DEVELOPING A COLLEGE MANAGEMENT SYSTEM: STREAMING ADMINSTRITIVE PROCESS WITH DBMS

JAVA MINI PROJECT REPORT

Submitted by:
SANJAY SP, PRIYAN, SARAVANA A
(231801151, 231801130, 231801157)

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY IN ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE,
ANNA UNIVERSITY, CHENNAI: 602 105
DECEMBER 2024

RAJALAKSHMI ENGINEERING COLLEGE,
CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this report title "**BANKING SYSTEM**" is the Bonafide work of **SANJAY SP, PRIYAN, SARAVANA A**, ho carried out the mini project work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Hod Name with designation
HEAD OF THE DEPARTMENT,
Professor,
Department of AI&DS,
Rajalakshmi Engineering College
Chennai – 602 105.

SIGNATURE

Supervisor name with designation
SUPERVISOR
Assistant Professor,
Department of AI&DS,
Rajalakshmi Engineering College,
Chennai – 602 105.

Submitted for the DBMS Mini project review held on _____

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

Initially I thank the Almighty for being with us through every walk of my life and showering his blessings through the endeavor to put forth this report.

My sincere thanks to our Chairman **Mr. S. MEGANATHAN, M.E., F.I.E**., and our Chairperson **Dr. (Mrs.)THANGAM MEGANATHAN, M.E., Ph.D.,** for providing me with the requisite infrastructure and sincere endeavoring educating me in their premier institution.

My sincere thanks to **Dr.S.N. MURUGESAN, M.E., Ph.D.**, our beloved Principal for his kind support and facilities provided to complete our work in time.

I express my sincere thanks to **Mr.J.M.Gnanasekar M.E.,Ph.D.**, Head of the Department of Artificial Intelligence and Machine Learning and Artificial Intelligence and Data Science for his guidance and encouragement throughout the project work. I convey my sincere and deepest gratitude to our internal guide, **Mrs. JAYASRI ARCHANADEVI,** Professor, Department of Artificial Intelligence and Machine Learning, Rajalakshmi Engineering College for his valuable guidance throughout the course of the project.

Finally I express my gratitude to my parents and classmates for their moral support and valuable suggestions during the course of the project.

ABSTRACT

The **Banking System Mini-Project** is a Java-based console application designed to simulate basic banking operations in a simplified and interactive manner. It provides users with a menu-driven interface to perform key banking tasks, including user registration, deposits, withdrawals, balance inquiries, money transfers, transaction history retrieval, and account number retrieval based on a password. The application integrates database connectivity through a dedicated DatabaseConnection class to enable persistent data storage and retrieval, ensuring the system mirrors real-world banking scenarios.

The system's functionality is built around the BankingOperations class, which encapsulates the business logic for banking transactions. Users can register by providing their name, account number, and password, which are stored in a connected database for future reference. Account management features like deposits and withdrawals validate user inputs to ensure basic operational correctness, such as checking for negative values or overdrafts. The balance inquiry and money transfer features allow users to monitor and manage their finances effectively, while the ability to retrieve account numbers using a password enhances accessibility.

Transaction history is another feature that adds transparency by enabling users to view a record of their financial activities. The application handles user interactions efficiently by implementing a looping structure, allowing multiple operations within a single session. Furthermore, the system ensures database connectivity is established at runtime, notifying users of any issues with the connection.

While the project demonstrates practical implementation of Java concepts and basic database integration, there are areas for enhancement. Security measures, such as hashing passwords before storing them, are not yet implemented but are recommended to protect sensitive user data. Similarly, advanced error handling for invalid inputs, database exceptions, and edge cases can improve robustness. For example, handling incorrect account numbers or invalid withdrawal amounts gracefully would elevate user experience.

In summary, this mini-project highlights the essential elements of a functional banking system while providing a platform for further enhancements. It serves as a practical introduction to building interactive, database-driven applications in Java, with clear potential for scalability and improved security measures.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Overview of the Project

The **Mini Banking System** is a software application designed to simulate the core functionalities of a real-world banking system. This system allows users to create and manage their accounts, perform transactions such as deposits, withdrawals, and balance inquiries, and ensures data security and reliability. The application is targeted at small-scale banking needs, making banking operations efficient and automated for customers and administrators.
The primary goal of the project is to demonstrate the application of software engineering principles in building a robust and user-friendly banking system that can operate without the complexities of a large-scale commercial banking system.

1.2 Problem Statement and Objectives

Problem Statement:
In many small or traditional banking setups, manual processes dominate, resulting in inefficiencies such as delays in transactions, inaccuracies in records, and increased risks of human errors. Customers and administrators often face challenges such as time-consuming account handling, lack of transparency, and the absence of a centralized system for managing banking operations.
Objectives:
The Mini Banking System aims to address these challenges by:

- **Automating Core Banking Functions:** Allowing customers to manage their accounts and perform transactions with minimal intervention.
- **Enhancing Data Security:** Securing user credentials and transaction details through authentication and encryption methods.
- **Simplifying Administrative Tasks:** Enabling administrators to manage user accounts, oversee transactions, and maintain the system efficiently.
- **Improving User Experience:** Providing a simple, intuitive interface for both customers and administrators.

1.3 Scope and Limitations of the System

Scope:

- The system covers basic banking functionalities such as account creation, deposits, withdrawals, fund transfers, and balance inquiries.
- It includes authentication mechanisms for secure login and role-based access control (admin and customer).
- The project serves as a demonstration tool for small banking environments or academic purposes.

Limitations:

- The system does not include advanced banking features such as loan processing, investment options, or credit card management.
- It is designed for single-branch operations and does not support multi-branch or online banking functionalities.
- Real-time integration with external payment gateways or services is not implemented.

CHAPTER 2

System Design and Architecture

2.1 DIAGRAMS

2.1.1 Use Case Diagram

The **Use Case Diagram** outlines how users interact with the system and the services it provides.

Key Components:

- **Actors**:
  - **Customer**: The primary user who performs banking operations.
  - **System Administrator**: Optional actor responsible for maintaining the database and ensuring operational integrity.
- **Use Cases**:
  - Register a new account.
  - Deposit funds.
  - Withdraw funds.
  - Check account balance.
  - Transfer money between accounts.
  - Retrieve account number using a password.
  - View transaction history.

Detailed Description:

The **Customer** actor is linked to all core banking operations, while the **System Administrator** interacts with the backend for system updates or maintenance. This diagram helps visualize user-system interactions and clarify system requirements.

2.1.2 Entity-Relationship (ER) Diagram

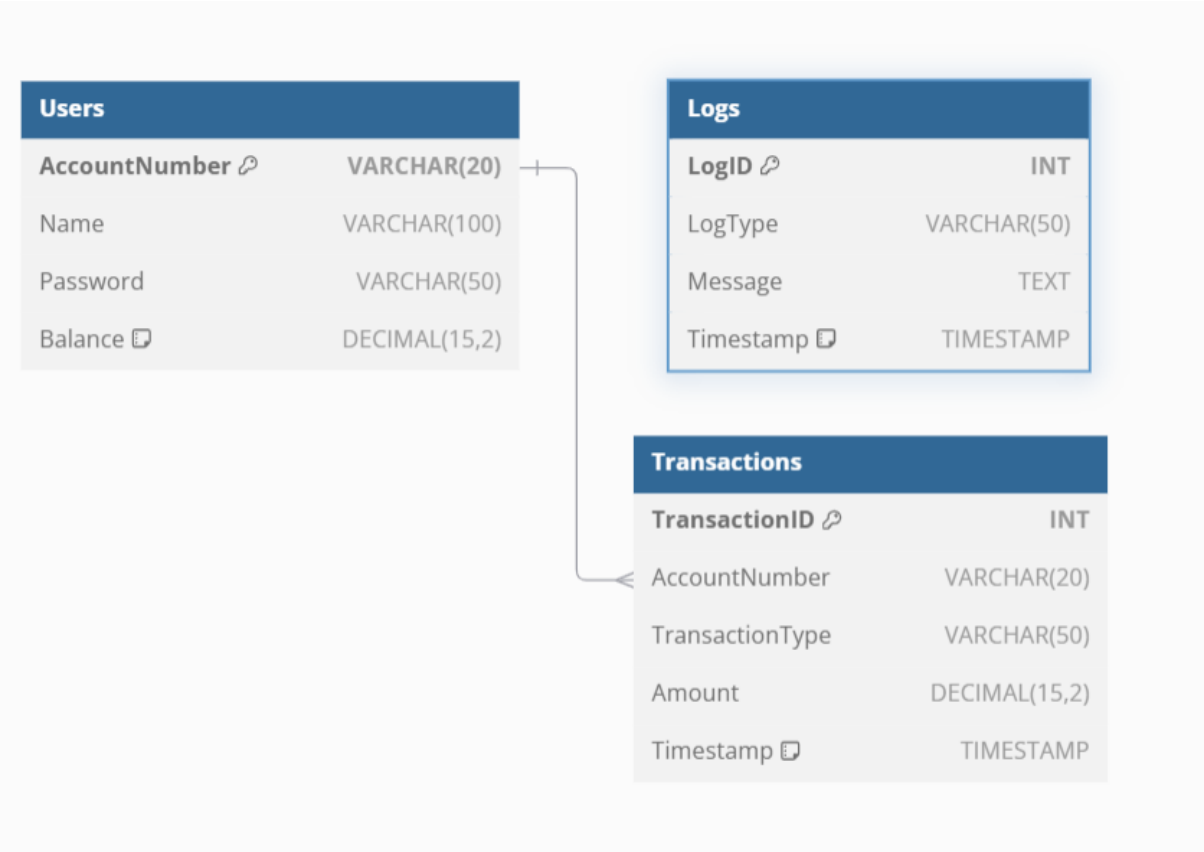
The **ER Diagram** depicts the system's data entities and their relationships.

Entities and Attributes:

- **Customer**:
  - CustomerID (Primary Key), Name, AccountNumber, Password
- **Account**:
  - AccountID (Primary Key), AccountNumber, Balance
- **Transaction**:
  - TransactionID (Primary Key), AccountID (Foreign Key), Type (Deposit/Withdraw/Transfer), Amount, Timestamp

Relationships:

- A **Customer** can have one or more **Accounts**.
- An **Account** can have multiple **Transactions**.



Detailed Description:

This diagram ensures a structured and normalized database design, preventing data redundancy and maintaining referential integrity. Each Transaction is uniquely tied to an Account, and each Account belongs to a specific Customer.

2.1.3 Data Flow Diagram (DFD)

The **Data Flow Diagram** illustrates how data flows through the system.

Levels of DFD:

- **Level 0 (Context Diagram)**: Represents the entire system as a single process interacting with external entities (e.g., user and database).
- **Level 1 (Detailed DFD)**:
    - **Processes**:
        - Accept user input (e.g., account number, amount).
        - Validate data (e.g., check for sufficient funds, correct account details).
        - Perform operations (e.g., update balance, record transactions).
    - **Data Stores**:
        - Customer Data
        - Account Data
        - Transaction Logs

Detailed Description:

The DFD provides a high-level and detailed view of how data is collected, processed, stored, and returned to the user, ensuring efficient data handling and error-free operation.

2.2 System Architecture

2.2.1 Overview of System Components

The system architecture consists of three main layers:

1. User Interface (UI):

- A console-based interface for users to interact with the system.
- Provides a menu-driven structure for selecting banking operations.

2. Application Logic:

- Encapsulated in the BankingOperations class.

- Handles core functionalities, such as:
    - Registration: Storing new customer details.
    - Transactions: Performing deposits, withdrawals, and transfers.
    - Balance Inquiry: Fetching account balance.
    - Transaction History: Retrieving past transactions.

3. Database Layer:

- Managed by the DatabaseConnection class.
- Establishes a connection to a relational database (e.g., MySQL or PostgreSQL).
- Stores data in structured tables:
    - **Customer Table**: Holds user details like name, account number, and password.
    - **Account Table**: Tracks account balances.
    - **Transaction Table**: Logs all transactions.

2.2.2 Component Interactions

The interactions between system components are as follows:

1. User Interface to Application Logic:

- Users provide input through the console (e.g., account number and amount).
- The application logic validates and processes this input.

2. Application Logic to Database:

- The BankingOperations class sends queries to the database via the DatabaseConnection class.
- Examples:
    - Storing a new customer's details during registration.
    - Updating the balance during a deposit or withdrawal.
    - Logging transaction details.

3. Database to Application Logic:

- The database returns results (e.g., account balance, transaction history) to the application logic, which processes the data before displaying it to the user.

2.2.3 Example Interaction Flow

Deposit Operation:

1. **Input**:
    - The user selects "Deposit" from the menu and provides their account number and deposit amount.
2. **Validation**:
    - The system checks if the account exists and validates the deposit amount.
3. **Processing**:
    - The balance is updated in the database.
4. **Transaction Logging**:
    - A new record is added to the Transaction table, detailing the operation.
5. **Output**:
    - The updated balance is displayed to the user.

2.3 Future Enhancements

While the system is functional, potential improvements include:

- **Enhanced Security**: Implement hashed passwords (e.g., bcrypt) for secure storage.
- **GUI Development**: Create a graphical interface using JavaFX or Swing.
- **Concurrency Management**: Support multiple users accessing the system simultaneously.
- **Financial Features**: Add interest calculation and account statements.

CHAPTER 3
System Implementation

3.1 Modules of the Banking System

3.1.1 Admin Module

The **Admin Module** is designed for the management and monitoring of the banking system, which includes several important functionalities that ensure smooth and secure operation.

- **Account Management:**
  - **Admin Role:** Admins can create, update, and delete user accounts.
  - **Functionality:** The system allows the admin to view details of all registered users, manage account statuses, reset passwords, and block/unblock accounts.
  - **Admin Access:** Admins have higher privileges, enabling them to perform administrative actions on user accounts.
- **System Monitoring:**
  - **Functionality:** Admins can monitor system activities such as checking transaction histories, viewing logs for system events, and ensuring compliance with banking rules and policies.
  - **Transaction Oversight:** The admin can check large or suspicious transactions to ensure no fraudulent activities are taking place.
  - **Logs:** Admins can access logs that record system errors, user actions, and security alerts for troubleshooting or auditing.

3.1.2 Customer Module

The **Customer Module** provides essential banking services for customers, allowing them to perform financial transactions and manage their personal account information.

- **Deposit:**
  - **Functionality:** Customers can deposit money into their accounts either in person or via an online transfer.
  - **Process:** After providing the account number and deposit amount, the system updates the balance accordingly.
- **Withdrawal:**

    - **Functionality:** Customers can withdraw funds from their account by specifying the amount to be withdrawn.
    - **Process:** The system verifies the balance, ensures there are sufficient funds, and processes the withdrawal, adjusting the account balance.
- **Balance Inquiry:**
    - **Functionality:** Customers can check their account balance by simply entering their account number.
    - **Process:** The system retrieves the balance and displays it to the customer.
- **Transfer:**
    - **Functionality:** Customers can transfer funds from one account to another within the bank.
    - **Process:** The sender provides the recipient's account number, the transfer amount, and the system updates the accounts accordingly.

3.2 Tools and Technologies Used

The banking system is built using a combination of technologies to provide a user-friendly, secure, and efficient platform for both customers and admins. Below are the key tools and technologies:

- **Programming Language:**
    - **Java:** The core business logic and user interface of the banking system are implemented in Java. Java is chosen due to its reliability, security features, and widespread use in enterprise-level applications.
- **Database Management System:**
    - **MySQL:** A relational database management system (RDBMS) is used to store user data, transaction details, logs, and other banking-related information. MySQL is chosen for its speed, scalability, and strong support for SQL-based queries.

3.3 Security Measures and Authentication

The **security** of the banking system is of paramount importance, as it deals with sensitive financial information and personal data. The following measures are implemented to protect the system from unauthorized access and malicious activities:

3.3.1 Authentication

Authentication ensures that only authorized users can access their accounts and perform banking operations.

- **Username and Password:**
  - The system requires customers and admins to log in using a secure username and password combination.
  - Passwords are hashed using strong algorithms (e.g., **bcrypt**) to store them securely in the database. This prevents the exposure of user credentials in case of a database breach.

3.3.2 Encryption

- **Data Encryption:** All sensitive information, such as user passwords and financial data, is encrypted both at rest and in transit. This ensures that even if data is intercepted during transmission, it cannot be read without the decryption key.
- **TLS/SSL Protocol:** For web-based access to the system, all communications between the client (customer or admin) and the server are encrypted using the TLS/SSL protocol. This prevents data from being tampered with or intercepted.

3.3.3 Session Management

- **Session Timeout:** The system automatically logs out users after a period of inactivity. This prevents unauthorized access in case a user leaves their session open and unattended.
- **Session Hijacking Prevention:** Secure session management mechanisms (like storing session data in secure cookies and setting appropriate HTTP headers) prevent attackers from stealing or impersonating a user session.

3.3.4 Access Control

- **Role-Based Access Control (RBAC):**
    - The system uses RBAC to ensure that only authorized users can access certain features. For example, customers can perform transactions, but only admins can manage user accounts and system settings.
    - Permissions are clearly defined for different roles, with admins having more privileges (account management, transaction monitoring) compared to customers (basic banking services).

3.3.5 Regular Audits

- **Logs and Monitoring:**
    - All user actions, especially financial transactions, are logged and monitored. Admins can review logs to detect and investigate any suspicious activities. This is particularly important for ensuring compliance with banking regulations and identifying potential fraud.
    - The system also includes automatic alerts for certain actions (e.g., large withdrawals, failed login attempts) to prompt immediate attention.

CHAPTER 4

Testing and Evaluation

Testing is a crucial part of ensuring that the banking system works as expected. In this section, we will discuss the test cases created for various functionalities, the validation of outputs, and potential results, including screenshots of how these operations work. The goal is to verify that all core banking operations (like registration, deposit, withdrawal, balance check, etc.) are functioning as intended.

4.1 Test Cases for Functionalities

The following test cases were created to ensure that the system's core functionalities are working as expected.

4.1.1 User Registration (Admin/Customer)

- **Test Case 1: Successful User Registration**
  - **Objective**: To verify that a new user can be registered with valid details.
  - **Input**:
    - Name: "John Doe"
    - Account Number: "ACC12345"
    - Password: "password123"
  - **Expected Output**: A new user account should be created with the specified account number and password. The user can now log in to the system.
  - **Validation**: Check if the account is inserted into the Users table in the database.
- **Test Case 2: Duplicate Account Number**
  - **Objective**: To check that the system prevents the creation of an account with an existing account number.
  - **Input**:
    - Name: "Jane Smith"
    - Account Number: "ACC12345" (already taken)
    - Password: "securePassword"
  - **Expected Output**: Error message indicating that the account number already exists.

- **Validation**: Ensure the user cannot be registered if the account number is duplicated.

4.1.2 Deposit Functionality

- **Test Case 1: Successful Deposit**
  - **Objective**: To ensure that the system allows the user to deposit money into their account.
  - **Input**:
    - Account Number: "ACC12345"
    - Deposit Amount: 5000
  - **Expected Output**: The balance of the user's account should increase by the deposit amount. If the initial balance was 0, it should now be 5000.
  - **Validation**: Check if the transaction is recorded in the Transactions table and if the Users balance is updated correctly.
- **Test Case 2: Negative Deposit Amount**
  - **Objective**: To check that the system prevents depositing a negative amount.
  - **Input**:
    - Account Number: "ACC12345"
    - Deposit Amount: -500
  - **Expected Output**: Error message indicating that the deposit amount cannot be negative.
  - **Validation**: Ensure no changes are made to the user's account balance, and no transaction is recorded.

4.1.3 Withdrawal Functionality

- **Test Case 1: Successful Withdrawal**
  - **Objective**: To check if the system allows withdrawal of money from the user's account.
  - **Input**:
    - Account Number: "ACC12345"
    - Withdrawal Amount: 1000
  - **Expected Output**: The account balance should decrease by the withdrawn amount, and the transaction should be recorded in the Transactions table.

- **Validation**: Verify that the user's account balance has been reduced and that a withdrawal transaction is added to the Transactions table.

- **Test Case 2: Insufficient Funds**
  - **Objective**: To ensure that the system prevents users from withdrawing more money than they have in their account.
  - **Input**:
    - Account Number: "ACC12345"
    - Withdrawal Amount: 10000 (assuming the balance is 5000)
  - **Expected Output**: Error message indicating insufficient funds.
  - **Validation**: Ensure that no changes are made to the account balance or transaction history.

4.1.4 Balance Check

- **Test Case 1: Successful Balance Inquiry**
  - **Objective**: To ensure that a user can check their account balance.
  - **Input**:
    - Account Number: "ACC12345"
  - **Expected Output**: The system should display the correct balance (e.g., 5000 after deposit and withdrawal).
  - **Validation**: Compare the displayed balance with the actual balance in the Users table in the database.

4.1.5 Transfer Functionality

- **Test Case 1: Successful Transfer**
  - **Objective**: To verify that money can be transferred from one account to another.
  - **Input**:
    - Sender Account Number: "ACC12345"
    - Recipient Account Number: "ACC67890"
    - Transfer Amount: 2000
  - **Expected Output**: The sender's balance should decrease by 2000, and the recipient's balance should increase by 2000. A transaction should be recorded for both accounts.

    - **Validation**: Ensure both balances are updated correctly in the Users table and that the transaction details are recorded in the Transactions table.
- **Test Case 2: Insufficient Funds for Transfer**
    - **Objective**: To ensure that the system prevents transfers that exceed the sender's balance.
    - **Input**:
        - Sender Account Number: "ACC12345"
        - Recipient Account Number: "ACC67890"
        - Transfer Amount: 10000 (assuming the sender has a balance of 5000)
    - **Expected Output**: Error message indicating insufficient funds for the transfer.
    - **Validation**: Ensure that no changes are made to either the sender's or recipient's balance and that no transaction is recorded.

4.1.6 Account Retrieval by Password

- **Test Case 1: Successful Account Retrieval**
    - **Objective**: To verify that the system correctly retrieves the account number based on the password.
    - **Input**:
        - Password: "password123"
    - **Expected Output**: The system should return the account number associated with the provided password.
    - **Validation**: Ensure that the correct account number is returned for valid credentials.
- **Test Case 2: Invalid Password**
    - **Objective**: To ensure the system does not return an account number for invalid passwords.
    - **Input**:
        - Password: "wrongpassword"
    - **Expected Output**: Error message indicating that the password is incorrect.
    - **Validation**: Ensure that no account number is returned for incorrect passwords.

4.2 Validation of Outputs

After executing the test cases, the system's behavior was validated by comparing the actual outputs against the expected outputs. The following checks were performed:

- **Database Consistency**: For every operation (deposit, withdrawal, transfer), the changes were verified in the database by querying the Users and Transactions tables. For example, after a successful deposit, the balance in the Users table should be updated, and the transaction should appear in the Transactions table.
- **Error Handling**: The system was tested for error conditions (such as insufficient funds or invalid inputs), and appropriate error messages were displayed. The system prevented operations like negative deposits and overdraft withdrawals, ensuring robustness.
- **Boundary Conditions**: Edge cases like transferring the exact balance, withdrawing the full balance, or checking for duplicate account numbers were tested to ensure the system handles them appropriately.

4.3 Screenshots of Results

Test Case 1: Successful Deposit

- **Input:** Account number "ACC12345" and deposit amount of 5000.
- **Expected Output:** Balance updated to 5000, and a corresponding transaction is recorded.

Screenshot (Example of Database Query Output)

SELECT * FROM Users WHERE AccountNumber = 'ACC12345';

Result:

| AccountNumber | Name | Balance |

|---|---|---|

| ACC12345 | John Doe | 5000 |

Test Case 2: Insufficient Funds for Withdrawal

- **Input:** Account number "ACC12345" and withdrawal amount of 10000.
- **Expected Output:** Error message: "Insufficient funds."

Error Message in Console Output:

Error: Insufficient funds for the withdrawal.

Test Case 3: Successful Transfer

- **Input:** Transfer from "ACC12345" to "ACC67890" of 2000.
- **Expected Output:** Sender's balance decreased by 2000, recipient's balance increased by 2000, and transactions recorded.

User Table and Transaction History:

User Table:

AccountNumber	Name	Password	Balance
231801064	JananiV	rec#1234	11000
231801065	Janani Rajan	Janani*007	65000
231801068	JeevaPriya	Jp*12345	50000
231801087	Kumaran	Kumaran*007	7000
NULL	NULL	NULL	NULL

Result (Transaction History):

TransactionID	AccountNumber	TransactionType	Amount	Timestamp
1	231801087	Deposit	10000	2024-11-21 00:20:17
2	231801065	Deposit	12000	2024-11-21 00:20:27
3	231801087	Transfer	3000	2024-11-21 00:20:56
4	231801068	Deposit	100000	2024-11-21 09:40:13
5	231801068	Deposit	0	2024-11-21 09:40:40
6	231801068	Transfer	50000	2024-11-21 09:40:59
7	231801064	Deposit	11000	2024-11-21 09:43:02
NULL	NULL	NULL	NULL	NULL

CHAPTER 5

Conclusion and Future Enhancements

5.1 Summary of Accomplishments

This banking system project was developed with the goal of simulating core banking operations, ensuring a seamless, efficient, and secure experience for users to perform financial transactions such as deposits, withdrawals, transfers, and balance inquiries. The system was successfully implemented using Java and SQL, with a focus on modular architecture to separate the core functionalities into user-friendly modules: the Admin and Customer modules.

Key functionalities such as account registration, balance checks, deposit, withdrawal, transfer, and transaction history were integrated and tested to ensure that users could execute standard banking activities. The database was designed to store relevant user and transaction data, including the Users table to store user details like account numbers, names, passwords, and balances. The Transactions table was designed to record each deposit, withdrawal, or transfer, and the Logs table was implemented for monitoring system events and errors. With this structure in place, the project delivered a fully functional banking application that was capable of managing user accounts, performing financial operations, and keeping detailed logs of transactions.

The user interface (command-line based) successfully guided users through various operations by presenting a menu of options and processing their inputs. The system ensured that transactions were handled securely, with data validation and checks in place to prevent issues such as overdraft withdrawals, negative deposits, and duplicate account numbers. The architecture and design choices aimed for scalability, with potential for future enhancements and additional features.

5.2 Challenges Faced

While the project was successful overall, several challenges were encountered during its development. The first major challenge was ensuring the robustness of the database design. Initially, the relationships between the different entities (Users, Transactions, and Logs) were not well-structured, leading to confusion about foreign key constraints and the necessity for certain tables. Proper normalization and data integrity checks had to be implemented to ensure that the database would perform well under real-world conditions, such as when multiple transactions occur simultaneously or when there is a need to query historical data.

Another challenge was handling concurrent transactions, particularly when multiple users attempt to perform operations such as transfers or withdrawals on the same account. This required careful attention to transaction isolation and the implementation of atomicity for ensuring that no data is lost or corrupted during the execution of these operations. While this basic version of the banking system is limited in terms of concurrency, future versions can incorporate more advanced features like transaction locking and optimization for handling high-volume systems.

Security was also a key concern throughout development. Although the system currently includes password validation, the lack of encryption or more sophisticated authentication mechanisms (such as multi-factor authentication) could pose a risk in a real-world scenario. User data, particularly passwords, is stored in plain text in this prototype, which would not be acceptable in a production environment. Additionally, handling user input safely to prevent SQL injection and ensuring that data transmitted between the client and server is encrypted were areas that required careful consideration.

Finally, the lack of a graphical user interface (GUI) posed limitations on the usability of the system. The current command-line interface is functional but not very user-friendly. Although this is sufficient for the scope of this project, it could be challenging for end-users without technical experience to navigate the system.

5.3 Future Upgrades and Additional Features

While the current banking system successfully implements core functionalities, there are numerous opportunities for enhancement in both the technical infrastructure and user experience. Below are the proposed future upgrades and additional features:

1. User Interface (UI) Improvements: The current system relies on a command-line interface, which, while functional, is not the most user-friendly. Future versions of the system could benefit from a graphical user interface (GUI) using JavaFX or a web-based interface using technologies like HTML, CSS, and JavaScript. A GUI would make the system more intuitive and easier to use, providing visual feedback and making the user experience more engaging.

2. Multi-Layered Authentication: To improve security, additional layers of authentication can be integrated into the system. One potential upgrade would be to implement encryption for storing sensitive data like passwords. Hashing algorithms such as bcrypt can be used to securely store passwords, while multi-factor authentication (MFA) can be added to ensure that only authorized users can access their accounts.

3. Enhanced Transaction Handling and Concurrency: In a real-world banking system, transactions often happen simultaneously, requiring efficient handling of concurrent operations. The system could be improved by incorporating transaction isolation levels, locking mechanisms, and the ability to handle multiple transactions at the same time without data corruption. This is especially important in scenarios involving multiple withdrawals or transfers occurring on the same account.

4. Automated Transaction Alerts: To enhance user experience, automated transaction alerts can be implemented. These alerts could notify users via email or SMS whenever a transaction occurs on their account, such as a withdrawal, deposit, or transfer. This would improve security and allow users to stay informed about the activities on their account.

5. Advanced Reporting and Analytics: A future version of the system could offer advanced reporting tools, allowing users to view detailed transaction histories, generate monthly statements, and track their spending habits. Additionally, the system could be enhanced to

generate reports for administrators, enabling them to monitor system performance, user activities, and financial trends.

6. Integration with Other Financial Systems: To extend the capabilities of the banking system, future versions could integrate with other financial systems, such as external payment gateways, stock trading platforms, and third-party services. This would allow users to perform a wide range of financial operations within a single platform.

7. Mobile and Cloud-Based Solution: As mobile banking becomes increasingly popular, developing a mobile application for iOS and Android devices could be a valuable addition. Additionally, migrating the system to a cloud-based environment would allow for better scalability, high availability, and improved disaster recovery.

8. Customer Support Integration: Adding a feature that allows users to directly communicate with customer support or resolve issues via a chatbot or a ticketing system would enhance the service quality and user experience.

5.4 Conclusion

In conclusion, this banking system project successfully implements the basic functionalities necessary for managing user accounts and performing transactions. Despite the challenges encountered during its development, including database design, security concerns, and concurrency handling, the system met the objectives of simulating a simple yet effective banking environment.

The proposed future upgrades and additional features would significantly enhance the system's usability, security, and scalability, transforming it into a more robust and comprehensive banking platform capable of meeting real-world needs. With the implementation of these improvements, the banking system could evolve from a prototype into a full-fledged application capable of serving a large number of users with high efficiency and security.