Banking Management System – Final Report

Abstract

This project aimed to design and implement a Java-based **Banking Management System** using JDBC for database interaction. The system supports the core functionalities expected from a banking software: managing customers, bank accounts, transactions, and loans. Through this project, we successfully developed a menu-driven console application that allows CRUD operations and query-based analysis. Key achievements include a robust JDBC connection manager, modularized code structure, and a relational schema that ensures data consistency and normalization. The implementation helped bridge theoretical database concepts with practical Java programming.

Introduction

The System

The Banking Management System was developed to satisfy the following key requirements:

1. **Customer Management**
    - Add new customers
    - Delete customers
    - View customer details
2. **Bank Account Management**
    - Open new accounts
    - Update account balances
    - Close accounts
    - View all accounts
3. **Transaction Handling**
    - Perform deposits and withdrawals
    - Track transaction history for each account
4. **Loan Management**
    - Issue loans
    - Display all loans and associated customers
5. **Database Operations**
    - JDBC-based DDL and DML interaction
    - Data validation and exception handling

The backend database includes four main tables: **Customer, BankAccount , BankTransaction**, and **Loan**.

Each module was designed to handle the respective entity using object-oriented principle.

Conclusion

This project enabled us to apply concepts of object-oriented programming, database normalization, and JDBC connectivity in a real-world banking context. We learned how to:

- Implement full CRUD operations across multiple entities
- Use JDBC to connect Java applications with a relational database
- Structure and normalize a banking-related database
- Develop a console-based, menu-driven system to interact with users

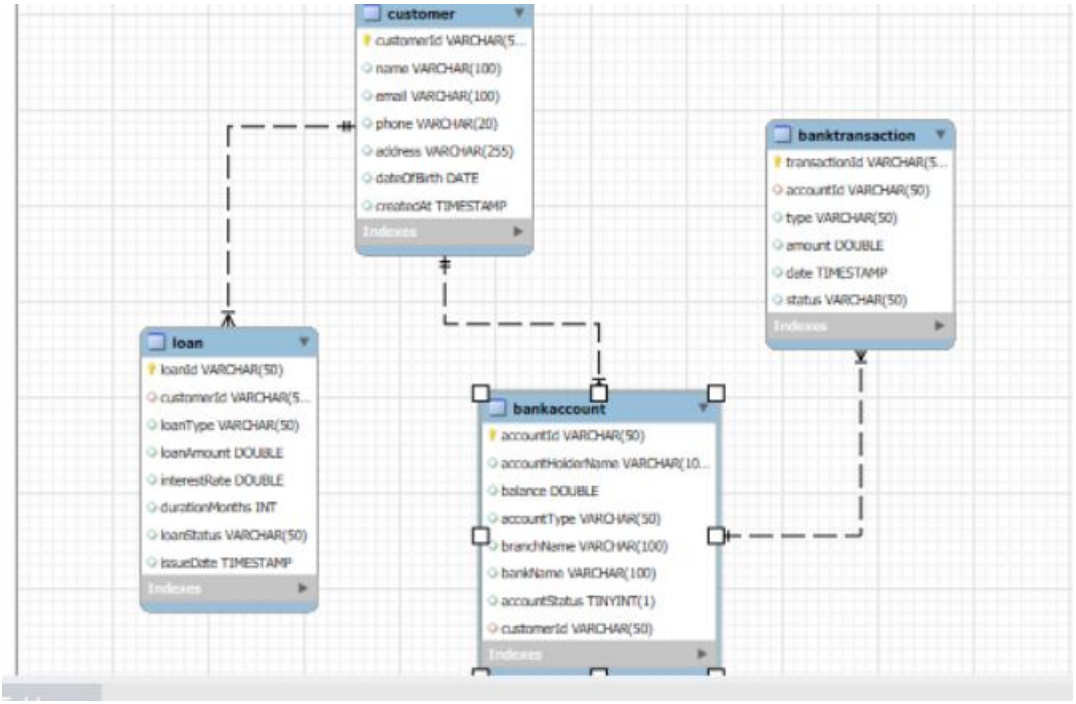
We also learned the importance of maintaining modular and reusable code, especially when dealing with database operations across multiple tables. Error handling, user input validation, and clear function separation became essential parts of the development process.

Bibliography

1. **Oracle JDBC Documentation** – https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/
    - Used to configure and manage JDBC connections.
2. **Java: The Complete Reference by Herbert Schildt**
    - Provided syntax help and object-oriented design techniques.
3. **GeeksForGeeks - JDBC in Java** – https://www.geeksforgeeks.org/jdbc-in-java/
    - Helped in understanding practical use-cases of JDBC.
4. **W3Schools - SQL Tutorial** – https://www.w3schools.com/sql/
    - Referenced frequently for creating and managing SQL tables.

Appendix

ER-Diagram



Source Code

Limit to 1000 rows

```sql
-- Create Database
CREATE DATABASE IF NOT EXISTS BankingManagementSystem;

-- Use the created database
USE BankingManagementSystem;

-- ============================================
-- Table: Customer
-- Stores personal details of customers.
-- ============================================
CREATE TABLE Customer (
    customerId VARCHAR(50) PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100),
    phone VARCHAR(20),
    address VARCHAR(255),
    dateOfBirth DATE,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- ============================================
```

```sql
-- Table: BankAccount
-- Stores bank account details for each customer.
-- ========================================
CREATE TABLE BankAccount (
    accountId VARCHAR(50) PRIMARY KEY,
    accountHolderName VARCHAR(100),
    balance DOUBLE,
    accountType VARCHAR(50),
    branchName VARCHAR(100),
    bankName VARCHAR(100),
    accountStatus BOOLEAN,
    customerId VARCHAR(50),
    FOREIGN KEY (customerId) REFERENCES Customer(customerId)
);

-- ========================================
-- Table: BankTransaction
-- Records transactions related to bank accounts.
-- ========================================
CREATE TABLE BankTransaction (
    transactionId VARCHAR(50) PRIMARY KEY,
    status VARCHAR(50),
    FOREIGN KEY (accountId) REFERENCES BankAccount(accountId)
);

-- ========================================
-- Table: Loan
-- Records loan details for customers.
-- ========================================
CREATE TABLE Loan (
    loanId VARCHAR(50) PRIMARY KEY,
    customerId VARCHAR(50),
    loanType VARCHAR(50),
    loanAmount DOUBLE,
    interestRate DOUBLE,
    durationMonths INT,
    loanStatus VARCHAR(50),
    issueDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (customerId) REFERENCES Customer(customerId)
);
```

Java Source Code Files

1.

Customer.java

2.

BankAccount.java

3.

BankTransaction.java

4.

Loan.java

5.

BankingDatabaseManager.java

6.

BankingApp.java

Brief User's Guide

1. Launch Application
   Run DatabaseBankingManagement class from the terminal or IDE.
2. Menu Options
   The system displays a menu:
   - 1. Add Customer
   - 2. Open Bank Account
   - 3. Perform Transaction
   - 4. Issue Loan
   - 5. View Reports
   - 6. Exit
3. User Input
   Follow on-screen prompts to provide input (e.g., customer name, account ID, transaction type).
4. Database Connection
   Ensure your MySQL database is running and configured with the proper credentials in your JDBC connection string.
5. Error Handling
   Input validations are in place to ensure data consistency and catch invalid operations (e.g., overdrafts, non-existent IDs).
