**CS 631, Section 005 – Group Project (Bank System)**

**Team members: Chun Jie Chong, Kanishk Dnyaneshwar Shedsale**

**System Requirements:**

Requires a 64-bit processor and operating system

OS: Windows 10

Processor: Intel Core i3 or above

Memory: 4 GB RAM

Storage: 200 MB required

**Additions to the project specifications:**

1. More detailed address information is added for branches.
2. Role (attribute/column) is added for employee to different between managers and regular employees.
3. Fees is added if there is an overdraft.

**ER-Diagram:**

Diagram

Description automatically generated

**Major Design Decisions for ER-Diagram:**

1. We decided to change the relationship from between branch and customer to between branch and account. The relationship between branch and account makes more sense since Banks are supposed to manage the accounts, not the customers and customers care about their accounts, they don’t have to be associated with a bank unless they need a personal banker, which is already specified by the relationship (has personal banker) between customer and employee.
2. We make loan as one of the account types. This simplifies everything about loan since everything happens with loan will become a transaction. This helps us to minimize the amount of tables we need to create and also the query will be simpler or more general since everything is an account.
3. We also added an attribute for Transaction which is CurrBalance. This represents the balance of the account after this transaction. This is like a screenshot of the balance after each transaction. The balance in the account entity will only reflect the latest balance after all the transactions. If we want to find out what’s the balance of the account after a specific transaction, let’s say 2 months ago, we can then look at the CurrBalance attribute. If not, we have to do all the calculations manually back to that point of transaction.
4. Based on Professor Oria’s suggestion, we created two levels of transactions (Transaction, TransactionType) to reduce data redundancy.

**Relational Database Design:**

**Step 1: Handling Entities**

Branch (BranchID, Name, StreetNo, City, State, Zip);

Employee (SSN, FirstName, LastName, TelephoneNo, StartDate);

Customer (SSN, FirstName, LastName, StreetNo, AptNo, City, State, Zip);

TransactionType (Code, Name, Fees);

Account (AccountNo, balance, LastDatAccessed);

**Step 2: Weak Entities**

Dependent (DependentID, FirstName, LastName, EmpSSN);

Transaction (TransactionNo, Date, Hour, Amount, CurrBalance, AccountNo, Code);

**Step 3: 1:1 Relationships**

Branch (BranchID, Name, StreetNo, City, State, Zip, ManagerSSN, AssistantManagerSSN);

// since the overdraft relationship between accounts is partial participation (not all accounts

// must be overdrafted), created a table for this relationship.

Account\_Overdraft (TransferFromAccountNo, TransferToAccountNo, Amount, Fees);

**Step 4: 1:N Relationships**

// some of the 1:N relationships has been identified in step 2 (inserting foreign key)

// some relationships have weak participations on the n side that’s why we need to create

// a table for them

Employee (SSN, FirstName, LastName, TelephoneNo, StartDate, BranchID, Role);

Account (AccountNo, Balance, LastDateAccessed, BranchID);

HasPersonalBanker (EmpSSN, CustomerSSN);

**Step 5: M:N Relationships**

Customer\_Account (CustomerSSN, AccountNo);

**Step 6: Multivalued Attribute**

There is no multivalued attribute.

**Step 7: Higher Order Relationships**

There is no higher order relationships.

**Step 8: Specialization**

Entity Account has disjoint subclasses.

// We will go with the option that puts everything into one table (Account)

// Since none of the attributes in the subclasses are keys, they can be null in some cases.

// By doing this, it will be easier to do a query since everything about the account is in one table

// and also make query time shorter since we don't have to do join.

Account (AccountNo, Balance, LastDateAccessed, BranchID, Rate, LoanAmount, LoanPayment);

**Final Set of Relations with Normalization in 3NF are:**

Branch (BranchID, Name, StreetNo, City, State, Zip, ManagerSSN, AssistantManagerSSN);

Employee (SSN, FirstName, LastName, TelephoneNo, StartDate, BranchID, Role);

Dependent (DependentID, FirstName, LastName, EmpSSN);

Customer (SSN, FirstName, LastName, StreetNo, AptNo, City, State, Zip);

Account (AccountNo, Balance, LastDateAccessed, BranchID, Rate, LoanAmount, LoanPayment);

Account\_Overdraft (TransferFromAccountNo, TransferToAccountNo, Amount, Fees);

Transaction (TransactionNo, Date, Hour, Amount, CurrBalance, AccountNo, Code);

TransactionType (Code, Name, Fees);

Customer\_Account (CustomerSSN, AccountNo);

HasPersonalBanker (EmpSSN, CustomerSSN);

**Major Design Decisions for Relational Database:**

1. The only major design decision for the relational database is that we decided to have everything about an account (all disjoint subclasses and the parent class) in one table. Although there will be some null values in some columns, that will be fine since they are only attributes. This will make query time shorter since we don’t have to join tables to get information about an account. Also, this helps simplifying creation, updating, and deletion of an account since everything about it is in one table. We insert account into one table instead of doing it to multiple tables. The situation applies to updating and deleting an account as well.

**Application Program Design:**

There will be a main page for all the required applications:

Graphical user interface, application, website

Description automatically generated

Below are the pages for all the 3 required applications

1. Customer Application

Graphical user interface

Description automatically generated

1. Passbook Application

Table

Description automatically generated

1. Transaction Application

Graphical user interface, application

Description automatically generated

Below are the forms to enter information for all the applications

1. Customer Form (create, modify, delete customer)

Graphical user interface

Description automatically generated

1. Search Form (for passbook, client deletion, client modification

Graphical user interface

Description automatically generated

Customer form will be filled up with customer information for modifying or deletion.

**Graphical user interface

Description automatically generated**

**Major Design Decision for Application Program Design:**

1. We created triggers for a couple of scenarios in our application such as deleting a customer, inserting a transaction, and overdrafting.
2. We realized there is a flaw in the transaction table. We won’t be able to get a screenshot of the balance of the account after a specific transaction. Therefore, we added a column in the transaction table called “CurrBalance”. This will also help us to present the balance forward in the passbook application.
3. Overdraft will only happen with transfer. An account will only be allowed to be overdrafted once. Unless the balance of the account becomes positive again, no more transactions will be allowed for this account. There is an overdraft limit of $100. No accounts can overdraft more than $100, meaning that any account can only go as low as -$100 for the balance.
4. Monthly Service Charge can be generated through UI. An manual check is added to make sure Monthly Service Charge can only be generated once a month.

**List of Relational Instances that have been used in this application:**

Customer (SSN, FirstName, LastName, StreetNo, AptNo, City, State, Zip);

Account (AccountNo, Balance, LastDateAccessed, BranchID, Rate, LoanAmount, LoanPayment);

Account\_Overdraft (TransferFromAccountNo, TransferToAccountNo, Amount, Fees);

Transaction (TransactionNo, Date, Hour, Amount, CurrBalance, AccountNo, Code);

TransactionType (Code, Name, Fees);

Customer\_Account (CustomerSSN, AccountNo);

**Data used in the application:**

Customer Table:

Table

Description automatically generated

Account Table:



Customer Account Table:

Text

Description automatically generated with low confidence

Transaction Table:

Table

Description automatically generated

TransactionType Table:

Graphical user interface, application

Description automatically generated

Account\_Overdraft Table (when an account is overdrafted, this table will be populated):



**Queries used in DB for the application:**

create table branch (

branchID int auto\_increment,

branchName varchar(20),

streetNo varchar(20),

city varchar(20),

state varchar(20),

zip varchar(20),

managerssn varchar(20),

assistantmanagerssn varchar(20),

primary key (branchID)

);

create table employee (

ssn varchar(20),

firstName varchar(20),

lastName varchar(20),

telephoneNo varchar(20),

startDate date,

branchID int,

myRole varchar(20),

primary key (ssn)

);

create table dependent (

dependentID int auto\_increment,

firstName varchar(20),

lastName varchar(20),

empssn varchar(20),

primary key (dependentID)

);

create table customer (

ssn varchar(20),

firstName varchar(20),

lastName varchar(20),

streetNo varchar(20),

aptno varchar(20),

city varchar(20),

state varchar(20),

zip varchar(20),

primary key (ssn)

);

DROP TABLE MYACCOUNT;

create table myAccount (

accountNo int (7) zerofill auto\_increment,

balance double,

lastDateAccessed date,

branchID int,

rate varchar(20),

loanamount double,

loanpayment double,

primary key (accountno)

);

create table account\_overdraft (

transferFromAccountNo int,

transferToAccountNo int,

amount double,

fees double,

primary key (transferFromAccountNo, transferToAccountNo)

);

create table myTransaction (

transactionNo int (7) auto\_increment,

transdate date,

transhour time,

amount double,

currBalance double,

accountNo int,

transcode varchar(3),

primary key (transactionNo)

);

create table transactiontype (

TransCode varchar(3),

TransName varchar(30),

fees double,

primary key(transcode)

);

create table customer\_account (

customerssn varchar(20),

accountNo int,

primary key (customerssn, accountno)

);

create table haspersonalbanker (

empssn varchar(20),

customerssn varchar(20),

primary key (empssn, customerssn)

);

**Inserting data for some of the tables:**

-- Data for Branch

insert into branch (branchName, streetno, city, state, zip, managerssn, assistantmanagerssn) values ('Newark Branch', '123 Main St', 'Newark', 'New Jersey', '07029', '123-12-123', '789-12-123');

select \* from branch;

-- Data for Employee

insert into employee values ('123-12-123', 'John', 'Smith', '520-444-5555', '2020-1-1', 123, 'Manager');

insert into employee values ('456-12-456', 'Avery', 'Bradley', '670-444-9555', '2020-1-1', 123, 'Employee');

insert into employee values ('789-12-123', 'Tom', 'Brady', '432-444-5355', '2020-1-1', 123, 'Assistant Manager');

select \* from employee;

-- Data for Dependent

insert into dependent (firstName, lastName, empssn) values ('Joe', 'Brady', '789-12-123');

select \* from dependent;

-- Data for TransactionType

insert into transactionType values ('WD', 'Cash Withdrawal', '2.00');

insert into transactionType values ('DEP', 'Cash Deposit', '0');

insert into transactionType values ('MSC', 'Monthly Service Charge', '10.00');

insert into transactionType values ('CD', 'Check Deposit', '0');

insert into transactionType values ('CW', 'Check Withdrawal', '0');

insert into transactionType values ('OD', 'Overdraft Fees', '5.00');

**Triggers for the application:**

// when an account is overdrafted, we insert into this table, and after insert, it creates

// a transaction for the overdraft fees

CREATE DEFINER=`root`@`localhost` TRIGGER `account\_overdraft\_AFTER\_INSERT` AFTER INSERT ON `account\_overdraft` FOR EACH ROW BEGIN

declare tDate date;

declare tTime time;

declare bal double;

select current\_date() into tDate;

select current\_time() into tTime;

select balance into bal from myaccount where accountno = new.transferFromAccountNo;

insert into mytransaction (transdate, transhour, amount, currBalance, accountNo, transcode)

values (tDate, tTime, 5, bal - 5, new.transferFromAccountNo, 'OD');

END

// when we delete a customer, we delete the tuples associated with this customer

// in the relationship table as well

CREATE DEFINER=`root`@`localhost` TRIGGER `customer\_AFTER\_DELETE` AFTER DELETE ON `customer` FOR EACH ROW BEGIN

delete from customer\_account where customerssn = old.ssn;

END

// after we delete the tuple in the relationship table

// we check if the account still has any owners or not.

// if no owners associated with the account anymore

// we delete the account from account table as well

CREATE DEFINER=`root`@`localhost` TRIGGER `customer\_account\_AFTER\_DELETE` AFTER DELETE ON `customer\_account` FOR EACH ROW BEGIN

declare totalNumberOfAccount int;

select count(accountNo) into totalNumberOfAccount

from customer\_account

where accountNo = old.accountNo;

if (totalNumberOfAccount <= 0) then

delete from myaccount where accountNo = old.accountNo;

end if;

END

// if an overdrafted account has positive balance again, we delete it from the overdraft table

CREATE DEFINER=`root`@`localhost` TRIGGER `myaccount\_AFTER\_UPDATE` AFTER UPDATE ON `myaccount` FOR EACH ROW BEGIN

if (old.balance < 0 and new.balance >=0) then

delete from account\_overdraft where transferFromAccountNo = new.AccountNo;

end if;

END

// if we delete an account, the associated transactions will be deleted as well

// tuples related in the overdraft table will be deleted as well

CREATE DEFINER=`root`@`localhost` TRIGGER `myaccount\_AFTER\_DELETE` AFTER DELETE ON `myaccount` FOR EACH ROW BEGIN

delete from mytransaction where accountNo = old.accountNo;

delete from account\_overdraft where transferFromAccountNo = old.accountNo;

END

// if there is a transaction, after the transaction occurred, the balance of the account will be

// updated

CREATE DEFINER=`root`@`localhost` TRIGGER `mytransaction\_AFTER\_INSERT` AFTER INSERT ON `mytransaction` FOR EACH ROW BEGIN

declare currFees double;

select fees into currFees from transactiontype where transcode = new.transcode;

update myAccount set balance = new.currBalance, lastDateAccessed = new.transdate where accountNo = new.accountNo;

END

**The code will be provided in another file (Eclipse Project file).**