**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 accounts

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

a. Name

b. Address

(i) House number

(ii) Apartment number (if any)

(iii) Street Number / Name

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 at two addresses or more (one is Primary and the others as secondary)

**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.

CUSTOMER\_NAME, CUST\_CONTACT INFO (including the HOUSE NUMBER, STEET NAME, APARTMENT NUMBER, CITY, STATE, ZIP)

ACCOUNT\_INFO - ACCOUNT NUMBER, ACCOUNT TYPE, ACCOUNT HOLDERS INFO and ACCT\_BAL

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, and we should have the following attributes:

CUSTOMER\_NAME, CUSTOMER\_ADDRESS, ACCOUNT\_ID, ACCOUNT\_TYPE, ACCT\_BAL

For simplicity, let the CUSTOMER\_ADDRESS column represent the CUST\_CONTACT\_INFO(including the HOUSE NUMBER, STEET NAME, APARTMENT NUMBER, CITY, STATE, ZIP)

But such "many to many" relationships cannot be represented in a single table physically. In fact any RDBMS (Relational Database Management Systems) modelling software (such as ER WIN) will decompose such many to many tables (at the logical level) to one-to many relationship at the physical level using an intermediate mapping table. Why it does that?

Because, let us imagine that we create a table with the 4 attributes CUSTOMER\_NAME, CUSTOMER\_ADDRESS, ACCOUNT\_ID, ACCOUNT\_TYPE, ACCT\_BAL. Since the customer names can be same, for 2 different customers, let us use a column called CUST\_ID to uniquely identify the customer. So we will have the following columns:

CUST\_ID, CUSTOMER\_NAME, CUSTOMER\_ADDRESS, ACCOUNT\_ID, ACCOUNT\_TYPE, ACCT\_BAL. So the primary key for that table is (CUST\_ID ACCOUNT\_ID). However, such table design cannot accommodate the scenario, where customer details need to be maintained even though he/she 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 CUST\_ID, CUSTOMER\_NAME, CUSTOMER\_ADDRESS, ACCOUNT\_ID, ACCOUNT\_TYPE, ACCT\_BAL must be decomposed into 4 tables:

(CUST\_ID, CUST\_ADDRESS)

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

(CUST\_ID, ACCTOUNT\_ID)

(ACCT\_TYPE, ACCT\_TYPE\_DESCRIPTION)

The above 4 tables must be able to capture all the possible scenarios:

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

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

The above described process to obtain the 4 tables is called the “Normalization process”, which basically says to group the relevant data together, and separate the irrelevant data.

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 should not be any partial dependencies of non-unique columns on the parts of Primary key columns.

For instance, in our above example, the CUST\_ID and ACCOUNT\_ID is the Primary Key, but the column ACCT\_BAL is dependent on ACCOUNT\_ID, and NOT on CUST\_ID. Since a customer can have more than one account, the account balances are dependent on ACCOINT\_ID, but not on CUST\_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.

**Third Normal Form:** The non-keys should not be dependent on other non-keys. In our above example, the column ACCOUNT\_TYPE is not dependent on the Primary Key values, but it is an independent attribute. This is the reason, why we are not able to maintain all the account types, if none of the existing accounts cover all the possible account types. NOTE that if a table is in 3rd Normal form, then it is already in 1st and 2nd Normal forms.

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

**Creating the physical database objects:**

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

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);**