Group 6: Bank Project

Anbar Saleem (aas27@njit.edu), Saumya Dwivedi (sd96@njit.edu)

CS331-H01

Business Requirements

Branches

Each bank is organized into branches which are located in a particular city with an address and is identified by a unique branch-ID and name. The bank monitors the assets of each branch and each branch of the bank has a manager and an assistant manager who are employees of the bank.

Customers

Bank customers are identified by their social security numbers. The bank stores each customer's name and address (apartment number, street number, state, city, and zip code). Customers may have several accounts and can take out loans that are managed as accounts as well. They also may be associated with a particular branch and may have a personal banker who works with the customer on their loan and bank transactions.

Employees

Bank employees are also identified by their social security numbers. The bank stores the name and telephone numbers of each employee, and the names of the employee's dependents. The bank also keeps track of the employee's start date and, thus, length of employment. An employee at the bank works for one of the branches and reports to the manager of that branch. may have a manager and the manager is in charge of a certain number of employees.

Accounts

The bank offers several types of accounts, savings, checking, money market, and loan accounts. An account can be held by more than one customer, and a customer can have more than one account. Each account is assigned a unique account number. The bank maintains a record of each account balance and the most recent date on which the account was accessed by each customer holding the account. In addition, saving and loan accounts have fixed interest rates, money market accounts have variable interest rates regularly updated based on the stock market. and overdrafts are recorded for each checking account.

Loans

A loan originates at a particular branch and can be held by one or more customers. A loan is identified by a unique loan number (similar to an account number). For each loan, the bank keeps track of the loan amount and the loan monthly re-payment amount.

Transactions

The bank keeps track of all the transactions. A transaction is identified by a unique code and has a type of name. For example, "WD" is the code for withdrawal, and "CD" is for customer deposit. When a

customer makes a transaction, the transaction record should identify the transaction code, the date, the hour, the amount, and the account. Some transactions are free but the bank charges for most of them. If a customer makes a chargeable transaction,the charge is also registered as a chargeless transaction.

Entity-Relationship and Relational Database Design

For our Entity-Relationship Design, complementary to the given specifications, we had to make a number of assumptions. These assumptions allowed us to specify relationship constraints and the type of relationship between entities. Additionally, they allowed us to choose the relationships that are and aren't represented in the ER diagram, choose the type of data that is stored in each attribute, and add any other attributes that were deemed necessary. The assumptions are listed below:

Assumptions

- A branch must only have one manager and an employee may or may not be a branch manager.
- A branch must have one assistant manager; an employee may or may not be an assistant branch manager.
- A customer must hold at least one account; an account must be held by at least one customer.
- The customer's personal banker is a part of the branch the customer is associated with; therefore, customer branch association isn't represented as a separate relationship.
- An employee may be a personal banker for one or more customers or no customers; a customer may or may not have a personal banker.
- An employee may or may not have one or more dependents, but a dependent must be related to only one employee.
- A branch must have at least 2 employees (manager and assistant manager); an employee must work for a single branch.
- An employee may or may not have a manager; an employee may or may not manage employees and if they do, it may be multiple employees.
- Managing a branch and managing employees are two separate relationships; this was assumed due to confusing wording and typos in the Database Design Requirements.
- A loan must originate from a branch; a branch may have 0 or more loans that originate from it.
- A transaction must be made by a single account; an account may have 0 or more transactions associated with it.
- While transactions are made by customers, the specific customer who made it does not need to be tracked in the database.
- Chargeable transactions are recorded as two separate transactions; the first is the actual transaction while the second is the charge, thereby treating them both as chargeless transactions. Charges will be represented with their own codes.
- The overdrafts attribute of a checking account refers to the amount of times a transaction is executed by a customer for which they do not have sufficient funds (counter).
- Transaction_id is a unique attribute we created that is assigned to transactions to distinguish them from other transactions as transaction codes aren't unique since codes are assigned based on the types of transactions being made.

These assumptions and specifications outlined in the business requirements coalesce to become the Enhanced Entity Relationship Diagram seen below (Figure 1). We chose to do an EER rather than a simpler ER in order to represent Loan, Savings, Checking, and Money_Market entities as Account entities. This was then translated into a Relational Schema, also seen below (Figure 2).

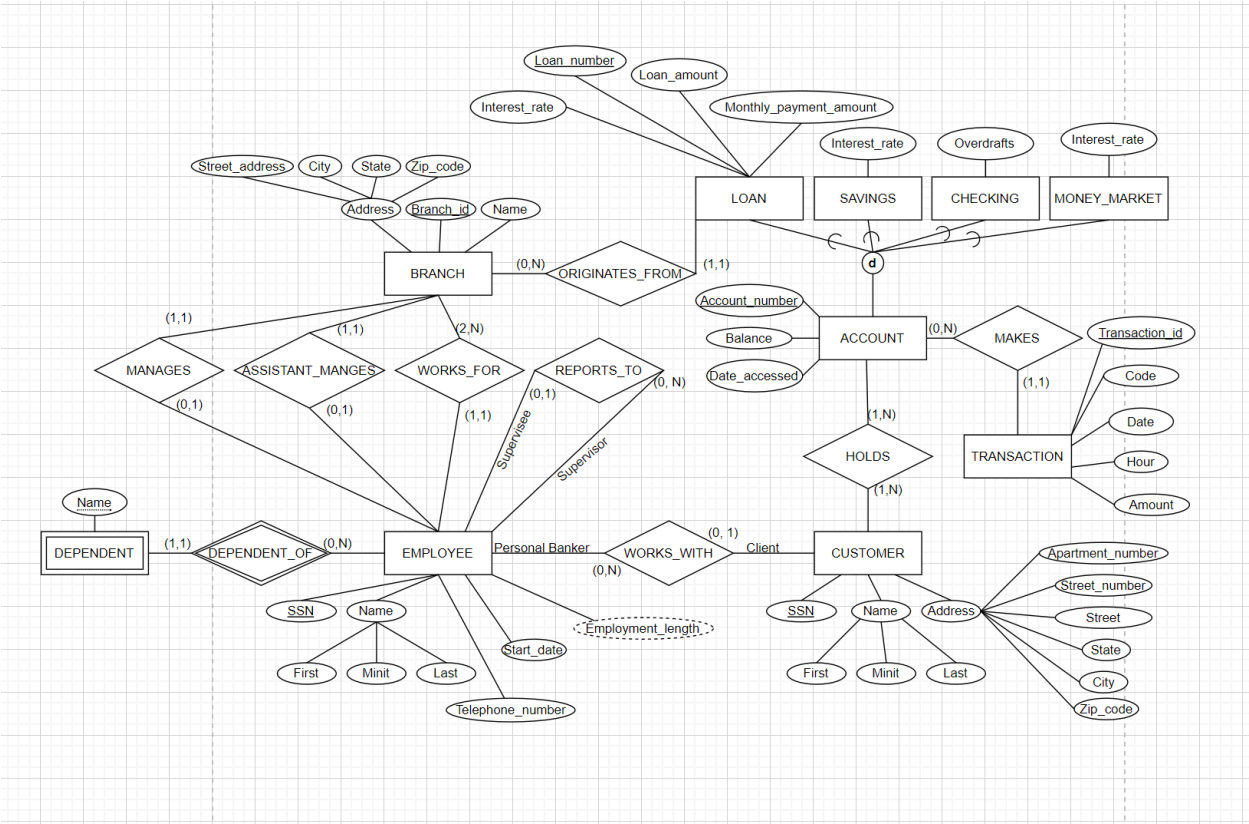


Figure 1: Bank EER Diagram

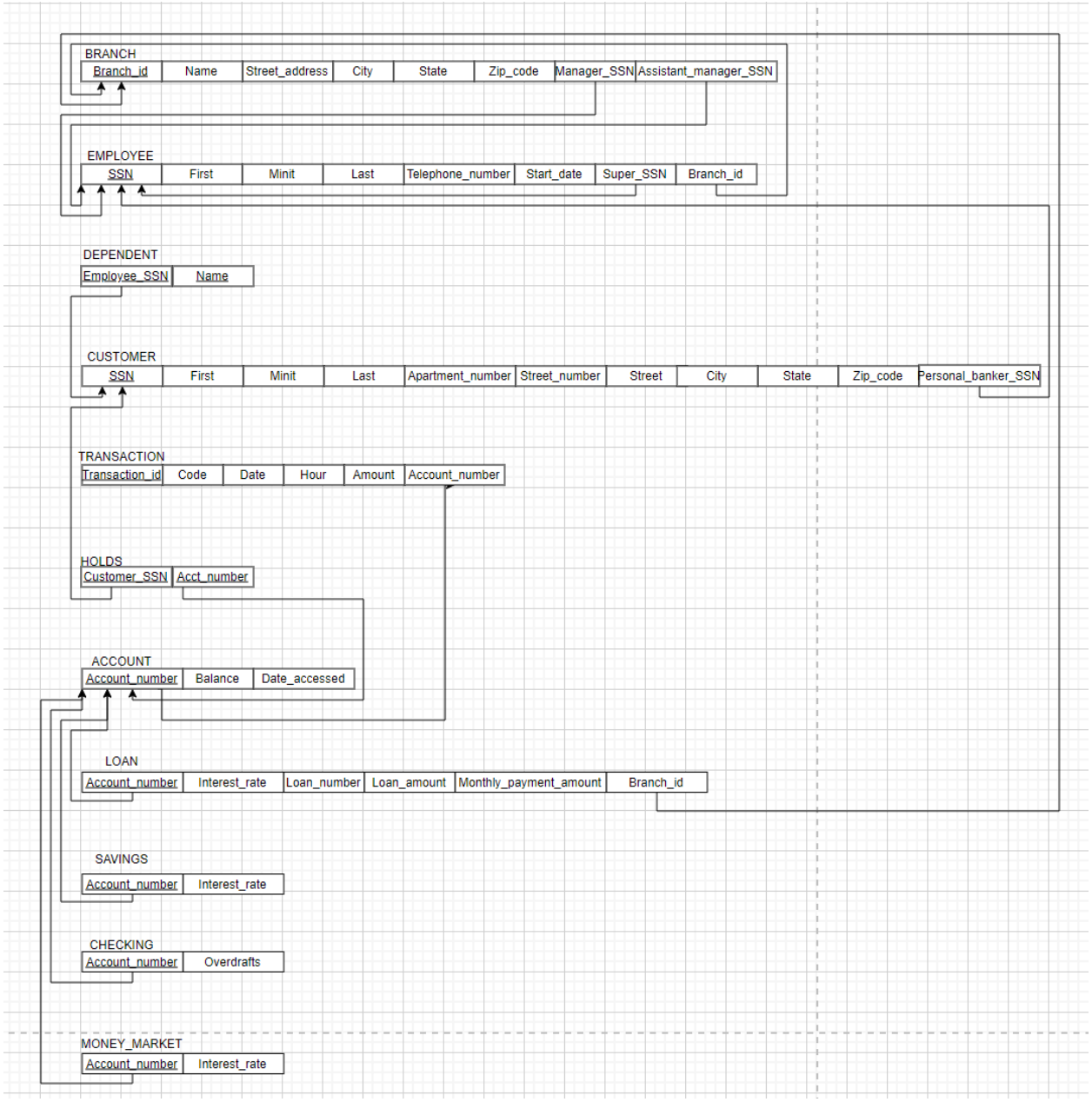
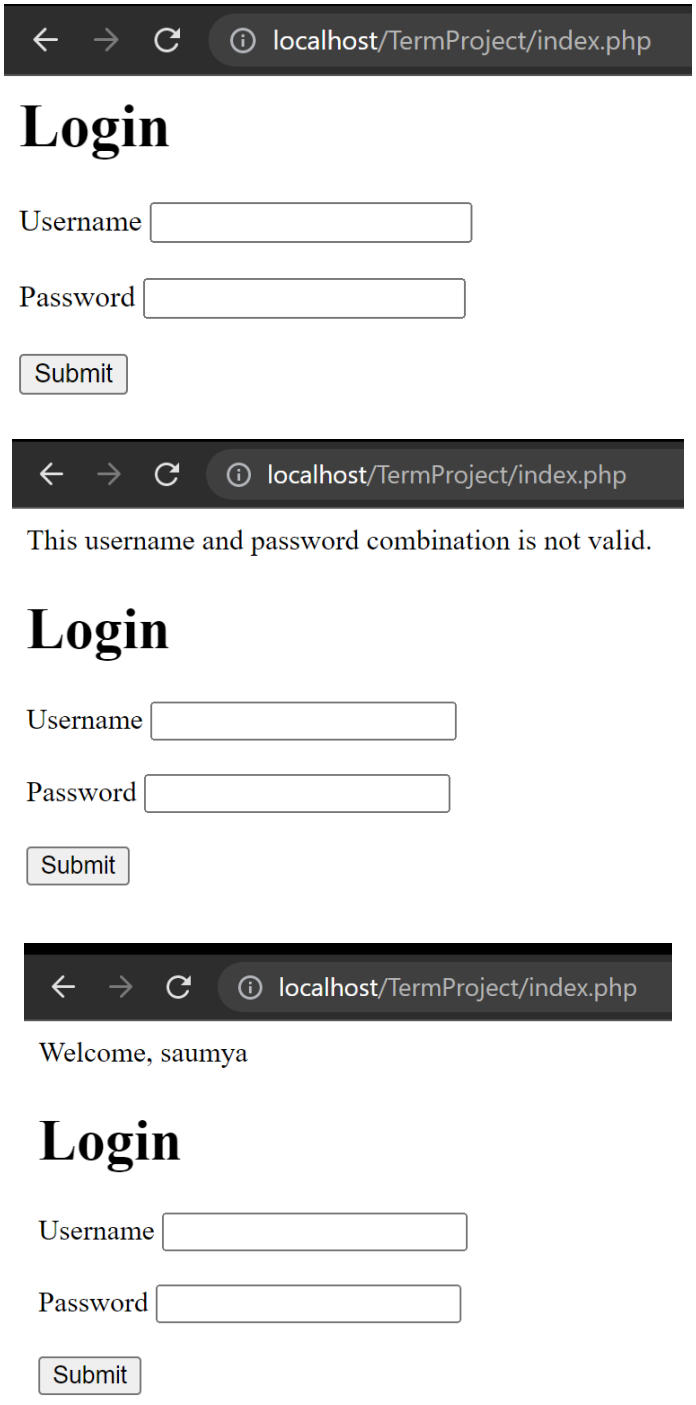


Figure 2: Bank Relational Schema from EER Diagram

Application Program Design

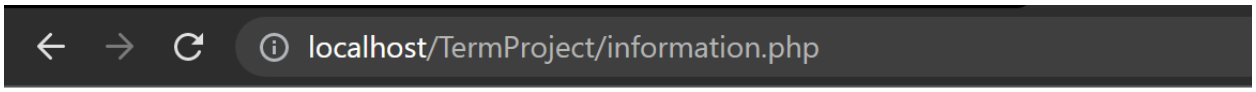


Retrieve the names of customers along with how many accounts they hold

Generate

Retrieve the names of employees who are personal bankers for more than two customers

Generate



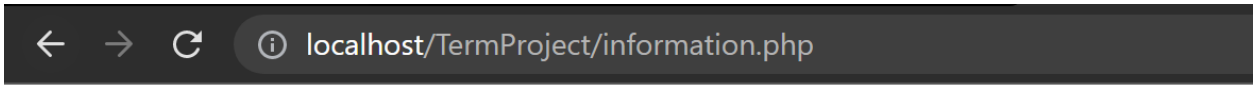
Customer	Number of Accounts
Barrack	4
Bejamin	4
Himothy	4
John	4
Thomas	4

Retrieve the names of customers along with how many accounts they hold

Generate

Retrieve the names of employees who are personal bankers for more than two customers

Generate



Employee Number of Customers

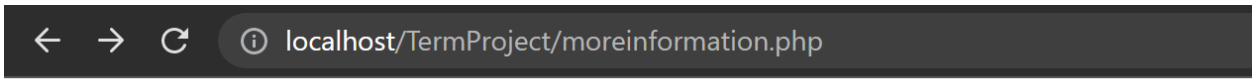
Richard 3

Retrieve the names of customers along with how many accounts they hold

Generate

Retrieve the names of employees who are personal bankers for more than two customers

Generate

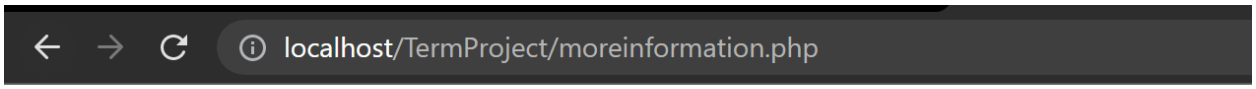


Retrieve the name of the employee who is responsible for all the dependents

Generate

Retrieve the account numbers of all accounts held by a customer with the first name “Himothy”

Generate



Employee

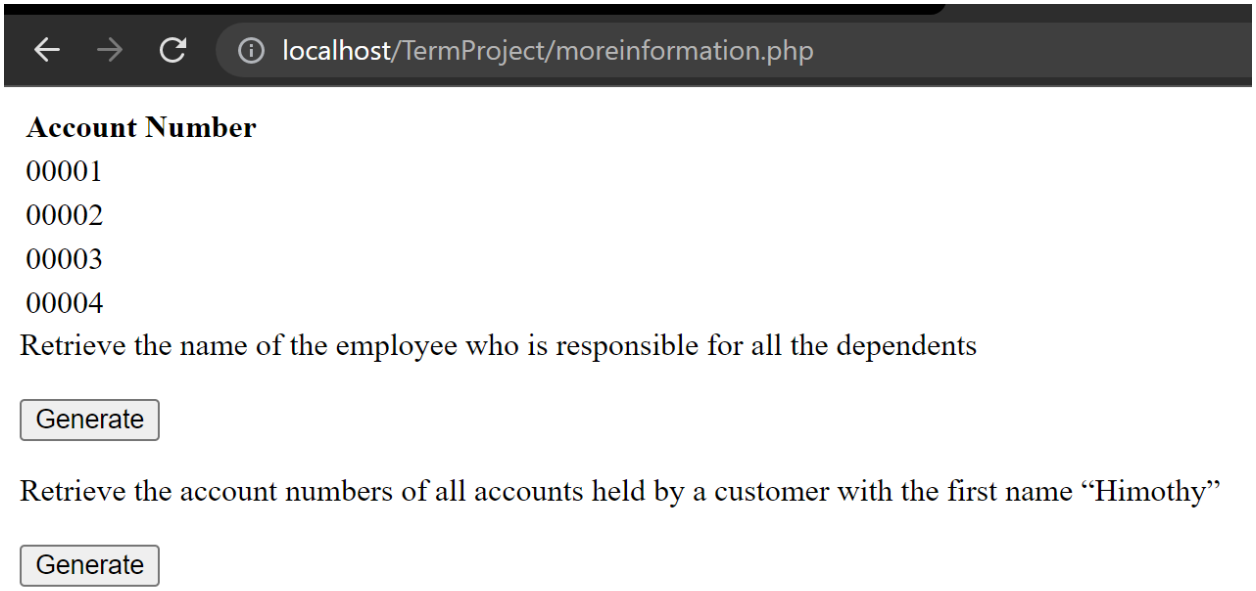
Bruce

Retrieve the name of the employee who is responsible for all the dependents

Generate

Retrieve the account numbers of all accounts held by a customer with the first name “Himothy”

Generate



Shown above are images of the 3 web pages that were designed to interface with the database. The first 3 images show the login page without anything happening, with invalid username/password submitted, and with valid data submitted. The next 3 images show the second web page without anything happening, with the first generate button clicked, and the second generate button clicked. Finally, the last three images show the third web page without anything happening, with the first generate button clicked, and with the second generate button clicked.

Normalization of Relations

	ACCOUNT_NUMBER	BALANCE	DATE_ACCESSED
1	00001	1000	18-JUN-22
2	00002	2000	20-JUN-22
3	00003	3000	27-JUN-22
4	00004	4000	17-JUL-22
5	00005	5000	18-JUL-22
6	00006	6000	19-JUL-22
7	00007	7000	24-JUL-22
8	00008	8000	25-JUL-22
9	00009	9000	29-JUL-22
10	00010	10000	01-AUG-22
11	00011	11000	26-SEP-22
12	00012	12000	03-OCT-22
13	00013	13000	04-OCT-22
14	00014	14000	09-OCT-22
15	00015	15000	12-OCT-22
16	00016	16000	18-OCT-22
17	00017	17000	20-OCT-22
18	00018	18000	21-OCT-22
19	00019	19000	22-OCT-22
20	00020	20000	03-NOV-22

This is the ACCOUNT relation. Account_number is the primary key for the relation. There are no foreign keys. In this relation, all other attributes are functionally determined by Account_number. Since no partial-key or transitive dependencies exist, the relation is in 3NF.

	BRANCH_ID	NAME	STREET_ADDRESS	CITY	STATE	ZIP_CODE	MANAGER_SSN	ASSISTANT_MANAGER_SSN
1	001	South Salem	223 Appleton Street	Salem	MA	01970	000000001	000000002
2	002	North Salem	543 Pear Street	Salem	MA	01970	000000003	000000004
3	003	East Salem	784 Orange Street	Salem	MA	01970	000000005	000000006
4	004	Newark	100 High Street	Newark	NJ	07102	000000007	000000008
5	005	New Brunswick	120 Low Street	New Brunswick	NJ	08901	000000009	000000010

This is the BRANCH relation. Branch_id is the primary key for this relation. The foreign keys are Manager_ssn and Assistant_manager_ssn, which both reference Ssn from the EMPLOYEE relation. In this relation, all attributes are functionally determined by the Branch_Id. Assuming that different branches can have the same name and branches can share the same address, there are no transitive functional dependencies; this means that neither Name nor Street_address determine the rest of the attributes. This is a reasonable assumption because different branches may have the same name if they're located in a town with the same name but in a different location. Additionally, two branches may be located at the same address as they coexist at that location. Therefore, since no partial-key or transitive dependencies exist, the relation is in 3NF.

ACCOUNT_NUMBER	OVERDRAFTS
00003	0
00007	2
00011	0
00015	1
00019	0

This is the CHECKING relation. Account_number is the primary key for this relation. Additionally, Account_number is a foreign key that references Account_number from the ACCOUNT relation. In this relation, Account_number functionally determines all other attributes. Therefore, since no partial-key or transitive dependencies exist, the relation is in 3NF.

	SSN	FIRST	MINIT	LAST	APARTMENT_NUMBER	STREET_NUMBER	STREET	CITY	STATE	ZIP_CODE	PERSONAL_BANKER_SSN
1	100000000	Himothy	W	Wins	(null)	432	Rainbow Road	Florence	MA	45223	000000002
2	200000000	Thomas	R	Jefferson	(null)	578	Stain Lane	Happytown	MA	45224	000000002
3	300000000	Bejamin	E	Franklin	(null)	598	Candy Road	Greenville	MA	45225	000000002
4	400000000	John	P	Adams	(null)	403	Pacman Avenue	Whoville	MA	45226	000000001
5	500000000	Barrack	O	Obama	(null)	306	Lake Avenue	Auburn	MA	45227	000000001

This is the CUSTOMER relation. Ssn is the primary key for this relation. Personal_banker_ssn is a foreign key that references Ssn from the EMPLOYEE relation. In this relation, Ssn functionally determines all other attributes. Since full names can be the same amongst multiple people, no partial-key or transitive functional dependencies exist. Therefore, the relation is in 3NF.

	EMPLOYEE_SSN	N...
1	000000001	Tim
2	000000001	Barbara
3	000000001	Ace
4	000000001	Jim
5	000000001	Jason

This is the DEPENDENT relation. The primary key is a combination of the Employee_ssn and Name. Employee_ssn is also a foreign key that references Ssn from the EMPLOYEE relation. Since there are no other attributes besides the composite key and the entity is weak, there aren't any functional dependencies. Therefore, the relation is in 3NF.

	SSN	FIRST	MINIT	LAST	TELEPHONE_NUMBER	START_DATE	SUPER_SSN	BRANCH_ID
1	000000001	Bruce	T	Wayne	1111111111	24-JAN-21	(null)	001
2	000000002	Richard	L	Grayson	2222222222	18-FEB-21	000000001	001
3	000000005	Tony	I	Stark	5555555555	09-JUL-21	(null)	003
4	000000006	Bruce	H	Banner	6666666666	19-JUL-21	000000005	003
5	000000007	Thor	O	Odinson	7777777777	28-JUL-21	(null)	004
6	000000008	Loki	O	Odinson	8888888888	31-JUL-21	000000007	004
7	000000009	Peter	S	Parker	9999999999	19-AUG-21	(null)	005
8	000000010	Flash	L	Thompson	1010101010	20-AUG-21	000000009	005
9	000000003	Clark	S	Kent	3333333333	17-MAR-21	(null)	002
10	000000004	Diana	W	Prince	4444444444	26-MAY-21	000000003	002

This is the EMPLOYEE relation. The primary key is Ssn. Super_ssn is a foreign key that references Ssn from the EMPLOYEE relation (itself). Additionally, Branch_id is a foreign key that references Branch_id

from the BRANCH relation. In this relation, Ssn functionally determines all other attributes. As previously stated, names can be identical, so they do not cause a transitive functional dependency. We will also say that employees can share the same telephone number; for example, a married couple that uses the same home phone as their number. Additionally, the supervisor of an employee is independent of the branch that the employee is a part of. Therefore, we can say that no partial-key or transitive dependencies exist, so the relation is in 3NF.

	CUSTOMER_SSN	ACCT_NUMBER
1	100000000	00001
2	100000000	00002
3	100000000	00003
4	100000000	00004
5	200000000	00005
6	200000000	00006
7	200000000	00007
8	200000000	00008
9	300000000	00009
10	300000000	00010
11	300000000	00011
12	300000000	00012
13	400000000	00013
14	400000000	00014
15	400000000	00015
16	400000000	00016
17	500000000	00017
18	500000000	00018
19	500000000	00019
20	500000000	00020

This is the HOLDS relation. The primary key is a combination of Customer_ssn and Acct_number. Customer_ssn is a foreign key that references Ssn from the CUSTOMER relation. Acct_number is a foreign key that references Account_number from the ACCOUNT relation. In this relation, there aren't any attributes besides the composite key, so no partial-key or transitive functional dependencies exist, thereby making the relation in 3NF.

	ACCOUNT_NUMBER	INTEREST_RATE	LOAN_NUMBER	LOAN_AMOUNT	MONTHLY_PAYMENT_AMOUNT	BRANCH_ID
1	00001	8	10101	100000	5000	001
2	00005	9	50505	200000	10000	001
3	00009	8	90909	300000	15000	001
4	00013	9	13013	400000	20000	001
5	00017	8	17017	500000	25000	001

This is the LOAN relation. The primary key is Account_number. However, Account_number is also a foreign key that references Account_number in the ACCOUNT relation. Additionally, Branch_id is a foreign key that references Branch_id from the BRANCH relation. In this relation, Account_number functionally determines all other attributes. However, Loan_number also functionally determines all other attributes. To change this relation into 3NF, we would separate this relation into two separate ones. The

first one would contain Account_number and Loan_number. The second one would contain Loan_number along with Interest_rate, Loan_amount, Monthly_payment_amount, and branch_id. However, a more reasonable idea is just to remove Loan_number because it is redundant, using Account_number instead. After this change, the relation would be in 3NF.

	ACCOUNT_NUMBER	INTEREST_RATE
1	00004	8
2	00008	7
3	00016	7
4	00020	8
5	00012	8

This is the MONEY_MARKET relation. The primary key is Account_number. Account_number is also a foreign key that references Account_number in the ACCOUNT relation. In this relation, only Account_number functionally determines all other attributes. Therefore, there are no partial-key or transitive functional dependencies, thereby making the relation in 3NF.

	ACCOUNT_NUMBER	INTEREST_RATE
1	00002	2
2	00006	3
3	00010	2
4	00014	3
5	00018	2

This is the SAVINGS relation. The primary key is Account_number. Account_number is also a foreign key that references Account_number in the ACCOUNT relation. In this relation, only Account_number functionally determines all other attributes. Therefore, there are no partial-key or transitive functional dependencies, thereby making the relation in 3NF.

	TRANSACTION_ID	CODE	DATETIME	AMOUNT	ACCOUNT_NUMBER
1	11111	CD	15-NOV-22	100	00003
2	22222	WD	15-NOV-22	5	00003
3	33333	CD	15-NOV-22	200	00003
4	44444	WD	15-NOV-22	10	00003
5	55555	CD	15-NOV-22	300	00003

This is the TRANSACT relation. The primary key is Transaction_id. Account_number is a foreign key that references Account_number in the ACCOUNT relation. In this relation, Transaction_id functionally determines all other attributes of the relation. Therefore, no other partial-key or transitive functional dependencies exist, thereby making this relation in 3NF.

SQL Queries

#1: Retrieve the names of customers along with how many accounts they hold

```
SELECT C.First, COUNT(*)
FROM CUSTOMER C, HOLDS H
WHERE C.SSN = H.Customer_SSN
GROUP BY C.First;
```

	FIRST	COUNT(*)
1	Thomas	4
2	Bejamin	4
3	Himothy	4
4	John	4
5	Barrack	4

#2: Retrieve the names of employees who are personal bankers for more than two customers and how many employees they are personal bankers for

```
SELECT E.First, COUNT(*)
FROM CUSTOMER C, EMPLOYEE E
WHERE C.Personal_banker_SSN = E.SSN
GROUP BY E.First
HAVING COUNT(*) > 2
```

	FIRST	COUNT(*)
1	Richard	3

#3: Retrieve the name of the employee who is responsible for all the dependents

```
SELECT E.First
FROM EMPLOYEE E
WHERE E.SSN = ALL
    (SELECT D.Employee_SSN
    FROM DEPENDENT D);
```

	FIRST
1	Bruce

#4: Retrieve the account numbers of all accounts held by a customer with the first name “Himothy”

```
SELECT A.Account_number
FROM ACCOUNT A
WHERE A.Account_number IN
    (SELECT H.Acct_number
    FROM CUSTOMER C, HOLDS H
    WHERE C.SSN = H.Customer_SSN AND C.First = 'Himothy');
```

	ACCOUNT_NUMBER
1	00001
2	00002
3	00003
4	00004

Conclusion

For Phase 1, the most difficult part of the project was the Entity-Relationship Design. Due to instruction ambiguity and differing interpretations of the business requirements, we had to spend a lot of time discussing and making assumptions before even creating the diagram. However, since the Entity-Relationship diagram was already created, creating the Relational Schema was relatively easy as it was a mere translation of the ER diagram.

For Phase 2, the work was more tedious than difficult. Writing out the SQL queries to create each database and insert data was very time consuming. The difficulty arose in working with SQL Developer, trying to debug SQL code, run queries while maintaining referential integrity constraints or disabling them when necessary, and so on. However, writing simple and complex queries for this phase was relatively quick and easy.

For Phase 3, the biggest hurdle was figuring out how to connect the database to the web pages. Using PHP to connect the two also proved to be difficult as it was a language that not all of us had worked with

prior to taking this class. By utilizing given resources and online documentation, we were able to successfully overcome these challenges.

While we came into this class having some experience with databases, we did not expect to receive such comprehensive knowledge about database system design and management. We also did not expect to end up actually applying what we learned to create usable web pages that interface with the database, but learning how to do so was a valuable experience.

If we would do it all over again, we would definitely spend more time communicating while working on each phase of the project. Additionally, knowing the future phases and having complete knowledge about databases now would allow us to best complete each phase, so that work from one could translate to the next. Overall, we had a great time not only learning but applying what we learned.

Note: PHP web pages are attached as part of submission on Canvas.