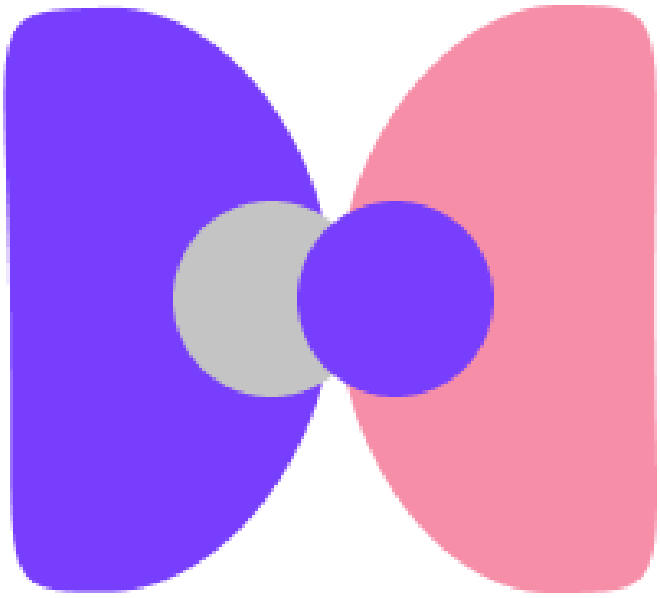

# DUKH BANK

**Dhruv Mishra (2020296)**
**Hunar Kaur (2020303)**
**Kshitij Bajaj (2020565)**
**Utkarsh Pal(2020144)**

## PROBLEM STATEMENT:

Banking sector is one of the most important sectors for the development of a nation. It is the way that people safekeep their funds or get the money they require for various business activities. The major problem with the current banking systems is their inefficient database management systems. Their current database management systems are unreliable, error prone and unsystematic. Hence, we thought of designing a database management system for banking systems of our own!

## SCOPE OF THE PROJECT:

In the 21st century, an efficient banking sector is the need of the hour. Banking system's primary goal is to provide financial services to people. It is a way to keep track of everything in the bank. One of the main aspects of the banking systems is the humongous amount of data they need to store. The banking sector's functionality will improve if the storing of this data is made more efficient and systematic. In this project we have tried to show the various aspects of a banking system's database management and cover the basic functionality of a Banking System. Our project is a database management system for financial institutions, banks. Our database stores the account details, employee details and bank details. It will also store a users' every transaction with the transaction details and details regarding the loans, fixed deposits, credit/debit card of the users. This database also enables the bank employees to efficiently access their customers' account details. It's not only useful for the customers but also for the bank employees and managers.

## ER FORMULATION

ER LINK:
https://miro.com/welcomeonboard/TXhFM0JMZ3BMRTlReXNBWkpaMWVhdnoyUGZiU2E5OWhPdXZlbTh4ajJ6bk1xdk54SlhqVlRUeTJ0UFh3dk5YYnwzNDU4NzY0NTE5Nzc3MjQ2Nzcx?invite_link_id=423189002209

## STAKEHOLDERS IDENTIFIED:

Following are the stakeholders:

1. **USER/CUSTOMER :**

   Users or customers are the main stakeholders of our application. They will use our banking system for the following:

- Depositing Money
- Withdrawing Money

- Balance Enquiry (The customer can check their balance)
- Changing Passwords/PIN (The customer can change the passwords and pin numbers)
- Apply for loan
- Apply for fixed deposit
- Money transfer (The user can transfer money to another account of the same bank

2. **BANK EMPLOYEES:**

   These stakeholders work for our application. They will be responsible for creating ,managing and closing of user's accounts. Each branch hires some amount of employees. The bank employees can view the list of all the account holders.

3. **SHAREHOLDERS OF BANKS:**

   These stakeholders are the owners of the bank. They are responsible for providing financial backing in return for potential profits over the lifetime of the bank.

4. **THE GOVERNMENT:**

   The government plays an important role in the financial system. They control the operations of the financial institutions through laws and regulations.They also set the regulations that decide what banks can and cannot do.Government guarantees the safekeeping of the bank deposits up to a certain amount so that the funds deposited with the banks don't reduce drastically.

5. **FINANCIAL MARKETS:**

   The financial markets permit banks to borrow money and help them provide loans to the users who wish to apply for a loan.

6. **BOARD OF DIRECTORS:**

   These stakeholders are responsible for safeguarding the interests of bank's depositors, bank's creditors, and shareholders through lawful and ethical administration of the banking system.

## ENTITIES DEFINED AND ITS ATTRIBUTES:

- ★ **Employee**
  - EmpID
  - Name
    - First name
    - Last name
  - Salary
  - Date of Birth
  - Address
  - Gender
  - Phone Number
  - Email ID

- ★ **Account**
  - Account Number
  - Name
    - First name
    - Last name
  - Date of Birth
  - Gender
  - Phone Number
  - Email
  - Address
  - Balance
  - Account Type

- ★ **Transactions**
  - Transaction ID
  - Timestamp
  - Sender
  - Receiver
  - Mode of Payment
  - Amount

- ★ **Bank Branches**
  - Branch ID
  - Name of Branch
  - Address
  - Staff Size
  - Manager EmpID
  - Date of Opening

★ **Loans**
- Loan Type
- Date of Sanction
- Next Due Amount
- Tenure
- Amount
- Amount Paid
- Next Payment Date
- Interest date

★ **Debit Cards**
- Card Number
- Account Number
- Card Network
- Expiry Date
- Date of Issue
- CVV
- Account Balance

★ **Credit Cards**
- Card Number
- Account Number
- Card Network
- Expiry Date
- Date of Issue
- CVV
- Withdrawal Limit
- Used Limit

★ **Fixed Deposits**
- FD Number
- Principal Amount
- Current Amount
- Interest Rate
- Date of Opening
- Maturation Period

★ **Customer Login details**
- Account Number
- Hash of Password

★ **Employee Login details**
  - EmpID
  - Hash of Password

## RELATIONSHIPS ESTABLISHED:

- EMPLOYEE ***works in*** BRANCH.
  - Date of Joining
- EMPLOYEE ***creates/closes*** ACCOUNT.
- EMPLOYEE ***logs-in using*** EMPLOYEE LOGIN DETAILS
- ACCOUNT HOLDER ***creates*** FIXED DEPOSITS.
- ACCOUNT HOLDER ***logs-in using*** ACCOUNT LOGIN DETAILS.
- ACCOUNT HOLDER ***takes*** LOANS.
- ACCOUNT HOLDER ***uses*** CARDS(DEBIT/CREDIT) ***for making*** TRANSACTIONS.

## IDENTIFICATION OF WEAK ENTITY:

The weak entity identified is LOAN. LOAN is a weak entity as it doesn't have a primary key and hence it's rows can't be identified using its attributes alone. It uses Account number, which is the primary key of the entity ACCOUNT, in conjunction with its attribute LOAN TYPE, which acts as a discriminator, and together they form a primary key for the entity LOAN.

## ENTITIES PARTICIPATION TYPE:

- Participation of ACCOUNT in *creates/closes* relation is total.
- Participation of EMPLOYEE in *works in* relation is total.
- Participation of FIXED DEPOSITS in *creates FD* relation is total.
- Participation of LOANS in *takes* relation is total.
- Participation of CARDS in *used for making* relation is total.
- Participation of TRANSACTIONS in *used for making* relation is total.
- Participation of EMPLOYEE in *creates/closes* relation is partial.
- Participation of ACCOUNT in *takes* relation is partial.
- Participation of ACCOUNT in *used for making* relation is partial.
- Participation of BRANCH in *works in* relation is partial.
- Participation of ACCOUNT in *creates FD* relation is partial.

## RELATIONSHIP ROLES AND CONSTRAINTS:

- Every EMPLOYEE works in a single BRANCH and every branch has atleast 1 employee.
- EMPLOYEES **create/**Manage multiple account ACCOUNTS and every account is managed by a single employee.
- Every EMPLOYEE logs-in using a single EMPLOYEE LOGIN DETAIL
- ACCOUNT HOLDERS Can create any number of FIXED DEPOSITS and every fixed deposit must be created by a single account.
- Every ACCOUNT HOLDER logs-in using a single ACCOUNT LOGIN DETAIL.
- ACCOUNT HOLDER can take upto 5 different types of LOANS and every loan must be taken by a single account.
- ACCOUNT HOLDER can use upto 2 CARDS(each of DEBIT & CREDIT) for making TRANSACTIONS and every card must be used by a single ACCOUNT and every transaction should be made by a single card and a single account.

- one-to-many relationship between a Branch and an Employee.
- one-to-many relationship between an Account and Fixed deposit.
- one-to-many relationship between an Account and Card. (part of a ternary relationship)

- one-to-many relationship between an Account and Transaction.(part of a ternary relationship)
- one-to-many relationship between a Cards and Transaction. (part of a ternary relationship)
- one-to-many relationship between an Account and Loan.
- one-to-many relationship between an Employee and Account.
- one-to-one relationship between Employee and Employee login details.
- one-to-one relationship between Account and Customer login details.

## IDENTIFICATION OF TERNARY RELATIONSHIP:

The ternary relationship identified is USED FOR MAKING. Account holders use CARDS to make TRANSACTIONS from their ACCOUNTS. Since these three entities interact simultaneously and go hand-in-hand, they share a ternary relationship.

# Relational Schema

**Employee** [EmpID, First name, Last name, Salary, DOB, Designation, Address, Gender, {Phone Number} ,Email ID]

**Account** [Account Number, First name,Last name, DOB, Gender, {Phone Number}, Email, Address, Balance, Account Type]

**Transactions** [Transaction ID, Timestamp, Sender, Receiver, Mode of Payment, Amount]

**Bank Branches** [Branch ID, Name of Branch, Address, Staff Size, Manager EmpID, Date of Opening]

**Loans** [Loan Type, Account Number ,Date of Sanction, Next Due Amount, Tenure, Amount, Amount Paid, Next Payment Date, Interest date]

**Debit Cards** [Card Number, Account Number, Card Network, Expiry Date, Date of Issue, CVV, Account Balance]

**Credit Cards** [Card Number, Account Number, Card Network, Expiry Date, Date of Issue, CVV, Withdrawal Limit, Used Limit]

**Employee Login details** [EmpID, Hash of Password]

**Customer Login details** [Account No, Hash of Password]

**Fixed Deposits** [FD Number, Principal Amount, Current Amount, Interest Rate, Date of Opening, Maturation Period]

**Manages**[EmpID, Account Number]

***Works in**[EmpID,Branch ID,Date of Joining]

***Creates FD**[FD Number,Account No]

***** Updated schema based on suggestions/feedback given in the midterm evaluation: In the Mid-evaluation of the project we forgot to include two tables in our relational schema. We have now included the two tables in our relational schema and hence the schema is complete now.**

# Constraints in DDL

```
CREATE TABLE IF NOT EXISTS `dukh_bank`.`Accounts` (
      `Account Number` BIGINT(12) NOT NULL,
      `First Name` VARCHAR(45) NOT NULL,
      `Last Name` VARCHAR(45) NOT NULL,
      `Date Of Birth` DATE NOT NULL,
      `Phone Number` BIGINT(10) NOT NULL,
      `Email` VARCHAR(45) NOT NULL,
      `Gender` VARCHAR(45) NOT NULL,
      `Balance` FLOAT NOT NULL DEFAULT 0,
      `Account Type` VARCHAR(45) NOT NULL,
      `Address` VARCHAR(100) NOT NULL,
      PRIMARY KEY (`Account Number`),
      UNIQUE INDEX `Phone Number_UNIQUE` (`Phone Number` ASC) VISIBLE,
      UNIQUE INDEX `Email_UNIQUE` (`Email` ASC) VISIBLE,
      UNIQUE INDEX `Account Number_UNIQUE` (`Account Number` ASC) VISIBLE)
```

```
CREATE TABLE IF NOT EXISTS `dukh_bank`.`Employee` (
    `EmpID` INT NOT NULL AUTO_INCREMENT,
    `First Name` VARCHAR(45) NOT NULL,
    `Last Name` VARCHAR(45) NOT NULL,
    `Date Of Birth` DATE NOT NULL,
    `Salary` FLOAT NOT NULL,
    `Designation` VARCHAR(45) NOT NULL,
    `Address` VARCHAR(180) NOT NULL,
    `Gender` VARCHAR(45) NOT NULL,
    `Phone Number` BIGINT(10) NOT NULL,
    `Email Address` VARCHAR(45) NOT NULL,
    PRIMARY KEY (`EmpID`),
    UNIQUE INDEX `EmpID_UNIQUE` (`EmpID` ASC) VISIBLE,
    UNIQUE INDEX `Phone Number_UNIQUE` (`Phone Number` ASC) VISIBLE,
    UNIQUE INDEX `Email Address_UNIQUE` (`Email Address` ASC) VISIBLE)
```

```sql
CREATE TABLE IF NOT EXISTS `dukh_bank`.`Fixed Deposits` (
  `FD Number` INT NOT NULL AUTO_INCREMENT,
  `Principal Amount` FLOAT NOT NULL,
  `Current Amount` FLOAT NOT NULL,
  `Interest Rate` INT(2) NOT NULL,
  `Date of Opening` DATE NOT NULL,
  `Maturation Period` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`FD Number`))
```

```sql
CREATE TABLE IF NOT EXISTS `dukh_bank`.`Transactions` (
  `TransactionID` INT NOT NULL,
  `Amount` FLOAT NOT NULL,
  `Sender` BIGINT(12) NOT NULL,
  `Receiver` BIGINT(12) NOT NULL,
  `Mode Of Payment` VARCHAR(45) NOT NULL,
  `Timestamp` DATETIME(2) NOT NULL,
  PRIMARY KEY (`TransactionID`),
  UNIQUE INDEX `TransactionID_UNIQUE` (`TransactionID` ASC) VISIBLE)
```

```sql
CREATE TABLE IF NOT EXISTS `dukh_bank`.`Debit Card` (
  `Card Number` BIGINT(16) NOT NULL,
  `Card Network` VARCHAR(45) NOT NULL,
  `Expiry Date` DATE NOT NULL,
  `Date of Issue` DATE NOT NULL,
  `CVV` INT(3) NOT NULL,
  `Account Balance` FLOAT NOT NULL DEFAULT 0,
  `Account Number` BIGINT(12) NOT NULL,
  PRIMARY KEY (`Card Number`))
```

```
CREATE TABLE IF NOT EXISTS `dukh_bank`.`Credit Card` (
  `Card Number` BIGINT(16) NOT NULL,
  `Card Network` VARCHAR(45) NOT NULL,
  `Expiry Date` DATE NOT NULL,
  `Date of Issue` DATE NOT NULL,
  `CVV` INT(3) NOT NULL,
  `Withdrawl Limit` FLOAT NOT NULL DEFAULT 0,
  `Account Number` BIGINT(12) NOT NULL,
  `Used Limit` FLOAT NOT NULL,
  PRIMARY KEY (`Card Number`))
```

```
CREATE TABLE IF NOT EXISTS `dukh_bank`.`Customer Login Details` (
  `Account Number` BIGINT(12) NOT NULL,
  `Password Hash` VARCHAR(256) NOT NULL,
  PRIMARY KEY (`Account Number`),
  UNIQUE INDEX `Account Number_UNIQUE` (`Account Number` ASC) VISIBLE)


CREATE TABLE IF NOT EXISTS `dukh_bank`.`Employee Login Details` (
      `EmpID` INT NOT NULL,
      `Password Hash` VARCHAR(256) NOT NULL,
      PRIMARY KEY (`EmpID`),
      UNIQUE INDEX `EmpID_UNIQUE` (`EmpID` ASC) VISIBLE)


CREATE TABLE IF NOT EXISTS `dukh_bank`.`Loans` (
  `Amount` FLOAT NOT NULL,
  `Account Number` BIGINT(12) NOT NULL,
  `Date of Sanction` DATE NOT NULL,
  `Loan Type` VARCHAR(45) NOT NULL,
  `Interest Rate` FLOAT NOT NULL,
  `Next Due Amount` FLOAT NOT NULL,
  `Next Payment Date` DATE NOT NULL,
  `Tenure` INT(2) NOT NULL,
  `Amount Paid` FLOAT NOT NULL,
  PRIMARY KEY (`Loan Type`,`Account Number`) )
```

```
CREATE TABLE IF NOT EXISTS `dukh_bank`.`works in` (
    `Branches_Branch ID` INT NOT NULL,
    `Employee_EmpID` INT NOT NULL UNIQUE,
    `Date of Joining` DATE NOT NULL,
    PRIMARY KEY (`Employee_EmpID`))

CREATE TABLE IF NOT EXISTS `dukh_bank`.`Creates FD` (
  `Account Number` BIGINT(12) NOT NULL,
  `FD Number` INT NOT NULL UNIQUE PRIMARY KEY )

CREATE TABLE IF NOT EXISTS `dukh_bank`.`manages` (
  `Account Number` BIGINT(12) NOT NULL UNIQUE PRIMARY KEY,
  `Employee_EmpID` INT NOT NULL)
```

```
CREATE TABLE IF NOT EXISTS `dukh_bank`.`Branches` (
      `Branch ID` INT NOT NULL AUTO_INCREMENT,
      `Name of Branch` VARCHAR(45) NOT NULL,
      `Address` VARCHAR(180) NOT NULL,
      `Staff Size` INT(4) NOT NULL,
      `Manager Emp ID` BIGINT(12) NOT NULL,
      `Date of Opening` DATE NOT NULL,
      PRIMARY KEY (`Branch ID`),
      UNIQUE INDEX `Branch ID_UNIQUE` (`Branch ID` ASC) VISIBLE,
      UNIQUE INDEX `Manager Emp ID_UNIQUE` (`Manager Emp ID` ASC) VISIBLE)
```

# VIEWS AND GRANTS

We have created 2 users in our sql database;

1) User (This is selected whenever someone login as a customer)
2) Employee (This is selected whenever someone login as a bank employee)

**User granted privileges :-**

Select :- Accounts, Fixed Deposits,Transactions,Debit Card, Credit Card, Loans,Creates FD,Customer Login details

Insert :- Fixed Deposits , Transactions, Loans,Creates FD,Loans

Update :- Accounts , Fixed Deposits, Transactions,Debit card , Credit card,Loans , Creates FD

**Employee granted privileges :-**

Select :- Accounts, Fixed Deposits,Debit Card, Credit Card, Loans,Creates FD,Employee Login Details,

Insert :- Loans,Accounts , credit card , Debit Card

Update :- Accounts , Debit card , Credit card,Loans

Delete :- transactions,

## Views:-

We have created appropriate views for both the users. For example:-

- For employee :- this view contains the password hash and employee id.

```
String query2 = "Create view passwd_view AS SELECT `Password Hash`,`EmpID` FROM `Employee Login Details` Where `EmpID` = " + username ;
Statement st = mydb.createStatement();
st.executeUpdate(query2);
query = "Select `Password Hash` from passwd_view where `EmpID` = " + username;
```

- For user :- this view contains the password hash and account no.

```
String query2 = "Create view passwd_view AS SELECT `Password Hash`,`Account Number` FROM `Customer Login Details` Where `Account Number` = " + username ;
Statement st = mydb.createStatement();
st.executeUpdate(query2);
query = "Select `Password Hash` from passwd_view where `Account Number` = " + username;
```

# SQL - Queries :

1. The bank wants to award some goodies to the top 5 customers with the highest current account balance and who haven't availed a single loan till date. Find the details of these customers and display them based on the descending order of their current account balance.

   SELECT A.`Account Number`, A.`First Name`,A.`Balance`, RANK() OVER(ORDER BY A.`Balance` desc) AS "Rank" FROM Accounts A WHERE NOT EXISTS ( SELECT L.`Account Number` FROM Loans L WHERE A.`Account Number` = L.`Account Number` );

2. The Board of directors of the D.U.K.H Bank wants to award employee of the year to the employee who is handling the most user accounts at present. Create a view of details of the employees and display them based on the descending order of the number of user accounts they have handled.`

   Select Employee_empid,`Number of Accounts`,dense_rank() over (order by `Number of Accounts` desc) As "Employee Rank" From (Select Employee_empid , count(*) as `Number of Accounts` from manages group by Employee_empid) as M;

3. There was a data breach in Branch-3 of our bank. Write a query to find all the accounts belonging to that branch so that the Account holders can be notified and other appropriate steps can be taken.

   *SELECT DISTINCT M.`Account Number` AS `Account Number` from Manages M,Employee E,`Works In` W where E.empID in (SELECT WW.Employee_EmpID from `Works In` WW where WW.`Branches_Branch ID`=3 ) AND E.empID=M.Employee_EmpID;*

4. The manager of the bank wants to inform all the credit card users whose credit cards will expire in the year 2022, regarding the renewal of the credit cards in order to keep them well informed. Find the details of only the credit card users(not debit cards users) whose card's expiry date is in 2022.

*SELECT A.`Account Number`,A.`First Name`,A.`Last Name`, A.Balance From Accounts A, `Credit Card` C WHERE NOT EXISTS( Select D.`Account Number` from `Debit Card` D where C.`Account Number` = D.`Account Number`) AND C.`Expiry Date` between "2022-01-01" and "2020-12-31";*

5. The Census commissioner of the government of India wants to conduct a census on the gender ratio of the bank manager working in the banking sector. Find the total number of males and female managers working in a particular branch of D.U.K.H bank.

   Select employee.gender, count(gender)  from( branches join employee on branches.`manager emp id` = employee.empid) group by gender;

6. The HR Department of D.U.K.H bank with Branch ID 3 wants to ensure that the average, minimum and maximum salary of both its male and female employees lie in the same bracket. Find the employees name, their salary, their emp id, their gender, and the average, minimum and maximum salary of both the gender respectively.

Select E.`First Name`,E.`Last Name`, E.`EmpID`,E.`Salary`,E.`Gender` , AVG(E.`Salary`) OVER (PARTITION BY E.`Gender`) As `Average Salary`, MIN(E.`Salary`) OVER (PARTITION BY E.`Gender`) As `Minimum Salary`, MAX(E.`Salary`) OVER (PARTITION BY E.`Gender`) As `Maximum Salary` FROM Employee E, Branches B, `Works In` W WHERE E.`EmpID` = W.`Employee_EmpID` AND W.`Branches_Branch ID`= 3 AND B.`Branch ID`=3;

7.Some people like to invest a lot in Fixed Deposit. Find the Account Holders who have more money in their FDs than in their Account Balance.

*SELECT * FROM (SELECT A.`Account Number` AS `Account Number` , sum(F.`Current Amount`) AS `FD Balance`, A.Balance as Balance from Accounts A, `Fixed Deposits` F, `Creates FD` C where A.`Account Number`= C.`Account Number` AND C.`FD Number`=F.`FD Number` group by `Account Number`) AS FD_SUMS WHERE FD_SUMS.`FD Balance`>=FD_SUMS.Balance;*

8.The Management wants to know which payment modes are used the most. Write a query to give the frequency of all the payment modes provided by the bank.

*SELECT `Mode of Payment`, count(*) AS Frequency FROM transactions GROUP BY `Mode of Payment`*

9. Older people often avoid trying out newer technology. Find all the Account holders born before 1970 who neither own a credit or a debit card so that our bank can target them with a new scheme.

SELECT A.`Account Number` from Accounts A where A.`Date Of Birth` < '19700101' AND not exists( select * from `credit card` c, `debit card` d where A.`Account Number`=c.`Account Number` OR A.`Account Number`=d.`Account Number` );

10.The Education Ministry wants to encourage women to pursue higher education. So, they want to help women who have recently taken education loans by refunding the interest paid by them. Write a query to find all the female account holders who have taken an education loan after 31.12.2008 along with the amount of Loan taken and Amount already paid by them.

*SELECT A.`Account Number`, L.`Amount Paid`, L.`Amount` from Accounts A, Loans L where L.`Date of sanction` > 20081231 AND L.`Loan Type`='education' AND L.`Account Number`= A.`Account Number` AND A.Gender='Female';*

**INDEXING:-**

**Indexing is done for the attributes to enable much faster searching for every query. This is more suitable for queries with larger datasets as it reduces the time of searching to O(log n) from O(n)**

**We have indexed all attributes that possess the uniqueness trait within a table. This includes the primary keys for every table.**
**Other than that we indexed the following attributes as they were frequently used in our embedded sql queries:**

1. **Balances in every table are indexed in descending order.**
2. **Dates in every table are indexed in ascending order.**
3. **Account Number is indexed in credit cards table to allow for easier display of credit cards of each user**
4. **Account Number is indexed in debit cards table table to allow for easier display of debit cards of each user**
5. **Account Number is indexed in loans table to easily sort by account number for each user**
6. **Account Number is indexed in creates FD Table to easily sort by account number for each user**

**The following is one of the indexes we created:**

```
    }
}
CREATE TABLE IF NOT EXISTS `dukh_bank`.`Credit Card` (
      `Card Number` BIGINT(16) NOT NULL,
      `Card Network` VARCHAR(45) NOT NULL,
      `Date of Issue` DATE NOT NULL,
      `Expiry Date` DATE AS (DATE_ADD(`date of issue`, INTERVAL 5 YEAR)),
      `CVV` INT(3) NOT NULL,
      `Withdrawl Limit` FLOAT NOT NULL DEFAULT 0,
      `Account Number` BIGINT(12) NOT NULL,
      `Used Limit` FLOAT NOT NULL,
      PRIMARY KEY (`Card Number`),
      UNIQUE INDEX `Card Number_UNIQUE` (`Card Number` ASC) VISIBLE)

    create index Account_number_index on `Credit Card`(`Account Number` ASC)
```

# Triggers

We have made various useful triggers in our database to maintain data integrity and support commonly used actions.

Whenever we try to remove/break a Fixed Deposit, Trigger Break_FD obtains the value of the FD, adds it to the balance of the Parent Account and removes the row containing that FD's number from the `Creates FD` table. This cleans up the process of breaking the FD and makes sure that we don't have inconsistent records in our database.

```
DELIMITER $$

create trigger Break_FD
before delete
on `Fixed Deposits` for each row
begin
  declare acc_no bigint(12);
  declare b float;

  select `Account Number` into acc_no from `Creates FD` where `FD Number`=OLD.`FD Number`;
  delete from `Creates FD` where `FD Number`=OLD.`FD Number`;
  select `Current Amount` into b from `Fixed Deposits` where `FD Number`=OLD.`FD Number`;

  Update Accounts Set balance=balance+b where `Account Number`=acc_no;

end$$
```

Trigger Deactivate_Credit_Card makes sure that every time we remove a credit card, the amount spent using the credit card is first deducted from the parent Account.

```
DELIMITER $$

create trigger Deactivate_Credit_Card
before delete
on `Credit Card` for each row
begin
  Update Accounts Set balance=balance-OLD.`Used Limit` where `Account Number`=OLD.`Account
Number`;
end$$
```

Whenever we delete an Account, we need to update a lot of tables. If we do it manually every time, we are bound to miss out on something and mess up our database. Trigger Delete_Account takes care of all these loose ends for us. It deletes the records of all the FDs, Loans, Cards, Transactions, and passwords held by that account.

```
DELIMITER $$

create trigger Delete_Acoount
before delete
on `Accounts` for each row
begin

  Delete from `Fixed Deposits` where `FD Number` in (Select `FD Number` from `creates FD`
where `Account Number`=OLD.`Account Number`);
  Delete from `creates fd` where `Account Number`=OLD.`Account Number`;
  Delete from `Transactions` where Sender=OLD.`Account Number` OR receiver=OLD.`Account
Number`;
  Delete from `Debit Card` where `Account Number`=OLD.`Account Number`;
  Delete from `Credit Card` where `Account Number`=OLD.`Account Number`;
  Delete from `Customer Login Details` where `Account Number`=OLD.`Account Number`;
  Delete from `Loans` where `Account Number`=OLD.`Account Number`;
  Delete from `Manages` where `Account Number`=OLD.`Account Number`;

end$$
```

```
DELIMITER $$

create trigger Handle_Loan
before Update
on `Loans` for each row
begin
  if(new.`Next Due Amount`=0) then
    set new.`Next Payment Date` = DATE_ADD(old.`Next Payment Date`, INTERVAL 1 month), new.`Next Due
Amount`=new.`Amount`/new.tenure;

  end if;

end$$
```

Trigger Handle_loan adjusts next payment date and next due amount whenever the user tries to settle an installment of the loan.

# The UI

The following are some screenshots of our fully functional and interactive UI: