BANK MANAGEMENT SYSTEM

Abstract :

This project aims to create an all-in-one banking system to make bank operations more efficient and improve customer satisfaction. The system will include many features such as managing customer accounts, handling transactions, processing loan applications, overseeing branches and ATMs, issuing cards, making transfers, calculating interest, and managing employee roles. By using advanced technology and focusing on user needs, the system will provide a smooth and secure banking experience for both customers and employees. The main goal is to simplify banking processes, reduce manual work, and enhance service quality, which will help build customer loyalty and trust.

Introduction :

In today's financial world, banks are always looking for new ways to improve their services and make their operations run more smoothly. This project aims to create a complete banking system that meets the many needs of modern banks.

The system will have several key parts, including:

1. **Customer Account Management**: Easily handling customer information and account details.
2. **Transaction Handling**: Making deposits, withdrawals, and payments quick and secure.
3. **Loan Application Processing**: Simplifying the process of applying for and approving loans.
4. **Branch and ATM Management**: Overseeing the operations of multiple branches and ATMs efficiently.
5. **Card Issuance**: Managing the creation and distribution of credit and debit cards.
6. **Fund Transfers**: Ensuring safe and efficient money transfers, both domestically and internationally.
7. **Interest Calculation**: Automatically calculating interest for different types of accounts and loans.
8. **Employee Role Management**: Managing employee roles and responsibilities effectively.

By combining these parts into one system, we aim to provide a complete solution that helps banks manage their operations better and offer improved services to their customers. This system will make banking easier for everyone, reduce the need for manual work, and ensure high-quality service, helping to build customer loyalty and trust.

Functional requirements:

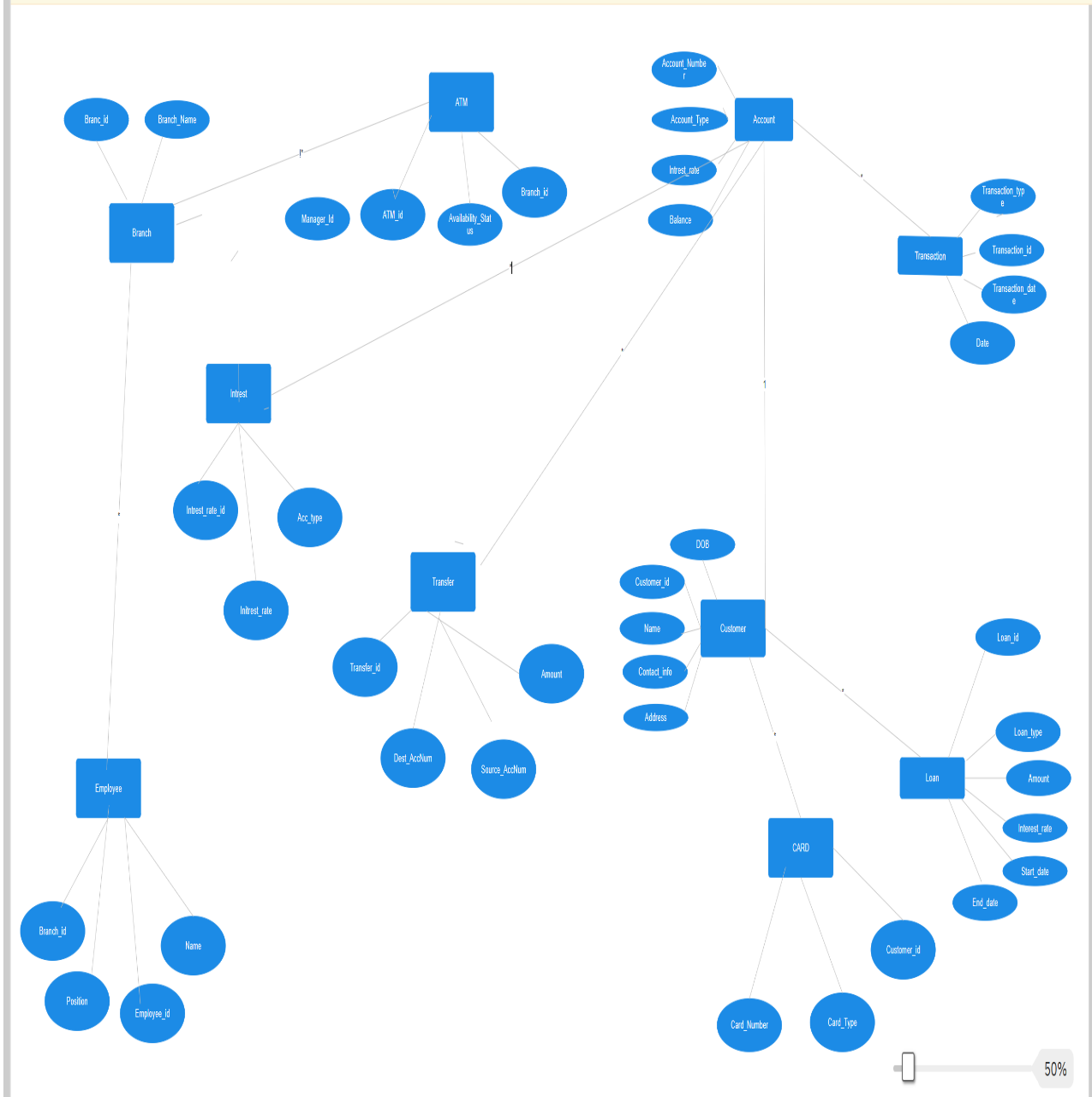
ER diagram:

An Entity-Relationship (ER) diagram illustrates the logical structure of a database system. It represents entities as rectangles, attributes within entities, and relationships between entities as lines connecting them. Entities represent real-world objects like customers or products, while relationships show how entities are connected. Cardinality and participation constraints describe the nature and degree of these relationships, aiding in database design and normalization.

Class diagram:

This diagram represents the static structure of the system by showing classes, their attributes, methods, and relationships. For instance, classes like Account, Customer, Transaction would be interconnected through associations depicting their relationships.

ER diagram:



Queries to create database:

* Create database team4;
* Use team4;
* Create table for account,transactions,customer,loan,branch,atm,card,transfer,interest,employee
* Insert 50 records into each table;
* Every table should consist at least one primary key;
* There should be relationship among the tables;
* Retrive all records from all the tables;
* Use aggregate functions and apply it on each table;

Class/UML attribute & methods diagram:

|  |
| --- |
| ACCOUNT |
| -accountId: int  -balance: float  -type: AccountType  -customerId: int |
| +getAccountId(): int  +getBalance(); float  +getType(): AccountType  +getCustomerId(): int |

|  |
| --- |
| TRANSACTION |
| -transactionId: int  -amount: float  -date: Date  -type: TransactionType  -sourceAccountId: int  -destinationAccountId: int |
| + Transaction(transactionId: int, amount: float, date: Date, type: TransactionType, sourceAccountId: int, destinationAccountId: int)  + getTransactionId(): int  + getAmount(): float  + setAmount(amount: float): void  + getDate(): Date  + setDate(date: Date): void  + getType(): TransactionType  + setType(type: TransactionType): void  + getSourceAccountId(): int  + setSourceAccountId(accountId: int): void  + getDestinationAccountId(): int  + setDestinationAccountId(accountId: int): void |

|  |
| --- |
| CUSTOMER |
| - customerId: int  - firstName: String  - lastName: String  - address: String  - phoneNumber: String  - accounts: List<Account> |
| + Customer(customerId: int, firstName: String, lastName: String, address: String, phoneNumber: String)  + getCustomerId(): int  + getFirstName(): String  + setFirstName(firstName: String): void  + getLastName(): String  + setLastName(lastName: String): void  + getAddress(): String  + setAddress(address: String): void  + getPhoneNumber(): String  + setPhoneNumber(phoneNumber: String): void  + getAccounts(): List<Account>  + addAccount(account: Account): void  + removeAccount(account: Account): void |

|  |
| --- |
| LOAN |
| - loanId: int  - amount: float  - interestRate: float  - duration: int  - customerid: int |
| + Loan(loanId: int, amount: float, interestRate: float, duration: int, customer: Customer)  + getLoanId(): int  + getAmount(): float  + setAmount(amount: float): void  + getInterestRate(): float  + setInterestRate(interestRate: float): void  + getDuration(): int  + setDuration(duration: int): void  + getCustomerid(): int  + setCustomer(customer: Customer): void  + calculateInterest(): float |

|  |
| --- |
| BRANCH |
| - branchId: int  - location: String  - employees: List<Employee>  - accounts: List<Account>  - atms: List<ATM> |
| + Branch(branchId: int, location: String)  + getBranchId(): int  + getLocation(): String  + setLocation(location: String): void  + getEmployees(): List<Employee>  + addEmployee(employee: Employee): void  + getAccounts(): List<Account>  + addAccount(account: Account): void  + getATMs(): List<ATM>  + addATM(atm: ATM): void |

|  |
| --- |
| ATM |
| - atmId: int  - location: String  - cashBalance: float |
| + ATM(atmId: int, location: String)  + getAtmId(): int  + getLocation(): String  + setLocation(location: String): void  + getCashBalance(): float  + setCashBalance(cashBalance: float): void  + withdraw(amount: float): void  + deposit(amount: float): void  + checkBalance(): float |

|  |
| --- |
| CARD |
| - cardId: int  - cardType: CardType  - expirationDate: Date  - customerId: int |
| + Card(cardId: int, cardType: CardType, expirationDate: Date, customerId: int)  + getCardId(): int  + setCardId(cardId: int): void  + getCardType(): CardType  + setCardType(cardType: CardType): void  + getExpirationDate(): Date  + setExpirationDate(expirationDate: Date): void  + getCustomerId(): int  + setCustomerId(customerId: int): void |

|  |
| --- |
| TRANSFER |
| - transferId: int  - sourceAccountId: int  - destinationAccountId: int  - amount: float  - date: Date |
| + Transfer(transferId: int, sourceAccountId: int, destinationAccountId: int, amount: float, date: Date)  + getTransferId(): int  + setTransferId(transferId: int): void  + getSourceAccountId(): int  + setSourceAccountId(sourceAccountId: int): void  + getDestinationAccountId(): int  + setDestinationAccountId(destinationAccountId: int): void  + getAmount(): float  + setAmount(amount: float): void  + getDate(): Date  + setDate(date: Date): void |

|  |
| --- |
| INTEREST |
| - interestId: int  - amount: float  - date: Date  - accountId: int |
| + Interest(interestId: int, amount: float, date: Date, accountId: int)  + getInterestId(): int  + setInterestId(interestId: int): void  + getAmount(): float  + setAmount(amount: float): void  + getDate(): Date  + setDate(date: Date): void  + getAccountId(): int  + setAccountId(accountId: int): void |

|  |
| --- |
| EMPLOYEE |
| - employeeId: int  - firstName: String  - lastName: String  - role: String  - branchId: int |
| + Employee(employeeId: int, firstName: String, lastName: String, role: String, branchId: int)  + getEmployeeId(): int  + setEmployeeId(employeeId: int): void  + getFirstName(): String  + setFirstName(firstName: String): void  + getLastName(): String  + setLastName(lastName: String): void  + getRole(): String  + setRole(role: String): void  + getBranchId(): int  + setBranchId(branchId: int): void |

All class java file codes:

1.Customer.java:

public class Customer {

private int customerId;

private String firstName;

private String lastName;

private String email;

private String phone;

// Constructor, getters, and setters

}

2.Account.java:

public class Account {

private int accountId;

private Customer customer;

private String accountType;

private double balance;

// Constructor, getters, and setters

}

3.Transaction.java:

import java.util.Date;

public class Transaction {

private int transactionId;

private Account account;

private String transactionType;

private double amount;

private Date transactionDate;

// Constructor, getters, and setters

}

4.Loan.java:

public class Loan {

private int loanId;

private Customer customer;

private double amount;

private double interestRate;

private int duration;

// Constructor, getters, and setters

}

5.Branch.java:

public class Branch {

private int branchId;

private String branchName;

private String location;

// Constructor, getters, and setters

}

6.ATM.java:

public class ATM {

private int atmId;

private Branch branch;

// Constructor, getters, and setters

}

7.Card.java:

public class Card {

private int cardNumber;

private Account account;

private String cardType;

// Constructor, getters, and setters

}

8Transfer.java:

public class Transfer {

private int transferId;

private Account senderAccount;

private Account receiverAccount;

private double amount;

// Constructor, getters, and setters

}

9.Interest.java:

public class Interest {

private double interestRate;

// Constructor, getters, and setters

}

10.Employee.java:

public class Employee {

private int employeeId;

private String firstName;

private String lastName;

private String position;

private Branch branch;

// Constructor, getters, and setters

}

Challenges list:

* Relationships in ER diagram
* Faced difficulties while creating the tables
* Foreign key issues
* Choosing data types
* Defining structure.