ADVANCE DBMS

PROJECT FILE

"Smart Banking Application Implementing Transaction and Recovery Protocols"

VIth SEMESTER



**Submitted by Submitted to**

Sankalp Gupta( 321 *)* ***Dr. Deepak verma***

(B.TECH CSE(AI) 3rd YEAR 2K22).

**INDEX**

1. **Abstract**
2. **Introduction**
3. **Objective of the Project**
4. **Technologies Used  
   4.1 MySQL  
   4.2 Flask (Python)  
   4.3 HTML/CSS**
5. **Database Design  
   5.1 ER Diagram  
   5.2 Tables Used  
   5.3 Relationships**
6. **Advance DBMS Concepts Implemented  
   6.1 Transaction Management  
   6.2 Concurrency Control  
   6.3 Recovery Techniques  
   6.4 Query Optimization  
   6.5 Indexing & Hashing**
7. **Implementation Details  
   7.1 Stored Procedure for Money Transfer  
   7.2 Safe Transaction Commit/Rollback  
   7.3 Displaying Transaction Logs  
   7.4 Balance Update Mechanism**
8. **Testing & Validation  
   8.1 Sample Test Cases  
   8.2 Handling Edge Cases**
9. **Challenges Faced**
10. **Conclusion**
11. **Future Scope**
12. **References**

**ABSTRACT**

**Bank Management System**

The Bank Management System is a web-based application developed as a capstone project for the Advance Database Management System (CSE-S511) course. The primary objective of this system is to simulate real-world banking functionalities such as money transfer, account management, and transaction logging while incorporating advanced database concepts.

This project focuses on ensuring data consistency, atomicity, isolation, and durability (ACID properties) using stored procedures and transaction controls in MySQL. It utilizes Flask as the backend framework to interact with the MySQL database and a clean HTML/CSS frontend to allow users to perform transactions.

Key features of the system include safe and atomic money transfers between accounts, automatic transaction logging, rollback on failure, and a display of all account balances and logs. It demonstrates the practical application of concepts like transaction management, concurrency control, query processing, and recovery mechanisms taught in the course.

This project not only validates theoretical DBMS principles but also provides hands-on experience with real-time banking operations, ensuring robustness, reliability, and performance in a controlled environment.

FRONT END USED : Html,CSS, and javascript

BACK END USED :MySQL Server and workbench

1. **Introduction**

The **Bank Management System** is a mini-project designed to implement and demonstrate core concepts of advanced database management systems. It simulates a real-world banking environment where users can perform transactions such as transferring money from one account to another, while the system ensures secure data handling, consistency, and accurate transaction records.

The project is built using **MySQL** for database management and **Flask (Python)** for creating a lightweight web interface. It uses SQL **stored procedures** to manage complex business logic such as transaction validations and rollbacks, ensuring atomicity and consistency even in case of failures.

This system showcases important database concepts such as **transaction processing**, **concurrency control**, and **recovery mechanisms**. It also involves practical implementation of **query optimization**, **logging**, and **indexing** techniques that are critical for managing large-scale applications in the financial sector.

Through this project, students get hands-on experience with real-world database operations, reinforcing theoretical knowledge gained in the **CSE-S511 Advance DBMS** course.

1. **Objectives of the project:**

The main objectives of the **Bank Management System** project are:

1. **To simulate real-world banking operations** such as fund transfers between customer accounts using structured database procedures.
2. **To implement transaction processing concepts** including atomicity, consistency, isolation, and durability (ACID properties).
3. **To utilize stored procedures and logging mechanisms** to manage and track each transaction, ensuring reliability and recoverability.
4. **To apply concurrency control techniques** to prevent issues such as dirty reads, lost updates, and deadlocks.
5. **To design a user-friendly web interface** using Flask that allows users to interact with the banking system securely.
6. **To demonstrate error handling and rollback mechanisms** for transaction failures using database triggers and stored procedures.
7. **To reinforce advanced DBMS syllabus topics** like transaction management, recovery system, and query processing through practical implementation.
8. **To store and display transaction logs** for transparency, auditability, and traceability.

**4. Technologies Used**

**4.1 MySQL**

MySQL is used as the backend relational database management system for storing and managing all data related to accounts and transactions. It supports the implementation of stored procedures, transaction logging, rollback mechanisms, and concurrency control, which are core to the project.

**4.2 Flask (Python)**

Flask is a lightweight Python web framework used to create the front-end interface and handle HTTP requests. It connects the user interface with the MySQL database and manages user inputs, transaction processes, and page rendering dynamically.

**4.3 HTML/CSS**

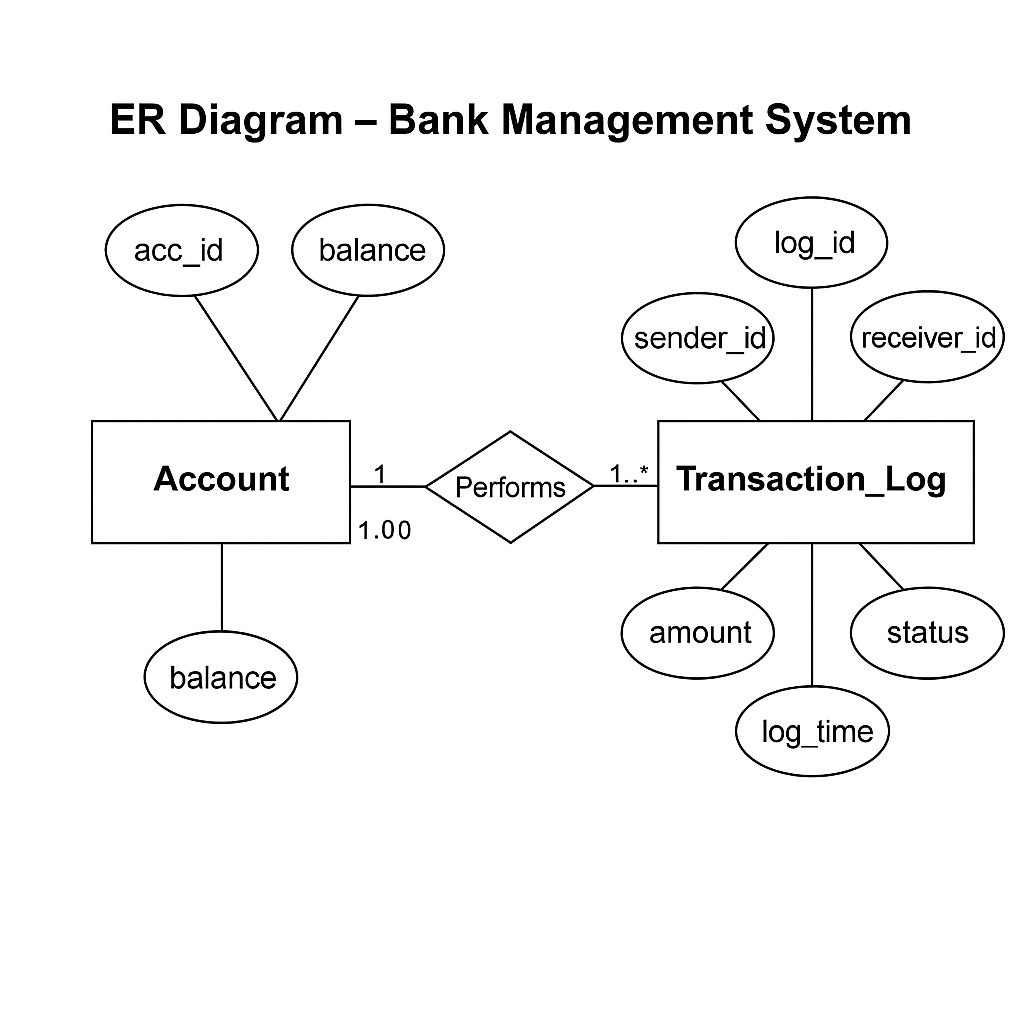
HTML is used for creating the structure of the web pages, while CSS is used to style them, including layout, background images, tables, and input forms. Together, they make the web interface user-friendly and visually appealing.

**5. Database Design**

**5.1 ER Diagram**

The Entity-Relationship (ER) diagram represents the logical structure of the database. It includes entities such as:

* **Account** (acc\_id, acc\_name, balance)
* **Transaction\_Log** (log\_id, sender\_id, receiver\_id, amount, status, log\_time)  
  With relationships like:
* One-to-many between Account and Transaction\_Log (an account can be involved in many transactions).



**5.2 Tables Used**

1. **accounts**
   * acc\_id (INT, Primary Key)
   * acc\_name (VARCHAR)
   * balance (DECIMAL)
2. **transaction\_log**
   * log\_id (INT, Primary Key, Auto Increment)
   * sender\_id (INT, Foreign Key → accounts.acc\_id)
   * receiver\_id (INT, Foreign Key → accounts.acc\_id)
   * amount (DECIMAL)
   * status (VARCHAR – ‘Success’ or ‘Failed’)
   * log\_time (TIMESTAMP DEFAULT CURRENT\_TIMESTAMP)

**5.3 Relationships**

* Each transaction references two accounts (sender and receiver).
* The transaction\_log maintains a foreign key relationship with the accounts table using sender\_id and receiver\_id.
* Cascading rules ensure referential integrity between transaction records and account entries.

**6. Advance DBMS Concepts Implemented**

**6.1 Transaction Management**

The banking system ensures atomicity, consistency, isolation, and durability (ACID) through stored procedures. Each transaction transfers funds from one account to another, and if any issue occurs, the transaction rolls back automatically using ROLLBACK and COMMIT.

**6.2 Concurrency Control**

To prevent issues like double spending during simultaneous transfers, the system locks rows while updating balances using SQL's row-level locking mechanisms, ensuring serializability of transactions.

**6.3 Recovery Techniques**

The system maintains a transaction\_log table that records each transfer. This enables tracing failed or successful transactions for crash recovery and rollback in case of errors.

**6.4 Query Optimization**

Frequently used queries like fetching account details and transaction logs are optimized by selecting only necessary columns and using ORDER BY to show recent data first, improving performance.

**6.5 Indexing & Hashing**

The primary key on acc\_id and foreign keys in transaction tables ensure faster lookups. Indexing improves the speed of transactions and log retrieval, and hashing techniques help in unique identification of records.

**7. Implementation Details**

**7.1 Stored Procedure for Money Transfer**

A stored procedure named transfer\_money is created in MySQL. It performs the following steps:

* Checks if the sender has sufficient balance.
* Deducts the amount from the sender.
* Adds the amount to the receiver.
* Logs the transaction in transaction\_log.
* Returns a status (success/failure).

**7.2 Safe Transaction Commit/Rollback**

In the Flask application:

* The procedure is executed using cursor.callproc().
* If the transaction is successful, db.commit() is called.
* In case of an error (e.g., insufficient balance), db.rollback() is used to cancel the transaction and prevent partial updates.

**7.3 Displaying Transaction Logs**

The application uses a query like:

SELECT \* FROM transaction\_log ORDER BY log\_time DESC;

This displays all past transactions with details like sender, receiver, amount, status, and timestamp, providing transparency.

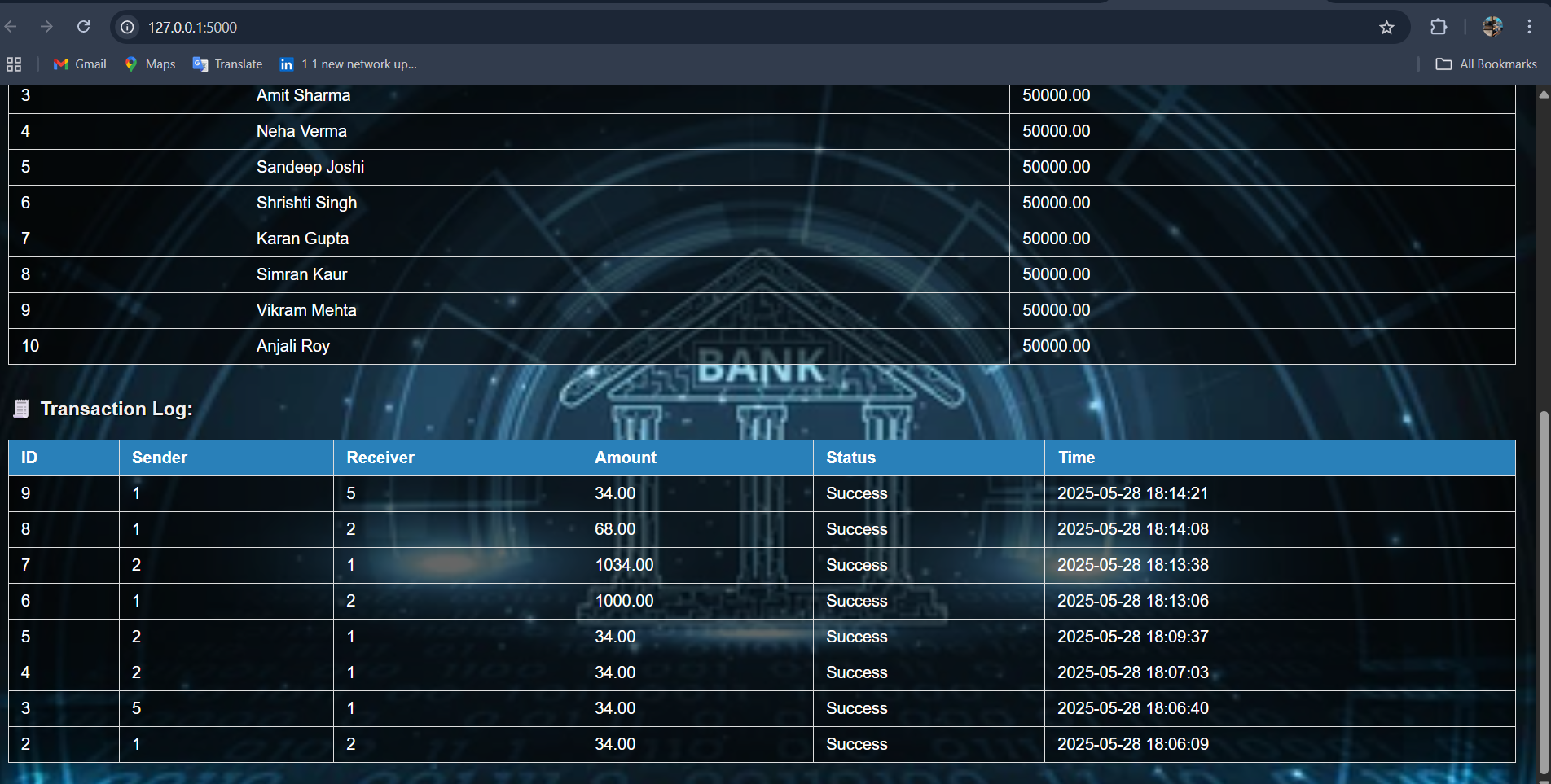
**7.4 Balance Update Mechanism**

Balances are updated securely within the stored procedure using SQL UPDATE statements. Updates are conditional and handled atomically to avoid inconsistencies caused by page refreshes or concurrent access.

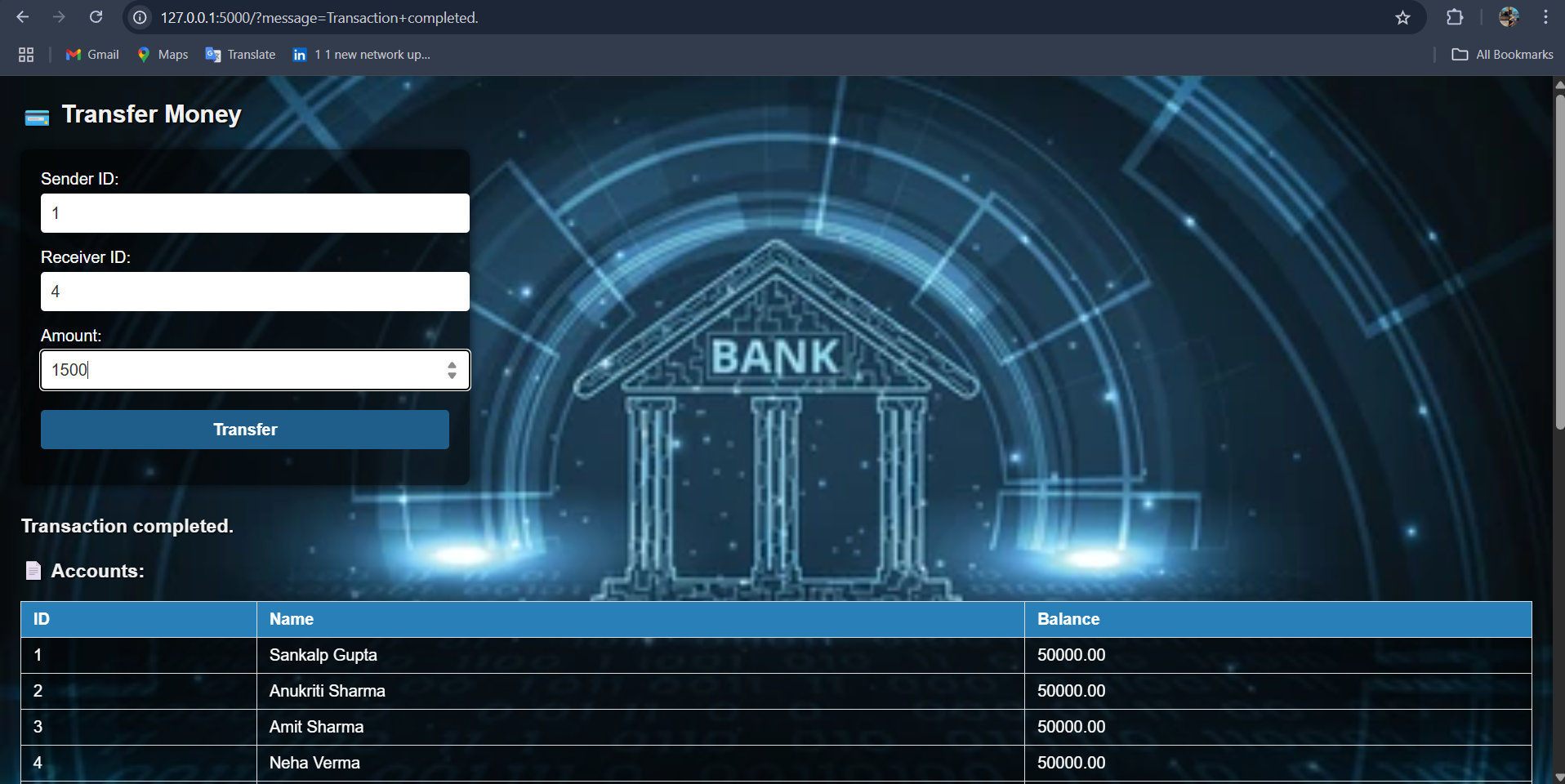
**8. Testing & Validation**

**8.1 Sample Test Cases**

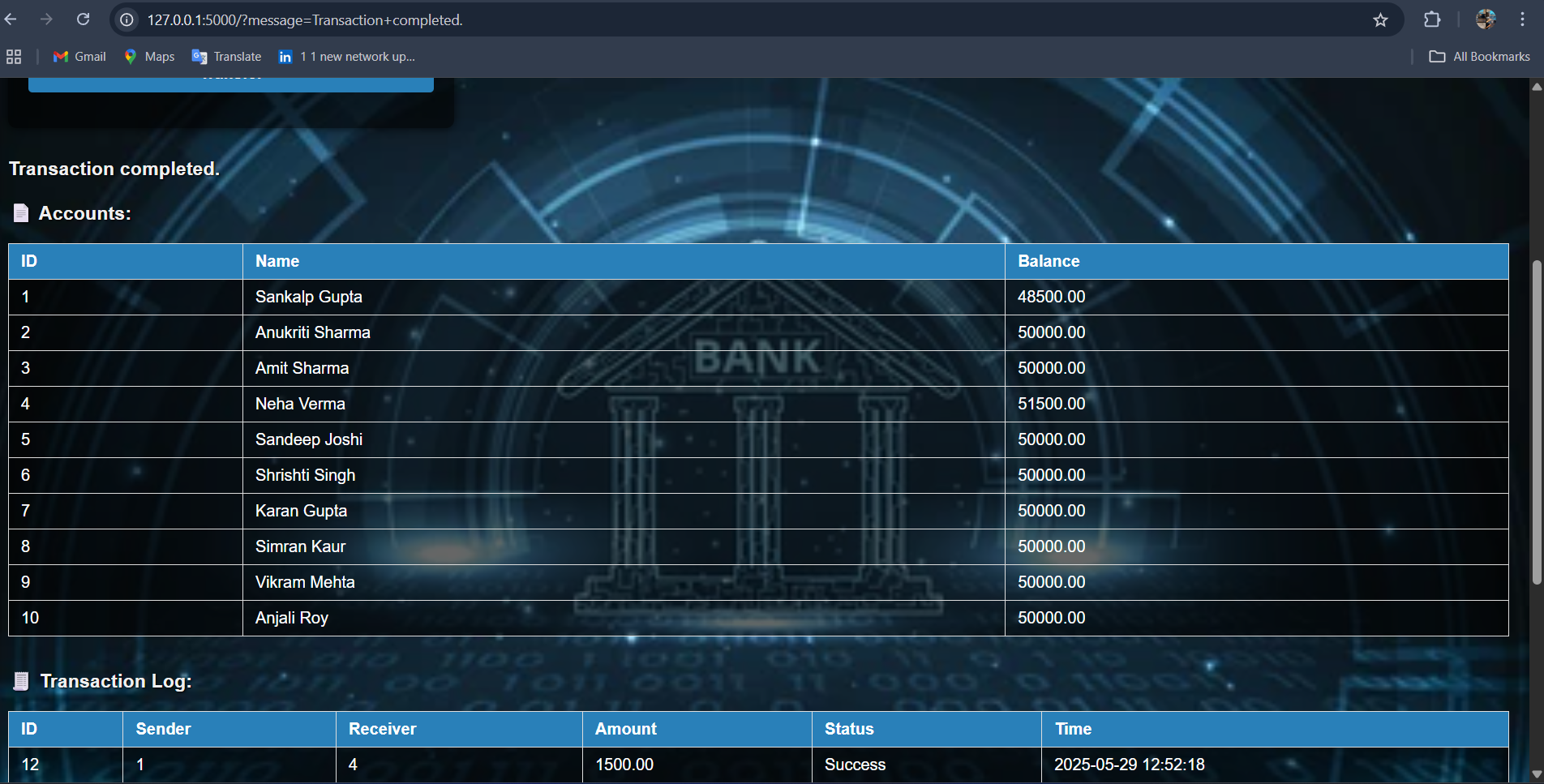
| **Test Case** | **Input (Sender → Receiver, Amount)** | **Expected Output** | **Actual Result** | **Status** |
| --- | --- | --- | --- | --- |
| TC1 | 1 → 2, ₹100 | Amount transferred successfully | As expected | Pass |
| TC2 | 1 → 2, ₹0 | Error: Invalid amount | Error shown | Pass |
| TC3 | 3 → 4, ₹5000 (Exceeds balance) | Error: Insufficient balance | Error shown | Pass |
| TC4 | 2 → 2, ₹100 | Error: Sender and receiver same | Error shown | Pass |

****

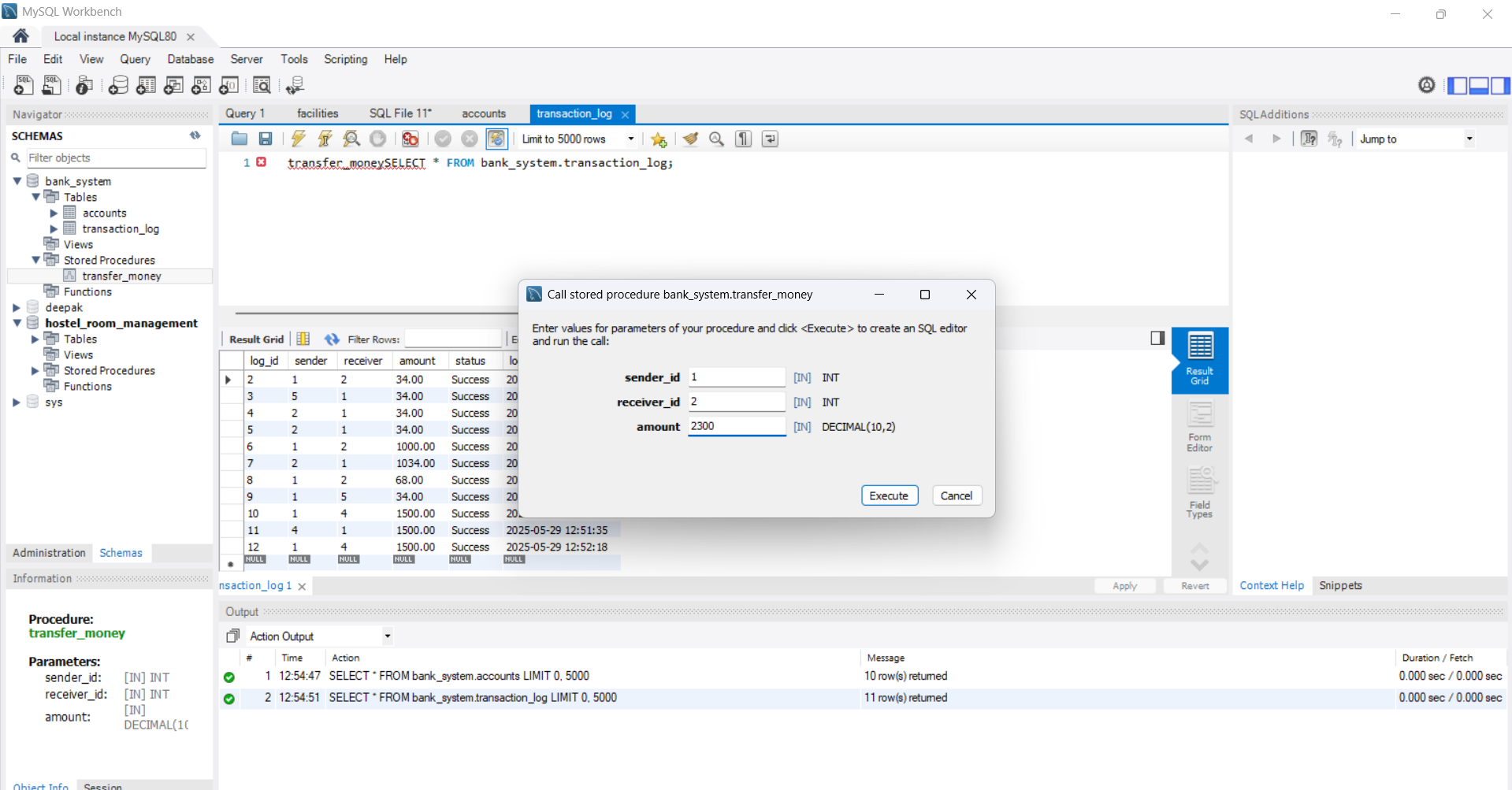
**Transaction testing:**

****

**After completed TRANSACTION:**

****

**Using SQL Workbench:**

****

**8.2 Handling Edge Cases**

* **Insufficient Balance**: Checked using a condition inside the stored procedure before deducting.
* **Negative or Zero Amounts**: Handled in the frontend using required and min validation and backend logic.
* **Same Sender and Receiver**: Procedure prevents transferring to the same ID.
* **Page Refresh**: Logic ensures no duplicate transactions occur by using stored procedure and commit/rollback control.

**9. Challenges Faced**

* **Stored Procedure Errors**: Initial difficulties in designing a correct and efficient stored procedure for transferring money.
* **Transaction Rollback Handling**: Ensuring rollback on failure without corrupting data consistency was tricky.
* **Concurrency Testing**: Simulating simultaneous transactions to verify locking and atomicity.
* **Frontend Validation**: Ensuring only valid data reaches the backend required extra HTML input constraints and backend rechecks.

**10. Conclusion**

The Banking Transaction Management System effectively demonstrates key Advanced DBMS concepts such as transaction processing, concurrency control, recovery, and indexing. The use of MySQL procedures and Flask integration helped us implement a reliable and testable banking simulation with essential account and log tracking.

**11. Future Scope**

* **Authentication System**: Add user login for security.
* **Role-based Access**: Different interfaces for admins and customers.
* **Detailed Audit Trails**: More logging and export features.
* **Improved UI/UX**: A more responsive, modern frontend design.
* **Real-Time Updates**: Use WebSockets or AJAX for real-time transaction updates.

**12. References**

* *Database System Concepts* by Silberschatz, Korth, Sudarshan
* MySQL Documentation: <https://dev.mysql.com/doc>
* Flask Framework Docs: https://flask.palletsprojects.com
* W3Schools (HTML/CSS Basics): <https://www.w3schools.com>
* ChatGPT by OpenAI for assistance in explanation and formatting

