



**BERLIN SCHOOL OF  
BUSINESS & INNOVATION**

**Essay / Assignment Title: Database System Design for a Banking System**

**Programme title: Enterprise Data Warehouses and Database Management  
Systems**

**Name: Sanket Arjanbhai Savaliya**

**Year: 2023-24**

## CONTENTS

INTRODUCTION.....	4
CHAPTER ONE.....	5
CHAPTER TWO.....	9
CHAPTER THREE.....	22
CUNCLUSION.....	24
BIBILOGRAPHY.....	25

## Statement of compliance with academic ethics and the avoidance of plagiarism

I honestly declare that this dissertation is entirely my own work and none of its part has been copied from printed or electronic sources, translated from foreign sources and reproduced from essays of other researchers or students. Wherever I have been based on ideas or other people texts I clearly declare it through the good use of references following academic ethics.

(In the case that is proved that part of the essay does not constitute an original work, but a copy of an already published essay or from another source, the student will be expelled permanently from the postgraduate program).

Name and Surname (Capital letters):

SANKET SAVALIYA

.....

Date: 5/10/2023

## INTRODUCTION

Massive volumes of data must be efficiently organized and managed, which requires database management systems (DBMS). Because they handle structured data so effectively and ensure data integrity and consistency, relational database management systems (RDBMS) have become more popular than other DBMS kinds. We shall look into MySQL's use as an RDBMS in the stock market in this context.

Users can store, organize, and retrieve data in a methodical fashion with the use of a program known as a database management system (DBMS). Data security, concurrency management, data integrity, and query optimization are just a few of the crucial characteristics that DBMS provides to guarantee reliable and efficient data operations. Employing a DBMS may help businesses increase data accessibility, speed up data processes, and support decision-making.

Databases' creation, maintenance, and administration Database design is crucial for ensuring data reliability, efficiency, and accuracy. This section discusses the principles and techniques of database design, including entity-relationship (ER) modeling, normalization, and schema refining. It also includes responsibilities for database administration and maintenance, including backup and recovery, security controls, and performance optimization.

## CHAPTER ONE

- ✚ **Design a database system for a bank. This bank can have several branches. In addition, any branch can have many clients. Also, any client can have, maximum, 10 accounts. Moreover, the clients can transfer money to their own accounts as well as to any other account (in this bank). You need to describe all the entities and attributes. In addition, by providing your own supportive reasons, you are requested to choose the proper DBMS (Relational or Non-Relational).**

It's essential to design a reliable database system for a bank with several branches, numerous customers, and complicated transactional needs. Such a database system has to manage branch, client, account, and transaction data effectively while maintaining data consistency, security, and scalability.

### **Entities and Attributes:**

#### 1. Branch:

- “Branch” entity is shows the details with their respective attributes.
- The Branch table includes:

Branch\_id (Identify number for the branch)

Branch\_name (Name of the Branch)

Address (Branch address)

Phone (number of branch)

Manager\_name (Manager of branch)

#### 2. Client:

- “Client” entity includes a number of things that keep important data on Account in the bank.

- The Client table includes:

Client\_ID (Identify number for the Client)

Client\_FirstName (Name of the Client)

Client\_LastName (Name of the client Last name)

Client\_DateOfBirth (client's dateofbirth)

Client\_Address (Home address of client)

Client\_Phone (Phone number of client)

Client\_Email(Client's Email address)

### 3. Account:

- “Account” entity is shows the details Client's account details and important information.
- The Account table includes:

Account\_id (Identify number for the Account)

Account\_type (Type of account)

Account\_Balance (Show Account Balance)

Account\_Opendate (opening account date)

Client\_id (clientid for identity of client)

Branch\_ID (branchid for identity of branch)

### 4. Transaction:

- “Transaction” entity is shows the details millions of transacation.
- The Transaction table includes:

Transaction\_id (Identify number for the Transaction)

Transaction\_Date (Date of Transaction)

Transaction\_Amount (Amount of money)

FromAccountID (being transferred)

ToaccountID (Finish transaction)

#### 5. Transfer:

- “Transfer” entity is shows the details of Client’s Transfer amount.
- The Transfer table includes:

Transfer\_id (Identify number for the transfer)

SourceAccountID (Account of send transfer money)

DestinationAccountID (Account of receive money)

Amount (total amount)

TransferDate (Date of transfer)

### **The proper DBMS (Relational or Non-Relational):**

**Data Consistency:** RDBMS systems, such as MySQL, PostgreSQL, or Oracle, are renowned for their robust support for data consistency and integrity. In the financial industry, where data accuracy is critical, this is essential.

**ACID Compliance:** Transactions are reliably executed by RDBMS systems because they uphold the ACID (Atomicity, Consistency, Isolation, Durability) principles. Maintaining these qualities is essential in banking, where financial transactions are involved.

**Structured Data:** The established schemas used by RDBMS systems make them excellent at managing structured data, which is how bank data is often structured.

**Complex Queries:** For the purpose of producing reports, audits, and compliance checks, banking systems frequently need complicated queries. Such queries may be efficiently handled by RDBMS systems.

### **Non-Relational DBMS:**

**Scalability:** For the purpose of producing reports, audits, and compliance checks, banking systems frequently need complicated queries. Such queries may be efficiently handled by RDBMS systems.

**Flexibility:** NoSQL databases provide semi-structured or unstructured data, allowing for data model changes without requiring major schema revisions. As financial systems develop, this flexibility may be advantageous.

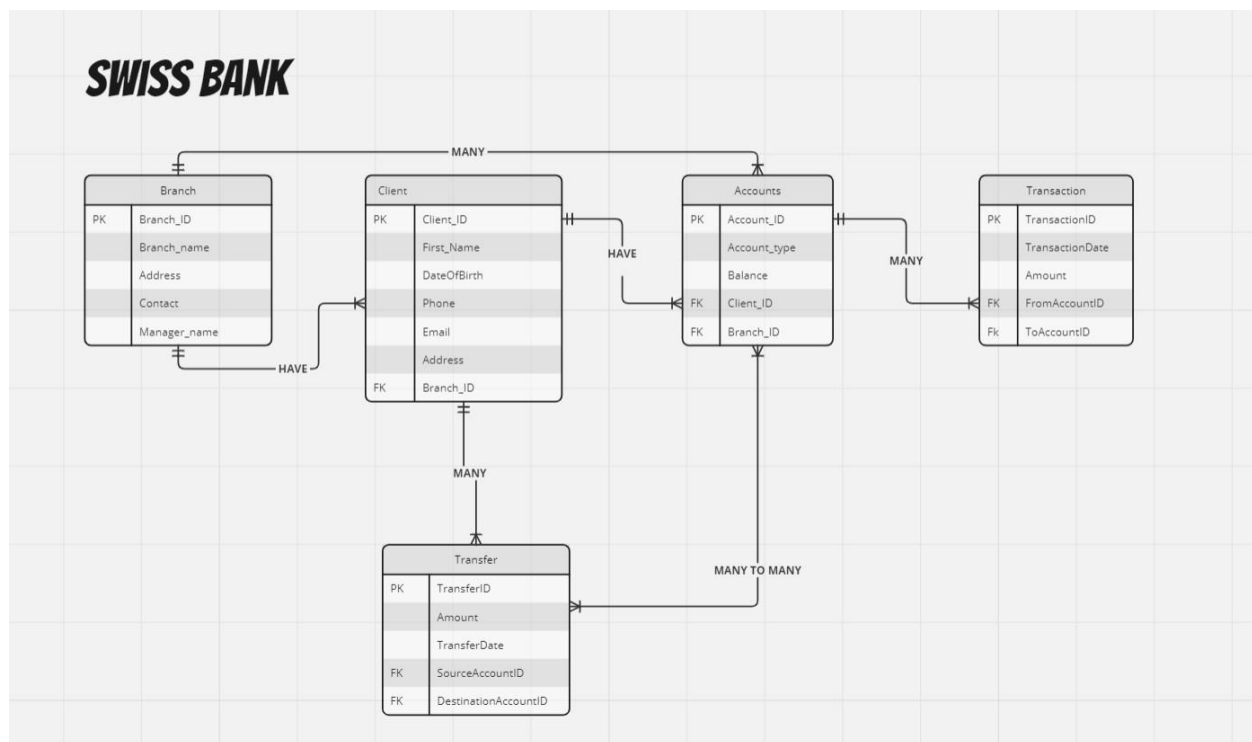
**High Throughput:** NoSQL databases are renowned for their capacity to manage large amounts of data reads and writes, which is beneficial for online banking services with a large user base.

The final decision about the DBMS will be made in light of the unique requirements, projected growth, and financial resources of the bank. An RDBMS is a natural choice for a bank with an emphasis on data integrity, demanding reporting specifications, and a structured data model. A NoSQL database, however, may be taken into consideration if the bank expects significant expansion and is able to handle the challenges of guaranteeing data consistency in such a setting. The decision should be in line with the technical capabilities and long-term strategic aims of the bank.



## CHAPTER TWO

- ✚ Draw an ER Diagram for your database. Describe all possible schemas. Also, define all the tables (including attributes) and specify the types and length using SQL commands (your tables need to be filled by some meaningful data). Write, at least, eight SQL queries and explain the output of the queries\*. Note that these eight queries must be complementary for/to each other in the form that they collectively shape a report. This means that they should not be structurally the same (with the same purpose).



(Above image self-created by using Miro)

The ER diagram is created by using Miro website. (created, n.d.) The entities and their connections in the Bank database system are represented by the ER diagram.

### Schemas:

1. **Branch**(**Branch\_ID(PK)**, Branch\_name, Address, Contact, Manager\_name)
2. **Client** (**Client\_ID(PK)**, Firstname, Dateofbirth, Phone, Email, Address, **BranchID(FK)**)
3. **Accounts**(**Account\_ID(PK)**, Accounttype, Balance, **Client\_ID(FK)**, **Branch\_ID(FK)**)
4. **Transaction**(**Transaction\_ID(PK)**, TransactionDate, Amount, **FromaccountID(FK)**, **ToAccountID(FK)**)
5. **Transfer**(**Transfer\_ID(PK)**, Amount, Transferdate, **SourceAccountID(FK)**, **DestinationAccountID(FK)**)

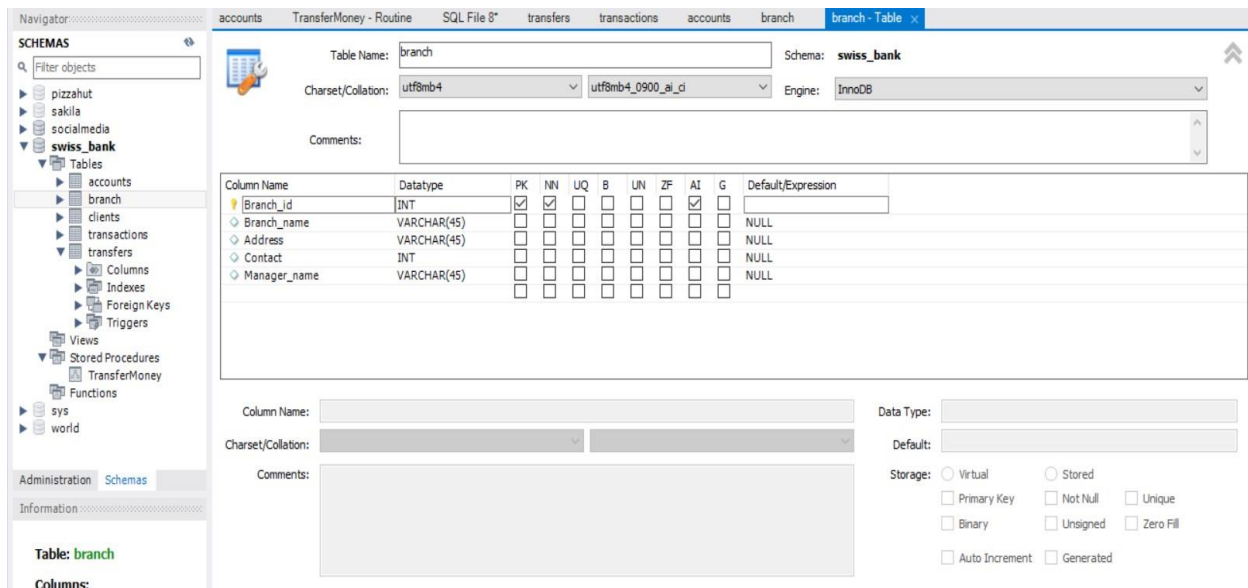
The explanations of each table's relations are as follows:

1. Branch:
  - The Branch entity has a one-to-many relationship with the client and accounts entity. A branch can be associated with many client and account with that.
2. Client:
  - One-to-Many (Each branch may have a number of customers, but each client only has a single branch affiliation).
3. Account:
  - One-to-Many Relationship with the customer (Each customer may have several accounts, but each account only has access to one client).
  - One-to-Many with Branch (Each branch may have a lot of accounts, but every account only has one branch as its owner).
4. Transaction:
  - Many-to-Many (Many-to-many relationships can be seen when several accounts are involved in numerous transactions. Money can be moved by customers between several accounts).
5. Transfer:
  - Many-to-Many (Many-to-many relationships are shown by the fact that numerous transfers might include many accounts. Transfers between several accounts can be started by clients).

In Conclusion, The associations between the tables in your bank database are established by these primary keys and foreign keys. They uphold referential integrity restrictions and guarantee data integrity. For instance, the one-to-many link between branches and customers is established by using the BranchID foreign key from the Client record to refer to the BranchID primary key from the Branch table. Similar to this, the Client and Branch tables are referred to by foreign keys in the Account, Transaction, and Transfer columns, providing data links throughout the whole database.

## Creating Tables in MySQL Workbench:

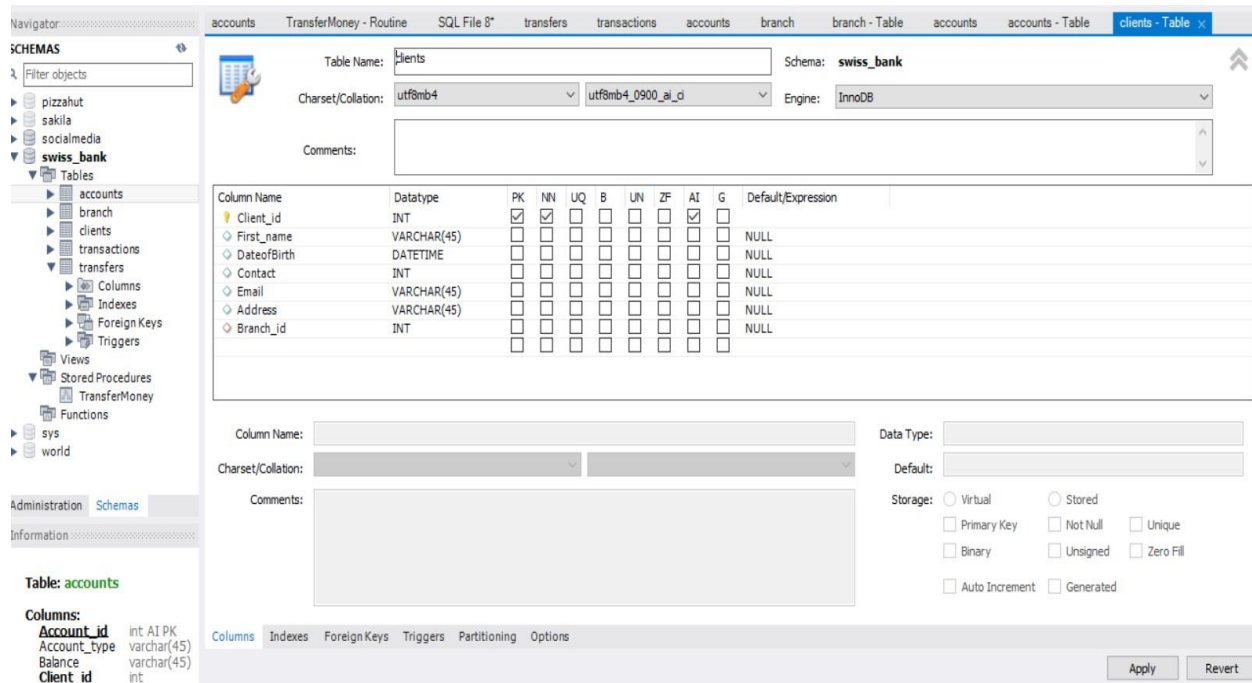
### 1. “Branch Table”



(Above image self-created by using MySQL Workbench)

The "Swiss bank" database has been built and chosen for usage. Within this database, a table called "branch" has been made to house data about banks. The table has columns for branch\_id, an integer, branch\_name, address, Contact and manager\_name, all of which are varchar and contact is int. As the primary key, the branch\_id column makes sure that each user has a special identification. The information about the users who participate in the bank will be contained in this table.

## 2. "Client Table"



(Above image self-created by using MySQL Workbench)

The "Client" table is built with columns for order information, including Client\_id as the primary key, Firstname to specify client name, Dateofbirth for the client identity, Contact for the client contact, Email for specific identity, address to show the Client address, Branch\_id as a foreign key referencing the "client" table. These foreign keys provide connections between the tables for Branch, Account and Transfer.

### 3. "Account Table"

The screenshot shows the MySQL Workbench interface for creating or editing a table. The 'accounts' table is selected in the 'swiss\_bank' schema. The table properties panel shows the following details:

- Table Name:** accounts
- Schema:** swiss\_bank
- Charset/Collation:** utf8mb4 / utf8mb4\_0900\_ai\_ci
- Engine:** InnoDB

The column structure is as follows:

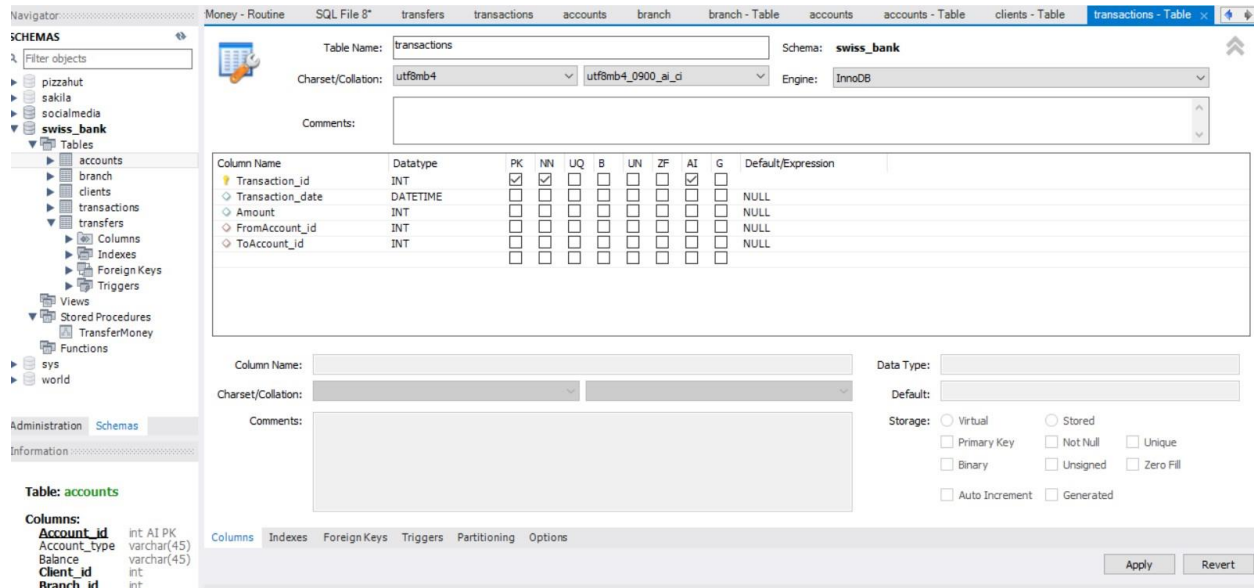
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
Account_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Account_type	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
Balance	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
Client_id	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
Branch_id	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Below the table structure, there are fields for 'Column Name', 'Data Type', 'Charset/Collation', 'Default', and 'Storage' options. The 'Columns' tab is selected at the bottom.

(Above image self-created by using MySQL Workbench)

To hold data on account, columns were added to the "account" database. It has a main key called "account\_id" that enables distinct stock identification. The "Account\_type", "Account\_Balance," columns and others include pertinent information about the Account. Furthermore, the "client\_id" column creates a foreign key link with the "branch" table, enabling the connection of certain branch and client. In the database system, this table offers organized storage for bank data.

#### 4. "Transaction Table"



(Above image self-created by using MySQL Workbench)

The "Transaction\_ID" table is made up of four columns: "Transaction\_id" as the table's primary key, "Transaction\_Date" to store the date of transactions, "Amount" to keep track of the entry amount and "Fromaccount\_ID" and "Toaccount\_ID" as a foreign key referring to the "accounts" table. This table's purpose is to keep track of and manage transactions in the bank, and the foreign key constraint links each transaction entry to a particular client.

## 5. "Transfer Table"

Table Name:  Schema: **swiss\_bank**

Charset/Collation:   Engine:

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
Transfer_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Amount	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
Transfer_date	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
SourceAccount_id	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
DestinationAccount_id	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Column Name:  Data Type:

Charset/Collation:   Default:

Comments:

Storage: ☐ Virtual ☐ Stored

☐ Primary Key ☐ Not Null ☐ Unique

☐ Binary ☐ Unsigned ☐ Zero Fill

☐ Auto Increment ☐ Generated

Columns Indexes Foreign Keys Triggers Partitioning Options

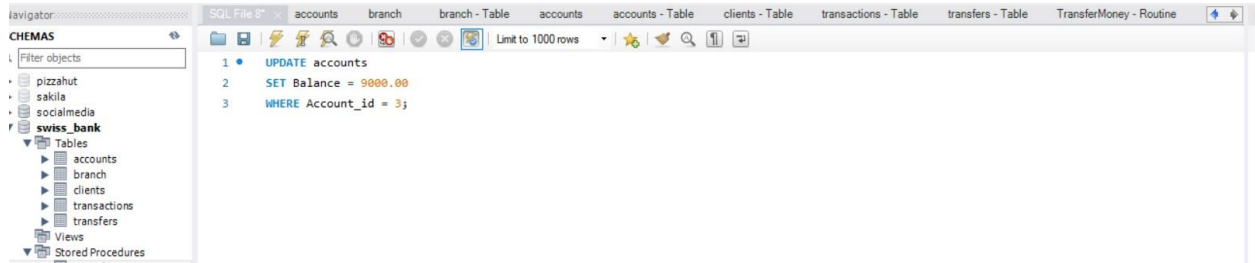
Apply Revert

(Above image self-created by using MySQL Workbench)

There are five columns added to the "transfer" table: Transfer\_id (the the primary key), Amount (a decimal value), transfer\_date, sourceaccount\_id, destinationaccount\_ID. A foreign key exists the sourceaccount\_id and destinationaccount\_ID column.

## SQL Queries:

### 1. Query for update accounts and set balance in account table



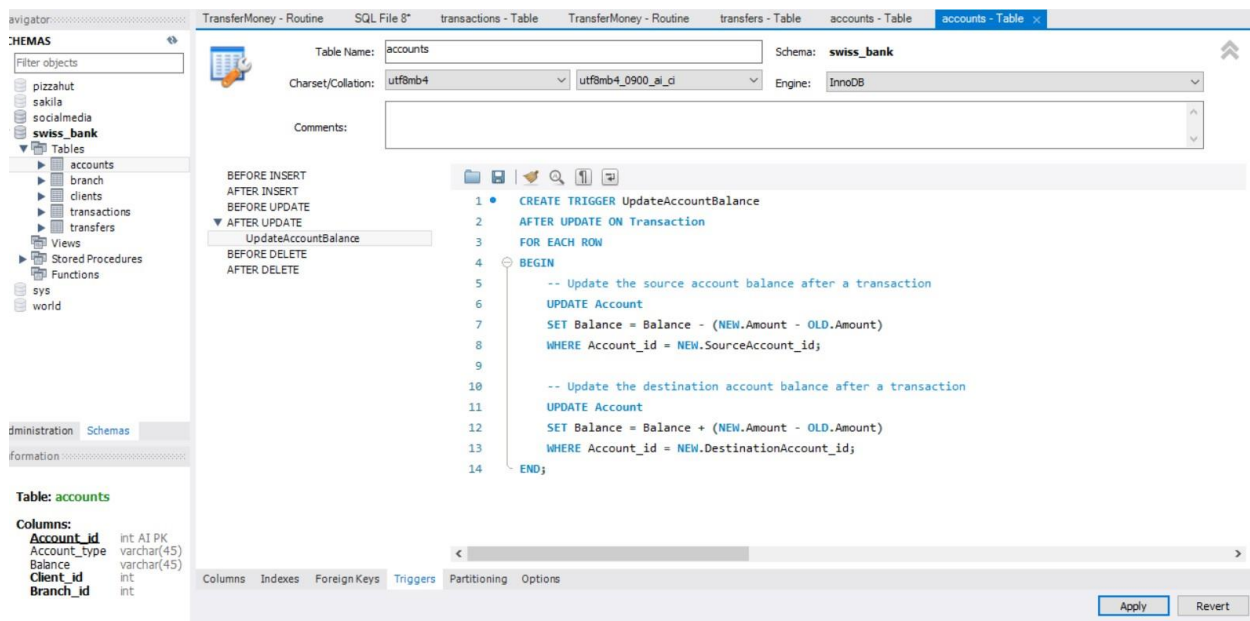
(Above image self-created by using MySQL Workbench)

UPDATE Account identifies the "Account" table, which is the one you wish to change.

The "Balance" column is updated to the new amount you want to update (in this example, \$5000.00) with the command updated Balance = 9000.00.

WHERE The criteria to identify the account you wish to update is AccountID = 3, which is specified. Replace 3 with the AccountID you really wish to change.

I used “after update trigger” to run above mentioned query. Trigger is mentioned below:



(Above image self-created by using MySQL Workbench)



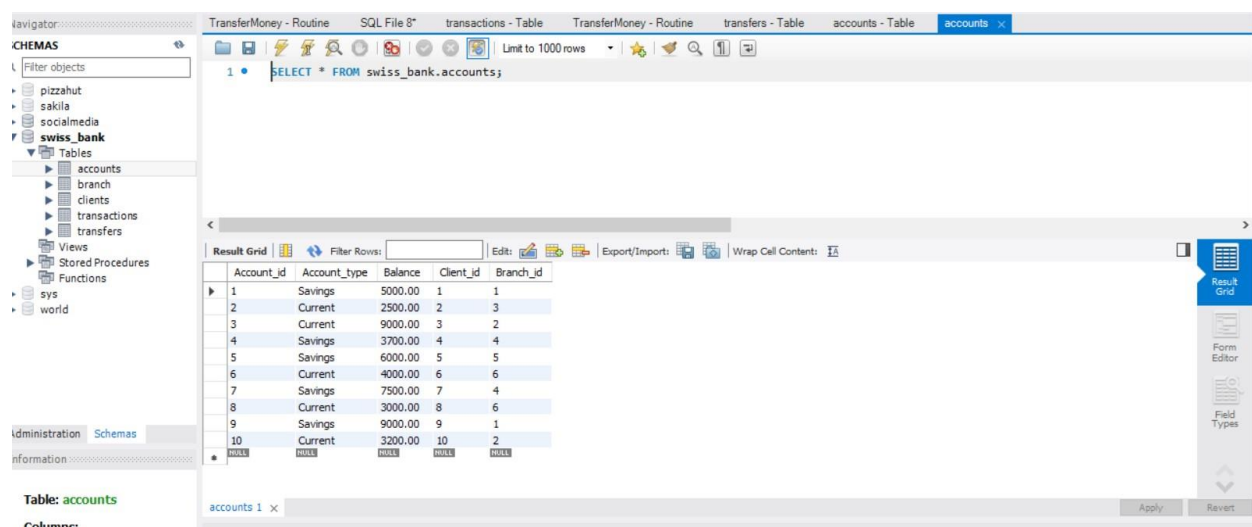
The "Transaction" table's trigger is configured to fire AFTER INSERT, which means it will go into effect following the insertion of a new transaction.

The transaction amount (NEW.Amount) is subtracted from the source account's balance to update it, and the transaction amount is added to the destination account's balance to update it.

Make careful to adjust the trigger so that it complies with your unique database's naming standards and structure, including the names of your tables and columns.

## Output:

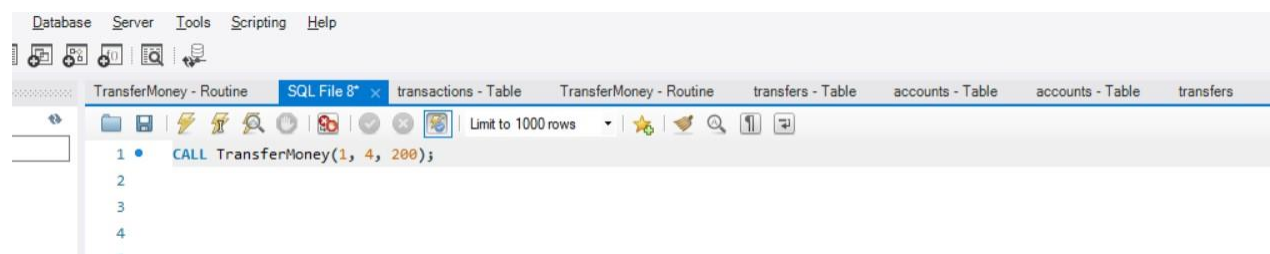
Update\_accountbalance table



The screenshot shows the MySQL Workbench interface. On the left, the 'Navigator' pane displays the database schema 'CHEMAS' with tables 'accounts', 'branch', 'clients', 'transactions', and 'transfers'. The 'accounts' table is selected. The main window shows the 'Result Grid' for the query 'SELECT \* FROM swiss\_bank.accounts;'. The table has 10 rows of data.

Account_id	Account_type	Balance	Client_id	Branch_id
1	Savings	5000.00	1	1
2	Current	2500.00	2	3
3	Current	9000.00	3	2
4	Savings	3700.00	4	4
5	Savings	6000.00	5	5
6	Current	4000.00	6	6
7	Savings	7500.00	7	4
8	Current	3000.00	8	6
9	Savings	9000.00	9	1
10	Current	3200.00	10	2

## 2. Query for create new Transfer using stored procedures.



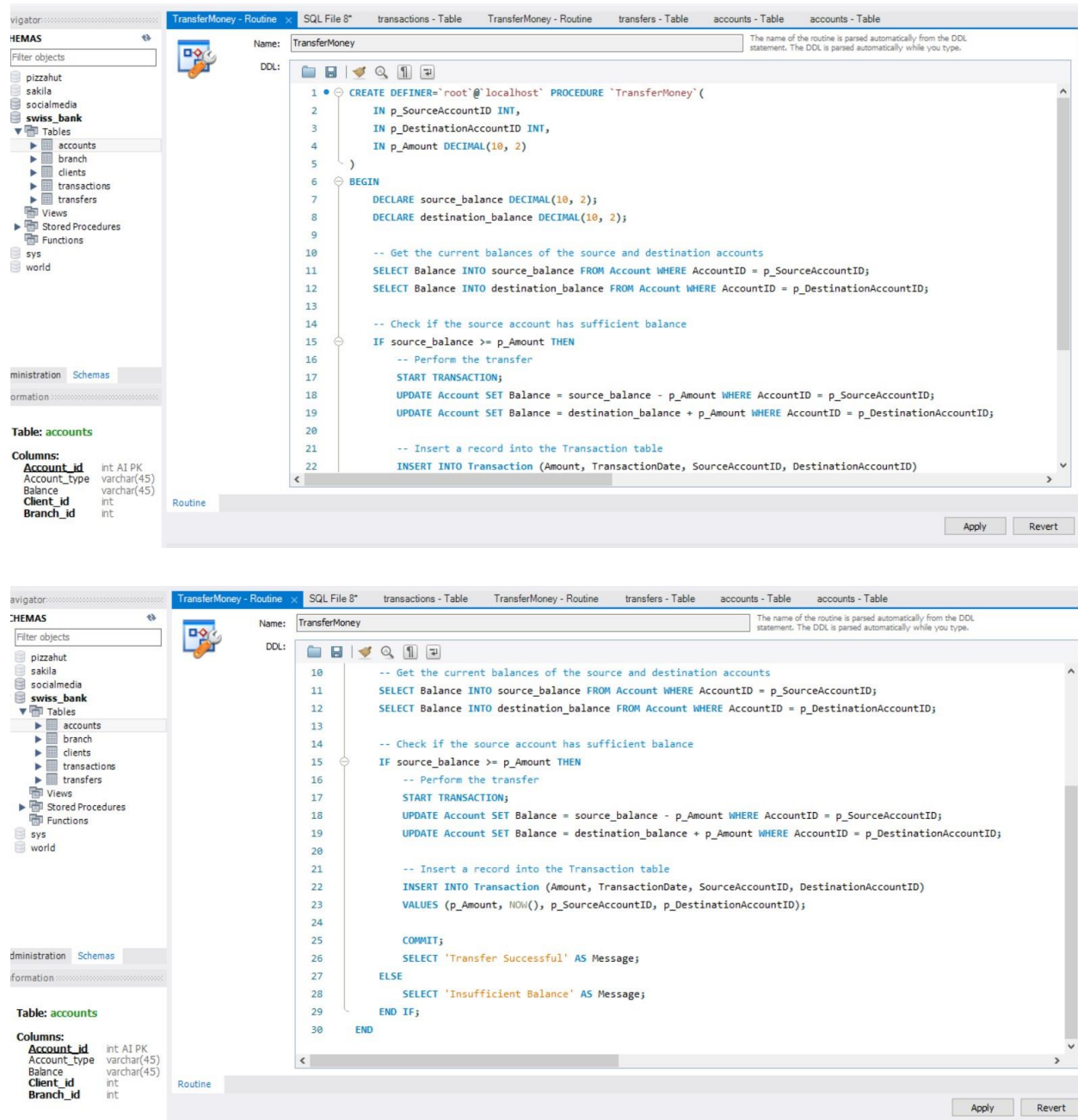
The screenshot shows the MySQL Workbench interface with the 'SQL File 8\*' tab active. The query editor contains the following SQL statement:

```
1 • CALL TransferMoney(1, 4, 200);
```

(Above image self-created by using MySQL Workbench)

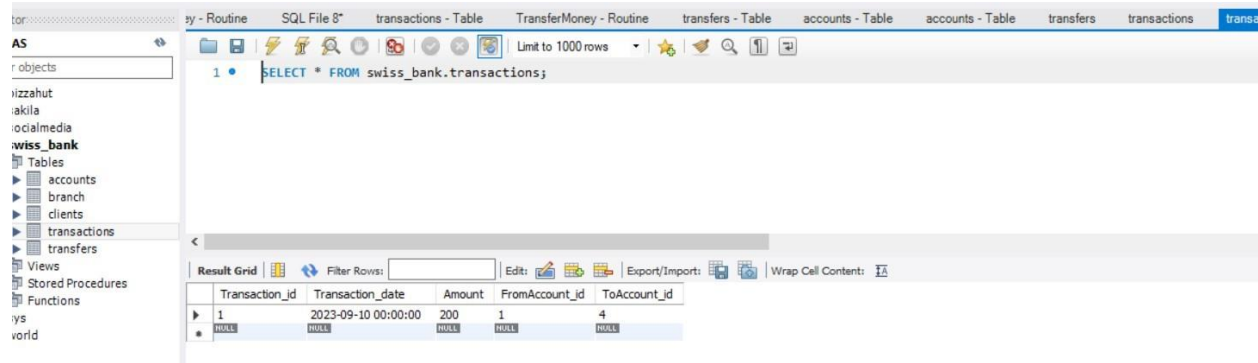
It is TransferMoney stored procedure with parameters. However, based on the parameters, the TransferMoney procedure is expected to transfer 200 from AccountID 1 to AccountID 4.

To run this query, I used stored procedures and two triggers are mentioned below:



(Above image self-created by using MySQL Workbench)

Output:

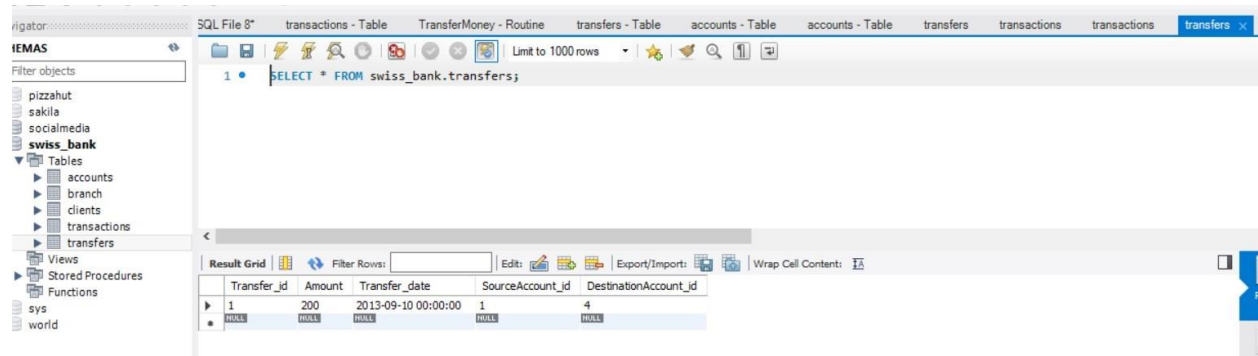


The screenshot shows the MySQL Workbench interface. The left sidebar displays a database schema with a table named 'transactions' under the 'swiss\_bank' database. The main window shows a SQL query: `SELECT * FROM swiss_bank.transactions;`. The result grid below the query displays the following data:

Transaction_id	Transaction_date	Amount	FromAccount_id	ToAccount_id
1	2023-09-10 00:00:00	200	1	4

(Above image self-created by using MySQL Workbench)

In the output transaction table stores the order details what is the transaction amount, date, fromaccountid, toaccountid which transaction is made and who is the transaction in the account. In query we selected “1” number fromaccountid and toaccountid is 4 so that transaction process between both accounts. Whole details are saved in the transfer table.



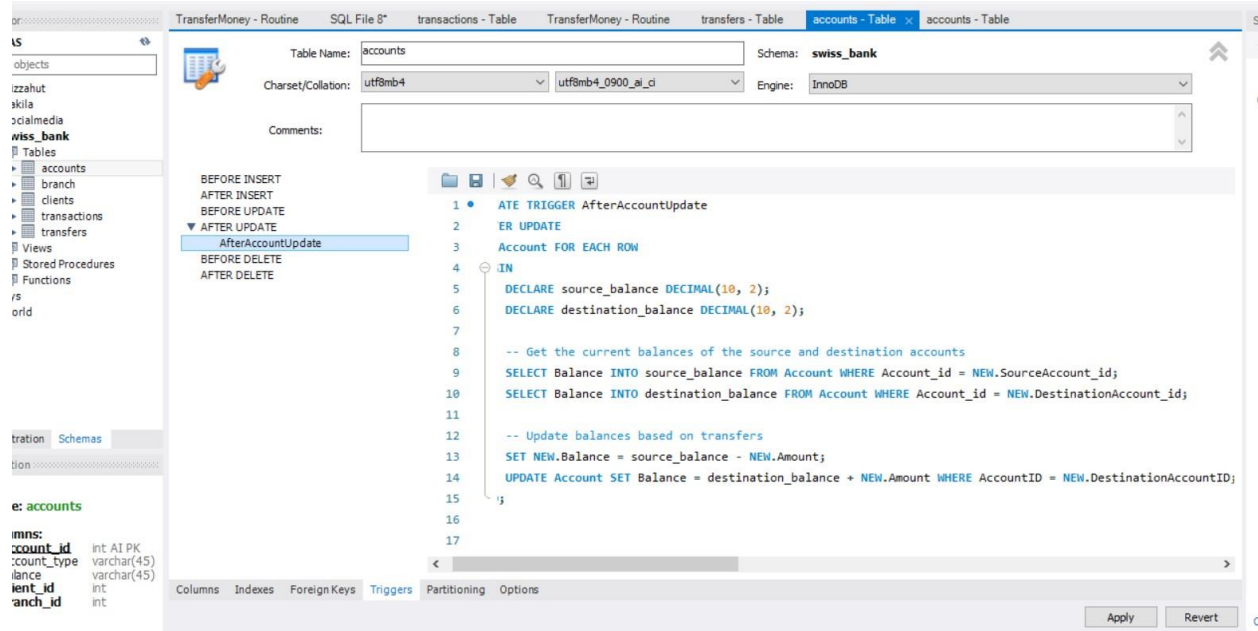
The screenshot shows the MySQL Workbench interface. The left sidebar displays a database schema with a table named 'transfers' under the 'swiss\_bank' database. The main window shows a SQL query: `SELECT * FROM swiss_bank.transfers;`. The result grid below the query displays the following data:

Transfer_id	Amount	Transfer_date	SourceAccount_id	DestinationAccount_id
1	200	2013-09-10 00:00:00	1	4

(Above image self created by using MySQL Workbench)

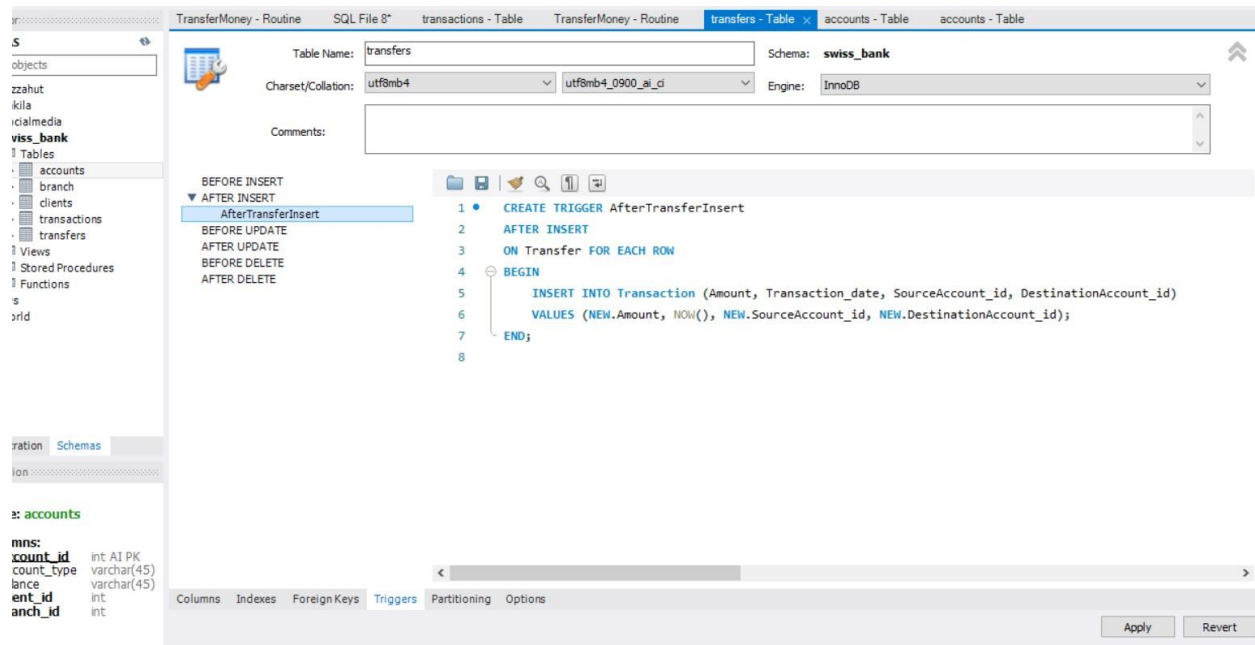
This image is from transfer table. This table stores the transfer details about quantity the transfer amount of whole transaction and branch details and account detail.

To run above mentioned stored procedures query I used two triggers which is mentioned below:



(Above image self-created by using MySQL Workbench)

First trigger is "before insert trigger". This trigger is use for update the account status. Before each insert operation on the "account" database, the trigger mentioned above, called "afteraccountupdate," is generated to be executed. 'root'@'localhost' is the definer for the trigger. It determines if the status of the new transaction is "Pending" and, if it is, adjusts the status inside the trigger code to "Complete." This trigger makes sure that, upon the successful completion of the stored process, the status of any new transaction with a "Pending" status is automatically changed to "Complete."



(Above image self-created by using MySQL Workbench)

Second trigger is for “after insert trigger”. A trigger called "insert\_transfer" is created by the provided code snippet and is activated whenever an insertion takes place on the "transaction" database. To add a new row to the "Transactions" table, the trigger runs a SQL command. The transaction\_id, transaction\_amount, transaction\_date, branch\_id, and account\_id columns in the "Transactions" database are filled with the pertinent data that was retrieved from the inserted entry in the "orders" table (NEW)

## CHAPTER THREE

- ✚ **Describe and analyse CAP theorem as well as the important factors that affect the efficiency of your database (e.g., Transactions, Consistency, Scalability).**

A foundational idea in distributed systems, the CAP theorem, commonly referred to as Brewer's theorem, was developed by computer scientist Eric Brewer. It claims that a distributed system cannot give all three of the following guarantees at the same time:

1. **Consistency (C):** The most recent write or an error are returned by every read action on the system. In other words, the system's nodes are in agreement with the data's present state.
2. **Availability (A):** The system responds to each request (read or write), without ensuring that the information is the most current. The term "availability" means that the system is constantly responsive and functional.
3. **Partition Tolerance (P):** Even in the face of network partitions, i.e., network faults that prohibit some nodes from interacting with others, the system can still function.

According to the CAP theorem, only two of these three guarantees may be met at once in a distributed system. This implies that you must compromise based on the needs and priorities of your system. Now let's think about how this relates to variables influencing database effectiveness:

1. **Transactions:** In a database system, transactions guarantee the accuracy of the data. The decision between availability and consistency may have an impact on how transactions are handled. Transactions may take longer in systems that prioritize consistency (CP) since it's important to make sure that all nodes concur on the data. While systems that prioritize availability (AP) may enable speedier transactions, they may also increase the risk of data consistency.
2. **Consistency:** Strong consistency is frequently required of database systems, particularly in situations where data correctness is crucial. For instance, healthcare databases and

banking systems demand exact consistency. Strong consistency can be difficult to achieve and have an influence on system performance, especially for distributed databases.

3. **Scalability:** When a database has to scale, achieving both consistency and availability (CA) can be difficult. Maintaining consensus (consistency) while guaranteeing high availability might become more challenging as the number of nodes and users rises. For effective scalability, distributed databases must carefully balance these factors.
4. **Latency:** Lower latency is frequently implied by availability (A), which is important in real-time applications. For instance, in an e-commerce system, it is more crucial to respond to a client very away, even if the information is not totally accurate, than to wait the answer in order to provide more consistency.
5. **Data Partitioning:** Data in distributed databases is frequently divided among several nodes. A major challenge is how to handle network partitions (P). In the event of a network failure, distributed databases must efficiently manage data synchronization and recovery.
6. **Data Replication:** Many distributed databases use data replication to increase availability. Due to the necessity to propagate modifications across several copies of the data, this may have an impact on consistency. For keeping consistency and availability, effective replication and dispute resolution systems are crucial.

Finally, it should be noted that a variety of factors, including as transaction management, consistency procedures, scalability, indexing, and the underlying hardware architecture, influence how effective a database is. By effectively addressing these factors, database managers may improve the database system's performance, data integrity, and overall effectiveness.

## CONCLUDING REMARKS

Finally, establishing a dependable database system for a bank with complicated transactional demands necessitates careful consideration of several factors, including the selection of a database management system (DBMS), schema design, and SQL query formulation. The ER diagram visualizes the relationships between database elements, allowing for a clear understanding of how data is arranged.

The selection of a relational (RDBMS) or non-relational DBMS should be based on the bank's unique requirements, growth expectations, and financial resources. An RDBMS such as MySQL, PostgreSQL, or Oracle is a logical solution for a bank with a strong emphasis on data integrity, complicated reporting needs, and structured data. A NoSQL database, on the other hand, may be considered if the bank anticipates significant expansion and is prepared to meet the issues of guaranteeing data consistency in such a dynamic environment.

The CAP theorem was also explained, highlighting the trade-offs between Consistency, Availability, and Partition Tolerance. Transaction processing, data consistency, scalability, latency, data partitioning, and data replication are all important factors in determining database performance and dependability.

The selection of a database management system (DBMS), schema design, and the implementation of queries and data management techniques are all critical components of developing a system that fulfills the particular demands of the financial industry while maintaining the greatest levels of data security and consistency. It is a dynamic and developing process that need continual maintenance and optimization to guarantee the database meets the bank's operational needs and long-term strategic goals.



## BIBLIOGRAPHY

<https://www.mysql.com/>

<https://www.oracle.com/database/what-is-a-relational-database/#:~:text=The%20software%20used%20to%20store,storage%2C%20access%2C%20and%20performance>

<https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-what-is-database-management-system>

<https://www.techtarget.com/searchdatamanagement/definition/RDBMS-relational-database-management-system>

<https://www.analyticsvidhya.com/blog/2020/08/a-beginners-guide-to-cap-theorem-for-data-engineering/>

<https://www.gosink.in/demystifying-the-cap-theorem-a-comprehensive-guide-to-distributed-databases/>

<https://www.ibm.com/topics/cap-theorem>