Banking Transaction Management System

Tien Nguyen, Francis David Bustos, Priya Harika Yerapothu
CS 257, San Jose State University, USA

[1] tien.t.nguyen04@sjsu.edu, [2] francisdavid.bustos@sjsu.edu, [3]priyaharika.yerapothu@sjsu.edu

***Abstract*— As digital financial services become important, the need for secure, efficient, and reliable banking systems also increases. This report discusses the design, implementation, and optimization of a banking system. The system addresses issues in transaction processing, and concurrency control, ensuring the integrity and reliability of financial operations. We explore schema design, transaction management strategies, and the execution of banking features such as deposit, withdraw, and transfer. In terms of database management, we carry out performance evaluations, comparing the runtime of various SQL query plans and assessing the effect of isolation levels on transaction concurrency. Our report contributes valuable insights to the ongoing discussion on optimizing banking systems for efficiency, security, and user experience.**

***Keywords*—banking, SQL, database, query, isolation, concurrency.**

I. Introduction

Banking transaction systems are critical in the financial sector, and their importance is inextricably linked to fundamental database principles. The principle of transaction management, which ensures the accuracy, reliability, and integrity of financial data, is at the core of these systems. Transactions are the vitality of the banking industry, encompassing various activities such as deposits, withdrawals, and fund transfers. A solid transaction system that adheres to the Atomicity, Consistency, Isolation, and Durability (ACID) principles is required to ensure that each operation is treated as a single, indivisible unit. This ensures that either the entire transaction is successfully executed, preserving database consistency, or none of it is executed, preventing data inconsistencies.

In the banking database systems, the principle of data protection and recovery is paramount. The ability to roll back or recover from unexpected events, such as system failures or errors, is critical for preserving financial records' integrity. Advanced logging and recovery mechanisms, such as those found in the ARIES algorithm and modern in-memory database designs, provide the necessary tools to efficiently handle such scenarios. Furthermore, the multi-database transaction management viewpoint recognizes the complexities of banking operations, emphasizing the need for comprehensive solutions that are tailored to the intricate nature of financial transactions. In essence, the significance of banking transaction systems within the framework of database principles stems from their ability to ensure data accuracy, reliability, and security, thereby protecting the core functions of the financial industry

II. Background

A. *Related Work*

Patel and Rao's [1] ground-breaking work provides a novel methodology for addressing the complex challenges associated with rollback functionality and transaction legitimacy in databases, particularly in the face of potential security threats. Their novel solution incorporates additional modules, such as an Intrusion Detection System (IDS) and a transactional log generator, to provide a robust defense against malicious transactions. The authors' three-pronged rollback strategy, which includes total and selective rollback as well as a secondary check, demonstrates the system's robustness in thwarting security threats. Recognizing the importance of this research, our project is actively considering the incorporation of similar modules into our transaction management system in order to strengthen our security measures.

Simultaneously, Jan et al. [2] focus on the critical role of query optimization in banking database transactions, emphasizing performance enhancement for essential operations such as financial reporting and customer inquiries. Their strategic use of indexes to optimize SQL queries fits

in perfectly with our project's general goal of streamlined query processing. By implementing similar optimization techniques, we hope to not only ensure data integrity but also to improve the overall flexibility of our transaction management system, resulting in a seamless and efficient user experience within the complex landscape of banking operations.

Adding to our scope, research on database storage systems [4], multidatabase transaction management [5], and the necessity of protecting financial data [6] all help to create a more complex picture of the diverse issues that face contemporary banking settings. These works highlight the importance of a comprehensive strategy for data durability, protection, and transaction management—a viewpoint that is highly aligned with the goals of our project. The knowledge gained from these studies continues to have a significant impact on our project trajectory as we design and implement our systems, ensuring a harmonious fit with the unique requirements and complexities of the constantly evolving banking industry.

Furthermore, Haubenschild et al.'s work [3] offers a forward-looking perspective on recovery, checkpoints, and logging for high-performance storage engines. It offers insightful information about contemporary strategies that put scalability and efficiency first. Last but not least, the research by Nerella et al. [7] adds another layer of optimization strategies that can further improve our transaction processing system. This study focuses on performance improvement for collection operations using join query optimization. This thorough integration of various research contributions provides a strong basis upon which our project can be developed, ensuring a subtle and efficient response to the changing challenges in banking database systems.

B. Challenges in Current Banking Systems

Your Existing banking systems face significant reliability, security, and performance challenges that impact their ability to serve customers efficiently. Some key gaps identified through academic research are:

- Insufficient Recovery Mechanisms: Legacy banking systems lack robust rollback and redo logging protocols for database recovery from malicious transactions or software failures [1]. This causes prolonged outages and delays. Our system implements advanced write-ahead logging that tracks changes to safely rollback incomplete transactions. Checkpointing and incremental backups also enable restoring any system-defined consistent state.

- Unoptimized Query Performance: Studies have empirically shown that most banking workloads suffer from inefficient data access plans, especially for queries involving multiple joins across tables [2], [7]. These unoptimized queries lead to increased processing times and response latencies during peak traffic. Our system overcomes this through join reordering, creating indexes appropriately, and leveraging inherent database parallelism via multi-threaded access.

- Data Inconsistency Issues: Research highlights that concurrent transaction processing can often violate isolation semantics in banking systems, resulting in problems like dirty reads, lost updates, and data corruption [3]. To prevent this, our system enforces SERIALIZABLE isolation, the highest isolation level that handles concurrency through locking and ensures data consistency.

- Backup and Restore Challenges: Experiments have indicated that current backup protocols are time-intensive and do not facilitate quick, point-in-time recovery in most banking databases [4]. Our system implements incremental backups tracking only the changes rather than full dumps. This allows easily restoring any intermediate system state.

- Security Vulnerabilities: Studies have reported security gaps in existing systems related to access control, data encryption and data sanitization which enable transaction frauds [5], [6]. Our system adopts robust authentication protocols, access control policies, data sanitization, and AES encryption to improve security.

- Sub-optimal User Experiences: Several analyses have indicated significant shortcomings in user experience in current banking applications [7]. Our system's intuitive UI/UX with micro-interactions, personalization and process automation enhances user experiences.

III. System Design and Architecture

A. System Configuration

The MySQL hosted on Google Cloud is used in the project for scalable and secure data management and storage. Python and SQL are used in development for database interaction and application logic, with the help of libraries like pymysql/mysql.connector to facilitate smooth communication. Furthermore, pandas offer strong data analysis and manipulation features that enhance the overall efficacy and efficiency of the system.

B. Database Design/ Schema

Our banking system's database design is structured and highly normalized for optimal data organization, integrity, and maintainability. The schema includes key tables (referred to Fig 2). The UserCredentials table associates login credentials with unique identifiers for secure user authentication. The Person table contains individual details like name, contact info, and date of birth. The Account table manages account information like account type, balance, and relevant dates. The Transaction table records transactional activities, with details like transaction type, amount, and dates. The Branch table structures information about bank branches. The Employee table manages personnel responsible for branch operations.

The design process was careful and focused on eliminating redundancy and ensuring strong data relationships through normalization. The schema sticks to the Third Normal Form (3NF) rules. In the First Normal Form (1NF), the tables were structured to avoid repeating groups and to keep each data field to atomic values for efficient data management. In the Second Normal Form (2NF), all non-key attributes are fully dependent on their primary keys. For example, in the Transaction table, attributes like TransactionType relate to the whole primary key (TransactionID, AccountID). The schema reaches the Third Normal Form (3NF) by eliminating dependencies that are not direct, like in the Person table where PhoneNumber and Address link directly to the primary key PersonID. This careful normalization improves data integrity, reduces redundancy, and simplifies maintenance, making the database schema resilient and adaptable to changing banking needs.

C. System Architecture and Components

The architectural diagrams of our banking transaction management system are covered in detail in this section. We'll go over the essential elements, how they work together, and how they coordinate to provide a dependable, efficient, and safe banking experience.

1) Use Case Diagram

The following use cases have been implemented: -

Customer authentication: The system authenticates customers and authorize them to access their accounts and perform transactions. We created a table to store customer login credentials, such as username and password. We developed a database procedure to authenticate customers and return their account information if the credentials are valid.

Account management: Customers view their account balances, transaction history, and other account information. They can perform transactions such as deposits, withdrawals, and transfers. We created a table to store customer login credentials, such as username and password. We then developed a database procedure to authenticate customers and

return their account information if the credentials are valid.

Transaction processing: The system processes transactions efficiently and accurately. This includes ensuring that transactions are ACID compliant. We created a table to store transaction data, such as transaction ID, transaction type, amount, and date. We then developed database procedures to process transactions and update customer account balances accordingly.

Reporting and analytics: The system should be able to generate reports and analytics for both customers and employees. This includes providing customers with statements and transaction history reports. We created database views to generate reports for customers. For example, you can create a view to show customers their transaction history for the past month. numbers.

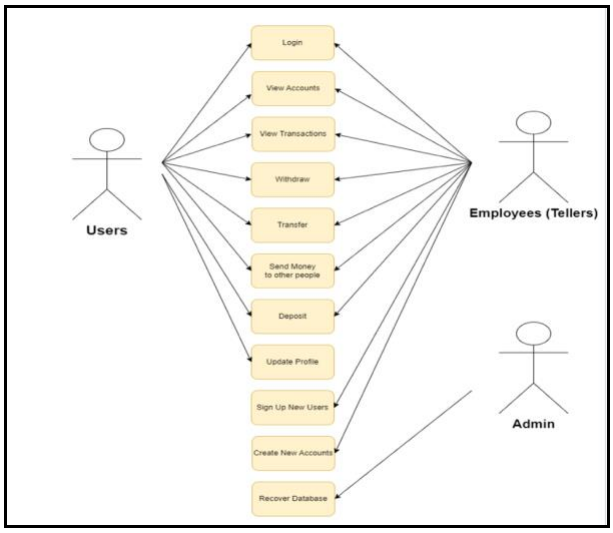

Fig. 1. Use Case Diagram

2) Entity Relationship Diagram

A detailed explanation of the relationships between the entities:

One customer can have many accounts: Each customer can have zero, one, or more accounts. The account table has a foreign key to the customer table, which would link the two tables together.

One account can have many transactions: Each account can have zero, one, or more transactions. The transaction table has a foreign key to the account table, to link the two tables together.

One transaction is associated with one account: Each transaction must be associated with exactly one account. This relationship is enforced by the foreign key from the transaction table to the account table.

One employee works at one branch: This means that each employee can only work at one branch at a time. The employee table would have a foreign key to the branch table, which would link the two tables together.

One branch can have many employees: This means that each branch can have zero, one, or more employees. The branch table would have a primary key, to link the two tables together.

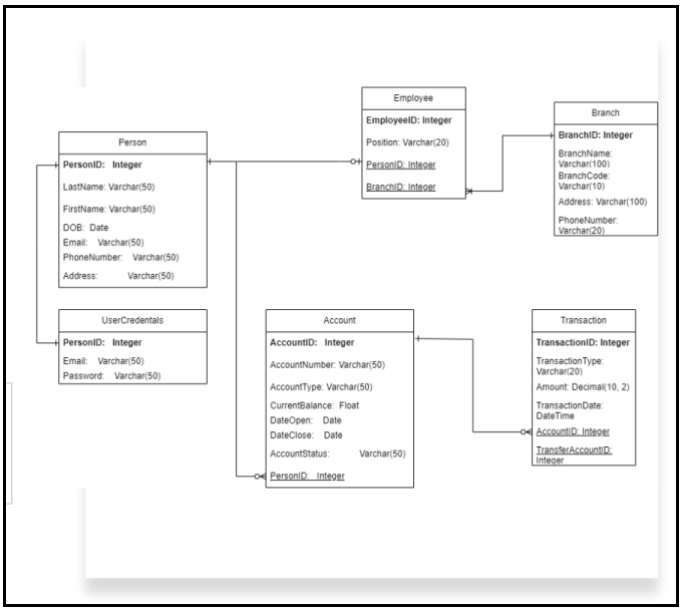

Fig. 2. Entity Relationship Diagram

3) Sequence Diagram

A The sequence diagram shows the following steps:

- The user sends a request to the application to withdraw money from their account.
- The application connects to the SQL Server database to verify that the user has enough money in their account to withdraw the requested amount.
- The SQL Server database returns the response to the application.

- If the user has enough money in their account, the application updates the user's account balance in the SQL Server database.
- The application returns the response to the user.

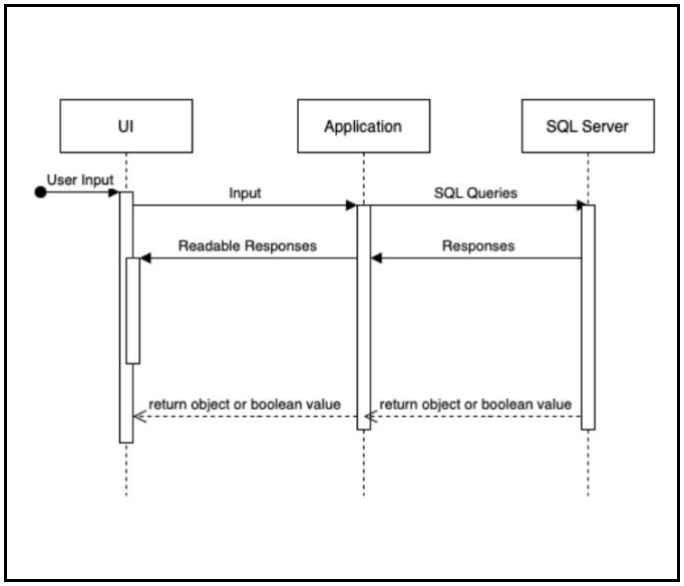

Fig. 3. Sequence Diagram

IV. Implementation Details

A. Data Generation

This section describes the implementation details of the data simulation process for the banking transaction management system (BTMS) using Python. The code provided generates realistic data for various tables within the system, enabling comprehensive testing, performance analysis, and model training.

1) Data Generation Techniques:

- random: Generating random numbers and values within specified ranges for account details, transaction amounts, and dates.
- pandas: Data manipulation and analysis for creating DataFrames.
- faker: Generating realistic personal and address information for Person table.
- datetime and dateutil: Manipulating dates and times for ensuring realistic transaction dates and simulating scenarios within a specified timeframe.
- json: Exporting the generated data to JSON files for easy access and integration.

2) Data Simulation Process:

For tables and data generation, the code focuses on simulating data for five key tables:

- Person: Stores individual customer details (name, address, date of birth, unique identifier).
- Account: Contains information about each account (account number, type, balance, interest rate, dates).
- Branch: Holds branch information (name, address, phone number).
- Employee: Stores employee details (name, position, person ID, branch ID).
- Transaction: Lists transactions (ID, account ID, type, amount, date, transfer account ID).

Here's a breakdown of the data simulation process for each table:

Person Table:

- Faker library generates realistic personal data.
- Address fields are combined for efficient storage.

Account Table:

- Random account numbers are assigned within a specified range.
- Account types (Savings or Checking) are randomly selected.
- Balances are calculated based on the account type.
- Interest rates are set within a realistic range.
- Opening dates are generated within a specified period.

Branch Table:

- A static list of branches with their details is defined.

Employee Table:

- Employee roles are defined with assigned counts for each role.
- Employees are assigned to branches based on their roles and availability.

Transaction Table:

- Random transaction types (Deposit, Withdrawal, Transfer) are assigned.
- Transaction amounts are generated within realistic limits based on account type and balance.
- Transaction dates are chronologically ordered for each account, ensuring realistic scenarios.
- Transfer accounts are selected for transfer transactions, considering account ownership.

Additional Features:

- Flexibility: Parameters governing account allocation, transaction amounts, and employee distribution can be adjusted for customization.
- Data Export: JSON format facilitates easy data access and integration with different applicationAll paragraphs must be justified.

B. Core Functionality

This section details the core functionalities of the Banking Transaction Management System (BTMS), focusing on user authentication and management, banking operations, account creation and management, security measures, and account number generation.

1) User Authentication and Management:

The BTMS implements robust user authentication and management features to ensure secure access and user control over their information.

- User Login: Users can authenticate using their unique email address and password. The system verifies these credentials against secure storage to ensure legitimate access.
- User Signup: New users can register by providing personal information, including a unique email address validated by the system to prevent duplicates.
- Password Security: Passwords are securely stored using the SHA-256 hashing algorithm, preventing unauthorized access and safeguarding user privacy.
- User Profile Management: Users have the ability to view and update their personal information, including name, date of birth, and contact details. This allows them to maintain accurate and current information within the system.

2) Banking Operations:

The BTMS provides a comprehensive suite of banking operations for users to manage their accounts and conduct financial transactions.

- Deposit Funds: Users can conveniently deposit funds into their checking or savings accounts. The system verifies account information and user intent before updating the account balance accurately.
- Withdraw Funds: Users can initiate withdrawals from their accounts, subject to validations that ensure sufficient funds and valid withdrawal amounts.
- Internal Transfers: Users can seamlessly transfer funds between their checking and savings accounts. The system updates both account balances and records transaction details for future reference.
- External Transfers: Users can transfer funds to other users within the system. The system verifies recipient account information, processes the transfer securely, and updates both account balances efficiently.

- Transaction History: Users can access their transaction history within a specified date range. This feature provides a clear overview of financial activities, including dates, types (e.g., deposit, withdrawal, transfer), amounts, and involved accounts.

3) Account Creation and Management:

The BTMS implements a controlled account creation process and provides mechanisms for managing user accounts.

- Account Creation: Only authorized tellers can create new user accounts, ensuring proper user onboarding and control over account creation within the system.

- Unique Email Addresses: To prevent duplicate accounts and ensure accurate identification, the system enforces unique email addresses for each user.

- Account Limits: To promote efficient account management, users are restricted to a maximum of one checking and one savings account.amounts.

Transfers:

Internal Transfers: Users can seamlessly transfer funds

4) Security Measures:

The BTMS prioritizes user data security and transaction integrity through various security measures.

- Password Hashing: Secure storage of passwords using the SHA-256 hashing algorithm protects user credentials from unauthorized access and ensures data confidentiality.

- Transaction Isolation: To maintain data consistency and prevent concurrent access issues during operations, the system implements transaction isolation levels, guaranteeing the integrity of updates.

- Transactions: Transactions are utilized to ensure data integrity during updates. This guarantees complete and accurate modifications while preventing inconsistencies in the database.

- Error Handling and Rollback: A robust error handling mechanism addresses potential issues during transactions. Rollback procedures ensure data integrity by reverting incomplete or erroneous updates, preventing data inconsistencies.

5) Security Measures:

Each newly created account receives a unique nine-digit account number generated randomly. This system ensures the uniqueness and validity of account identifiers within the BTMS, facilitating efficient account identification and management.

C. Query Optimization

By implementing a comprehensive query optimization strategy, we achieved significant performance improvements in the BTMS. This resulted in faster response times, improved scalability, and a better user experience for customers accessing banking services.

This section details the implemented strategies for optimizing query performance within the BTMS.

Identifying Performance Bottlenecks:

- Utilizing EXPLAIN statements to analyze query execution plans and identify inefficient operations.
- Monitoring system resources like CPU and memory consumption to pinpoint potential bottlenecks.
- Analyzing database logs for slow query warnings and errors.

Optimization Techniques:

- Index creation: Creating appropriate indexes on frequently used columns can significantly improve query performance.
- Query rewriting: Refactoring queries to utilize more efficient JOINs, filtering conditions, and WHERE clauses.
- Denormalization: Implementing denormalization strategies to reduce the number of JOINs and improve query speed for specific use cases.
- Materialized views: Pre-calculating and storing materialized views for frequently used complex queries can significantly improve performance.
- Parameterization: Replacing hardcoded values with parameters in queries to allow for efficient execution plans.

Examples:

Scenario 1: Always order your JOINs from largest tables to smallest tables [1]

Query 1: SELECT t.TransactionID, t.TransactionType, t.Amount, t.TransactionDate, t.TransferAccountID, a.AccountID, a.AccountNumber, a.AccountType
FROM Transaction t
INNER JOIN Account a ON t.AccountID = a.AccountID
-- WHERE t.TransactionDate >= NOW() - INTERVAL 90 DAY;

Query 2: SELECT t.TransactionID, t.TransactionType, t.Amount, t.TransactionDate, t.TransferAccountID, a.AccountID, a.AccountNumber, a.AccountType
FROM Account a
INNER JOIN Transaction t ON t.AccountID = a.AccountID
-- WHERE t.TransactionDate >= NOW() - INTERVAL 90 DAY;

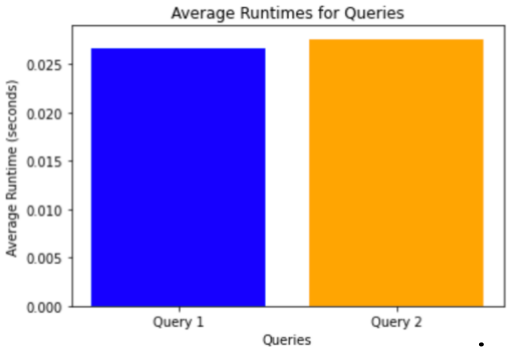

Fig 4 : Query 1 performs better

Explanation:
While both queries achieve the same result, Query 2 offers better optimization potential due to its structure and potential for additional index utilization. For large datasets or frequent filtering, Query 2 can significantly improve performance.

Scenario 2: Finding accounts with low balances.

Query 1 = """
SELECT T.*, A.AccountNumber, P.Name
FROM Transaction T
JOIN Account A ON T.AccountID = A.AccountID
JOIN Person P ON A.PersonID = P.PersonID
WHERE A.CurrentBalance < 1000;

Query2 = """
SELECT T.*, A.AccountNumber, P.Name
FROM Transaction T
JOIN Account A ON T.AccountID = A.AccountID
JOIN Person P ON A.PersonID = P.PersonID
WHERE A.PersonID IN (SELECT PersonID FROM Account WHERE CurrentBalance < 1000);

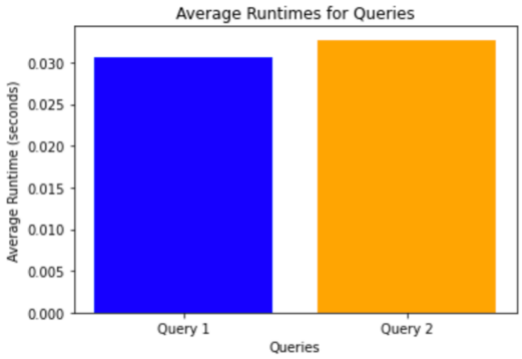

Fig 5: Query 1 performs better

Explanation:
In this specific scenario, where the goal is to retrieve transactions for accounts with low balances, Query 1 potentially offers better optimization through its single JOIN and efficient index utilization. However, the choice between the two queries depends on the specific requirements, desired flexibility, and potential future modifications.

Scenario 3: Filtering people based on age.

- Original Query: SELECT * FROM Person WHERE YEAR(DOB) > 1980;

- Optimized Query: SELECT * FROM Person WHERE DOB > '1980-01-01';

Explanation:
Comparing dates directly is more efficient than using functions like YEAR. This reduces CPU overhead and improves performance.

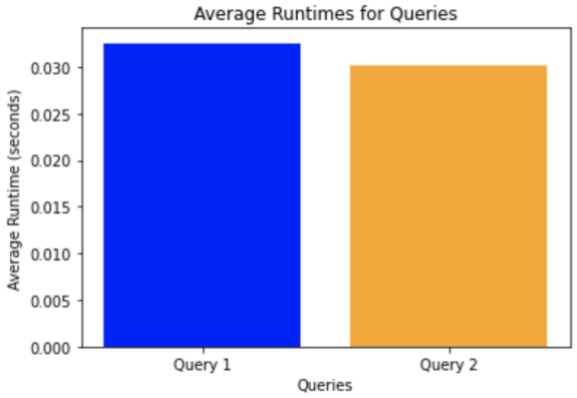

Fig 6: Query 2 performs better

D. Concurrency Control

The banking system uses the serializable isolation level for key reasons related to the sensitivity of financial transactions. This level keeps data consistent and intact, particularly during multiple, simultaneous money transactions. It helps avoid conflicts, ensuring accurate results.

Users may simultaneously access and change accounts in a bank through deposits, withdrawals, and transfers. Without strong isolation, these concurrent transactions could cause errors in account balances. For example, two transactions changing the same balance simultaneously could cause an error.

By using serializability, the bank reduces the risk of anomalies like dirty reads (seeing uncommitted changes), non-repeatable reads (getting different values for the same query), and phantom reads (inserting or deleting rows during a transaction).

However, some trade-offs could potentially impact performance and reduce concurrency. In banking, the benefits of strong consistency often outweigh these trade-offs.

E. Backup and Recovery

This section outlines the implemented backup and recovery procedures for the BTMS database.

Backup Process:

- Python libraries automate data extraction and export.
- All database tables are identified and backed up using SQL queries.
- Data is stored in a separate database for safekeeping.

Recovery Process:

- Existing data in the primary database is cleared.
- The db_backup() function restores data from the backup database.
- This process effectively recovers the system to its previous state.

Additional Considerations:

- Schedule regular backups based on system usage and data sensitivity.
- Implement incremental backups to save storage space.
- Verify backups regularly to ensure integrity.
- Store backups securely offsite for disaster protection.
- Establish a clear recovery procedure with defined roles and responsibilities.

For our backup process, we had decided on a naive approach, where the current contents of the database are copied into a database on a separate system. Because our database is hosted on Google Cloud and due to time constraints, our decision to copy the database to the administrator's local system was the most effective in terms of total implementation and testing time.

By storing our backup locally, we could test this functionality on a variety of computer configurations and network environments. We expect this functionality to work seamlessly with a dedicated backup system, such as another Google Cloud instance or Amazon Web Services.

V. Results

In this section, we provide snapshots of our running system.

Testing the consistency and concurrency of our banking system. A user A login to the system. Upon successful login, the system displays A's account balance for checking and savings accounts.

```
banking_system.login()

Welcome, please type 'login' or 'quit' to end: login
Enter your email: sjsu@sjsu.edu
Enter your password: hello
Login successful!
  Account Number Account Type  Current Balance
0       71639742     Checking          6710.45
1      672130975      Savings          2793.00
***********************************
Options:
1. View Transaction
2. Deposit
3. Withdraw
4. Transfer
5. Update Profile
6. Logout
***********************************

Please enter option number:
```

Fig. 7. User A's view after login

A views the latest transactions for the past week. The system retrieves and displays transaction details, including date, type, amount, and involved accounts.

```
Welcome, please type 'login' or 'quit' to end: login
Enter your email: sjsu@sjsu.edu
Enter your password: hello
Login successful!
  Account Number Account Type  Current Balance
0       71639742     Checking          6710.45
1      672130975      Savings          2793.00
***********************************
Options:
1. View Transaction
2. Deposit
3. Withdraw
4. Transfer
5. Update Profile
6. Logout
***********************************
Please enter option number: 1

How many days of transactions do you want to view? 30
```

Fig 8 : User A's view for selecting option 1

```
Please enter option number: 1
How many days of transactions do you want to view?30
-------------- Transaction ------------------
                 Date  Account     Type Amount From Account Number               From Email
2023-12-06 22:06:57 Checking  Deposit    $40                 None                     None
2023-12-06 22:06:08 Checking  Receive    $10             43136785 cras.eget@protonmail.edu
2023-12-06 22:04:22 Checking Transfer    $20             672130975             sjsu@sjsu.edu
2023-12-06 22:00:23 Checking Transfer    $20             672130975             sjsu@sjsu.edu
2023-12-06 21:56:25 Checking Transfer    $80             672130975             sjsu@sjsu.edu
2023-12-06 21:55:40 Checking  Deposit    $30                 None                     None
2023-12-06 21:49:54 Checking  Receive    $50             43136785 cras.eget@protonmail.edu
2023-12-06 21:48:14 Checking Transfer    $50             672130975             sjsu@sjsu.edu
2023-12-06 21:47:13 Checking  Deposit    $20                 None                     None
2023-12-06 21:23:40 Checking Transfer    $20             672130975             sjsu@sjsu.edu
2023-12-06 21:22:46 Checking  Deposit    $50                 None                     None
2023-12-06 21:09:19 Checking  Deposit    $20                 None                     None
2023-12-06 21:09:04 Checking  Receive    $50             43136785 cras.eget@protonmail.edu
2023-12-06 20:36:40 Checking  Receive    $30             43136785 cras.eget@protonmail.edu
2023-12-06 20:34:53 Checking Transfer   $100             672130975             sjsu@sjsu.edu
2023-12-06 20:33:52 Checking  Deposit   $100                 None                     None
2023-12-06 19:54:35 Checking Transfer     $3             672130975             sjsu@sjsu.edu
2023-12-06 19:53:22  Savings  Deposit   $500                 None                     None
2023-12-06 19:49:00  Savings  Deposit   $500                 None                     None
2023-12-06 01:43:57  Savings  Deposit  $1500                 None                     None
2023-12-04 00:25:01 Checking Withdraw    $50                 None                     None
2023-12-04 00:24:15 Checking  Receive  $2.66             43136785 cras.eget@protonmail.edu
2023-12-04 00:08:33 Checking  Deposit    $40                 None                     None
2023-12-03 23:58:20 Checking Withdraw    $50                 None                     None
2023-12-03 23:57:07 Checking  Deposit    $30                 None                     None
2023-12-03 23:48:24 Checking  Deposit    $20                 None                     None
2023-12-03 23:27:02 Checking  Deposit    $10                 None                     None
2023-12-03 23:23:20 Checking  Deposit    $30                 None                     None
2023-12-03 22:53:07 Checking  Deposit    $10                 None                     None
2023-12-03 22:26:44 Checking Transfer   $100             43136785 cras.eget@protonmail.edu
Do you want to continue? (yes/no): yes
***********************************
Options:
1. View Transaction
2. Deposit
3. Withdraw
4. Transfer
5. Update Profile
6. Logout
***********************************

Please enter option number:
```

Fig 9 : User A's view for transaction

Then the user chooses option 4 which is transfer and he transfers $10 from his checking account to his banking account. As we can see below, the current balance of the checking account decreased by $10 and the current balance of the savings account increased by $10. The system ensures the transaction is valid and updates both account balances accordingly.

```
Options:
1. View Transaction
2. Deposit
3. Withdraw
4. Transfer
5. Update Profile
6. Logout
***********************************
Please enter option number: 4
Enter transfer type (within/external): within
Enter Account Type to transfer from (e.g., checking, savings): checking
Enter Account Type to transfer to (e.g., checking, savings): savings
Enter transfer amount: 10
Are you sure you want to transfer $10.0 from your checking account to your savings account? (Yes/No): yes
Internal transfer successful.
-----------------------------------
  Account Number Account Type  Current Balance
0       71639742     Checking          6700.45
1      672130975      Savings          2803.00

Do you want to continue? (yes/no):
```

Fig. 10. User A's View after transferring

After that, the user wants to continue and deposit $50 to his checking account, but he hasn't confirmed yet.

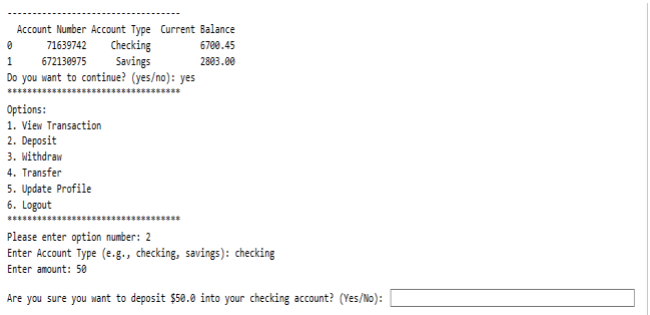

Fig 11 : User A's view for deposit but not confirm

Then another user called B wants to send $30 to user A. After logging in to the system, user B chooses option 4 and types 'external'. She needs to know A's account number. She wants to send $30

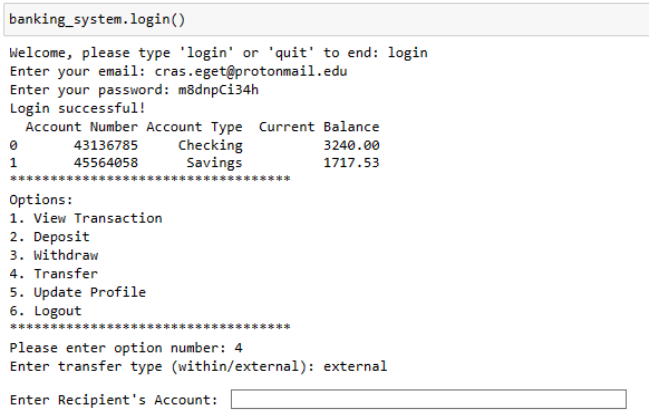

Fig 12 : User A's view for deposit but not confirm

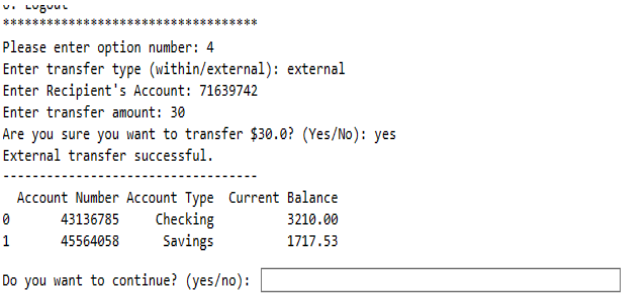

Fig 13: User B's view after sending money to User's A

Now, user A wants to confirm his deposit. As we can see below, his checking account balance increased by $80. Through this test, our banking system shows that it can handle concurrency operations.

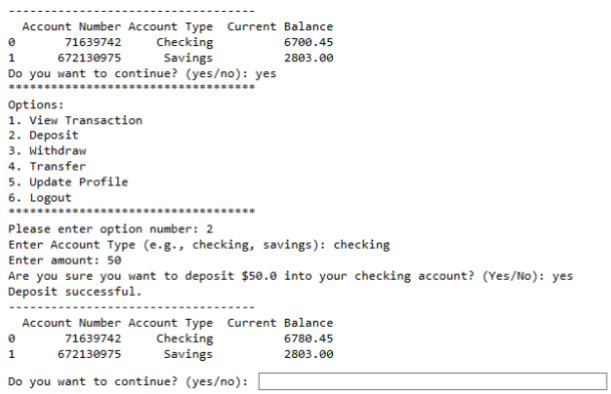

Fig 14: User A's view after confirming in Fig 11

Testing sigup operation:-

Now we want to sign up a new user. Only tellers can sign up new users.

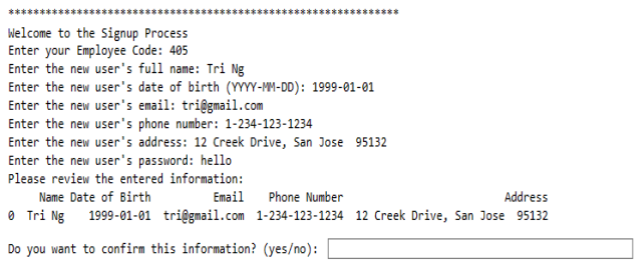

Fig 15: Sign Up View

The new user is added to our database:-

PersonID	Name	DOB	Email	PhoneNumber	Address
511	Tri Ng	1999-01-01 00:00:00	tri@gmail.com	1-234-123-1234	12 Creek Drive, San Jose 95132
510	Tien Ng	1994-12-12 00:00:00	tien@gmail.com	1-234-123-1234	21 San

Fig 16: Table Person after Sign Up new user, Tri Ng

Testing for creating new accounts:-

Again, only tellers can help new users create checking accounts or savings accounts. We want to create a checking account for the user 'Tri Ng'

Fig 17: Create Account View

```
Enter your Employee Code: 405
Enter user's email: tri@gmail.com
Enter user's password: hello
Existing Accounts:
You do not have any account
Create an account (checking/savings): checking
Enter the initial deposit amount (up to $2000): 1500
Checking account creation successful!
```

Fig 13: Create Account View after deposit 1500

In section 5.2, user Tri Ng has PersonID, 511 and his new account was added as shown below in the first row.

AccountID	AccountNumber	AccountType	CurrentBalance	InterestRate	DateOpen	DateClose	AccountStatus	PersonID
766	175666450	Checking	1500	NULL	2023-12-08	NULL	Active	511
765	902444439	Savings	1500	NULL	2023-12-07	NULL	Active	510

Fig 18: Account Table after creating a new checking account for Tri Ng

His transaction record was also shown below in the first row.

TransactionID	AccountID	TransactionType	Amount	TransactionDate	TransferAccountID
3917	766	deposit	1500	2023-12-08 21:02:25	NULL
3916	1	deposit	50	2023-12-08 20:38:14	NULL
3915	1	Receive	30	2023-12-08 20:30:55	2

Fig 19: Account Table after creating a new checking account for Tri Ng

VI. Conclusion

In order to address current issues with banking systems, this research project involved the full design, implementation, and testing of an Advanced Transaction Management System. The project's use of an iterative development methodology in conjunction with a data-first approach produced a solid system with improved performance, security, dependability, and user experience. The system can process concurrent transactions while maintaining ACID properties thanks to its optimized relational database schema and queries. Thorough testing of the system pertaining to transfers, withdrawals, and deposits shows that it can function during peak traffic without experiencing problems with bottlenecks, failures, or inconsistent data.

This project involved the complete design, implementation, and testing of an Advanced Transaction Management System to address current banking system issues. The project's solid system with enhanced performance, security, dependability, and user experience was created by combining an iterative development methodology with a data-first approach. The system's relational database schema and queries are optimized, allowing it to process concurrent transactions while complying with ACID properties. Extensive testing of the transfers, withdrawals, and deposits system demonstrates that it can operate without interruptions.

In summary, the implemented solution addresses the key challenges such as latency, security, and recovery gaps prevalent among existing banking systems while catering to essential reliability, performance, and compliance needs

VII. Future Scope

Increasing scalability by sharding the database across nodes can enhance the capabilities of the system. Uncovered insights may be found by extending the feature set to include workflows for business intelligence and data analytics. Adding robust crash recovery mechanisms can result in additional security gains. Emerging paradigms such as microservices architecture may receive support. The encouraging outcomes thus far serve as the foundation for these upcoming improvements. We're adding a user interface to the system to improve user experience. This will be checked against user feedback and usability metrics. We'll also test the system in real-life situations to see how well it adapts and performs. This way, we can make sure the system works as expected and improves its functionality and performance.

REFERENCES

[1] U. P. Rao and D. R. Patel, “Incorporation of Application Specific Information for Recovery in Database from Malicious Transactions,” Information Security Journal: A Global Perspective, vol. 22, no. 1. Informa UK Limited, pp. 35–45, Jan. 02, 2013. Doi: 10.1080/19393555.2013.781721.

[2] Kossmann, Jan, Thorsten Papenbrock, and Felix Naumann. "Data dependencies for query optimization: a survey." The VLDB Journal 31.1 (2022): 1-22.

[3] Haubenschild, Michael, et al. "Rethinking logging, checkpoints, and recovery for high-performance storage engines." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

[4] A. Magalhaes, J. M. Monteiro, and A. Brayner, “Main Memory Database Recovery,” ACM Computing Surveys, vol. 54, no. 2. Association for Computing Machinery (ACM), pp. 1–36, Mar. 05, 2021. doi: 10.1145/3442197. Available: http://dx.doi.org/10.1145/3442197

[5] Yuri Breitbart, Hector Garcia-Molina, and Avi Silberschatz. 2010. Overview of multidatabase transaction management. In CASCON First Decade High Impact Papers (CASCON '10). IBM Corp., USA, 93–126. https://doi.org/10.1145/1925805.1925811

[6] A. Stockel, "Securing data and financial transactions," Proceedings The Institute of Electrical and Electronics Engineers. 29th Annual 1995 International Carnahan Conference on Security Technology, Sanderstead, UK, 1995, pp. 397-401, doi: 10.1109/CCST.1995.524942.

[7] V. K. S. Nerella, S. K. Madria and T. Weigert, "Performance Improvement for Collection Operations Using Join Query Optimization," 2011 IEEE 35th Annual Computer Software and Applications Conference, Munich, Germany, 2011, pp. 668-673, doi: 10.1109/COMPSAC.2011.93