COMPSCI 2DB3 Assignment 3

Elite Lu

# Part 1

For this first part, I will be using the SQL to create relations and using these relations, I will proceed to create the SQL code.

**Member**(mid: int, email: text, hash: BINARY(64), name: text, address: text)

Email must be unique. Because of this, I will be using VARCHAR instead of CLOB. In addition to this, salt was not incorporated in the diagram but was mentioned to be associated with the passhash. I ended up storing the passhash as CHAR(64) as BINARY is not supported.

CREATE TABLE Member

(

mid INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

email VARCHAR(100) NOT NULL UNIQUE,

passhash CHAR(64) FOR BIT DATA NOT NULL,

name VARCHAR(100) NOT NULL,

address CLOB NOT NULL

);

With the member completed, I will focus on the different categories of Member, which are Personal and Organization along with the relationship isRep.

**Personal** (mid: int)

Every mid is from Member.

CREATE TABLE Personal

(

mid INT NOT NULL PRIMARY KEY,

FOREIGN KEY(mid) REFERENCES Member

);

**Organization** (mid: int, type: text)

Every mid is from Member.

CREATE TABLE Organization

(

mid INT NOT NULL PRIMARY KEY,

type VARCHAR(100) NOT NULL,

FOREIGN KEY(mid) REFERENCES Member

);

For the relationship IsRep, it is between an Organization and Personal entities. However, one constraint on the organization is that every organization must have a Personal entity related to it. Because of this, I will not make another table to represent IsRep but modify the original Organization table which also has a Personal.mid stored as one of the variables.

**Organization** (mid: int, type: text, midPersonal: int)

Every mid is from Member.

CREATE TABLE Organization

(

mid INT NOT NULL,

type VARCHAR(100) NOT NULL,

midPersonal INT NOT NULL,

FOREIGN KEY(mid) REFERENCES Member,

FOREIGN KEY(midPersonal) REFERENCES Personal.mid,

PRIMARY KEY(mid, midPersonal),

);

**Subscription** (mid: int, startyear: int, startmonth: int)

The mid is referenced from member.mid. startyear and startmonth will be integers instead of dates to avoid confusion. I will perform a check for the month to ensure it is between 1 and 12 inclusive along with the year being less than or equal to 2023 (this year).

CREATE TABLE Subscription(

mid INT NOT NULL,

startyear INT NOT NULL,

startmonth INT NOT NULL,

FOREIGN KEY(mid) REFERENCES Member,

PRIMARY KEY(mid, startyear, startmonth),

CHECK(startmonth >= 1 AND startmonth <= 12 AND startyear <= 2023)

)

Like Member, there are two ISA relations with PrePaid and LongTerm. For both, I will proceed to implement the ER method since the NULL method would be unable to properly implement PayFor. The OO method will not be used because like the NULL method, they both cannot express PayFor.

**PrePaid** (mid: int, startyear: int, startmonth: int, charges: int)

Self-explanatory, where mid, startyear, and startmonth are all obtained from Subscription.

CREATE TABLE PrePaid(

mid INT NOT NULL,

startyear INT NOT NULL,

startmonth INT NOT NULL,

charges INT NOT NULL,

FOREIGN KEY(mid, startyear, startmonth) REFERENCES Subscription,

PRIMARY KEY(mid, startyear, startmonth)

)

**LongTerm** (mid: int, startyear: int, startmonth: int, tier: int)

Self-explanatory, where mid, startyear, and startmonth are all obtained from Subscription.

CREATE TABLE LongTerm(

mid INT NOT NULL,

startyear INT NOT NULL,

startmonth INT NOT NULL,

tier INT NOT NULL,

FOREIGN KEY(mid, startyear, startmonth) REFERENCES Subscription,

PRIMARY KEY(mid, startyear, startmonth)

)

Before considering PayBy, I will first consider the weak entity Payment.

**Payment** (mid: int, startyear: int, startmonth: int, term: int)

All but the attribute term are referenced from LongTerm entities. In addition to this, terms can only be positive numbers.

CREATE TABLE Payment(

mid INT NOT NULL,

startyear INT NOT NULL,

startmonth INT NOT NULL,

term INT NOT NULL,

FOREIGN KEY(mid, startyear, startmonth) REFERENCES LongTerm,

PRIMARY KEY(mid, startyear, startmonth, term),

CHECK(term >= 0)

)

The last relationship would be PayBy, which is between Personal and Payment. There is no need for another table. Rather, I will go back to the table for Payment and modify it.

CREATE TABLE Payment(

mid INT NOT NULL,

startyear INT NOT NULL,

startmonth INT NOT NULL,

term INT NOT NULL,

midPersonal INT NOT NULL REFERENCES Personal,

FOREIGN KEY(mid, startyear, startmonth) REFERENCES LongTerm,

PRIMARY KEY(mid, startyear, startmonth, term),

CHECK(term >= 0)

)

# Part 2

For the donation system, here are each of the corresponding tables and the appropriate SQL code for each:

* **DonorEvent**(eid, name, date, description, private)

**Solution:** Translating this schema into the appropriate SQL will be done with the simplest method, which is the use of automatically generated identifiers. The description can be left as null since it is not the most important piece of information whereas every other field is necessary:

CREATE TABLEDonorEvent

(

eid INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

name VARCHAR(100) NOT NULL,

date DATE NOT NULL,

description CLOB NULL,

private BOOLEAN NOT NULL

);

* **DonorInvite**(eid, mid, confirmed, amount, referredByoptional)

**Solution:** For this solution, eid is from the previous table and mid is from the entity Member. For confirmed, this will just be a boolean. The two additional constraints are that for the same event, a person cannot refer more than the amount field. In addition to this, a direct invite person can invite people and those people that the direct invited person will have their referredBy field as the member’s mid. However, that invited person cannot refer another person. This is all assuming the conditions that a member can only be registered to an event once. However, translating this to purely SQL without the conditions, it is as follows:

CREATE TABLEDonorInvite

(

eid INT NOT NULL REFERENCES DonorEvent,

mid INT NOT NULL REFERENCES Member,

confirmed BOOLEAN NOT NULL,

amount INT NOT NULL,

referredBy INT NULL,

PRIMARY KEY(eid, mid)

);

If given the ability to do so, I would also add a multi table assertion.

* **Referrals**(eid, mid, memberName, rid, referredName)

**Solution:** For this solution, all the fields are from the DonorInvite table, which makes this easy. The answer is shown below:

CREATE VIEW Referrals AS

(

SELECT d.eid, d.mid, m.name, d.referredBy AS rid, m2.name AS referredName

FROM DonorInvite d, MEMBER m, MEMBER m2

WHERE referredBy IS NOT NULL AND m.mid = d.mid AND m2.mid = d.referredBy

);

* **Donation**(eid, mid, amount, attributed­optional, messageoptional)

**Solution:** For this solution, I will reference eid from a DonorEvent entity and an mid from a Member entity. In addition to this, I checked if the mid is not equal to the attributed because it is ridiculous for a person to donate to themselves. The following is shown below:

CREATE TABLE Donation

(

eid INT NOT NULL REFERENCES DonorEvent,

mid INT NOT NULL,

amount INT NOT NULL,

attributed INT NULL,

FOREIGN KEY (mid) REFERENCES MEMBER,

FOREIGN KEY (attributed) REFERENCES MEMBER(mid),

PRIMARY KEY(eid, mid),

CHECK (mid <> attributed)

);