Assignment: The relational data model and SQL

Part One: The subscription service
Rational:

In the rational, we use the following text annotations:

an entity	The highlighted piece of text resembled an entity-like object.
an attribute	The highlighted piece of text resembled an attribute of some entity-like object.
a relationship	The highlighted piece of text resembled a relationship between entity-like objects.
a constraint	The highlighted piece of text resembled a constraint on data.
~~not relevant~~	The highlighted piece of text is not relevant (e.g., context, application details, ...)

~~The proposed subscription model supports two disjoint types of membership accounts:~~

1. a **personal account** held **by a single private person** (e.g., a community member interested in art); or
2. an **organization account** (e.g., held by a local company or not-for-profit organization).

Organization accounts are managed by **one-or-more representatives** (with **personal accounts**) that can manage the organization account (e.g., borrow art pieces in the name of the organization). Furthermore, **organization accounts** have a ***type*** (e.g., company, local government, not-for-profit, etc.). **Members** can log in to their account via their **unique** ***e-mail*** **address** and a password. ~~According to the consultant, the password should be stored as a~~ ***hashed*** **value that incorporates a salt value**.

Each **account** can hold **a single subscription at any single time**. That **subscription** can be a **pre-paid subscription** that allows the account to borrow ***a fixed number of art pieces*** or a **long-term subscription**. **Long-term subscriptions** belong to a ***tier*** that determines the subscription cost and how many art pieces can be borrowed simultaneously. ~~The consultant noted that the diagram does not enforce that an account can only hold a single long-term subscription at any single time.~~

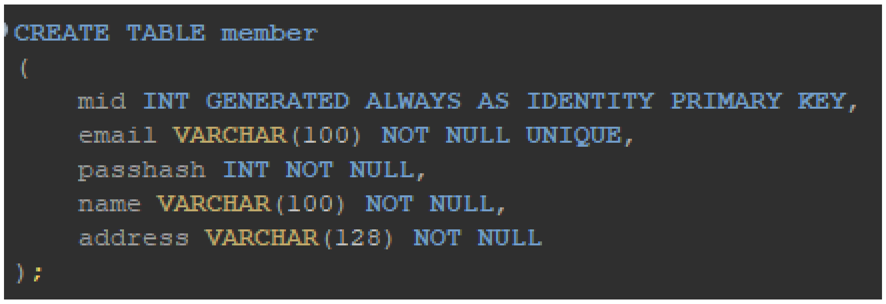
For **long-term subscriptions**, a **payment history covering each payment installment** is maintained via the ***payment terms*** that have been paid ~~(term one representing the first month of the subscription, term two the second month, and so on)~~. For each payment term, the **paying account is also stored in case of a organization account** ~~(to keep track which representative paid). In addition, the system maintains a history of all subscriptions held by an account. The ER-Diagram for the subscription features can be found in Figure 1.~~

Solution:

We start with the **Members** entity as it is not a weak entity and does not partake in any one-to-many relationships.

Members:

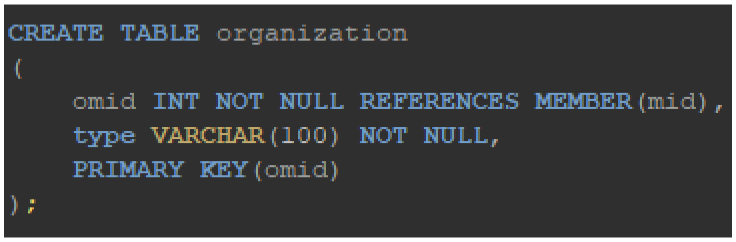
- It has an ISA hierarchy for Personal and Organization Accounts which are connected through a relationship “IsRep”, so we need to use ER method to separate the entities out.
- The email attribute must be unique, so we use VARCHAR which is UNIQUE
- The password must be a hash value that incorporates a salt value, so we use INT
- The mid is the primary key which should be generated automatically when the account is created
- The name and address are represented by VARCHAR and can not be NULL



```
CREATE TABLE member
(
    mid INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    email VARCHAR(100) NOT NULL UNIQUE,
    passhash INT NOT NULL,
    name VARCHAR(100) NOT NULL,
    address VARCHAR(128) NOT NULL
);
```

Organization:

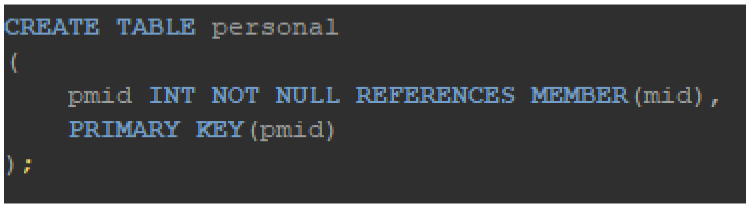
- It is an ISA hierarchy of Members so it acquires all the attributes and keys from it
    - However, I changed the ‘mid’ to ‘omid’ so it is easy to understand the ‘mid’ is from the organization account
    - ‘omid’ stays as the primary key
- The only attribute it has is ‘type’ which can be represented by VARCHAR



```
CREATE TABLE organization
(
    omid INT NOT NULL REFERENCES MEMBER(mid),
    type VARCHAR(100) NOT NULL,
    PRIMARY KEY(omid)
);
```

Personal

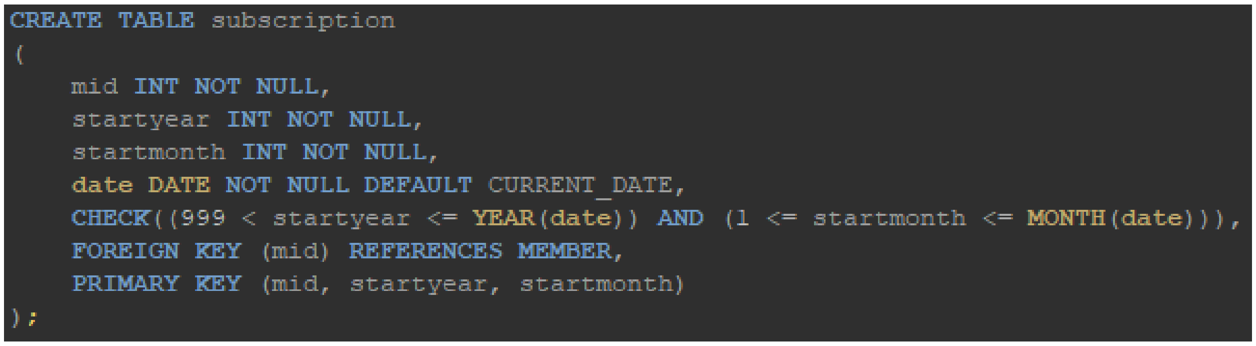
- It is an ISA hierarchy of Members so it acquires all the attributes and keys from it
    - However, I changed the ‘mid’ to ‘pmid’ so it is easy to understand the ‘mid’ is from the personal account
    - ‘pmid’ stays as the primary key
- There are no attributes specific to this entity



```
CREATE TABLE personal
(
    pmid INT NOT NULL REFERENCES MEMBER(mid),
    PRIMARY KEY(pmid)
);
```

Subscription:

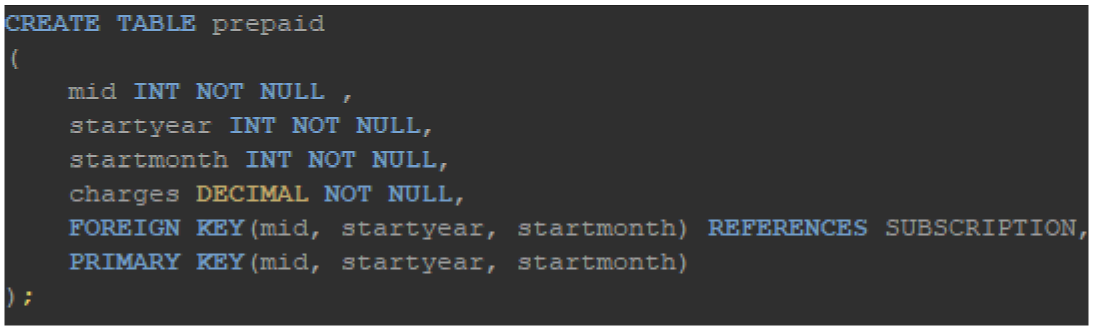
- This is a weak entity of Member with a one-to-one relationship
- It has an ISA hierarchy for PrePaid and LongTerm and LongTerm has an exclusive relationship with the Payment entity, so we use the ER method to separate the entities out
- ‘startyear’ and ‘startmonth’ are the partial keys and ‘mid’ is the foreign key referencing from the Member entity
    - This allows for the constraint where each account can hold a single subscription at any single time with the
- The primary key would be the pair (<u>mid</u>, *startyear*, *startmonth*)
- Also, I added the extra constraints:
    - For ‘startyear’, it should be greater than 999 and less than or equal to current year. So it can be 4 digits and it can not be more than the current year
    - For ‘startmonth’, it should be greater than or equal to 1 and less than or equal to 12. So it can accurately represent the months in integers



```
CREATE TABLE subscription
(
    mid INT NOT NULL,
    startyear INT NOT NULL,
    startmonth INT NOT NULL,
    date DATE NOT NULL DEFAULT CURRENT_DATE,
    CHECK((999 < startyear <= YEAR(date)) AND (1 <= startmonth <= MONTH(date))),
    FOREIGN KEY (mid) REFERENCES MEMBER,
    PRIMARY KEY (mid, startyear, startmonth)
);
```

PrePaid:

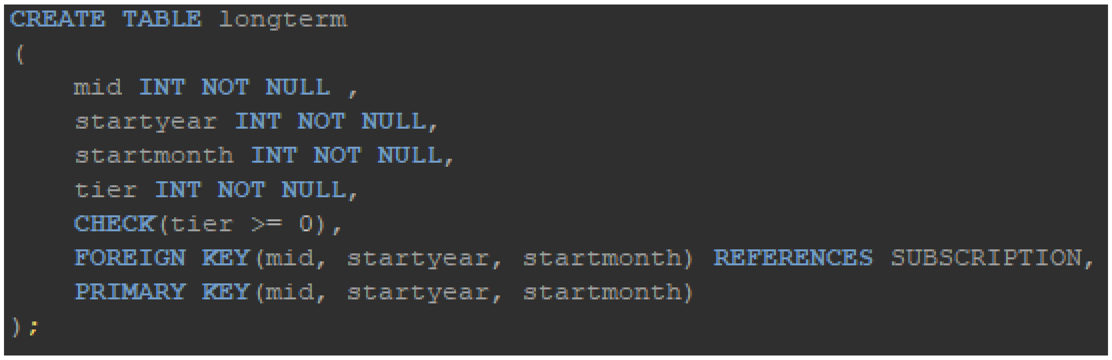
- It is an ISA hierarchy of Subscription so it acquires all the attributes and keys
    - The foreign key from the Subscription becomes the primary key
- The only attribute this entity has is 'charges' which should be defined by DECIMAL as it is a cost (can be a decimal value).
- The 'charges' can go positive and negative



```
CREATE TABLE prepaid
(
    mid INT NOT NULL ,
    startyear INT NOT NULL,
    startmonth INT NOT NULL,
    charges DECIMAL NOT NULL,
    FOREIGN KEY(mid, startyear, startmonth) REFERENCES SUBSCRIPTION,
    PRIMARY KEY(mid, startyear, startmonth)
);
```

LongTerm:

- It is an ISA hierarchy of Subscription so it acquires all the attributes and keys
    - The foreign key from the Subscription becomes the primary key
- This entity's only attribute is 'tier' which determines the subscription cost and how many art pieces can be borrowed simultaneously. Thus, it is a level of structure which must be positive.
- Also, the constraint mentioned by the consultant that an account can only hold a single long-term subscription at any single time, is already enforced by Subscription constraints.

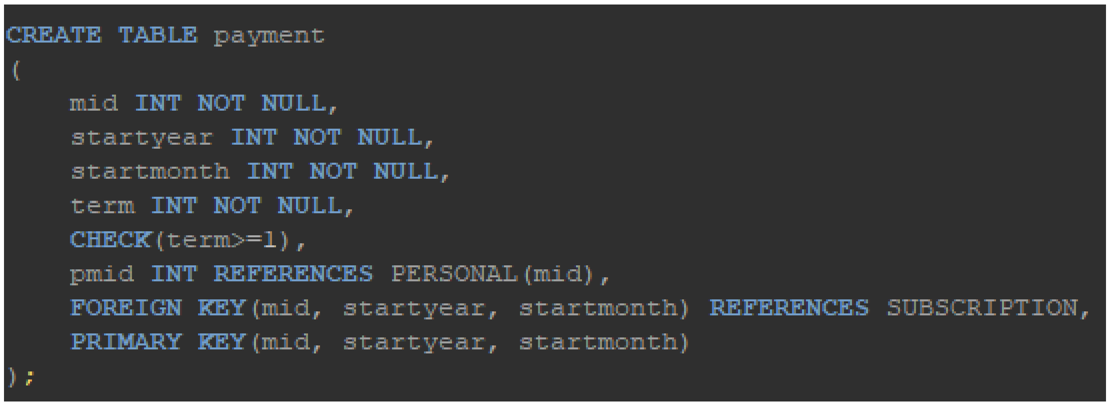


```
CREATE TABLE longterm
(
    mid INT NOT NULL ,
    startyear INT NOT NULL,
    startmonth INT NOT NULL,
    tier INT NOT NULL,
    CHECK(tier >= 0),
    FOREIGN KEY(mid, startyear, startmonth) REFERENCES SUBSCRIPTION,
    PRIMARY KEY(mid, startyear, startmonth)
);
```

Payment:

- This is a weak entity of LongTerm with a one-to-one relationship so it inherits all the attributes and keys from LongTerm
    - The foreign key from the Subscription becomes the primary key
- The payment terms are recorded for each payment which can be defined by INT and there must be at least 1 term or more to be considered a payment

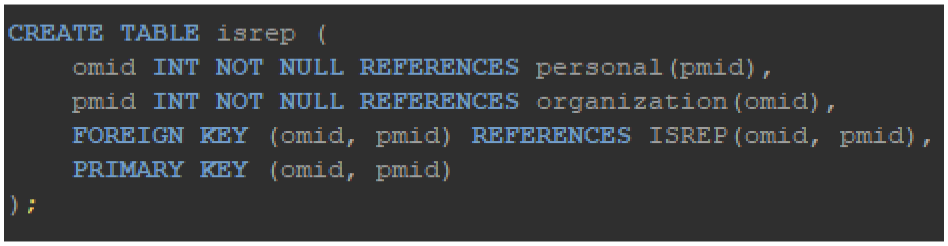
- For an organization account, one of the personal accounts ‘mid’ is stored, but if it is a personal account then it can be NULL



```
CREATE TABLE payment
(
    mid INT NOT NULL,
    startyear INT NOT NULL,
    startmonth INT NOT NULL,
    term INT NOT NULL,
    CHECK(term>=1),
    pmid INT REFERENCES PERSONAL(mid),
    FOREIGN KEY(mid, startyear, startmonth) REFERENCES SUBSCRIPTION,
    PRIMARY KEY(mid, startyear, startmonth)
);
```

IsRep Relationship:

- It has a many-to-one total relationship as the Organization account must have at least one Personal account.
    - This constraint can be filled by making the foreign key a pair of (omid, pmid) which can be referenced by ISREP(omid, pmid). This way omid can have one or more pmid in it.
    - We make the foreign key as the primary key

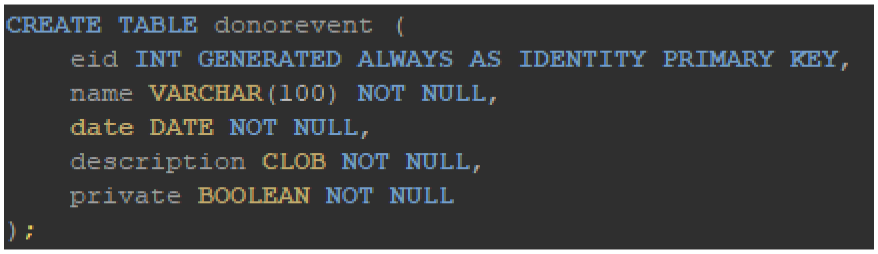


```
CREATE TABLE isrep (
    omid INT NOT NULL REFERENCES personal(pmid),
    pmid INT NOT NULL REFERENCES organization(omid),
    FOREIGN KEY (omid, pmid) REFERENCES ISREP(omid, pmid),
    PRIMARY KEY (omid, pmid)
);
```

Part Two: The Donation System

DonorEvent:

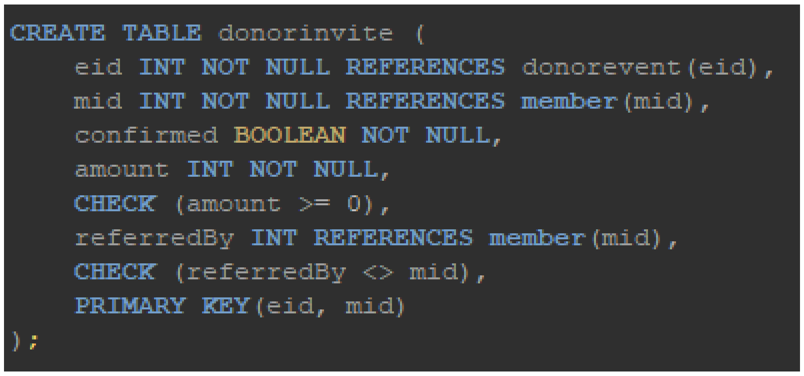
- **‘**eid’ is the primary key which should be generated automatically when the event is created
- ‘name’ can be represented by VARCHAR
- ‘date’ can be represented by DATE
- ‘description’ can be represented by CLOB, as it could be a long description with lots of characters
- ‘private’ can be BOOLEAN which can state if the event is private or not



```
CREATE TABLE donorevent (
    eid INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    date DATE NOT NULL,
    description CLOB NOT NULL,
    private BOOLEAN NOT NULL
);
```

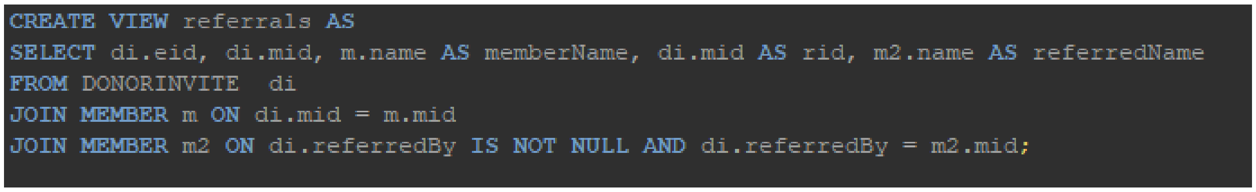
DonorInvite:

- **‘**eid’ is referenced from the DonorEvent
- ‘mid’ is referenced from the Member
- ‘confirmed’ can be represented by BOOLEAN which state if the member will be attending the event or not
- ‘amount’ can be represented by INT and it must be positive as it represents the number of people
- ‘referredBy’ is also referenced from the Member as the direct invite members can only invite other members that are in the Member table
  - I put a constraint that ‘referredBy’ can not be the same as the ‘mid’ or in other words, the direct invite member can not invite themself.
- The constraint where the direct invite member can not exceed the amount of referrals given is not possible as we would need to use the SELECT statement to count the number of invites inside the CHECK.
- The constraint where the invited member can not refer other members is also not possible for the same reason stated above
- The primary key would be the pair of ‘eid’ and ‘mid’ as there can be many events and many members for those events



```
CREATE TABLE donorinvite (
    eid INT NOT NULL REFERENCES donorevent(eid),
    mid INT NOT NULL REFERENCES member(mid),
    confirmed BOOLEAN NOT NULL,
    amount INT NOT NULL,
    CHECK (amount >= 0),
    referredBy INT REFERENCES member(mid),
    CHECK (referredBy <> mid),
    PRIMARY KEY(eid, mid)
);
```

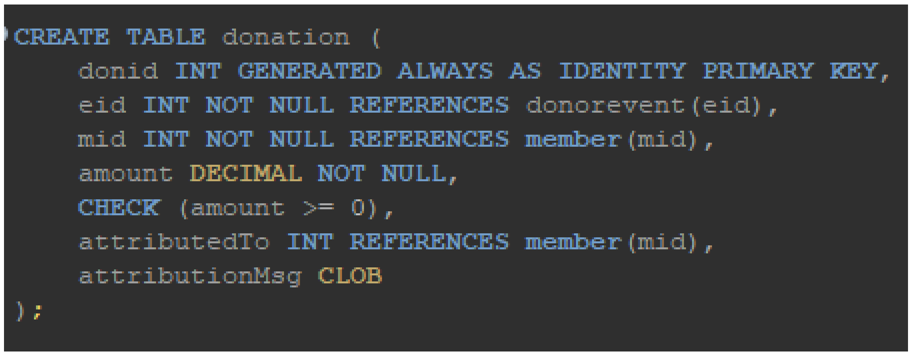
Referrals:



```
CREATE VIEW referrals AS
SELECT di.eid, di.mid, m.name AS memberName, di.mid AS rid, m2.name AS referredName
FROM DONORINVITE  di
JOIN MEMBER m ON di.mid = m.mid
JOIN MEMBER m2 ON di.referredBy IS NOT NULL AND di.referredBy = m2.mid;
```

Donation:

- ‘donid’ is the primary key which should be generated automatically for each donation item
- **‘**eid’ is referenced from the DonorEvent
- ‘mid’ is referenced from the Member
- ‘amount’ can be represented by DECIMAL and it must be positive as it represents a cost donated
- ‘attributedTo’ is referenced from the Member assuming that you can only receive a donation if you are in the DonorEvent
- ‘attributionMsg’ can be represented by CLOB, as it could be a long message with lots of characters
- The constraint where people can make several donations during an event can be resolved through the ‘donid’ as it can uniquely identify each donation item even if the event and member donating it is the same.



```
CREATE TABLE donation (
    donid INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    eid INT NOT NULL REFERENCES donorevent(eid),
    mid INT NOT NULL REFERENCES member(mid),
    amount DECIMAL NOT NULL,
    CHECK (amount >= 0),
    attributedTo INT REFERENCES member(mid),
    attributionMsg CLOB
);
```