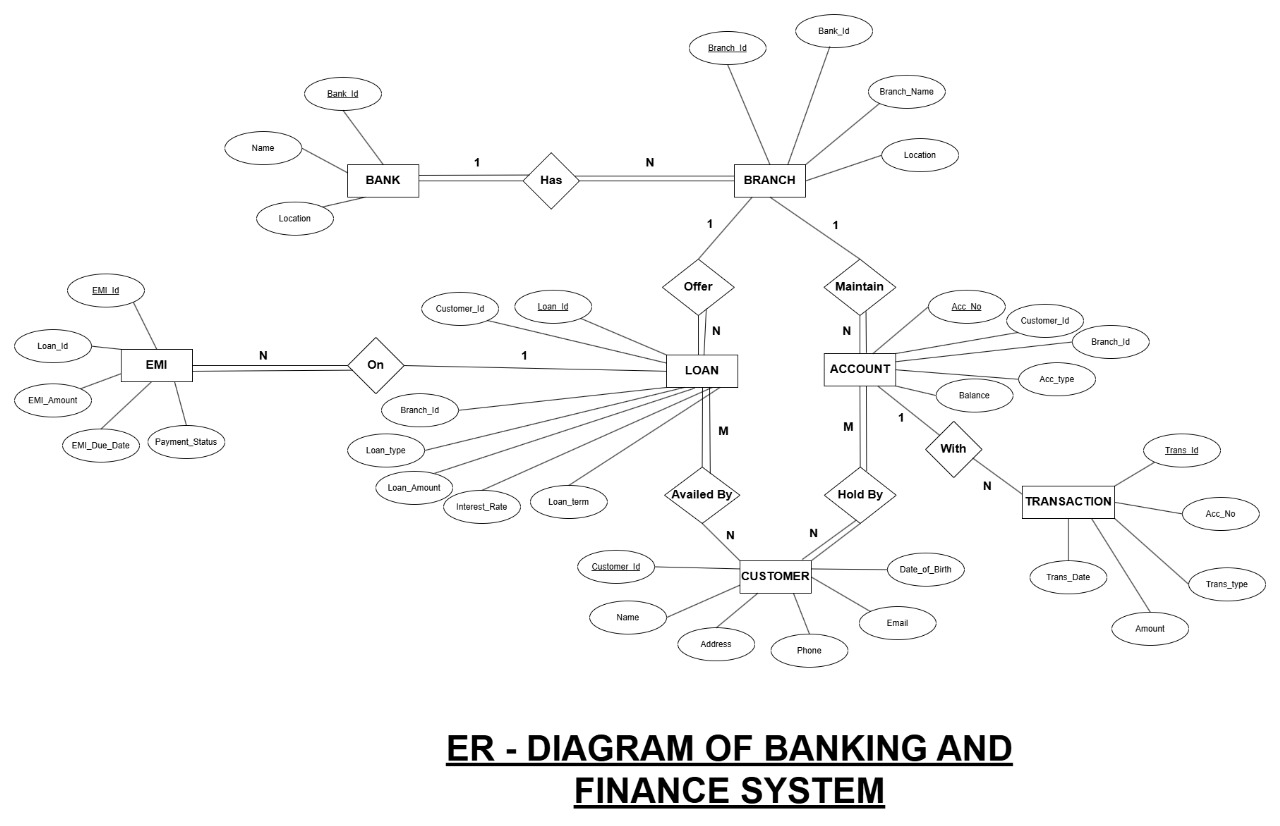
**3. System Design**

**3.1 ER Diagram**

The Entity Relationship (ER) diagram provides a high-level logical representation of the banking and finance system. It outlines the primary entities, their attributes, and the relationships between them. The ER model ensures clarity in data flow and lays the foundation for effective database design.

The major entities include:

* **BANK**: Contains Bank\_Id, Name, Location
* **BRANCH**: Linked to BANK, with Branch\_Id, Branch\_Name, Location
* **CUSTOMER**: Includes personal details such as Customer\_Id, Name, Address, Phone, Email, Date\_of\_Birth
* **ACCOUNT**: Holds information like Acc\_No, Acc\_Type, Balance, and links to both CUSTOMER and BRANCH
* **TRANSACTION**: Captures transaction records with Trans\_Id, Trans\_Date, Trans\_Type, Amount
* **LOAN**: Contains loan details including Loan\_Id, Loan\_Type, Loan\_Amount, Interest\_Rate, Loan\_Term
* **EMI**: Breaks down loans into manageable payments with EMI\_Id, EMI\_Amount, EMI\_Due\_Date, Payment\_Status.



*Figure 3.1:Entity Relationship Diagram*

## **3.2 Mapping from ER Diagram to Schema Diagram**

The ER diagram is translated into a relational schema to define the database structure using tables, primary keys, and foreign keys. The following mappings have been implemented based on the database schema used in the project:

### ****BANK****

CREATE TABLE BANK (

Bank\_Id INT PRIMARY KEY,

Name VARCHAR(100),

Location VARCHAR(100)

);

 Each bank has a unique Bank\_Id, along with its name and location.

* Bank\_Id serves as the primary key.

**BRANCH**

CREATE TABLE BRANCH (

Branch\_Id INT PRIMARY KEY,

Bank\_Id INT FOREIGN KEY REFERENCES BANK(Bank\_Id),

Branch\_Name VARCHAR(100),

Location VARCHAR(100)

);

* Each branch is linked to a bank through the foreign key Bank\_Id.
* Contains specific details like branch name and location.
* Branch\_Id is the primary key

#### **CUSTOMER**

CREATE TABLE CUSTOMER (

Customer\_Id INT PRIMARY KEY,

Name VARCHAR (100),

Address VARCHAR (255),

Phone VARCHAR (15),

Email VARCHAR (100),

Date\_of\_Birth DATE

);

* Holds customer personal and contact details.
* Customer\_Id acts as the primary key.

#### **ACCOUNT**

CREATE TABLE ACCOUNT (

Acc\_No INT PRIMARY KEY,

Customer\_Id INT FOREIGN KEY REFERENCES CUSTOMER(Customer\_Id),

Branch\_Id INT FOREIGN KEY REFERENCES BRANCH(Branch\_Id),

Acc\_type VARCHAR (50),

Balance DECIMAL (18, 2)

);

* Represents an account owned by a customer at a specific branch.
* Linked to both CUSTOMER and BRANCH tables via foreign keys.
* Acc\_No is the primary key.

#### **ACCOUNT\_TRANSACTION**

CREATE TABLE ACCOUNT\_TRANSACTION (

Trans\_Id INT PRIMARY KEY,

Acc\_No INT FOREIGN KEY REFERENCES ACCOUNT(Acc\_No),

Trans\_Date DATE,

Trans\_type VARCHAR (50),

Amount DECIMAL (18, 2)

);

* Records all transactions associated with an account.
* Includes details such as transaction type, amount, and date.
* Acc\_No serves as a foreign key to the ACCOUNT table.

#### **LOAN**

CREATE TABLE LOAN (

Loan\_Id INT PRIMARY KEY,

Customer\_Id INT FOREIGN KEY REFERENCES CUSTOMER(Customer\_Id),

Branch\_Id INT FOREIGN KEY REFERENCES BRANCH(Branch\_Id),

Loan\_type VARCHAR (50),

Loan\_Amount DECIMAL (18, 2),

Interest\_Rate FLOAT,

Loan\_term INT

);

* Stores information about loans issued to customers.
* Each loan is associated with a customer and a branch.
* Loan\_Id is the primary key.

#### **EMI**

#### CREATE TABLE EMI (

EMI\_Id INT PRIMARY KEY,

Loan\_Id INT FOREIGN KEY REFERENCES LOAN(Loan\_Id),

EMI\_Amount DECIMAL (18, 2),

EMI\_Due\_Date DATE,

Payment\_Status VARCHAR (50)

);

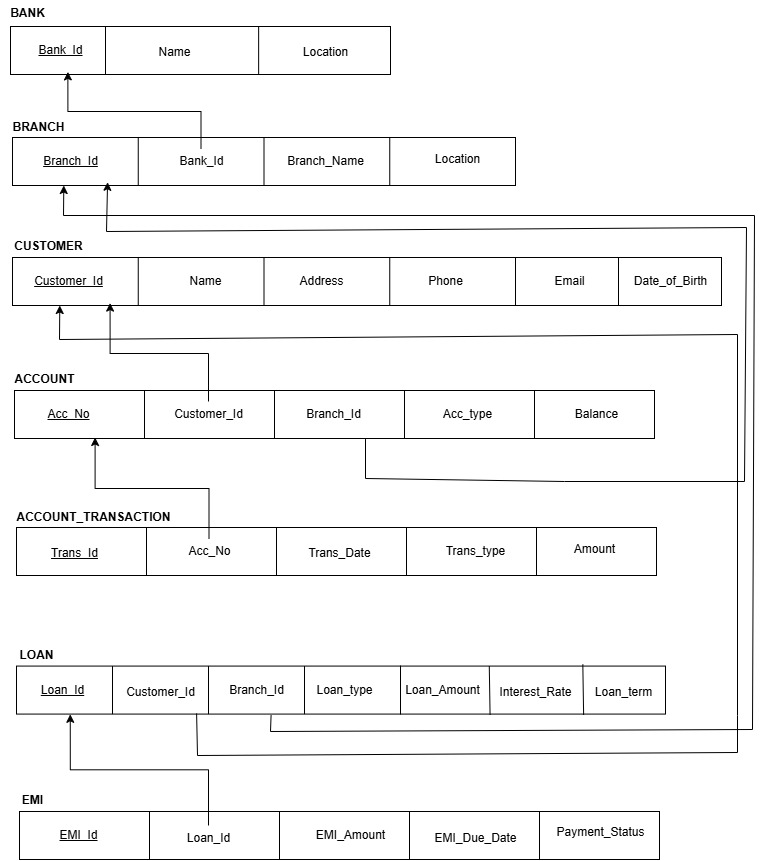
* Represents EMI payments for a loan.
* Linked to the LOAN table via foreign key Loan\_Id.
* EMI\_Id is the primary key.

**3.3 Assumptions**

To simplify the system and ensure consistency, the following assumptions are made:

1. Each **branch** belongs to only one **bank**.
2. A **customer** can hold multiple accounts and avail multiple loans.
3. Each **account** is held by one customer only (for simplicity, joint accounts are not considered).
4. **Transactions** are linked to accounts, and each transaction is associated with a single account.
5. **Loans** are branch-specific and customer-specific.
6. An **EMI** is associated with only one loan, and each loan can have multiple EMIs.
7. Proper validation is assumed at the UI level for data such as phone numbers, email formats, and account numbers.

**3.4 Schema Diagram**



*Figure 3.4: Schema Diagram*

**4. Implementation**

This section presents the end-to-end implementation of the Banking and Finance Database System, detailing the logic in pseudocode, the database schema, frontend-backend technologies, stored procedures, and integration workflows. The system is designed to automate and streamline core banking operations with accuracy and reliability.

* 1. **Pseudocode Used**

The logical flow for core operations is designed using structured pseudocode, guiding the backend functionality. Major modules include:

* **Customer Registration**

START

INPUT customer details

VALIDATE inputs (name, contact, ID)

IF valid THEN

INSERT data into CUSTOMER table

DISPLAY success message

ELSE

DISPLAY error message

ENDIF

END

* **Account Creation**

START

INPUT customer ID and account type

VALIDATE customer existence

IF exists THEN

GENERATE account number

INSERT data into ACCOUNT table

DISPLAY confirmation

ELSE

DISPLAY "Customer not found"

ENDIF

END

* **Transaction Processing**

START

INPUT account number, type (Deposit/Withdraw), and amount

VALIDATE account and balance (if withdraw)

IF valid THEN

INSERT transaction into ACCOUNT\_TRANSACTION

UPDATE account balance

DISPLAY updated balance

ELSE

DISPLAY error

ENDIF

END

* **Loan and EMI Entry**

START

INPUT loan details and customer ID

VALIDATE customer and data

IF valid THEN

INSERT into LOAN and EMI tables

DISPLAY loan ID and schedule

ELSE

DISPLAY error

ENDIF

END

**4.2 Tables Used**

The backend relies on a relational database model using SQL Server. The database is built on normalized tables, ensuring data consistency and efficient management of relationships. Key tables include:

* **BANK**: Stores overall bank details.
* **BRANCH**: Represents bank branches.
* **CUSTOMER**: Maintains customer records.
* **ACCOUNT**: Manages customer accounts.
* **ACCOUNT\_TRANSACTION**: Records all deposits and withdrawals.
* **LOAN**: Holds loan issuance information.
* **EMI**: Tracks EMI schedules and payments.

Each table is defined with appropriate data types, primary keys, and foreign key constraints. Data integrity is enforced using SQL constraints, normalization principles, and referential integrity.

### ****4.3 Frontend Development****

The frontend is designed using **HTML, CSS, and JavaScript**, providing a clean and intuitive interface for managing banking data. Each form corresponds to a database entity (Bank, Branch, Customer, Account, Loan, EMI, Transaction) and includes input validation. JavaScript handles dynamic behaviour and uses the fetch() API to send data to the backend as JSON via HTTP POST requests.

### ****4.4 Backend Development****

The backend is built using **Node.js** with the **Express.js** framework. RESTful API routes are defined (e.g., /api/bank, /api/customer, etc.) to handle incoming requests. Upon receiving form data, routes construct and execute corresponding SQL queries for data insertion or retrieval.

The server listens on **port 8081**, handles static file serving, and processes API requests. A connection pool in db.js manages communication with the SQL Server database, ensuring performance and stability.

### ****4.5 Database Design and SQL Implementation****

The system uses **Microsoft SQL Server** as the relational database engine. Tables are created using DDL commands, and foreign key constraints are used to define relationships. Data types, indexing, and table constraints are chosen to ensure efficiency and data consistency.

### ****4.6 Stored Procedures****

To simplify logic and encourage code reuse, multiple stored procedures are implemented:

* AddBank: Adds new bank entries.
* AddCustomer: Registers a new customer.
* OpenAccount: Opens accounts for customers.
* MakeTransaction: Records deposits or withdrawals.
* AddLoan: Issues new loans.
* AddEMI: Records EMI details.
* GetTransactionHistory, GetAccountBalance, GetEMIStatus: Fetch reports and account summaries.

These procedures reduce redundancy, encapsulate logic, and improve maintainability.

### ****4.7 Trigger Implementation****

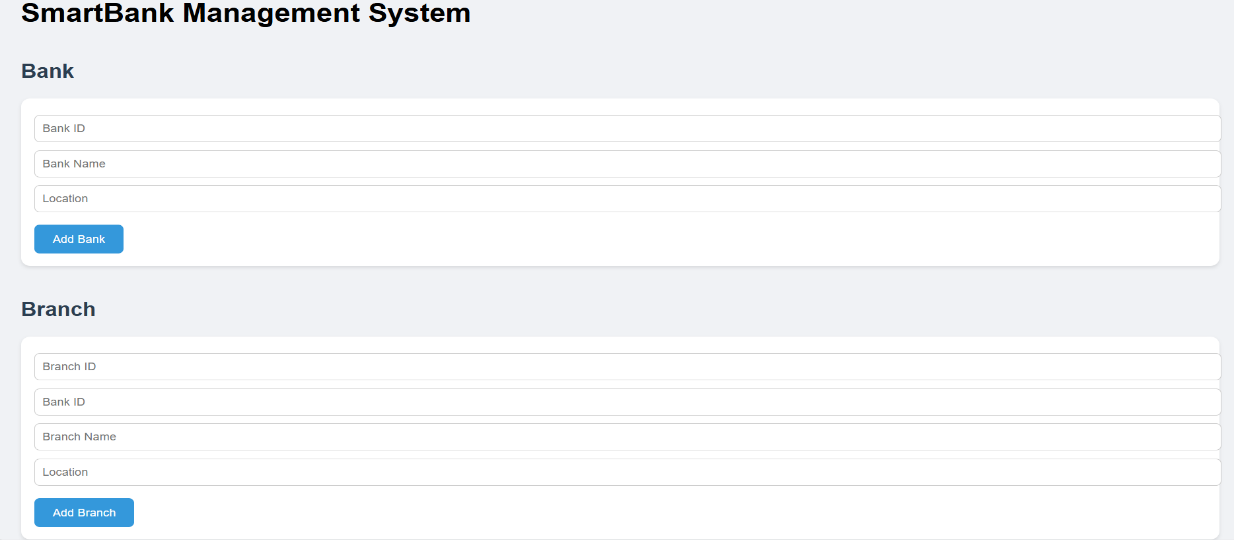
A key feature is the trigger **trg\_CheckWithdrawalLimit**, implemented on the ACCOUNT\_TRANSACTION table. This trigger:

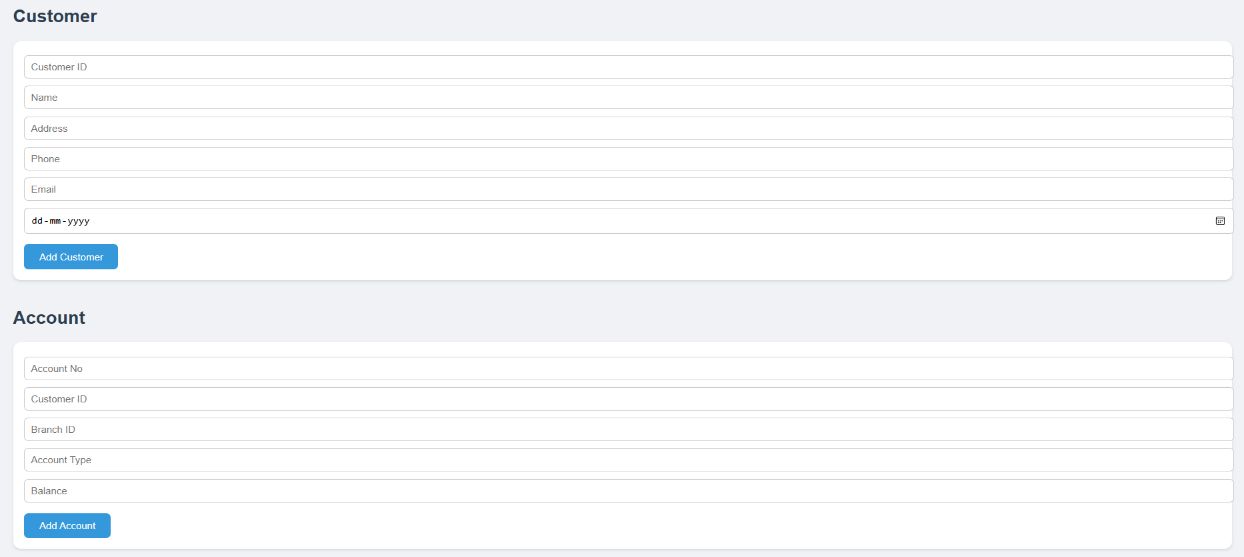
* Blocks withdrawal attempts that exceed the available balance.
* Automatically updates the account balance based on transaction type.
* Ensures transactional integrity before committing changes.

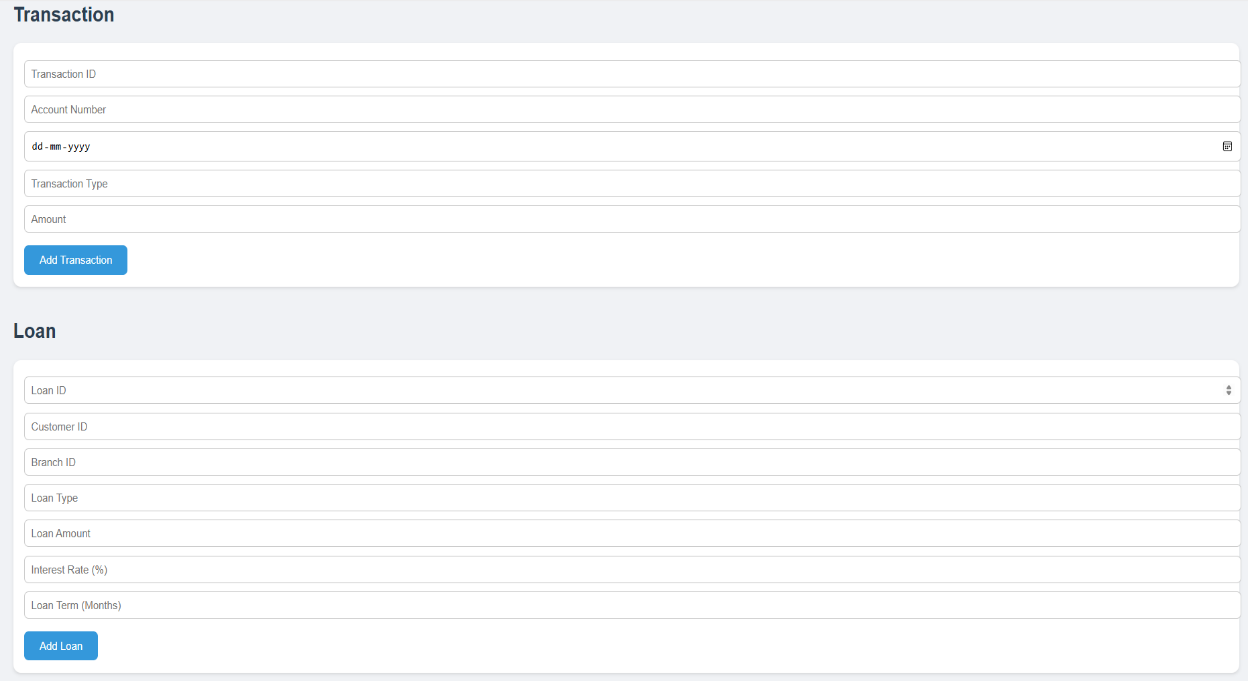
### ****4.8 Integration Flow****

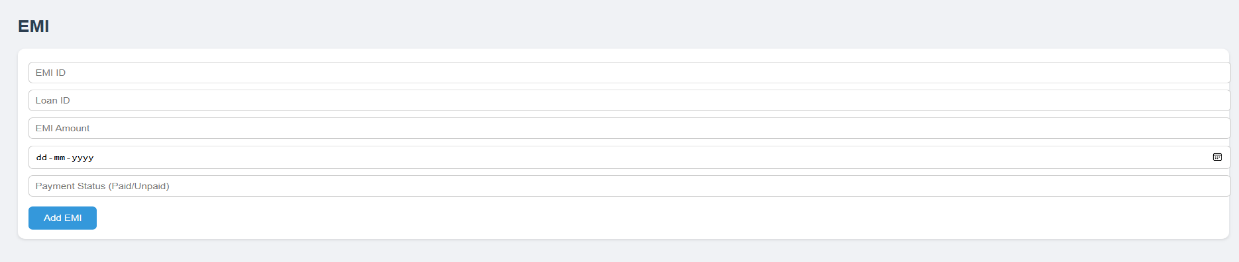
The system follows a clear flow for data processing and interaction:

1. **User Input**: Data is entered via the frontend interface.
2. **Data Submission**: JavaScript sends requests using fetch() to backend routes.
3. **Server Processing**: Node.js processes data and interacts with SQL Server.
4. **Validation & Execution**: SQL triggers and procedures handle logic and ensure correctness.
5. **Feedback**: Server responses are returned to the frontend, confirming actions.

**5. Results and Discussions**

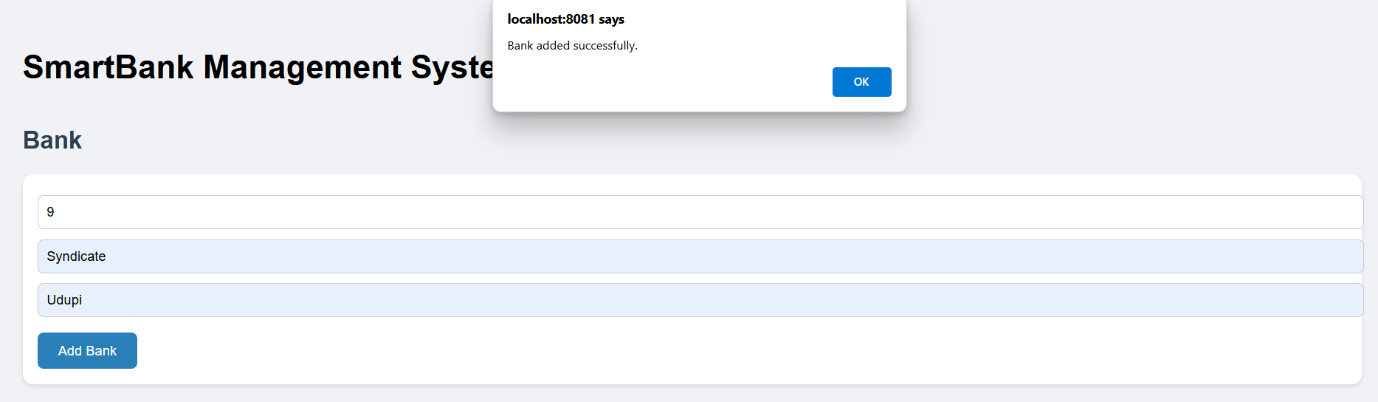


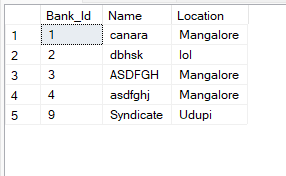




### Bank Module

**Frontend:**  
The user inputs Bank ID, Name, and Location in the Bank form.

**Backend (SQL):**  
The data appears correctly in the BANK table.

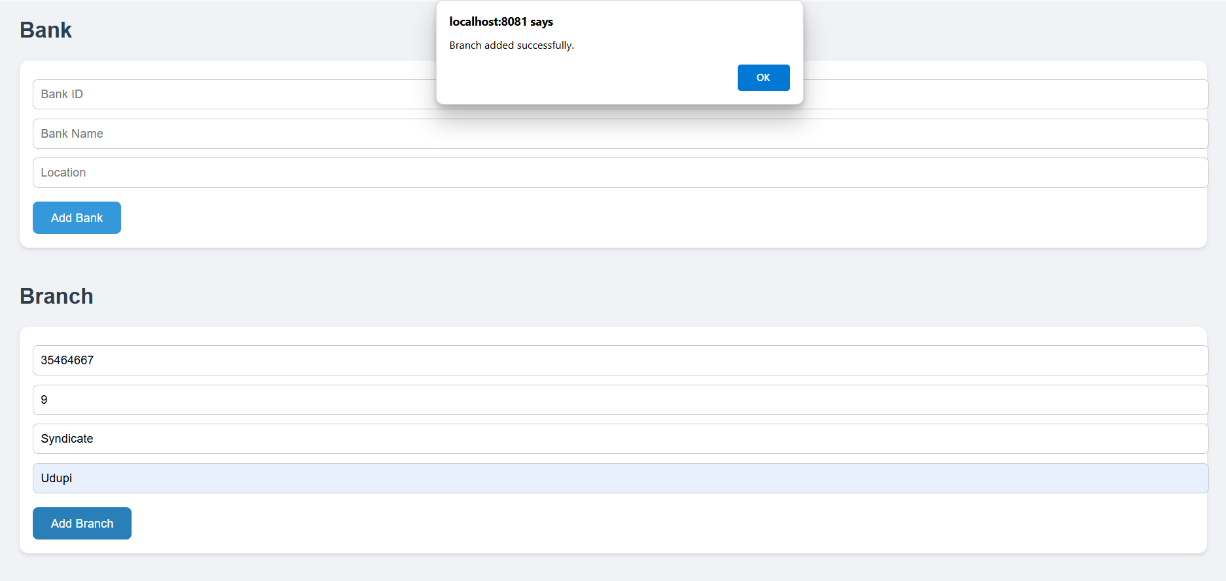


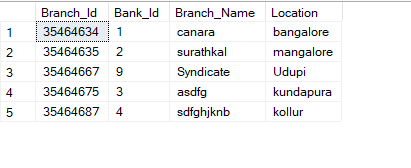
Result: Bank added successfully and reflected in the database.

### Branch Module

**Frontend:**  
The user fills Branch ID, Bank ID, Branch Name, and Location.

**Backend (SQL):**  
Entry is created in the BRANCH table.

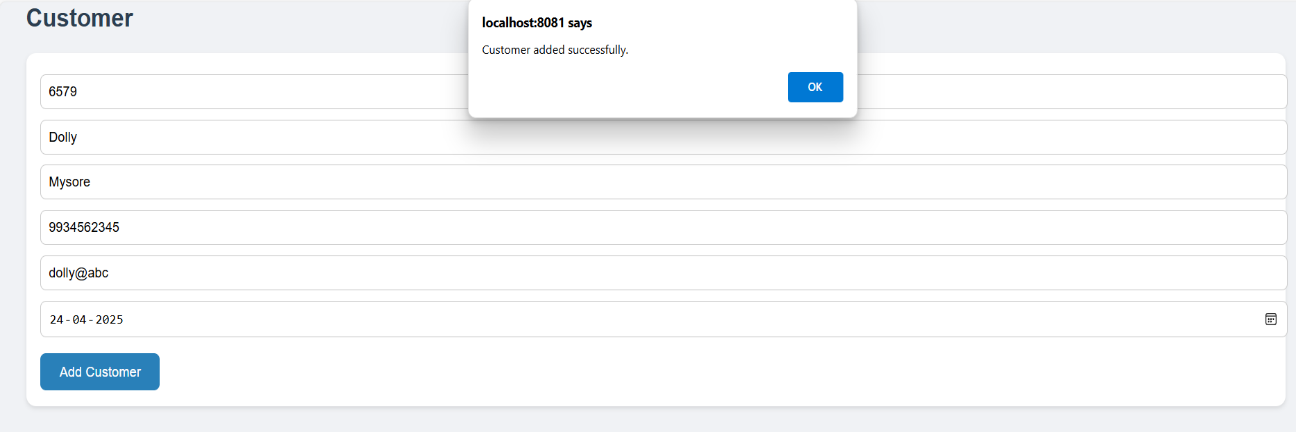


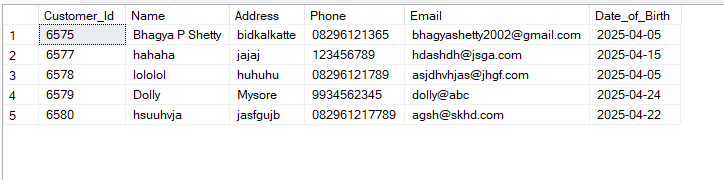


Result: Branch details added and confirmed in the backend.

### Customer Module

**Frontend:**  
Customer information such as ID, Name, Address, Phone, Email, and DOB is entered.

**Backend (SQL):**  
The data appears in the CUSTOMER table.

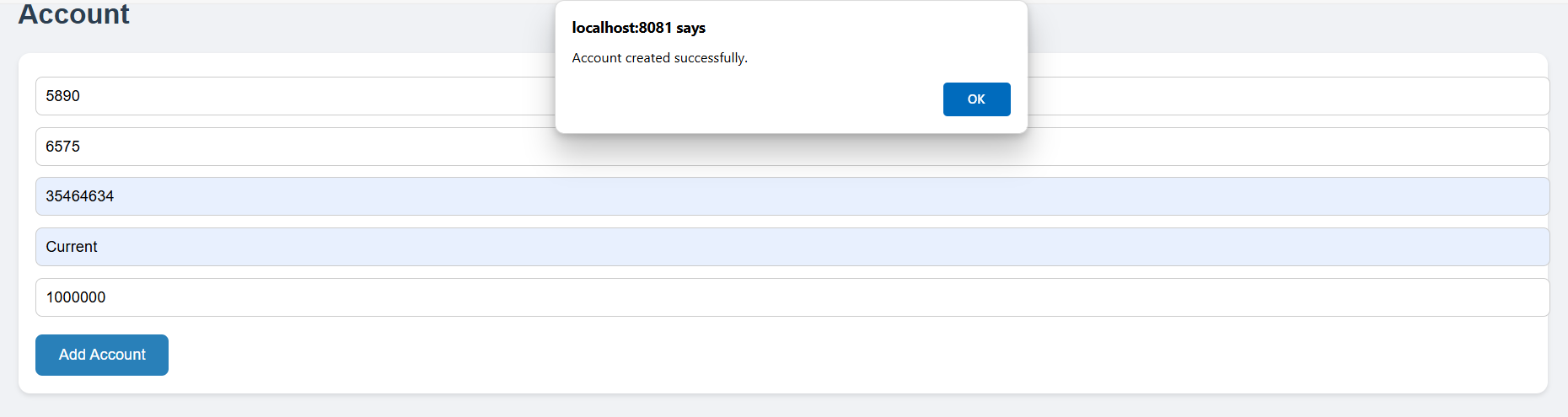


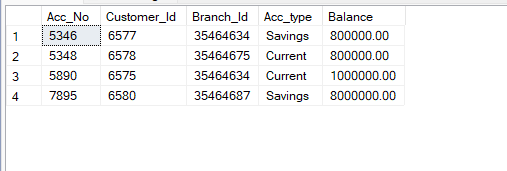
Result: Customer registered successfully in the system.

### Account Module

**Frontend:**  
Account Number, Customer ID, Branch ID, Type, and Balance are provided.

**Backend (SQL):**  
Entry created in the ACCOUNT table.

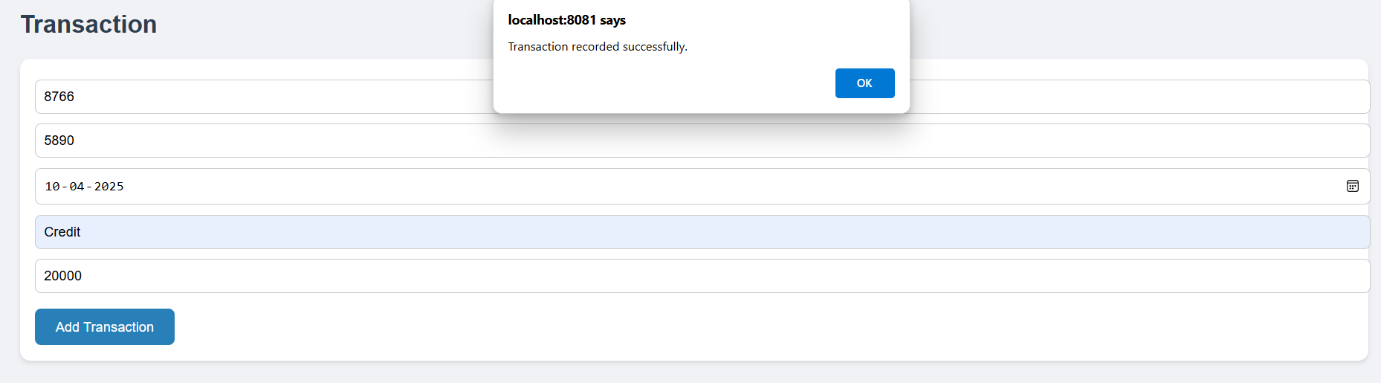
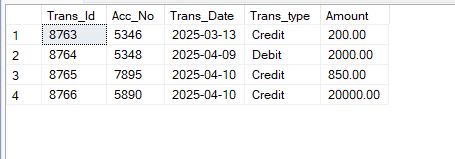




Result: Account data inserted accurately into the database.

### Transaction Module

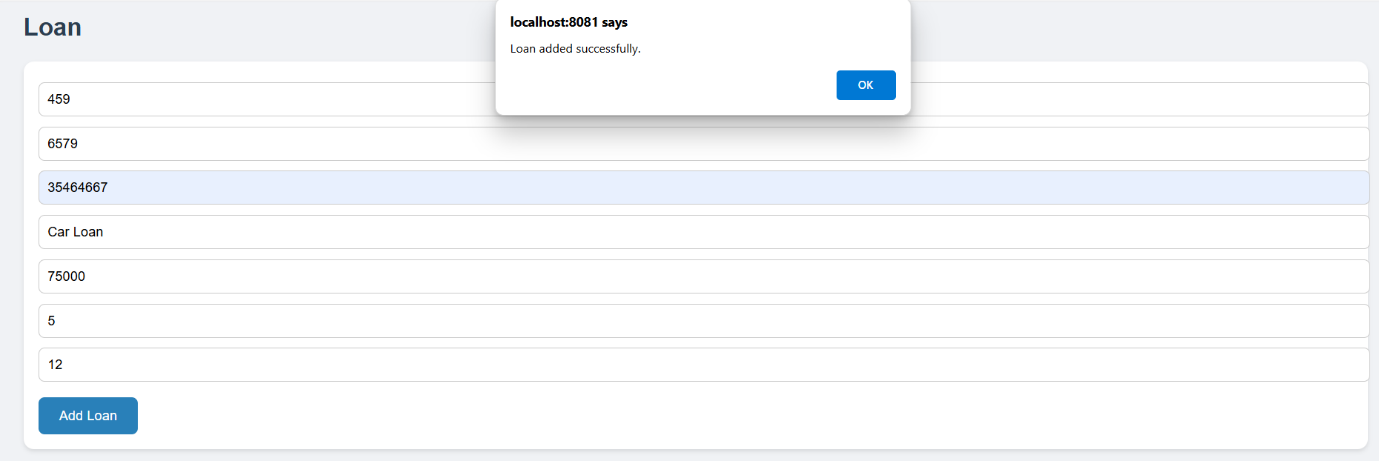
**Frontend:**  
User inputs Account Number, Transaction Date, Type, and Amount.

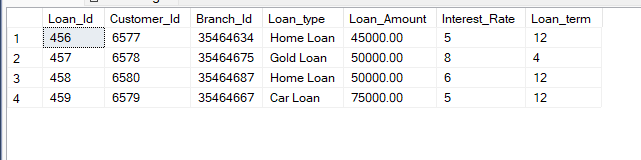
**Backend (SQL):**  
Entry is seen in ACCOUNT\_TRANSACTION table.

Result: Transaction recorded successfully in the backend.

### Loan Module

**Frontend:**  
Loan details including Loan ID, Customer ID, Branch ID, Type, Amount, Interest, and Term are entered.

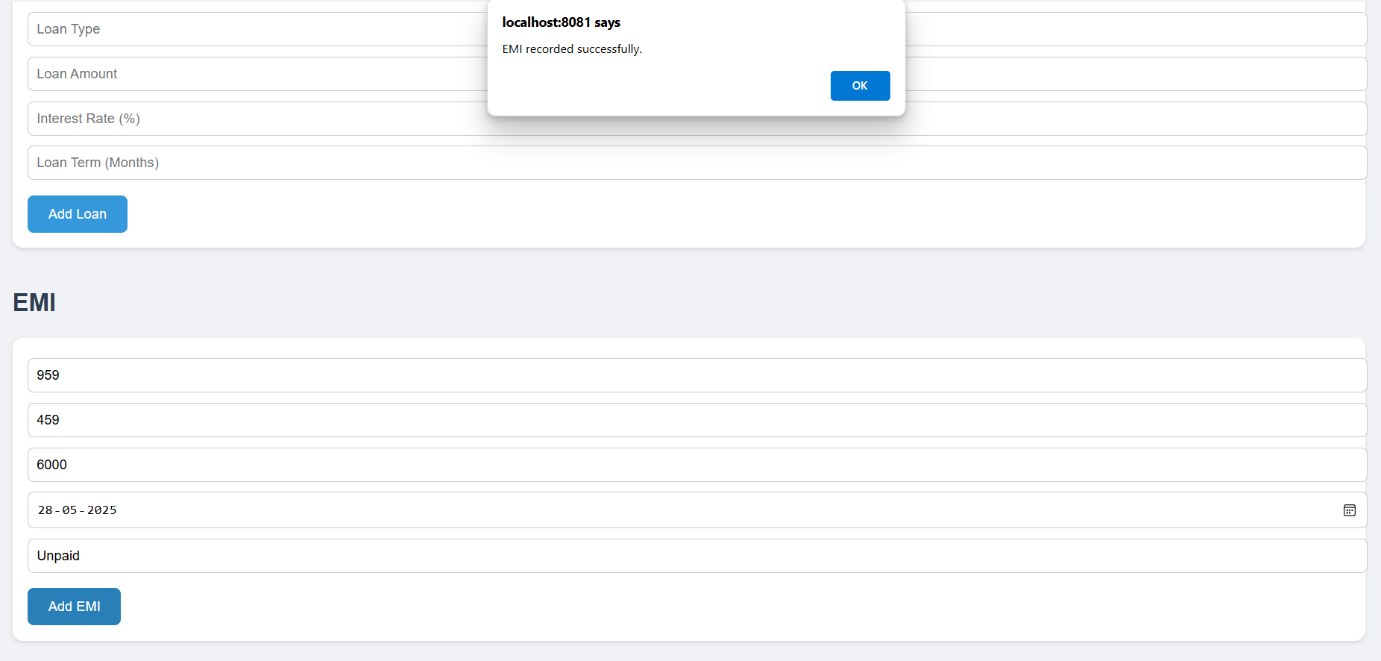
**Backend (SQL):**  
The LOAN table reflects the data.

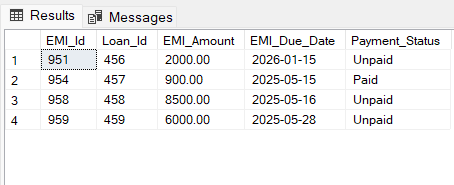


Result: Loan issued and logged in the database.

### EMI Module

**Frontend:**  
EMI details such as EMI ID, Loan ID, Amount, Due Date, and Payment Status are submitted.

**Backend (SQL):**  
Data is inserted into the EMI table.



Result: EMI record added and visible in backend.

1. **Conclusion and Future Work**

### Conclusion

The Banking and Finance Database Management System was successfully designed and implemented to manage key financial entities such as banks, branches, customers, accounts, loans, transactions, and EMIs. By integrating a user-friendly frontend with a robust SQL backend, the system ensures seamless data entry, real-time updates, and consistent data integrity. The project demonstrates secure and efficient handling of financial operations, fulfilling essential requirements such as customer account creation, transaction tracking, loan management, and EMI scheduling.

Each module was tested and verified by submitting form data through the UI, with successful storage and reflection in the corresponding SQL tables, thereby validating the complete end-to-end flow. Additionally, SQL triggers and stored procedures enhanced automation and error handling within the system.

### Future Work

* **Authentication & Role-based Access:** Implement login and permission-based modules to differentiate between admin, staff, and customers.
* **Dashboard and Reports:** Add visual dashboards with graphs and summaries for account balances, loan status, and transaction trends.
* **Notification System:** Integrate SMS or email alerts for due EMIs, low balances, and transaction confirmations.
* **Mobile Responsiveness:** Enhance frontend to support mobile devices for better accessibility.
* **Audit Trails:** Add logs for all actions performed by users to track system usage and improve security.

**References**

 **Microsoft SQL Server Documentation**  
<https://learn.microsoft.com/en-us/sql/sql-server>

 **Node.js Documentation**  
https://nodejs.org/en/docs/

 **Express.js Guide**  
<https://expressjs.com/>

 **W3Schools – HTML, CSS, JavaScript Tutorials**  
<https://www.w3schools.com/>

 **MDN Web Docs – JavaScript and Web Development**  
<https://developer.mozilla.org/>

 **GeeksforGeeks – DBMS and Web Technologies**  
<https://www.geeksforgeeks.org/>

 Course materials and lecture notes from NMAM Institute of Technology