**PROJECT REPORT – CSC540 ODL Project.**

**Team Members:**

Naval Kumar Gupta

Unity Id: nkgupta2

Student Id: 200022910

Gargi Baruwa

Unity Id: gbaruwa

Student Id: 200022738

Contents

[0. Assumptions Taken in the Projects: 2](#_Toc371184876)

[1. Problem Statement: 2](#_Toc371184877)

[2. Entities and Relationships 3](#_Toc371184878)

[3. Users 5](#_Toc371184879)

[4. E-R Diagrams 5](#_Toc371184880)

[5. Relational Schema 7](#_Toc371184881)

[6. Functional Dependencies and Normal Forms: 8](#_Toc371184882)

[7. APIs and SQL Queries: 8](#_Toc371184883)

[8. Use-case Scenario: 13](#_Toc371184884)

[9. Extra Features Added: 15](#_Toc371184885)

# Assumptions Taken in the Projects:

1. When a patient A submits a health friend request to another patient B, not only does the patient A become the health friend of patient B, but patient B also becomes the health friend of patient A.
2. Health friend A will receive friend alerts for his friend B, when the patient B has not viewed his alerts for more than 7 days which is calculated using observation date and not recording date.
3. Out of bound menu options are not entered in the menu program.
4. All patients will be assigned “General” disease class when he enrols. When Physician assigns a disease to a patient, he will belong to both the added disease class and the default “general” disease class.
5. An observation type can have at most 2 attributes, for which a patient can enter values.
6. Apart from the 4 major diseases HIV, High Risk Pregnancy, Obesity and COPD respectively, General class is also a disease class which is class which is assigned to a patient when we enrols in the system.
7. Only Physicians can assign disease to a patient.
8. For generating alerts w.r.t to an observation which belongs to a new observation type apart from the 11 stated in the sample data of the project, we would have to modify the trigger “CHECK\_THRESHOLD”.
9. Aggregated reports can be generated for 10 observation types except diet. For adding a new observation type, code has to be modified accordingly to view the aggregated report for the new observation type.
10. For the observation type “Blood Pressure”, both Systolic and Diastolic values are entered together and not as different attributes.
11. For the “Diet” Observation type, we consider one food item and its amount as 1 observation. For multiple food items, they should be entered as different observations.
12. For adding an observation, only values are entered not their units like for Temperature 100 degree Fahrenheit, we enter the value as “100” and not “100F”

# Problem Statement:

The problem states to build an application where patients can record their observations of daily living so that physicians can have a more complete picture of a patient’s experiences in between doctors’ visits. There are two categories of users who will be using this system, patients and health supporters, i.e., physicians. Users can log into the system using their user id and password, based on which the system will identify whether they are patients or physicians and display the appropriate services to them accordingly.

A patient can enter general observations and observations specific to his disease. He can also add a new observation type. Patients can view their observations and alerts which are generated when an observation is beyond a normal range or value. A patient can have health friends who are patients that share similar characteristics with the patient. Health-friends help re-engage patients that seem to be disengaged from the system, for e.g., if a patient has not viewed his alerts for a long time, health-friends are made aware of this situation and they can contact the patient and ask him to check his alerts.

A physician can assign disease to patients and view observations recorded by patients. He can also add new observation types and associate an observation type with a disease class.

Using a Database is preferred in solving this problem due to the following reasons:

* The system needs to store a large volume of information pertaining to patients, physicians, diseases and observations. It also needs to maintain relationships between the different sets of information.
* The application will be accessed concurrently by multiple users, who might view the information, add or update the information in the system, at the same time.
* The application should support scenarios where a series of changes need be implemented on the information, such that all the changes have to be successful. If one of these changes is unsuccessful, then the whole series of changes should be rejected. These scenarios are supported by the notion of transactions in a Database.

# Entities and Relationships

**Entities:**

The users of the ODL application will be patients and physicians. So we have the following entities:

* USER: It includes all patients and health supporters.
* HEALTH SUPPORTER: It is a sub class of the USER entity and which does not include patients.
* PHYSICIAN: It is a subclass of the HEALTH SUPPORTER entity which includes all physicians and their information.
* PATIENT: It is a subclass of USER entity and consists of patients and their information.
* ADDRESS: This entity is a set of addresses of the patients registered in the system.

There are different disease classes to which a patient can belong, e.g. HIV, COPD, etc.

* DISEASE CLASS: This entity captures the set of disease classes.

Patients will record their observations for different observation types.

* OBSERVATION TYPE: This entity consists of the set of observation types along with their attributes.
* OBSERVATION\_CATEGORY: Each observation type will belong to one observation category (out of Behavioural, Physiological and Psychological). This entity contains the set of Observation Categories.
* OBSERVATION: This entity will contain the observations recorded by patients.

When observations recorded by a patient go beyond a predefined threshold, alerts will be sent to the patient.

* ALERTS: This entity consists of alerts generated for observations recorded by patients.
* HEALTHFRIEND ALERT: This entity consists of the alerts sent to healthfriends of a patient when the patient has not acknowledged his alerts for a certain time.

**Relationships**

Following are the relationships that will be used to build the application.

* ASSIGNS DISEASE: This is a relationship between PATIENT, DISEASE CLASS and PHYSICIAN entities. It records which patient has which disease and which physician has assigned it.
* HEALTH FRIEND: This relationship records the health-friends of patients.
* LIVES IN: This is a relationship between PATIENT and ADDRESS and records which patient lives in which address.
* ASSOCIATE DISEASE AND OBSERVATION TYPE: This relationship stores which observation types need to be recorded for which disease.
* BELONGS TO: This relationship records which observation type belongs to which observation category.
* WRITES LOG: This relationship stores which patient wrote an observation for which observation type and at what date and time.
* GENERATES: This relationship records which alert was generated for which observation entry.
* GENERATES FRIEND’S ALERT: This relationship records which alerts of a patient were visible to which of the patient’s health-friends and what was the message sent by the health-friend to the patient.

# Users

The users that can access the system are:

* 1. Physician
  2. Patient

Following are the roles specific to the users:

1. Physician:
   1. Assign patient to a disease class.
   2. Add new observation types and assign them to specific disease classes.
   3. Associate existing observation types to specific disease classes.
   4. View observations of specific patients.
   5. View consolidated list of observations of patients belonging to a specific observation type.
   6. View aggregated reports/trends for a specific observation type.
2. Patient:
   1. Add a new observation type.
   2. Add new observations belonging to his specific observation types and disease class.
   3. View existing observations for self.
   4. View alerts generated for his observations.
   5. Add new health friends.
   6. View observations of his existing health friends.
   7. View active alerts of his existing health friends.
   8. View health friends at risk, and send messages to them.

Physician and Patient view of the database would differ from each other as their roles differ from each other. Each of them handles different tasks and have specific menu options.

# E-R Diagrams

Consolidated ER Diagram:



Physician ER Diagram:

****

Patient ER Diagram:



Design:

We have two set of users, patient and health supporters, so we have a ISA hierarchy from the users. Physicians are health supporters so again we have another ISA hierarchy. For the projects we are specifically targeting the tasks related to a Physician rather than health supporters.

Physicians can assign diseases to patients, so we have a relationship “Assign Disease” among them. Also Physician can associate a Disease with an observation type, which is specified by the relationship “Associate Disease and Observation Type”.

Address determines where the patient lives, which is depicted in the relationship “Lives in”. Also patient can have health friends which are shown in the relationship “Health Friend” with the attribute Friendship initiation date. Patient writes observations for various observation types specific to his diseases shown by “Writes Log” relation. Alerts can be generated when the observation value exceeds the threshold value shown by the “Generates” relation. Health friends have the option to initiate a message for their friends who are at risk which is shown in the relation “Generate friend Alerts”.

Constraints captured by design:

* 1. Usage of ON DELETE CASCADE:

We have used the referential integrity constraint “ON DELETE CASCADE” while creation of tables to maintain the referential integrity. One usage of this is-

*CREATE TABLE HEALTH\_FRIEND(*

*PATIENT\_ID VARCHAR2(20) ,*

*HEALTH\_FRIEND\_ID VARCHAR2(20) ,*

*DATE\_OF\_INITIATION DATE NOT NULL,*

*PRIMARY KEY (PATIENT\_ID,HEALTH\_FRIEND\_ID) ,*

*FOREIGN KEY (PATIENT\_ID) REFERENCES PATIENT (PATIENT\_ID) ON DELETE CASCADE,*

*FOREIGN KEY (HEALTH\_FRIEND\_ID) REFERENCES PATIENT (PATIENT\_ID) ON DELETE CASCADE,*

*CONSTRAINT VALID\_HEALTH\_FRIEND CHECK (PATIENT\_ID <> HEALTH\_FRIEND\_ID)*

*);*

In the above DDL for Health friend, using the highlighted statement, in case if a patient is deleted from the patient table, all corresponding records will be deleted from the health friend table as well.

* 1. Usage of ON DELETE SET NULL:

We have used the referential integrity constraint “ON DELETE SET NULL” while creation of tables to maintain the referential integrity. One usage of this is-

*CREATE TABLE ASSIGN\_DISEASE(*

*PATIENT\_ID VARCHAR2(20) ,*

*CLASS\_ID INTEGER ,*

*PHYSICIAN\_ID VARCHAR2(20) ,*

*PRIMARY KEY (PATIENT\_ID,CLASS\_ID),*

*FOREIGN KEY (PATIENT\_ID) REFERENCES PATIENT (PATIENT\_ID) ON DELETE CASCADE,*

*FOREIGN KEY (CLASS\_ID) REFERENCES DISEASE\_CLASS (CLASS\_ID) ON DELETE CASCADE,*

*FOREIGN KEY (PHYSICIAN\_ID) REFERENCES PHYSICIAN (PHYSICIAN\_ID) ON DELETE SET NULL*

*);*

In the above DDL for Assign Disease using the highlighted statement, in case if a physician is deleted from the physician table the corresponding record should not be deleted as the patient is still in the system, so in this case we should assign “NULL” to the physician attribute.

# Relational Schema

Below is the relational schema for the project.



# Functional Dependencies and Normal Forms:

Following are the functional dependencies associated with the relations:



An example of the decomposition of relation into 3NF/BCNF is:

Consider the relation –

OBSERVATION\_TYPE\_CATEGORY (**type\_id**, type\_name, threshold\_val, additional\_info\_1, additional\_info\_2, category\_id, category\_name)

With the primary key as type\_id

This relation is not in 3NF/BCNF as category name is dependent on category id.

So we decompose the relation into 3 sub relations namely:

OBSERVATION\_TYPE(**type\_id**, type\_name, threshold\_val, additional\_info\_1, additional\_info\_2)

OBSERVATION\_CATEGORY(**category\_id**, category\_name)

OBSERVATION\_TYPE\_CATEGORY(category\_id, **type\_id**)

Now the relation is in BCNF.

# APIs and SQL Queries:

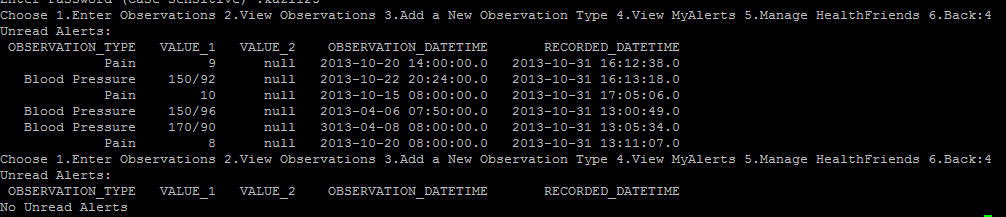
**List of APIs used in the project:**

* 1. UPDATE\_ALERTS(Patient id IN, Rows\_Updated OUT)

*Return*: no. of rows updated.

*Description*: The stored procedure “UPDATE\_ALERTS” is used to update the alert status of the alert, after the patient has read his alerts. It takes the patient id as the input parameter and returns the number of rows updated as the output parameter.

Example:



By usage of this Stored procedure, once patient has viewed his alerts, the alerts do not appear again.

**Transactions:**

Following is a scenario, where we have used transaction:

Example:

When a new patient enrols in the system, the system will ask basic information from the patient, depending on which the data will be stored in different tables. Typically for a new patient, his username and password will be stored in the USER\_LOGIN table, his personal details like age, sex, public status will be stored in the PATIENT table, his address details in the PATIENT\_ADDRESS table and an entry in the ASSIGN\_DISEASE table which assigns the patient a default “general” disease class.

Now these insertion mechanism should be atomic i.e. all should be executed or none, which is what we have followed in the project.

Below is the code snippet for the same:

*try {*

*……………….*

*……………….*

*int user\_login\_row = stmt.executeUpdate("INSERT INTO USER\_LOGIN VALUES(" + "'" + username + "','" + password + "','1')");*

*int patient\_row = stmt.executeUpdate("INSERT INTO PATIENT VALUES(" + "'" + username + "','" + patientName + "'," + age + ",'" + sex + "','" + publicStatus + "')");*

*int address\_row = stmt.executeUpdate("INSERT INTO PATIENT\_ADDRESS(PATIENT\_ID,STREET\_ADDRESS, CITY,STATE\_NAME,ZIP) VALUES(" + "'" + username + "','" + street + "','" + city + "','" + stateName + "'," + zip + ")");*

*//ASSIGN GENERAL DISEASE CLASS TO NEW PATIENT*

*int assign\_disease\_row = stmt.executeUpdate("INSERT INTO assign\_disease(PATIENT\_ID, CLASS\_ID) VALUES(" + "'" + username + "'," + class\_id + ")");*

*if(user\_login\_row==1 && patient\_row==1 && address\_row==1 && assign\_disease\_row==1)*

*{*

*stmt.executeUpdate("COMMIT");*

*System.out.println("User created");*

*}*

*else*

*{*

*stmt.executeUpdate("ROLLBACK");*

*System.out.println("User Not Created");*

*}*

*………………………*

*………………………*

*catch(Throwable oops){*

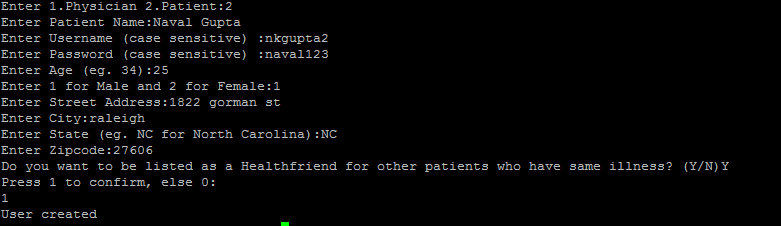
*stmt.executeUpdate("ROLLBACK");*

*System.out.println("Wrong Option Entered, Program will now exit");*

*oops.printStackTrace();*

*}*

*Execution:*



Following is another scenario of the use of transactions:

When a physician adds a new observation type, he adds various information regarding that observation like name, threshold value, additional information etc., these values should be inserted in the table OBSERVATION\_TYPE, also the observation type should be associated to an observation category which is stored in the table OBSERVATION\_TYPE\_CATEGORY, and finally an entry should be made in the DISEASE\_OBSERVATION\_TYPE which maintains the relation between a disease class and the new observation type.

Now these insertion mechanism should be atomic i.e. all should be executed or none, which is what we have followed in the project.

Below is the code snippet for the same:

*try {*

*……………….*

*……………….*

*int execute1 = stmt.executeUpdate("insert into OBSERVATION\_TYPE(TYPE\_NAME,THRESHOLD\_VAL, ADDITIONAL\_INFO\_1,ADDITIONAL\_INFO\_2) VALUES ('" + new\_obv\_type + "','" + threshold + "','" + new\_obv\_att1 + "','" + new\_obv\_att2 + "')" );*

*//use the new obv type id and cat id to insert a tuple in table OBSERVATION\_TYPE\_CATEGORY*

*rs = stmt.executeQuery("select TYPE\_ID from OBSERVATION\_TYPE where TYPE\_NAME = '" + new\_obv\_type +"'");*

*int obv\_type\_id = 0;*

*while(rs.next())*

*{*

*obv\_type\_id = rs.getInt("TYPE\_ID");*

*}*

*int execute2 = stmt.executeUpdate("insert into OBSERVATION\_TYPE\_CATEGORY VALUES (" + obv\_cat + "," + obv\_type\_id + ")");*

*// use new type id and disease id to insert a tuple in DISEASE\_OBSERVATION\_TYPE*

*int execute3 = stmt.executeUpdate("insert into DISEASE\_OBSERVATION\_TYPE VALUES (" + disease\_id + "," + obv\_type\_id +")" );*

*if(execute1==1 && execute2==1 && execute3==1)*

*{*

*stmt.executeUpdate("COMMIT");*

*System.out.println("New Observation type inserted");*

*}*

*else*

*{*

*stmt.executeUpdate("ROLLBACK");*

*System.out.println("New Observation type could not be inserted");*

*}*

*………………………*

*………………………*

*catch(Throwable oops){*

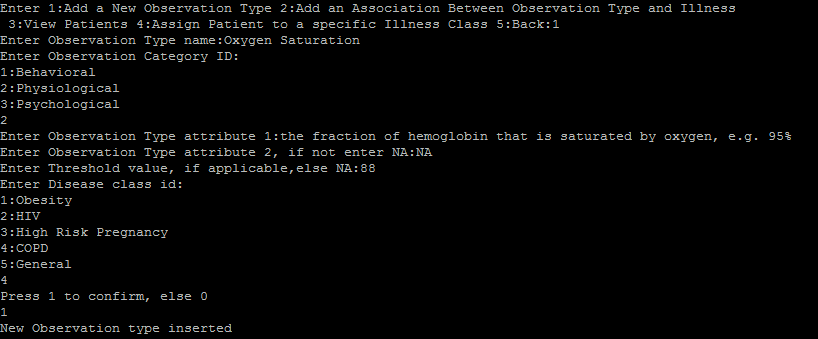
*stmt.executeUpdate("ROLLBACK");*

*System.out.println("Wrong Option Entered, Program will now exit");*

*oops.printStackTrace();*

*}*

*Execution:*

**

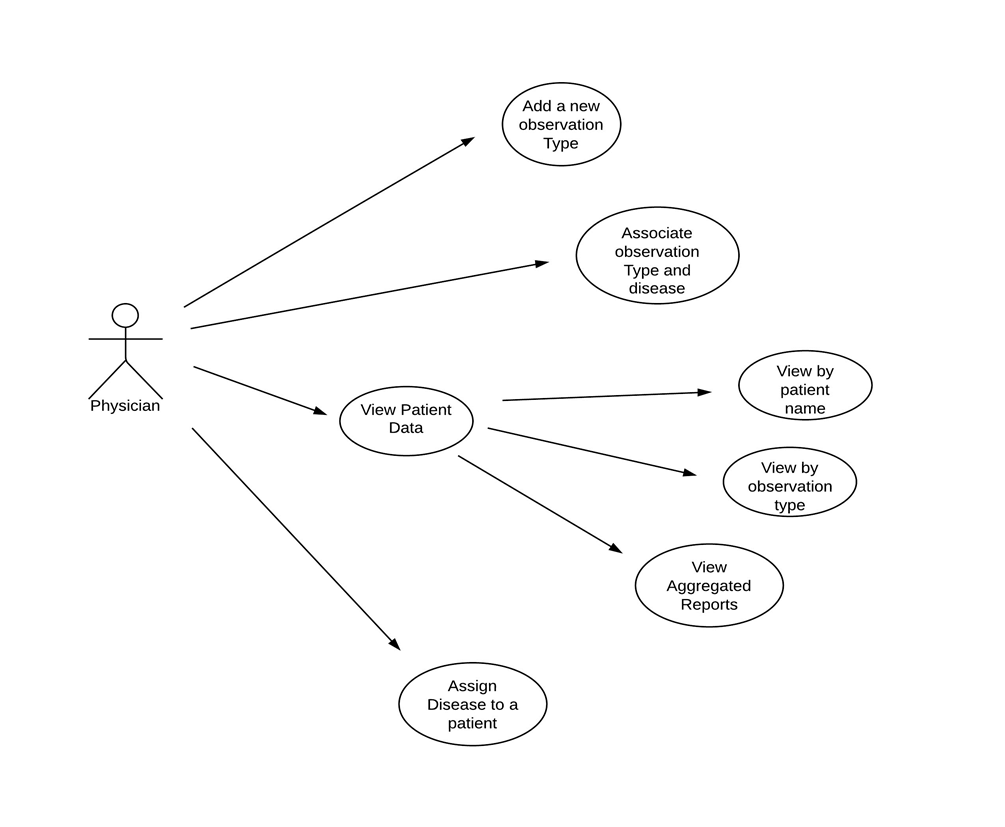
**Triggers:**

An instance of the use of trigger is when the patient enters an observation, if the value of the observation is greater than the threshold value, our system will automatically generate an alert for this observation using the trigger “CHECK\_THRESHOLD”.

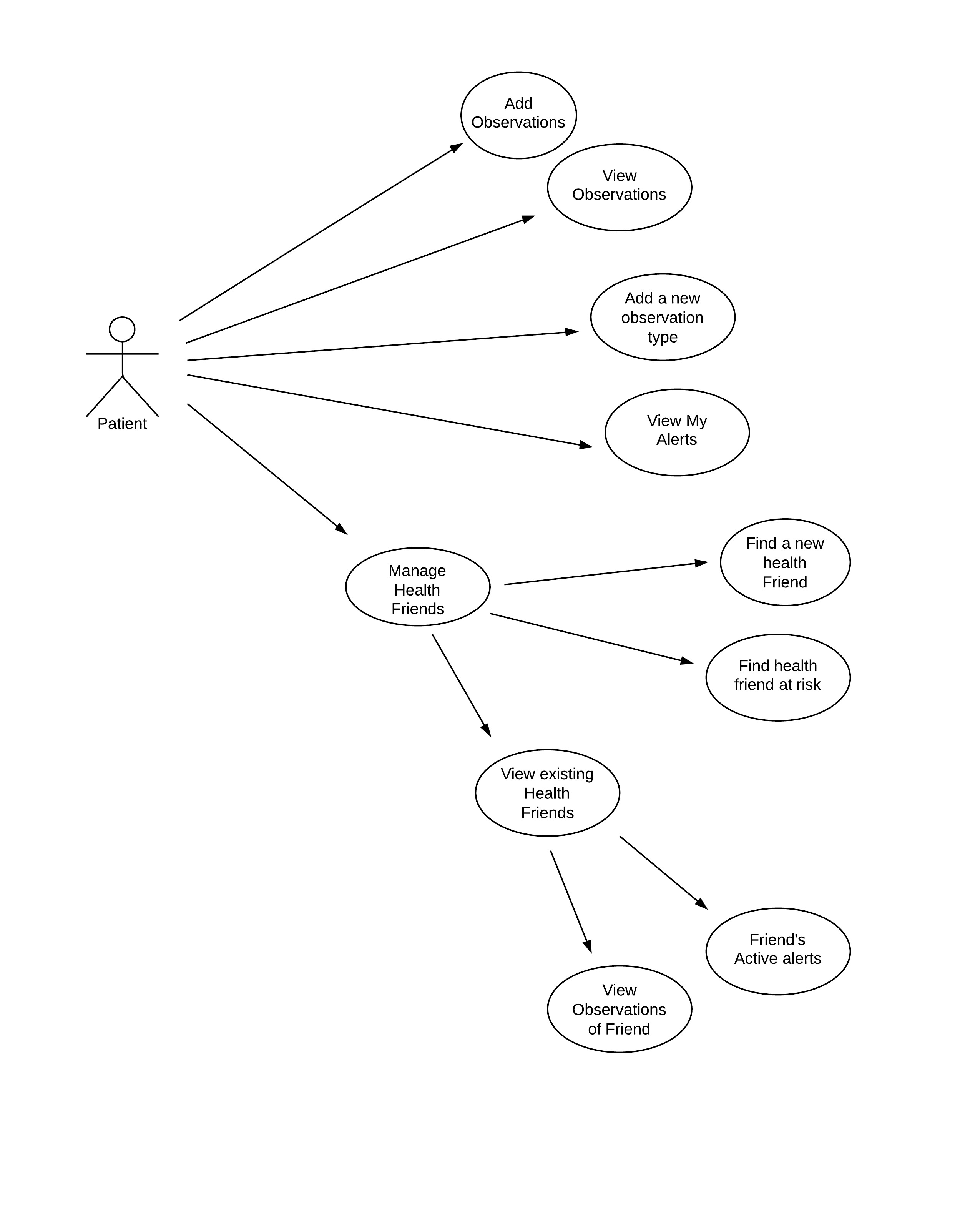
Another instance in which we have used triggers is in the generation of sequence numbers which is used in almost all the tables. For example in the table PATIENT\_ADDRESS, if a new patient enrols then his address details should be inserted in the PATIENT\_ADDRESS table, where ADDRESS\_ID is generated at run time using triggers.

# Use-case Scenario:

**Physician:**



The above is a use case diagram for a physician. It shows the functions that a physician can perform in the application. A physician can assign disease to a patient, add new observation types, associate an observation type with a disease and view patient information. A physician can view patient records in three different ways; viz., individual reports of patient by name, reports based on observation type and aggregated reports.

**Patient:**

The above is a use case diagram for a patient. It shows the functions that a patient can perform in the application. A patient can add observations, view his observations, add new observation types and view alerts generated by observations beyond a normal range or value. A patient can also add new health-friends and view observations and active alerts of existing health-friends. A patient can find health-friends who have not viewed their alerts for a long time and can contact them regarding this issue.

# Extra Features Added:

Following are the extra features added apart from the specifications provided:

* 1. Check if a user already exists in the system when a new user enrols with the same username.
  2. Prompting users if they enter invalid credentials.
  3. Ability of a physician to assign disease to a patient.
  4. When a physician associates an observation type with a disease class, check if the association is already present in the system.
  5. When a physician adds a new observation type, check if the new observation type is already present in the system.
  6. Physicians have the option of viewing aggregated reports on the basis of observation type.
  7. When a patient adds a new observation type, check if the new observation type is already present in the system.