NATIONAL INSTITUTE OF TECHNOLOGY
KARNATAKA, SURATKAL

BANK DATABASE MANGEMENT SYSTEM

IT252M
DATABASE MANAGEMENT SYSTEMS
MINOR PROJECT

Bank Database System Management

Shubham Kiran Swadi
211EE157

Akash Bantesh Patel
211MT036

Sumukh S K
211EE153

***Abstract*—This abstract introduces a Bank Database System application developed using MySQL and Tkinter. The application offers a user-friendly interface for managing banking operations, including customer accounts, transactions, and account management. It utilizes MySQL as the back-end for reliable data storage and retrieval. Tkinter facilitates the creation of an intuitive graphical user interface with features such as input validation and search capabilities. The system provides a comprehensive solution for efficient and secure banking management.**

***Index Terms*—bank, database, mysql, tkinter**

I. Introduction

The Bank Database project is a database management system (DBMS) project that simulates a banking system. The project will be implemented using MySQL and is designed to manage customer and account information for a fictional bank. The project enables users to perform various banking functions such as opening new accounts, depositing and withdrawing funds, and viewing transaction history.

II. Topic Selection

Banks form the backbone of financial and economic sectors. In this project we are trying to simulate a real-world scenario of a banking system. The project is highly customizable. We can add new features and functionalities according to the requirements of the project. The project can also be adapted to various types of users,including small businesses, banks and financial institutions. This project would help us gain practical experience in database systems and help us learn core concepts like design, normalization, indexing, querying and data manipulation. Database systems is a crucial part of banking, healthcare, logistics and finance, among others. Working on this project can help us open up various career opportunities in the future.

III. Problem Description

Banks deal with vast amounts of data on a daily basis, including customer information, account details, transaction records, and financial statements. A DBMS can help manage and store this data in an organized and efficient manner.

Data and Control Flow : Banks need to ensure the security and privacy of their customer data. A DBMS can provide robust security features, such as encryption and access control, to protect sensitive information.

IV. Objectives

The objective of this project is to design a good, efficient and a centralized Database management system for a typical bank with multiple branches in different locations. We hope to create a flawless and user friendly database management system. This Database management system is made using MySQL. The database has several salient features some of which are:-

- This database all the fields required all the day to day queries required for a typical bank
- This is also a really simple database hence it is very easy and efficient to work with.

V. Logical Design

A. Schema Diagram

A schema diagram provides a visual representation of the structure and relationships within a database schema. It illustrates the tables, columns, and their associations, helping to understand the overall database organization and design.

B. EER Diagram

An EER (Enhanced Entity-Relationship) diagram is a visual representation that depicts the relationships between entities in a database. It extends the traditional ER diagram by incorporating additional modeling concepts, allowing for a more detailed and precise depiction of data relationships and constraints.

VI. Advanced Logical Design

A. Normalization Techniques

Normalization is the process of organizing a database schema to minimize data redundancy and improve data integrity. It involves breaking down a table into smaller, more manageable tables and establishing relationships between them. In your case, you can apply normalization techniques to improve the design of the bank database. Steps that can be taken to normalize this database are-[1]

1) First Normal Form: 1NF: The Customers' table might already be in 1NF, as it likely has a primary key like 'customer_id' that uniquely identifies each customer.

[1] The mini project does not feature proper use of normalization techniques due to various constraints. These are just some normalization techniques that we would be useful in the future scope of the project.

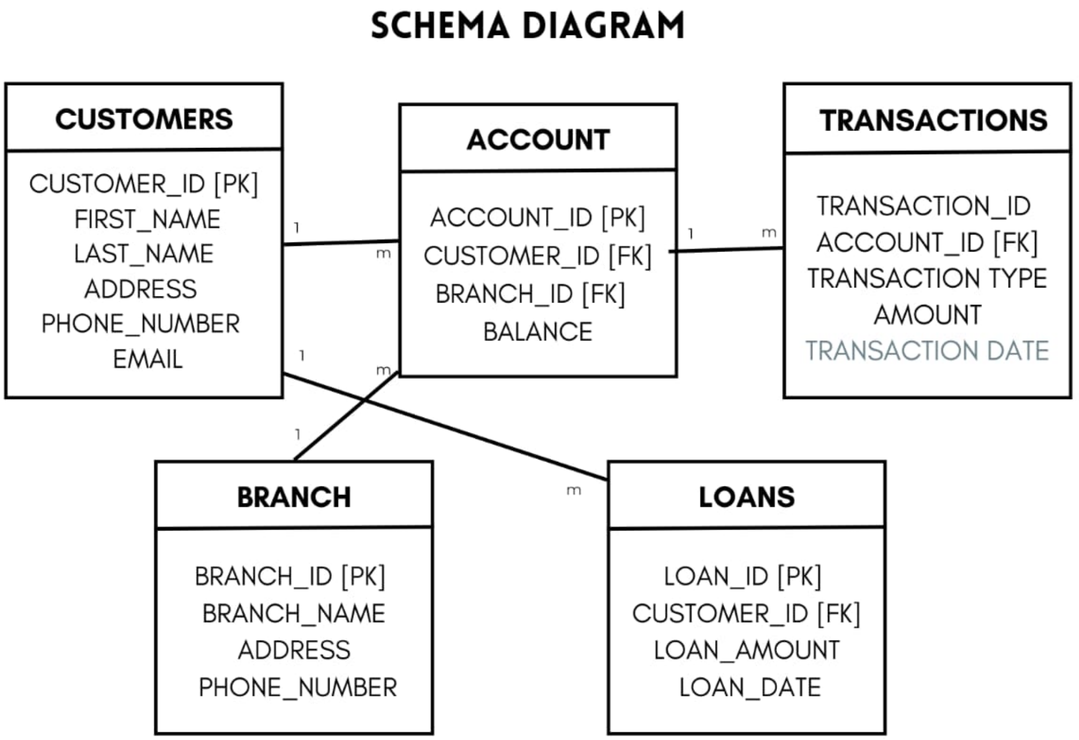


Fig. 1. Schema Diagram

2) Second Normal Form: 2NF: If the ’Account’ table has attributes like ’account_id’, ’customer_id’, and ’balance’, and the balance is dependent on the account ID only, you could move the balance attribute to a separate table called ’AccountBalances’, with columns ’account_id’ and ’balance’.

3) Third Normal Form: 3NF: If the ’Customers’ table has attributes like ’customer_id’, ’first_name’, ’last_name’, ’email’, and ’phone’, and the ’email’ attribute depends on the ’customer_id’ only, you could move the ’email’ attribute to a separate table called ’CustomerEmails’, with columns ’customer_id’ and ’email’.

VII. Implementation

A. Database Design

Here is the schema of the database used
Customers table:
customer_id (primary key)
first_name
last_name
address
phone_number
email

Account table:
account_id (primary key)
customer_id (foreign key referencing Customers table)
branch_id (foreign key referencing Branch table)
balance

Branch table:
branch_id (primary key)
branch_name
address
phone_number

Transactions table:
transaction_id (primary key)
account_id (foreign key referencing Account table)
transaction_type (e.g., ’withdraw’, ’deposit’)
amount
transaction_date

Loans table:
loan_id (primary key)
customer_id (foreign key referencing Customers table)
loan_amount
loan_date

B. Back-end Implementation

The back-end implementation of the bank database application was developed using Python with the MySQL connector. A connection was established with the MySQL database to interact with the tables and perform CRUD operations.

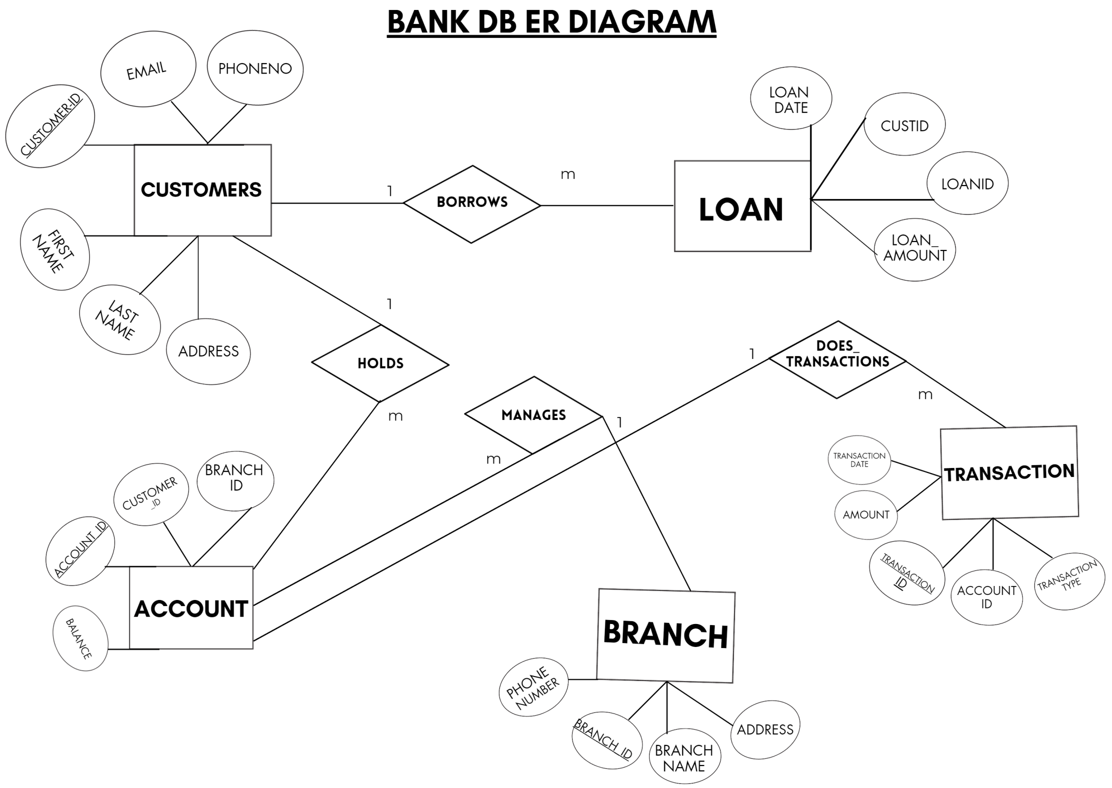
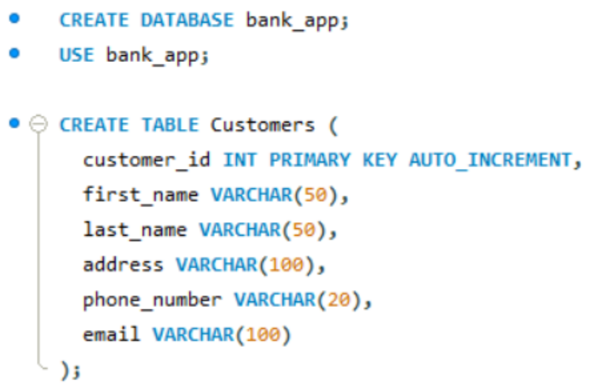


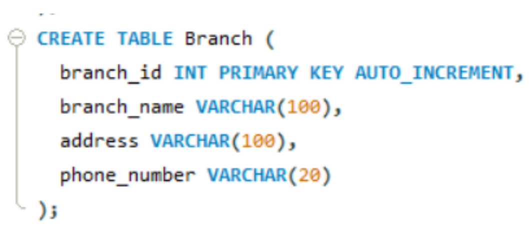
Fig. 2. Enhanced Entity Relationship(EER) Diagram

The back-end code included functions for creating customer accounts, managing transactions, and processing loan requests. The application utilized SQL queries to insert customer details into the Customers table and create new accounts in the Account table. The balance of each account was updated dynamically based on deposit, withdrawal, and loan transactions. Error handling mechanisms were implemented to ensure data integrity and handle exceptions. The back-end code was structured in a modular manner, allowing for easy maintenance and scalability. Overall, the back-end implementation provided a robust foundation for the bank database application, facilitating efficient data management and seamless integration with the front-end.



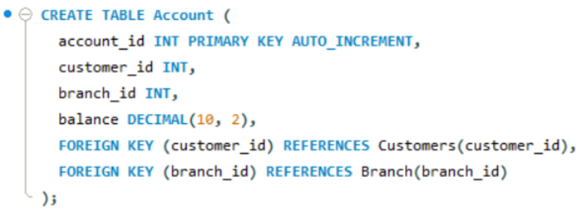
```
CREATE DATABASE bank_app;
USE bank_app;

CREATE TABLE Customers (
  customer_id INT PRIMARY KEY AUTO_INCREMENT,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  address VARCHAR(100),
  phone_number VARCHAR(20),
  email VARCHAR(100)
);
```

Fig. 3. Creating database and Customers table



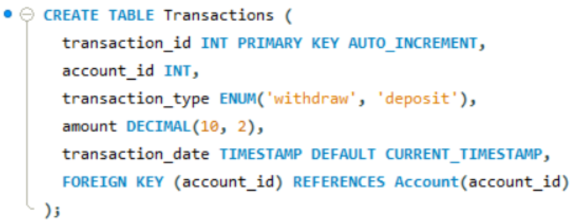
```
CREATE TABLE Branch (
  branch_id INT PRIMARY KEY AUTO_INCREMENT,
  branch_name VARCHAR(100),
  address VARCHAR(100),
  phone_number VARCHAR(20)
);
```

Fig. 4. Creating Branch table



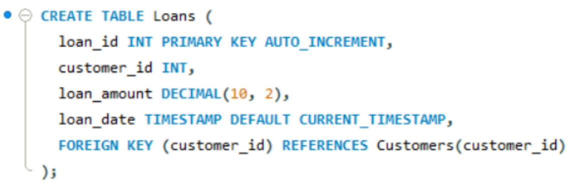
```
CREATE TABLE Account (
    account_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    branch_id INT,
    balance DECIMAL(10, 2),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
    FOREIGN KEY (branch_id) REFERENCES Branch(branch_id)
);
```

Fig. 5. Creating Account table



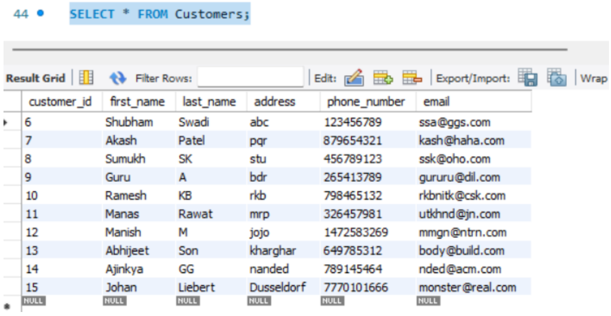
```
CREATE TABLE Transactions (
    transaction_id INT PRIMARY KEY AUTO_INCREMENT,
    account_id INT,
    transaction_type ENUM('withdraw', 'deposit'),
    amount DECIMAL(10, 2),
    transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (account_id) REFERENCES Account(account_id)
);
```

Fig. 6. Creating Transactions table



```
CREATE TABLE Loans (
    loan_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    loan_amount DECIMAL(10, 2),
    loan_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

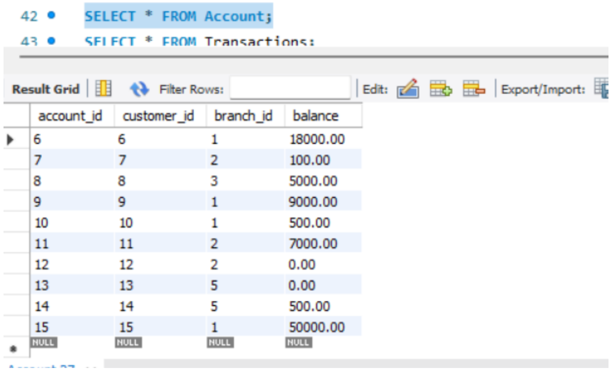
Fig. 7. Creating Loan Table



```
SELECT * FROM Customers;
```

customer_id	first_name	last_name	address	phone_number	email
6	Shubham	Swadi	abc	123456789	ssa@ggs.com
7	Akash	Patel	pqr	879654321	kash@haha.com
8	Sumukh	SK	stu	456789123	ssk@oho.com
9	Guru	A	bdr	265413789	gururu@dil.com
10	Ramesh	KB	rkb	798465132	rkbnitk@csk.com
11	Manas	Rawat	mrp	326457981	utkhnd@jn.com
12	Manish	M	jojo	1472583269	mmgn@ntrn.com
13	Abhijeet	Son	kharghar	649785312	body@build.com
14	Ajinkya	GG	nanded	789145464	nded@acm.com
15	Johan	Liebert	Dusseldorf	7770101666	monster@real.com
NULL	NULL	NULL	NULL	NULL	NULL

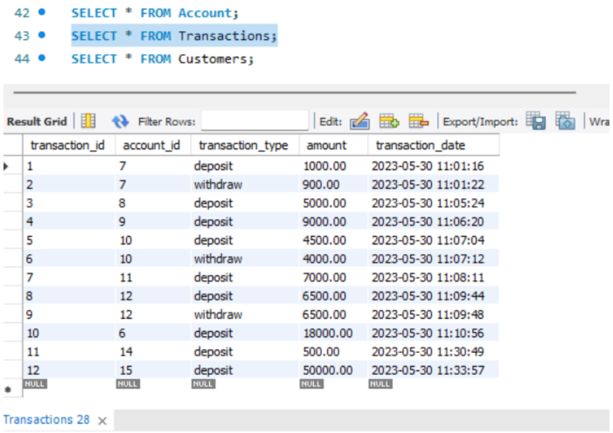
Fig. 8. Values entered into Customers table through GUI



```
SELECT * FROM Account;
SELECT * FROM Transactions;
```

account_id	customer_id	branch_id	balance
6	6	1	18000.00
7	7	2	100.00
8	8	3	5000.00
9	9	1	9000.00
10	10	1	500.00
11	11	2	7000.00
12	12	2	0.00
13	13	5	0.00
14	14	5	500.00
15	15	1	50000.00
NULL	NULL	NULL	NULL

Fig. 9. Values entered into Account table using GUI



```
SELECT * FROM Account;
SELECT * FROM Transactions;
SELECT * FROM Customers;
```

transaction_id	account_id	transaction_type	amount	transaction_date
1	7	deposit	1000.00	2023-05-30 11:01:16
2	7	withdraw	900.00	2023-05-30 11:01:22
3	8	deposit	5000.00	2023-05-30 11:05:24
4	9	deposit	9000.00	2023-05-30 11:06:20
5	10	deposit	4500.00	2023-05-30 11:07:04
6	10	withdraw	4000.00	2023-05-30 11:07:12
7	11	deposit	7000.00	2023-05-30 11:08:11
8	12	deposit	6500.00	2023-05-30 11:09:44
9	12	withdraw	6500.00	2023-05-30 11:09:48
10	6	deposit	18000.00	2023-05-30 11:10:56
11	14	deposit	500.00	2023-05-30 11:30:49
12	15	deposit	50000.00	2023-05-30 11:33:57
NULL	NULL	NULL	NULL	NULL

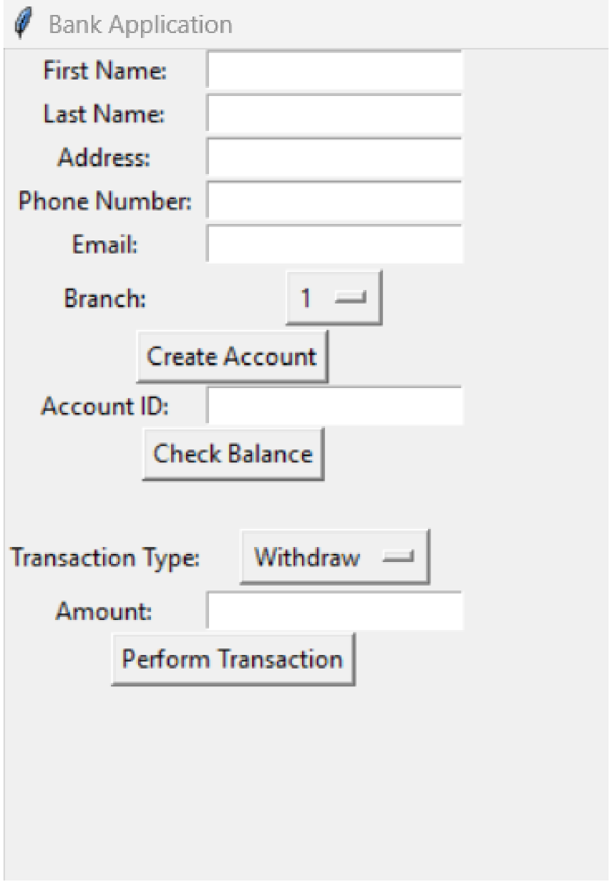
Fig. 10. Records of all transactions taking place through GUI in the Transaction table

C. Front-end Implementation

The front-end implementation of the bank database application was developed using the Tkinter library in Python. The user interface was designed to provide a user-friendly experience with intuitive forms, buttons, and labels. The application included screens for creating new customer accounts, performing transactions such as deposit and withdrawal, and applying for loans. The front-end forms captured user inputs and validated them before sending requests to the back-end for processing. Feedback messages were displayed to the user to provide real-time updates on account balances and transaction statuses. The front-end implementation focused on usability and aesthetics, ensuring a visually appealing and interactive interface for users to seamlessly navigate and interact with the application.

D. User Interface

The user interface of the bank database application was designed to be intuitive and user-friendly. The interface featured a clean and organized layout, with well-placed forms, buttons, and labels. Upon launching the application, users were presented with a login screen where they could enter their account credentials to access their accounts. Once logged in, they were greeted with a dashboard displaying their account details, such as the account balance and recent transactions. The main menu provided easy navigation to different functionalities, such as creating new accounts, performing transactions, and applying for loans. Forms for account creation and transaction processing were designed with clear and concise fields, allowing users to enter the required information accurately. Real-time feedback messages were displayed to users, confirming successful transactions or alerting them to any errors or validation issues. Overall, the user interface aimed to provide a seamless and visually pleasing experience, enabling users to efficiently manage their accounts and perform banking operations with ease.



Branch: 1

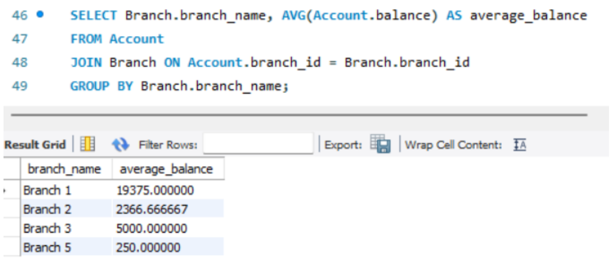
Transaction Type: Withdraw

Fig. 11. Graphical User Interface (GUI) made using tkinter

VIII. Sample Queries

Here are some of the sample queries which we used to test the database

Query 1: Retrieve the average account balance for each branch

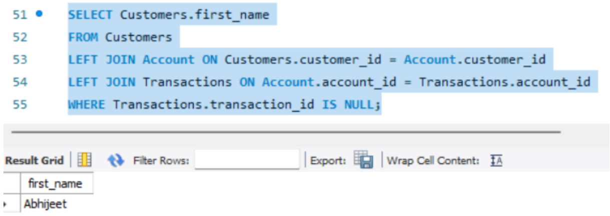


```sql
SELECT Branch.branch_name, AVG(Account.balance) AS average_balance
FROM Account
JOIN Branch ON Account.branch_id = Branch.branch_id
GROUP BY Branch.branch_name;
```

branch_name	average_balance
Branch 1	19375.000000
Branch 2	2366.666667
Branch 3	5000.000000
Branch 5	250.000000

Fig. 12. Query 1

Query 2: Retrieve the customers who have never made a transaction

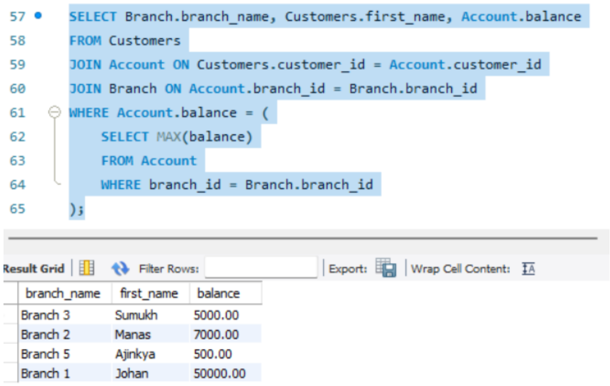


```sql
SELECT Customers.first_name
FROM Customers
LEFT JOIN Account ON Customers.customer_id = Account.customer_id
LEFT JOIN Transactions ON Account.account_id = Transactions.account_id
WHERE Transactions.transaction_id IS NULL;
```

first_name
Abhijeet

Fig. 13. Query 2

Query 3: Retrieve the customers with the highest account balance in each branch

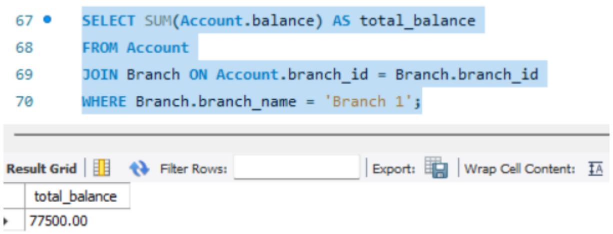


```sql
SELECT Branch.branch_name, Customers.first_name, Account.balance
FROM Customers
JOIN Account ON Customers.customer_id = Account.customer_id
JOIN Branch ON Account.branch_id = Branch.branch_id
WHERE Account.balance = (
    SELECT MAX(balance)
    FROM Account
    WHERE branch_id = Branch.branch_id
);
```

branch_name	first_name	balance
Branch 3	Sumukh	5000.00
Branch 2	Manas	7000.00
Branch 5	Ajinkya	500.00
Branch 1	Johan	50000.00

Fig. 14. Query 3

Query 4: Retrieve the total balance of all accounts in a specific branch

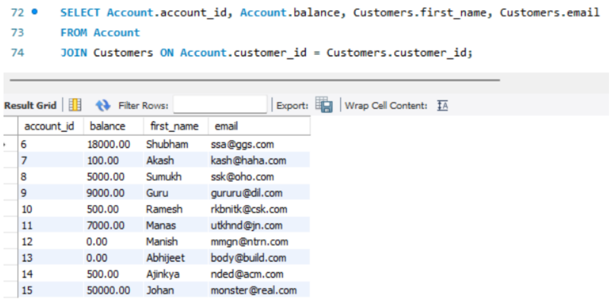


```sql
SELECT SUM(Account.balance) AS total_balance
FROM Account
JOIN Branch ON Account.branch_id = Branch.branch_id
WHERE Branch.branch_name = 'Branch 1';
```

total_balance
77500.00

Fig. 15. Query 4

Query 5: Retrieve account details along with customer information

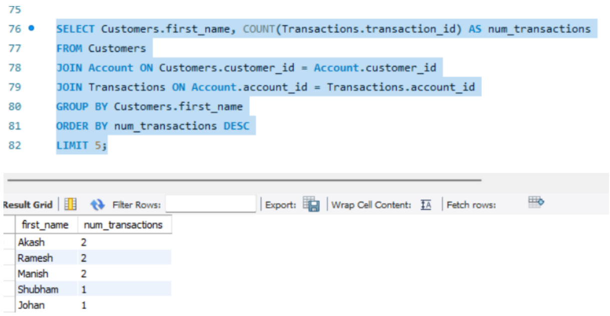


```sql
SELECT Account.account_id, Account.balance, Customers.first_name, Customers.email
FROM Account
JOIN Customers ON Account.customer_id = Customers.customer_id;
```

account_id	balance	first_name	email
6	18000.00	Shubham	ssa@ggs.com
7	100.00	Akash	kash@haha.com
8	5000.00	Sumukh	ssk@oho.com
9	9000.00	Guru	gururu@dil.com
10	500.00	Ramesh	rkbnitk@csk.com
11	7000.00	Manas	utkhnd@jn.com
12	0.00	Manish	mmgn@ntrn.com
13	0.00	Abhijeet	body@build.com
14	500.00	Ajinkya	nded@acm.com
15	50000.00	Johan	monster@real.com

Fig. 16. Query 5

Query 6: Retrieve the customers who have made the most number of transactions

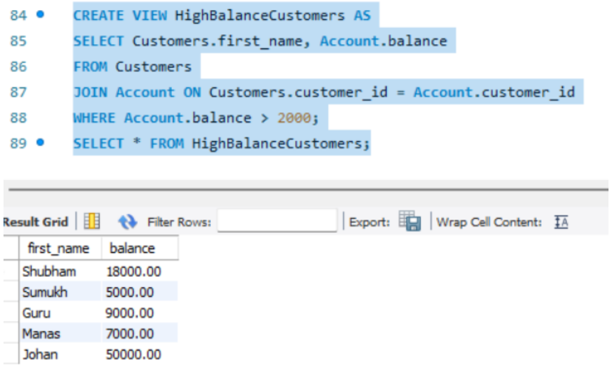


```sql
SELECT Customers.first_name, COUNT(Transactions.transaction_id) AS num_transactions
FROM Customers
JOIN Account ON Customers.customer_id = Account.customer_id
JOIN Transactions ON Account.account_id = Transactions.account_id
GROUP BY Customers.first_name
ORDER BY num_transactions DESC
LIMIT 5;
```

first_name	num_transactions
Akash	2
Ramesh	2
Manish	2
Shubham	1
Johan	1

Fig. 17. Query 6

Query 7: Create a view to retrieve the customers with high account balances above a certain threshold

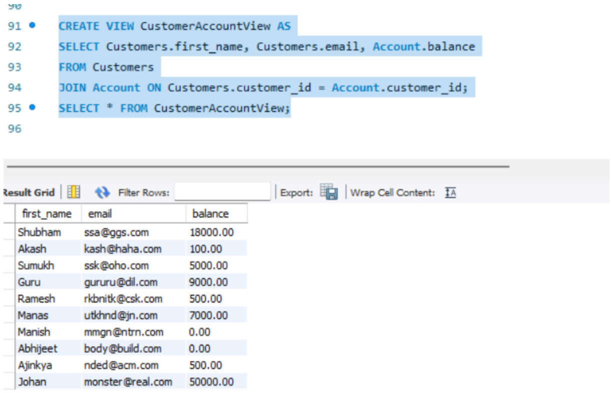


```sql
CREATE VIEW HighBalanceCustomers AS
SELECT Customers.first_name, Account.balance
FROM Customers
JOIN Account ON Customers.customer_id = Account.customer_id
WHERE Account.balance > 2000;
SELECT * FROM HighBalanceCustomers;
```

first_name	balance
Shubham	18000.00
Sumukh	5000.00
Guru	9000.00
Manas	7000.00
Johan	50000.00

Fig. 18. Query 7

Query 8: Create a view to retrieve customer details along with their account balance

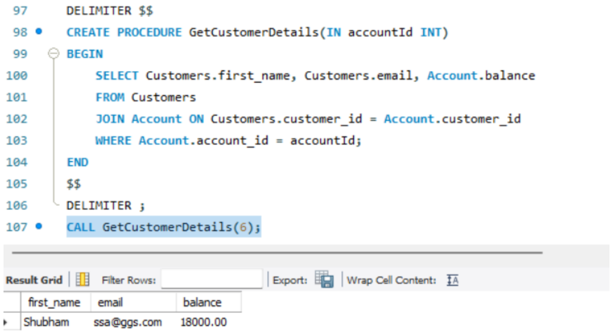


```sql
CREATE VIEW CustomerAccountView AS
SELECT Customers.first_name, Customers.email, Account.balance
FROM Customers
JOIN Account ON Customers.customer_id = Account.customer_id;
SELECT * FROM CustomerAccountView;
```

first_name	email	balance
Shubham	ssa@ggs.com	18000.00
Akash	kash@haha.com	100.00
Sumukh	ssk@oho.com	5000.00
Guru	gururu@dil.com	9000.00
Ramesh	rkbnitk@csk.com	500.00
Manas	utkhnd@jn.com	7000.00
Manish	mmgn@ntrn.com	0.00
Abhijeet	body@build.com	0.00
Ajinkya	nded@acm.com	500.00
Johan	monster@real.com	50000.00

Fig. 19. Query 8

Query 9: Create a stored procedure to retrieve customer details based on their account ID

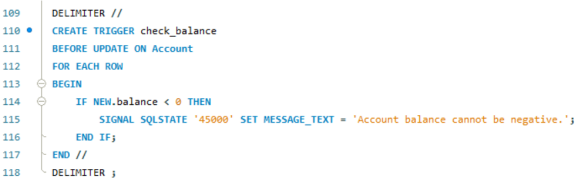


```sql
DELIMITER $$
CREATE PROCEDURE GetCustomerDetails(IN accountId INT)
BEGIN
    SELECT Customers.first_name, Customers.email, Account.balance
    FROM Customers
    JOIN Account ON Customers.customer_id = Account.customer_id
    WHERE Account.account_id = accountId;
END
$$
DELIMITER ;
CALL GetCustomerDetails(6);
```

first_name	email	balance
Shubham	ssa@ggs.com	18000.00

Fig. 20. Query 9

Query 10: Create a trigger to enforce a constraint that the account balance should not be negative



```sql
DELIMITER //
CREATE TRIGGER check_balance
BEFORE UPDATE ON Account
FOR EACH ROW
BEGIN
    IF NEW.balance < 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Account balance cannot be negative.';
    END IF;
END //
DELIMITER ;
```

Fig. 21. Query 10

IX. Usage

The Bank Database project can be used as a reference for anyone interested in learning about database management systems or building a banking system using a relational database. The project can also be used as a starting point for building a more complex banking system with additional features and functionality. The project is open source and available for anyone to use, modify, or distribute.

X. Conclusion

The Bank Database project is a comprehensive database management system that simulates a banking system. It is designed to manage customer and account information and offers a range of features to its users. The project has been implemented using MySQL and is open source, making it an excellent resource for anyone interested in learning about database management systems or building a banking system using a relational database.

References

[1] MySQL Documentation, Link

[2] javaTpoint, MySQL Tutorial

[3] W3schools, MySQL Tutorial

[4] Graphical User Interfaces with Tk, Documentation