CS 631 Final Bank Project

CS 631-007 Group 1

Shashwat Singh – ss4673@njit.edu

# Report on Project Phases

## Business Requirements

The business requirements for this banking project were to design a database that can handle multiple branches of a single bank. We opted to allow for multiple different banks to be represented in our database to make our design more robust. The banks would have branches with their own names, IDs, assets, and a manager and assistant manager. The bank’s customers needed to have their SSN, name, address, account information, and personal banker information. We chose to handle a customer’s potential multiple accounts through a holds account relationship relation. This would have been impossible by simply giving the customer relation an account number field. The bank’s employees needed their SSN, name, telephone number, dependents, start date, and which bank they work for. The database needed to be able to handle four different types of accounts (savings, checking, money market, and loan) with different descriptor values for the respective accounts. Loan accounts in particular had to have their loan amount, loan number, and interest kept track of. The final requirement for this project was to maintain a ledger of all transactions made by customers that stored the transaction type, date, time, amount, and the account.

## Entity-Relationship Design

Our next step in this project was to convert our business requirements into appropriate entities, attributes and relations in an Entity Relationship Diagram. The simple entities created were Bank, Employees, Account, and Customer. The weak entities were Branches and Transaction with their respective identifying entities being Bank and Account. We chose to handle the four different account types (saving, checking, money market, and loan) by making them subclasses of Account. Similarly, we opted to make the manager, assistant manager, and personal banker roles subclasses of Employees. Following this we created the appropriate relationships between these entities with cardinality and participation constraints. Most of these relationships are quite standard, but the most interesting relationship is the ternary relationship between Manager, Branches, and Employees.

## Logical Database Design

The next step for our project was to convert our conceptual Entity-Relationship design into concrete SQL tables. We followed the appropriate algorithm for converting our diagram into tables and wrote the required code. The tables we created to represent entities are Bank, Employee, Manager, Asst Manager, Personal Banker, Branches, Account, Savings, Checking, Money Mkt, Customer, Transaction, and Loan Account. Following the steps, we made sure to treat weak entities differently and included the appropriate foreign keys to their identifying entities. The tables all had their primary keys and foreign keys specified alongside delete conditions on the foreign keys. Their attribute fields were assigned to appropriate data types. Lastly, we made four relationship relations to deal with the M:N and N-ary relationships in our diagram. These are the Works In, Borrower, Holds Account, and Manages tables. With our database definition complete, we tested the integrity of our database by inserting dummy data into all of our tables and stress testing update and delete capabilities.

## Application Program Design

1. The first major design decision we faced was about deciding the flow of the web application. That is how to connect multiple web pages with each other and how the flow of the user interface should run.

For this, we did mock-ups of various web pages and the options it should have for navigation to the next web page.Based on the mock-ups, we wrote html pages for all the views. To achieve this, we divided all the web pages into logical sections and made a proper file structure.

We worked on the navigation bar that needs to be present in each web page. That is we decided the tabs and links on each page.

Then we started connecting one web page to the other to have a streamlined flow for a user.

We decided to have a separate web page for each functionality because that would clearly segregate the business logic for every submit request in the html forms and also helps in easily identifying the errors.

2. The second design decision we faced was regarding integrating the frontend with the backend i.e., database and displaying accurate results for various queries. For this, we had to figure out the php code for communicating with the database and carrying out various sql queries and displaying the result in an appropriate format on the frontend. We used POST method for collecting the data from the html forms. We resolved this design decision by verifying that all the tables in the database are in the required normal form so as to avoid inconsistencies. Then we tested each webpage by entering dummy data and verifying the results of the queries.

3.The third major design decision was about figuring out how to carry forward the changes made by one user to a different user. For example, when the customer requests to add money to his/her account or transfer money between accounts, this would require the approval of the admin. This was done using session variables by storing information from one web page and carrying it to the next web page based on the design of navigation. Also, each request from the user had to be identified by a unique ID. For this reason, we had to add additional tables to our existing database logical design and include these unique IDs as primary keys. Also, we had to add some columns to maintain the updated status of the data. For example, upon approval from the admin of the bank, the status column for a particular row (add money) needs to be updated as ‘approved’. This will ensure that the updated balance is reflected on the customer's web page.

4. The final design decision was regarding making the user interface look attractive and easy to use. For this, we worked on the CSS part and figured out good templates for easy navigation between multiple web pages. We asked a third person with no technical knowledge to use our website so as to understand how user-friendly our application is. Based on our observations, we made the necessary changes to the UI.

# Normalize Relations

## Primary Keys are underlined and Foreign Keys are italicized.

Bank (BankName)

Graphical user interface, text, application

Description automatically generated

Key: BankName

BankName 🡪 BankName

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

Employee (Ssn, PhoneNo, FName, LName, StartDate, DependentNames, *BankName*)

A screenshot of a computer

Description automatically generated with medium confidence

Key: Ssn, PhoneNo

Ssn 🡪 PhoneNo, FName, LName, StartDate, DependentNames, BankName

PhoneNo 🡪 Ssn, FName, LName, StartDate, DependentNames, BankName

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

Manager (*Essn*)

Graphical user interface, application

Description automatically generatedKey: Essn

Essn 🡪 Essn

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

AsstManager (*Essn*)

Graphical user interface, application, table

Description automatically generated

Key: Essn

Essn 🡪 Essn

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

PersonalBanker (*Essn*)

Text

Description automatically generated

Key: Essn

Essn 🡪 Essn

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

Branches (BranchId, *BankName*, BranchCity, Address, Assets)

Graphical user interface, application

Description automatically generated

Key: BranchId

BranchId 🡪 BankName, BranchCity, Address, Assets

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

Account (AccNo, AccBalance, AccType, *BranchId*)

Application, table, Excel

Description automatically generated

Key: AccNo

AccNo 🡪 AccBalance, AccType, BranchId

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

Savings (*AccNo*, Interest)

Table

Description automatically generated

Key: AccNo

AccNo 🡪 Interest

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

Checking (*AccNo*, OverdraftFee)

Graphical user interface, text

Description automatically generated with medium confidence

Key: AccNo

AccNo 🡪 OverdraftFee

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

MoneyMkt (*AccNo*, VariableInterest)

Table

Description automatically generated

Key: AccNo

AccNo 🡪 VariableInterest

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

Customer (Ssn, FName, LName, PhoneNo, Street, City, Zipcode, *Pbssn*)

A picture containing table

Description automatically generated

Key: Ssn, PhoneNo

Ssn 🡪 FName, LName, PhoneNo, Street, City, Zipcode, Pbssn

PhoneNo 🡪 Ssn, FName, LName, Street, City, Zipcode, Pbssn

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

Transaction (TransactionId, DateTime, Type, Amount, *AccNo*, Charges)

Table

Description automatically generated with medium confidence

Key: TransactionId, AccNo

TransactionId, AccNo 🡪 DateTime, Type, Amount, Charges

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

LoanAccount (LoanNo, LoanAmount, Interest, *AccNo*)

Table

Description automatically generated

Key: LoanNo, AccNo

LoanNo, AccNo 🡪 LoanAmount, Interest

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

WorksIn (*BranchId,* *Essn*)

Table

Description automatically generated

Key: BranchId, Essn

BranchId, Essn 🡪 BranchId, Essn

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

Borrower (*Cssn*, *LoanNo*)

Table

Description automatically generated with medium confidence

Key: Cssn, LoanNo

Cssn, LoanNo 🡪 Cssn, LoanNo

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

HoldsAccount (*Cssn*, *AccNo*)

Text

Description automatically generated with medium confidence

Key: Cssn, AccNo

Cssn, AccNo 🡪 Cssn, AccNo

This is in 3NF as there are no partial dependencies or transitive dependencies on non-keys

Manages (*BranchId*, *Mssn*, *Essn*, NumEmployees)

Graphical user interface, text, application

Description automatically generated

Key: BranchId, Mssn, Essn

BranchId, Mssn, Essn 🡪 NumEmployees

# Four Queries in English and SQL

**GROUP BY**

Display the largest balance accounts per account type in all banks.

Select MAX(AccBalance),AccType

From Account

Group By AccType;

Table

Description automatically generated

**GROUP BY and HAVING**

Display the largest balance accounts at the CapitalOne Branch 6752.

Select MAX(AccBalance), AccType, BranchId

From Account

Group By AccType, BranchId

Having BranchId = '6752';

Graphical user interface, table

Description automatically generated

**Nested query with ALL**

Display all the loan numbers and loan amount which are less than account balance in all savings type account.

Select LoanNo, LoanAmount

From LoanAccount L, Account A

Where L.AccNo =A.AccNo

AND  LoanAmount < ALL (

Select AccBalance

From Account

Where AccType = 'SAVINGS');

Table

Description automatically generated

**Nested query with IN**

Show all the managers and their Bank Name that manage over 15 employees.

Select M.Mssn, B.BankName

From Employee E, Manages M, Branches B

Where E.Ssn = M.Mssn AND M.BranchId = B.BranchId

AND M.Mssn IN ( Select Mssn

From Manages

Where NumEmployees > 15);/

Table

Description automatically generated

# Conclusion

Our group had a positive experience working on this project. For the first phase, we both individually worked on our ER diagrams and attempted to combine them afterwards. This was the hardest step in the project. Making sense of the business requirements into a cohesive ER diagram was very difficult. This also taught us that it would be prudent to meet up and grind out our work together, which we implemented starting from the second phase on. The easiest step felt like writing the SQL code for the database definition, insertions, updates, and deletes. There were hard parts in the database definition specifically whether to use cascade or set null for our foreign keys. We both learned when and why we had to use these which we didn’t expect to learn in this phase. Another thing we learned from converting our ER diagrams to DDL was the reasoning behind the algorithm we were taught in class. After creating the relationship relations and seeing how they work, the need for them became clear. Designing our own queries that utilized certain keywords was difficult since we chose to work backwards from designing a query and then translating it into English. If we did this again, we would have done this differently and formulated queries in English first and then translate them into SQL. Overall, this project definitely showed us all the steps involved in producing a working database from just a requirement sheet.