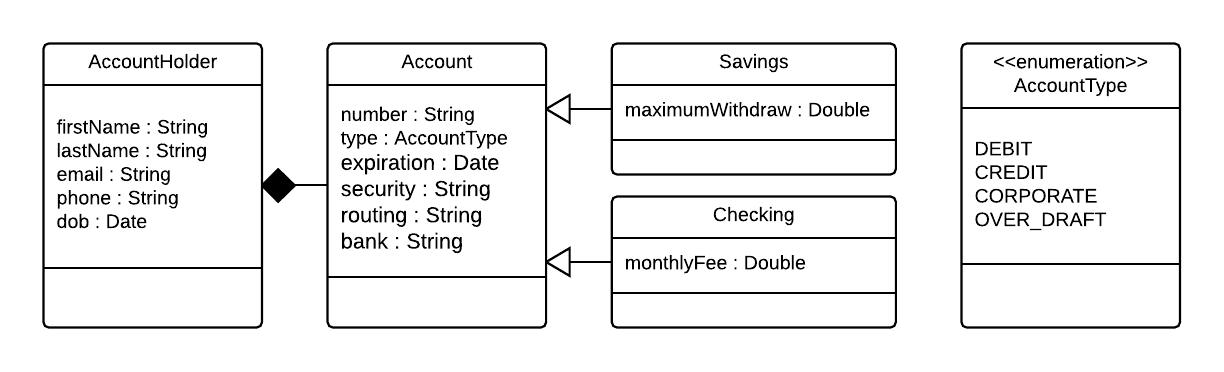
1. Consider a social networking Web site where sponsors can find families and children to sponsor (the sponsored). Both sponsors and sponsored users maintain some personal information such as first and last name, username, email, phone, date of birth, and description about themselves. The sponsored additionally keep track of their zip code, country and continent. All users maintain a public timeline where they can post and share their activities with friends. Posts include a title, date of posting, and content. Users can search for other users and send friend requests. If accepted as friends, users can post to their friend’s timeline and view their friend’s profiles. Users can comment on any post. Comments are similar to posts, but are only visible for a given post, they are not visible in the timeline. Sponsors can set up an account from where to draw funds to donate to their sponsored friends. Accounts are described by account numbers, type (debit card, credit card, bank account), expiration date, security code, routing number, and bank name. Users can donate by clicking on one of several donate buttons with fixed amounts of $1, $5, $10, $20 displayed on the profile of a sponsored user. Additionally users can donate by clicking on like buttons on posts or comments. Each like donates 10 cents from a sponsor user to a sponsored user. Users can view their donation history is various histograms. Users can remove their accounts in which case their personal profile information is removed as well. Posts and donation history are never removed. Create a UML diagram that captures the data and relations described above. Make sure to show cardinality, name your relations, remove redundant and inadequate relationships and attributes, annotate composition, aggregation, and/or generalization.



1. Consider the following UML diagram and provide the equivalent relational model. Do not use the enum data type supported by some databases (use a separate table instead). Use the multiple tables inheritance strategy. Implement all relevant constraints to enforce referential integrity, composition, aggregation, and generalization.  
     
   CREATE TABLE AccountHolder (  
    accountHolderId INT NOT NULL AUTO\_INCREMENT,  
    firstName VARCHAR(50) NOT NULL,  
    lastName VARCHAR(50) NOT NULL,  
    email VARCHAR(50) NOT NULL,  
    phone VARCHAR(10) NOT NULL,  
    dob DATE NOT NULL,  
    CONSTRAINT uq\_AccountHolder UNIQUE (firstName, lastName, dob),  
    CONSTRAINT pk\_AccountHolder\_acountHolder\_id PRIMARY KEY (accountHolderId)  
   );  
     
   CREATE TABLE AccountType (  
    accountType VARCHAR(50) NOT NULL,  
    CONSTRAINT pk\_AccountType\_accountType PRIMARY KEY (accountType)  
   );  
     
   INSERT INTO AccountType(accountType) VALUES (“DEBIT”);  
   INSERT INTO AccountType(accountType) VALUES (“CREDIT”);  
   INSERT INTO AccountType(accountType) VALUES (“CORPORATE”);  
   INSERT INTO AccountType(accountType) VALUES (“OVER\_DRAFT”);  
     
   CREATE TABLE Account (  
    number VARCHAR(50) NOT NULL,  
    accountHolderId INT NOT NULL,  
    type VARCHAR(50) NOT NULL,  
    expiration DATE NOT NULL,  
    security VARCHAR(50) NOT NULL,  
    routing VARCHAR(50) NOT NULL,  
    bank VARCHAR(50) NOT NULL,  
    CONSTRAINT pk\_Account\_number PRIMARY KEY (number),  
    CONSTRAINT fk\_Account\_accountHolderId FOREIGN KEY (accountHolderId)  
    REFERENCES AccountHolder(accountHolderId)  
    ON DELETE CASCADE ON UPDATE CASCADE, # Due to composition relationship.  
    CONSTRAINT fk\_Account\_type FOREIGN KEY (type)  
    REFERENCES AccountType(accountType)  
    ON DELETE CASCADE ON UPDATE CASCADE  
   );  
     
   CREATE TABLE Savings (  
    number VARCHAR(50) NOT NULL,  
    maximumWithdraw DOUBLE(7,2) NOT NULL,  
    CONSTRAINT pk\_Savings\_number PRIMARY KEY (number),  
    CONSTRAINT fk\_Savings\_number FOREIGN KEY (number)  
    REFERENCES Account(number)  
    ON DELETE CASCADE ON UPDATE CASCADE  
   );  
     
   CREATE TABLE Checking (  
    number VARCHAR(50) NOT NULL,  
    monthlyFee DOUBLE(7,2) NOT NULL,  
    CONSTRAINT pk\_Checking\_number PRIMARY KEY (number),  
    CONSTRAINT fk\_Checking\_number FOREIGN KEY (number)  
    REFERENCES Account(number)  
    ON DELETE CASCADE ON UPDATE CASCADE  
   );
2. Given the following relational model and use cases, write SQL statements that implement the use cases  
   1. Relational Model  
        
      CREATE TABLE Sponsor (  
       username VARCHAR (50) NOT NULL,  
       CONSTRAINT pk\_Sponsor PRIMARY KEY (username)  
      );  
        
      CREATE TABLE Sponsored (  
       username VARCHAR (50) NOT NULL,  
       CONSTRAINT pk\_Sponsored PRIMARY KEY (username)  
      );  
        
      CREATE TABLE Donation (  
       sponsorUsername VARCHAR (50),  
       sponsoredUsername VARCHAR (50),  
       amount double,  
       date Date NULL,  
       frequency VARCHAR (50),  
       CONSTRAINT fk\_Donation\_sponsorUserName FOREIGN KEY (sponsorUsername)  
       REFERENCES Sponsor(username),  
       CONSTRAINT fk\_Donation\_sponsoredUserName FOREIGN KEY (sponsoredUsername)  
       REFERENCES Sponsored(username),  
       CONSTRAINT fk\_Donation\_frequency FOREIGN KEY (frequency)  
       REFERENCES FrequencyType(type)  
      );  
        
      CREATE TABLE FrequencyType (  
       type VARCHAR (50),  
       CONSTRAINT pk\_FrequencyType PRIMARY KEY (type)  
      );  
        
      INSERT INTO FrequencyType (type) VALUES (‘ONE\_TIME’);  
      INSERT INTO FrequencyType (type) VALUES (‘DAILY’);  
      INSERT INTO FrequencyType (type) VALUES (‘WEEKLY’);  
      INSERT INTO FrequencyType (type) VALUES (‘MONTHLY’);  
      INSERT INTO FrequencyType (type) VALUES (‘YEARLY’);  
        
      CREATE TABLE Friend (  
       sponsorUsername VARCHAR (50),  
       sponsoredUsername VARCHAR (50),  
       CONSTRAINT pk\_Friend PRIMARY KEY (sponsorUsername, sponsoredUsername),  
       CONSTRAINT fk\_Friend\_sponsorUsername FOREIGN KEY (sponsorUsername)  
       REFERENCES Sponsor(username),  
       CONSTRAINT fk\_Friend\_sponsoredUsername FOREIGN KEY (sponsoredUsername)  
       REFERENCES Sponsored(username)  
      );
   2. Use Cases
      1. Username “tom” makes a one time donation of $25 to username “alice” today  
           
         # Optionally, insert “tom” and “alice” first.  
         INSERT INTO Donation (sponsorUsername, sponsoredUsername, date, amount, frequency)  
         VALUES (“tom”, “alice”, “10/12/2017”, 25.00, “ONE\_TIME”);
      2. Username “bob” sets up a monthly recurring donation to “edward” of $5 a month  
           
         # Optionally, insert “bob” and “edward” first.  
         INSERT INTO Donation (sponsorUsername, sponsoredUsername, date, amount, frequency)  
         VALUES (“bob”, “edward”, “10/12/2017”, 5.00, “MONTHLY”);
      3. Unfriend usernames “charlie” and “dan”  
           
         DELETE FROM Friend  
         WHERE (sponsorUsername = “charlie” AND sponsoredUsername = “dan”)  
          OR (sponsorUsername = “dan” AND sponsoredUsername = “charlie”);
      4. Increase username’s “bob” recurring donations to “edward” to weekly and to $10  
           
         UPDATE Donation  
         SET amount = 10.00, frequency = “WEEKLY”  
         WHERE sponsorUsername = “bob”   
          AND “sponsordedUsername = “edward” AND date = “10/12/2017”  
          AND amount = 5.00 AND frequency = “MONTHLY”;
      5. Find the most common donation frequency type  
           
         SELECT frequency, COUNT(\*) AS CNT  
         FROM Donation  
         GROUP BY frequency  
         ORDER BY CNT DESC  
         LIMIT 1;
      6. Find the top 10 sponsors with the most friends  
           
         SELECT sponsorUsername, COUNT(\*) AS CNT  
         FROM Friend  
         GROUP BY sponsorUsername  
         ORDER BY CNT DESC  
         LIMIT 10;
      7. Find the top 10 sponsoreds with the most one time donations  
           
         SELECT sponsoredUsername, COUNT(\*) AS CNT  
         FROM Donation  
         WHERE frequency = ‘ONE\_TIME’  
         GROUP BY sponsoredUsername  
         ORDER BY CNT DESC  
         LIMIT 10;
      8. Find all sponsors at the daily frequency who donated more than $1000 total for the current year  
           
         # What about weekly, monthly, yearly calculations?  
         SELECT sponsorUsername, SUM(DONATION\_SUB\_TOTAL) AS DONATION\_TOTAL  
         FROM (  
          # The sum of sponsors daily donations through today.  
          # Note a single sponsorUsername can have multiple daily donations.  
          SELECT sponsorUsername,  
          IF(date < ‘2017-01-01’,  
          TIMESTAMPDIFF(DAY, ‘2017-01-01’, CURRENT\_DATE()),  
          TIMESTAMPDIFF(DAY, date, CURRENT\_DATE())  
          # Similarly:  
          #DATEDIFF(CURRENT\_DATE(), ‘2017-01-01’),  
          #DATEDIFF(CURRENT\_DATE(), date)  
          ) \* amount AS DONATION\_SUB\_TOTAL  
          FROM Donation  
          WHERE frequency = ‘DAILY’) AS DAILY\_SUB\_TOTAL  
         GROUP BY sponsorUsername  
         HAVING SUM(DONATION\_SUB\_TOTAL) > 1000  
         ORDER BY DONATION\_TOTAL DESC;  
           
         # Alternatively:  
         SELECT sponsorUsername,  
          SUM(IF(date < ‘2017-01-01’,  
          TIMESTAMPDIFF(DAY, ‘2017-01-01’, CURRENT\_DATE()),  
          TIMESTAMPDIFF(DAY, date, CURRENT\_DATE())  
          ) \* amount) AS DONATION\_TOTAL  
         FROM Donation  
         WHERE frequency = ‘DAILY’  
         GROUP BY sponsorUserName  
         HAVING DONATION\_TOTAL > 1000  
         ORDER BY DONATION\_TOTAL DESC;