REPORT

Members:

Devyani Koshal// 2020055

Navidha Jain// 2020223

Siddhant Agarwal//2020247

Yuvraj Singh//2020270

GitHub Repository: https://github.com/siddhant20247/Banking_DBMS

Scope of the project:

The project provides reliable banking services to the customers.

- Customers can open accounts, open fixed deposits, request for cards (credit and debit) and request for loans.
- Employees can approve or reject loans. They can also access all accounts for a branch.
- Managers are a special kind of employees who have access to other employees(not in front-end).

It uses the concepts of Database Management System for managing the data.

Stakeholders:

- Customers
- Employees (including Managers)

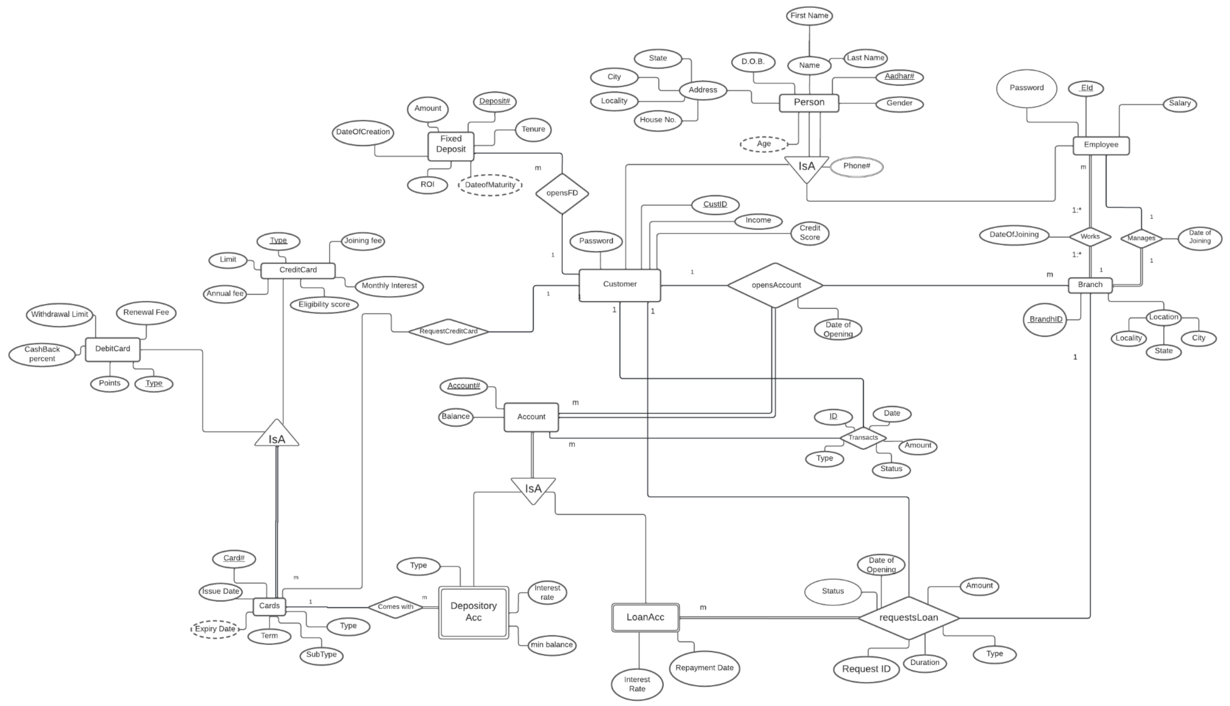
- **Entities:**
    - Person: handles the information related to a person with the person's Aadhar number as the primary key.
    - Employee: handles the information related to an employee with the employee ID
    - as the primary key.
    - Customer:handles the information related to a customer with the customer ID as the primary key.
    - Branch:handles the information related to a branch with the branch ID as the primary key.
    - Fixed Deposit:handles the information related to a fixed deposit with the deposit number as the primary key.
    - Account:handles the information related to an account with theaccount number number as the primary key.
    - Cards:handles the information related to Cards with the Card number as the primary key.

- **Relationships:**
    - Customer opens a Fixed Deposit.
    - Customer can request for a Credit card.
    - Customer can request a loan and gets a Loan Account in a Branch.
    - Customer opens an Account in a Branch.
    - Credit card and Debit Card are types of Cards.
    - Loan Account and Depository Account are types of Accounts.
    - Employee works at a Branch.
    - Customers and Employees are Persons.
    - Cards come with a Depository Account.
    - Employee can Manage a Branch.

Weak Entities:

- DepositoryAcc(Account#, InterestRate, minBalnce, Type, DebitCard#)
  Reason:Identifying Entity is Account, If there is no Account then the DepositoryAcc entity doesn't exist. There is no unique identifying attribute in DepositoryAcc and we require account number from Account to identify DepositoryAcc. The discriminator is the 'Type' attribute.
- LoanAcc(InterestRate, RepaymentDate, Account#)
  Reason: Identifying Entity is Account, If there is no Account then the LoanAcc entity doesn't exist. There is no unique identifying attribute in LoanAcc and we require account number from Account to identify LoanAcc. The discriminator is the 'Interest Rate' attribute.

UPDATED ER DIAGRAM:



Link:
https://lucid.app/lucidchart/906aa7aa-9dfb-4d57-b1a3-c2bdfd7f854e/edit?invitationId=inv_e32fc9f0-0bb5-48cd-81d0-4ce8fee4b901

Updated Relational Schema:

Person(<u>Aadhar#</u>, FirstName, LastName, DOB, Gender, HouseNo, Locality, City, State)
PhoneNumbers(<u>Aadhar#, Phone#</u>)
Customer(<u>CustID</u>, Password, Income, Credit Score, Aadhar#)
Employee(<u>Empid</u>, Salary, aadharno, Password)
Branch(<u>BId</u>, Locality, City, State)
Account(<u>Account#</u>, Balance)
FixedDeposit(<u>Depositno</u>, TenureMonth, DateOfCreation, Amount, ROI, Custid)
DepositoryAcc(<u>Account#</u>, InterestRate, minBalance, Type, debitcard#)
LoanAcc(InterestDate, repayementdate, <u>Account#</u>)
Cards(<u>Cardno</u>, Termyears, IssueDate, CType, CSubtype)
DebitCard(Renewal_Fee, Points, <u>CType</u>, Cashback_perecnt, Withdrawal_Limit)
CreditCard(<u>CType</u>, Joining_fee, Annual_Fee, cash_Limit, Eligibility_score, MonthlyInterest)

AccountOpened(CustID, branchID, <u>Account#</u>, DateOfOpening)
Transactions(<u>transactionID</u>, CustId, Account#, DateTime, Amount, Status, type)
LoanRequest(<u>RequestID</u>,DateofOpening, Amount, DurationMonths, Type, CustID, branchID, accno, status)
Manages(<u>Empid, Branchid, DOJ</u>)
creditcardrequest(<u>card#, custid</u>)
employeeworks(<u>empid, branchid, doj</u>)

Views:

1. View created ofemployees with their name and employee ID working at the branch whose ID is 2176:

Ans:

```
CREATE VIEW employees_of_branch_2176 AS
SELECT DISTINCT p.firstname, p.lastname, e.empid
FROM person p, employee e
WHERE p.aadharno = e.aadharno AND e.empID in
        (SELECT e1.empid
         FROM employee e1, employeeworks w, branch b
        WHERE e1.empid = w.empid AND w.branchid = b.branchid AND b.branchid = 2176
);
```

2. View created of customers with their name and Customer ID who have a Debit Card of type Master Card:

Ans:

```
CREATE VIEW MasterCardTypeDebit AS
SELECT DISTINCT p.firstname, p.lastname, cust.CustID
FROM person p,  customer cust
WHERE p.aadharno = cust.aadharno AND cust.CustID in
        (SELECT distinct cust1.CustID
        FROM customer cust1,  depositoryacc d1, accountopened a1
        WHERE cust1.CustID = a1.CustID AND a1.`Account#` in
                (SELECT distinct a2.`Account#`
                FROM  depositoryacc d2, accountopened a2
                WHERE a2.`Account#` = d2.`Account#` and d2.`DebitCard#`in
                        (Select cards.CardNo
                        FROM cards
                        where cards.CSubType = "MasterCard" and cards.CType="Debit")));
```

3. View created of customers with their name, customer ID, income who have some holdings in the branch with ID 2176 and whose income is more than 3 lakhs:

Ans:

```
CREATE VIEW customers_of_branch_2176_income_morethan_300000 AS
SELECT DISTINCT p.firstname, p.lastname, c.custID, c.Income, a1.BranchID
FROM person p, customer c, accountopened a1
WHERE p.aadharno = c.aadharno AND a1.BranchID=2176 AND c.CustID in
        (SELECT c1.CustID
        FROM customer c1, accountopened a
        WHERE c1.custID= a.CustID AND c1.Income>300000 AND a.BranchID =2176);
```

4. View created of female customers who own a fixed deposit in any branch.

Ans:

```
CREATE VIEW Female_Customers_with_FD AS
SELECT DISTINCT p.firstname, p.lastname, c.custID
FROM person p, customer c
WHERE p.Gender ='F' and  p.aadharno = c.aadharno AND  c.CustID in
        (SELECT c1.CustID
        FROM customer c1, fixeddeposits f
        WHERE c1.custID= f.CustID );
```

Grants:

1. A role for Administrator is created who has been granted the authority to SELECT, INSERT, UPDATE, DELETE the database tables of the entire Banking System.

Ans:

CREATE USER IF NOT EXISTS AdminBank@localhost IDENTIFIED BY 'Administrator';
CREATE ROLE IF NOT EXISTS 'Administrator';
GRANT SELECT, INSERT, UPDATE, DELETE ON BankingSystem.* TO 'Administrator';

2. **A role for Manager of the branch with ID 2176 is created who has been granted the authority to SELECT, INSERT, UPDATE, DELETE the database tables of the employees and customers of that branch.**

Ans:

CREATE USER IF NOT EXISTS BranchManager2176@localhost IDENTIFIED BY 'Manager';
CREATE ROLE IF NOT EXISTS 'Manager';
GRANT SELECT, INSERT, UPDATE, DELETE ON employees_of_branch_2176 TO 'Manager';
GRANT SELECT, INSERT, UPDATE, DELETE ON customers_of_branch_2176 TO 'Manager';

3. **A role for employees working at the branch with ID 2176 is created who have been given the authority to SELECT, INSERT, UPDATE, DELETE the database tables of the customers of that branch.**

Ans:

CREATE USER IF NOT EXISTS Employee1@localhost IDENTIFIED BY 'Employees_2176';
CREATE ROLE IF NOT EXISTS 'Employees_2176';
GRANT SELECT, INSERT, UPDATE, DELETE ON customers_of_branch_2176 TO 'Employees_2176';

Triggers:

1. Trigger to process pending transactions, when a customer requests for a transaction, it checks the status and proceeds accordingly.

Ans:

```
DELIMITER $$
CREATE TRIGGER `pendingTransactionsToAcc`
AFTER INSERT ON transactions FOR EACH ROW
BEGIN
IF ( new.`Status`="P")
         then
   IF new.`Type` = "CREDIT"
                  THEN
                  update account
                  set account.`Balance` = account.`Balance` + new.`Amount`
                  WHERE account.`Account#` = new.`Account#`;

         ELSE
   update account
   set account.`Balance` = account.`Balance` - new.`Amount`
         WHERE account.`Account#` = new.`Account#`;
   END IF;
END IF;
END
$$
```

2. Trigger to increment salary of an employee by 10,000 when promoted as manager

```
DELIMITER $$
CREATE TRIGGER `Promotion`
AFTER INSERT ON manages FOR EACH ROW
BEGIN
update employee
set Salary = Salary  + 10000
WHERE employee.`empId` = new.`empId`;
END
$$
```

3. Trigger to increase balance by 1000 on request of a credit card.

```
DELIMITER $$
CREATE TRIGGER `Cashback`
AFTER INSERT ON `creditcardrequests` FOR EACH ROW
BEGIN
update account
set Balance = Balance  + 1000
WHERE `account#` = (SELECT  a1.`account#` FROM accountopened a1 WHERE a1.`custid` = new.`custid`
LIMIT 1);
END
$$
```

4. Trigger to charge extra on transactions beyond 100000

Ans:

```
DELIMITER $$
CREATE TRIGGER `TransactionCharge`
AFTER INSERT ON transactions FOR EACH ROW
BEGIN
IF (new.`amount` > 100000) THEN
        update account
        set `Balance` = `Balance` - 5
        WHERE account.`Account#` = new.`Account#`;
END IF;
END
$$
```

SQL Queries:

1. Query to return the IDs of customers with credit score in range 550 to 750 who do not have a credit card.

Ans:

```
SELECT c.custid, c.`Credit Score`
FROM customer c
WHERE c.`Credit Score` BETWEEN 550 AND 750
AND c.custid NOT IN (
        SELECT DISTINCT custid
        FROM creditcardrequests);
```

2. Query to decrease interest rate by 1% for home loans.

Ans:

```
UPDATE loanacc l
SET InterestRate = InterestRate - 1
WHERE l.`Account#` IN (SELECT loanrequests.`Account#`
                    FROM loanrequests
                    WHERE loanrequests.type = 'HOME');
```

3.Query to return the name and IDs of those Employees working at the same branch for more than 10 years.

Ans:

```
SELECT p.firstname, p.lastname, e.empid
FROM person p NATURAL JOIN employee e
WHERE e.empid IN (SELECT empid FROM (
          SELECT max(DOJ) as latestjoining, empid
          FROM employeeworks
          GROUP BY empid
          HAVING DATEDIFF(CURDATE(), latestjoining) > 3650) AS currworks );
```

4. Query to return the account number of those accounts whose total withdrawal is less than 5000000.

Ans:

```
SELECT a.`Account#` , sum(t.amount) as total_withdrawal
FROM account a, transactions t
WHERE a.`Account#` = t.`Account#` AND t.type = 'DEBIT' AND t.status = 'C'
GROUP BY a.`Account#`
HAVING total_withdrawal < 5000000;
```

5. Query to return accounts with ex[ired debit cards.

Ans:

```
SELECT d.`Account#`, a.balance
FROM depositoryacc d, account a WHERE d.`Account#` = a.`Account#`
AND d.`DebitCard#` NOT IN (
          SELECT c.cardno
          FROM cards c
          WHERE  DATEDIFF(CURDATE(), c.IssueDate) - (c.term_months*30)  < 0);
```

6. Query to delete those accounts who have not made a transaction in the past 10 years.

Ans:

```
DELETE
FROM account
WHERE `account#` NOT IN ((SELECT DISTINCT `account#`
          FROM transactions
          WHERE DATEDIFF(CURDATE(), datetime) < 3653
           ) UNION (SELECT `account#` FROM loanacc));
```

7. Query to return the number of customers whose name start with ‘R’ and who also work at theat branch.

```
SELECT COUNT(DISTINCT p.AadharNo) as num_people FROM person p, customer c,
employee e
WHERE p.AadharNo = c.AadharNo AND p.AadharNo = e.AadharNo AND p.firstname LIKE
'R%';
```

Embedded SQL Queries:

1. Input a customer ID, query to retrieve that customer:

Ans:

```
async function loginCustomer(customerId, password) {
  const rows = await db.runQuery(
    `select * from customer where CustID = ${customerId}`
  );
  const data = helper.emptyOrRows(rows);
  if (data.length === 0) {
    return {
      data: "Customer not found",
    }
  }
  if (data[0].Password === password) {
    return {
      data,
    }
  } else {
    return {
      data: "Login Failed",
    }
  }
}
```

2. Input a customer ID, query to retrieve their total sum amount of assets in the bank.

Ans:

```
async function getTotalsByCustId(customerId) {
  const rows = await db.runQuery(
    `SELECT (SELECT sum(balance) FROM account NATURAL JOIN accountopened WHERE
balance > 0 GROUP BY custid HAVING custid = ${customerId}) + (SELECT sum(amount)
FROM fixeddeposits GROUP BY custid HAVING custid = ${customerId}) as assets, (SELECT
sum(balance) FROM account NATURAL JOIN accountopened WHERE balance < 0 GROUP
BY custid HAVING custid = ${customerId}) as liabilities;`
  );
  const data = helper.emptyOrRows(rows);

  return {
    data,
  }
}
```

3. Input an Employee ID, query to retrieve the branch’s manager.

Ans:

```
 async function getManagersByEid(eid) {
  const rows = await db.runQuery(
    `SELECT m1.*
      FROM manages m1
        LEFT OUTER JOIN manages m2
          ON (m1.branchid = m2.branchid AND m1.doj < m2.doj)
      WHERE m2.branchid IS NULL AND m1.empid = ${eid}`
  );
  const data = helper.emptyOrRows(rows);

  return {
    data,
  };
}
```

4. Input the required information, query to create a transaction.
Ans:
```
async function createTransaction(customerId, accountNumber, amount, transactionType) {
   const transactionId = await db.runQuery(
      `Select transactionid from transactions`
   );
   var transactionIdNumber;
   while (transactionId.contains(transactionIdNumber)) {
      transactionIdNumber = transactionId[0].transactionid + 1;
   }
   console.log(transactionIdNumber);
   const DateTime = new Date.now();
   const rows = await db.runQuery(
     `INSERT INTO transactions (transactionid , custid, \`Account#\`, amount, transactiontype,
datetime) VALUES (${transactionIdNumber}, ${customerId}, ${accountNumber}, ${amount},
'${transactionType}, ${DateTime}')`
   );
   const data = helper.emptyOrRows(rows);

   return {
      data,
   }
}
```

5. Input a request ID, query to reject a loan.
Ans:
```
async function rejectLoan(requestID) {
   console.log(requestID);
   const rows = await db.runQuery(
     `UPDATE loanrequests SET status = 'F' WHERE requestid = ${requestID};`
   );
   const data = helper.emptyOrRows(rows);

   return {
      data,
   }
}
```

Indexing:

Query 1:
CREATE INDEX indexForCS
ON customer(`Credit Score`);
Row reduction : 30 → 8

Query 2:
Added index on loan type reduce row nos 5 → 1

CREATE INDEX idxloanrequests
ON loanrequests(type);

Query 3:
CREATE INDEX idx10yrs
ON employeeworks(DOJ);

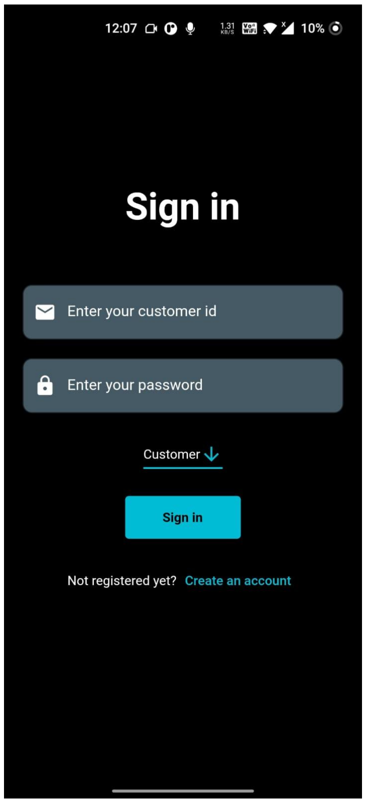
Query 4:
created index to reduce no. of rows to query on. The index on account no. was already present.
Added index on transaction type and status. Reduced row nos 50 → 32
CREATE INDEX idx
ON transactions(Type);
CREATE INDEX idx2
ON transactions(Status);

Query 5:
CREATE INDEX idxcards ON cards(`Term_Months`);
CREATE INDEX idxcards2 ON cards(`IssueDate`);
CREATE INDEX idxcards3 ON cards(`cardNo`);
CREATE INDEX idxdep ON depositoryacc(`DebitCard#`);

Query 6:
CREATE INDEX inactiveAccounts ON account(`Account#`);
CREATE INDEX inactiveAccounts2 ON transactions(`DateTime`);

Query 7:
CREATE INDEX indexForCount ON person(`firstname`);
Rows reduced to 50 → 5

FRONT-END DESIGN


12:07 10%

Sign in

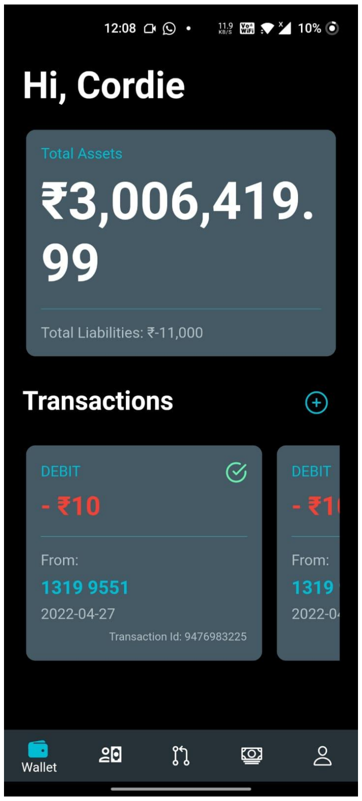
Enter your customer id

Enter your password

Customer

Sign in

Not registered yet? Create an account


12:08 10%

Hi, Cordie

Total Assets

₹3,006,419.99

Total Liabilities: ₹-11,000

Transactions

DEBIT

- ₹10

From:

1319 9551

2022-04-27

Transaction Id: 9476983225

DEBIT

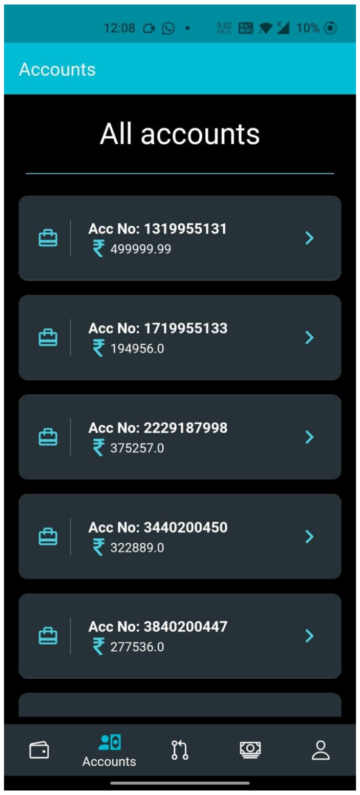
- ₹1

From:

1319

2022-0

Wallet


12:08 10%

Accounts

All accounts

Acc No: 1319955131
₹ 499999.99

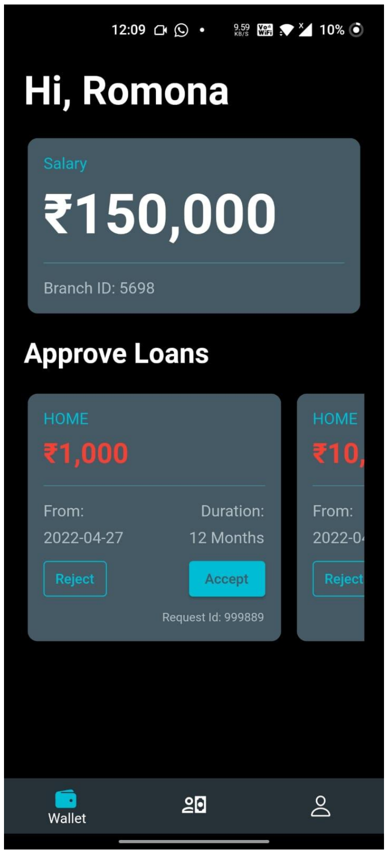
Acc No: 1719955133
₹ 194956.0

Acc No: 2229187998
₹ 375257.0

Acc No: 3440200450
₹ 322889.0

Acc No: 3840200447
₹ 277536.0

Accounts


12:09 10%

Hi, Romona

Salary

₹150,000

Branch ID: 5698

Approve Loans

HOME

₹1,000

From: 2022-04-27

Duration: 12 Months

Reject Accept

Request Id: 999889

HOME

₹10,

From:

2022-0

Reject

Wallet