Established – 1961 Subject:_____

SEVA SADAN'S

R. K. TALREJA COLLEGE

OF

ARTS, SCIENCE & COMMERCE ULHASNAGAR – 421003

CERTIFICATE

This is to certify that Mr./Ms. _____ of S.Y. Information Technology (SYIT) Roll No. _____ has satisfactorily completed the Open Source DataBase Management System Mini Project entitled

_____

during the academic year 2025 – 2026, as a part of the practical requirement. The project work is found to be satisfactory and is approved for submission.

PROF. INCHARGE **HEAD OF DEPT**

_____ _____

PROJECT DOCUMENTATION STRUCTURE

Open Source Database: MySQL

(Structure starts AFTER cover page, certificate & acknowledgement)

TABLE OF CONTENTS

1.INTRODUCTION

A database system is an organized collection of data stored electronically. It allows efficient storage, retrieval, updating, and management of data

It reduces data redundancy and maintains data consistency. It ensures data accuracy, security, and integrity.

It supports multiple users accessing data at the same time.

Banks handle a large number of daily transactions such as deposits, withdrawals, and fund transfers.

Manual or file-based systems can lead to errors, data loss, and inconsistencies.

Managing customer records and transaction history becomes difficult without a proper system.

There is a high risk of partial updates during system failures.

A Transaction Banking Management System ensures accurate, secure, and reliable transaction processing.

It maintains consistency even when multiple users perform transactions simultaneously. It follows ACID properties to ensure safe and complete transactions.

2.PROBLEM DEFINITION

In traditional banking environments, many operations were earlier handled using manual methods or unstructured systems such as paper records or basic file-based storage. These systems are inefficient for handling a large volume of daily banking transactions. As the number of customers and transactions increases, managing data manually becomes complex, time-consuming, and error-prone. This creates serious challenges in maintaining accuracy, security, and consistency of banking data.

Issues in Existing Manual / Unstructured System

High data redundancy due to repeated entry of customer and transaction details.

Data inconsistency caused by multiple records holding different values for the same data.

Difficulty in tracking transaction history accurately.

High chances of human errors during data entry and calculations.

Lack of proper security mechanisms to protect sensitive customer information.

No support for concurrent access by multiple users.

Risk of data loss due to system failure or improper record handling.

Slow data retrieval and report generation

3.OBJECTIVES OF THE PROJECT

To design and develop a database-driven Transaction Banking Management System. To store and manage customer and transaction data in a structured manner.

To ensure accurate and secure processing of banking transactions.

To implement transaction control mechanisms such as commit and rollback. To maintain data consistency and integrity using ACID properties.

To reduce data redundancy and eliminate inconsistencies.

To provide secure access to banking data through proper authorization. To support multi-user access without data conflicts.

To maintain a complete and reliable transaction history.

To generate reports for account details and transaction records.

To use MySQL as an open-source database solution for efficient data management.

To understand real-time banking transaction processing using database concepts.

4.SCOPE OF THE PROJECT

The scope of this project is limited to designing and implementing a banking transaction management system using MySQL. The system demonstrates how transactions are processed securely using database concepts such as ACID properties, COMMIT, and ROLLBACK.

Where the System Can Be

Used Small-scale banking

institutions Financial training

centers

Educational institutions for database learning

Business environments requiring secure transaction

recording Academic project demonstrations

Users of the System

Administrator (Admin)

Manages database access and user privileges

Performs backup and recovery

Maintains system

security Staff / Bank

Employee Handles

account creation

Performs deposits, withdrawals, and fund

transfer Updates customer information

Student (Academic User)

Uses the system for learning transaction

management Understands practical implementation

of SQL transactions Studies multi-user database

handling

Business User

Maintains financial transaction records

Tracks account balances and transaction

history Ensures data consistency and reliability

Subdomains of Application

1. Educational Use

Used in colleges and universities for teaching database management

concepts. Helps students understand real-world transaction processing.

2. Business Use

Can be adapted for small businesses to manage financial

transactions. Ensures secure and consistent record keeping.

3. Administrative Use

Useful for managing institutional funds and internal financial

operations. Provides controlled access through user privileges.

Academic Limitations

Designed for academic and learning purposes only.

Does not include advanced banking features such as online payment gateways or real- time ATM integration.

Secrity implementation is basic and not suitable for large-scale commerc

5.REQUIREMENT SPECIFICATION

1.1 Hardware Requirements

To successfully develop and run a Banking Transaction Management System, proper hardware is required to ensure smooth performance, fast processing, and secure data handling.

1. Computer / Laptop

A standard desktop computer or laptop is required to install the database software and run the banking application.

Explanation:

The computer acts as the main system where:

Database software (like MySQL) is installed

Banking application is executed

Transaction processing takes place

Reports and backups are generated

For development and testing purposes, any modern system with basic specifications is sufficient. However, in real banking environments, high-performance servers are used to handle thousands of transactions per second.

2. Minimum RAM

Minimum Requirement: 4 GB RAM

Recommended: 8 GB RAM or higher

Explanation:

RAM (Random Access Memory) is responsible for:

Running database software smoothly

Managing multiple transactions

simultaneously Supporting query execution

Handling background

processes If RAM is low:

System may become slow

Queries may take longer to execute

System may hang during heavy transaction processing

For better performance, 8 GB RAM is recommended because banking systems often deal with large datasets and multiple users.

3. Storage

Minimum Requirement: 10 GB free disk space

Explanation:

Storage is required for:

Installing database

software Storing

customer data Saving

transaction records

Maintaining log files

Creating backups

B **Computer / Laptop**

Minimum RAM

Storage

1.1 Software Requirements

Software	Purpose
MySQL Server	Database management
MySQL Workbench	Query execution
OS (Windows/Linux)	Platform
SQL	Query language

Explanation of the Software Stack

Each component plays a specific role in ensuring that a bank's transactions are handled without errors:

MySQL Server: Think of this as the "Brain" of the operation. It is the actual engine that handles data storage, security, and the ACID properties (Atomicity, Consistency, Isolation, Durability). In a banking system, the Server is responsible for making sure that if a system crashes mid-transfer, your money doesn't just vanish.

MySQL Workbench: This is the "Workspace" for the developer. Instead of typing into a black-and-white command prompt, Workbench allows you to visually design your tables, see the relationships between them (like how a Customer relates to an Account), and run test queries to see if your transaction logic works.

OS (Windows/Linux): This is the "Foundation." While the DBMS handles the data, the Operating System handles the hardware resources (like memory and CPU) that the MySQL Server needs to run. Banking systems often prefer Linux for its high-uptime stability, but Windows is common for development environments.

SQL (Structured Query Language): This is the "Language" of the project. It is the bridge between the user’ s intent and the data. Whether you are creating a table (DDL), inserting a new transaction (DML), or granting permissions to a

bank teller (DCL), SQL is the syntax used to make it happen.

6.SYSTEM DESIGN

6.1 System Architecture (Conceptual)

In a banking context, the architecture ensures that every operation— like transferring $100 from Alice to Bob— is treated as a single, indivisible unit of work.

How the User Interacts with the Database

The user doesn't talk to the database directly (that would be a security nightmare). Instead, they interact with a Client Interface (like a mobile app or web portal). When a user clicks "Transfer," the application sends a request to the Database Driver. This driver establishes a connection to the MySQL server, translating the user’ s intent into a structured SQL command.

Role of MySQL Server

MySQL acts as the "Source of Truth" and the "Enforcer." Its primary roles include:

Data Persistence: Ensuring that once a transaction is confirmed, it stays saved even if the power goes out.

Concurrency Control: Handling thousands of users checking their balances at the same time without data overlapping.

ACID Compliance: Specifically for banking, MySQL manages the Transaction Logs to ensure that if a transfer fails halfway through, the system "rolls back" to the previous state.

Query Execution Flow

Every time a query (like UPDATE accounts SET balance = balance - 100) is sent, it follows a specific path:

Parser: The server checks the SQL syntax for errors.

Optimizer: MySQL determines the most efficient way to find the data (e.g., using Indexes).

Executor: The server carries out the plan.

Storage Engine (InnoDB): This is where the actual data "writing" happens. It locks the specific rows being updated so no other process can mess with them until the transaction is complete.

7.DATABASE DESIGN

Database Design is the "blueprint" phase of your project. It is the process of deciding how to organize, store, and link data so that it is accurate, secure, and easy to retrieve.

Think of it like designing a physical bank building: before you lay any bricks, you need a floor plan that shows where the vault is, where the tellers sit, and how customers enter. In a database, that "floor plan" ensures that a customer's balance doesn't accidentally end up in someone else's account.

7.1 Entity Description

To manage a bank effectively, we need three core entities that represent the flow of value and ownership:

Customers: This entity stores the personal identification of the individuals using the bank. It acts as the anchor for all other data, ensuring that every account and transaction is legally tied to a verified user.

Accounts: This is the financial hub. It tracks the current "state" of a user's wealth. It links directly to the Customers entity and maintains the real-time balance and the specific type of account (e.g., Savings vs. Current).

Transactions: This is the most dynamic entity. It acts as a ledger that records every movement of funds. It doesn't just store "who" and "how much," but also the "when" and "status" (Pending, Completed, or Failed), which is vital for auditing.

Table Structure

This section defines exactly how data is categorized within the MySQL environment. We use specific data types like DECIMAL instead of FLOAT to avoid rounding errors during financial calculations.

Table: Customers

Attribute	Data Type	Description
customer_id	INT	Unique ID for each ban member.
full_name	VARCHAR(100)	The legal name of the account holder.
phone_number	VARCHAR(15)	Contact details for aler verification.
created_at	TIMESTAMP	The date the customer joined the bank.

Table: Accounts

Attribute	Data Type	Description
account_id	INT	Unique account number (e.g., 10-digit).
customer_id	INT	Links the account to a specific customer.
balance	DECIMAL(15,2)	Current funds (up to 15 digits, 2 decimal places).
status	VARCHAR(10)	Active, Frozen, or Closec

Table: Transactions

Attribute	Data Type	Description
`transaction_id`	INT	Unique reference nur for the receipt.
`sender_acc`	INT	The account the mor leaving.
`receiver_acc`	INT	The account receivin funds.
`amount`	DECIMAL(15,2)	The exact value bein moved.
`timestamp`	DATETIME	The exact second th transaction occurrec

Constraints Used

Constraints are the "guardrails" of your database. They prevent users (or buggy code) from entering nonsensical data, like a negative bank balance or a transaction to an account that doesn't exist.

1. Primary Key (PK)

Definition: A unique identifier for every row in a table.

Banking Use: Ensures no two customers have the same customer_id and no two transactions share a transaction_id. It makes searching for a specific record lightning- fast.

2. Foreign Key (FK)

Definition: A column that creates a link between two tables.

Banking Use: The customer_id in the Accounts table is a Foreign Key that points back to the Customers table. This ensures you can't create an account for a person who isn't a registered customer.

3. NOT NULL

Definition: Ensures that a column cannot be left empty.

Banking Use: You cannot have a transaction without an amount or an account without a balance. This constraint prevents "ghost" data from breaking your logic.

4. CHECK Constraint

Definition: Ensures that all values in a column satisfy a specific condition.

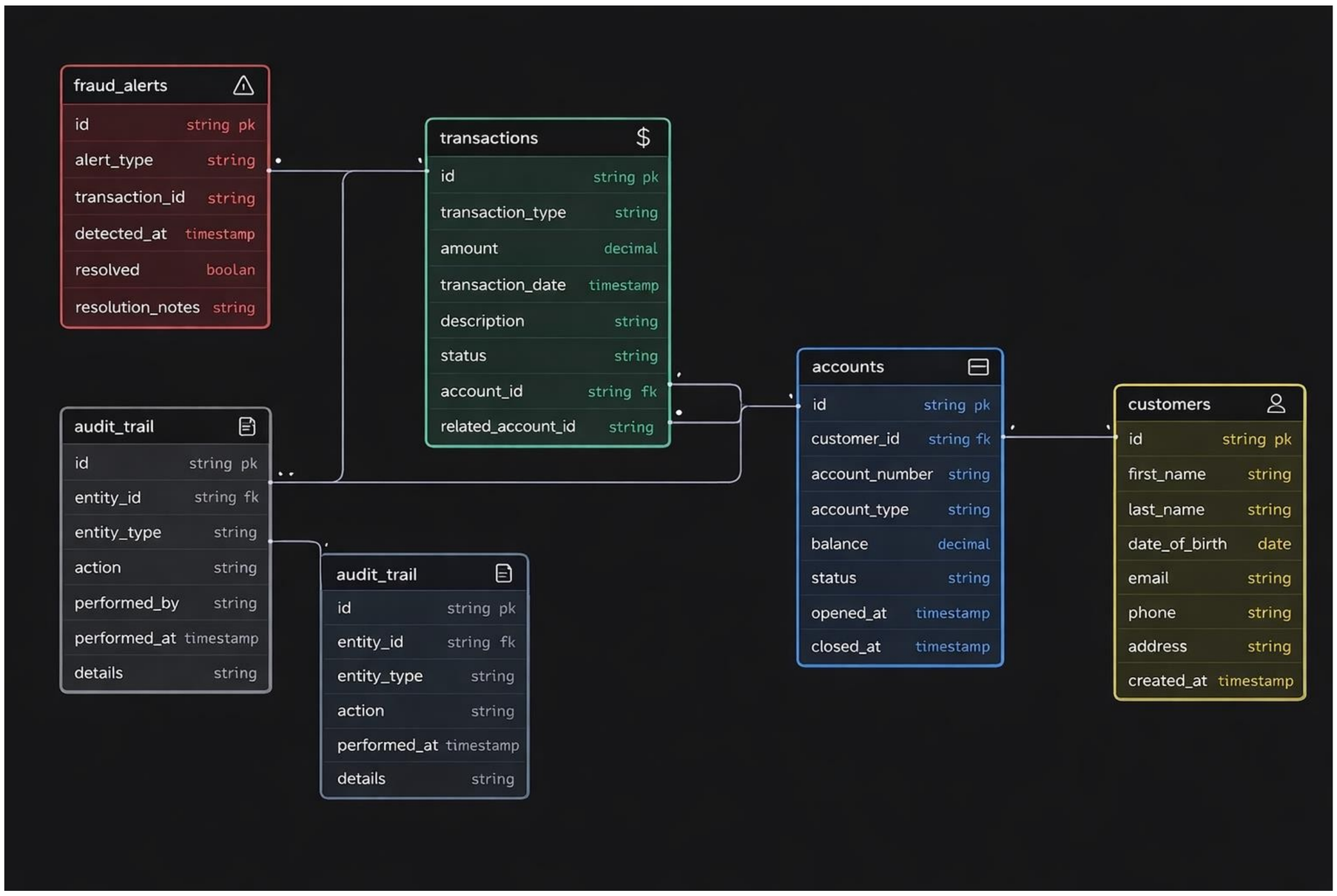
Banking Use: Used to ensure balance >= 0 (unless it’ s a credit account). It acts as a safety valve to stop a transaction if it would result in an illegal negative balance.

5. UNIQUE

Definition: Ensures all values in a column are different.

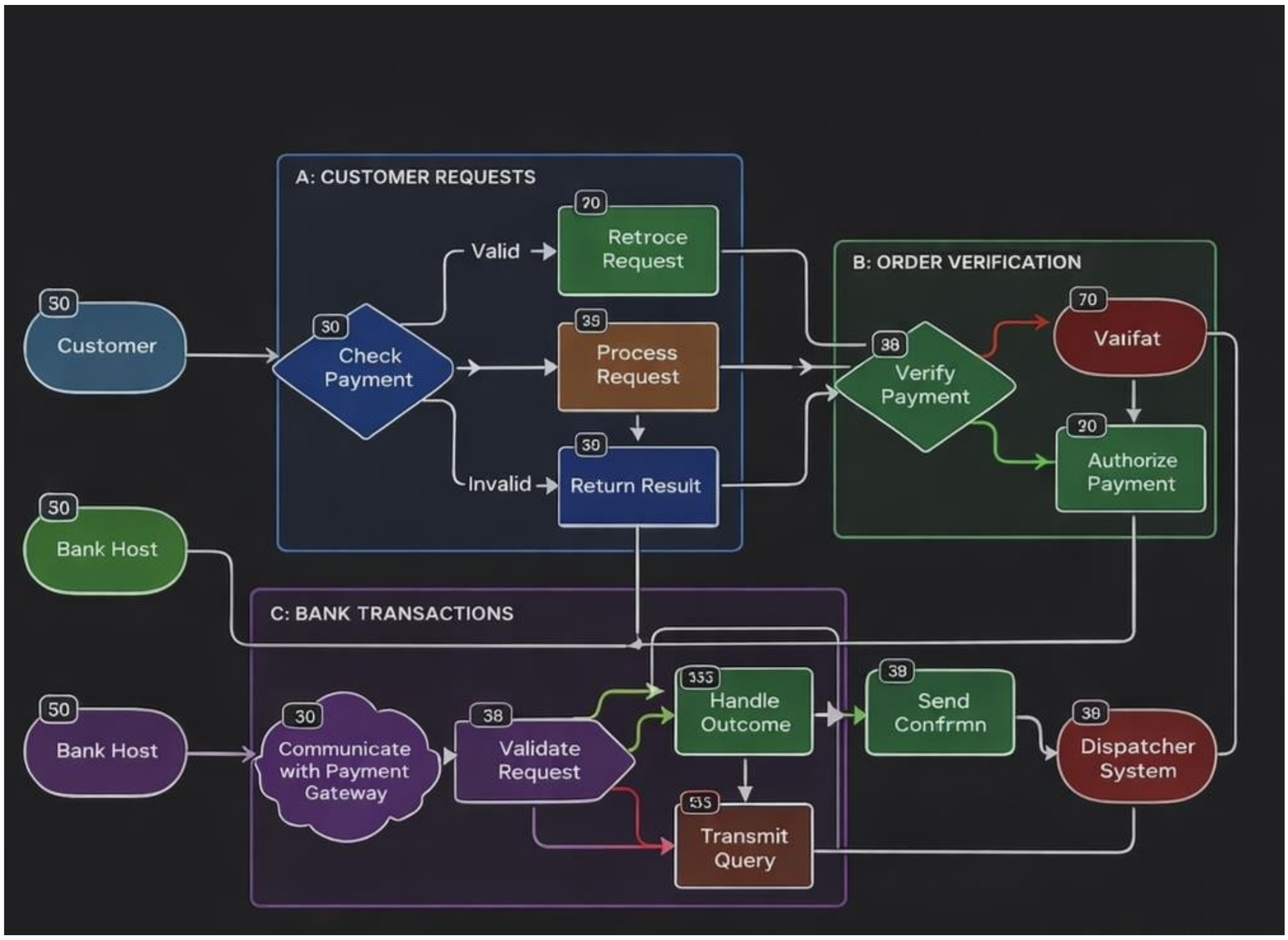
Banking Use: Often applied to email or phone_number so that two different people cannot sign up using the same contact information.

8.UML DIAGRAMS

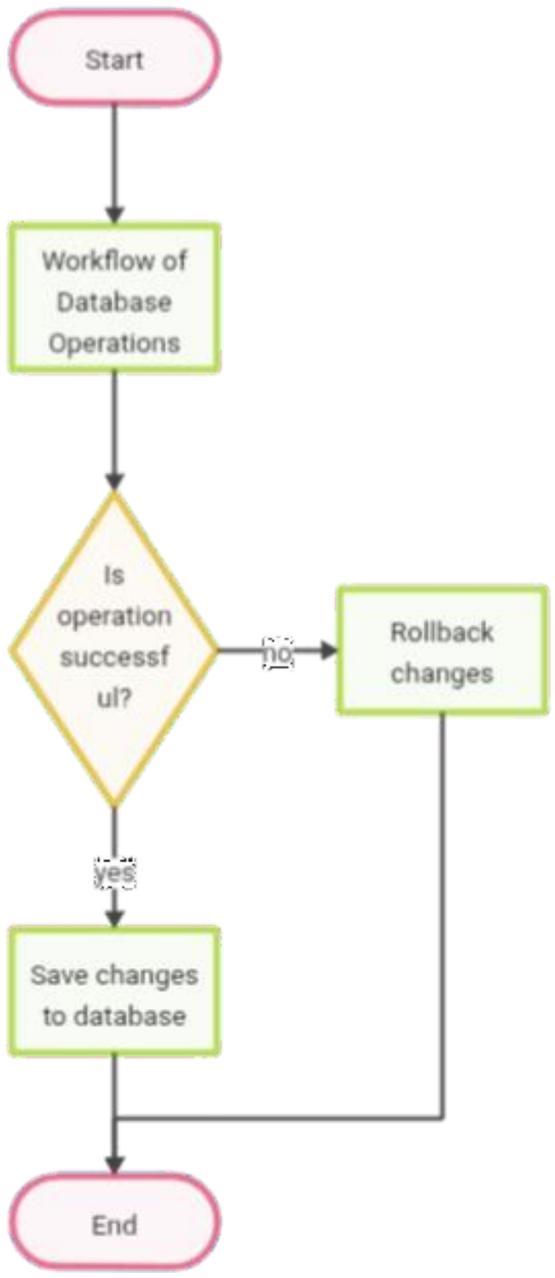


8.1 ER Diagram

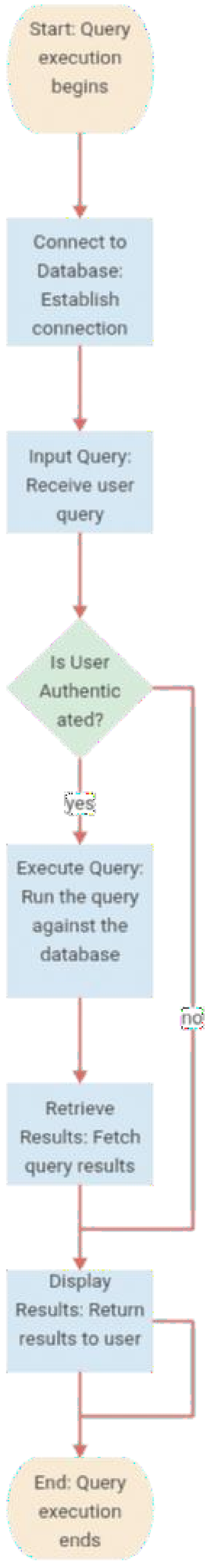
8.2 Use Case Diagram



8.2 Activity Diagram



8.3 Sequence Diagram



9.SQL IMPLEMENTATION

```
CREATE DATABASE banking_system; USE

banking_system;



CREATE TABLE account ( account_id

    INT PRIMARY KEY,

    customer_name VARCHAR(50),

    balance INT

);



CREATE TABLE bank_transaction (

    txn_id INT PRIMARY KEY,

    from_account INT,

    to_account INT,

    amount INT,

    txn_date DATE

);



-- Fixed: Use single quotes for strings and commas between value

sets INSERT INTO account VALUES
```

(101, 'Ravi', 5000),

(102, 'Neha', 3000),

(103, 'Manjeet', 6000),

(104, 'Suresh', 4500);

-- Scenario 1: Successful Transfer (COMMIT)

START TRANSACTION;

UPDATE account SET balance = balance - 1000 WHERE account_id = 101;

UPDATE account SET balance = balance + 1000 WHERE account_id = 102;

INSERT INTO bank_transaction VALUES (1, 101, 102, 1000, CURDATE());

COMMIT;

-- Scenario 2: Failed/Invalid Transfer (ROLLBACK)

START TRANSACTION;

UPDATE account SET balance = balance - 7000 WHERE account_id = 101;

INSERT INTO bank_transaction VALUES (2, 101, 102, 7000, CURDATE());

-- We rollback because Ravi doesn't have 7000 (Insufficient funds logic)

ROLLBACK;

SELECT * FROM account; SELECT *

FROM bank_transaction;

10.SYSTEM TESTING AND RESULT

In transaction management, we test to ensure the database follows ACID properties.

1. Query Correctness Testing

This testing verifies that the SQL engine interprets the commands exactly as intended.

Syntax Check: Ensuring START TRANSACTION, COMMIT, and ROLLBACK are placed correctly.

Logic Check: Verifying that the UPDATE statement actually targets the correct account_id and that the INSERT into the transaction log matches the amounts moved.

2. Data Validation

This ensures the data remains logical and within constraints during a

move. Primary Key Integrity: Testing that txn_id 1 cannot be reused.

Balance Validation: In your code, the second transaction is "invalid" because Ravi's balance (4000 after the first txn) cannot cover a 7000 withdrawal. The ROLLBACK ensures the data stays valid by reverting the mistake.

Output Verification

This is the final check where we compare the "Before" and "After" states of the database.

Stage	Account 101 (Ravi)	Account 102 (Neha)
Initial	5000	3000
After Commit	4000	4000
After Rollback	4000	4000

Sample Result Explanation

Output Screenshot (Simulated)

Query: `SELECT * FROM account;`

text

```
+------------+----------------+---------+
| account_id | customer_name | balance |
+------------+----------------+---------+
|        101 | Ravi           |    4000 |
|        102 | Neha           |    4000 |
|        103 | Manjeet        |    6000 |
|        104 | Suresh         |    4500 |
+------------+----------------+---------+
```

```
+--------+---------------+------------+---------
+------------+
| txn_id | from_account | to_account | amount
| txn_date   |
+--------+---------------+------------+---------
+------------+
|      1 |           101 |        102 |   1000
| 2026-02-17 |
+--------+---------------+------------+---------
+------------+
```

Analysis of Results:

The Success: Ravi started with 5000. After the first transaction, his balance correctly shows 4000, and Neha's increased from 3000 to 4000. The COMMIT ensured this was saved.

The Failure (The Rollback): The second transaction tried to take 7000 from Ravi. If this had succeeded, Ravi would have a balance of -3000. Because we issued a ROLLBACK, the database ignored those commands. Ravi's balance stayed at 4000, and Transaction #2 was never recorded in the log.

11.SECURITY, BACKUP AND RECOVERY

Security: Protecting the Vault

In a banking environment, not everyone should have the power to DELETE accounts or UPDATE balances. We manage this through User Privileges and Access Control.

User Privileges

MySQL uses the GRANT and REVOKE commands to define what a user can do. This follows the Principle of Least Privilege (PoLP)— giving a user only the access they strictly need.

Administrator (DBA): Has ALL PRIVILEGES (Can create tables, drop databases).

Bank Teller: Might only have SELECT, INSERT, and UPDATE privileges on the bank_transaction table.

Auditor: Might only have SELECT (Read-only) access to view logs without the ability to change them.

Access Control

This limits where and how a user connects.

Host-based security: Restricting a user to connect only from a specific IP address (e.g., teller@192.168.1.50).

Password Policies: Enforcing complex passwords and periodic rotations to prevent unauthorized entry.

Backup: The Safety Net

A backup is a representative snapshot of your database at a specific point in time. In MySQL, the industry standard tool is mysqldump.

mysqldump Explanation

mysqldump is a command-line utility that performs a Logical Backup. It produces a .sql file containing all the SQL statements (CREATE TABLE, INSERT INTO, etc.) required to rebuild the database from scratch.

Recovery: The Restoration

Recovery is the process of restoring the database to a functional state after data loss, a system crash, or a botched transaction.

Restore Concept

If the banking_system database is accidentally deleted or corrupted, we use the backup file created by mysqldump to "replay" the history of the database.

The Recovery Process:

Create a blank database: CREATE DATABASE banking_system;

Import the backup: Direct the .sql file back into MySQL.

12.FUTURE SCOPE AND CONCLUSION

Future Scope: Beyond the Console

While the current system handles core logic, a modern banking platform requires more layers to be truly functional and user-friendly.

Web Integration

The most immediate step is moving the database from a command-line interface to a Web- based User Interface (UI).

Technology Stack: Integration with a backend like Node.js, Python (Django/Flask), or PHP allows users to perform transactions through a browser.

Security Layers: Implementing SSL/TLS encryption and Multi-Factor Authentication (MFA) for logins to protect the account table from web-based attacks.

API Development: Creating RESTful APIs so that mobile apps can safely query balances and initiate transfers.

Advanced Reporting

Currently, we use simple SELECT statements. Future versions would include:

Automated Monthly Statements: Generating PDF reports of all records in the bank_transaction table for a specific account_id.

Audit Trails: Expanding the transaction table to include metadata like IP addresses, device IDs, and geographic locations to track every change.

Scheduled Tasks: Using MySQL Events to automatically calculate and apply monthly interest to all account balances.

Conclusion:

The Banking System Transaction Management project demonstrates the critical importance of data integrity in financial environments. By utilizing MySQL’ s transaction controls— specifically START TRANSACTION, COMMIT, and ROLLBACK— we ensure that money is never "lost" during a transfer.

Key Takeaways:

Reliability: The system guarantees that either both sides of a transfer succeed or neither does, maintaining the Atomicity of the bank's ledger.

Security: Through strict user privileges and robust backup strategies like mysqldump, the system is protected against both malicious intent and hardware failure.

Scalability: The structured nature of the account and bank_transaction tables provides a solid foundation for adding advanced web features and real-time

analytics.

13.REFERENCES

MySQL Official Documentation

Oracle Corporation. (2026). MySQL 8.0 Reference Manual. Retrieved from https://dev.mysql.com/doc/refman/8.0/en/

Used for: ACID compliance details, START TRANSACTION, COMMIT, and ROLLBACK syntax.

Oracle Corporation. (2026). MySQL Workbench Manual. Retrieved from https://dev.mysql.com/doc/workbench/en/

Used for: Database modeling and query execution testing.

14.GLOSSARY

DBMS (Database Management System): Software that interacts with end users, applications, and the database itself to capture and analyze data. In this project, it serves as the engine for storing account and transaction details.

SQL (Structured Query Language): The standard programming language used to manage and manipulate relational databases. It is the language used to write your UPDATE and INSERT commands.

Primary Key: A specific column (like account_id or txn_id) that uniquely identifies each record in a table. It prevents duplicate entries, ensuring no two customers share the same ID.

Foreign Key: A column or group of columns in one table that provides a link between data in two tables. In your system, from_account in the transaction table acts as a reference to the account_id in the account table.

MySQL: An open-source Relational Database Management System (RDBMS) that uses SQL. It is the specific platform used to execute your banking transactions.

ACID Properties: An acronym (Atomicity, Consistency, Isolation, Durability) representing the four key properties that guarantee database transactions are processed reliably.

Transaction: A single logical unit of work that accesses and possibly modifies the contents of a database (e.g., moving $1000 from Ravi to Neha).

Commit: A command that saves all changes made during the current transaction permanently to the database.

Rollback: A command that undoes all changes made during the current transaction, returning the database to its previous state if an error occurs.

Data Integrity: The overall accuracy, completeness, and consistency of data. Transaction management ensures that the "Total Bank Balance" remains the same before and after a transfer.

mysqldump: A utility used to create a logical backup of the database by generating a series of SQL statements that can be used to recreate the original database.