**Business Requirements:**

I am assuming a scenario to create a database for a Bank application. We have the following requirements:

1. A customer can open any one or more of the following types of accounts:

a. Savings account

b. Checking account

2. The system must be able to store the following customer data:

a. Name

b. Address consisting of the following

(i) House number

(ii) Apartment number (if any)

(iii) Street Number / Name

(iv) State, and Country

3. The system must be able to store the current account balances

4. An account can be a joint account with any number of account holders

5. A customer can open more than one account with the bank

6. A customer must be able to enter one or more addresses (like primary, secondary, temporary etc.). The address type need not be maintained in the system, since the bank sends correspondence to all the addresses associated to a customer.

**Brainstorming:**

So we have to maintain the following information in our database:

* ACCOUNT TYPE - Can be SAVINGS, and CHECKING, and the user should be able to enter new account types or delete existing account types, as needed. For instance, if the bank offers a new product called CREDIT CARD, then the bank should be able to quickly create such business product
* CUSTOMER CONTACT INFO consisting of CUSTOMER NAME, HOUSE NUMBER, STEET NAME, APARTMENT NUMBER, CITY, STATE, COUNTRY, ZIP)
* ACCOUNT DETAILS - ACCOUNT NUMBER, ACCOUNT TYPE, ACCOUNT HOLDERS INFORMATION and ACCTOUNT BALANCE

A customer can have more than one account and an account can be held by more than one customer (joint accounts). So this is a many to many relationship.

There are 2 types of data models. One is the logical model and the other is the physical model. In logical Model, we represent the models following the business requirements (such as identifying the types of relationships, unique keys). In physical data model, we represent the data types, primary keys, associate tables etc. The physical data model is highly dependent on the underlying RDBMS software you plan to use.

Another major important difference between physical and logical data model is the representation of Many-to-Many relationships. In logical model, we represent the many-to-many relationship directly, but in physical model it is not possible to represent the many-to-many relationships directly. All the many-to-many relationships at the logical level will be changed to One-to-Many relationships. Such conversion is necessary to meet the referential integrity constraints and to avoid the loss of data.

Most of the data modelling software products will automatically change the many to many table relationships (at the logical level) to one-to many relationship at the physical level using an intermediate mapping table (or associate table).

This can be better explained by an example. Let us imagine that we create a table with the 4 attributes CUSTOMER\_NAME, CUSTOMER\_ADDRESS, ACCOUNT\_ID, ACCOUNT\_TYPE and ACCOUNT\_BALANCE. Our idea is to represent the customer information, along with the accounts s/he is holding with the bank. Since the customer names can be same, for 2 different customers, let us use a column called CUSTOMER\_ID (it can be SSN - Social Security Number) to uniquely identify the customer. So we will have the following columns:

CUSTOMER\_ID, CUSTOMER\_NAME, CUSTOMER\_ADDRESS, ACCOUNT\_ID, ACCOUNT\_TYPE, and ACCOUNT\_BALANCE. We can use CUSTOMER\_ID and ACCOUNT\_ID combination to address each row individually in the table. So the primary key for that table is (CUSTOMER\_ID ACCOUNT\_ID). However, such table design (with columns: CUSTOMER\_ID, CUSTOMER\_NAME, CUSTOMER\_ADDRESS, ACCOUNT\_ID, ACCOUNT\_TYPE, ACCOUNT\_BALANCE) cannot accommodate the scenario, where customer details need to be maintained even though s/he does not have any accounts with the bank. For instance some banks maintain the customer information, even after an account has been closed, for future auditing purpose or for legal needs.

Also, we must NOTE that, the ACCOUNT\_TYPE details cannot be maintained completely. For example, if none of the existing customers have SAVINGS account type, how can we capture the information that the bank offers SAVINGS account? So the table design CUSTOMER\_ID, CUSTOMER\_NAME, CUSTOMER\_ADDRESS, ACCOUNT\_ID, ACCOUNT\_TYPE, and ACCOUNT\_BALANCE must be decomposed into 6 tables (obtained by applying the rules of normalization):

(CUSTOMER\_ID, CUSTOMER\_NAME, ADDRESS\_ID)

(CUSTOMER\_ID, ADDRESS\_ID)

(ADDRESS\_ID, ADDRESS)

(ACCOUNT\_ID, ACCOUNT\_TYPE, ACCTOUNT\_BALANCE)

(CUSTOMER\_ID, ACCOUNT\_ID)

(ACCOUNT\_ID, ACCOUNT\_TYPE)

The above 6 tables must be able to capture all the possible scenarios:

1. Maintaining a customer’s information even though he has NO account

2. Maintaining an account type, even though no accounts currently exists (related to an account type).

3. Ability to maintain more than one address for a customer

The 6 tables listed above were obtained by applying the “Normalization process”, which basically says to group the relevant data together, so that all the data can be represented without any data loss.

There are three important rules of Normalization:

**First Normal Form:** Each row in a table must be uniquely identified. In other words, we must have primary key on a table. If a table has primary key then it abides the first normal form

**Second Normal Form:** If the Primary Key has more than one column, then there must not be any partial dependencies of non-unique columns on the parts of Primary key columns.

For instance, in our above example, the (CUSTOMER\_ID, ACCOUNT\_ID) combination is the Primary Key. But the column ACCOUNT\_BALANCE is dependent on ACCOUNT\_ID, and not on CUSTOMER\_ID. Since a customer can have more than one account, how can you get the account balance of a particular account, just using the CUSTOMER\_ID? Therefore the account balances are dependent on ACCOUNT\_ID, but not on the CUSTOMER\_ID. Hence it violates the second Normal form. NOTE that if a table is in 2nd Normal Form, then it is already in 1st Normal Form. Following this rule, we obtain the tables (CUSTOMER\_ID, CUSTOMER\_ADDRESS), with CUSTOMER\_ID as the primary key, and (ACCOUNT\_ID, ACCOUNT\_TYPE, ACCT\_BALANCE) table with ACCOUNT\_ID as the primary key.

**Third Normal Form:** The non-keys should not be dependent on other non-keys or non-keys cannot be independent. In our above example (ACCOUNT\_ID, ACCOUNT\_TYPE, ACCTOUNT\_BALANCE), the column ACCOUNT\_TYPE is not dependent on the primary Key (ACCOUNT\_ID)

The ACCOUNT\_TYPE is an independent attribute. If we represent the ACCOUNT\_TYPE in the table (ACCOUNT\_ID, ACCOUNT\_TYPE, ACCT\_BAL), then there is a potential chance that we cannot represent all the available account types in the bank. For instance, if none of the customers have SAVINGS account type, then we would not be able to represent the SAVINGS Account type in our database. Hence we have to take maintain the ACCOUNT\_TYPE in a separate table. But the ACCOUNT table will still have ACCOUNT\_TYPE as the foreign key

We have the following tables, till now:

(CUSTOMER\_ID, CUSTOMER\_NAME, CUSTOMER\_ADDRESS)

(ACCOUNT\_ID, ACCOUNT\_TYPE, ACCOUNT\_BALANCE)

(ACCOUNT\_TYPE)

But how can we associate the account to a customer? We need a table called Associate table, which links the customers with their accounts. Also we must have another table to map the accounts to the account types. Now we will have the following tables:

(CUSTOMER\_ID, CUSTOMER\_NAME, CUSTOMER\_ADDRESS)

(ACCOUNT\_ID, ACCOUNT\_TYPE, ACCOUNT\_BAL)

(ACCOUNT\_TYPE)

(CUSTOMER\_ID, ACCOUNT\_ID)

The above 4 tables cannot enable a customer to represent more than one address. In the table (CUSTOMER\_ID, CUSTOMER\_ADDRESS), the CUSTOMER\_ID is the primary key, and more than one address cannot be represented in this table. To meet this requirement, the table (CUSTOMER\_ID, CUSTOMER\_NAME, CUSTOMER\_ADDRESS) is decomposed into (ADDRESS\_ID, ADDRESS), and (CUSTOMER\_ID, ADDRESS\_ID) and (CUSTOMER\_ID, CUSTOMER\_NAME). Finally we will have the following tables:

Following the rules of normal form, I obtained the following 6 tables. The data model diagrams (ER-diagrams) are provided as separate file in the github directory <https://github.com/msekhar12/Week_3_Assignment_3_607.git>

NOTE: To make the explanation simple, I did not include all the columns that will be used in the actual data model. But the below SQL code and the data models provided will have additional columns. Both the data model and SQL files are provided as separate files.

**Table 1: ACCOUNT\_TYPE (ACCOUNT\_CATEGORY, ACCOUNT\_CATEGORY\_NAME)**

The ACCOUNT\_TYPE table will represent the types of accounts that the bank can offer its customer to open. It will have 2 attributes: ACCOUNT\_CATEGORY represented by an Integer data type, and ACCOUNT\_CATEGORY\_NAME represented by CHAR(100). The ACCOUNT\_CATEGORY is the primary key for the table.

**Table 2: CUSTOMER (SSN, CUSTOMER\_FIRST\_NAME, CUSTOMER\_MIDDLE\_NAME,**

**CUSTOMER\_LAST\_NAME)**

The CUSTOMER Table represent the customer information, such as SSN (Social Security Number), first name, middle name (if any), and last name. SSN is used as the primary key for this table.

**Table 3: ACCOUNT (ACCOUNT\_ID, ACCOUNT\_CATEGORY, ACCOUNT\_BALANCE)**

The ACCOUNT Table will maintain the account details such as ACCOUNT\_ID, ACCOUNT\_CATEGORY, and ACCOUNT\_BALANCE. The primary key is ACCOUNT\_ID. The ACCOUNT\_CATEGORY is the foreign key to referencing the table ACCOUNT\_TYPE(ACCOUNT\_CATEGORY). Between the two tables: ACCOUNT\_TYPE and ACCOUNT, we will have a one to many relationships (each account from account table will refer one account type, and each account type in account category table will have zero or more accounts)

Between Customer and Account tables, we have many-to-many relationship (a customer can open multiple accounts and an account can be held by more than one customer – a joint account)

**Table 4: CUSTOMER\_ACCOUNT (SSN, ACCOUNT\_ID)**

The CUSTOMER\_ACCOUNT Table is an associate table, mapping linking the customer with accounts. NOTE that this table will be visible only at the Physical level.

**Table 5: ADDRESS (ADDRESS\_ID, HOUSE\_NUMBER, APT\_NUMBER, STREET, CITY, STATE, COUNTRY)**

The ADDRESS table will represent the addresses. It is uniquely identified by ADDRESS\_ID. Since a customer can have more than one address, and also an address can be shared by more than one customer (for example, if a joint account is held by wife and husband), we need to mention the ADDRESS Table separately.

**Table 6: CUSTOMER\_ADDRESS (SSN, ADDRESS\_ID)**

The CUSTOMER\_ADDRESS Table will associate Customer to an address. This way, a customer can maintain more than one address.

Please look into the detailed design diagrams in the github directory “https://github.com/msekhar12/Week\_3\_Assignment\_3\_607.git”

=============================================================================

**Creating the physical database objects (in PostgreSQL):**

=============================================================================

Following the above rules of Normalization, we get the following tables for the Banking application.

**Create the database first:**

Using pgAdmin tool, create a database using the GUI. The database name is BANK\_DB. Once the database is created, execute the following Data Definition Language statements:

--1. Create an account type table, to represent the possible account types at the bank. As per the business requirement, there are 2 types of accounts at the bank – SAVINGS and CHECKING. But the user can insert new account types, if needed in future into this table.

**CREATE TABLE ACCOUNT\_TYPE (ACCOUNT\_CATEGORY INT NOT NULL, ACCOUNT\_CATEGORY\_NAME CHAR(100) NOT NULL,**

**CONSTRAINT ACCOUNT\_TYPE\_PK PRIMARY KEY (ACCOUNT\_CATEGORY));**

--2. Create a customer table. The customer is uniquely identified by his/her SSN (Social Security Number). The customer name details are also maintained in this table

**CREATE TABLE CUSTOMER(**

**SSN BIGINT NOT NULL,**

**CUSTOMER\_FIRST\_NAME CHAR(60) NOT NULL,**

**CUSTOMER\_MIDDLE\_NAME CHAR(60),**

**CUSTOMER\_LAST\_NAME CHAR(60) NOT NULL, CONSTRAINT CUSTOMER\_PK PRIMARY KEY (SSN));**

--3. Create ACCOUNT Table (The account table maintains the ACCOUNT\_ID, ACCOUNT TYPE and ACCOUNT BALANCE

**CREATE TABLE ACCOUNT(ACCOUNT\_ID BIGINT NOT NULL,**

**ACCOUNT\_CATEGORY INT NOT NULL,**

**ACCOUNT\_BALANCE DECIMAL(30,2) NOT NULL,**

**CONSTRAINT ACCOUNT\_PK PRIMARY KEY (ACCOUNT\_ID),**

**CONSTRAINT ACCOUNT\_FK\_1 FOREIGN KEY (ACCOUNT\_CATEGORY)**

**REFERENCES ACCOUNT\_TYPE(ACCOUNT\_CATEGORY));**

--4. Create CUSTOMER - ACCOUNT Table, to map customers to accounts.

**CREATE TABLE CUSTOMER\_ACCOUNT (SSN BIGINT NOT NULL, ACCOUNT\_ID BIGINT NOT NULL,**

**CONSTRAINT CUSTOMER\_ACCOUNT\_PK PRIMARY KEY (SSN, ACCOUNT\_ID),**

**CONSTRAINT CUSTOMER\_ACCOUNT\_FK1 FOREIGN KEY (SSN) REFERENCES CUSTOMER(SSN),**

**CONSTRAINT CUSTOMER\_ACCOUNT\_FK2 FOREIGN KEY (ACCOUNT\_ID) REFERENCES ACCOUNT(ACCOUNT\_ID));**

--5. Create ADDRESS Table to hold addresses. Each row is uniquely identified by Address ID.

**CREATE TABLE ADDRESS(ADDRESS\_ID BIGINT NOT NULL, HOUSE\_NUMBER VARCHAR(254) NOT NULL,**

**APT\_NUMBER VARCHAR(254), STREET VARCHAR(254) NOT NULL, CITY VARCHAR(254) NOT NULL,**

**STATE CHAR(2) NOT NULL, COUNTRY CHAR(60) NOT NULL, ZIP CHAR(10) NOT NULL, CONSTRAINT ADDRESS\_PK PRIMARY KEY (ADDRESS\_ID)**

**);**

--6. Create a table to map CUSTOMER - ADDRESS. Note that a CUSTOMER can have two types of addresses - Primary and Secondary.

-- Also NOTE That I am using a CHECK Constraint to restrict the ADDRESS\_TYPE to P or S. P for Primary and S for Secondary.

**CREATE TABLE CUSTOMER\_ADDRESS (SSN BIGINT NOT NULL, ADDRESS\_ID BIGINT NOT NULL, ADDRESS\_TYPE CHAR(1) NOT NULL,**

**CONSTRAINT CUSTOMER\_ADDRESS\_PK PRIMARY KEY(SSN,ADDRESS\_ID, ADDRESS\_TYPE),**

**CONSTRAINT CUSTOMER\_ADDRESS\_FK1 FOREIGN KEY(SSN) REFERENCES CUSTOMER(SSN),**

**CONSTRAINT CUSTOMER\_ADDRESS\_FK2 FOREIGN KEY(ADDRESS\_ID) REFERENCES ADDRESS(ADDRESS\_ID),**

**CONSTRAINT CUSTOMER\_ADDRESS\_CK1 CHECK(ADDRESS\_TYPE IN ('P','S'))) ;**

=============================================================================

**Inserting sample data into physical tables**

=============================================================================

**INSERT INTO ACCOUNT\_TYPE VALUES(1, 'SAVINGS');**

**INSERT INTO ACCOUNT\_TYPE VALUES(2, 'CHECKING');**

**INSERT INTO CUSTOMER VALUES(123456789,'NICHOLAS',NULL,'PAUL');**

**INSERT INTO CUSTOMER VALUES(123456780,'JOHN','S','ABRAHAM');**

**INSERT INTO CUSTOMER VALUES(123456799,'NANCY','M','CHARLOTTE');**

**INSERT INTO CUSTOMER VALUES(123456710,'CHRIS','M','BOLAS');**

**INSERT INTO ACCOUNT VALUES(123,1,130);**

**INSERT INTO ACCOUNT VALUES(122,1,140);**

**INSERT INTO ACCOUNT VALUES(120,2,1300);**

**INSERT INTO ACCOUNT VALUES(129,2,10);**

**INSERT INTO ACCOUNT VALUES(110,1,1199);**

**INSERT INTO CUSTOMER\_ACCOUNT VALUES(123456789,123);**

**INSERT INTO CUSTOMER\_ACCOUNT VALUES(123456789,122);**

**INSERT INTO CUSTOMER\_ACCOUNT VALUES(123456780,123);**

**INSERT INTO CUSTOMER\_ACCOUNT VALUES(123456799,120);**

**INSERT INTO CUSTOMER\_ACCOUNT VALUES(123456799,129);**

**INSERT INTO CUSTOMER\_ACCOUNT VALUES(123456799,110);**

**INSERT INTO ADDRESS VALUES(1,'123','311','SOMERSET BLVD','NEW YORK', 'NY', 'USA', '08522');**

**INSERT INTO ADDRESS VALUES(2,'311',NULL,'KNIGHT STREET','DALLAS', 'TX', 'USA', '92312');**

**INSERT INTO ADDRESS VALUES(3,'A932','T12','WHITE STREET','BALTIMORE', 'MD', 'USA', '23212');**

**INSERT INTO ADDRESS VALUES(4,'W-835','SUITE-3','FOX RUN AVE','TAMPA', 'FL', 'USA', '89422');**

**INSERT INTO ADDRESS VALUES(5,'R99A',NULL,'168 STREET','SAINT LOUIS', 'MO', 'USA', '34522');**

**INSERT INTO CUSTOMER\_ADDRESS VALUES(123456789,2,'P');**

**INSERT INTO CUSTOMER\_ADDRESS VALUES(123456789,1,'S');**

**INSERT INTO CUSTOMER\_ADDRESS VALUES(123456780,3,'P');**

**INSERT INTO CUSTOMER\_ADDRESS VALUES(123456799,4,'P');**

**INSERT INTO CUSTOMER\_ADDRESS VALUES(123456710,5,'P');**

=============================================================================

**Querying Tables**

=============================================================================

--Querying the data (For many to many relationships). A customer can have multiple accounts and an account can be held by multiple people (joint account):

--This query find all the customers who has accounts, along with the customer names, account balances, you can use the following query:

**SELECT A.SSN, A.CUSTOMER\_FIRST\_NAME, A.CUSTOMER\_MIDDLE\_NAME, A.CUSTOMER\_LAST\_NAME, B.ACCOUNT\_ID, B.ACCOUNT\_BALANCE, D.ACCOUNT\_CATEGORY\_NAME FROM**

**CUSTOMER A, ACCOUNT B, CUSTOMER\_ACCOUNT C, ACCOUNT\_TYPE D**

**WHERE A.SSN = C.SSN AND C.ACCOUNT\_ID = B.ACCOUNT\_ID AND B.ACCOUNT\_CATEGORY = D.ACCOUNT\_CATEGORY;**

--To find the account numbers and their account types (categories), use the following query. This acts as an example for one to many (between ACCOUNT\_TYPE and ACCOUNT tables)

**SELECT A.ACCOUNT\_ID, B.ACCOUNT\_CATEGORY\_NAME FROM ACCOUNT A, ACCOUNT\_TYPE B WHERE A.ACCOUNT\_CATEGORY = B.ACCOUNT\_CATEGORY;**

--Additional query: To find all the customers who has NO account:

**SELECT \* FROM CUSTOMER WHERE SSN NOT IN (SELECT SSN FROM CUSTOMER\_ACCOUNT);**