Swift Bank Management System - Final Report

Group 7 (Akash Valathappan, Tarunya Daga, Riha Sindhe)

Problem Statement:

Swift Bank is a new private bank that aims to help local people with real-time transaction processing. The current system lacks security measures, and there is a need to implement a secure layer to protect sensitive customer information from potential cyber-attacks and fraud. To achieve this, the bank requires an efficient bank management system that can keep track of customer accounts, balances, deposits, transactions, and merchant details. The bank's primary objective is to provide real-time transaction processing for its customers. As the bank is new, it needs to create a database from scratch that is optimized and tailored to its needs. Databases are a crucial element in the banking sector for several reasons.

Firstly, banks deal with vast amounts of financial data such as customer information, account balances, and transactions. Databases provide a secure and organised way to manage and store this information. Secondly, real-time access to information is critical in the banking sector, and databases enable customers to access their account information at any time while allowing banks to retrieve customer data to process transactions quickly. Thirdly, banks use databases to manage risk by storing and analysing data related to credit and loan portfolios, fraud detection, and compliance with regulations. Fourthly, databases are used to comply with strict regulatory requirements related to data management, privacy, and security. Lastly, databases help banks to improve customer relationships by enabling them to track customer interactions and behaviour, allowing for personalised services and products that meet the unique needs of their customers.

The Swift Bank Management project aims to develop an optimised database system that will enable the bank to operate efficiently and provide better customer service. The goal of this project is to create a banking database that can guarantee consistent data, get rid of redundant data, and offer fast query processing and high-performance applications for the sector. It will meet the bank's needs effectively and help the bank achieve its objective of providing real-time transaction processing for its customers.

Functionality:

The Swift bank Database will offer information about customer profiles, and account details including the type of accounts (Savings, Checking), transaction history, and merchant details in case a dispute needs to be solved. The end-user will be able to query the database to obtain information about their account, transaction history, and merchant details and obtain prompt responses from this efficient, user-friendly database, efficient in handling large volumes of financial data which is designed to be easily maintained.

Entities:

1. Customer (has a unique identification number, i.e CustomerID)
2. Login (has a unique Username, i.e LoginUsername)
3. Account (AccountID)
4. Savings (SAccountID)
5. Checking (CAccountID)
6. Transactions (TransactionID)

7. Merchant (MerchantID)

Relationship between Entities:

Below are the relationships between above entities that one can infer from the rules defined:

Users ⇄ Customer
Customer ⇄ Account
Account ⇄ Account Type Savings
Account ⇄ Account Type Checking
Account Type Checking ⇄ Transactions
Transactions ⇄ Merchant

Cardinalities of relationships among entities:

Login (Optional One) ⇄ Customer (Mandatory One)
Customer (Mandatory One) ⇄ Account (Mandatory Many)
Account (Total Specialization). ⇄ Savings (Disjoint Constraint)
Account (Total Specialization). ⇄ Checking (Disjoint Constraint)
Checking (Mandatory One) ⇄ Transactions (Optional Many)
Transactions (Optional Many) ⇄ Merchant (Mandatory One)

Attributes of all entities:

Below are mandatory attributes deduced from business rules which are roughly grouped together by domain understanding and might get refined moving forward.

- CustomerID
- C_Name
- C_Address
- C_Phone
- C_Email
- C_JoiningDate

- AccountID
- AccountBalance
- AccountName
- RoutingNumber
- OpeningDate
- AccountType

- TransactionID
- TransactionDate
- TransactionAmount
- TransactionStatus

- MerchantID
- M_Name
- M_Address
- M_Phone
- M_Email

- LoginUsername
- Password

- SAccountID
- InterestRate

- CAccountID
- ATMWithdrawalCAP
- DebitCardNumber
- PIN

ER DIAGRAM:

For designing the ER Diagram, we started with segregation of each entity and its attributes. Below is our outcome and ER Diagram for the overall use case.

Entities:

Customer	
PK	CustomerID
	C_Name
	C_Address
	C_Phone
	C_Email
	C_JoiningDate

Account	
PK	AccountID
FK	CustomerID
	AccountName
	AccountBalance
	RoutingNumber
	OpeningDate
	AccountType

Transactions	
PK	TransactionID
	TransactionDate
	TransactionAmount
FK1	CAccountID
FK2	MerchantID
	TransactionStatus

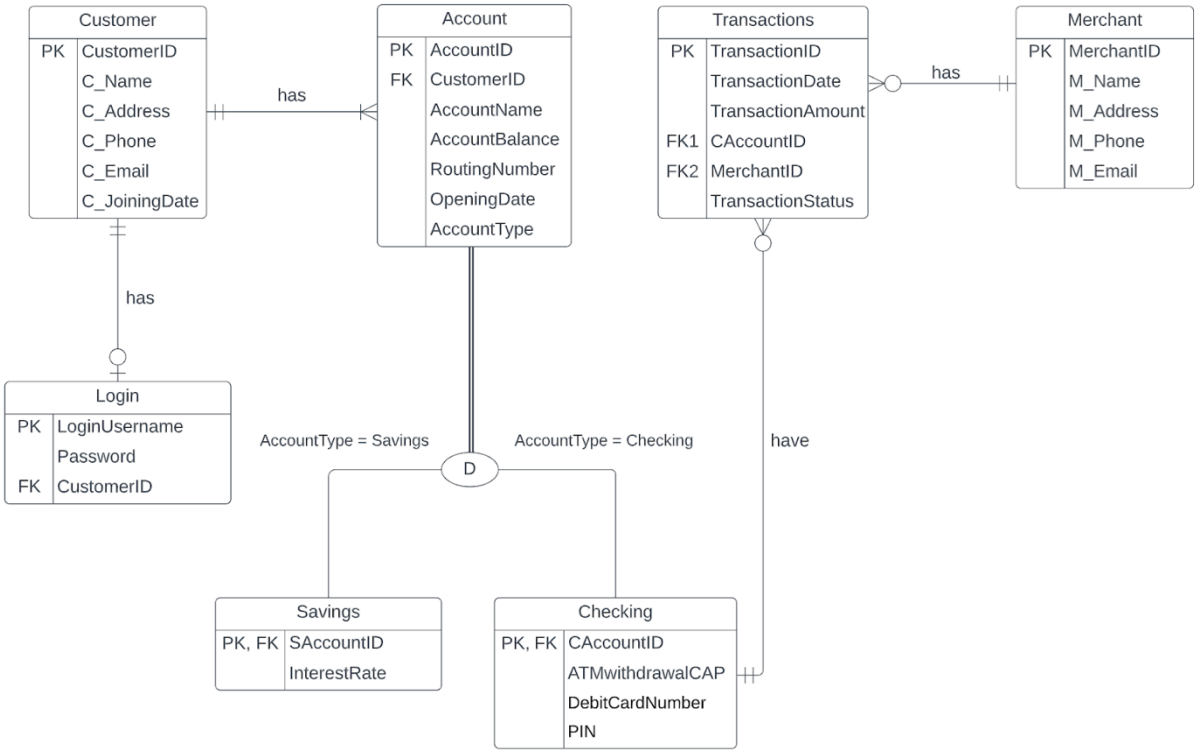
Merchant	
PK	MerchantID
	M_Name
	M_Address
	M_Phone
	M_Email

Login	
PK	LoginUsername
	Password
FK	CustomerID

Savings	
PK, FK	SAccountID
	InterestRate

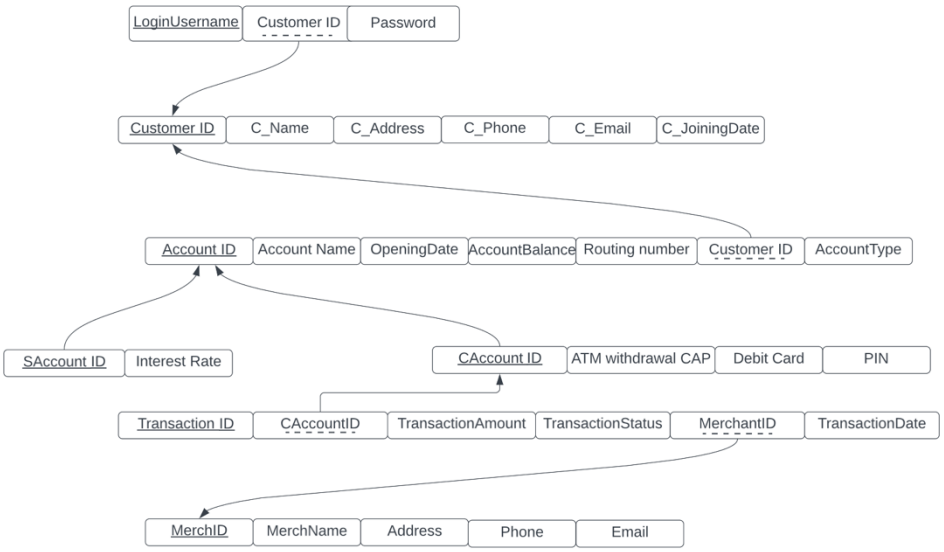
Checking	
PK, FK	CAccountID
	ATMwithdrawalCAP
	DebitCardNumber
	PIN

Diagram:



ER DIAGRAM TO RELATIONAL SCHEMA:

This step is a process of transforming a conceptual schema of the application domain into a schema for the data model underlying a particular DBMS, such as the relational or object-oriented data model. An ERD can be used to generate a relational schema by identifying the entities and their attributes and relationships, which can then be mapped to tables, columns, and foreign keys in a relational database. Each entity in the ERD becomes a table in the relational schema, and the attributes become the columns in the table. In summary, an ERD is a visual representation of a database structure, while a relational schema is a set of tables that represent the database structure.

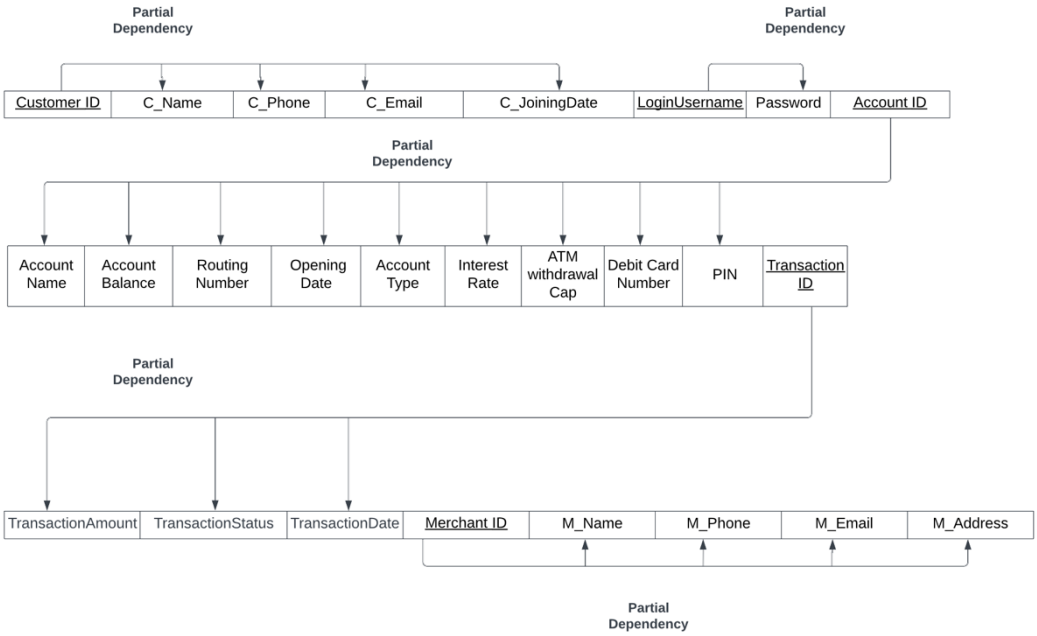


DATA NORMALIZATION:

Data normalization is the process of organizing data in a database in such a way that it reduces redundancy and dependency. Entities are well-structured and their relationships are clearly defined because of normalization. It guarantees that each item of data is stored in the database just once and helps prevent data duplication. Inconsistencies or inaccuracies in the data are less likely as a result, making it simpler to update and manage the data. Normalization is typically achieved through a series of steps The most used normal forms are first normal form (1NF), second normal form (2NF), and third normal form (3NF).

Normalization: 1st Normal Form (1NF)

First normal form (1NF) is a set of requirements for a table or relation in a database. It requires each attribute to contain atomic values that cannot be further decomposed. Each column must have a unique name, and each row must be unique and identifiable through a primary key. The order of rows and columns is not significant. Meeting these requirements helps reduce redundancy, simplify data management, and make it easier to manipulate and query the data in the database.

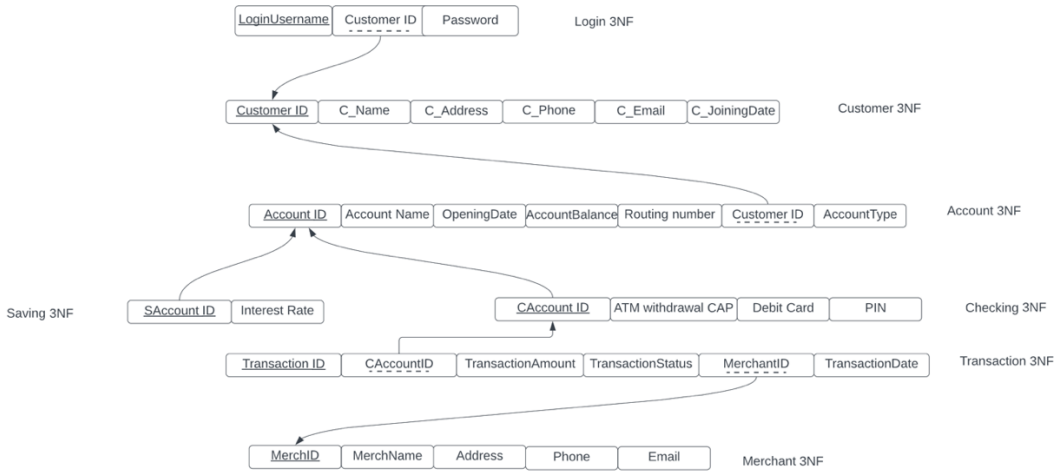


Normalization: 2nd Normal Form (2NF)

Our table doesn't have any transitive dependencies, it is already in 2NF as it meets the requirements of 1NF. This means that each non-key attribute in the table is fully dependent on the primary key, not just a part of it. Ensuring that a table is in 2NF helps reduce data redundancy and ensures that tables are well-structured and normalized.

Normalization: 3rd Normal Form (3 NF)

Third Normal Form (3NF) is a database requirement that eliminates transitive dependencies, ensuring that all non-key attributes depend only on the primary key. This reduces redundancy and improves data consistency, making it easier to manage and manipulate data. To meet the requirements of 3NF, a table should be **split into** multiple tables so that each table has no transitive dependencies, which reduces redundancy and improves data consistency.



Summary Table for each entity:

Customer Table Customer (CustomerID , C_Name , C_Address, C_Phone, C_Email, C_JoiningDate) **Datatype:** INT, VARCHAR(20), DATE respectively. **Additional Details:** All fields are required.	**Login Table** Login (LoginUsername , Password, CustomerID) **Datatype:** INT, VARCHAR(20) respectively. **Additional Details:** All fields are required. CustomerID is FK.
Account Table Account (AccountID, CustomerID , AccountName, AccountBalance, RoutingNumber, AccountNumber, OpeningDate, AccountType) **Datatype:** INT, VARCHAR(20), VARCHAR(40) ,DATE respectively. **Additional Details:** All fields are required. CustomerID is FK. AccountName can have values = 'Swift Savings Account', 'Swift Checking Account'. AccountType can have values = 'Savings', 'Checking'.	**Transactions Table** Transactions (TransactionID, TransactionTime, TransactionAmount, AccountID ,MerchantID, TransactionStatus) **Datatype:** INT, VARCHAR(20), VARCHAR(50) , TIMESTAMP , DECIMAL respectively. **Additional Details:** All fields are required. CAccountID , MerchantID are FK. TransactionStatus can have values = 'Cancelled', 'Successful', 'Disputed', 'Disputed then Resolved', 'Declined', 'Pending'.
Checking Table Checking (CAccountID, ATMwithdrawalCAP, DebitCardNumber, PIN) **Datatype:** INT. **Additional Details:** CAccountID is required, other fields are optional for accounts without Debit Card. CAccountID is FK.	**Merchant Table** Merchant (MerchantID, M_Name , M_Phone, M_Address, M_Phone, M_Email) **Datatype:** INT, VARCHAR(20) respectively. **Additional Details:** All fields are required.
Savings Table Savings (SAccountID, InterestRate) **Datatype:** INT, DECIMAL respectively. **Additional Details:** All fields are required. SAccountID is FK.	

Creation Of Tables:

Customer Table

Customer (CustomerID , C_Name , C_Address, C_Phone, C_Email, C_JoiningDate)

Datatype: INT, VARCHAR(20), DATE respectively.

Additional Details: All fields are required.

```
CREATE TABLE Customer (
  CustomerID INT NOT NULL,
  C_Name VARCHAR(20) NOT NULL,
  C_Address VARCHAR(20) NOT NULL,
  C_Phone INT NOT NULL,
  C_Email VARCHAR(20) NOT NULL,
  C_JoiningDate DATE NOT NULL,
  CONSTRAINT Customer_PK PRIMARY KEY(CustomerID)

);
```

Login Table

Login (LoginUsername , Password, CustomerID)

Datatype: INT, VARCHAR(20) respectively.

Additional Details: All fields are required. CustomerID is FK.

```
CREATE TABLE Login (
  LoginUsername VARCHAR(20) NOT NULL,
  Password VARCHAR(20) NOT NULL,
  CustomerID INT NOT NULL UNIQUE,
  CONSTRAINT Login_PK PRIMARY KEY(LoginUsername),
  CONSTRAINT Login_FK FOREIGN KEY(CustomerID)
  REFERENCES Customer(CustomerID)
);
```

Account Table

Account (AccountID, CustomerID , AccountName, AccountBalance, RoutingNumber, AccountNumber, OpeningDate, AccountType)

Datatype: INT, VARCHAR(20), VARCHAR(40) ,DATE respectively.

Additional Details: All fields are required. CustomerID is FK. AccountName can have values = 'Swift Savings Account', 'Swift Checking Account'. AccountType can have values = 'Savings', 'Checking'.

```
CREATE TABLE Account (
  AccountID INT NOT NULL,
  CustomerID INT NOT NULL,
  AccountName VARCHAR(40) NOT NULL
  CHECK (AccountName IN ('Swift Savings Account', 'Swift
Checking Account')),
  AccountBalance INT NOT NULL,
  RoutingNumber INT NOT NULL,
  OpeningDate DATE NOT NULL,
  AccountType VARCHAR(20) NOT NULL CHECK
(AccountType IN ('Savings', 'Checking')),
  CONSTRAINT Account_PK PRIMARY KEY(AccountID),
  CONSTRAINT Account_FK FOREIGN KEY (CustomerID)
  REFERENCES Customer(CustomerID)
);
```

Transactions Table

Transactions (TransactionID, TransactionTime, TransactionAmount, AccountID ,MerchantID, TransactionStatus)

Datatype: INT, VARCHAR(20), VARCHAR(50) , TIMESTAMP , DECIMAL respectively.

Additional Details: All fields are required. CAccountID , MerchantID are FK. TransactionStatus can have values = 'Cancelled', 'Successful', 'Disputed', 'Disputed then Resolved', 'Declined', 'Pending'.

```
CREATE TABLE Transactions(
  TransactionID VARCHAR(50) NOT NULL,
  TransactionTime TIMESTAMP NOT NULL,
  TransactionAmount DECIMAL NOT NULL,
  AccountID INT NOT NULL,
  MerchantID INT NOT NULL,
  TransactionStatus VARCHAR(50) NOT NULL
  CHECK (TransactionStatus IN ('Cancelled', 'Successful',
'Disputed', 'Disputed then Resolved', 'Declined', 'Pending')),
  CONSTRAINT Transactions_PK PRIMARY KEY(TransactionID),
  CONSTRAINT Transactions_FK1 FOREIGN KEY(AccountID)
REFERENCES Checking(CAccountID),
  CONSTRAINT Transactions_FK2 FOREIGN KEY(MerchantID)
REFERENCES Merchant(MerchantID)
);
```

Checking Table
Checking (CAccountID, ATMwithdrawalCAP, DebitCardNumber, PIN)
Datatype: INT.
Additional Details: CAccountID is required, other fields are optional for accounts without Debit Card. CAccountID is FK.

```
CREATE TABLE Checking (
  CAccountID INT NOT NULL,
  ATMwithdrawalCAP INT,
  DebitCardNumber INT,
  PIN INT,
  CONSTRAINT Checking_PK PRIMARY KEY(CAccountID),
  CONSTRAINT Checking_FK FOREIGN KEY(CAccountID)
  REFERENCES Account(AccountID)
);
```

Merchant Table
Merchant (MerchantID, M_Name , M_Phone, M_Address, M_Phone, M_Email)
Datatype: INT, VARCHAR(20) respectively.
Additional Details: All fields are required.

```
CREATE TABLE Merchant (
  MerchantID INT NOT NULL,
  M_Name VARCHAR(20) NOT NULL,
  M_Address VARCHAR(20) NOT NULL,
  M_Phone INT NOT NULL,
  M_Email VARCHAR(20) NOT NULL,
  CONSTRAINT Merchant_PK PRIMARY KEY(MerchantID)
);
```

Savings Table
Savings (SAccountID, InterestRate)
Datatype: INT, DECIMAL respectively.
Additional Details: All fields are required. SAccountID is FK.

```
CREATE TABLE Savings (
  SAccountID INT NOT NULL,
  InterestRate DECIMAL NOT NULL,
  CONSTRAINT Savings_PK PRIMARY KEY(SAccountID),
  CONSTRAINT Savings_FK FOREIGN KEY(SAccountID)
  REFERENCES Account(AccountID)
);
```

Insertion of Data in Tables:

```
/*************************Customer Table**********************************/
INSERT INTO Customer (CustomerID, C_Name, C_Address, C_Phone, C_Email, C_JoiningDate)
VALUES ('1', 'Tony Stark', 'Malibu', '5551234', 'tony@marvel.com',  TO_DATE('2014-02-10', 'YYYY-MM-DD'));
INSERT INTO Customer (CustomerID, C_Name, C_Address, C_Phone, C_Email, C_JoiningDate)
VALUES ('2', 'Michael Scofield', 'Fox River', '5555678', 'm.scofield@fxr.com',  TO_DATE('2014-02-17', 'YYYY-MM-DD'));
INSERT INTO Customer (CustomerID, C_Name, C_Address, C_Phone, C_Email, C_JoiningDate)
VALUES ('3', 'Walter White', 'Alburquerque', '5559012', 'ww@brb.com', TO_DATE('2015-03-10', 'YYYY-MM-DD'));
INSERT INTO Customer (CustomerID, C_Name, C_Address, C_Phone, C_Email, C_JoiningDate)
VALUES ('4', 'Bruce Wayne', 'Gotham', '5553456', 'bruce@wayne.com',  TO_DATE('2016-02-07', 'YYYY-MM-DD'));
INSERT INTO Customer(CustomerID, C_Name, C_Address, C_Phone, C_Email, C_JoiningDate)
VALUES ('5', 'Thomas Shelby', 'Birmingham', '8576352672', 'shelby@netflix.com', TO_DATE('2017-02-10', 'YYYY-MM-DD'));
INSERT INTO Customer(CustomerID, C_Name, C_Address, C_Phone, C_Email, C_JoiningDate)
VALUES ('6', 'Joe Goldberg', 'New York', '35267752', 'joe@you.com', TO_DATE('2018-04-10', 'YYYY-MM-DD'));
```

```
INSERT INTO Customer(CustomerID, C_Name, C_Address, C_Phone, C_Email, C_JoiningDate)
VALUES ('7', 'Jessie Pinkman', 'Alburquerque', '5559013', 'jpkmn@brb.com', TO_DATE('2018-10-
10', 'YYYY-MM-DD'));

/*****************************Users Table*****************************/
INSERT INTO Users (LoginUsername, Password, CustomerID)
VALUES ('tony_stark', 'ironman', '1');
INSERT INTO Users (LoginUsername, Password, CustomerID)
VALUES ('m_scofield', 'foxriver123', '2');
INSERT INTO Users (LoginUsername, Password, CustomerID)
VALUES ('wwhite', 'heisenberg', '3');
INSERT INTO Users (LoginUsername, Password, CustomerID)
VALUES ('bruce_wayne', 'iambatman', '4');
INSERT INTO Users (LoginUsername, Password, CustomerID)
VALUES ('t_shelby', 'peakyblinders', '5');
INSERT INTO Users (LoginUsername, Password, CustomerID)
VALUES ('j_goldberg', 'love', '6');
INSERT INTO Users (LoginUsername, Password, CustomerID)
VALUES ('jpinkman', 'yoyoyo', '7');

/*********************Account Table*****************************/
INSERT INTO Account (AccountID, CustomerID, AccountName, AccountBalance, RoutingNumber,
OpeningDate, AccountType)
VALUES ('10001', '1','Swift Checking Account', '2000000', '121000248', TO_DATE('2014-02-10', 'YYYY-
MM-DD'), 'Checking');
INSERT INTO Account (AccountID, CustomerID, AccountName, AccountBalance, RoutingNumber,
OpeningDate, AccountType)
VALUES ('10002', '2','Swift Savings Account', '5000', '121000248', TO_DATE('2014-02-17', 'YYYY-MM-
DD'), 'Savings');
INSERT INTO Account (AccountID, CustomerID, AccountName, AccountBalance, RoutingNumber,
OpeningDate, AccountType)
VALUES ('10003', '3', 'Swift Checking Account', '100000', '121000248', TO_DATE('2015-03-10', 'YYYY-
MM-DD'), 'Checking');
INSERT INTO Account (AccountID, CustomerID, AccountName, AccountBalance, RoutingNumber,
OpeningDate, AccountType)
VALUES ('10004', '4', 'Swift Checking Account', '15000000', '121000248', TO_DATE('2016-02-07', 'YYYY-
MM-DD'), 'Checking');
INSERT INTO Account (AccountID, CustomerID, AccountName, AccountBalance, RoutingNumber,
OpeningDate, AccountType)
VALUES ('10005', '1','Swift Savings Account', '70000', '121000248', TO_DATE('2018-02-10', 'YYYY-MM-
DD'), 'Savings');
INSERT INTO Account (AccountID, CustomerID, AccountName, AccountBalance, RoutingNumber,
OpeningDate, AccountType)
VALUES ('10006', '4', 'Swift Savings Account', '7000000', '121000248', TO_DATE('2019-02-10', 'YYYY-
MM-DD'), 'Savings');
INSERT INTO Account (AccountID, CustomerID, AccountName, AccountBalance, RoutingNumber,
OpeningDate, AccountType)
VALUES ('10007', '5','Swift Checking Account', '800000', '121000248', TO_DATE('2016-02-07', 'YYYY-
MM-DD'), 'Checking');
INSERT INTO Account (AccountID, CustomerID, AccountName, AccountBalance, RoutingNumber,
OpeningDate, AccountType)
VALUES ('10008', '6','Swift Checking Account', '90000', '121000248', TO_DATE('2016-02-07', 'YYYY-MM-
DD'), 'Checking');
```

```
INSERT INTO Account (AccountID, CustomerID, AccountName, AccountBalance,
RoutingNumber, OpeningDate, AccountType)
VALUES ('10009', '7','Swift Checking Account', '0', '121000248', TO_DATE('2016-02-07', 'YYYY-MM-DD'),
'Checking');

/*************************Savings Table*********************************/
INSERT INTO Savings (SAccountID, InterestRate)
VALUES ('10002', '2.5');
INSERT INTO Savings (SAccountID, InterestRate)
VALUES ('10005', '3');
INSERT INTO Savings (SAccountID, InterestRate)
VALUES ('10006', '4.25');

/*************************Checking Table*********************************/
INSERT INTO Checking (CAccountID, ATMwithdrawalCAP, DebitCardNumber, PIN)
VALUES ('10001', '10000', '1616101021287800', '1234');
INSERT INTO Checking (CAccountID) VALUES ('10003');
INSERT INTO Checking (CAccountID, ATMwithdrawalCAP, DebitCardNumber, PIN)
VALUES ('10004', '20000', '1186070241234567', '9012');
INSERT INTO Checking (CAccountID, ATMwithdrawalCAP, DebitCardNumber, PIN)
VALUES ('10007', '2500', '6011111111111117', '3456');
INSERT INTO Checking (CAccountID, ATMwithdrawalCAP, DebitCardNumber, PIN)
VALUES ('10008', '2500', '371449635398431', '0912');
INSERT INTO Checking (CAccountID) VALUES ('10009');

/************************Merchant Table*************************/
INSERT INTO Merchant (MerchantID, M_Name, M_Address, M_Phone, M_Email)
VALUES ('1', 'Amazon', 'Seattle', '5555678', 'help@amazon.com');
INSERT INTO Merchant (MerchantID, M_Name, M_Address, M_Phone, M_Email)
VALUES ('2', 'Target', 'Minneapolis', '5555679', 'help@target.com');
INSERT INTO Merchant (MerchantID, M_Name, M_Address, M_Phone, M_Email)
VALUES ('3', 'Best Buy', 'Richfield', '5555680', 'help@bestbuy.com');
INSERT INTO Merchant (MerchantID, M_Name, M_Address, M_Phone, M_Email)
VALUES ('4', 'Walmart', 'Bentonville', '5555681', 'help@walmart.com');
INSERT INTO Merchant (MerchantID, M_Name, M_Address, M_Phone, M_Email)
VALUES ('5', 'Costco', 'Issaquah', '5555682', 'help@costco.com');

/************************Transactions Table*************************/
INSERT INTO Transactions (TransactionID, TransactionTime, TransactionAmount, AccountID,
MerchantID, TransactionStatus)
VALUES ('8B5F7E25-63D5-417B', TO_DATE('01-01-2018 02:30:00', 'DD-MM-YYYY HH24:MI:SS'),
'500.99', '10001', '1', 'Successful');
INSERT INTO Transactions (TransactionID, TransactionTime, TransactionAmount, AccountID,
MerchantID, TransactionStatus)
VALUES ('9D685E72-FB1F-41EA', TO_DATE('08-11-2019 02:30:00', 'DD-MM-YYYY HH24:MI:SS'),
'800.59', '10007', '2', 'Declined');
INSERT INTO Transactions (TransactionID, TransactionTime, TransactionAmount, AccountID,
MerchantID, TransactionStatus)
VALUES ('C11E6DF9-6B2C-420B', TO_DATE('11-01-2020 02:30:00', 'DD-MM-YYYY HH24:MI:SS'), '1200',
'10004', '3', 'Declined');
INSERT INTO Transactions (TransactionID, TransactionTime, TransactionAmount, AccountID,
MerchantID, TransactionStatus)
```

VALUES ('1A3D9FEF-3D8F-4EEA', TO_DATE('12-07-2021 02:30:00', 'DD-MM-YYYY HH24:MI:SS'), '1999.80', '10008', '4', 'Successful');
INSERT INTO Transactions (TransactionID, TransactionTime, TransactionAmount, AccountID, MerchantID, TransactionStatus)
VALUES ('683D9FEF-3D8F-4EEA', TO_DATE('03-01-2022 02:30:00', 'DD-MM-YYYY HH24:MI:SS'), '5000.60', '10001', '5', 'Disputed then Resolved');
INSERT INTO Transactions (TransactionID, TransactionTime, TransactionAmount, AccountID, MerchantID, TransactionStatus)
VALUES ('2A9A285E-926C-47D4', TO_DATE('12-04-2022 02:30:00', 'DD-MM-YYYY HH24:MI:SS'), '200.75', '10007', '1', 'Disputed then Resolved');

/************************INSERTION OF DATA COMPLETED**********************************/

Data Loaded in DB:

```
select * from Customer;
select * from Login;
select * from Account;
select * from Savings;
select * from Checking;
select * from Merchant;
select * from Transactions;
```

CustomerID	C_Name	C_Address	C_Phone	C_Email	C_JoiningDate
1	Tony Stark	Malibu	5551234	tony@marvel.com	2014-02-10
2	Michael Scofield	Fox River	5555678	m.scofield@fxr.com	2014-02-17
3	Walter White	Alburquerque	5559012	ww@brb.com	2015-03-10
4	Bruce Wayne	Gotham	5553456	bruce@wayne.com	2016-02-07
5	Thomas Shelby	Birmingham	8576352672	shelby@netflix.com	2017-02-10
6	Joe Goldberg	New York	35267752	joe@you.com	2018-04-10
7	Jessie Pinkman	Alburquerque	5559013	jpkmn@brb.com	2018-10-10

LoginUsername	Password	CustomerID
tony_stark	ironman	1
m_scofield	foxriver123	2
wwhite	heisenberg	3
bruce_wayne	iambatman	4
t_shelby	peakyblinders	5
j_goldberg	love	6
jpinkman	yoyoyo	7

AccountID	CustomerID	AccountName	AccountBalance	RoutingNumber	OpeningDate	AccountType
10001	1	Swift Checking Account	2000000	121000248	2014-02-10	Checking
10002	2	Swift Savings Account	5000	121000248	2014-02-17	Savings
10003	3	Swift Checking Account	100000	121000248	2015-03-10	Checking
10004	4	Swift Checking Account	15000000	121000248	2016-02-07	Checking
10005	1	Swift Savings Account	70000	121000248	2018-02-10	Savings
10006	4	Swift Savings Account	7000000	121000248	2019-02-10	Savings
10007	5	Swift Checking Account	800000	121000248	2016-02-07	Checking
10008	6	Swift Checking Account	90000	121000248	2016-02-07	Checking
10009	7	Swift Checking Account	0	121000248	2016-02-07	Checking

SAccountID	InterestRate
10002	2.5
10005	3
10006	4.25

CAccountID	ATMwithdrawalCAP	DebitCardNumber	PIN
10001	10000	1616101021287800	1234
10003	NULL	NULL	NULL
10004	20000	1186070241234567	9012
10007	2500	6011111111111117	3456
10008	2500	371449635398431	912
10009	NULL	NULL	NULL

MerchantID	M_Name	M_Address	M_Phone	M_Email
1	Amazon	Seattle	5555678	help@amazon.com
2	Target	Minneapolis	5555679	help@target.com
3	Best Buy	Richfield	5555680	help@bestbuy.com
4	Walmart	Bentonville	5555681	help@walmart.com
5	Costco	Issaquah	5555682	help@costco.com

TransactionID	TransactionTime	TransactionAmount	AccountID	MerchantID	TransactionStatus
8B5F7E25-63D5-417B	01-01-2018 02:30:00	500.99	10001	1	Successful
9D685E72-FB1F-41EA	08-11-2019 02:30:00	800.59	10007	2	Declined
C11E6DF9-6B2C-420B	11-01-2020 02:30:00	1200	10004	3	Declined
1A3D9FEF-3D8F-4EEA	12-07-2021 02:30:00	1999.8	10008	4	Successful
683D9FEF-3D8F-4EEA	03-01-2022 02:30:00	5000.6	10001	5	Disputed then Resolved
2A9A285E-926C-47D4	12-04-2022 02:30:00	200.75	10007	1	Disputed then Resolved

Queries:

- **What if you want to retrieve the names and joining dates of all customers who joined after February 7th, 2016? Show Customer ID, Name and Joining date.**

SELECT CustomerID, C_Name, C_JoiningDate FROM Customer WHERE C_JoiningDate > TO_DATE('2016-02-07', 'YYYY-MM-DD');

```
SELECT CustomerID, C_Name, C_JoiningDate FROM Customer WHERE C_JoiningDate > TO_DATE('2016-02-07', 'YYYY-MM-DD');

SELECT AccountID, AccountType, AccountBalance  FROM Account WHERE AccountBalance > 5000;

SELECT M_Name, TransactionAmount FROM Merchant JOIN Transactions ON Merchant.MerchantID = Transactions.MerchantID WHERE TransactionStatus = 'Successful';

SELECT C.CustomerID, C.C_Name, A.AccountType, A.AccountBalance  FROM Customer C JOIN Account A ON C.CustomerID = A.CustomerID WHERE A.AccountBalance  > 10000;

SELECT Customer.CustomerID, C_Name, C_Email, SUM(AccountBalance ) as Total_AccountBalance  FROM Customer JOIN Account ON Customer.CustomerID = Account.CustomerID
GROUP BY Customer.CustomerID, C_Name, C_Email;

SELECT C.C_Name, A.AccountID, A.AccountName FROM Customer C JOIN Account A ON C.CustomerID = A.CustomerID WHERE A.AccountType = 'Savings';

SELECT T.TransactionID, T.TransactionTime, T.TransactionAmount, T.MerchantID, M.M_Name FROM Transactions T JOIN Merchant M ON T.MerchantID = M.MerchantID;
```

CustomerID	C_Name	C_JoiningDate
5	Thomas Shelby	2017-02-10
6	Joe Goldberg	2018-04-10
7	Jessie Pinkman	2018-10-10

- **What if you want to retrieve the account IDs, account types, and balances of all accounts with a balance greater than 5000? Show the account IDs, account types, and balances**
  SELECT AccountID, AccountType, AccountBalance FROM Account WHERE Balance > 5000;

AccountID	AccountType	AccountBalance
10001	Checking	2000000
10003	Checking	100000
10004	Checking	15000000
10005	Savings	70000
10006	Savings	7000000
10007	Checking	800000
10008	Checking	90000

- **What if you want to retrieve the merchant names and transaction amounts for all transactions with a 'Successful' status? Show the merchant names and transaction amounts**
  SELECT M_Name, TransactionAmount FROM Merchant JOIN Transactions ON Merchant.MerchantID = Transactions.MerchantID WHERE TransactionStatus = 'Successful';

M_Name	TransactionAmount
Amazon	500.99
Walmart	1999.8

- **What if you want to retrieve the customer ID, name, balance, and account type for all accounts that have a balance greater than 10000? Show the customer ID, name, balance, and account type**
  SELECT C.CustomerID, C.C_Name, A.AccountType, A.AccountBalance FROM Customer C JOIN Account A ON C.CustomerID = A.CustomerID WHERE A.AccountBalance > 10000;

CustomerID	C_Name	AccountType	AccountBalance
1	Tony Stark	Checking	2000000
3	Walter White	Checking	100000
4	Bruce Wayne	Checking	15000000
1	Tony Stark	Savings	70000
4	Bruce Wayne	Savings	7000000
5	Thomas Shelby	Checking	800000
6	Joe Goldberg	Checking	90000

- **What if you want to retrieve the customer ID, name, email, and total balance for all customers? Show the customer ID, name, email, and total balance**
  SELECT Customer.CustomerID, C_Name, C_Email, SUM(AccountBalance) as Total_AccountBalance FROM Customer JOIN Account ON Customer.CustomerID = Account.CustomerID GROUP BY Customer.CustomerID, C_Name, C_Email;

CustomerID	C_Name	C_Email	Total_AccountBalance
1	Tony Stark	tony@marvel.com	2070000
2	Michael Scofield	m.scofield@fxr.com	5000
3	Walter White	ww@brb.com	100000
4	Bruce Wayne	bruce@wayne.com	22000000
5	Thomas Shelby	shelby@netflix.com	800000
6	Joe Goldberg	joe@you.com	90000
7	Jessie Pinkman	jpkmn@brb.com	0

- **What if you want to retrieve the customer names, account IDs, and account types for all customers who have a 'Savings' account? Show the customer name, account ID, and account Name**
  SELECT C.C_Name, A.AccountID, A.AccountName FROM Customer C JOIN Account A ON C.CustomerID = A.CustomerID WHERE A.AccountType = 'Savings';

C_Name	AccountID	AccountName
Michael Scofield	10002	Swift Savings Account
Tony Stark	10005	Swift Savings Account
Bruce Wayne	10006	Swift Savings Account

- **What if you want to retrieve the Transaction ID, time, amount, and Merchant ID and Name for transactions that have a status of "Disputed then Resolved"? Show the Transaction ID, time, amount, and Merchant ID and Name**
  SELECT T.TransactionID, T.TransactionTime, T.TransactionAmount, T.MerchantID, M.M_Name FROM Transactions T JOIN Merchant M ON T.MerchantID = M.MerchantID;

TransactionID	TransactionTime	TransactionAmount	MerchantID	M_Name
8B5F7E25-63D5-417B	01-01-2018 02:30:00	500.99	1	Amazon
9D685E72-FB1F-41EA	08-11-2019 02:30:00	800.59	2	Target
C11E6DF9-6B2C-420B	11-01-2020 02:30:00	1200	3	Best Buy
1A3D9FEF-3D8F-4EEA	12-07-2021 02:30:00	1999.8	4	Walmart
683D9FEF-3D8F-4EEA	03-01-2022 02:30:00	5000.6	5	Costco
2A9A285E-926C-47D4	12-04-2022 02:30:00	200.75	1	Amazon

Learnings:

The creation of a bank management system database for Swift Bank was an interesting project that provided several takeaways. Ultimately, this project demonstrated the importance of thorough planning and design when creating a database, as well as the necessity of striking a balance between competing priorities including data security, performance, and ease of use.

Some of the key learnings are:
The first step in developing any database is to gather requirements from the stakeholders. In this project, we saw how the entrepreneurs had a clear idea of what they wanted the database to do, and how important it was to capture their requirements accurately and we performed ER modelling to visualize the entities, relationships and attributes that make up the database. We used Entity Relationship (ER) modelling to design the schema for the Swift Bank database. We also ensured that the data in the database was normalized up to the third normal form (3NF), which helped to reduce data redundancy and improve data integrity. It was a very crucial step to ensure that the database is well-structured and avoids redundancy.

We saw how the entrepreneurs emphasized the need for data security, especially for storing login usernames and passwords. We ensured that this sensitive information was stored securely and not accessible to unauthorized users. Database optimization was done as a part of this project to make sure the system performs well. In this project, we designed the database using SQL to ensure that queries could be executed quickly and efficiently, even as the data in the database grows.

Therefore, by working as a group on this project we have received hands-on experience in building a database for a real-world business scenario, creating tables and relationships, populating tables with data, and executing queries to retrieve data from the database.