THE CATHOLIC UNIVERSITY OF EASTERN AFRICA (C.U.E.A)

A.M..E.C.E.A

CMT 302: ADVANCED DATABASE SYSTEMS

BANKING TRANSACTION SYSTEM

GROUP 4

GROUP MEMBERS:

1. JESSICA WANJIRU - 1049486
2. NATASHA MWANGI - 1049508
3. EDWIN MUIRU - 1049042
4. SHARON KITAVI - 1049511
5. MARK NJOROGE – 1049378

SUBMISSION DATE: 19/11/2024

CHAPTER ONE

INTRODUCTION

1.1 Project Overview

In today's digitized world, banking transaction systems play a pivotal role in ensuring safe, efficient and reliable management of customer transactions. This project focuses on designing and developing a Banking Transaction System that aims at managing customer accounts and transactions. It is designed to meet requirements of modern banking systems including scalability and ease of use.

The proposed system integrates advanced database concepts such as normalization, indexing, stored procedures, CRUD operations and transaction management to ensure data consistency, reliability and good performance. The system will allow users to make account inquiries, transfers and withdrawals from their accounts as well as their bank cards, prioritizing user experience and fast transaction processing.

1.2 Rationale

The motivation for this project arises from the need for efficient, safe and reliable banking transaction systems. Traditional systems have the following challenges:

a. Data Integrity and Consistency: Ensuring millions of records are accurate remains a challenge.

b. User Experience: Inefficient systems usually result in delays and a poor user experience for customers.

c. Scalability: Many existing traditional systems fail to handle the surge in users and transactions at peak hours.

The system proposed in this project aims to address these challenges, helping financial institutions by offering a better alternative.

1.3 Objectives

The primary goal of this project is to design and implement a robust banking transaction system that leverages advanced database features. The following objectives guided the development process:

(i) Efficiency: Optimize transaction processing by implementing indexing and query optimization techniques to reduce response times.

(ii) Data Integrity: Employ ACID properties of database systems to ensure the accuracy and reliability of all transactions.

(iii) Scalability: Design a database schema and architecture that supports scalability to accommodate growing user demands.

(iv) User Experience: Provide a user-friendly interface for managing accounts and processing transactions with minimal latency.

1.4 Review of Existing Systems

1.4.1 "Design and Implementation of a Secure Online Banking System" by Adeyemo et al. (2020)

This research investigates the development of an online banking system which places an emphasis on user security and ease of transaction. Authors applied into practice such methods of encryption as RSA and AES for protection of valuable information in the course of transmission. It further used a relational database for the maintenance of client's transactions database with the support of indexing methods to speed up performance.

The research established that embedding security measures like stored procedures and triggers into database management minimized the attempts of fraud but allowed the integrity of data to persist. The system shown was scalable for several users operating in peak traffic times, thus being appropriate for present day banking surroundings.

1.4.2 "A Scalable Banking Transaction System Using NoSQL Databases" by Zhang et al. (2021)

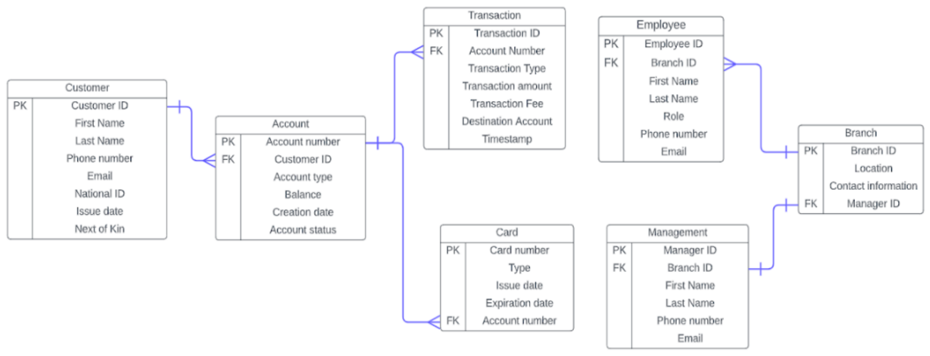
The present work analyzes the application of NoSQL databases in the implementation of banking transactional systems in view of the shortcomings of traditional RDBMS. The authors provided distributed architecture and harnessed NoSQL solutions like MongoDB to address unstructured and partially unstructured financial data.

The authors of the work concluded that NoSQL systems are more suited for applications with high transaction volume, rather than applications requiring a high degree of data integrity. The standard practices of sharding and replication allowed the system to achieve excellent fault tolerance and scalability – handling tens of millions transactions per second.

CHAPTER TWO

SYSTEM DESIGN

2.1 ER Diagram



2.2 Table Structures

2.2.1. Customer Table

Column Name	Data Type	Constraint
CustomerID	INT	PRIMARY KEY
FirstName	VARCHAR(50)	NOT NULL
LastName	VARCHAR(50)	NOT NULL
PhoneNumber	VARCHAR(15)	NOT NULL
Email	VARCHAR(100)	UNIQUE, NOT NULL
NationalID	VARCHAR(20)	NOT NULL
AccountCreationDate	DATE	NOT NULL
NextOfKin	VARCHAR(100)	NOT NULL

2.2.2 Account Table

Column Name	Data Type	Constraint
AccountNumber	INT	PRIMARY KEY
CustomerID	INT	FOREIGN KEY
AccountType	VARCHAR(20)	NOT NULL
Balance	DECIMAL(5,2)	NOT NULL
CreationDate	DATE	NOT NULL
AccountStatus	VARCHAR(20)	NOT NULL

2.2.3 Transactions Table

Column Name	Data Type	Constraint
TransactionID	INT	AUTO, PRIMARY KEY
AccountNumber	INT	FOREIGN KEY
TransactionType	VARCHAR(50)	NOT NULL
TransactionAmount	DECIMAL(15,2)	NOT NULL
TransactionFee	DECIMAL(10,2)	NOT NULL
Timestamp	TIMESTAMP	CURRENT_TIMESTAMP

2.2.4 Loan Table

Column Name	Data Type	Constraint
LoanID	INT	PRIMARY KEY
CustomerID	INT	FOREIGN KEY
LoanAmount	DECIMAL(15,2)	NOT NULL
LoanType	VARCHAR(50)	NOT NULL
InterestRate	DECIMAL(5,2)	NOT NULL
IssueDate	DATE	NOT NULL
LoanTerm	INT	NOT NULL
OutstandingBalance	DECIMAL(15,2)	NOT NULL
LoanStatus	VARCHAR(50)	

2.2.5 Card Table

Column Name	Data Type	Constraint
CardNumber	VARCHAR(16)	PRIMARY KEY
Type	VARCHAR(20)	NOT NULL
IssueDate	DATE	
ExpirationDate	DATE	
AccountNumber	INT	FOREIGN KEY

2.2.6 Branch Table

Column Name	Data Type	Constraint
BranchID	INT	PRIMARY KEY
Location	VARCHAR(100)	NOT NULL
ContactInformation	VARCHAR(100)	NOT NULL
ManagerID	INT	FOREIGN KEY

2.2.7 Employees Table

Column Name	Data Type	Constraint
EmployeeID	INT	PRIMARY KEY
BranchID	INT	FOREIGN KEY
FirstName	VARCHAR(50)	NOT NULL
LastName	VARCHAR(50)	NOT NULL
Role	VARCHAR(50)	NOT NULL
PhoneNumber	VARCHAR(15)	NOT NULL
Email	VARCHAR(100)	UNIQUE

CHAPTER THREE

IMPLEMENTATION

3.1 CRUD Operations and Advanced SQL Queries

1. Create Operations

Create operations were implemented to add:

i. New customers: Records for customers who register for the banking service.
ii. New accounts: Accounts associated with registered users
iii. New cards: Cards associated with the accounts
iv. Branch
v. Employees at the branch
vi. Loans acquired by the user

2. Read Operations

The system uses SQL SELECT queries to fetch data based on specific criteria.

For example, in read operations were implemented to:

i. Retrieve all accounts, customers, loans acquired, employees and branches

3. Update Operations

UPDATE operations were implemented using triggers and events to:

i. Update the Savings accounts balance by adding interest every 1 minute
ii. Update the account balances in case of a withdrawal or deposit
iii. Update the Outstanding loan balance with interest every 1 minute

4. Advanced SQL queries

Triggers

- **i.** To prevent negative balances: This ensures that no withdrawal transaction results in a negative account balance.
- ii. Updating Account Balances: This updates an account's balance whenever a deposit or withdrawal occurs.

Events

- i. Applying Interest to Savings Accounts: This event applies interest to active savings accounts every minute.
- ii. Updating Outstanding Loan Balances: This event calculates interest on active loans monthly.

Stored procedures

- i. Adding a New Customer: This procedure inserts a new customer record into the database.
- ii. Adding a New Transaction: This procedure records a new transaction for an account.
- iii. Updating Loan Status: This procedure updates the status of a loan based on repayment status.

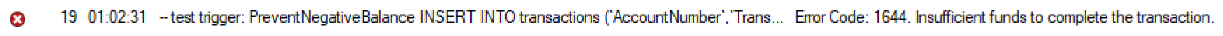
CHAPTER FOUR

TESTING AND VALIDATION

4.1 Description of Testing Results

1. Trigger testing
   a) Preventing negative balances
   Input: An attempt to withdraw 590000.00 form account 1379 with a balance of 117980.67
   Output: an error message is displayed: *"Insufficient funds to complete the transaction."*

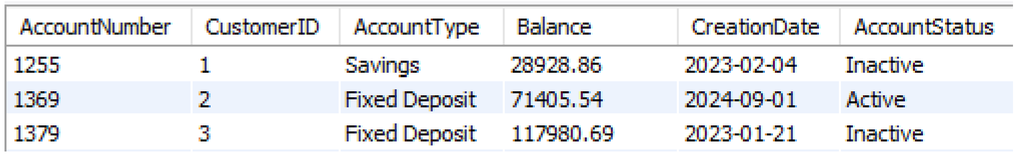
Figure 1 TRIGGER RESULT



```
19  01:02:31  --test trigger: PreventNegativeBalance INSERT INTO transactions (`AccountNumber`,`Trans...   Error Code: 1644. Insufficient funds to complete the transaction.
```

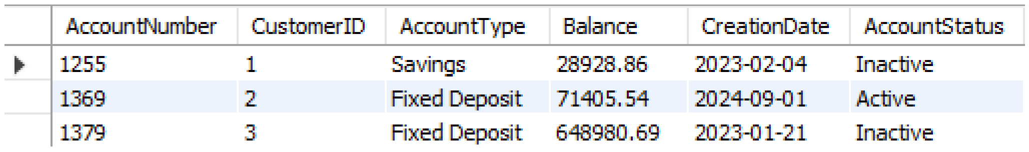
   b) Updating account balance
   Input: Deposit 590000.00 into an account, 1379, with a balance of 117980.67.

Figure 2 OLD BALANCE FOR 1379



AccountNumber	CustomerID	AccountType	Balance	CreationDate	AccountStatus
1255	1	Savings	28928.86	2023-02-04	Inactive
1369	2	Fixed Deposit	71405.54	2024-09-01	Active
1379	3	Fixed Deposit	117980.69	2023-01-21	Inactive

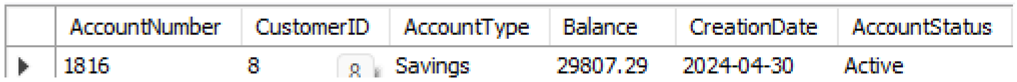
Figure 3 NEW BALANCE FOR 1379



AccountNumber	CustomerID	AccountType	Balance	CreationDate	AccountStatus
1255	1	Savings	28928.86	2023-02-04	Inactive
1369	2	Fixed Deposit	71405.54	2024-09-01	Active
1379	3	Fixed Deposit	648980.69	2023-01-21	Inactive

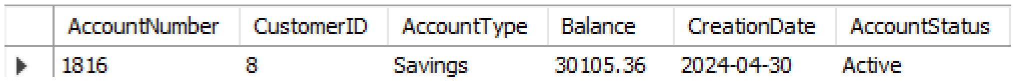
2. Event testing
   a) Applying interest to savings accounts

Figure 4 OLD BALANCE FOR ACTIVE SAVINGS ACCOUNT



AccountNumber	CustomerID	AccountType	Balance	CreationDate	AccountStatus
1816	8	Savings	29807.29	2024-04-30	Active

Figure 5 NEW BALANCE FOR ACTIVE SAVINGS ACCOUNT



AccountNumber	CustomerID	AccountType	Balance	CreationDate	AccountStatus
1816	8	Savings	30105.36	2024-04-30	Active

CONCLUSION

The Banking Transaction System designed in this project effectively illustrates the use of advanced database management systems techniques in solving problems of the traditional banking systems. By its conception, the system achieves tremendous goals in terms of efficiency, scalability, data quality, and ease of use.

Among others system features are triggers that avoid negative balances, automatic interest calculations on events, as well as databases for easier and faster customer transactions and management. Due to the amalgamation of the ACID principles together with the query optimization techniques, these transaction processes are assured of being persistent and reliable. Also, the testing and validation exercises give results that point out the fact that the system will effectively achieve the set targets.

This program provides the baseline to which improved banking systems can be built, which is a crucial asset for any bank seeking to enhance the services it provides to its clients and increase its efficiency in operations.

RECOMMENDATIONS

1. Stricter Security Policies

Consider introducing stronger security measures including multi-factor authentication and encryption of sensitive data, limiting the potential for cyber-attacks and unauthorized access.

2. Expandability of the System

See the possibility of adding NoSQL databases to support the existing relational database, improving throughput under stressful loads and working with unstructured data.

3. Mobile and Internet Application

Create simple to use mobile and web interfaces in order to improve accessibility and convenience for the customers, thereby increasing the scope and usage of the system.

4. Improvement of Results

System functionality can be further enhanced by regular performance measurements and configuration for high transaction periods.

5. Business Intelligence

Consider embedding analytics in the system that will enable the study of customer activity, dynamics of transactions, and financial parameters of its performance.

These recommendations would also further enhance the strength of the system this makes it operational effective since it will be resistant and flexible to future changes in technology and the users of the system.

REFERENCES

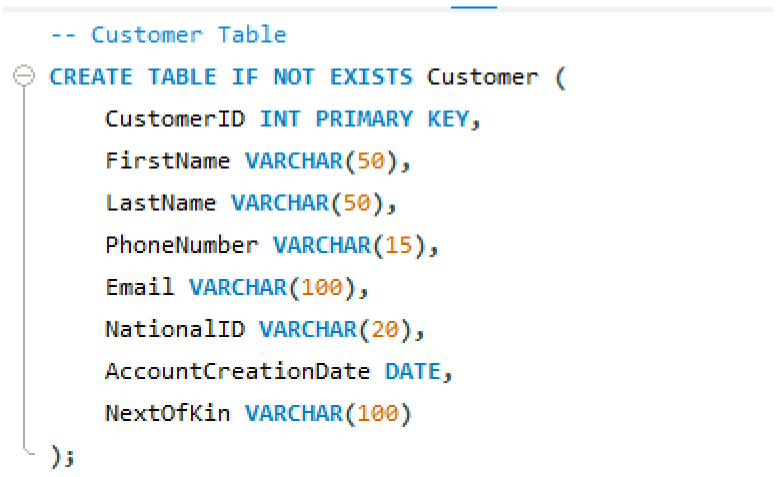
Adeyemo, K., & Olaleye, A. (2020). Design and Implementation of a Secure Online Banking System. *Journal of Financial Technology Research*, 8(4), 101-115.

Zhang, W., Chen, J., & Liu, F. (2021). A Scalable Banking Transaction System Using NoSQL Databases. *International Journal of Computer Science and Applications*, 15(2), 78-89.

APPENDICES

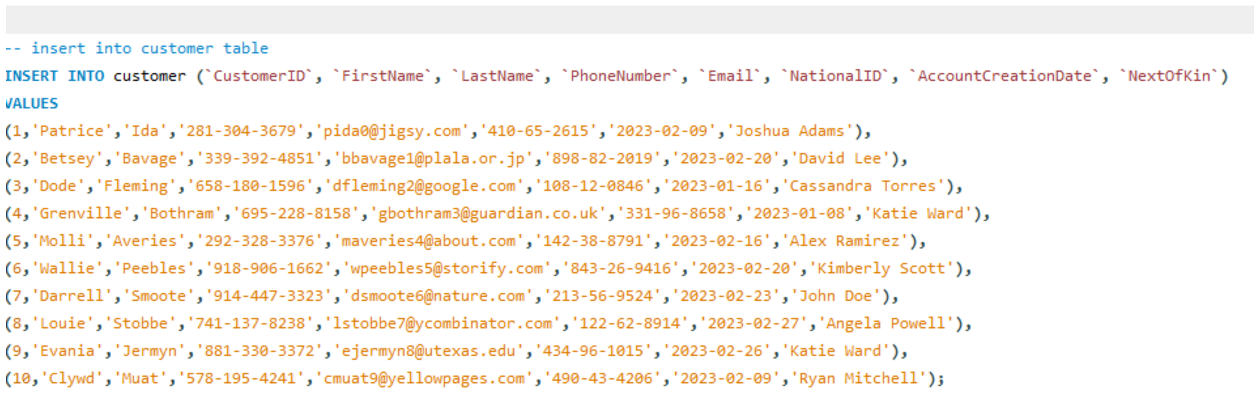
Code snippets

Figure 6 CREATE TABLES



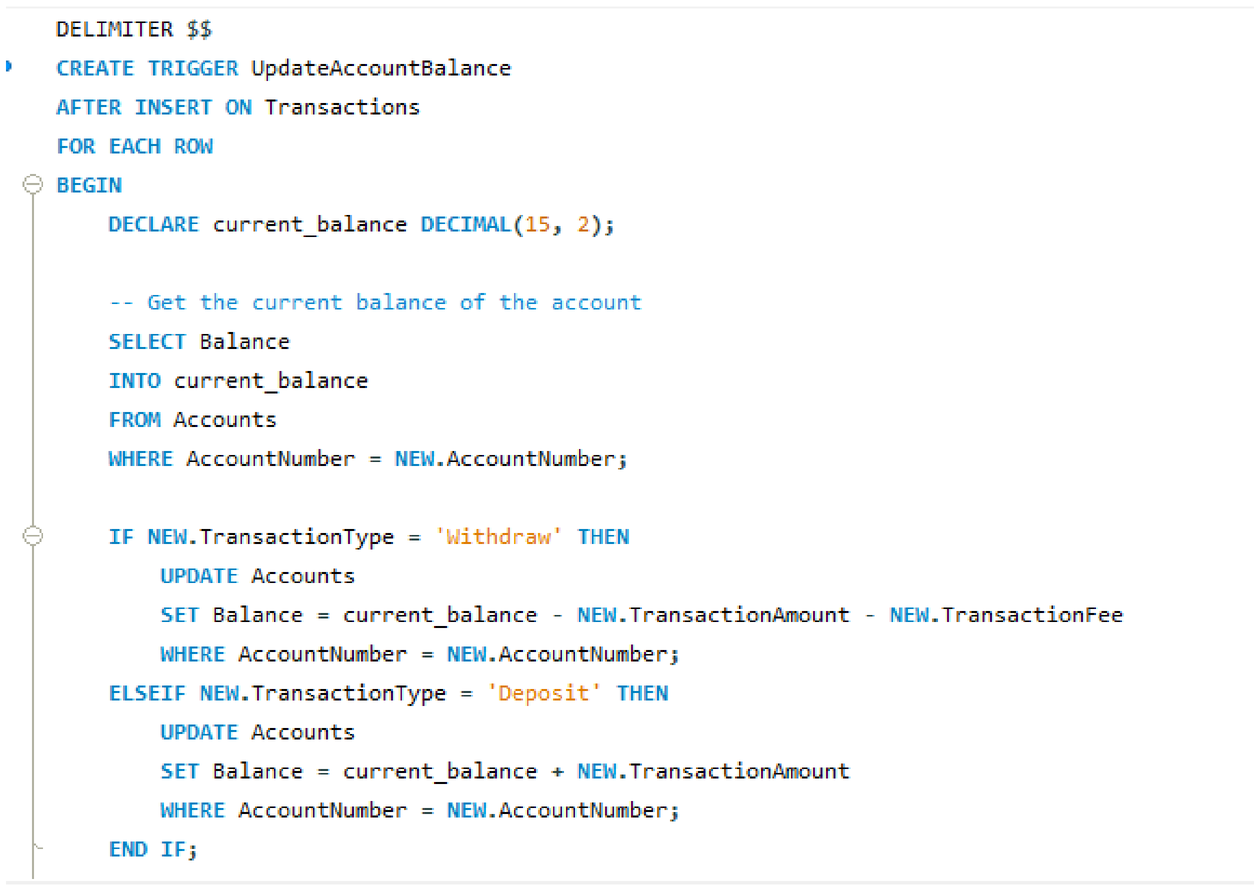
```
-- Customer Table
CREATE TABLE IF NOT EXISTS Customer (
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    PhoneNumber VARCHAR(15),
    Email VARCHAR(100),
    NationalID VARCHAR(20),
    AccountCreationDate DATE,
    NextOfKin VARCHAR(100)
);
```

Figure 7 Create operation



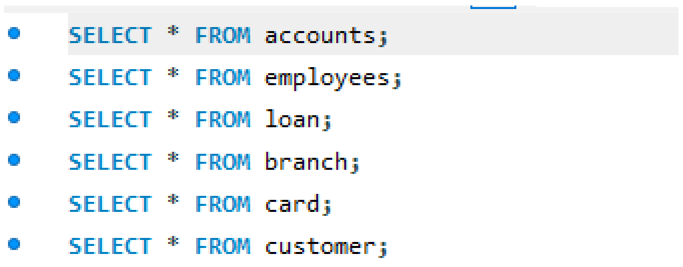
```
-- insert into customer table
INSERT INTO customer (`CustomerID`, `FirstName`, `LastName`, `PhoneNumber`, `Email`, `NationalID`, `AccountCreationDate`, `NextOfKin`)
VALUES
(1,'Patrice','Ida','281-304-3679','pida0@jigsy.com','410-65-2615','2023-02-09','Joshua Adams'),
(2,'Betsey','Bavage','339-392-4851','bbavage1@plala.or.jp','898-82-2019','2023-02-20','David Lee'),
(3,'Dode','Fleming','658-180-1596','dfleming2@google.com','108-12-0846','2023-01-16','Cassandra Torres'),
(4,'Grenville','Bothram','695-228-8158','gbothram3@guardian.co.uk','331-96-8658','2023-01-08','Katie Ward'),
(5,'Molli','Averies','292-328-3376','maveries4@about.com','142-38-8791','2023-02-16','Alex Ramirez'),
(6,'Wallie','Peebles','918-906-1662','wpeebles5@storify.com','843-26-9416','2023-02-20','Kimberly Scott'),
(7,'Darrell','Smoote','914-447-3323','dsmoote6@nature.com','213-56-9524','2023-02-23','John Doe'),
(8,'Louie','Stobbe','741-137-8238','lstobbe7@ycombinator.com','122-62-8914','2023-02-27','Angela Powell'),
(9,'Evania','Jermyn','881-330-3372','ejermyn8@utexas.edu','434-96-1015','2023-02-26','Katie Ward'),
(10,'Clywd','Muat','578-195-4241','cmuat9@yellowpages.com','490-43-4206','2023-02-09','Ryan Mitchell');
```

Figure 8 Update operations



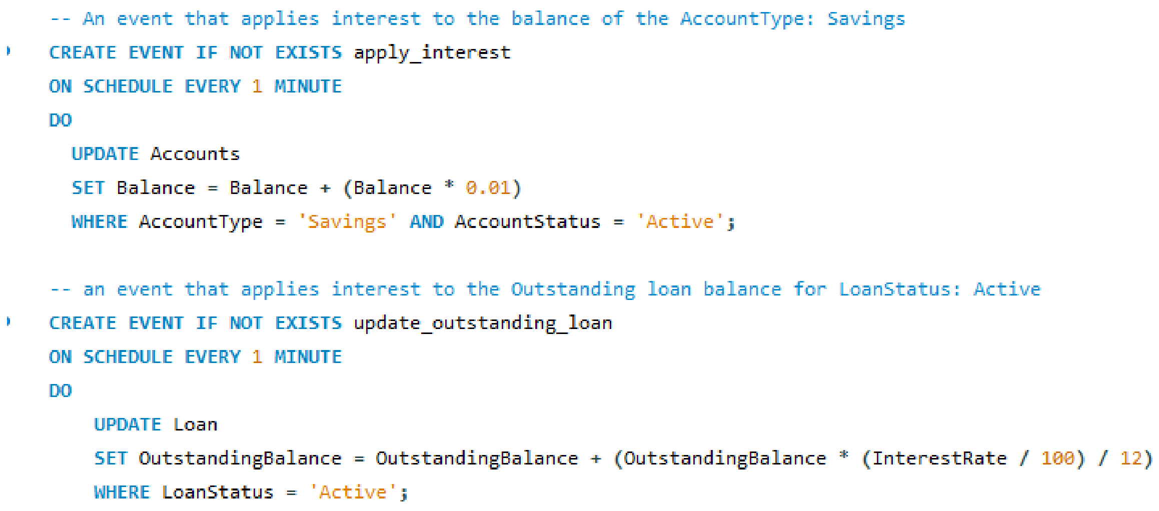
```
DELIMITER $$
CREATE TRIGGER UpdateAccountBalance
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
    DECLARE current_balance DECIMAL(15, 2);

    -- Get the current balance of the account
    SELECT Balance
    INTO current_balance
    FROM Accounts
    WHERE AccountNumber = NEW.AccountNumber;

    IF NEW.TransactionType = 'Withdraw' THEN
        UPDATE Accounts
        SET Balance = current_balance - NEW.TransactionAmount - NEW.TransactionFee
        WHERE AccountNumber = NEW.AccountNumber;
    ELSEIF NEW.TransactionType = 'Deposit' THEN
        UPDATE Accounts
        SET Balance = current_balance + NEW.TransactionAmount
        WHERE AccountNumber = NEW.AccountNumber;
    END IF;
```

Figure 9 Read operations



```
SELECT * FROM accounts;
SELECT * FROM employees;
SELECT * FROM loan;
SELECT * FROM branch;
SELECT * FROM card;
SELECT * FROM customer;
```

Figure 10 Advanced SQL queries



```
-- An event that applies interest to the balance of the AccountType: Savings
CREATE EVENT IF NOT EXISTS apply_interest
ON SCHEDULE EVERY 1 MINUTE
DO
  UPDATE Accounts
  SET Balance = Balance + (Balance * 0.01)
  WHERE AccountType = 'Savings' AND AccountStatus = 'Active';

-- an event that applies interest to the Outstanding loan balance for LoanStatus: Active
CREATE EVENT IF NOT EXISTS update_outstanding_loan
ON SCHEDULE EVERY 1 MINUTE
DO
    UPDATE Loan
    SET OutstandingBalance = OutstandingBalance + (OutstandingBalance * (InterestRate / 100) / 12)
    WHERE LoanStatus = 'Active';
```