Project Report

Bank Database Management System

Course: CS257 Database and Information System

Team Members:	• **Aayush Yadav - 230001001** • **Divyam Maru - 230001025** • **Srujan Patel - 230001063** • **Nandini Kumari -230001056**

Project Details

Problem Definition

The **Banking Database Management System** is a comprehensive system designed to streamline banking operations and improve the customer experience. It manages vital tasks such as customer account management, loan tracking, transaction processing, employee and branch management, and credit card processing.

Problem Statement: Banks often face challenges related to managing large amounts of sensitive data while ensuring operational efficiency and customer satisfaction. This project aims to address these challenges by creating a robust database system to handle complex banking operations effectively, catering to both customers and employees.

Database Design (ER modelling)

The ER model is the foundation of the system, built to represent real-world entities and their relationships effectively. Below is an overview of the key entities and their attributes:

Branch

- BranchID (Primary Key)
- ManagerID
- BranchName
- Location
- PhoneNumber

Customer

- CustomerID (Primary Key)
- Firstname
- Middlename
- Lastname
- DateOfBirth
- Age
- Location
- PhoneNumber
- Email
- DateJoined
- Passwords
- SSN (Unique)

Account

- AccountID (Primary Key)
- CustomerID (Foreign Key -> Customer.CustomerID)
- BranchID (Foreign Key -> Branch.BranchID)
- AccountType
- Balance
- Interest
- InterestAddDateEffective
- DateOpened

Loan

- LoanID (Primary Key)
- CustomerID (Foreign Key -> Customer.CustomerID)
- LoanType
- PrincipleAmount
- InterestRate
- StartDate
- EndDate
- MonthlyPayment
- Collateral

Employee

- EmployeeID (Primary Key)
- BranchID (Foreign Key -> Branch.BranchID)

- Firstname
- Middlename
- Lastname
- Position
- Salary
- HireDate
- PhoneNumber

Transaction

- TransactionID (Primary Key)
- AccountID (Foreign Key -> Account.AccountID)
- BranchID (Foreign Key -> Branch.BranchID)
- TransactionType
- Amount
- Status
- TransactionDate
- TransactionTime
- Description
- TransactionFrom(Foreign Key -> Account.AccountID)
- TransactionTo(Foreign Key -> Account.AccountID)

University

- UniversityID (Primary Key)
- UniversityName
- Location
- PhoneNumber
- Website

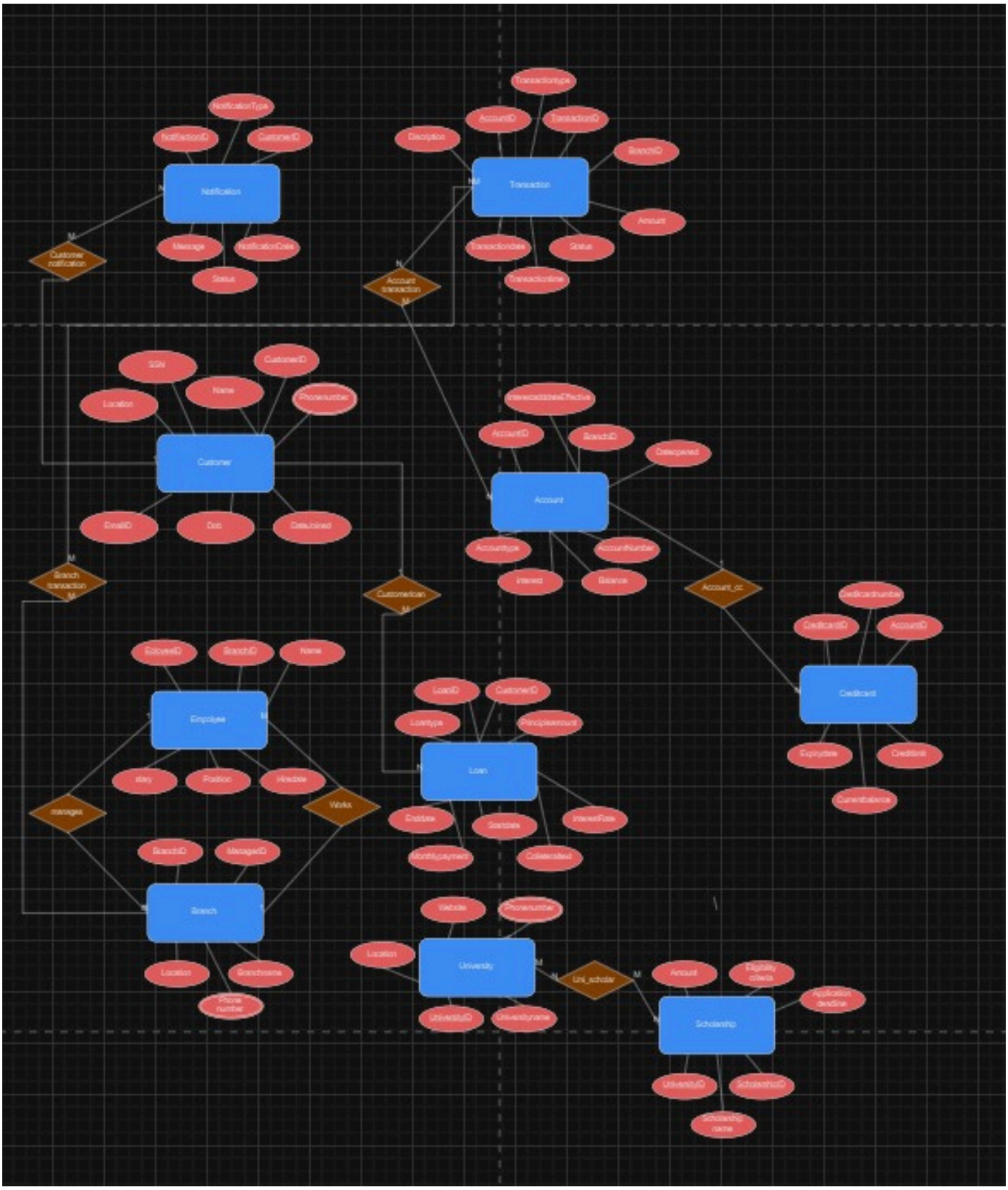
Scholarship

- ScholarshipID (Primary Key)
- UniversityID (Foreign Key -> University.UniversityID)
- ScholarshipName
- Amount
- EligibilityCriteria
- ApplicationDeadline

CreditCard

- CreditCardID (Primary Key)
- AccountID (Foreign Key -> Account.AccountID)
- CreditCardNumber
- ExpiryDate
- CreditLimit
- CurrentBalance

ER-Diagram



https://drive.google.com/file/d/1O8uSsGVstxxonvrOmFb1cKKvSy-v1oOU/view?usp=drive_link

This is the .drawio file that shows the ER diagram we have designed. Open

https://draw.io and upload the file there to view the ER diagram entirely.

Note: Due to the massivity of the ER diagram we have designed, providing screenshots was not feasible and we had to provide the link itself.

Design and Implementation of transactions

The system supports several transaction types essential to banking operations:

Transaction 1: User Registration

Registers a new user by storing their mobile number, email, and encrypted password in the database.

- **Validates the Input**: Checks if the mobile number, email, and password are provided and valid.
- **Encrypts the Password**: Hashes the password using bcrypt before storing it.
- **Creates the User Table**: If it doesn't exist, the users table is created.
- **Inserts the User Data**: The user's mobile, email, and hashed password are inserted into the users table.

Maintaining ACID Properties:

1. **Atomicity:** Ensured by treating the registration process as a single unit. If any step fails, the entire operation is rolled back, preventing partial data storage.
2. **Consistency:** Validated input ensures correct data format. The table schema is created if missing, and constraints maintain data integrity.
3. **Isolation:** Transactions are isolated from each other, ensuring that one registration does not interfere with another, even in concurrent operations.
4. **Durability:** Once committed, the data is permanently stored, protected by the database's transaction logs and recovery mechanisms, ensuring no data loss.

Transaction 2: Transfering Funds

Transfer Process:

- **Retrieve Sender Account Details**:
  Fetches the sender's account details (ID, balance, branch, and name) based on the provided senderAccountNumber.
- **Sender Information Validation**:
  Ensures that the sender's name matches the one provided in the request. If not, an error is thrown.
- **Validate Recipient Account**:
  Checks if the recipient's account exists using the provided recipientAccountNumber.
- **Balance Check**:
  Verifies if the sender has sufficient funds for the transaction. If the balance is insufficient, an error is thrown.
- **Transaction Execution**:
  - Debits the sender's account by the transfer amount.
  - Credits the recipient's account with the same amount.

- **Log Transaction**:
  Inserts a record of the transaction into the Transaction table, including details like the sender and recipient accounts, amount, transaction date and time, and description.

Maintaining ACID Properties

1. **Atomicity**: beginTransaction() and commit() ensure all operations are executed as a single unit. If any step fails, rollback() undoes all changes, maintaining atomicity.
2. **Consistency**: Transactions validate conditions (e.g., sufficient balance, account existence) to ensure the database remains consistent. Schema constraints also preserve data integrity.
3. **Isolation**: Each transaction operates independently, using the database's isolation level to prevent interference from concurrent operations, ensuring consistency.
4. **Durability**: Once committed, changes are permanently recorded, and the data remains intact even after a system crash, due to the database's logging mechanisms.

Transaction 3: Logging in

Verifies the user's credentials to allow access to the system.

- Validates the provided email and password.
- Retrieves the user data based on the email.
- Checks if the provided password matches the stored password.
- Logs the login attempt and the result.

Maintaining ACID Properties

1. **Atomicity:** The login process is atomic, ensuring that all steps succeed or none at all. If a failure occurs (e.g., incorrect credentials), no partial changes are made.
2. **Consistency:** Input validation ensures both email and password are provided, and the database query checks for valid credentials, maintaining data integrity.
3. **Isolation:** Each login attempt is isolated, preventing interference from concurrent requests and ensuring consistent access to user data.
4. **Durability:** Once the login process completes, the result is permanently recorded, ensuring data persistence even if a system crash occurs.

Transaction 4: Loan Application

Facilitates the process of applying for a loan.

- Validates the user's eligibility and the provided loan application details (e.g., loan amount, term, purpose).
- Checks the user's credit history, income, and other necessary criteria.
- Retrieves relevant financial data to evaluate the loan application.
- Updates the database with the loan application status (e.g., approved, pending, rejected).
- Logs the loan application attempt and the result.

Maintaining ACID Properties

1. **Atomicity:** The loan application process is atomic, ensuring that all steps succeed or none at all. For example, if validation or credit history checks fail, no partial data about the application is saved.
2. **Consistency:** Validation of user inputs, credit checks, and income verification ensures that the loan application adheres to business rules and maintains the integrity of the financial data.
3. **Isolation:** Each loan application is processed independently, ensuring that concurrent applications or database queries do not affect the evaluation or outcome of other loan applications.
4. **Durability:** Once the loan application is submitted and its status updated in the database, the changes are permanently recorded.

Transaction 4: Credit Card Application

Facilitates the process of applying for a credit card.

- Validates the applicant's details (e.g., name, age, income, employment, and address).
- Checks the applicant's credit history, credit score, and existing liabilities.
- Retrieves necessary financial and personal data to evaluate eligibility.
- Updates the database with the credit card application status (e.g., approved, pending, rejected).
- Logs the credit card application attempt and its outcome.

Maintaining ACID Properties

1. **Atomicity:** The credit card application process is atomic, ensuring that either all the steps are completed successfully or none are executed.
2. **Consistency:**
   Validation of applicant details, credit score, and financial status ensures the application adheres to the bank's policies and maintains data integrity in the database.
3. **Isolation:**
   Each credit card application is processed in isolation, ensuring that concurrent applications or other database operations do not affect the outcome or create data inconsistencies.
4. **Durability:**
   Once the credit card application is submitted and its status updated in the database, the changes are permanently recorded, ensuring data persistence even in the event of a system crash.

Transaction 5: Retrieving transaction history

For retrieving the transaction history the following steps are taken:

- **Validates** the provided accountNumber.
- **Retrieves** the accountId based on the accountNumber.
- **Fetches** the transaction history related to the retrieved accountId.

- **Classifies** transactions as either 'incoming' or 'outgoing' based on the transaction sender or receiver.
- **Logs** the transaction fetching attempt and any errors encountered.

Maintaining ACID Properties

1. **Atomicity**:
   The process ensures that both the account ID retrieval and transaction fetching succeed or fail together, preventing partial changes in case of errors.
2. **Consistency**:
   Input validation and database queries ensure data integrity, returning valid transaction history for existing accounts only.
3. **Isolation**:
   Transaction fetching is isolated, ensuring no interference from concurrent operations, maintaining consistent data retrieval.
4. **Durability**:
   Once the transaction history is fetched and returned, it is permanently recorded, maintaining integrity even in case of a system crash.