Observations of Daily Living - Health Portal

Database Management Systems Project

Developed By

Akshay Naval

(asnaval)

Sonia Ghanekar

(ssghanek)

Table Of Content.

1) Problem Statement

Write a program to create a health portal which allows patients and health professionals to monitor and manage crucial information about the health states of patients with chronic diseases.

1.1 Why use Database

Here are some of the factors that makes usage of database beneficial

- Reduced data redundancy
- Data Independence
- Efficient Data Access
- Reduced updating errors and increased consistency
- Greater data integrity and independence from application programs
- Improved data security
- Ability to access data concurrently
- Recovery from failure
- Helps us storing data in organized manner
- Reduced Application development time

In the given scenario we need to store a lot of information about patient, classes, categories, types, questions, observations etc. For storing this info in a systematic manner, DBMS was the best suitable possible option.

1.2 Application Requirements

Our application allows users to register them as patients. Once registered they can login into the system and record observations. There are health professionals who can also login into the system and can perform a specific set of operations.

Patients can have multiple classes from COPD, HIV, Obesity, High Risk Pregnancy. Each class have some specific observation types that needs to be captured. Every patient after logging in can fill the appropriate observations. Patients can also view alerts. Alerts are generated if your observation surpasses the threshold. Patients can also add other patients with same illness as friends. If your friends are not viewing there alerts you will be notified for that by an alert. Patients can also add a new Observation Type.

Health Professional can login and see aggregated reports. They can also change patient information. They are capable of defining which observation type should be asked for which class of patients.

1.3 Assumptions

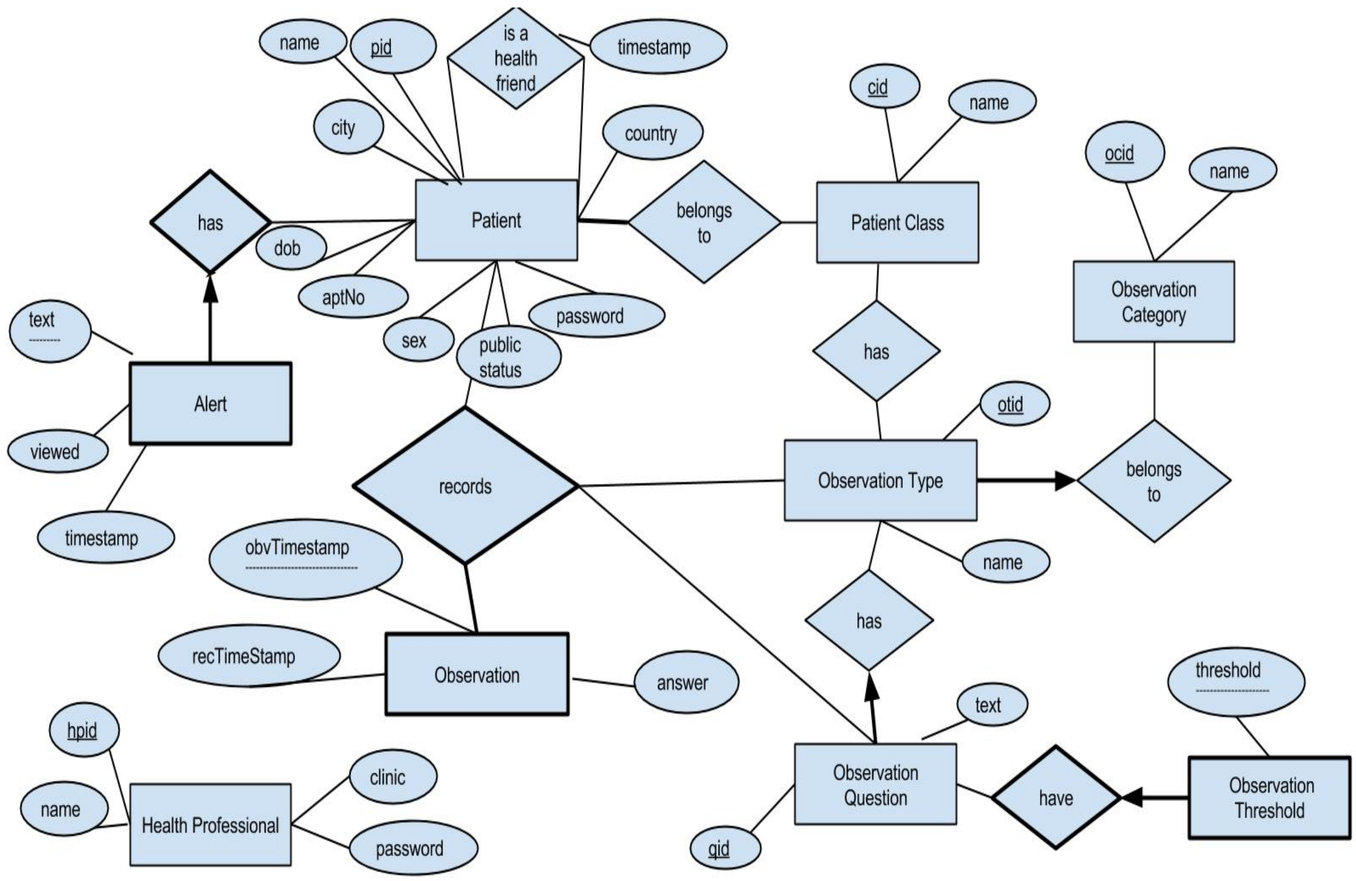
We have made the following assumptions while designing our application:

- Only one user can be logged in at a time
- When a patient is registered into the system, he is put in “General” class of patient until a health professional adds other classes to it.
- All the inputs given by the user are correct. We are not checking for parsing errors
- For us one observation is for one question in a type. For eg. Observation blood pressure have 2 questions (Systolic, diastolic). Each of them will be added as separate observation in the system.
- We are following the twitter model for friendship. Friendship is one way eg. If patient 1 adds patient 2 as his friend, it does not mean that patient 2 has patient 1 as his friend.
- If the threshold value for an observation type is negative, it means that the given value is a the minimum value and if it is positive it means it is the maximum possible above which alerts will be generated.

2) Entities and Relationships

2.1 E-R Diagram

E-R Diagram based on the following application requirements is as follows:



3) Relational Schema

We have derived the relational schema based on the above ER diagram as follows:

PATIENT (pid: integer, name: String, aptNo: integer, city: string, country: string, sex string, publicStatus: string, password: string)

PATIENTCLASS (cid: integer, name: string)

PATIENTCLASSRELATIONSHIP (pid: integer, cid: integer)

OBSERVATIONCATEGORY (ocid: integer, name: string)

OBSERVATIONTYPE (ocid: integer, name: string, otid: integer)

OBSERVATIONQUESTION (qid: integer, text: string, otid: integer)

OBSERVATION (pid: integer, otid: integer, qid: integer, obvTimestamp timestamp, recTimestamp: timestamp, answer: string)

PATIENTCLASSOBVTYPEMAPPER (cid: integer, otid: integer)

ALERT (pid: integer, text: string, viewed boolean, timestamp timestamp)

OBSERVATIONTHRESHOLD (qid: integer, threshold: number)

HEALTHFRIEND (pid: integer, fid: integer, timestamp: timestamp)

HEALTHPROFESSIONAL (hpid: integer, name: string, clinic: string, password: string)

4) Users

There are two types of users accessing our system.
1) Patients
2) Health Professional

The view of the database changes according to the user who has logged into the system.
eg: When a new user registers himself as a patient, he is shown a different set of options than those which are given to a health professional. Similarly a patient does not have the permissions of updating the observation categories as well as assigning patients to different classes. Thus, a health professional is basically an admin for the system and the patient is the end-user.

5) Functional dependencies and normal forms

The functional dependencies are as follows:

1) Patient
   pid -> dob, name, aptNo, city, ctry, sex, ps, pwd

2) PatientClass
   cid -> cname

3) PatientClassRelationship
   No functional dependency

4) ObservationCategory
   ocid -> ocname

5) ObservationType
   otid -> otname, ocid

6) ObservationQuestion
   qid -> text, otid

7) Observation
   pid, otid, qid, obsTimestamp -> recTimestamp, answer

8) PatientClassObvTypeMapper
No functional dependency

9) Alert
pid, text -> viewed, alertTimestamp

10) ObservationThreshold
qid -> threshold

11) HealthFriend
pid, fid -> friendshipTimestamp

12) HealthProfessional
hpid -> hpname, clinic, hppassword

All the above relations in the **BCNF normal form**.

6) Use case scenarios

We are dealing with two types of users in our application namely patients and health professionals.

Some of the use cases for Patients are:

1) A user can register himself as a patient.
2) The patient can login into the system and record his observations according to his observation type.
3) A patient can view his records for a specific time period.
4) A patient can view his pending alerts.
5) A patient can make health friends.
6) A patient can clear the read alerts

Some of the use cases for a health professional are:

1) Health professional can login into the system and enter a new observation type.
2) Health professional can change the category of an observation type.
3) Health professional can add a new association between observation type and patient class.
4) Health professional can change the patient class.

7) SQL queries

7.1 Retrieval SQL queries - used to find specific information

1) Find patients with the lowest weight amongst HIV patients.

```
SELECT pid FROM Patient p, Observation o, PatientClassRelationship pc
WHERE p.pid=o.pid AND
o.to_number(answer) = (SELECT MIN(to_number(answer)) FROM observation WHERE otid =
(SELECT otid
        FROM ObservationType
        WHERE name = 'Weight')) AND
p.pid = pc.pid AND pc.cid = (SELECT cid FROM PatientClass WHERE name = 'HIV');
```

2) Of all Obesity and High Risk Patients, find patients with the highest blood pressure.

```
SELECT pid FROM patient p, Observation o, PatientClassRelationship pc
WHERE p.pid=o.pid AND
o.to_number(answer) = (SELECT MAX(to_number(answer))
                        FROM Observation
                        WHERE otid = (SELECT otid
                                        FROM ObservationType
                                        WHERE name = 'Blood Pressure'))
AND
p.pid = pc.pid AND
pc.cid = (SELECT cid
        FROM PatientClass
        WHERE name in ('High Risk Pregnancy','Obesity'));
```

3) Find patients who have HealthFriends with no outstanding alerts.

```
SELECT DISTINCT pid
FROM patient
WHERE pid not in
(SELECT pid HealthFriend
WHERE fid IN
(SELECT pid FROM alert WHERE viewed = ‘0’));
```

4) Find patients who live in same city as healthfriend.

```
SELECT p1.pid FROM Patient p1, HealthFriend h, Patient p2
WHERE p1.pid = h.pid AND
p2.pid = h.fid AND
p1.city = p2.city AND
p1.pid <> p2.pid
```

5) For Patient X, list their healthfriends, ordered by date in which friendships were initiated.

(Here the patient id will be given by the user.)

```
SELECT fid FROM HealthFriend WHERE pid = ? ORDER BY timestamp ASC
```

7.2 Reporting queries - used to find more general information

1) For each patient, find the number of healthfriends made in the last month.

```
SELECT pid, COUNT(fid)
FROM HealthFriend
WHERE timestamp >= sysdate - interval '1' month
GROUP BY pid
```

2) For each patient and each type of observation, show the number of such observations recorded by the patients.

```
SELECT p.pid,ot.otid,COUNT(CASE WHEN p.pid=o.pid AND ot.otid=o.otid THEN 1 END)
"Observation Count"
FROM patient p, ObservationType ot, Observation o
group by p.pid,ot.otid;
```

3) For each patient, and each of their healthfriends, list the number of lingering alerts of the healthfriend.

```
SELECT p.pid,f.fid,COUNT(case when p.pid=f.pid AND f.fid=a.pid AND a.viewed = '0' then 1
end) "Lingering Alerts"
FROM patient p, healthfriend f, alert a
GROUP BY p.pid, f.fid
HAVING p.pid = f.pid
ORDER BY p.pid
```

8) Limitations

1. The system has no GUI. It is menu driven.
2. New observation type do not have threshold values.