Designing a Database for Hospital Activities Management

Sang Truong

April 30, 2020

1 Database design decision

The database in figure 1 for a hospital to manage its activities. Entities in this system include `Patient`, `Doctor`, `Room`, and `Lab`. Every time a patient visit the hospital, he/she will be assigned a new `patientID` since his or her information might change since the last time he or she visited. Indeed, the person might have a new disease (`patientDisease`), a new insurance condition (`patientInsurance`), or a new phone number (`patientPhone`). While visiting the hospital, the patient might need to see one or a few `Doctor`(s) or do one or a few diagnostics in `Lab`. Each `Doctor` might belong to more than one `Lab` and each `Lab` might have multiple `Doctor`. A doctor can either meet with patients or works in lab(s), or both. In addition, the patient might need to stay in the hospital overnight. In that case, the patient will be assigned to a `Room`. For each service listed above, the patient will receive a `charge` from the `ServiceMenu` table. Every patient can have many services and every service and serve many patients. Some services in the hospital might be free.

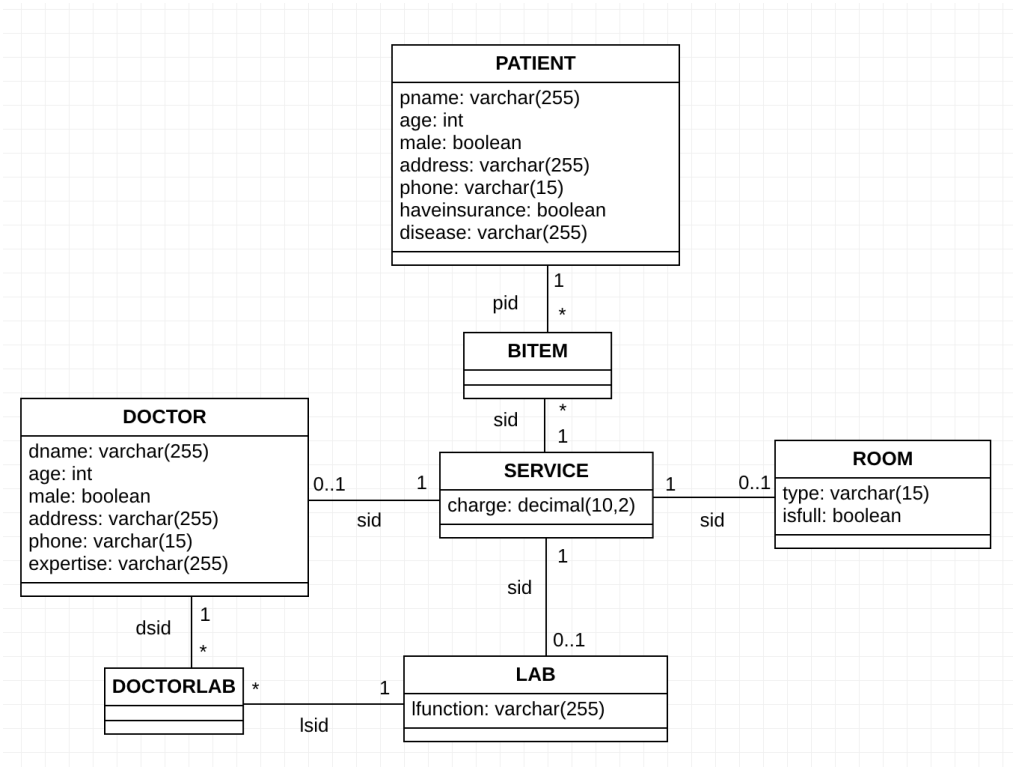

Figure 1: Class diagram for database designed for hospital activities management.

The synthesized primary key of each table is not shown in the diagram. The foreign key that connects 2 tables are shown on each line that connects 2 classes. By convention, the synthesized foreign key name is the same as the referenced table. In the cases of `Doctor-SeriveMenu, Room-SeriveMenu, Lab-ServiceMenu`, the foreign key is the primary key of each table, namely `doctorID, roomNum, labID`, respectively. Using primary key as the foreign key in these tables ensures the relationship is 0-or-1-to-1. Each entry `Doctor, Room, Lab` table gives extra information for an entry in `ServiceMenu` table. In the future, if the hospital adds more type of service, such as transportation, another table, say `Transportation`, just needs to be added to the database with a 0-or-1-to-1 relation with `ServiceMenu`. Thus, this design can be generalized quite well.

The `BillableItem` table is a reification of the many-to-many `Patient-Service` relationship, recognizing that each patient can have many services, and each service can serve many patients. Similarly, the many-to-many `Doctor-Lab` relationship is reified by `DoctorLab` table.

There are 2 paths to go from `DoctorLab` to `ServiceMenu`. Going through `Lab`, an entry ϵ_L in `DoctorLab` tells us which lab the doctor in ϵ_L belong to and charge associating with that doctor as a lab technician. Entry ϵ_L only tells us `doctorID` of the doctor, which will lead us to doctor information if we go from entry ϵ_L to the corresponding entry in `Doctor` table. Going from `DoctorLab` to `ServiceMenu` through `Doctor`, an entry ϵ_D in `DoctorLab` will tell us the charge associating with the doctor as he/she meets with patients. Thus, although there are 2 paths from `DoctorLab` to `ServiceMenu`, there is no redundancy since the 2 paths tell us different information. Other than that, there is no more loop connecting 2 tables in the designed database. To the best of my knowledge, I do not see redundancy (as defined in the text book) in this design.

There are a few constraints in this design. `patientName, patientAddress, patientDisease, doctorName, doctorAddress, doctorExpertise, labFunction` fields are each set at 255 characters, while patientPhone, doctorPhone, roomType fields are each set to 15 characters. These numbers are arbitrary. In addition, doctorAge, patientAge, charge are non-negative.

A database dictionary, which can help us interpret information from the table, is necessary. For example, a `True` in `patientGender` is associated with male.

2 Sample queries

- Given a patientID, identify service(s) he/she uses during that visit.
- Given a serviceID, identify what service(s) associating with that serviceID.
- Given a doctorID, identify patient(s) who visit this doctor.
- Given a labID, identify doctor(s) who work(s) in that lab.

3 Sample restricted access for the user

One possible view is to show the patient a table of all services that he/she used during his/her visit, together with the service details (such as which doctor, which type of room, and which type of test) and the fees of each service.

4 SQL schema

```
create table PATIENT (
  pid int not null,
  pname varchar(255) not null,
  age int not null,
  male boolean not null,
  address varchar(255) not null,
  phone varchar(15) not null,
  haveinsurance boolean not null,
  disease varchar(255) not null,
  primary key (pid),
  check (age >=0)
);

create table SERVICE (
  sid int not null,
  charge decimal(10,2) not null,
  primary key (sid),
  check (charge >= 0)
);

create table DOCTOR (
  did int not null,
  dname varchar(255) not null,
  age int not null,
  male boolean not null,
  address varchar(255) not null,
  phone varchar(15) not null,
  expertise varchar(255) not null,
  sid int not null,
  primary key (sid),
  check (age >= 0),
  foreign key (sid) references SERVICE
    on delete no action
);

create table ROOM (
  rid int not null,
  type varchar(15) not null,
  isfull boolean not null,
  sid int not null,
  primary key (sid),
```

```
  foreign key (sid) references SERVICE
    on delete no action
);

create table LAB (
  lid int not null,
  lfunction varchar(255) not null,
  sid int not null,
  primary key (sid),
  foreign key (sid) references SERVICE
    on delete no action
);

create table BITEM (
  pid int not null,
  sid int not null,
  foreign key (pid) references PATIENT
    on delete no action,
  foreign key (sid) references SERVICE
    on delete no action
);

create table DOCTORLAB (
  dsid int not null,
  lsid int not null,
  foreign key (dsid) references DOCTOR
    on delete no action,
  foreign key (lsid) references LAB
    on delete no action
);
```