CS 387 Project

Maulik Shah (13D100004)
Shantanu Thakoor (13D100003)
Aman Goel (130050041)

Banking Made Easy

Introduction

Through this project work, we aim to create a simplified version of a banking application. The application will act as a portal which provides it's users with various banking facilities like account management and transaction management. The aim is to learn how to create an application which uses a database in its backend. We also plan to make the application safe and secure so that it is not susceptible to vulnerabilities and security attacks.

Approach

We have divided the project work in the following stages:

1. We will first decide the tables and the respective attributes needed for our database
2. We will then sketch an ER diagram of the database
3. We will then plan out the appropriate screens needed for our application
4. We will then move on to the programming part of the application
5. We will then create the associated tests for our application to ensure that it works correctly

Tools used

We will be using the following tools to design our application:

1. **Gliffy** - this is a tool that is used to draw diagrams and flowcharts. We will used Gliffy for the design of the ER diagrams
2. **Eclipse - this is the major tool that we will be using for our coding part**
3. **Git - for version control and distributed coding**

Database Design

Our database will consist of the following tables:

1. **Customer** - The customer table will store the data of each customer of the bank. Attributes stored -
    a. customer ID - char(n) primary key
    b. first name - varchar(n) not null
    c. last name - varchar(n) not null
    d. age - integer not null (age > 0)
    e. address - varchar(n) not null
    f. email ID - varchar(n)
    g. gender - enum not null
    h. contact number - integer not null (check the number of digits in mob. number is 10)
2. **Account** - The account table will store the data of each account managed by the bank. Attributes stored -
    a. account number - char(n) primary key
    b. account type - enum not null
    c. minimum balance - numeric not null (minimum balance > 0)
    d. current balance - numeric not null (current balance > 0)
    e. first owner ID - char(n) not null (foreign key to customer table)
    f. second owner ID - char(n) (foreign key to customer table - can be pointing to a dummy customer)
    g. interest rate - numeric not null (interest rate > 0)
    h. branch ID - char(n) not null (foreign key to the branch table)
    i. opening date - date not null
3. **Branch** - The branch table will store the details of the various branches of the bank. Attributes stored -
    a. branch ID - char(n) primary key
    b. address - varchar(n) not null
    c. IFSC code - char(n) not null
    d. Contact number - integer not null (check the number of digits in mob. number is 10)
4. **Transaction** - The transaction table stores the information about each transaction that happens in the bank. Attributes stored -
    a. transaction ID - integer not null
    b. transaction type - enum not null (debit or credit)
    c. transaction method - enum not null (cash/cheque/demand draft/etc)
    d. date - date not null
    e. cheque number - char(n) (can be null if paid by cash)
    f. account number - char(n) not null (foreign key to the account table)

g. amount - numeric not null (amount > 0)
h. remarks - varchar(n)
i. balance - numeric not null (balance > 0)

(account number, transaction ID) is the primary key

5. **Transfer** - The transfer table stores the data of each transfer that happens across 2 bank accounts. Attributes stored -
    a. transfer ID - integer not null
    b. account number of sender - char(n) not null (foreign key to the account table)
    c. account number of receiver - char(n) not null
    d. amount - numeric not null
    e. date - date not null
    f. remarks - varchar(n)
    g. balance of sender - numeric not null
    h. balance of receiver - numeric not null

    (account number of sender, account number of receiver, transfer ID) is the primary key

6. **Login** - The login table stores the login credentials of various customers of the bank. Note that customer ID will act as the username for each customer and will be allotted by the bank. Attributes stored -
    a. password - varchar(n) not null
    b. last login date - date not null
    c. account number - char(n) primary key (foreign key to account table)
7. **Cheque book** - We want to provide the facility of issuing cheque books to customers. However, we want to ensure that a customer requests a new cheque book only when he has completely used the last cheque book issued to him/her. For this, we maintain the cheque book table. Attributes stored -
    a. account number - char(n) not null (foreign key to the account table)
    b. cheque book ID - char(n) not null
    c. number of cheques in the cheque book - integer (check > 0)
    d. first cheque ID - char(n) not null
    e. last cheque ID - char(n) not null
    f. date of issue - date not null
    g. issuing branch ID - char(n) (foreign key to the branch table)

    (account number, cheque book ID) is the primary key

8. **Demand draft** - Our application has a facility of issuing demand drafts as well. Attributes stored -

a. account number - char(n) not null (foreign key to the account table)
b. date - date not null
c. amount - numeric not null (amount > 0)
d. draft ID - char(n) not null(primary key)
e. issuing branch ID - char(n) (foreign key to the branch table)
f. Payee name - varchar(n)
g. balance - numeric not null (balance > 0)

9. **Debit Card** - We also provide the facility of issuing debit cards. Attributes stored -
    a. account number - char(n) primary key (foreign key to account table)
    b. card number - integer not null
    c. expiry date - date not null
    d. issuing branch ID - char(n) not null (foreign key to branch table)
    e. CVV code - integer not null
    f. online password - varchar(n) not null

    Note - each account is allowed to have just 1 debit card and each debit card belongs to exactly 1 account

10. **Utility bill** - For providing bill payment facility. Attributes stored -
    a. account number - char(n) not null (foreign key to account table)
    b. bill ID - integer not null
    c. bill amount - numeric not null (bill amount > 0)
    d. bill type - enum
    e. bill paid to - varchar(n)
    f. balance - numeric not null (balance > 0)

    (account number, bill ID) is the primary key

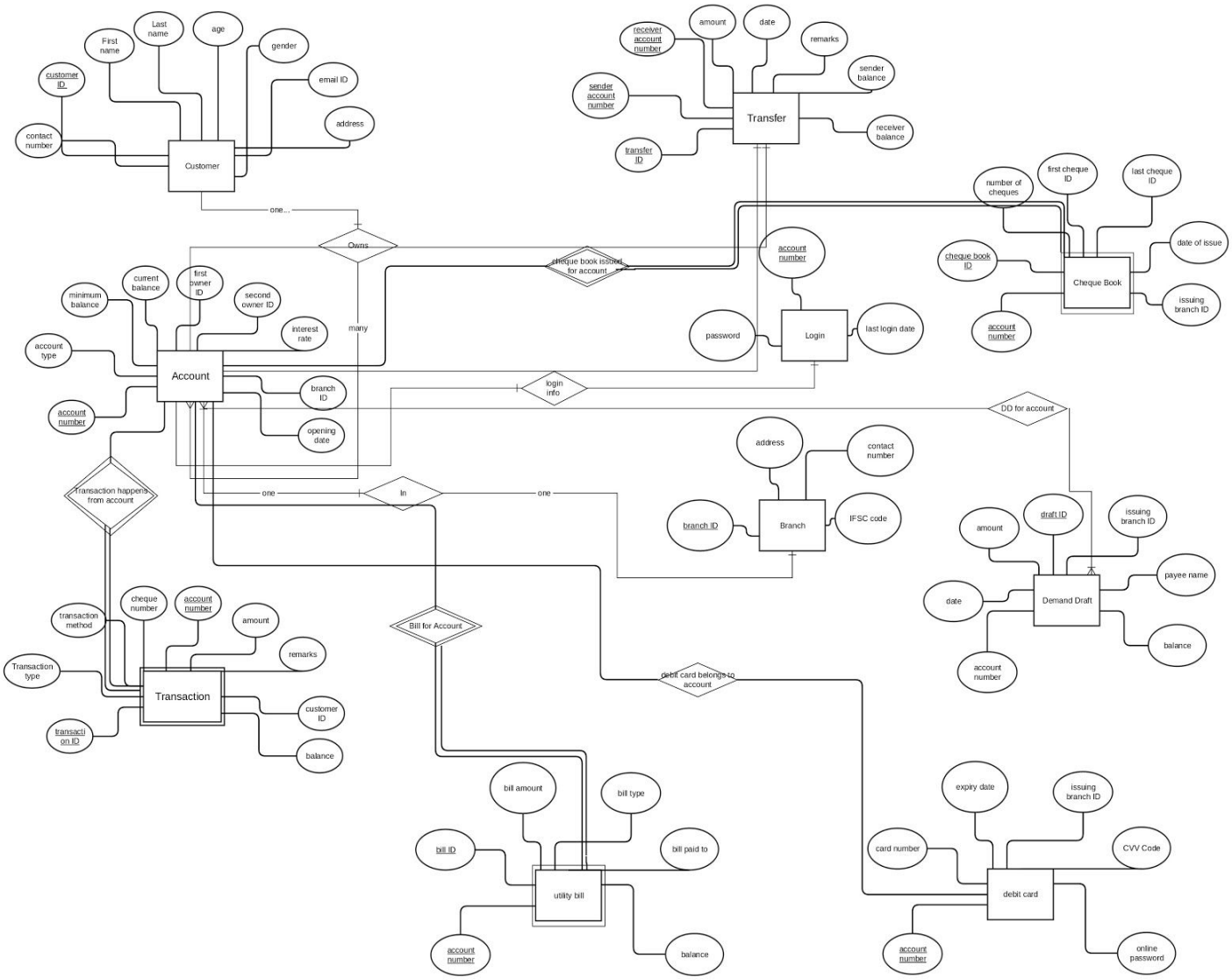
Customer Interfaces of the application:

1) Customer ID registration: Provide details of all the required fields in the customer database table, security questions, etc.
2) Forgot password screen: Based on answers to security questions, issue a new password for a certain account
3) Request extra features: Depending on some conditions, a customer may request a premium account/credit card, automatic bill payment facility, etc.
4) Passbook feature: Check transaction summary of certain accounts
5) Initiate transfer: an account owner should be able to transfer funds within the bank on his own

Employee Interfaces of the application:

1) Create new account: after verifying documents by a customer, create an account according to his application, linking it to his customer ID
2) Initiate transaction: when a customer comes to deposit/withdraw money, after verification the employee should add a new transaction to the corresponding account
3) Demand draft creation and issuing
4) Chequebook request and issuing
5) Close a certain account
6) Convert an account between single and joint account

ER Diagram:



Normalization of the database tables:

1. In order to avoid redundant data, we tried to normalize the database tables
2. This helped us to reduce the storage requirements for the database

For example, we were first planning to store the customer information with the account information in just 1 table. But then we felt that this will lead to redundancy in the database because 1 customer may have several accounts and 1 account can belong to several customers as well. And so, we divided our table to 2 separate tables - 1 for the customers and 1 for the accounts.

Another example - we were doing the same mistake for transactions and cheque books and then we realized that we can avoid redundant data by creating a separate table for transactions and a separate table for cheque books.

Indexes of database:

1) Since we will often do inner join between tables based on customer ID and account number, we should make these two attributes an index for more efficient querying
2) Since we will have to do a join between transfer/transaction and chequebook/demand draft, we should also create indexes on these attributes (draft ID and chequebook ID)

Implementation details:

We will be needing 2 Java classes for the application:

- Customer class - this class will provide all the functionality that a customer needs. It will have various methods that will be needed by a customer to use the application. The class will interact with the database to provide results of various queries of the customers.
- Bank Employee class - this class will act as an interface between database and a bank employee and will provide the necessary functions that a bank employee will need in order to fulfil the requests of the customer.

Servelets used:

We will be using a different servelet for each screen provided to both customers and bank employees. We will use only post requests. This is for security purposes so that the important data doesn't go as a URL parameter.

Safety of the application:

In order to make the application safe and secure for the users, we will try to implement several security measures on the application. Some of these are listed below:

- Avoiding SQL injection attacks - we will use prepared statements for this purpose
- Avoiding XSS (cross site scripting) attacks - we will sanitize user input before passing it to a database query

Sample test cases:

There are several corner cases we will have to test our application on: transfer to nonexistent account, withdrawing more money than balance, handling joint account logins correctly, security to be maintained in the forgot password option, etc.