Banking Management System

Neelambari Verma 12/13/17 Data Management and Data Design

Table of Contents

BANKING MANAGEMENT SYSTEM

1. Title of the Project

Banking Management System

2. Objective of the project

Bank Management System manages the database and retrieves the details of the customers with accuracy. The banking system will maintain working for different accounts information, withdrawal, and deposit amount and also to keep a record for daily banking transactions.

This project will hold the list of customers. It will have different type of accounts like checking, savings and joint account. The database should provide the facilities of adding the new Customer account, deletion of account and modification of existing customer account. Along with this, block transactions for any account, show all required transactions.

3. Structure of the Project

3.1 Proposed System

The objective and goals of the proposed system are:

- To open any account required by the user.
- Authorization should be given only to the user i.e. Bank Employee to access various functions available in the system.
- It will maintain the list of Accounts, customer record and Transaction record.
- To maintain the status of the Account and maintain balance status easily.
- The user i.e. Employee can check Account with an Employee ID and password and which can work out with Account holders of the bank.
- Stored Procedure functions to update the Account type or to View the transaction list of a customer.
- Trigger function will be used to update the balance.
- View to display the Customer, Account Details, number of Account in each Branch.

4. Module Description of the project:

4.1 Roles and Task

The Banking System Project consists of three functional elements from an end-user perspective:

- The user admin will have all the access to the database.
- The user developer will have select access on Transaction and Account and Update /delete access on Branch.
- The user tester will have all the access on Transaction and Insert access on Account.
- The customer will have select access on all the table. Manager module, Employee transaction module and Customer transaction module and each will have its own Privileges.

4.2 Privileges

1. User Admin

CREATE User 'admin'@'localhost';

Granting all access to Admin

GRANT ALL ON mydb_bank.* TO 'admin'@'localhost'

2. User Developer

CREATE User 'developer'@'localhost' IDENTIFIED BY 'password';

Granting Access

GRANT ALL ON mydb_bank.Customers TO 'developer'@'localhost';

GRANT SELECT ON mydb_bank.Transactions TO 'developer'@'localhost';

GRANT SELECT ON mydb_bank.Accounts TO 'developer'@'localhost';

GRANT SELECT ON mydb_bank.Branch TO 'developer'@'localhost';

GRANT UPDATE ON mydb_bank.Branch TO 'developer'@'localhost';

GRANT DELETE ON mydb_bank.Branch TO 'developer'@'localhost';

GRANT SELECT ON mydb_bank.Checkings TO 'developer'@'localhost';

GRANT SELECT ON mydb_bank.Savings TO 'developer'@'localhost';

GRANT SELECT ON mydb_bank.Loan TO 'developer'@'localhost';

3. User Tester

CREATE User 'tester'@'localhost' IDENTIFIED BY 'pass';

Granting Access

GRANT SELECT ON mydb_bank.Customers TO 'tester'@'localhost';

GRANT ALL ON mydb_bank.Transactions TO 'tester'@'localhost';

GRANT SELECT ON mydb_bank.Transactions TO 'tester'@'localhost';

GRANT INSERT ON mydb_bank.Accounts TO 'tester'@'localhost';

GRANT DELETE ON Banking_Final.Accounts TO 'tester'@'localhost';

GRANT SELECT ON mydb_bank.Branch TO 'tester'@'localhost';

4. User

CREATE User 'user'@'localhost' IDENTIFIED BY 'pass';

Granting Access

GRANT SELECT ON mydb_bank.Customers TO 'user'@'localhost';

GRANT SELECT ON mydb_bank.Transactions TO 'user'@'localhost';

GRANT SELECT ON mydb_bank.Accounts TO 'user'@'localhost';

GRANT SELECT ON mydb_bank.Branch TO 'user'@'localhost';

GRANT SELECT ON mydb_bank.Checkings TO 'user'@'localhost';

GRANT SELECT ON mydb_bank.Savings TO 'user'@'localhost';

GRANT SELECT ON mydb_bank.Loan TO 'user'@'localhost';

Revoking Access

REVOKE INSERT ON mydb_bank.Accounts FROM 'tester'@'localhost';

REVOKE DELETE ON mydb_bank.Accounts FROM 'tester'@'localhost';

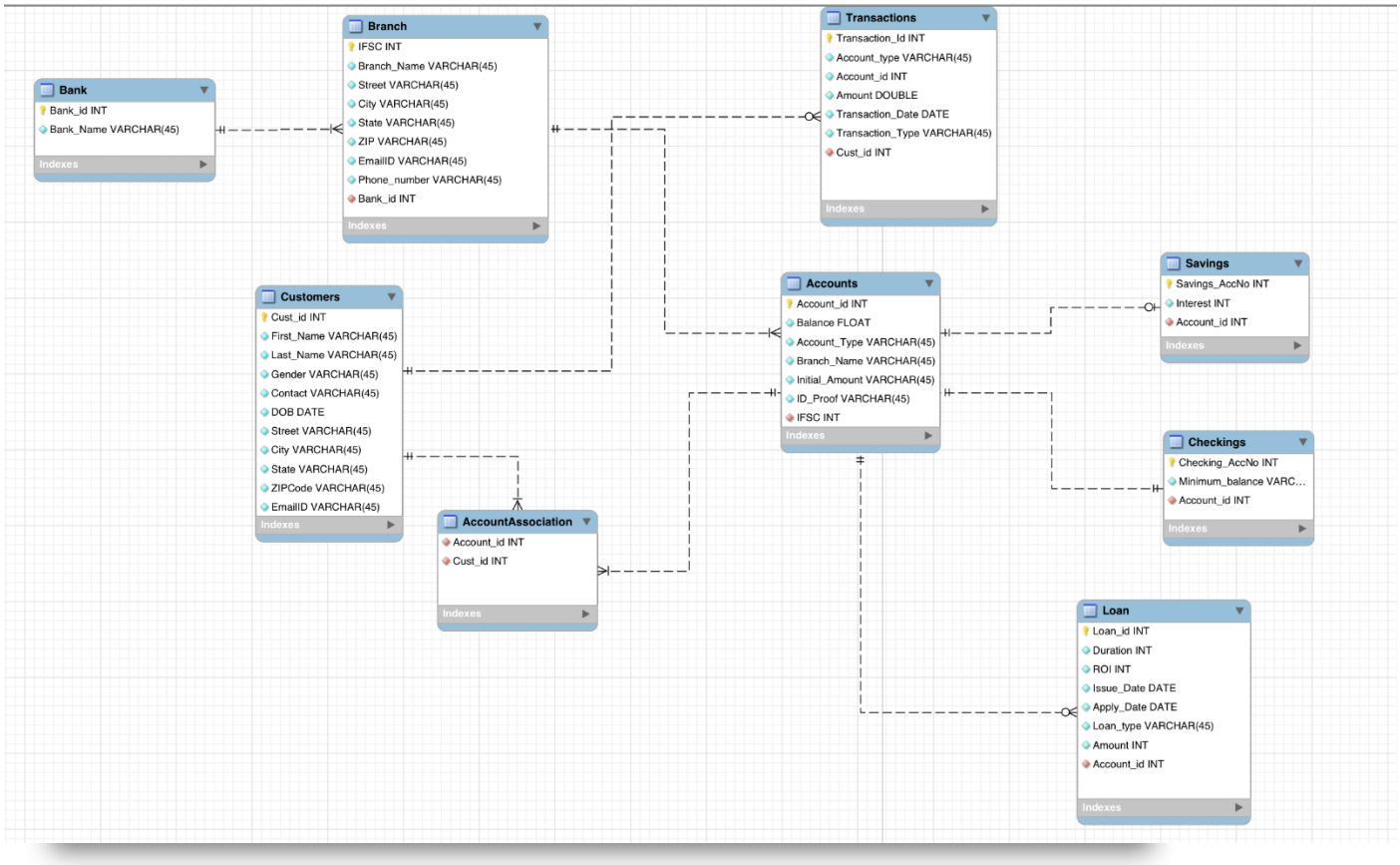
REVOKE UPDATE ON mydb_bank.Branch FROM 'developer'@'localhost';

REVOKE DELETE ON mydb_bank.Branch FROM 'developer'@'localhost';

5. Design of the project:

The design of the project will include customer table, transaction table, account table and bank branch table, Accounts table, Checkings table, Savings table, AccountAssociation table, Loans, . Here, the customer will maintain a set of relationships in the database. The customer has many-to-many relation with account since a customer can have joint account. Also, an account has one-to-many relation with the transaction and the bank branch table.

6. EER diagram



Identified the basic element and attributes required to build the Banking management system. There are 9 basic entities in the ER diagram.

Bank, Customers, Branch, Accounts, Transactions, Savings, Checkings, Loan, AccountAssociation.

With respect to database designing, determining the relationships with entities was an important factor, and at the modelling stage I implemented the database rules. I tried to improve the entity relationship as I went through the project implementation and also adjust the relationship accordingly. Business process understanding is much important during the database design stage. I tried my best to improve the process flow of the Bank Management System.

7. Relationship and Cardinalities

(a) **Bank** and **Branch** One to Many

A bank must have at least one Branch

A Branch must belong to exactly one Bank

(b) **Branch** and **Accounts** One to Many

A Branch will have one or many Accounts

An account will be only in one Branch.

(c) **Customers** and **Transactions** One to Many

A Customer can have zero or many transactions

A Transaction will be only done by a Customer

(d) **Accounts** and **Savings** One to One

An account will have exactly one Savings account

Each Saving Account must belong to exactly one account

(e) **Accounts** and **Checking** One to One

An account will have exactly one Checking account

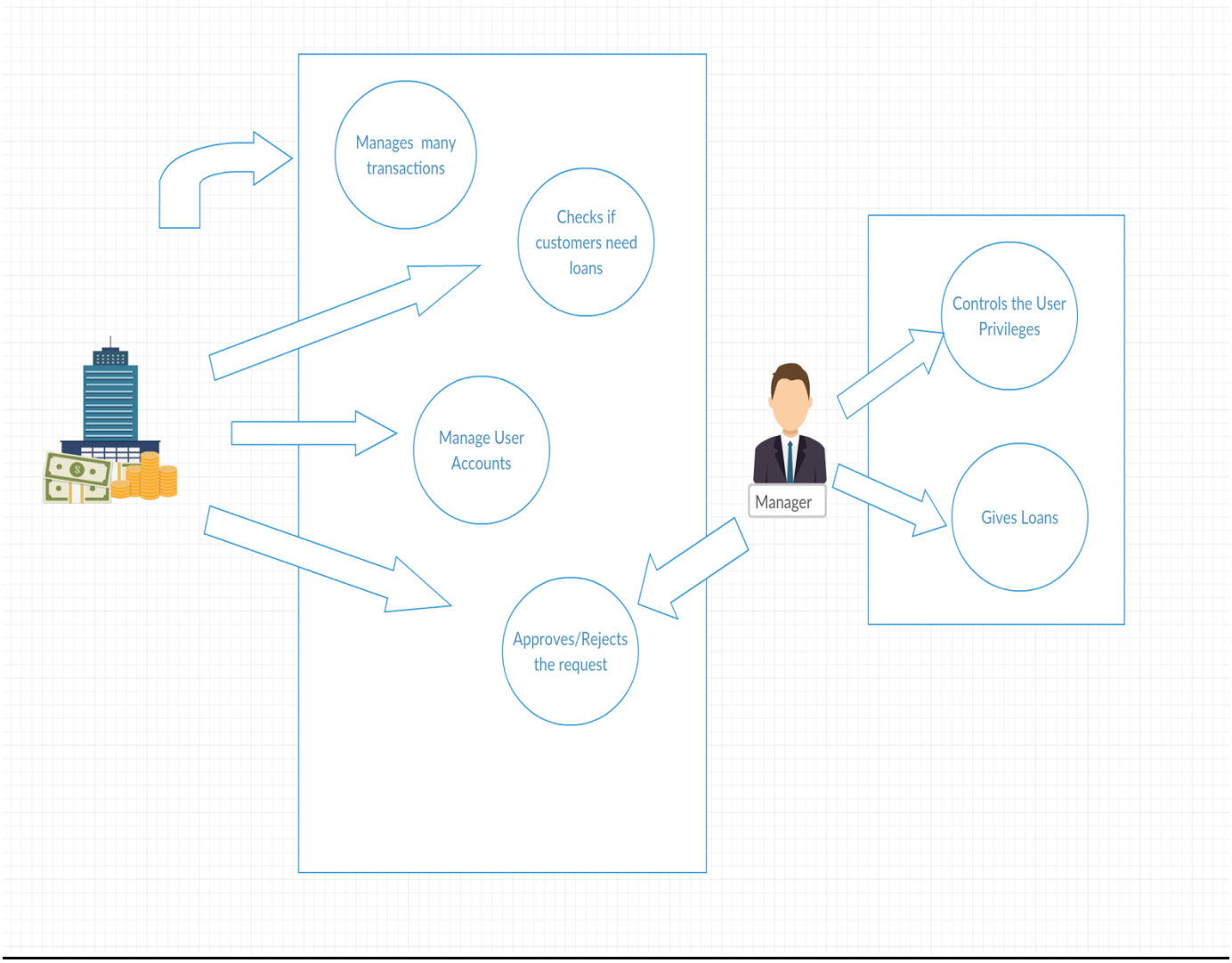
Each Checking Account must belong to exactly one account

(f) **Accounts** and **Loan** One to Many

An account can have zero loan or many loans

A Loan must belong to exactly one account

8. USE CASE DIAGRAM



9. Security

Security is the main issues of any system. This system is secured using all possible secure code methods to avoid any breach of data. This system deals with Views, which are used to control the data that is accessed by the user.

10. Views

1. To display the number of Account in each Branch

```
CREATE VIEW No_Of_AccountBranch AS
SELECT a.Account_id, COUNT(a.Account_id) AS 'Number of Account', b.IFSC, b.Branch_Name,
CONCAT_WS(' ',b.Street, b.City, b.State, b.ZIP) AS 'Branch Address',
b.EmailID, b.Phone_number
FROM Branch b INNER JOIN Accounts a ON b.IFSC = a.IFSC
GROUP BY a.IFSC
ORDER BY a.IFSC;

-- CALLING THE VIEW
SELECT * FROM No_Of_AccountBranch;
```

2. Display the Customer and Account Details

```
CREATE VIEW Customer_View AS
SELECT c.Cust_id, c.First_Name, c.Last_Name, CONCAT_WS(' ',c.Street, c.City, c.State, c.ZIPCode)
AS 'Customer Address',
a.Account_id, a.Balance, a.Account_Type
FROM Customers c
INNER JOIN AccountAssociation aa ON c.Cust_id = aa.Cust_id
```

INNER JOIN Accounts a ON aa.Account_id = a.Account_id

ORDER BY c.Cust_id;

-- CALLING THE VIEW

SELECT * FROM Customer_View;

DROP VIEW Customer_View;

```
14
15  -- View 2 : Display the Customer and Account Detials
16  CREATE VIEW Customer_View AS
17  SELECT c.Cust_id, c.First_Name, c.Last_Name, CONCAT_WS(' ',c.Street, c.City, c.State, c.ZIPCode) AS 'Customer Address',
18  a.Account_id, a.Balance, a.Account_Type
19  FROM Customers c
20  INNER JOIN AccountAssociation aa ON c.Cust_id = aa.Cust_id
21  INNER JOIN Accounts a ON aa.Account_id = a.Account_id
22  ORDER BY c.Cust_id;
23
24  -- CALLING THE VIEW
25
26  SELECT * FROM Customer_View;
27
```

100% 1:27

Result Grid | Filter Rows: Search | Export:

Cust_id	First_Name	Last_Name	Customer Address	Account_id	Balance	Account_Type
1001	Casey	Kirby	2273 Luctus Road Worcester MA 80900	10001	1883	Savings
1002	Graiden	Ray	Ap #456-7010 Quisque Road Owensboro KY 11015	10002	1500	Checkings
1003	Kylie	Quinlan	394-2314 Condimentum Rd. Portland OR 97829	10003	28800	Checkings
1004	Barrett	Raymond	Ap #386-9223 Pellentesque St. Lansing MI 90995	10004	4532	Savings

3. Display the Customer details, his account and transactions

CREATE VIEW Bank_View AS

SELECT c.Cust_id, c.First_Name, c.Last_Name, CONCAT_WS(' ',c.Street, c.City, c.State, c.ZIPCode) AS 'Customer Address',

a.Account_id, a.Balance, a.Account_type, b.IFSC, b.Branch_Name, CONCAT_WS(' ',b.Street, b.City, b.State, b.ZIP) AS 'Branch Address',

t.Transaction_id, t.Amount, t.Transaction_Date, t.Transaction_Type

FROM Customers c

INNER JOIN AccountAssociation aa ON c.Cust_id = aa.Cust_id

INNER JOIN Accounts a ON aa.Account_id = a.Account_id

INNER JOIN Branch b ON b.IFSC = a.IFSC

INNER JOIN Transactions t ON t.Account_id = a.Account_id

ORDER BY c.Cust_id;

-- CALLING THE VIEW

SELECT * FROM Bank_View;

DROP View Bank_View;

```
31
32    -- View 3 : Display the Customer details, his account and transactions
33 •  CREATE VIEW Bank_View AS
34    SELECT c.Cust_id, c.First_Name, c.Last_Name, CONCAT_WS(' ',c.Street, c.City, c.State, c.ZIPCode) AS 'Customer Address',
35    a.Account_id, a.Balance, a.Account_type, b.IFSC, b.Branch_Name, CONCAT_WS(' ',b.Street, b.City, b.State, b.ZIP) AS 'Branch Address',
36    t.Transaction_id, t.Amount, t.Transaction_Date, t.Transaction_Type
37    FROM Customers c
38    INNER JOIN AccountAssociation aa ON c.Cust_id = aa.Cust_id
39    INNER JOIN Accounts a ON aa.Account_id = a.Account_id
40    INNER JOIN Branch b ON b.IFSC = a.IFSC
41    INNER JOIN Transactions t ON t.Account_id = a.Account_id
42    ORDER BY c.Cust_id;
43
44    -- CALLING THE VIEW
45
46 •  SELECT * FROM Bank_View;
47
```

100% 25:46

Result Grid | Filter Rows: Search | Export:

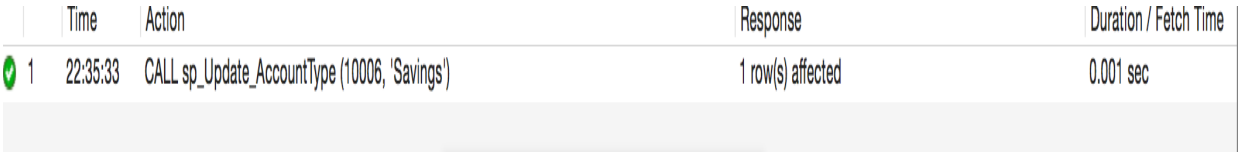
Cust_id	First_Name	Last_Name	Customer Address	Account_id	Balance	Account_type	IFSC	Branch_Name	Branch Address	Transaction_id	Amount	Transactic
1001	Casey	Kirby	2273 Luctus Road Worcester MA 80900	10001	1883	Savings	101	Head Office	8638 Non, Av. Springfield MA 97758	5008	2178	2014-04-19
1001	Casey	Kirby	2273 Luctus Road Worcester MA 80900	10001	1883	Savings	101	Head Office	8638 Non, Av. Springfield MA 97758	5001	1109	2010-08-12
1002	Graiden	Ray	Ap #456-7010 Quisque Road Owensboro KY 11015	10002	1500	Checkings	102	Atlanta	P.O. Box 882, 7111 Nulla St. Atlanta GA 71201	5010	1109	2010-08-12
1002	Graiden	Ray	Ap #456-7010 Quisque Road Owensboro KY 11015	10002	1500	Checkings	102	Atlanta	P.O. Box 882, 7111 Nulla St. Atlanta GA 71201	5002	2000	2010-08-12
1002	Graiden	Ray	Ap #456-7010 Quisque Road Owensboro KY 11015	10002	1500	Checkings	102	Atlanta	P.O. Box 882, 7111 Nulla St. Atlanta GA 71201	5007	845	2005-01-11
1003	Kylie	Quinlan	394-2314 Condimentum Rd. Portland OR 97829	10003	28800	Checkings	102	Atlanta	P.O. Box 882, 7111 Nulla St. Atlanta GA 71201	5003	2345	2010-08-12
1003	Kylie	Quinlan	394-2314 Condimentum Rd. Portland OR 97829	10003	28800	Checkings	102	Atlanta	P.O. Box 882, 7111 Nulla St. Atlanta GA 71201	5004	321	2005-01-31
1003	Kylie	Quinlan	394-2314 Condimentum Rd. Portland OR 97829	10003	28800	Checkings	102	Atlanta	P.O. Box 882, 7111 Nulla St. Atlanta GA 71201	5005	50	2013-08-03
1004	Barrett	Raymond	Ap #386-9223 Pellentesque St. Lansing MI 90995	10004	4532	Savings	103	Warren	P.O. Box 273, 4282 Ultrices. Road Warren MI 1...	5006	500	2009-03-27

11. Procedures

1. Stored Procedure to Update the Account Type

- This stored proc will get the Account id and Account type as input and update the Account for the account number in the account table.

```
Use mydb_bank;


DELIMITER //
CREATE PROCEDURE sp_Update2_Account_Type (
  IN ActNumber int, Acttype VARCHAR(45)
 )

BEGIN

 UPDATE Accounts
 SET Account_Type = Acttype
 WHERE Account_id = ActNumber;

END //
DELIMITER ;

-- CALLING THE PROCEDURE

CALL sp_Update2_Account_Type (10001, 'Savings');
```



	Time	Action	Response	Duration / Fetch Time
1	22:35:33	CALL sp_Update_AccountType (10006, 'Savings')	1 row(s) affected	0.001 sec

2. Stored Procedure to View the Transaction List of a Customer

- The mentioned stored proc will get the Customer id as input and display all the customer detail like customer name, account, transactions for the particular customer.

```
DELIMITER $$
CREATE PROCEDURE sp_View_TransactionList (
   Cust_id int
)
 BEGIN

  SELECT c.Cust_id, c.First_Name, c.Last_Name, a.Account_id, a.Balance, a.Account_Type,
                    t.Transaction_id, t.Transaction_Date, t.Amount, t.Transaction_Type
  FROM Customers c INNER JOIN AccountAssociation aa ON c.Cust_id = aa.Cust_id
  INNER JOIN Accounts a ON aa.Account_id = a.Account_id
  INNER JOIN Transactions t ON t.Account_id = a.Account_id
  WHERE c.Cust_id = Cust_id;

END $$
DELIMITER ;

-- CALLING THE PROCEDURE

CALL sp_View_TransactionList (1001);
```

DROP PROCEDURE sp_Update1_Account_Type;

```
26      -- Stored Procedure to View the Transaction List of a Customer
27
28      DELIMITER $$
29      CREATE PROCEDURE sp_View_TransactionList (
30          Cust_id int
31      )
32      BEGIN
33
34        SELECT c.Cust_id, c.First_Name, c.Last_Name, a.Account_id, a.Balance, a.Account_Type,
35              t.Transaction_id, t.Transaction_Date, t.Amount, t.Transaction_Type
36        FROM Customers c INNER JOIN AccountAssociation aa ON c.Cust_id = aa.Cust_id
37        INNER JOIN Accounts a ON aa.Account_id = a.Account_id
38        INNER JOIN Transactions t ON t.Account_id = a.Account_id
39        WHERE c.Cust_id = Cust_id;
40
41      END $$
42      DELIMITER ;
43
44
45      -- CALLING THE PROCEDURE
46
47      CALL sp_View_TransactionList (1001);
48
49
```

100% 1:48

Result Grid | Filter Rows: Search | Export:

Cust_id	First_Name	Last_Name	Account_id	Balance	Account_Type	Transaction_id	Transaction_Date	Amount	Transaction_Type
1001	Casey	Kirby	10001	1883	Savings	5001	2010-08-12	1109	Withdrawal
1001	Casey	Kirby	10001	1883	Savings	5008	2014-04-19	2178	Withdrawal

12. Triggers

- In the below example "AFTER" a new row is inserted into the transaction table it will check if the transaction type of the new inserted transaction using the keyword "NEW" for Deposit or Withdrawal and it will update the balance in the Account table based on it. The balance is increased if the transaction type is Deposit and vice-versa for Withdrawal.

Trigger Using Insert

```
Use mydb_bank;


DELIMITER $$
CREATE TRIGGER Update_Bal
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN

    IF(NEW.Transaction_Type = 'Deposit')
  THEN
  UPDATE Accounts
  SET Balance = (Balance + NEW.Amount)
  WHERE Account_id = NEW.Account_id;
  ELSEIF (NEW.Transaction_Type = 'Withdrawal')
  THEN
  UPDATE Accounts
  SET Balance = (Balance - NEW.Amount)
  WHERE Account_id = NEW.Account_id;
  END IF;
```

```
END$$


-- Change the MySQL delimiter back to ';'
DELIMITER ;
```

- Here, before trigger is used where before deleting a branch from the Bank Branch table, the account which is associated with the IFSC is updated with the head office.

Trigger Using Delete

```
-- If a branch is deleted the customer will be added to head office IFSC of the bank using the OLD keyword which gets the IFSC to be deleted.


DELIMITER //
CREATE TRIGGER Update_Branch1
BEFORE DELETE ON Branch
FOR EACH ROW
BEGIN

        UPDATE Accounts
    SET IFSC = 101
    WHERE IFSC = OLD.IFSC;



END//


-- Change the MySQL delimiter back to ';'
DELIMITER ;
```

- Here, if the transaction is updated the balance in the account table is updated based on the account id. It gets the old amount before the transaction is updated using the keyword OLD and update it with new amount using the keyword NEW.

Trigger Using Update

```
DROP TRIGGER Update_Branch1 ;


DELIMITER $$
CREATE TRIGGER Update_Balance_Account
AFTER UPDATE ON Transactions
FOR EACH ROW
BEGIN

        IF(NEW.Transaction_Type = 'Deposit')
    THEN
    UPDATE Accounts
    SET Balance = ((Balance - OLD.Amount) + NEW.Amount)
    WHERE Account_id = NEW.Account_id;
    ELSEIF (NEW.Transaction_Type = 'Withdrawal')
    THEN
    UPDATE Accounts
    SET Balance = ((Balance - OLD.Amount) - NEW.Amount)
    WHERE Account_id = NEW.Account_id;
    END IF;
END$$

-- Change the MySQL delimiter back to ';'
DELIMITER ;
```

13. Future scope of the project:

This project has range of scopes which will fulfill user requirement, to improve database performance, and also query processing time. There are few future enhancements that are possible and they are:

- OLAP applications
- Web Interface for new banking
- Linking with other banks and government agencies through Web services.
- Data security and system security.
- Electronic Data Integration (EDI) system for ATM machine.