ABSTRACT

The Decentralized Transaction Verification System presents a secure and tamper-resistant approach to managing financial transactions within a relational database environment. Traditional centralized database systems are susceptible to data tampering, fraud, and single points of failure. This project addresses these vulnerabilities by simulating blockchain-like features using MySQL, PL/SQL and SHA2 functions. Core functionalities include cryptographic hashing (SHA-256), sequential block linking, and a Proof-of-Work (PoW) simulation for transaction validation.

The system architecture consists of four key tables: Users, Wallets, Transactions, and Blocks, each designed to enforce data integrity and traceability. Transactions are cryptographically hashed and linked in a chain-like structure, ensuring immutability. Triggers and stored procedures validate each transaction, check for sufficient wallet balance, and prevent double-spending. Additionally, a basic frontend and hosted with streamlit, facilitates user interaction.

This SQL-based solution demonstrates how robust security, verifiability, and data consistency can be achieved without relying on external blockchain frameworks. Future enhancements include integrating Merkle trees for optimized querying and automated auditing features. The project showcases an innovative method for secure transaction management entirely within the bounds of relational database systems.

Chapter 1

Introduction

Database systems form the backbone of modern digital infrastructure, enabling the structured storage, retrieval, and management of vast volumes of data. Among their most critical applications is in the handling of financial transactions, where accuracy, consistency, and security are paramount. Traditional Relational Database Management Systems (RDBMS) such as Oracle SQL are designed to ensure data integrity through well-defined schemas, constraints, and transaction control mechanisms. However, these centralized systems are inherently vulnerable to risks such as unauthorized access, data tampering, and single points of failure.

As digital transactions continue to grow in scale and sensitivity, there is a pressing need for more robust solutions that go beyond conventional database safeguards. This has given rise to decentralized models like blockchain, which introduce features such as cryptographic hashing, data immutability, and consensus mechanisms to secure transactions. However, integrating such advanced features directly into a traditional SQL-based system remains an area of active exploration.

In this context, our project, Decentralized Transaction Verification System, bridges the gap between conventional relational databases and decentralized security principles. By leveraging MySQL, PL/SQL and SHA2 we design and implement a tamper-proof transaction management system that simulates core blockchain features—such as hashing, linked blocks, and a Proof-of-Work (PoW) mechanism—within a familiar SQL environment. This innovative approach ensures that transactions are validated, securely stored, and immutable, all while maintaining the structure and efficiency of a traditional database system.

Through this project, we aim to demonstrate how relational databases can be enhanced with cryptographic techniques to offer blockchain-level security, providing a scalable and practical solution for secure transaction verification in real-world applications.

Chapter 2

Synopsis

2.1 Proposed System

The proposed system simulates a simplified blockchain using MySQL to model essential blockchain functions such as transaction validation, mining, balance updates, and activity logging.

2.1.1. Transaction Validation

Transactions are validated through a `BEFORE INSERT` trigger on the `blockchain_transactions` table. The trigger ensures:

- The sender has sufficient balance (including transaction fees).
- Transactions are not self-directed.
- Amounts and fees are positive.

Invalid transactions are blocked and logged with appropriate messages for audit purposes.

2.1.2. Mining and Block Formation

A stored procedure handles the mining process by:

- Selecting valid transactions from the mempool.
- Grouping them into a new block.
- Crediting the miner with a fixed reward and the total transaction fees.
- Updating balances of all involved parties.

A system-generated reward transaction is added to the block for transparency.

2.1.3. Balance Calculation

Balances are computed dynamically by aggregating all incoming and outgoing transactions for each address, ensuring real-time consistency and eliminating the need for manual updates.

2.1.4. Logging and Automation

All valid and invalid actions are logged with timestamps and memos. Triggers ensure automation of:

- Transaction validation,
- Reward distribution,
- And balance enforcement.

2.1.5. Extensibility

Though implemented centrally, the system mimics key blockchain principles and can be extended to support cryptographic hashing, timestamping, and smart contracts in future iterations.

2.2 Objectives

2.2.1 Simulate Blockchain Functionality using SQL: To create a simplified yet functional model of a blockchain using MySQL database features like tables, triggers, and stored procedures.

2.2.2 Implement Block Mining Mechanism: To design a procedure that simulates the mining of blocks, including timestamping and chaining through hash references.

2.2.3 Enable Transaction Handling with Fees and Memos: To allow users to make transactions that include custom memos and fees, mimicking real-world use cases of blockchain.

2.2.4 Distribute Mining Rewards Automatically: To automate the allocation of rewards to miners after a block is successfully mined.

2.2.5 Ensure Data Integrity and Security: To enforce transaction validation and balance checks using triggers, ensuring the system mimics the immutability and reliability of an actual blockchain.

2.2.6 Track and Log All Activities: To maintain logs of all transactions and block creation events for audit and analysis purposes.

Chapter 3

Functional Requirements

3.1 Database Tablese

3.1.1 Users Table: Stores user account details including user ID and username.

3.1.2 Transactions Table: Stores transaction data such as sender, receiver, amount, fee, memo, status, and timestamp.

3.1.3 Blocks Table: Maintains block metadata such as block ID, miner ID, timestamp and total reward.

3.1.4 Wallets Table: Tracks wallets of every user to make transactions.

3.2. Stored Procedures

3.2.1 make_transaction(sender, receiver, amount, fee, memo): Adds a transaction request to the transactions table.

3.2.2 mine_block(miner_id): Groups unconfirmed transactions into a block, distributes rewards and fees to the miner, and confirms transactions.

3.3 Triggers

3.3.1 validate_transaction_before_insert: Validates transactions upon insertion to ensure:

- Sender exists
- Receiver exists
- Sufficient balance
- Positive amount and fee
- Prevents invalid or malicious transactions

3.4 Functions

3.4.1 generate_hash: Generates hash codes to encrypt all transactions using the SHA256 function.

3.5 Security & Logging

3.5.1 Status Tracking: Each transaction has a status field to denote success or failure.

3.5.2 Auditability: Failed transactions are retained with a reason, ensuring transparency and accountability.

Chapter 4

Detailed Design

4.1 ER Diagram

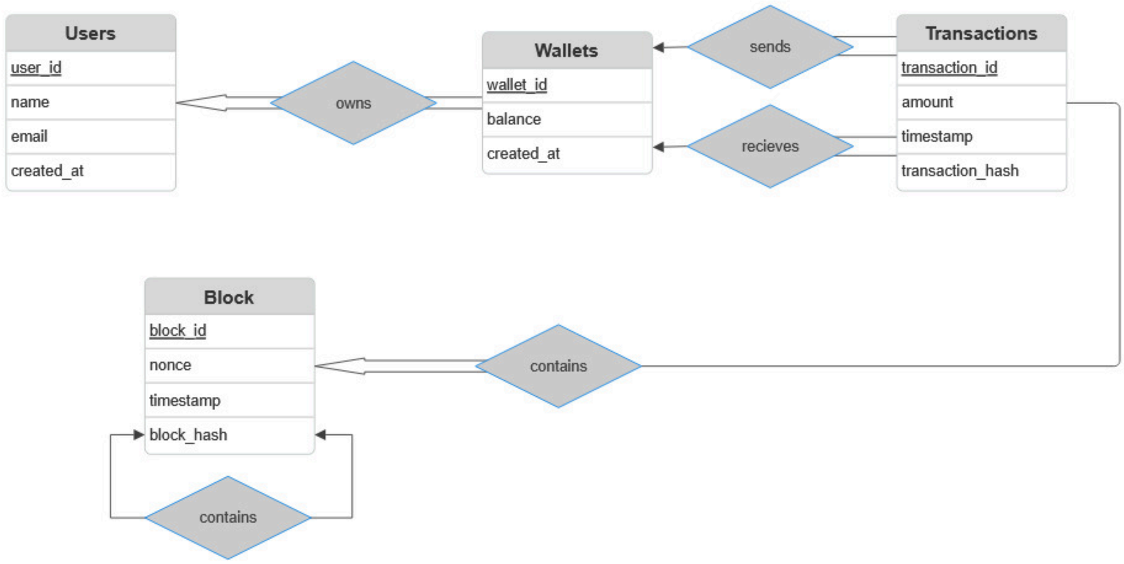

Figure 4.1 ER Diagram

4.2 Schema Diagram

Users (user_id, name, email, created_at)

Wallets (wallet_id, user_id, balance, created_at)

Transactions (transaction_id, block_id, amount, sender_wallet_id, reciever_wallet_id, timestamp, transaction_hash)

Blocks (block_id, block_hash, previous_block_id, timestamp, nonce)

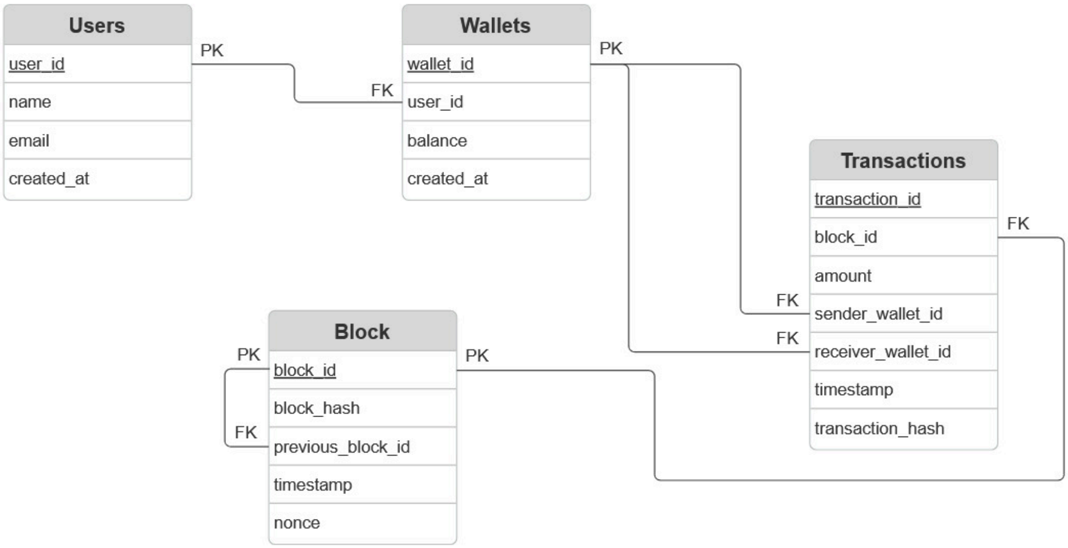

Figure 4.2 Schema Diagram

4.3 Data Dictionary

USERS

Column	Data type (size)	Constraint	Constraint Name
user_id	int	Primary Key, AUTO_INCREEMENT	
name	varchar(100)	NOT NULL	
email	varchar(255)	NOT NULL, UNIQUE	
password	varchar(255)	NOT NULL	
created_at	TIMESTAMP	DEFAULT, CURRENT_TIMESTAMP	

WALLETS

Column	Data type (size)	Constraint	Constraint Name
wallet_id	int	Primary Key, AUTO_INCREEMENT	
user_id	int	NOT NULL, REFERENCES USERS	
balance	decimal(20.8)	NOT NULL, DEFAULT 0	
created_at	TIMESTAMP	DEFAULT, CURRENT_TIMESTAMP	

BLOCKS

Column	Data type (size)	Constraint	Constraint Name
block_id	int	Primary Key, AUTO_INCREEMENT	
block_hash	int	NOT NULL,UNIQUE	
previous_block_id	int	DEFAULT NULL, REFERENCES BLOCKS	
timestamp	TIMESTAMP	DEFAULT, CURRENT_TIMESTAMP	
nonce	int	NOT NULL	

TRANSACTIONS

Column	Data type (size)	Constraint	Constraint Name
transaction tid	int	Primary Key, AUTO_INCREEMENT	
block_id	int	NOT NULL, REFERENCES Blocks ON DELETE SET NULL	
amount	decimal(20,8)	CHECK (amount > 0), NOT NULL	
sender_wallet_id	int	NOT NULL, REFERENCES Wallets ON DELETE CASCADE	
receiver_wallet_id	int	NOT NULL, REFERENCES Wallets ON DELETE CASCADE	
timestamp	TIMESTAMP	DEFAULT, CURRENT_TIMESTAMP	
transaction hash	varchar(64)	UNIQUE, NOT NULL	

4.4 Relational Model Implementation

CREATE TABLE Users (user_id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100) NOT NULL, email VARCHAR(255) UNIQUE NOT NULL, password VARCHAR(255) NOT NULL, created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP);

CREATE TABLE Wallets (wallet_id INT AUTO_INCREMENT PRIMARY KEY, user_id INT NOT NULL, balance DECIMAL(20,8) DEFAULT 0 NOT NULL, created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (user_id) REFERENCES Users(user_id) ON DELETE CASCADE);

CREATE TABLE Blocks (block_id INT AUTO_INCREMENT PRIMARY KEY, block_hash VARCHAR(64) UNIQUE NOT NULL, previous_block_id INT DEFAULT NULL, timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP, nonce INT NOT NULL, FOREIGN KEY (previous_block_id) REFERENCES Blocks(block_id) ON DELETE CASCADE);

CREATE TABLE Transactions (transaction_id INT AUTO_INCREMENT PRIMARY KEY, block_id INT DEFAULT NULL, amount DECIMAL(20,8) CHECK (amount > 0) NOT NULL, sender_wallet_id INT NOT NULL, receiver_wallet_id INT NOT NULL, timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP, transaction_hash VARCHAR(64) UNIQUE NOT NULL, FOREIGN KEY (block_id) REFERENCES Blocks(block_id) ON DELETE SET NULL, FOREIGN KEY (sender_wallet_id) REFERENCES Wallets(wallet_id) ON DELETE CASCADE, FOREIGN KEY (receiver_wallet_id) REFERENCES Wallets(wallet_id) ON DELETE CASCADE);

Chapter 6

6. Result

Homepage

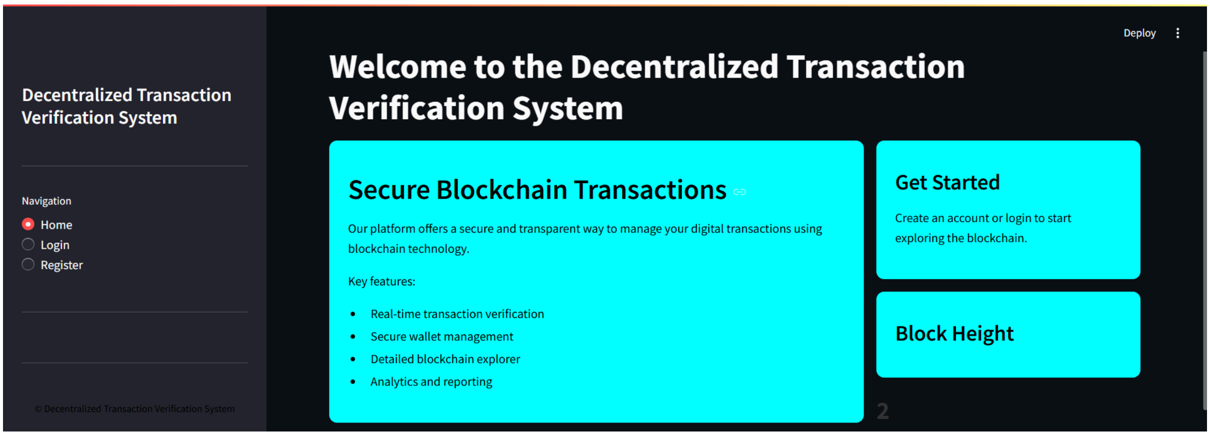

Figure 6.1 Homepage

Recent Transactions

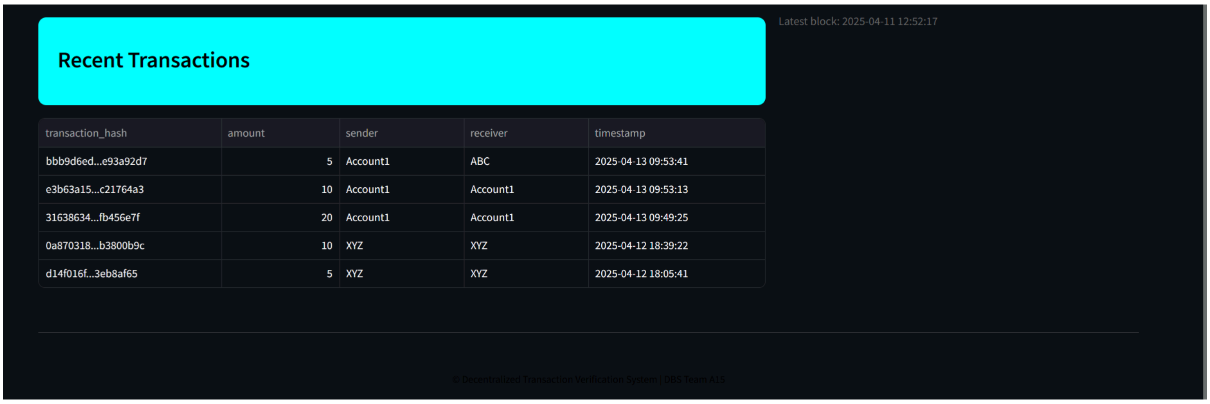

Figure 6.2 Recent Transactions

New Account Creation

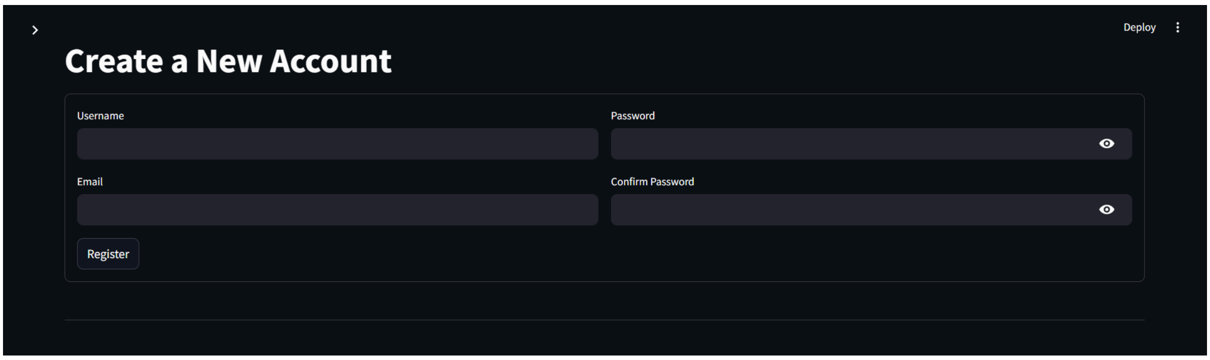

Figure 6.3 New Account

User Dashboard

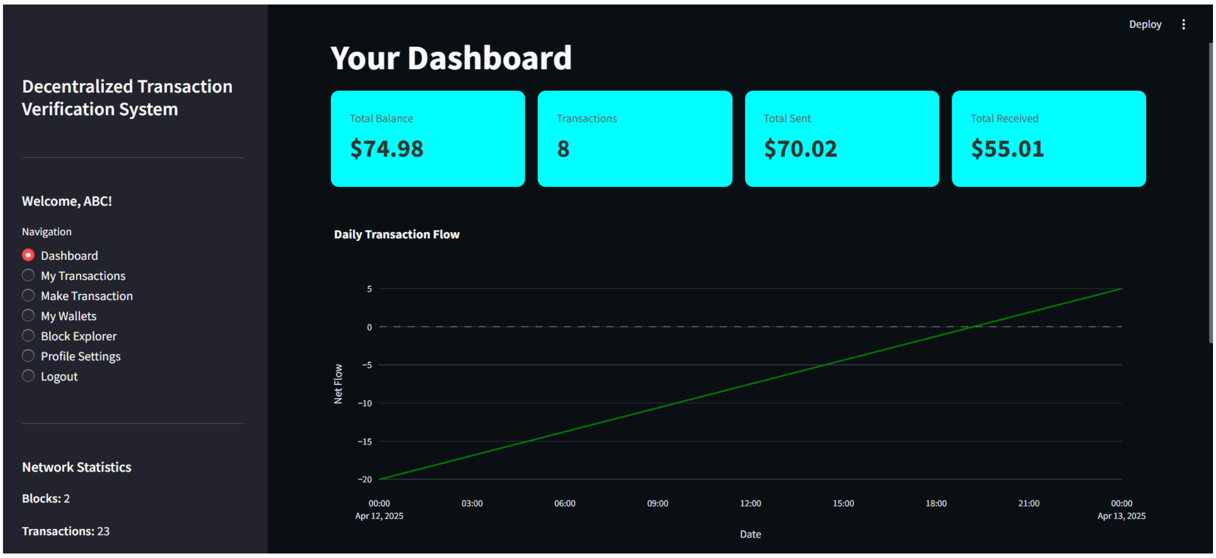

Figure 6.4 Dashboard

User Transaction List

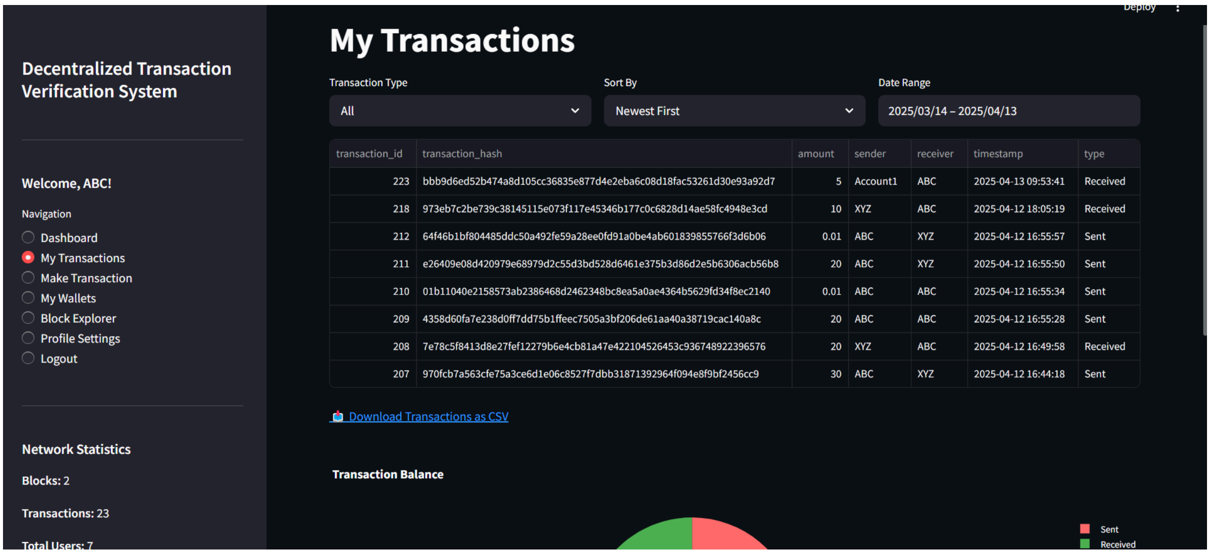

Figure 6.5 Transactions

User Send/Receive ratio represented graphically

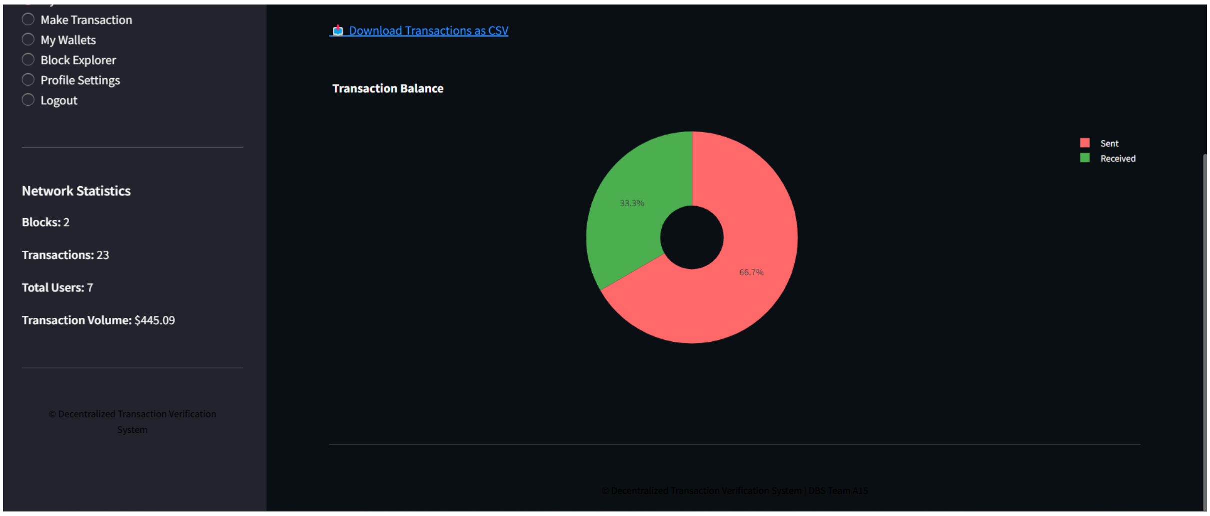

Figure 6.6 Send/Recieve Ratio

User wallets summary

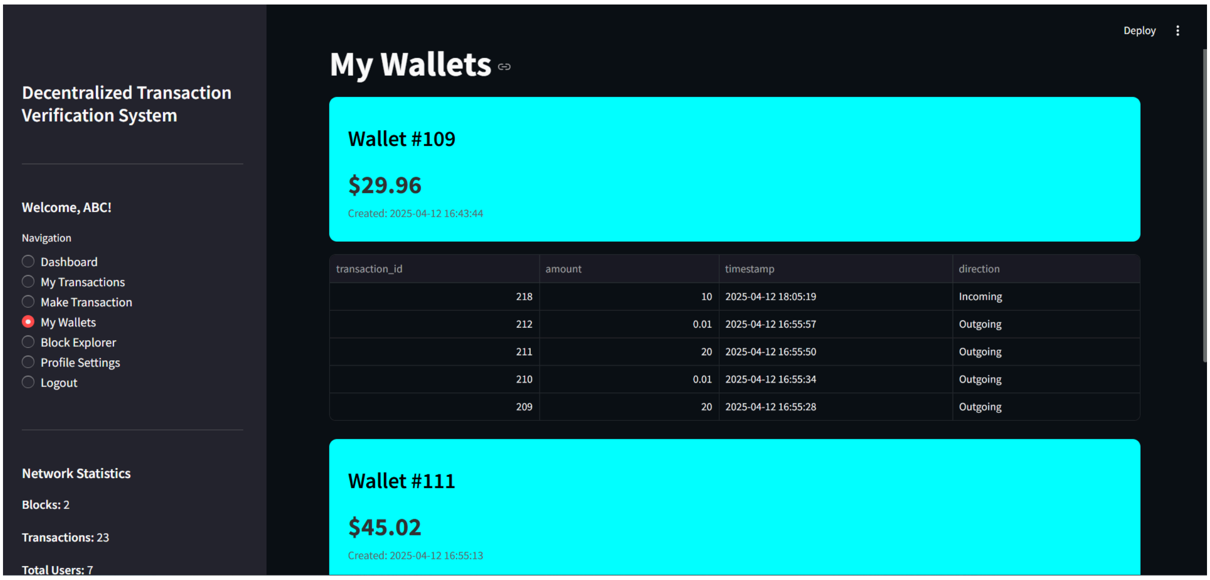

Figure 6.7 Wallet Summary

Block Information

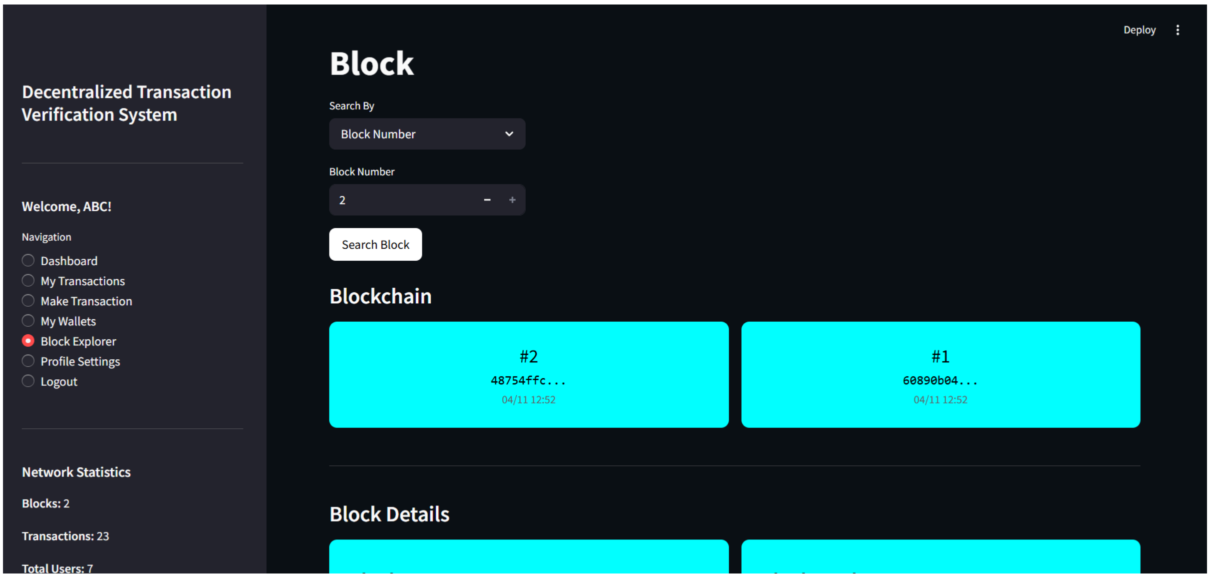

Figure 6.8 Block

Block Transactions and security verification

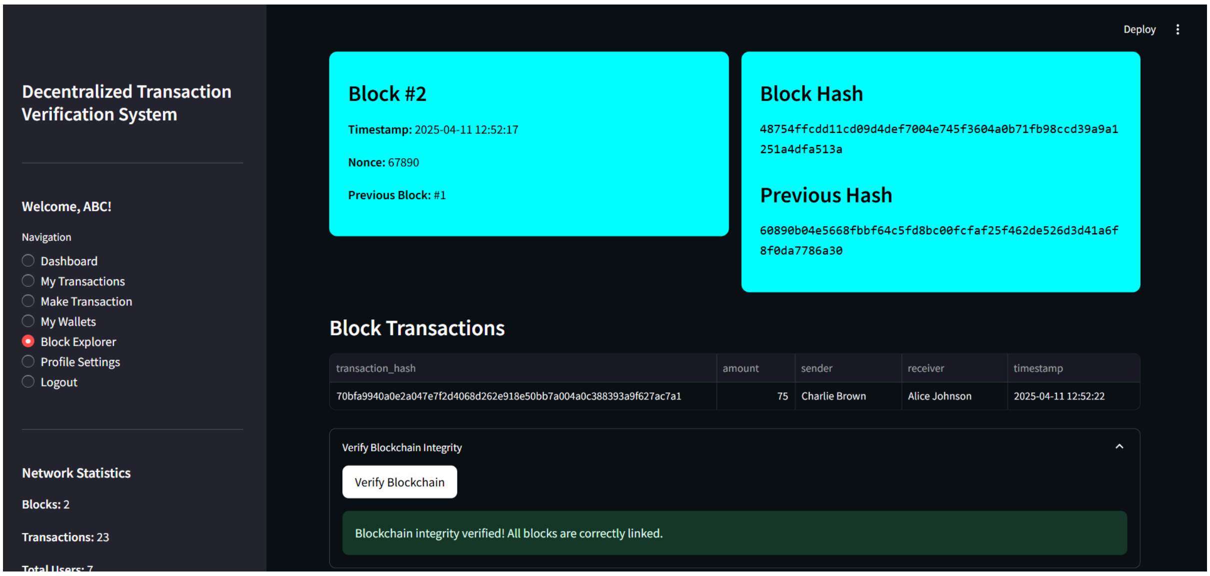

Figure 6.9 Block Transactions

Profile Settings and option to reset password

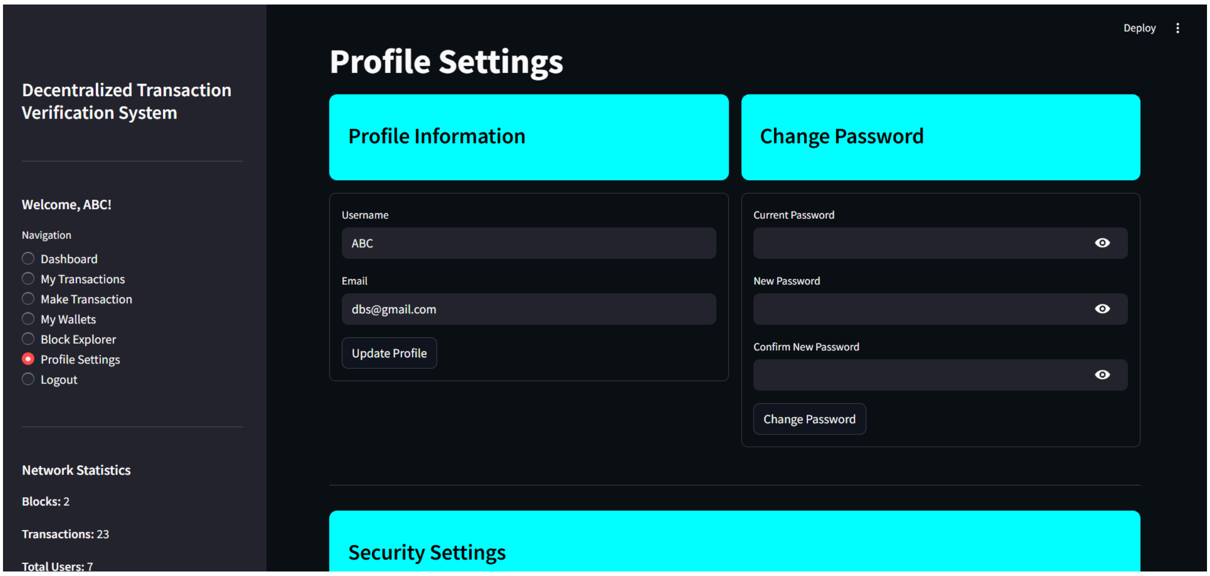

Figure 6.10 Profile Settings

Transaction page (sending to same user's different wallet)

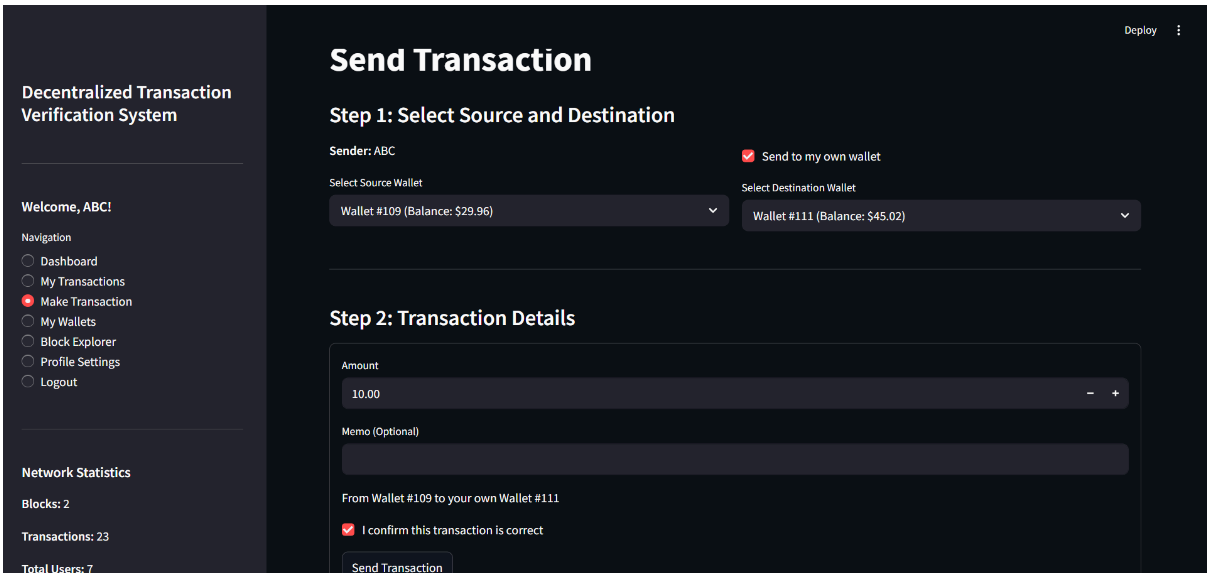

Figure 6.11 Transaction page (sending to same user's different wallet)

Transaction page (sending to different user)

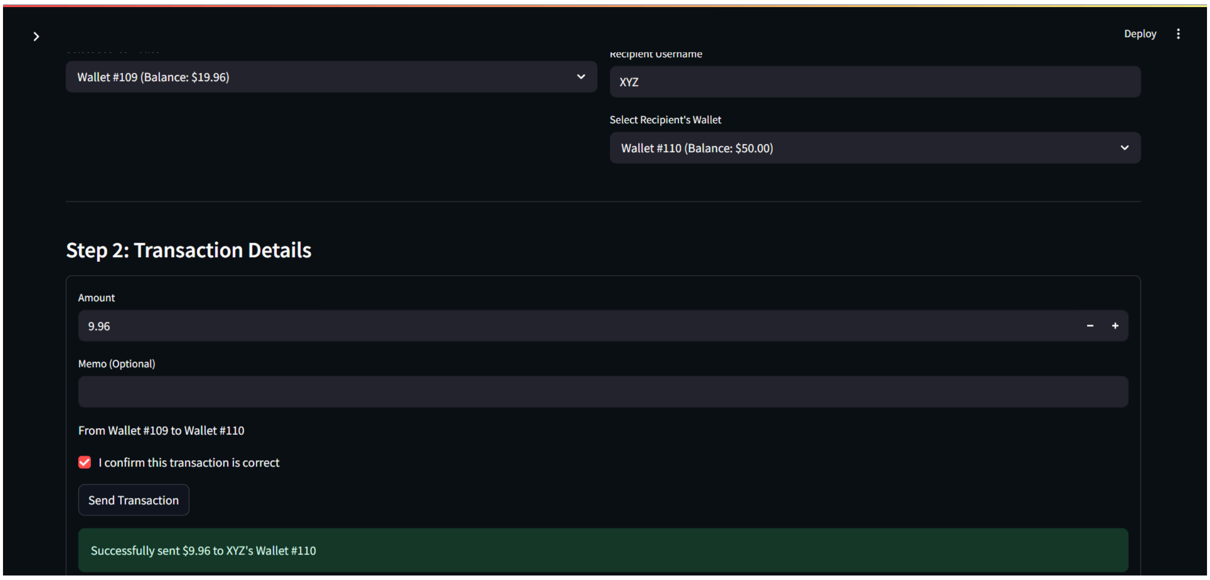

Figure 6.12 Transaction page (sending to different user)

Creating Account with Existing Email

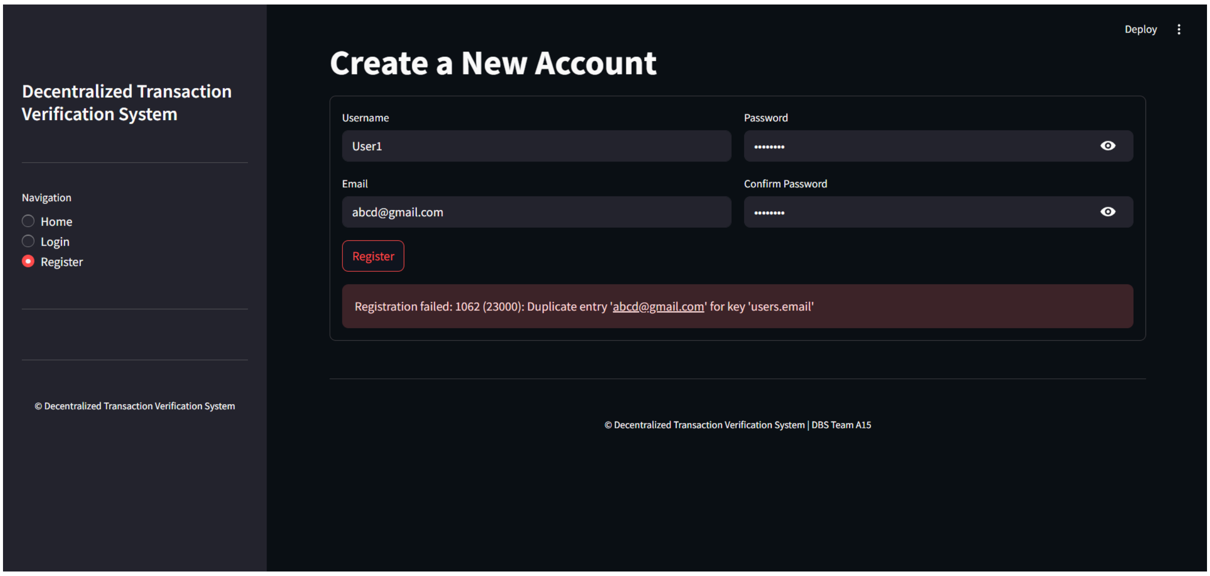

Figure 6.13 Creating Account with Existing Email

mysql> select * from transactions;

transaction_id	block_id	amount	sender_wallet_id	receiver_wallet_id	timestamp	transaction_hash	fee	memo	status
201	1	50.00000000	101	103	2025-04-11 12:52:22	5235dde3806c7f929ad562925c85905afaa495c2c3de6ccff50a340ee2af6b1c	0.00000000	NULL	pending
202	1	100.00000000	103	104	2025-04-11 12:52:22	f8bda198c44c47ae6ee074ed38aa1044c5b72231748a4637a03989ee26bed23f	0.00000000	NULL	pending
203	2	75.00000000	104	102	2025-04-11 12:52:22	70bfa9940a0e2a047e7f2d4068d262e918e50bb7a004a0c388393a9f627ac7a1	0.00000000	NULL	pending
204	NULL	0.01000000	103	101	2025-04-11 13:20:06	b85a53c667c1141024928bf9fbfe7df248c5975bf5c9abb748df89c313a451e8	0.00000000	NULL	pending
205	NULL	0.04000000	101	103	2025-04-12 16:14:20	7be88487ae65439a64745b7f9665d26271149b99509070f9d2324f3588512420	0.00000000	NULL	pending
206	NULL	20.00000000	108	101	2025-04-12 16:42:48	6527117f4d905444dab09e6ccac023553fc7f809fc57ba821b25d45866c8a5f4	0.00000000	NULL	pending
207	NULL	30.00000000	109	108	2025-04-12 16:44:18	970fcb7a563cfe75a3ce6d1e06c8527f7dbb3187139296 4f094e8f9bf2456cc9	0.00000000	NULL	pending
208	NULL	20.00000000	108	109	2025-04-12 16:49:58	7e78c5f8413d8e27fef12279b6e4cb81a47e422104526453c936748922396576	0.00000000	NULL	pending
209	NULL	20.00000000	109	111	2025-04-12 16:55:28	4358d60fa7e238d0ff7dd75b1ffeec7505a3bf206de61aa40a38719cac140a8c	0.00000000		pending
210	NULL	0.01000000	109	111	2025-04-12 16:55:34	01b11040e2158573ab2386468d2462348bc8ea5a0ae4364b5629fd34f8ec2140	0.00000000		pending
211	NULL	20.00000000	109	108	2025-04-12 16:55:50	e26409e08d420979e68979d2c55d3bd528d6461e375b3d86d2e5b6306acb56b8	0.00000000		pending
212	NULL	0.01000000	109	108	2025-04-12 16:55:57	64f46b1bf804485ddc50a492fe59a28ee0fd91a0be4ab601839855766f3d6b06	0.00000000		pending
213	NULL	20.00000000	108	110	2025-04-12 16:58:04	313c72f6da81874f3d81da80c6435647944ed5ab42e6c77fb5ba413798aafd1c	0.00000000		pending
214	NULL	0.01000000	108	110	2025-04-12 16:58:10	e01d3716a0098e4e94cdcfbf9d4d4ed83af09162c790e2c5b5dd78449a06a7b9	0.00000000		pending
215	NULL	0.01000000	110	101	2025-04-12 16:58:51	32f8c158f61d09a6c222af0c1cefea2b2d0ee54391bcb56a845f29cdc4d64692	0.00000000		pending
216	NULL	20.00000000	110	108	2025-04-12 17:54:56	11fe97274071e67c9f58fd48a5b675f7d8fff9994379a8d16b3e55e44fb28a1d	0.00000000		pending
217	NULL	10.00000000	108	110	2025-04-12 17:59:36	486b589a9cb0817f3210c94e6d8cc6d23757bade077a81386			

Figure 6.14 Transactions Data Table

Trigger action when Amount is more than Balance

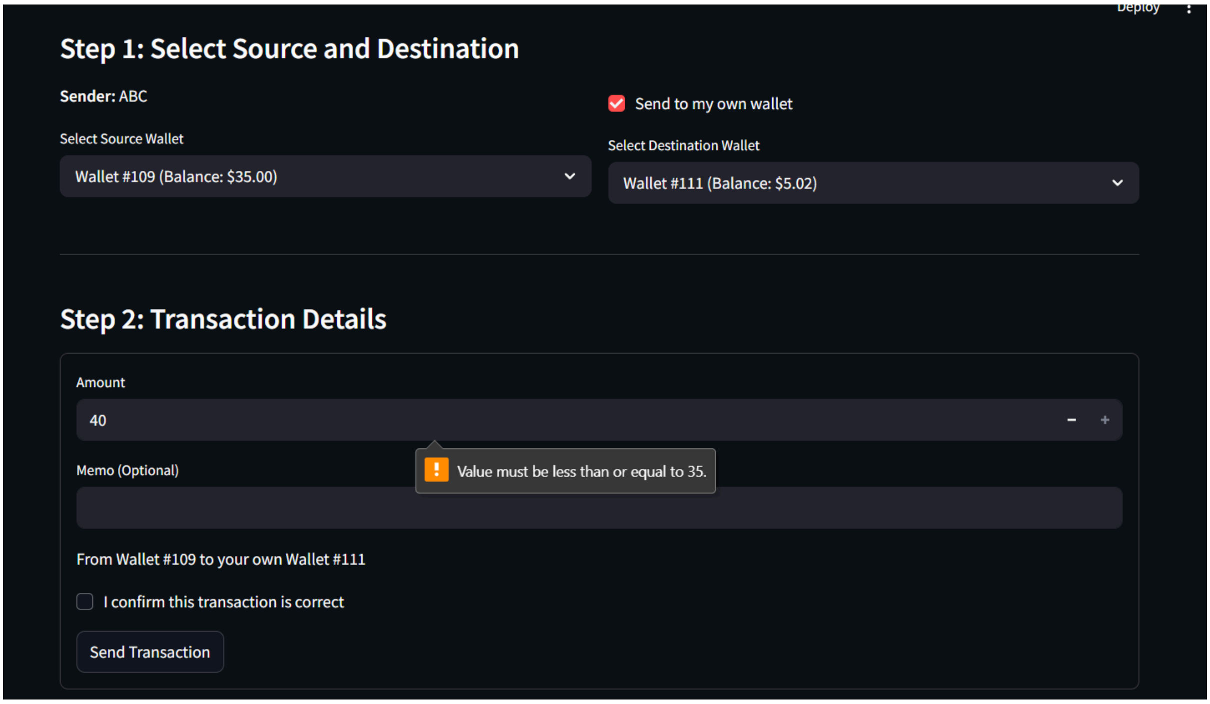

Figure 6.15 Trigger action when Amount is more than Balance

Chapter 7

7. Conclusion and Future Work

7.1 Conclusion

This project successfully demonstrates the core principles of blockchain technology using MySQL. By simulating key features such as block mining, transaction handling with fees and memos, reward distribution, and integrity enforcement through triggers and stored procedures, it provides a foundational understanding of how decentralized ledger systems operate. The use of SQL-based logic not only makes the concept accessible for database enthusiasts but also highlights the flexibility of relational databases in mimicking real-world blockchain behavior. This simulation lays the groundwork for more advanced implementations, serving as a valuable educational and experimental tool in the growing field of blockchain systems.

7.2 Scope for Future Work

7.2.1 Smart Contract Simulation: Extend the system to support smart contracts by allowing users to define programmable conditions for automated transactions.

7.2.2 Consensus Mechanism Integration: Implement consensus algorithms like Proof-of-Work (PoW), Proof-of-Stake (PoS), or Delegated Proof-of-Stake (DPoS) to make block validation more realistic and decentralized.

7.2.3 Wallet and Encryption System: Incorporate digital wallet management and basic public-private key encryption for secure identity and transaction handling.

7.2.4 Transaction Explorer UI: Develop a web-based dashboard to visually track blocks, transactions, miner rewards, and user balances for better user interaction.

7.2.5 Scalability Testing: Conduct performance analysis and optimize SQL procedures and triggers to handle larger volumes of transactions and users efficiently.

7.2.6 Fraud Detection and Anomaly Monitoring: Integrate triggers or stored procedures for identifying suspicious transaction patterns or attempts to exploit the system.

7.2.7 Integration with External APIs: Allow the system to simulate real-world usage by integrating with APIs for currency conversion, external data triggers, or other financial services.

7.2.8 Backup and Recovery Mechanisms: Add procedures to ensure secure backup of blockchain data and implement a recovery system in case of data corruption.

7.2.9 User Authentication and Access Control: Add login functionality and role-based access to differentiate between miners, regular users, and administrators. Also scope to add two-factor authentication.