**End Term Project Report**

**On**

**Introduction to Databases (CSE 3151)**

**Submitted by**

**Name : Pooja Ghosh**

**Reg. No. : 2141007065**

**Branch : Computer Science and Engineering**

**Semester : 6th**

**Section : 26 – ‘Z’**

**Session : 2023-2024**

**Admission Batch : 2021**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**FACULTY OF ENGINEERING & TECHNOLOGY (ITER)**

**SIKSHA ‘O’ ANUSANDHAN DEEMED TO BE UNIVERSITY**

**BHUBANESWAR, ODISHA – 751030**

**CONTENTS**

Title Page i

1.Problem Statement 1

* 1. Objective 1
  2. Requisite 1
  3. Overview 1
  4. Project Description 2
  5. Program Skeleton 3
  6. Test Cases 6

2. Abstract 7

3. Introduction 8

4. System Overview 9

5. Implementation 11

6. Testing 21

7. Conclusion 26

**Laboratory Assignments Subject: Introduction to Databases**

**Subject code: CSE 3151**

**Assignment 6: Mini project using JDBC connectivity**

# Objective of this Assignment:

* To design a miniature Project for a Banking Management System using Java, Oracle and JDBC.

# Requisite:

* Completion of IDB Laboratory Assignment-4
* Basic Java Programming knowledge

**Overview of the Project:** A Banking Management System is to be designed, putting together the concepts learnt in theory and practised in laboratory. The Project will integrate a Java frontend menu driven program to the backend Banking Database designed in Oracle through JDBC connectivity.

**Project Description:** The Java program provides an interface to the user to access, insert, delete and update the database. The program handles user input, output to and from the database for the said operations. User should be able to do the following operations:

* 1. Show Customer Records:

Using this option the details of all the customers should be displayed in particular format.

* 1. Add Customer Record:

Using this option the user needs to provide the information such as cust\_no, name, phoneno and city through user input, which will be saved in database. After that using option 1, details of all the customers will be displayed in particular format.

* 1. Delete Customer Record:

Using this option the user needs to provide the cust\_no of a customer through user input and all the information related to that customer will be deleted from the database. After that using option 1, details of all the customers will be displayed in particular format.

* 1. Update Customer Information:

Using this option the user needs to provide the cust\_no of a customer through user input and based on the following choice the information related to the customer will be updated.

* + 1. : Update name
    2. : Update Phoneno.
    3. : Update city

After that using option 1, details of all the customers will be displayed in particular format.

* 1. Show Account Details of a Customer:

Using this option the user needs to provide the cust\_no of a customer through user input and all the information of that customer along with his account\_no, type, balance, branch\_code, branch\_name and branch\_city will be displayed in proper format.

* 1. Show Loan Details of a Customer:

Using this option the user needs to provide the cust\_no of a customer through user input and all the information of that customer along with his loan\_no, loan amount, branch\_code, branch\_name and branch\_city will be displayed in proper format.

* 1. Deposit Money to an Account:

Using this option the user needs to provide the account\_no of a customer and the amount to be deposited through user input. According to the deposited amount the updated balance will be verified in proper format using option 5.

* 1. Withdraw Money from an Account:

Using this option the user needs to provide the account\_no of a customer and the amount to be withdraw through user input. According to the withdraw amount the updated balance will be verified in proper format using option 5.

* 1. Exit the Program

The operations are choice based. Appropriate option has to be chosen from a switch case based menu driven program and the operation on the database is performed accordingly. The output is displayed in the terminal screen with appropriate messages from the database as displayed by Oracle during direct access. Exceptions should be handled properly by the Java program. The output should be displayed in a formatted way for clarity of understanding and visual.

# Program Skeleton:

import java.sql.\*; import java.io.\*;

public class myjdbcproj{

public static void main(String args[]) throws IOException

{

Connection con=null; Statement stmt=null;

// Declare common variables if any try{

// Load the driver class Class.forName("oracle.jdbc.driver.OracleDriver");

// Create the connection object

String conurl="jdbc:oracle:thin:@172.17.144.110:1521:ora11g"; con=DriverManager.getConnection(conurl,"<user ID>","<user password>"); stmt=con.createStatement();

do

{

System.out.println("\n\n\*\*\*\*\* Banking Management System\*\*\*\*\*");

// Display the menu

System.out.println("Enter your choice(1-9):");

// Accept user's choice

switch(choice\_variable)

{

case 1:

// Display customer records break;

case 2:

// Add customer record

// Accept input for each column from user

break; case 3:

// Delete customer record

// Accept customer number from user break;

case 4:

// Update customer record

// Accept customer number from user

System.out.println("Enter 1: For Name 2: For Phone no 3: For City to update:");

// Accept user's choice switch(choice\_variable\_1)

{

case 1:

// Update customer's name break;

case 2:

// Update customer's phone number

break; case 3:

// Update customer's city break;

}

break; case 5:

// Display account details

// Accept customer number from user break;

case 6:

// Display loan details

// Accept customer number from user

// Display the number of loans the customer has or

// Congratulation if he customer has no loan break;

case 7:

//Deposit money

// Accept the account number to be deposited in

// Message for transaction completion break;

case 8:

//Withdraw money

// Accept the account number to be withdrawn from

// Handle appropriate withdral ckeck conditions

// Message for transaction completion break;

case 9:

// Exit the menu break;

default:

// Handle wrong choice of option

}

}while(condition);

} //try closing catch(Exception e)

{ // Handling exception}

}// main closing

}// End class

# Test Cases:

The program should able to produce correct answer or appropriate error message corresponding to the following testcases:

1. Show Customer Records
2. Add Customer Record: <C0011, ANWESHA DAS, 9999999999, BHUB>, < c0012, SACHIN SINGH, 9898989898, CTC>, <C0013, ARJUN MISHRA, 7777777777, BBSR>
3. Delete Customer Record: <C0013>, <C0016>
4. Update Customer Record for any attribute except Customer Number: <C0011> [ Update each column once]
5. Show Account Details of a Customer: <C0003>, <c0005>, <C0016>
6. Show Loan Details of a Customer: <C0003>, <c0005>, <C0008>, <C0016>
7. Deposit Money to an Account: <A0008, 800>, <a0005, 10000>

8. Withdraw Money from an Account: <A0008, 800>, <A0008, 8000><a0005, 10000>

1. Exit the Program
2. Enter choice 10

**ABSTRACT**

This project report details the development of a miniature Banking Management System using Java for the frontend, Oracle for database management, and JDBC for connectivity. The core objective of this project is to create an integrated system that combines theoretical knowledge and practical skills acquired in laboratory exercises. The system is designed to perform essential banking operations such as managing customer records and handling account transactions, which include viewing, adding, deleting, and updating customer information, as well as managing deposits and withdrawals. The project demonstrates effective use of Java programming to create a user-friendly interface, while leveraging Oracle's robust database capabilities to ensure reliable data storage and retrieval. Through the JDBC connectivity, the project showcases efficient interaction between the application and the database. The successful implementation of this project highlights the seamless integration of different technologies to solve real-world problems in banking management.

**INTRODUCTION**

In the contemporary banking industry, efficient data management and transaction handling are paramount. Banks require robust systems to manage customer data and financial transactions to maintain operational efficiency and customer satisfaction. This project aims to design and implement a Banking Management System (BMS) that utilizes Java for the frontend development, Oracle for database management, and JDBC for database connectivity. This project builds upon the foundational knowledge gained, aiming to integrate practical programming skills with database management concepts.

The system is designed to be user-friendly and menu-driven, allowing users to perform a variety of banking operations with ease. By leveraging Java's capabilities for creating interactive interfaces and Oracle's powerful database management features, the BMS facilitates efficient management of customer records and account details. The inclusion of JDBC ensures that the communication between the Java application and the Oracle database is seamless and efficient.

This project not only focuses on the technical implementation but also on ensuring data integrity and consistency through proper validation and exception handling. The functionalities provided by the system, such as adding, viewing, updating, and deleting customer records, as well as managing financial transactions, are critical for the daily operations of a bank. By addressing these needs, the project demonstrates the practical application of theoretical concepts in a real-world scenario, providing a comprehensive solution for banking management.

The introduction of this Banking Management System is a step towards automating banking operations, thereby reducing manual errors, enhancing data accuracy, and improving overall efficiency. The project serves as a testament to the effective integration of software development and database management, offering a scalable solution that can be expanded and customized to meet specific banking requirements.

**SYSTEM OVERVIEW (Architecture and Functionality)**

**Architecture**

The Banking Management System is designed with a layered architecture, comprising the following components:

1. **User Interface Layer:** Developed using Java, this layer interacts with the user, capturing input and displaying output. The interface is menu-driven, allowing users to select various banking operations.

2**. Application Logic Layer:** This layer contains the business logic to process user inputs, validate data, and execute necessary operations. It handles the CRUD (Create, Read, Update, Delete) operations on customer records and manages transactions.

3. **Database Layer:** Implemented using Oracle, this layer stores all customer and account-related data. The database schema includes tables for customer details, account information, and transaction records.

4. **Connectivity Layer:** JDBC (Java Database Connectivity) serves as the middleware, enabling communication between the Java application and the Oracle database. It ensures that SQL queries are executed and results are retrieved efficiently.

**Functionality**

The Banking Management System offers the following functionalities:

1. **Show Customer Records:** Displays details of all customers in a specified format.

2. **Add Customer Record:** Allows the user to add a new customer by providing customer number, name, phone number, and city. The new record is saved in the database.

3. **Delete Customer Record:** Enables the user to delete a customer’s record by providing the customer number. All related information is removed from the database.

4. **Update Customer Information:** Allows updating specific customer details like name, phone number, or city by providing the customer number.

5. **Show Account Details of a Customer:** Displays account details for a specific customer, including customer number, account type, balance, branch code, branch name, and branch city.

6. **Show Loan Details of a Customer:** Displays loan information for a specific customer, including loan number, loan amount, branch code, branch name, and branch city.

7. **Deposit Money to an Account:** Enables depositing a specified amount into a customer’s account and updates the balance.

8. **Withdraw Money from an Account:** Allows withdrawal of a specified amount from a customer’s account and updates the balance.

9. **Exit the Program:** Provides an option to exit the application.

**IMPLEMENTATION**

**CODE:**

import java.sql.\*;

import java.util.Scanner;

public class BankingManagement {

private static final String URL = "jdbc:oracle:thin:@localhost:1521:orcl";

private static final String USER = "system";

private static final String PASSWORD = "Pooja@1008";

public static void main(String[] args) {

// TODO Auto-generated method stub

Scanner scanner=new Scanner(System.in);

try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD)) {

createTables();

int flag=1;

int userch;

while(flag==1) {

addCustomer(scanner);

addAccount(scanner);

addLoan(scanner);

System.out.println("Added Succesfully.");

userch=scanner.nextInt();

if(userch==0) {

flag=0;

}

}

while (true) {

System.out.println("1. Show Customer Records");

System.out.println("2. Add Customer Record");

System.out.println("3. Delete Customer Record");

System.out.println("4. Update Customer Information");

System.out.println("5. Show Account Details of a Customer");

System.out.println("6. Show Loan Details of a Customer");

System.out.println("7. Deposit Money to an Account");

System.out.println("8. Withdraw Money from an Account");

System.out.println("9. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

showCustomerRecords(connection);

break;

case 2:

addCustomerRecord(connection, scanner);

break;

case 3:

deleteCustomerRecord(connection, scanner);

break;

case 4:

updateCustomerInformation(connection, scanner);

break;

case 5:

showAccountDetails(connection, scanner);

break;

case 6:

showLoanDetails(connection, scanner);

break;

case 7:

depositMoney(connection, scanner);

break;

case 8:

withdrawMoney(connection, scanner);

break;

case 9:

System.out.println("Exiting...");

return;

default:

System.out.println("Invalid choice. Please try again.");

}

}

}

catch (SQLException e) {

e.printStackTrace();

}

}

public static void createTables() throws SQLException {

try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);

Statement stmt = conn.createStatement()) {

// Created CUSTOMER table

String createCustomerTable = "CREATE TABLE CUSTOMER01 ("

+ "CUST\_NO INT PRIMARY KEY, "

+ "NAME VARCHAR(100), "

+ "PHONENO VARCHAR(15), "

+ "CITY VARCHAR(50)"

+ ")";

stmt.executeUpdate(createCustomerTable);

System.out.println("CUSTOMER table created successfully.");

// Created ACCOUNT table

String createAccountTable = "CREATE TABLE ACCOUNT01 ("

+ "ACCOUNT\_NO INT PRIMARY KEY, "

+ "CUST\_NO INT, "

+ "TYPE VARCHAR(20), "

+ "BALANCE DECIMAL(15, 2), "

+ "BRANCH\_CODE VARCHAR(10), "

+ "BRANCH\_NAME VARCHAR(100), "

+ "BRANCH\_CITY VARCHAR(50), "

+ "FOREIGN KEY (CUST\_NO) REFERENCES CUSTOMER01(CUST\_NO)"

+ ")";

stmt.executeUpdate(createAccountTable);

System.out.println("ACCOUNT table created successfully.");

// Created LOAN table

String createLoanTable = "CREATE TABLE LOAN01 ("

+ "LOAN\_NO INT PRIMARY KEY, "

+ "CUST\_NO INT, "

+ "LOAN\_AMOUNT DECIMAL(15, 2), "

+ "BRANCH\_CODE VARCHAR(10), "

+ "BRANCH\_NAME VARCHAR(100), "

+ "BRANCH\_CITY VARCHAR(50), "

+ "FOREIGN KEY (CUST\_NO) REFERENCES CUSTOMER(CUST\_NO)"

+ ")";

stmt.executeUpdate(createLoanTable);

System.out.println("LOAN table created successfully.");

} catch (SQLException e) {

e.printStackTrace();

throw e;

}

}

//Method to add a customer--

public static void addCustomer(Scanner scanner) throws SQLException {

System.out.print("Enter Customer No: ");

int custNo = scanner.nextInt();

scanner.nextLine(); // consume newline

System.out.print("Enter Name: ");

String name = scanner.nextLine();

System.out.print("Enter Phone No: ");

String phoneNo = scanner.nextLine();

System.out.print("Enter City: ");

String city = scanner.nextLine();

String query = "INSERT INTO CUSTOMER01 (CUST\_NO, NAME, PHONENO, CITY) VALUES (?, ?, ?, ?)";

try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);

PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setInt(1, custNo);

pstmt.setString(2, name);

pstmt.setString(3, phoneNo);

pstmt.setString(4, city);

pstmt.executeUpdate();

System.out.println("Customer added successfully.");

}

}

// Method to add an account--

public static void addAccount(Scanner scanner) throws SQLException {

System.out.print("Enter Account No: ");

int accountNo = scanner.nextInt();

System.out.print("Enter Customer No: ");

int custNo = scanner.nextInt();

scanner.nextLine(); // consume newline

System.out.print("Enter Account Type: ");

String type = scanner.nextLine();

System.out.print("Enter Balance: ");

double balance = scanner.nextDouble();

scanner.nextLine(); // consume newline

System.out.print("Enter Branch Code: ");

String branchCode = scanner.nextLine();

System.out.print("Enter Branch Name: ");

String branchName = scanner.nextLine();

System.out.print("Enter Branch City: ");

String branchCity = scanner.nextLine();

String query = "INSERT INTO ACCOUNT01 (ACCOUNT\_NO, CUST\_NO, TYPE, BALANCE, BRANCH\_CODE, BRANCH\_NAME, BRANCH\_CITY) VALUES (?, ?, ?, ?, ?, ?, ?)";

try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);

PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setInt(1, accountNo);

pstmt.setInt(2, custNo);

pstmt.setString(3, type);

pstmt.setDouble(4, balance);

pstmt.setString(5, branchCode);

pstmt.setString(6, branchName);

pstmt.setString(7, branchCity);

pstmt.executeUpdate();

System.out.println("Account added successfully.");

}

}

// Method to add a loan--

public static void addLoan(Scanner scanner) throws SQLException {

System.out.print("Enter Loan No: ");

int loanNo = scanner.nextInt();

System.out.print("Enter Customer No: ");

int custNo = scanner.nextInt();

System.out.print("Enter Loan Amount: ");

double loanAmount = scanner.nextDouble();

scanner.nextLine(); // consume newline

System.out.print("Enter Branch Code: ");

String branchCode = scanner.nextLine();

System.out.print("Enter Branch Name: ");

String branchName = scanner.nextLine();

System.out.print("Enter Branch City: ");

String branchCity = scanner.nextLine();

String query = "INSERT INTO LOAN01 (LOAN\_NO, CUST\_NO, LOAN\_AMOUNT, BRANCH\_CODE, BRANCH\_NAME, BRANCH\_CITY) VALUES (?, ?, ?, ?, ?, ?)";

try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);

PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setInt(1, loanNo);

pstmt.setInt(2, custNo);

pstmt.setDouble(3, loanAmount);

pstmt.setString(4, branchCode);

pstmt.setString(5, branchName);

pstmt.setString(6, branchCity);

pstmt.executeUpdate();

System.out.println("Loan added successfully.");

}

}

// Method to show customer records--

public static void showCustomerRecords(Connection conn) throws SQLException {

String query = "SELECT \* FROM CUSTOMER01";

try (Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(query)) {

while (rs.next()) {

System.out.println("CUST\_NO: " + rs.getInt("CUST\_NO"));

System.out.println("NAME: " + rs.getString("NAME"));

System.out.println("PHONENO: " + rs.getString("PHONENO"));

System.out.println("CITY: " + rs.getString("CITY"));

System.out.println("-----------------------");

}

}

}

// Method to add a customer record

public static void addCustomerRecord(Connection conn, Scanner scanner) throws SQLException {

System.out.print("Enter Customer No: ");

int custNo = scanner.nextInt();

scanner.nextLine(); // consume newline

System.out.print("Enter Name: ");

String name = scanner.nextLine();

System.out.print("Enter Phone No: ");

String phoneNo = scanner.nextLine();

System.out.print("Enter City: ");

String city = scanner.nextLine();

String query = "INSERT INTO CUSTOMER01 (CUST\_NO, NAME, PHONENO, CITY) VALUES (?, ?, ?, ?)";

try (PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setInt(1, custNo);

pstmt.setString(2, name);

pstmt.setString(3, phoneNo);

pstmt.setString(4, city);

pstmt.executeUpdate();

System.out.println("Customer added successfully.");

}

}

// Method to delete a customer record

public static void deleteCustomerRecord(Connection conn, Scanner scanner) throws SQLException {

System.out.print("Enter Customer No to delete: ");

int custNo = scanner.nextInt();

String query = "DELETE FROM CUSTOMER01 WHERE CUST\_NO = ?";

try (PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setInt(1, custNo);

int rows = pstmt.executeUpdate();

if (rows > 0) {

System.out.println("Customer deleted successfully.");

} else {

System.out.println("Customer not found.");

}

}

}

// Method to update customer information

public static void updateCustomerInformation(Connection conn, Scanner scanner) throws SQLException {

System.out.print("Enter Customer No to update: ");

int custNo = scanner.nextInt();

scanner.nextLine(); // consume newline

System.out.println("1. Update Name");

System.out.println("2. Update Phone No");

System.out.println("3. Update City");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

scanner.nextLine(); // consume newline

String query;

String newValue;

switch (choice) {

case 1:

System.out.print("Enter new Name: ");

newValue = scanner.nextLine();

query = "UPDATE CUSTOMER01 SET NAME = ? WHERE CUST\_NO = ?";

break;

case 2:

System.out.print("Enter new Phone No: ");

newValue = scanner.nextLine();

query = "UPDATE CUSTOMER01 SET PHONENO = ? WHERE CUST\_NO = ?";

break;

case 3:

System.out.print("Enter new City: ");

newValue = scanner.nextLine();

query = "UPDATE CUSTOMER01 SET CITY = ? WHERE CUST\_NO = ?";

break;

default:

System.out.println("Invalid choice. Returning to main menu.");

return;

}

try (PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setString(1, newValue);

pstmt.setInt(2, custNo);

pstmt.executeUpdate();

System.out.println("Customer information updated successfully.");

}

}

// Method to show account details of a customer

public static void showAccountDetails(Connection conn, Scanner scanner) throws SQLException {

System.out.print("Enter Customer No: ");

int custNo = scanner.nextInt();

String query = "SELECT \* FROM ACCOUNT01 WHERE CUST\_NO = ?";

try (PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setInt(1, custNo);

try (ResultSet rs = pstmt.executeQuery()) {

while (rs.next()) {

System.out.println("ACCOUNT\_NO: " + rs.getInt("ACCOUNT\_NO"));

System.out.println("TYPE: " + rs.getString("TYPE"));

System.out.println("BALANCE: " + rs.getDouble("BALANCE"));

System.out.println("BRANCH\_CODE: " + rs.getString("BRANCH\_CODE"));

System.out.println("BRANCH\_NAME: " + rs.getString("BRANCH\_NAME"));

System.out.println("BRANCH\_CITY: " + rs.getString("BRANCH\_CITY"));

System.out.println("-----------------------");

}

}

}

}

// Method to show loan details of a customer

public static void showLoanDetails(Connection conn, Scanner scanner) throws SQLException {

System.out.print("Enter Customer No: ");

int custNo = scanner.nextInt();

String query = "SELECT \* FROM LOAN01 WHERE CUST\_NO = ?";

try (PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setInt(1, custNo);

try (ResultSet rs = pstmt.executeQuery()) {

while (rs.next()) {

System.out.println("LOAN\_NO: " + rs.getInt("LOAN\_NO"));

System.out.println("LOAN\_AMOUNT: " + rs.getDouble("LOAN\_AMOUNT"));

System.out.println("BRANCH\_CODE: " + rs.getString("BRANCH\_CODE"));

System.out.println("BRANCH\_NAME: " + rs.getString("BRANCH\_NAME"));

System.out.println("BRANCH\_CITY: " + rs.getString("BRANCH\_CITY"));

System.out.println("-----------------------");

}

}

}

}

// Method to deposit money to an account

public static void depositMoney(Connection conn, Scanner scanner) throws SQLException {

System.out.print("Enter Account No: ");

int accountNo = scanner.nextInt();

System.out.print("Enter amount to deposit: ");

double amount = scanner.nextDouble();

String query = "UPDATE ACCOUNT01 SET BALANCE = BALANCE + ? WHERE ACCOUNT\_NO = ?";

try (PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setDouble(1, amount);

pstmt.setInt(2, accountNo);

int rows = pstmt.executeUpdate();

if (rows > 0) {

System.out.println("Deposit successful.");

} else {

System.out.println("Account not found.");

}

}

}

// Method to withdraw money from an account

public static void withdrawMoney(Connection conn, Scanner scanner) throws SQLException {

System.out.print("Enter Account No: ");

int accountNo = scanner.nextInt();

System.out.print("Enter amount to withdraw: ");

double amount = scanner.nextDouble();

String checkBalanceQuery = "SELECT BALANCE FROM ACCOUNT01 WHERE ACCOUNT\_NO = ?";

try (PreparedStatement checkPstmt = conn.prepareStatement(checkBalanceQuery)) {

checkPstmt.setInt(1, accountNo);

try (ResultSet rs = checkPstmt.executeQuery()) {

if (rs.next()) {

double balance = rs.getDouble("BALANCE");

if (balance >= amount) {

String query = "UPDATE ACCOUNT01 SET BALANCE = BALANCE - ? WHERE ACCOUNT\_NO = ?";

try (PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setDouble(1, amount);

pstmt.setInt(2, accountNo);

pstmt.executeUpdate();

System.out.println("Withdrawal successful.");

}

} else {

System.out.println("Insufficient balance.");

}

} else {

System.out.println("Account not found.");

}

}

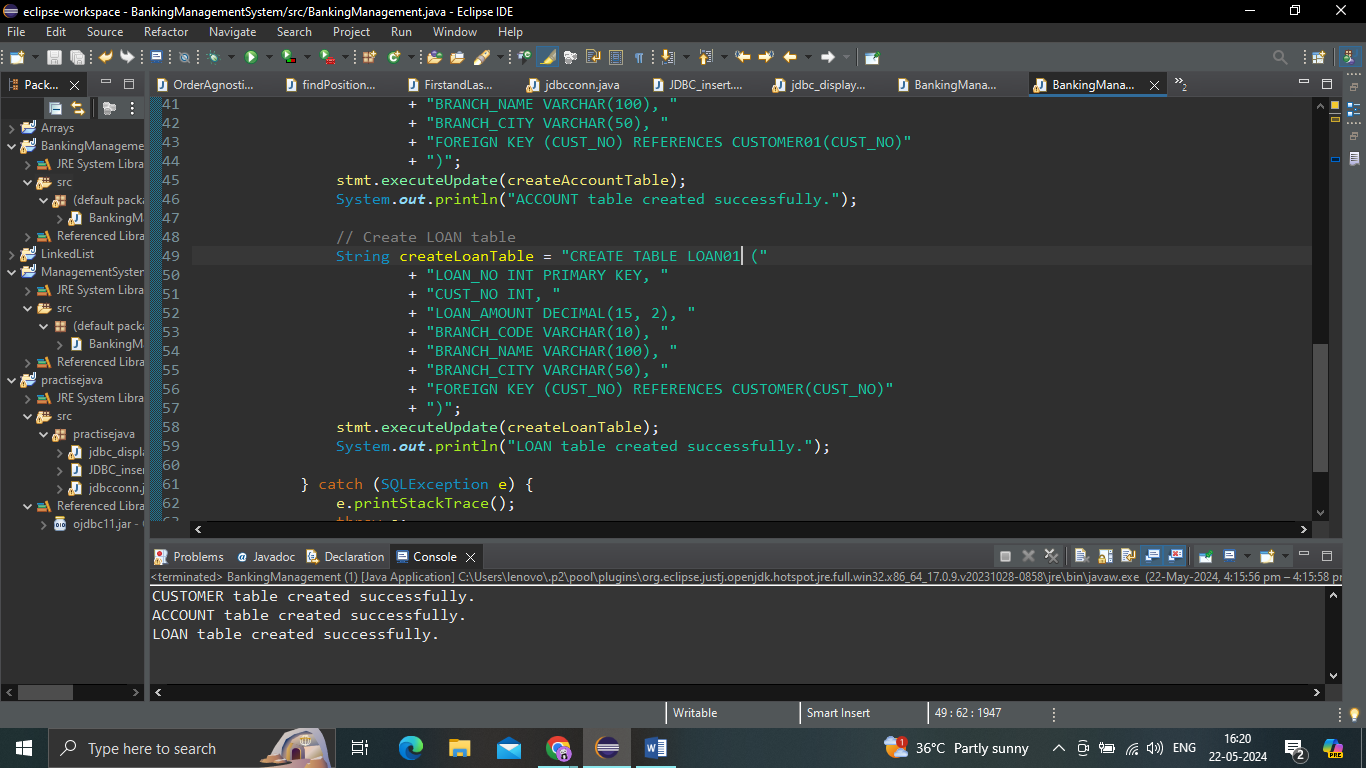
}

}

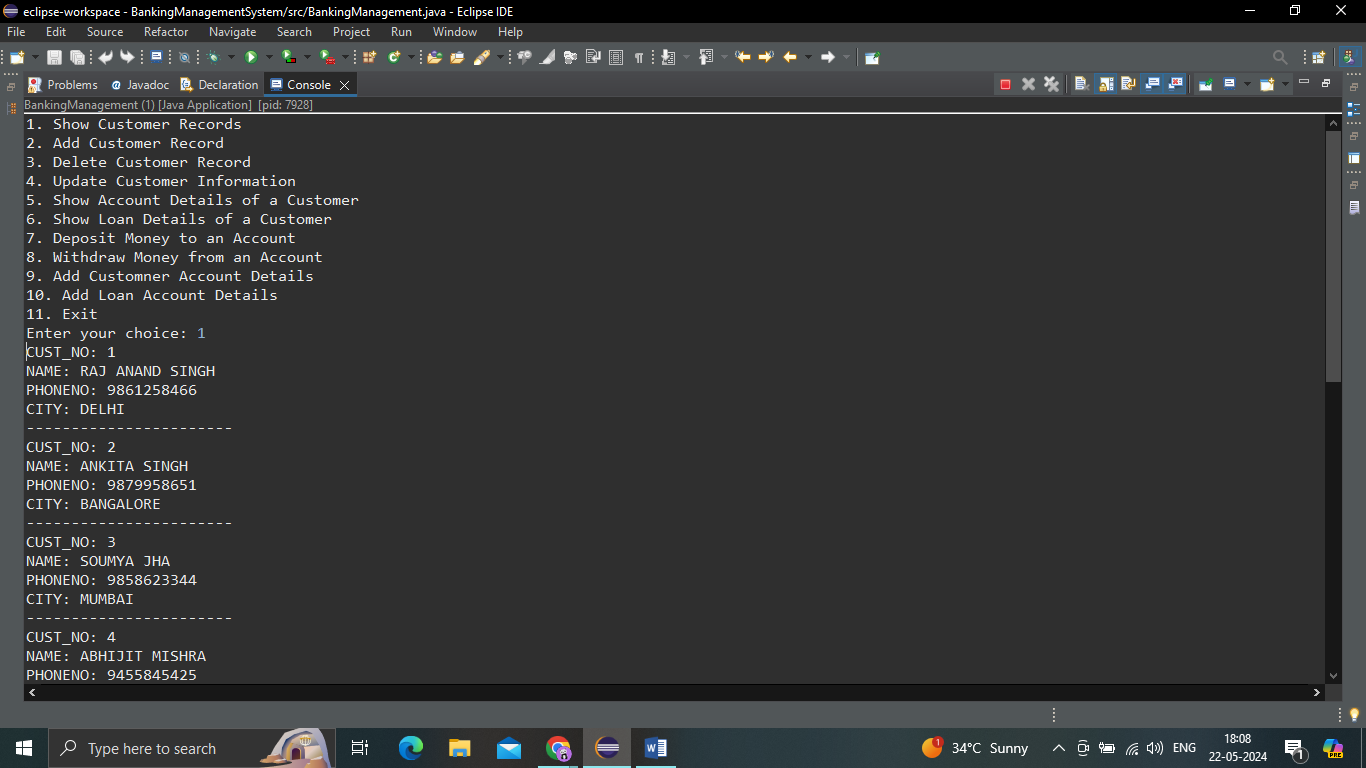
}

**TESTING**

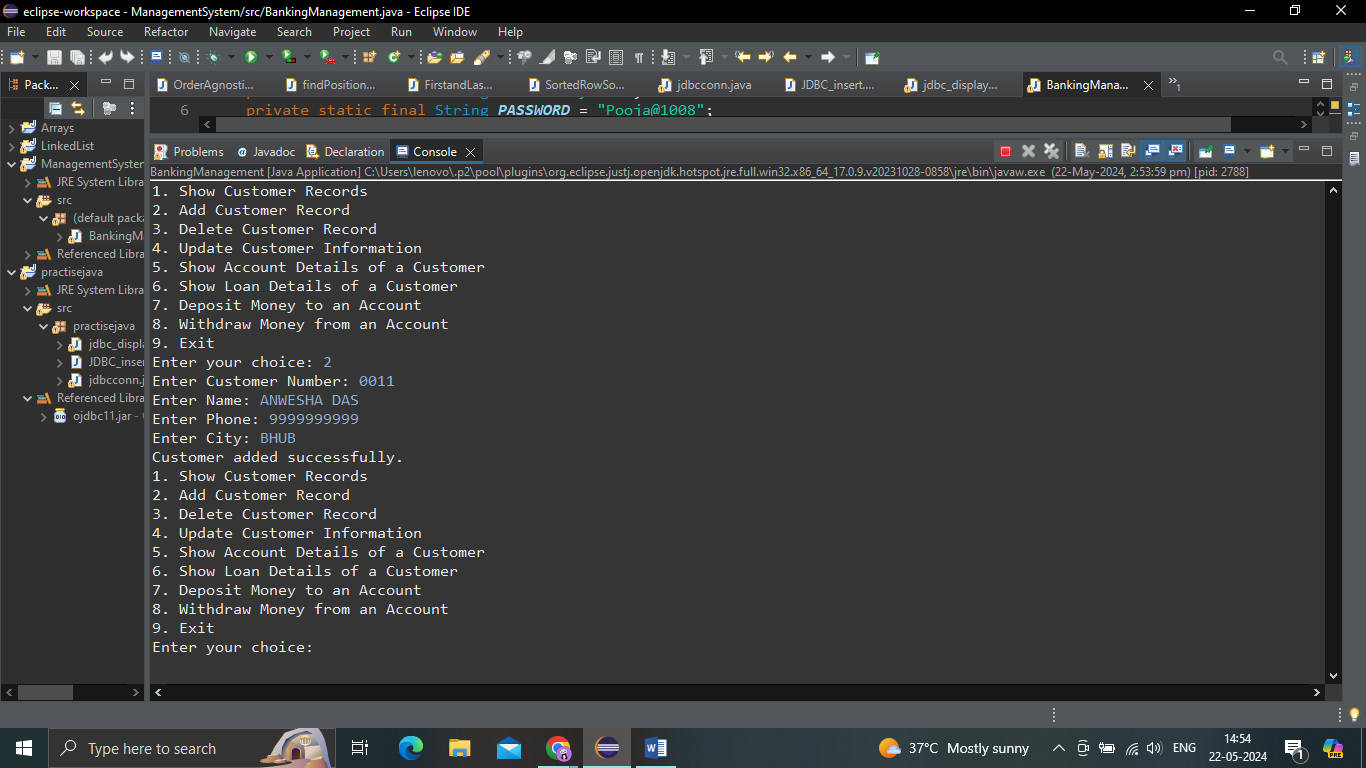
TABLE CREATION



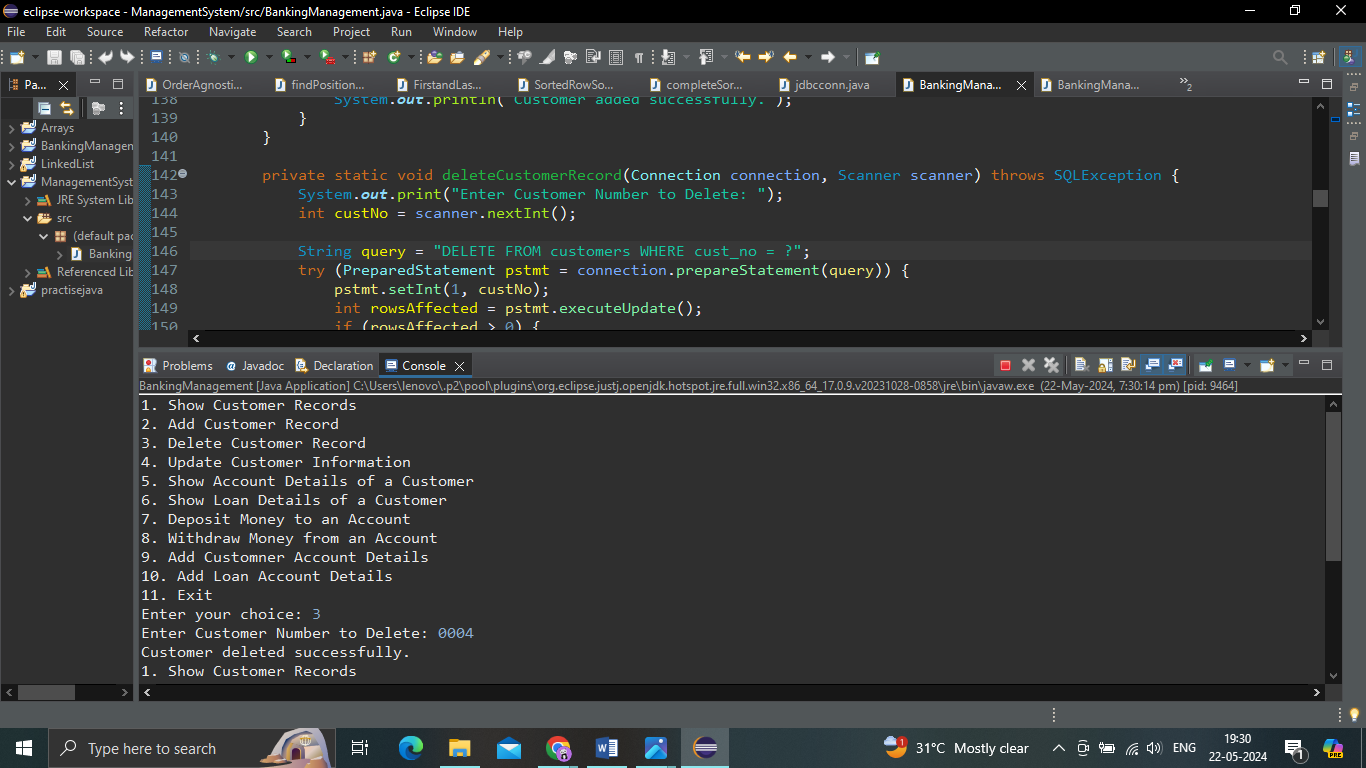
TEST CASE 1: Show Customer Records



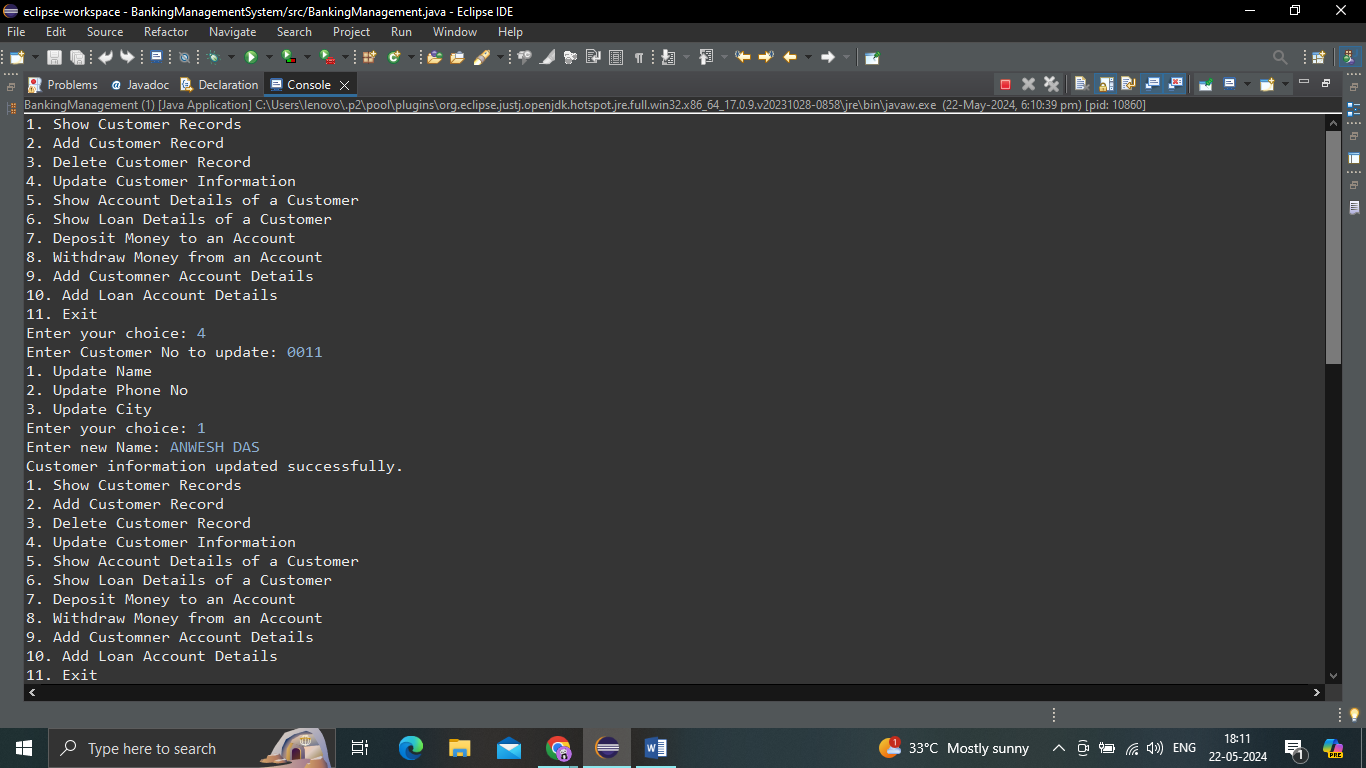
TEST CASE 2. Add Customer Record

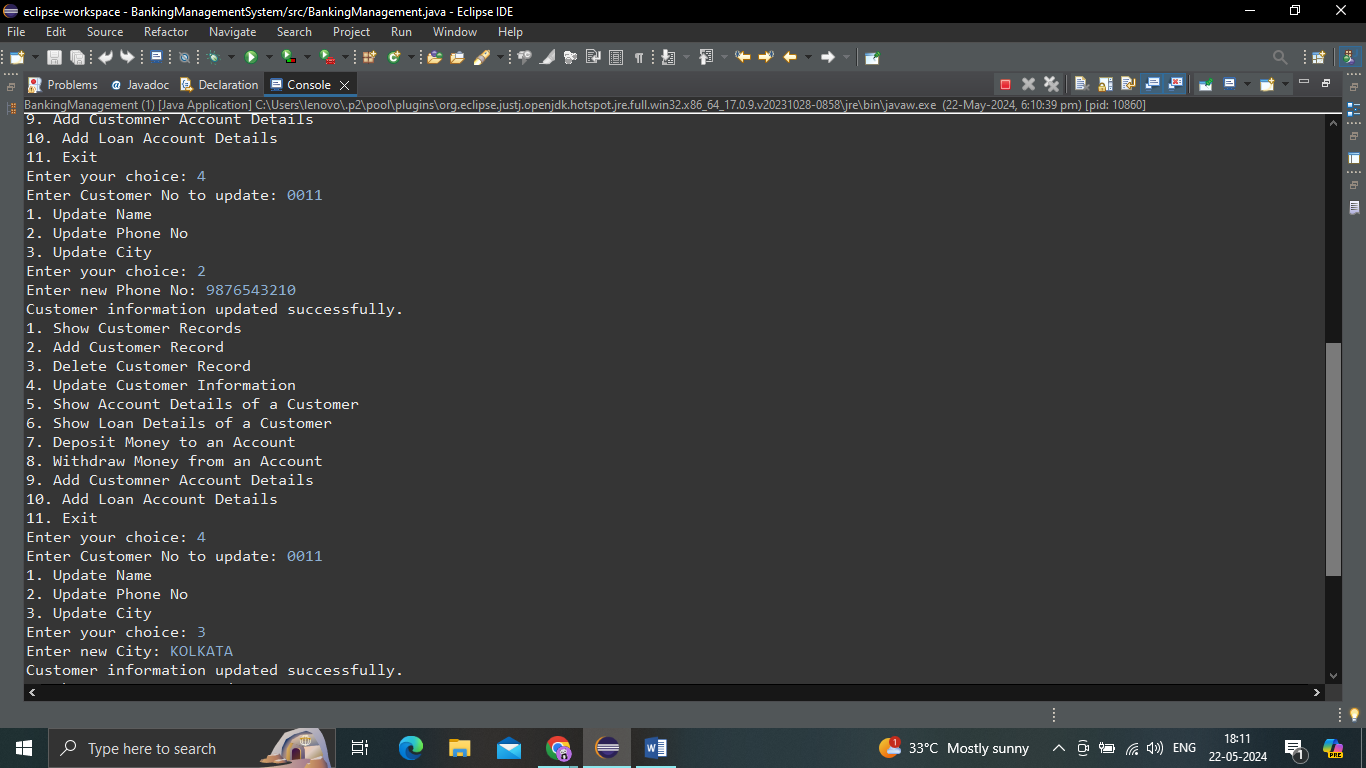


TEST CASE 3. Delete Customer Record

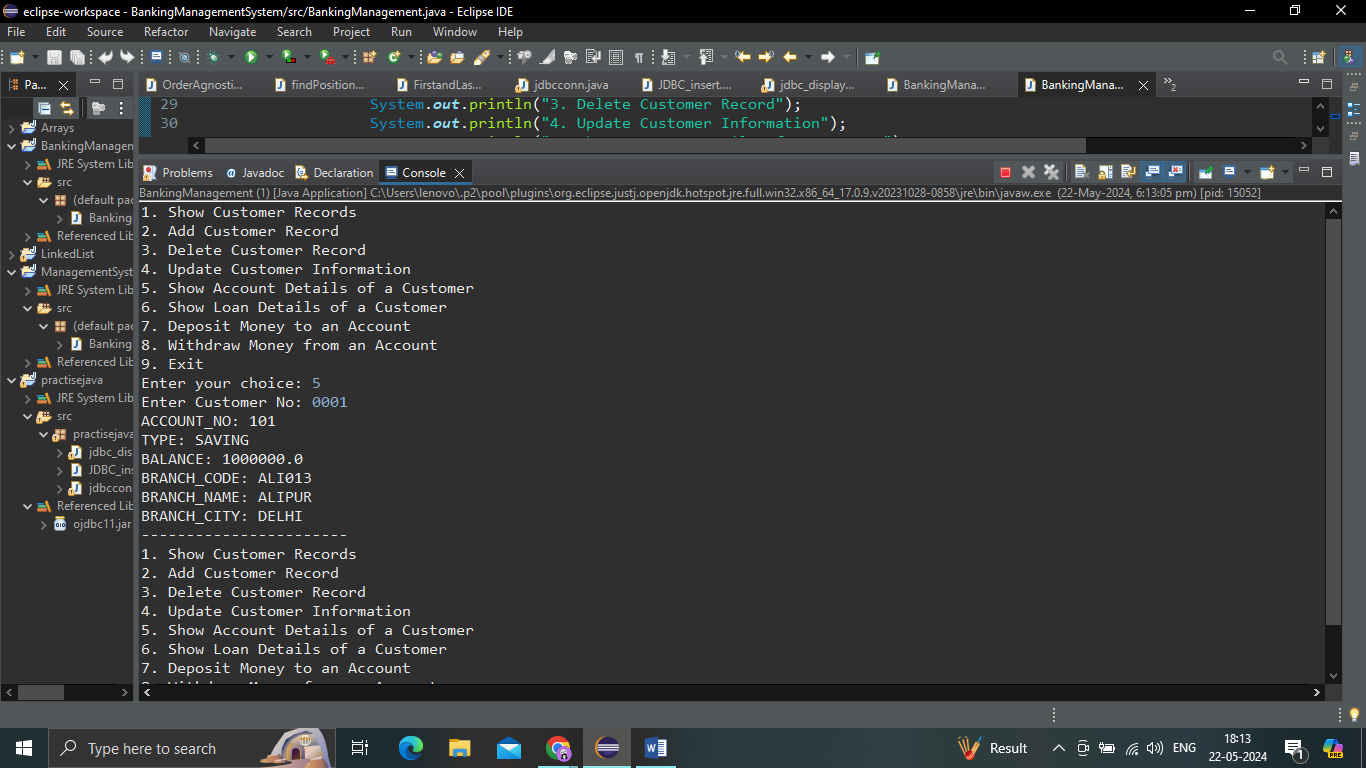


TEST CASE 4. Update Customer Information

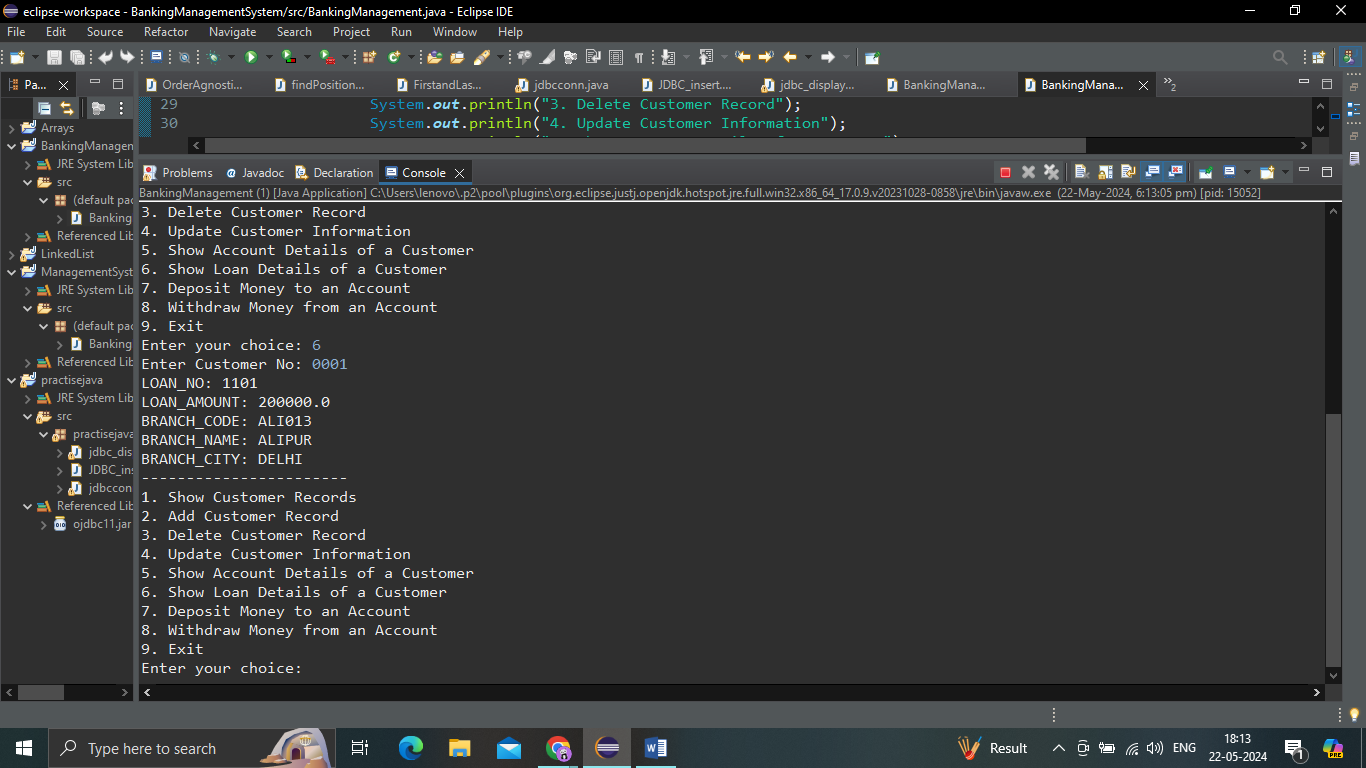




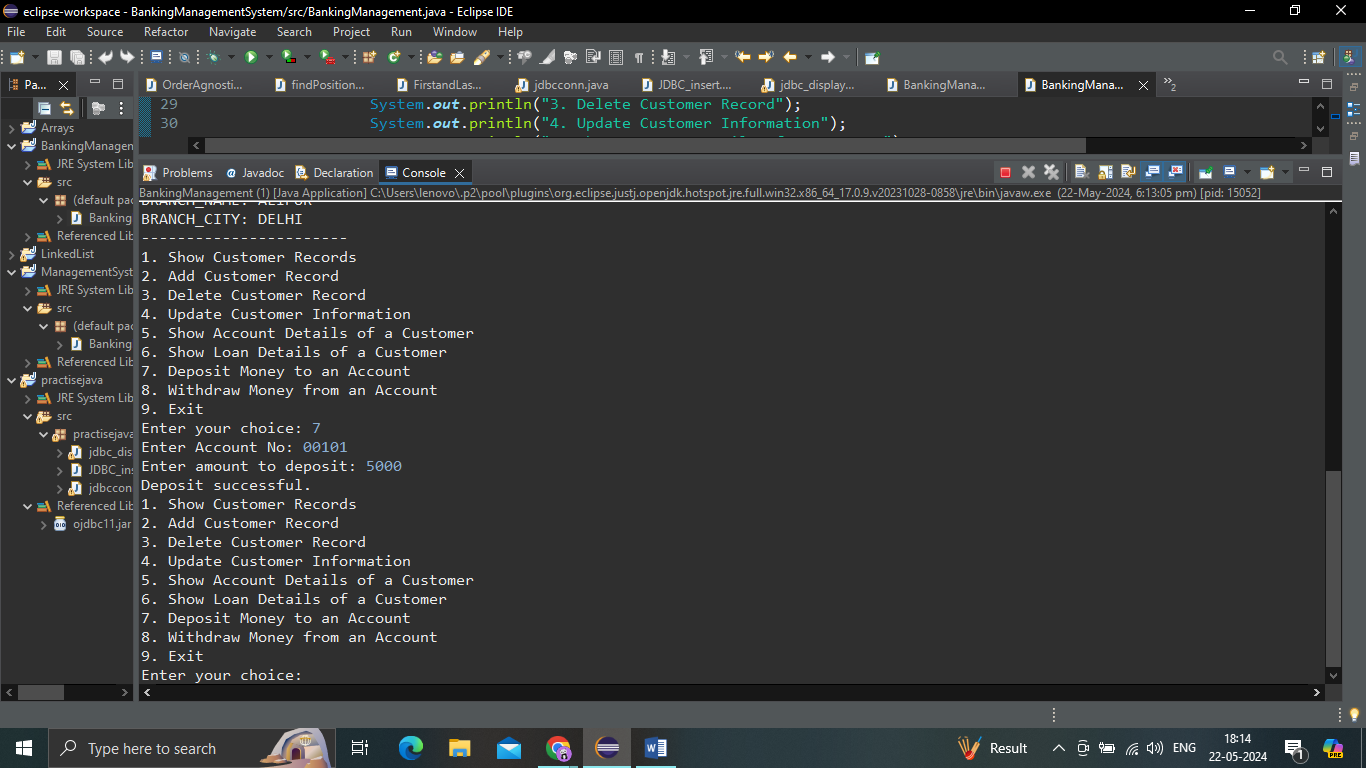
TEST CASE 5. Show Account Details of a Customer



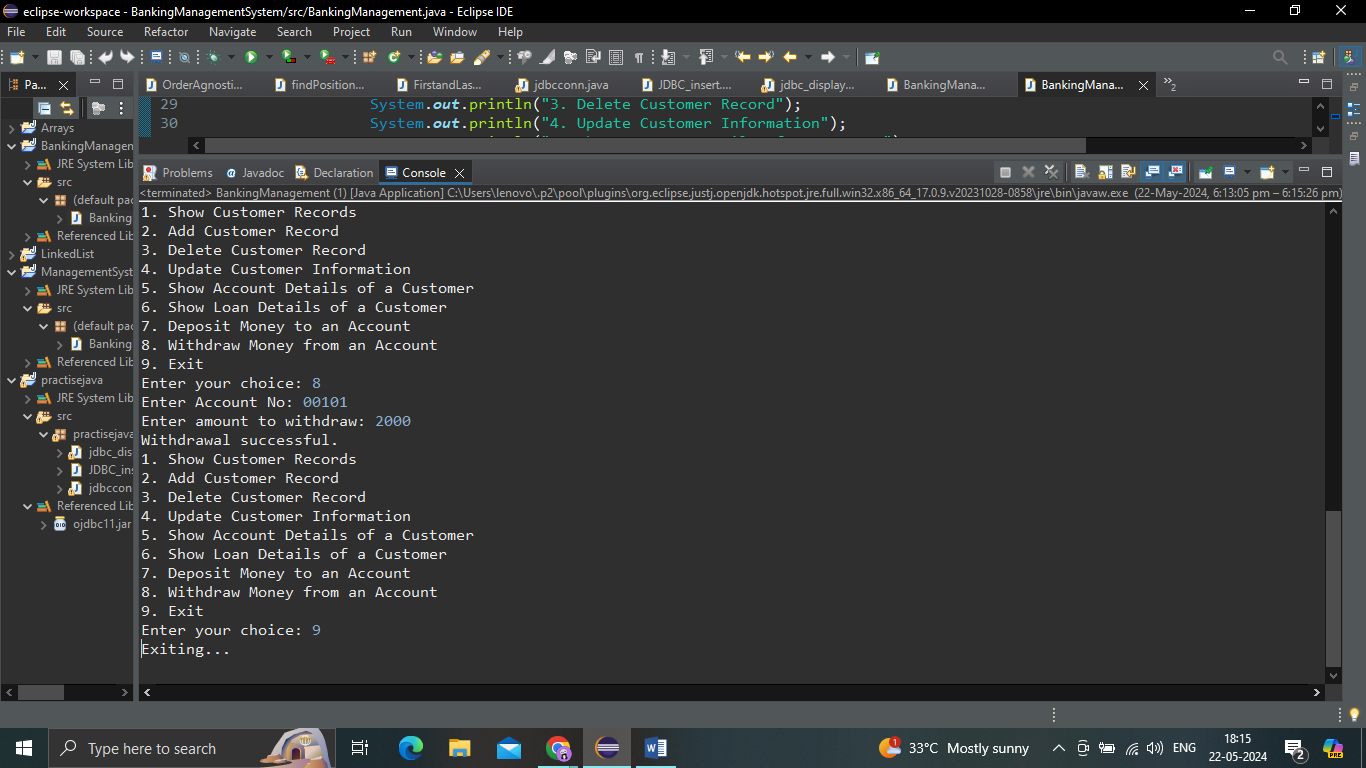
TEST CASE 6. Show Loan Details of a Customer



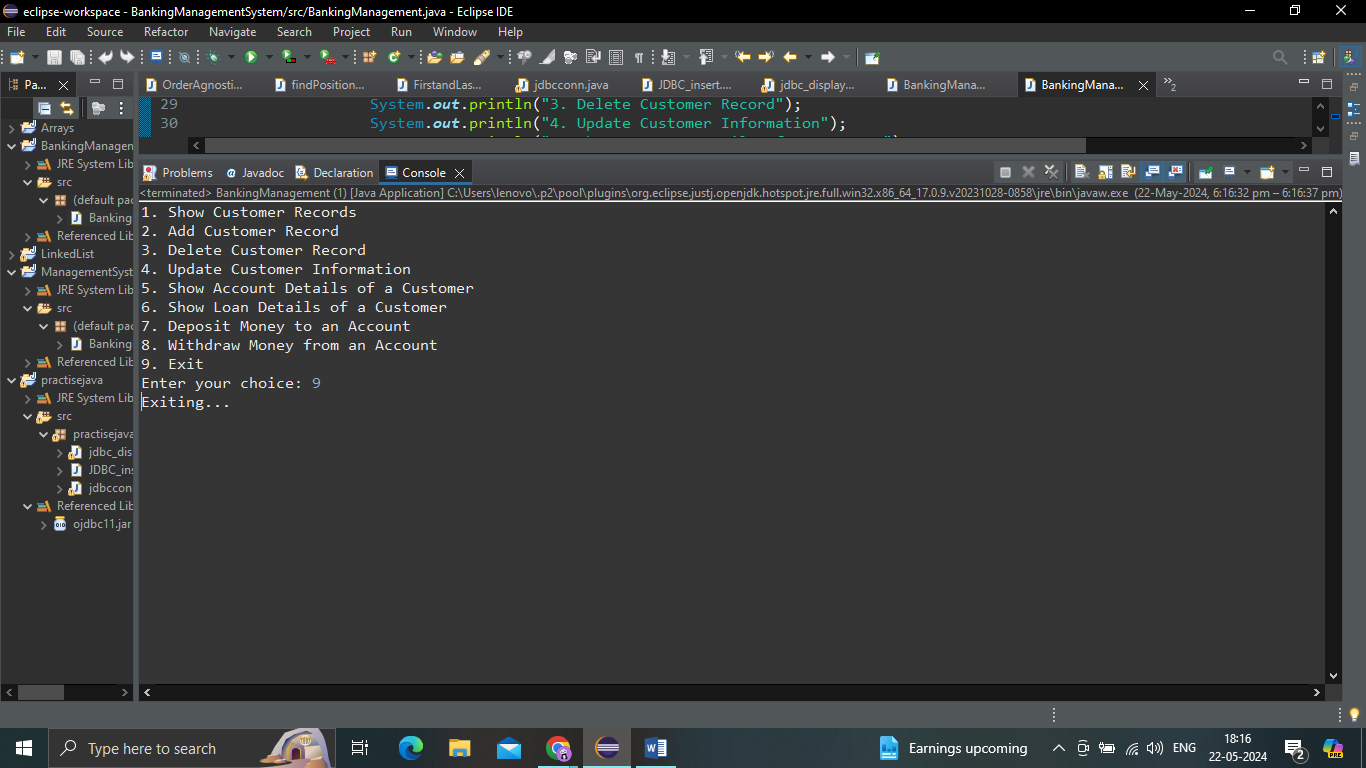
TEST CASE 7. Deposit Money to an Account



TEST CASE 8. Withdraw Money from an Account



TEST CASE 9. Exit



**CONCLUSION**

The Banking Management System developed using Java, Oracle, and JDBC demonstrates the effective integration of frontend and backend technologies to manage banking operations efficiently. This project has successfully met its objectives by providing a user-friendly console-based interface that performs essential banking functions, such as managing customer records and handling financial transactions.

Throughout the project, the integration of Java for the application layer, Oracle for the database layer, and JDBC for connectivity has proven to be a robust solution for banking management. The system ensures data integrity, consistency, and reliability, which are crucial for banking applications. The menu-driven approach simplifies user interactions, making it easy to navigate through different operations.

The implementation includes comprehensive features for viewing, adding, deleting, and updating customer information, as well as handling deposits and withdrawals. The proper handling of exceptions and the display of formatted output ensure clarity and ease of use.

Testing of the system has confirmed its functionality, with all operations performing as expected. The project highlights the practical application of theoretical concepts in a real-world scenario, showcasing the potential for further enhancements, such as the development of a graphical user interface for an even better user experience.

In conclusion, the Banking Management System is a testament to the power of combining programming skills with database management to create a functional, efficient, and user-friendly application. It serves as a foundational project that can be expanded and refined to meet the evolving needs of the banking industry.