Database Systems Lab – Mini Project

Title: Banking System Database

Team Members:

1) Rajeev Veeraraghavan (170905252)
2) Arjun Goel (170905284)

Abstract:

In a large scale banking system, there is a need to store a vast variety of data. A bank needs to keep track of its customers, their accounts and the transactions made by them between these accounts. A customer can have more than one account and can transfer money between them which adds to the complexity of the problem. A bank has multiple branches, with the customer having the option of creating an account in each of these branches. On top of this, there are also different types of accounts an individual can have, such as current and savings accounts. A person may also take a loan from the bank, on which he is charged an interest, while he earns interest on the money he has deposited, with the interest rate varying depending on the principal amount and also on the financial status of the individual. It then becomes necessary to store the date of the transaction as well since the debt increases with time.

Thus, there are many transactions taking place every day in a bank, and the database needs to be updated each time. In such a scenario, we can develop a relational database that can represent the real life scenario and store the results of all transactions in its tables. The database will have relations representing customer's details, details of the various types of accounts, details of when transactions were made, and details of the various branches of the bank. By developing and employing a relational database, we can efficiently keep track of the daily activities taking place in a bank.

In conclusion, an efficient relational database is required for the smooth running of a bank. The aim of our project is to develop such a database.

Problem Statement:

The aim of our project is to come up with a banking database, capable of addressing most of the issues discussed previously. Since most bank databases are very large, with a huge number of tables, we chose to create a small, but robust database, capable of satisfying the needs of a banking system, while remaining small and easy to comprehend. Our database coupled with

our front end Java application serve to simulate the working of a small banking system featuring multiple banks.

To address the functional needs of a bank database, we created the following tables:

Bank (code, name, head_address)

Branch (br_no, code, address)

Account (acc_no, balance, acc_type, br_no, code)

Customer (aadhaar, cname, phone_no, c_address)

Loan (loan_no, loan_type, amount, br_no, code, aadhaar)

Customer_accounts (aadhaar, acc_no)

The following sample data was taken:

Banks:

101 ,'Syndicate','Manipal'

102 ,'ICICI','Delhi'

103 ,'HDFC','Mumbai'

104 ,'SBI','Chennai'

Branches:

1,101,'Pune'

2,101,'Sikkim'

3,101,'Jaipur'

4,101,'Mangalore'

1,102,'Nagpur'

2,102,'Lucknow'

3,102,'Mumbai'

4,102,'Gurgaon

Customers:

1000,'Prithvi',12345,'Pune'

2000,'Vaishnav',23456,'Delhi’

3000,'Eshan',34567,'Kolkata'

4000,'Anubhav',45678,'Chennai'

Accounts:

10001 ,400000,'savings',1,101

10002 ,500000,'current',1,101

10003 ,200000 ,'savings',2,101

10004 ,100000 ,'other',3,101

10005, 300000,'joint',1,102

Loans:

111,'housing', 10000000,1,101,1000

121,'educational',5000000, 2,101,2000)

131,'business' ,2500000, 3,101,3000

141,'vehicle',7500000, 4,101,4000

Customer_accounts:

1000 ,10001

2000 ,10002

3000,10003

4000,10004

1000,10005

2000,10005

DDL Commands Used to Create Tables

```
CREATE TABLE bank(
        code NUMBER(8) PRIMARY KEY,
        name VARCHAR (30) NOT NULL,
        head_address VARCHAR(40));

CREATE TABLE branch(
      br_no NUMBER(8) not null,
      code references bank(code) ON DELETE CASCADE,
      address VARCHAR(40),
      PRIMARY KEY(code,br_no));

CREATE TABLE account(
        acc_no NUMBER(8) PRIMARY KEY,
        balance NUMBER(10) NOT NULL,
        acc_type VARCHAR(8) CHECK (acc_type IN('savings','current','joint','other')),
        br_no NUMBER(8) NOT NULL,
        code number(8) NOT NULL,
        FOREIGN KEY (br_no, code) REFERENCES branch(br_no,code) ON DELETE CASCADE);

CREATE TABLE customer(
        aadhaar NUMBER(8) PRIMARY KEY,
        c_name VARCHAR(25) NOT NULL,
        phone_no NUMBER(13),
        c_address VARCHAR(30));
```

```
CREATE TABLE loan(
        loan_no NUMBER(8) PRIMARY KEY,
        loan_type VARCHAR(20) CHECK (loan_type
        IN('educational','housing','marriage','vehicle','business')),
        amount NUMBER (10) NOT NULL,
        br_no NUMBER(8) NOT NULL,
        code NUMBER(8) NOT NULL,
        aadhaar REFERENCES customer(aadhaar),
        FOREIGN KEY (br_no, code) REFERENCES branch(br_no,code) ON DELETE CASCADE);

CREATE TABLE customer_accounts(
        aadhaar REFERENCES customer(aadhaar) ON DELETE CASCADE,
        acc_no REFERENCES account(acc_no) ON DELETE CASCADE);
```

The following two tables were added later on to give add more features to the database:

```
CREATE TABLE log_account
(
    acc_no NUMBER(8),
    balance NUMBER(10),
    acc_type VARCHAR(8),
    action_done VARCHAR(10) CHECK(action_done IN ('inserted','deleted') )
);
CREATE TABLE transfer_log(
    acc_no_1 NUMBER(8),
    acc_no_2 NUMBER(8),
    amt NUMBER(7),
    date_of_transfer DATE
```

);

Values for these two tables weren’t inserted directly, but instead triggers and procedures were used to insert values into these two tables. They allowed the database more functionality, as they kept a record of past events.

List of Queries

Simple queries

- Query to select the names of all customers along with their account numbers, type and balance

```
SELECT C.name, A.acc_no, A.type, A.balance

FROM customer C INNER JOIN customer_accounts CA ON C.aadhaar  = A.
aadhaar INNER JOIN account A ON CA.acc_no = A.acc_no

ORDER BY A.acc_no;
```

- Select the name and loan number of all customers who have taken a loan

```
SELECT C.c_name,L.loan_no

FROM customer C INNER JOIN loan L ON C.aadhaar = L. aadhaar;
```

Complex Queries

- Query to find the name of the bank which has the maximum number of customers

```
WITH bank_cust(bcode,name,num_cust)  AS

    (SELECT B.code,B.name,COUNT(C.aadhaar)

     FROM Bank B INNER JOIN Account A ON A.code = B.code INNER JOIN
     customer_accounts CA ON CA.acc_no = A.acc_no INNER JOIN customer C ON
     CA. aadhaar = C.aadhaar

     GROUP BY B.code,B.name),

bank_cust_max(num_cust) AS

    (SELECT MAX(num_cust) FROM bank_cust)

SELECT B1.bcode,B1.name

FROM bank_cust B1, bank_cust_max B2

WHERE B1.num_cust = B2.num_cust;
```

- Query to display the details of those banks which have a branch in a city where any bank is head quartered

```
WITH city_add AS (SELECT head_address FROM bank)
SELECT B.code, B.name
FROM bank B INNER JOIN branch Br ON B.code = Br.code
WHERE Br.address IN (SELECT * FROM city_add);
```

Queries used by front end application:

For displaying entire details:

```
SELECT * FROM bank;
SELECT * FROM customer;
```

For selecting account number while logging in:

```
SELECT acc_no FROM account;
```

For deleting certain values:

```
DELETE FROM account WHERE acc_no = x;
```

For depositing some money in account

```
UPDATE account SET balance = balance + amt WHERE acc_no = x;
```

For more complex processes such as withdrawals and money transfers, separate procedures were created and exceptions were raised when consistency of database was violated.

Procedure for withdrawal

```
CREATE OR REPLACE PROCEDURE withdrawal (amnt number,accno account.acc_no%TYPE, sf IN
OUT NUMBER) AS
        CantWithDraw EXCEPTION;
        current_amnt account.balance%TYPE;
BEGIN
 SELECT balance into current_amnt FROM account WHERE acc_no = accno;

 IF amnt > current_amnt THEN
     RAISE CantWithDraw;
 ELSE
     UPDATE account SET balance = balance - amnt WHERE acc_no = accno;
     COMMIT;
     sf := 0;
 END IF;

 EXCEPTION
     WHEN CantWithDraw THEN
          DBMS_OUTPUT.PUT_LINE('Can not Withdraw');
          sf := 1;
END;
/
```

Procedure for money transfer

```
CREATE OR REPLACE PROCEDURE transfer (acc_1 Account.acc_no%Type, acc_2
Account.acc_no%Type, amt NUMBER, sf IN OUT NUMBER ) AS
     bal account.balance%TYPE;
     insufficientBalanceException EXCEPTION;
```

```
BEGIN
    SELECT balance INTO bal FROM account WHERE acc_no = acc_1;
    bal := bal - amt;
    IF bal < 0 THEN
        RAISE insufficientBalanceException;
    ELSE
        COMMIT;
        UPDATE account SET balance = bal WHERE acc_no = acc_1;
        UPDATE account SET balance = balance + amt WHERE acc_no = acc_2;
        COMMIT;
        INSERT INTO transfer_log VALUES (acc_1, acc_2, amt, CURRENT_DATE);
        sf := 0;
    END IF;
    EXCEPTION
        WHEN insufficientBalanceException THEN
            DBMS_OUTPUT.PUT_LINE('Not enough balance');
            sf := 1;
END;
/
```

In both the above procedures, the IN OUT parameter sf is set to 1 whenever the desired transaction couldn't complete successfully.

A trigger for storing every insertion and deletion that takes place on the account table

```
CREATE OR REPLACE TRIGGER change_acc
AFTER INSERT OR DELETE ON account
```

```
FOR EACH ROW
BEGIN
 case
     WHEN INSERTING THEN
          INSERT INTO log_account
VALUES(:NEW.acc_no,:NEW.balance,:NEW.acc_type,'inserted');
     WHEN DELETING THEN
          INSERT INTO log_account VALUES(:OLD.acc_no,:OLD.balance,:OLD.acc_type,'deleted');
 end case;
end;
/
```

JAVA Code for Functional Design (DB Connectivity and Access)

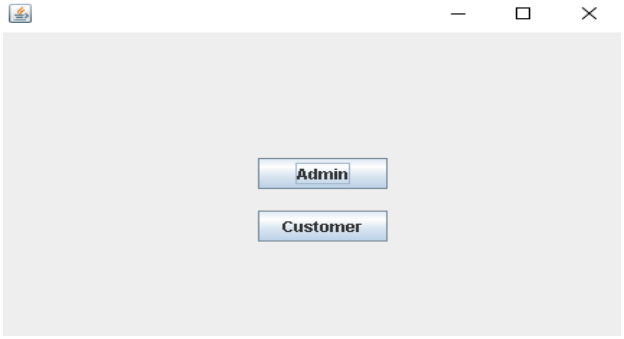
Code for connecting to the database

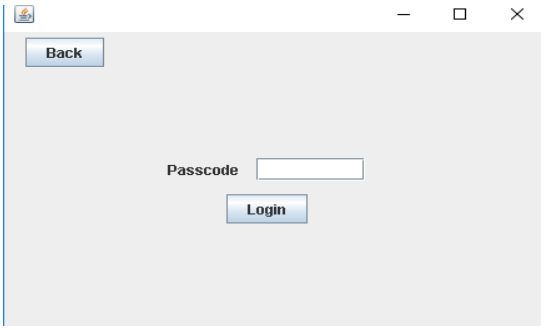
```
try {


	Class.forName("oracle.jdbc.driver.OracleDriver");

	Connection con =
	DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL"
	, "system", "rajeev");

	Statement stmt = con.createStatement();

	stmt.executeQuery("……”));
	con.close();

}
catch (SQLException ex) {

	System.out.println(ex);

}
catch (ClassNotFoundException ex) {

	System.out.println(ex);
}
```

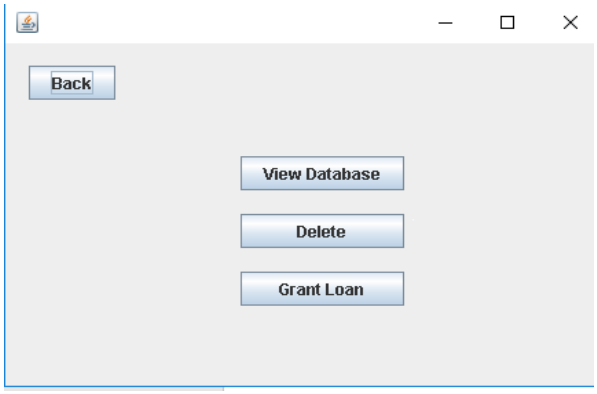
Code for shifting JFrames

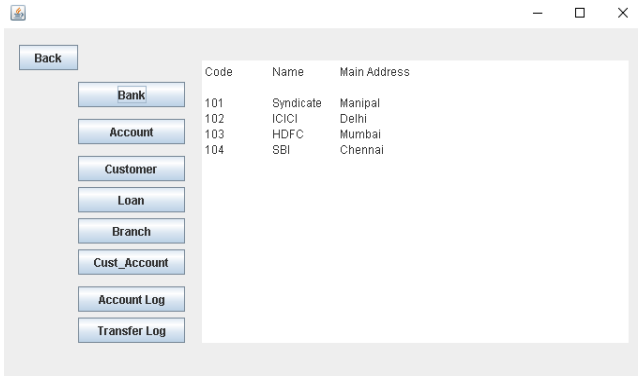
```
JButton btnAdmin = new JButton("Admin");
		btnAdmin.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				dispose();
				AdminLogin adLog = new AdminLogin();
				adLog.setVisible(true);
			}
		});
```

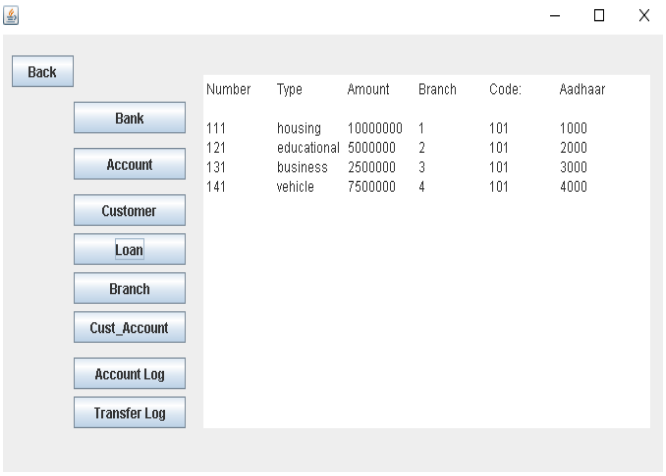
UI – Screen shots

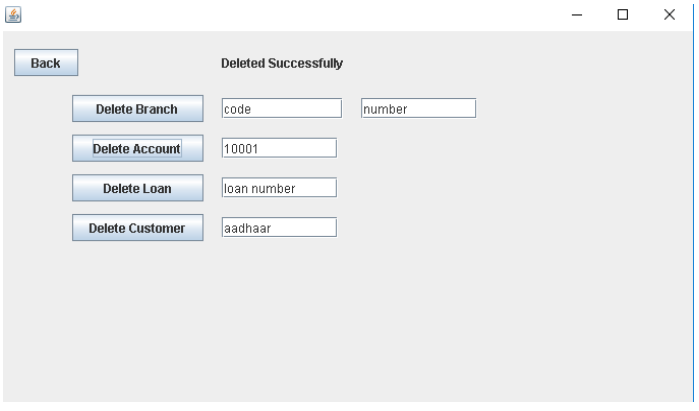


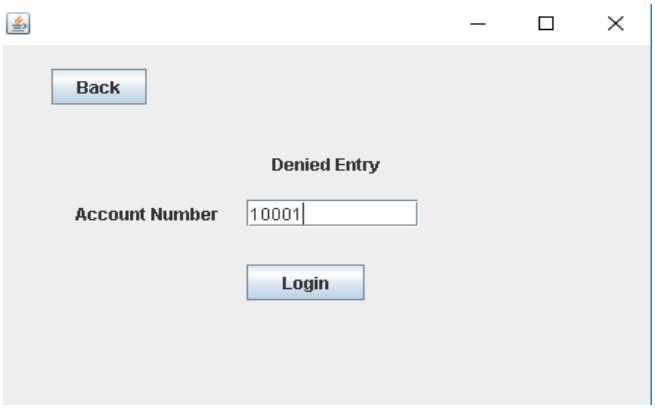


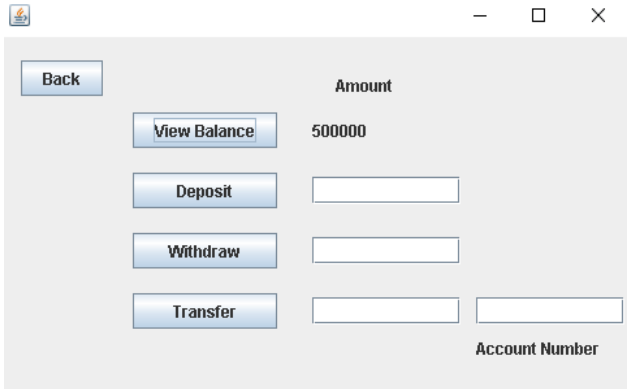


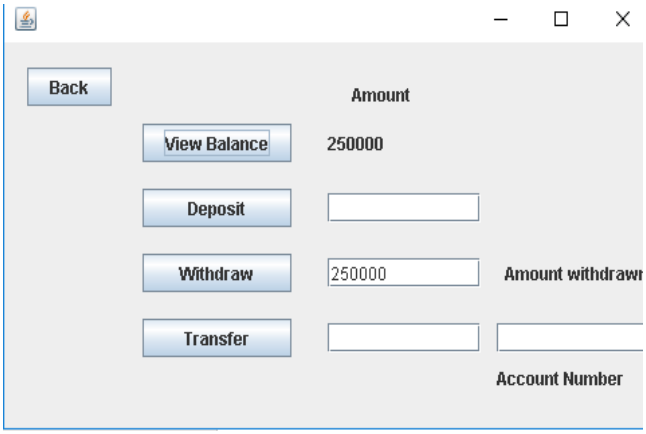


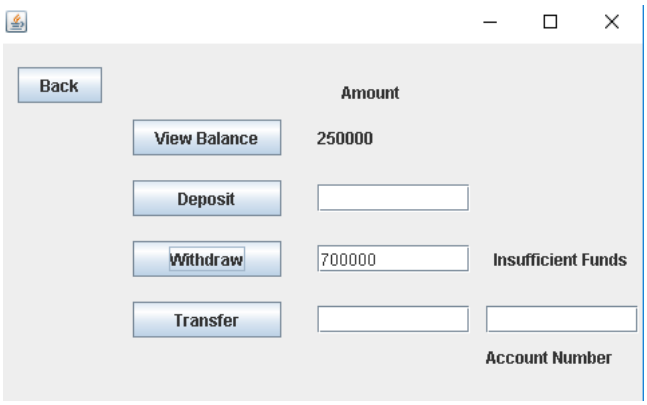


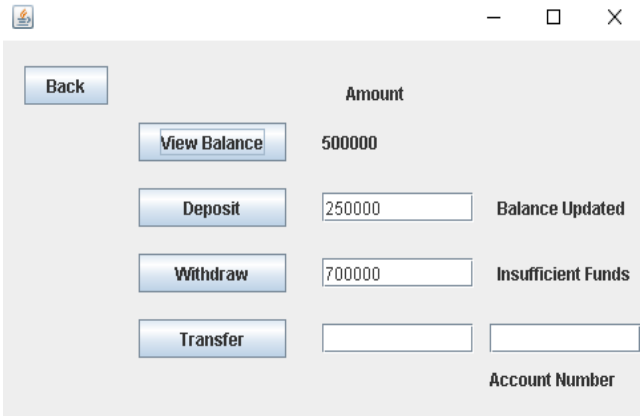


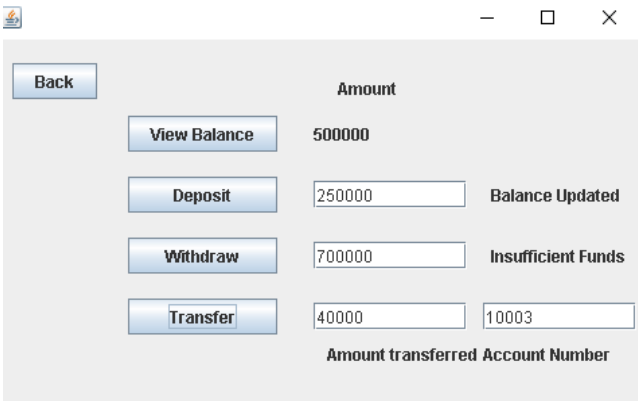

Back
Denied Entry
Account Number
10001
Login


Back
Amount
View Balance
500000
Deposit
Withdraw
Transfer
Account Number


Back
Amount
View Balance
250000
Deposit
Withdraw
250000
Amount withdrawn
Transfer
Account Number


Back
Amount
View Balance
250000
Deposit
Withdraw
700000
Insufficient Funds
Transfer
Account Number


Back
Amount
View Balance
500000
Deposit
250000
Balance Updated
Withdraw
700000
Insufficient Funds
Transfer
Account Number


Back
Amount
View Balance
500000
Deposit
250000
Balance Updated
Withdraw
700000
Insufficient Funds
Transfer
40000
10003
Amount transferred
Account Number