
Personal Health Management (PHM)
Database Application
Final Report

Created by

JINGJUAN DENG
MENGYING WANG
RAN TAN
SHUAI CHEN

NC State University

CSC 540 - DATABASE MANAGEMENT SYSTEM
DR. KEMAFOR OGAN
OCTOBER 31, 2016

1 Description

In this project, a Personal Health Management Database Application is implemented to assist people (both healthy and otherwise) tracking and managing health information. People can track their "observations about certain health indicators and activities of daily living", whose patterns can be treated as a reflection of their health status. Tracking such information helps people monitoring their health status through alerts, whenever something is wrong or patients become slack about health management. As is becoming increasingly common, one or more health supporters are encouraged to assist, motivate and encourage patients to comply with directives. Consequently, a useful personal health management application will help recording and monitoring specific health indicators, raising alerts to notify both patient and their supporters (where appropriate). It significantly saves the effort to track and monitor personal health information.

In this report, we will first describe the E-R Model of the application, with attributes and key-constraint details for each entity and relationship. Secondly, a Relational Model is introduced, which describes the schema and data in the database. Thirdly, different kinds of constraints involved in this application is elaborated. The last part in the report is the snapshots for required query results.

2 E-R Model

2.1 E-R Model Diagram

This part describes entities and their keys and attributes. Entities are in bold font format and key of each entity is underlined.

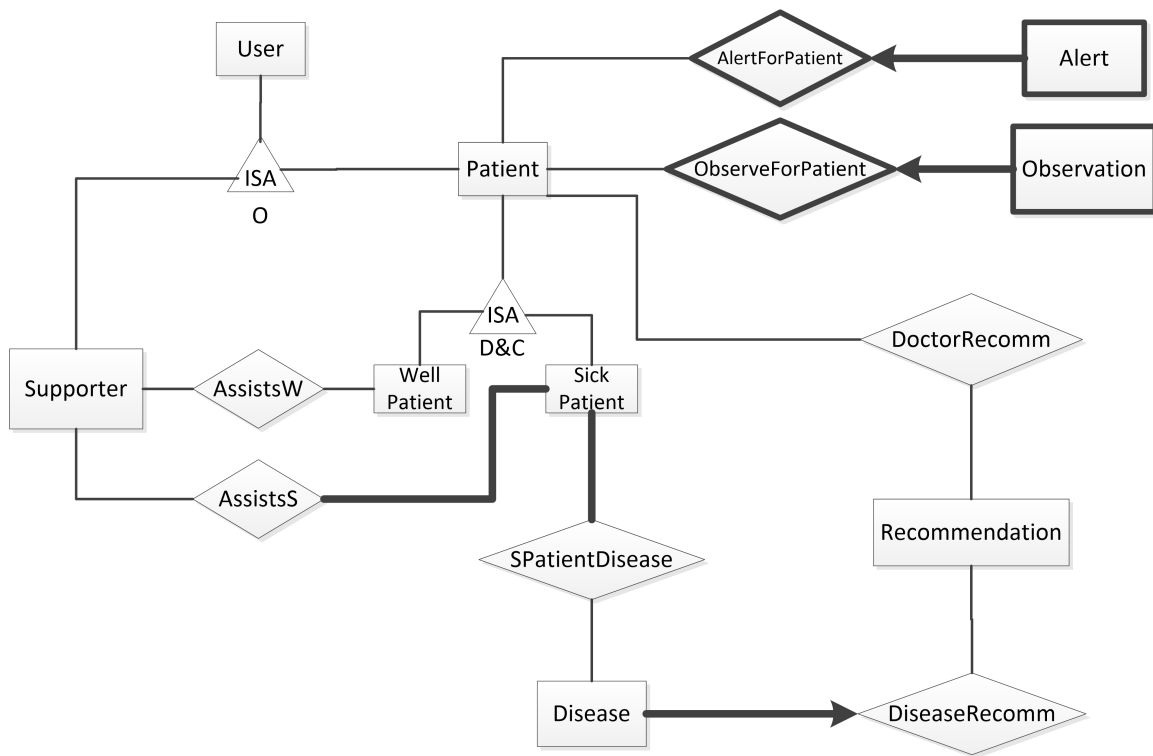


Figure 2.1: Flow Chart of Login Stage

Users (UserID, Pw)

User is an entity that use the system. Every user should be identical from others by ID. The primary key is UserID.

Patient (UserID, Name, DOB, Addr, Gender, SickOrNot)

-**WellPatient** (UserID) is a subset of Patient

-**SickPatient** (UserID) is a subset of Patient

Patient is one kind of user whose health observation should be recorded and supervised. Patient is inherited from user, its primary key is UserID.

Supporter(UserID, Name, Phone)

Supporter is an entity inherited from User whose responsibility is supervising patient's health

condition. The primary key is UserID.

Observation (UserID, ObserveTime, RecTime, Weight_lb, BPS_mmHg, BPD_mmHg, Pain_level, Mood_level, Oxy_pct, Temperature_F)

Observation is a weak entity of Patient. It records health information of patients at each observing time. The primary key is ObserveTime(partial key) and UserID(foreign key).

(BPS — Blood Pressure Systolic, BPD — Blood Pressure Diastolic, pct – percentage)

Disease (Did, Dname)

This entity represents types of disease. For example, heart disease, HIV, etc. Its primary key is Did.

Recommendation (Rid, Weight_bool, Wfreq, Wuplimit_lb, Wlowlimit_lb, BP_bool, BPfreq, BPSuplimit_mmHg, BPSlowlimit_mmHg, BPDuplimit_mmHg, BPDlowlimit_mmHg, Pain_bool, Pfreq, Mood_bool, Mfreq, Oxy_bool, Ofreq, Ouplimit_pct, Olowlimit_pct, Temperature_bool, Tfreq, Tuplimit_F, Tlowlimit_F)

Recommendation is an entity includes patient health observing recommendation. It combines general recommendation for well patient, different kinds of disease recommendations and doctor's recommendations for specific patients. The primary key is Rid, and will be automatically increased when new recommendations are added.

Alert (UserID, Time, Atype, Amsg, Active)

Alert is generated by the query result of observation and recommendation. This is an weak entity to patient, with UserID and Time together as the primary key.

2.2 Relationship

This part describes relationships and their keys and attributes. Relationships are in bold font format and their foreign keys are underlined.

ObserveForPatient (UserID, ObserveTime)

ObserveForPatient is a binary relationship between Patient and Observation, inherited by well patient and sick patient. Foreign key is UserID and ObserveTime.

AssistsW (S_UserID, W_UserID, Since, Priority)

AssistsW is a binary relationship between Supporter and WellPatient. Describe supporters associated with a well patient. Foreign key is S_UserID and W_UserID.

AssistsS (S_UserID, Sick_UserID, Since, Priority)

AssistsS is a binary relationship between Supporter and SickPatient. Describe supporters associated with a sick patient. Foreign key is S_UserID and Sick_UserID.

PatientDisease (UserID, Did)

PatientDisease is a binary relationship between SickPatient and Disease. Describe what diseases a patient has. The foreign key is UserID and Did.

DoctorRecomm (UserID, Rid)

DoctorRecomm is a binary relationship between Patient and Recommendation, inherited by well and sick patient. It describes doctor's recommendation offered for specific patients. The foreign key is UserID and Rid.

DiseaseRecomm (Did, Rid)

DiseaseRecomm is a binary relationship of Disease and Recommendation. It describes recommendation for specific disease. The foreign key is Did and Rid.

Hierarchical relationship: the entities **WellPatient**, **SickPatient** are subclasses of entity **Patient**.

2.3 Constraints

2.3.1 Key constraints and participation constraints

For primary keys in each table, they obey the primary key constraint. The database requires them to be not null. And the primary key can not repeat in a table. For foreign keys in each table, they must obey the foreign key constraint. When table associated contain foreign key changed, other tables associated with the foreign key must be checked.

Particularly, in our relational schema, the key constraints and participation constraints are listed in table 2.1.

Observation is a weak entity of Patient. In this case, Patient and Observation must participate in a one-to-many relationship set, and Observation must have total participation in the relationship. As a result, the relationship require each record of Observation must has and only has one patient.

Entity	Relationship	Key Constraint	Participation Constraint
Observation	ObserveForPatient	Foreign key constraint depend on Patient(UserID). No more than one observation should be recorded for the same patient at the same time.	Total
Alert	AlertForPatient	Foreign key constraint depend on Patient(UserID). No more than one alert should be recorded for the same patient at the same time.	Total
SickPatient	AssistS		Total
SickPatient	SickPatientDisease		Total
Disease	DiseaseRecomm	At most one	Total

Table 2.1: Key Constraints and Participation Constraints

Alert is also a weak entity for Patient, which means Patient and Alert must participate in a one-to-many relationship set, and Alert must have total participation in the relationship.

As for a sick patient, one must have one or more diseases and supporters. So every record in SickPatient must participate in relationships AssistsS and SickPatientDisease.

2.3.2 Hierarchy constraints

Superclass	Subclasses	Overlap	Covering
Users	Supporter and Patient	Allowed	Not required
Patient	Well Patient and Sick Patient	Disallowed	Required

Table 2.2: Hierarchy Constraints

Supporter and Patient are inherited from User. This hierarchy allows overlap and do not require covering. In this case, a user can be a Supporter as well as a Patient. There exist users who are neither Patients nor Supporters.

Well Patient and Sick Patient are inherited from Patient: This hierarchy disallows overlap and require covering constraint. In this case, a user cannot be both a Well Patient and a Sick Patient. There exist no patients who are neither Well Patients nor Sick Patient.

3 Relation Model

Table3.1 lists all tables in the system database. When create these table, we add appropriate constraints using CHECK or TRIGGER. And we output the notice to screen when user violate some constraints. Besides constraints, we also check the possible arrange of an observation value by DBMS. Specific constraints will be illustrated in the table part.

Table name	Key	Discription
User	UserID	User account
Patient	UserID	Store patients' information
WellPatient	UserID	Index of well patient
SickPatient	UserID	Index of sick patient
Supporter	UserID	Store supporters' information
ObservationForPatient	UserID(partial), ObserveTime(foreign)	Combine entity Observation and relationship ObserveForPatient as record of patient's health observation
Recommendation	Rid	Observing recommendation
DiseaseDRecomm	Did(foreign),Rid(foreign)	Combine Disease and DiseaseRecomm
Alert	Time(partial), UserID(foreign)	Alert records
AssistsW	SUserID(foreign), WUserID(foreign)	Associate supporter with well patient
AssistsS	SUserID(foreign), SickUserID(foreign)	Associate supporter with sick patient
PatientDisease	UserID(foreign), Did(foreign)	Associate sick patient with disease
DoctorRecomm	Did(foreign), Rid(foreign)	Associate doctor's recommendation with Patient

Table 3.1: Relational Schema

3.1 Schema

The following sections introduce the relational schema, including CREATE TABLE queries, constraints and functional dependencies.

3.1.1 Users

Constraints:

Table Users has primary key constraints that the primary key UserID must be set as not null and can not repeat. Although password is not key, it should be not null according to the application requirement.

SQL:

```
CREATE TABLE Users (  
  UserID VARCHAR2(20) NOT NULL,  
  Pw VARCHAR2(20) NOT NULL  
  Pw RAW(200) NOT NULL,  
  CONSTRAINT U_UserID_Should_be_Unique PRIMARY KEY (UserID));
```

Functional Dependencies:

User: F_user = { UserID -> Pw }

3.1.2 Patient

Constraints:

UserID in table Patient can not be null. And if any record in table Users is deleted, the related record in table Patient must be deleted as well. Data value range is check for SickOrNot which should not be negative.

SQL:

```
CREATE TABLE Patient  
  UserID VARCHAR2(20) NOT NULL,  
  Name VARCHAR2(40) NOT NULL,  
  DOB DATE NOT NULL,  
  Gender NUMBER(1) NOT NULL,  
  Address VARCHAR2(50) NOT NULL,  
  SickOrNot NUMBER(1) NOT NULL,  
  CONSTRAINT P_UserID_Should_be_Unique PRIMARY KEY (UserID),
```

FOREIGN KEY (UserID) REFERENCES Users ON DELETE CASCADE,
CONSTRAINT P_SickOrNot_Out_Of_Range CHECK (SickOrNot <= 1 AND SickOrNot >= 0));

Functional Dependencies:

F_patient = { UserID -> Name DOB Addr Gender }

3.1.3 WellPatient

Constraints:

UserID in table WellPatient can not be null. And if any record in table Patient is deleted, the related record in table WellPatient must be deleted as well. Well patients can have 0 to 2 supporters. This constraint is checked in the table creation statement.

SQL:

```
CREATE TABLE WellPatient
UserID VARCHAR2(20),
PRIMARY KEY (UserID),
FOREIGN KEY (UserID) REFERENCES Patient ON DELETE CASCADE)
```

Functional Dependencies:

F_WellPatient={ ϕ }

3.1.4 SickPatient

Constraints:

UserID in table SickPatient can not be null. And if any record in table Patient is deleted, the related record in table SickPatient must be deleted as well. Sick patients should have 1 to 2 supporters. This constraint is checked in the table creation statement.

SQL:

```
CREATE TABLE SickPatient (
UserID VARCHAR2(20),
PRIMARY KEY (UserID),
FOREIGN KEY (UserID) REFERENCES Patient ON DELETE CASCADE)
```

Functional Dependencies:

F_SickPatient={ ϕ }

3.1.5 Supporter

Constraints:

UserID and the connecting information in table Supporter can not be null. And if any record in table Users is deleted, the related record in table Supporter must be deleted as well.

SQL:

```
CREATE TABLE Supporter (  
  UserID VARCHAR2(20) NOT NULL,  
  Name VARCHAR2(40) NOT NULL,  
  Mobile VARCHAR2(20) NOT NULL,  
  CONSTRAINT S_UserID_Should_be_Unique PRIMARY KEY (UserID),  
  FOREIGN KEY (UserID) REFERENCES Users ON DELETE CASCADE)
```

Functional Dependencies:

F_supporter = { UserID -> Contact }

3.1.6 Observation_ForPatient

Constraints:

Primary key ObserveTime and foreign key UserID in this table can not be null. And if any record in table Patient is deleted, the related record in this table must be deleted as well. Consider the participation and key constraint of entity Observation in relationship ObserveForPatient, we design to combine these two table as one.

SQL:

```
CREATE TABLE Observation_ForPatient (  
  UserID VARCHAR2(20) NOT NULL,  
  ObserveTime TIMESTAMP NOT NULL,  
  RecTime TIMESTAMP NOT NULL,  
  Weight_lb NUMBER(3),  
  BPS_mmHg NUMBER(3),  
  BPD_mmHg NUMBER(3),  
  Pain_level NUMBER(2),
```

Mood_level NUMBER(1),
 Oxy_pct NUMBER(3),
 Temperature_F NUMBER(3), CONSTRAINT O_UserID_ObsvTime_Not_Unique PRIMARY KEY
 (UserID, ObserveTime),
 FOREIGN KEY (UserID) REFERENCES Patient ON DELETE CASCADE,
 CONSTRAINT O_Weight_lb_Out_Of_Range CHECK ((Weight_lb >= 0) OR (Weight_lb IS NULL)),
 CONSTRAINT O_BPS_mmHg_Out_Of_Range CHECK ((BPS_mmHg >= 0) OR (BPS_mmHg IS
 NULL)),
 CONSTRAINT O_BPD_mmHg_Out_Of_Range CHECK ((BPD_mmHg >= 0) OR (BPD_mmHg IS
 NULL)),
 CONSTRAINT O_Pain_level_Out_Of_Range CHECK ((Pain_level >= 0 AND Pain_level <= 10) OR
 (Pain_level IS NULL)),
 CONSTRAINT O_Mood_level_Out_Of_Range CHECK ((Mood_level >= 1 AND Mood_level <= 3)
 OR (Mood_level IS NULL)),
 CONSTRAINT O_Oxy_pct_Out_Of_Range CHECK ((Oxy_pct >= 0 AND Oxy_pct <= 100) OR (Oxy_pct
 IS NULL)),
 CONSTRAINT O_Temperature_F_Out_Of_Range CHECK ((Temperature_F >= 0) OR (Tempera-
 ture_F IS NULL)))

Functional Dependencies:

F_observation = { UserID Observetime -> RecTime Weight Pressure Pain Mood Oxy Tempera-
 ture) }

3.1.7 Recommendation

Constraints:

Primary key Rid in table Recommendation can not be null. Only available options should be
 the options for observation types associated with the patient classes that the patient belongs to.
 This can be achieved by check values when the available flag is on. Every recommended health
 indicator has to be in certain value range, so corresponding constraints are added to check if
 input values are out of normal range.

SQL:

CREATE TABLE Recommendation (
 Rid NUMBER(3) NOT NULL,
 Weight_bool NUMBER(1) NOT NULL,
 Wfreq NUMBER(3),

Wuplimit_lb NUMBER(3),
 Wlowlimit_lb NUMBER(3),
 BP_bool NUMBER(1) NOT NULL,
 BPfreq NUMBER(3),
 BPSuplimit_mmHg NUMBER(3),
 BPSlowlimit_mmHg NUMBER(3),
 BPDuplimit_mmHg NUMBER(3),
 BPDlowlimit_mmHg NUMBER(3),
 Pain_bool NUMBER(1) NOT NULL,
 Pfreq NUMBER(3),
 Pvalue NUMBER(2),
 Mood_bool NUMBER(1) NOT NULL,
 Mfreq NUMBER(3),
 Mvalue NUMBER(1),
 Oxy_bool NUMBER(1) NOT NULL,
 Ofreq NUMBER(3),
 Ouplimit_pct NUMBER(3),
 Olowlimit_pct NUMBER(3),
 Temp_bool NUMBER(1) NOT NULL,
 Tfreq NUMBER(3),
 Tuplimit_F NUMBER(3),
 Tlowlimit_F NUMBER(3),
 AlertThres_pct NUMBER(3),
 ConsecutiveNum NUMBER(2),
 CONSTRAINT R_Rid_Should_be_Unique PRIMARY KEY (Rid),
 CONSTRAINT R_Wfreq_Out_Of_Range CHECK ((Wfreq > 0) OR (Wfreq IS NULL)),
 CONSTRAINT R_Wuplimit_lb_Out_Of_Range CHECK ((Wuplimit_lb >= 0) OR (Wuplimit_lb IS NULL)),
 CONSTRAINT R_Wlowlimit_lb_Out_Of_Range CHECK ((Wlowlimit_lb >= 0) OR (Wlowlimit_lb IS NULL)),
 CONSTRAINT R_Wuplimit_LessThan_Wlowlimit CHECK ((Wuplimit_lb >= Wlowlimit_lb) OR ((Wuplimit_lb IS NULL) AND (Wlowlimit_lb IS NULL))),
 CONSTRAINT R_Weight_Recom_illegal CHECK (((Weight_bool = 0) AND (Wfreq IS NULL) AND (Wuplimit_lb IS NULL) AND (Wlowlimit_lb IS NULL)) OR ((Weight_bool = 1) AND (Wfreq IS NOT NULL) AND (Wuplimit_lb IS NOT NULL) AND (Wlowlimit_lb IS NOT NULL))),
 CONSTRAINT R_BPfreq_Out_Of_Range CHECK ((BPfreq > 0) OR (BPfreq IS NULL)),
 CONSTRAINT R_BPSuplimit_Out_Of_Range CHECK ((BPSuplimit_mmHg >= 0) OR (BPSuplimit_mmHg

IS NULL)),

CONSTRAINT R_BPSlowlimit_Out_Of_Range CHECK ((BPSlowlimit_mmHg >= 0) OR (BPSlowlimit_mmHg IS NULL)),

CONSTRAINT R_BPSuplim_LessThan_lowlimit CHECK ((BPSuplimit_mmHg >= BPSlowlimit_mmHg) OR ((BPSuplimit_mmHg IS NULL) AND (BPSlowlimit_mmHg IS NULL))),

CONSTRAINT R_BPDuplimit_Out_Of_Range CHECK ((BPDuplimit_mmHg >= 0) OR (BPDuplimit_mmHg IS NULL)),

CONSTRAINT R_BPDlowlimit_Out_Of_Range CHECK ((BPDlowlimit_mmHg >= 0) OR (BPDlowlimit_mmHg IS NULL)),

CONSTRAINT R_BPDupLim_LessThan_lowlimit CHECK ((BPDuplimit_mmHg >= BPDlowlimit_mmHg) OR ((BPDuplimit_mmHg IS NULL) AND (BPDlowlimit_mmHg IS NULL))),

CONSTRAINT R_BP_Recom_illegal CHECK (((BP_bool = 0) AND (BPfreq IS NULL) AND (BPSuplimit_mmHg IS NULL) AND (BPSlowlimit_mmHg IS NULL) AND (BPDuplimit_mmHg IS NULL) AND (BPDlowlimit_mmHg IS NULL)) OR ((BP_bool = 1) AND (BPfreq IS NOT NULL) AND (BPSuplimit_mmHg IS NOT NULL) AND (BPSlowlimit_mmHg IS NOT NULL) AND (BPDuplimit_mmHg IS NOT NULL) AND (BPDlowlimit_mmHg IS NOT NULL))),

CONSTRAINT R_Pfreq_Out_Of_Range CHECK ((Pfreq > 0) OR (Pfreq IS NULL)),

CONSTRAINT R_Pvalue_Out_Of_Range CHECK ((Pvalue >= 0 AND Pvalue <= 10) OR (Pvalue IS NULL)),

CONSTRAINT R_Pain_Recom_illegal CHECK (((Pain_bool = 0) AND (Pfreq IS NULL) AND (Pvalue IS NULL)) OR ((Pain_bool = 1) AND (Pfreq IS NOT NULL) AND (Pvalue IS NOT NULL))),

CONSTRAINT R_Mfreq_Out_Of_Range CHECK ((Mfreq > 0) OR (Mfreq IS NULL)),

CONSTRAINT R_Mvalue_Out_Of_Range CHECK ((Mvalue >= 1 AND Mvalue <= 3) OR (Mvalue IS NULL)),

CONSTRAINT R_Mood_Recom_illegal CHECK (((Mood_bool = 0) AND (Mfreq IS NULL) AND (Mvalue IS NULL)) OR ((Mood_bool = 1) AND (Mfreq IS NOT NULL) AND (Mvalue IS NOT NULL))),

CONSTRAINT R_Ofreq_Out_Of_Range CHECK ((Ofreq > 0) OR (Ofreq IS NULL)),

CONSTRAINT R_Ouplimit_pct_Out_Of_Range CHECK ((Ouplimit_pct >= 0) OR (Ouplimit_pct IS NULL)),

CONSTRAINT R_Olowlimit_pct_Out_Of_Range CHECK ((Olowlimit_pct >= 0) OR (Olowlimit_pct IS NULL)),

CONSTRAINT R_Ouplimit_LessThan_Olowlimit CHECK ((Ouplimit_pct >= Olowlimit_pct) OR ((Ouplimit_pct IS NULL) AND (Olowlimit_pct IS NULL))),

CONSTRAINT R_Oxy_Recom_illegal CHECK (((Oxy_bool = 0) AND (Ofreq IS NULL) AND (Ouplimit_pct IS NULL) AND (Olowlimit_pct IS NULL)) OR ((Oxy_bool = 1) AND (Ofreq IS NOT NULL) AND (Ouplimit_pct IS NOT NULL) AND (Olowlimit_pct IS NOT NULL))),

```

CONSTRAINT R_Tfreq_Out_Of_Range CHECK ((Tfreq > 0) OR (Tfreq IS NULL)),
CONSTRAINT R_Tuplimit_F_Out_Of_Range CHECK ((Tuplimit_F >= 0) OR (Tuplimit_F IS NULL)),
CONSTRAINT R_Tlowlimit_F_Out_Of_Range CHECK ((Tlowlimit_F >= 0) OR (Tlowlimit_F IS
NULL)),
CONSTRAINT R_Tuplimit_LessThan_Tlowlimit CHECK ((Tuplimit_F >= Tlowlimit_F) OR ((Tu-
plimit_F IS NULL) AND (Tlowlimit_F IS NULL))),
CONSTRAINT R_Temp_Recom_illegal CHECK (((Temp_bool =0) AND (Tfreq IS NULL) AND
(Tuplimit_F IS NULL) AND (Tlowlimit_F IS NULL)) OR ((Temp_bool =1) AND (Tfreq IS NOT
NULL) AND (Tuplimit_F IS NOT NULL) AND (Tlowlimit_F IS NOT NULL))));

```

Functional Dependencies:

F_recommendation = { Rid -> Weight Wfreq Wuplimit Wlowlimit Pressure BPfreq BPuplimit
BPlowlimit Pain Pfreq Mood Mfreq Oxy Ofreq Ouplimit Olowlimit Temperature Tfreq Tuplimit
Tlowlimit AlertThres ConsecutiveNum }

3.1.8 Disease_DRecomm

Constraints:

Primary key Did can not be null. Foreign key Rid in Disease_DRecomm can not be null. And if any record in table Recommendation is deleted, the related record in table Disease_DRecomm must be deleted as well. Consider the participation and key constraint of entity Disease in relationship DiseaseRecomm, we combine these two tables as one.

SQL:

```

CREATE TABLE Disease_DRecomm (
Did VARCHAR2(20) NOT NULL,
Dname VARCHAR2(40) NOT NULL,
Rid NUMBER(3) NOT NULL,
CONSTRAINT DDR_Did_Should_be_Unique PRIMARY KEY (Did),
FOREIGN KEY (Rid) REFERENCES Recommendation ON DELETE CASCADE)

```

Functional Dependencies:

F_diseaserecomm == { Did -> Dname }

3.1.9 Alert

Constraints:

Primary key UserID and Time in this table can not be null. And if any record in table Patient is deleted, the related record in this table must be deleted as well.

SQL:

```
CREATE TABLE Alert (  
  UserID VARCHAR2(20) NOT NULL,  
  Time TIMESTAMP WITH LOCAL TIME ZONE NOT NULL,  
  AType NUMBER(1) NOT NULL,  
  AMsg VARCHAR2(100),  
  Active NUMBER(1) NOT NULL,  
  PRIMARY KEY (UserID, Time),  
  FOREIGN KEY (UserID) REFERENCES Patient ON DELETE CASCADE);
```

Functional Dependencies:

F_alert = { UserID Time -> Amsg Read Active }

3.1.10 AssistsW

Constraints:

Foreign key UserID in this table can not be null. And if any record in table Supporter and table WellPatient is deleted, the related record in this table must be deleted as well. A well patient can have 0 to 2 supporters, so the value of priority should be in range of 0 to 2.

SQL:

```
CREATE TABLE AssistsW ( P_UserID VARCHAR2(20) NOT NULL,  
  S_UserID VARCHAR2(20) NOT NULL,  
  Since DATE NOT NULL,  
  Priority NUMBER(1) NOT NULL,  
  CONSTRAINT AS_PUID_Prioy_NOT_Unique PRIMARY KEY (P_UserID, Priority),  
  FOREIGN KEY (S_UserID) REFERENCES Supporter ON DELETE CASCADE,  
  FOREIGN KEY (P_UserID) REFERENCES Patient ON DELETE CASCADE,  
  CONSTRAINT AS_PUID_SUID_NOT_Unique UNIQUE (P_UserID, S_UserID),  
  CONSTRAINT AS_PUID_SUID_Should_diff CHECK (P_UserID <> S_UserID),  
  CONSTRAINT AS_Priority_Out_Of_Range CHECK ((Priority >= 0) OR (Priority <= 2));
```


Functional Dependencies:

F_assistsw = { Support_UserID Well_UserID -> Since Priority }

3.1.11 AssistsS**Constraints:**

Foreign key UserID in this table can not be null. And if any record in table Supporter and table SickPatient is deleted, the related record in this table must be deleted as well. We use CHECK to ensure two constraints, one is that every sick patient has at least one and at most two supporters. Another is one can not be the supporter of self.

SQL:

```
CREATE TABLE AssistsS (  
P_UserID VARCHAR2(20) NOT NULL,  
S_UserID VARCHAR2(20) NOT NULL,  
Since DATE NOT NULL,  
Priority NUMBER(1) NOT NULL,  
CONSTRAINT AS_PUID_Prioy_NOT_Unique PRIMARY KEY (P_UserID, Priority),  
FOREIGN KEY (S_UserID) REFERENCES Supporter ON DELETE CASCADE,  
FOREIGN KEY (P_UserID) REFERENCES Patient ON DELETE CASCADE,  
CONSTRAINT AS_PUID_SUID_NOT_Unique UNIQUE (P_UserID, S_UserID),  
CONSTRAINT AS_PUID_SUID_Should_diff CHECK (P_UserID <> S_UserID),  
CONSTRAINT AS_Priority_Out_Of_Range CHECK ((Priority >= 1) OR (Priority <= 2));
```

Functional Dependencies:

F_assistss = { Support_UserID Sick_UserID -> Since Priority }

3.1.12 PatientDisease**Constraints:**

Foreign key UserID and Did in this table can not be null. And if any record in table SickPatient and table Disease_DRecomm is deleted, the related record in this table must be deleted as well.

SQL:

```
CREATE TABLE PatientDisease (  
UserID VARCHAR2(20) NOT NULL,
```

Did VARCHAR2(20) NOT NULL,
Since DATE NOT NULL,
CONSTRAINT PD_Did_UserId_NOT_Unique PRIMARY KEY (UserID, Did),
FOREIGN KEY (UserID) REFERENCES Patient ON DELETE CASCADE,
FOREIGN KEY (Did) REFERENCES Disease_DRecomm ON DELETE CASCADE);

Functional Dependencies: $F_{\text{PatientDisease}} = \{ \phi \}$

3.1.13 DoctorRecomm

Constraints:

Foreign key UserID and Rid in this table can not be null. And if any record in table Patient and table Recommendation is deleted, the related record in this table must be deleted as well. Every user should have at most one specific recommendation, so UserID and Rid pair in this table should be unique.

SQL:

```
CREATE TABLE DoctorRecomm (  
  UserID VARCHAR2(20) NOT NULL,  
  Rid NUMBER(3) NOT NULL,  
  CONSTRAINT DR_User_Only_Has_One_DoctorRe UNIQUE (UserID),  
  CONSTRAINT DR_Did_UserId_NOT_Unique PRIMARY KEY (UserID, Rid),  
  FOREIGN KEY (UserID) REFERENCES Patient ON DELETE CASCADE,  
  FOREIGN KEY (Rid) REFERENCES Recommendation ON DELETE CASCADE);
```

Functional Dependencies: $F_{\text{doctorrecomm}} = \{ \phi \}$

4 Constraints Implemented

4.1 Implemented in Database

4.1.1 Generate Alerts

Outside-the-limit alert when a certain threshold percentage number of consecutive readings are over the specified limits for the patient. For example, if more than 70% Each record of relation "Alert" keeps two attributes: "read" and "active". A user can only deactivate the alert when she acknowledges that she is informed of the message.

Low-activity alert which helps to identify patients that seem to be disengaged from the system. For example, the situation can be when the recommended frequency of an observation type is X and patients haven't recorded any activity by certain threshold beyond X. Both patients and Health Supporters can view these alerts. Such alerts are cleared in one of two ways: Either the Health Supporter clears them (essentially representing the fact that they have intervened in some way) or a patient enters an observation for the missing observation type. When a supporter acknowledges the awareness of the alert, the value of "read" is updated. While, a patient can't update the "read" attribute. The alert can always be deactivated by entering a new observation.

4.1.2 Alternate between sick patient and well patient

The default identity of a new patient is well patient. The user can change to sick patient by add disease to diagnose. A trigger helps change a SickOrNot tag when user add or delete a disease from diagnoses (Adding a disease will result in tag plus 1, deleting a disease will result in tag minus 1). So a patient will be treated as sick patient when his/her SickOrNot tag is not zero. On the contrary, SickOrNot tag is 0 indicates a well patient. As a result, the application can automatically change users function view by checking the SickOrNot value.

4.1.3 Insert value within reasonable range

We implement some checks in table creation SQL statement to check if input values are within a proper range. For an example, people's weight can not be negative.

4.1.4 Health supporter access permission

Health supporter can't visit patients' observation records before authorized date. We design an "since" attribute to record this date. Every time a supporter want to see a patient's observation, the "since" attribute must be checked.

4.1.5 Encrypted password

In order to keep password safety in our database, we use MD5 method to encrypt password storage. A function GET_MD5 is used to encrypted password at every insert actions. During log in, input password will also be encrypted to compare with password in database.

4.1.6 Observation record rights

Only available options should be the options for observation types associated with the patient classes that the patient belongs to. This can be achieved by using an available flag to allow inputting certain types of observations.

4.2 Implemented in Application

4.2.1 Sick patient total participation constraint

A sick patient must have one or more diseases and supporters. So every record in SickPatient must participate in relationships AssistsS and SickPatientDisease. In DBMS we can hardly check a many to many relationship, so this constraint is implemented in application. Our implementation is that when a user's identity changes to sick patient, he/she has to input one or more disease and supporters. During an edit action, the system will also check the presence of supporter and disease, if either of them is empty, the action will not succeed.

4.2.2 Hierarchy constraint of User, Supporter and Patient

In our design, Supporter and Patient are subclass of Users. This ISA relationship allow overlap between supporters and patients. Which means that one can be both a patient and a supporter. So patient and supporter must be stored in separate tables. Although every user has only one account, they can enter different entries for using different functions. That is how the overlap implemented.

4.2.3 Health supporter priority

Supporters have different priority. For a patient has two supporters, one supporter's priority is primary and the other one is secondary. If the primary supporter is deleted, the secondary supporter will be automatically set to primary. This constraint can hardly be implemented in DBMS because of two reasons. Firstly, in Oracle we can not get information from a modifying table in a trigger. Secondly, we can not query a table in CHECK function. So we have to implement this constraint using SQL with application. For every supporter edit operation, the application will sent a transaction to check if a primary supporter exists.

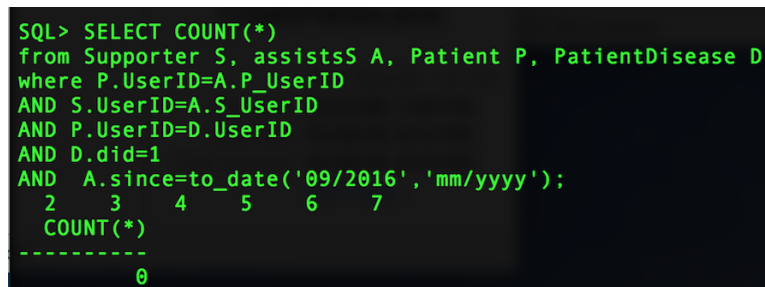
5 SQL Query and Result

This part shows how we implement queries in our database design. The sample data is offered by instructor and teaching assistants.

5.1 List the number of health supporters that were authorized in the month of September 2016 by patients suffering from heart disease.

SQL Query:

```
SELECT COUNT(*)
from Supporter S, assistsS A, Patient P, PatientDisease D
where P.UserID=A.P_UserID
AND S.UserID=A.S_UserID
AND P.UserID=D.UserID
AND D.did=1
AND A.since=to_date('09/2016','mm/yyyy');
```



```
SQL> SELECT COUNT(*)
      from Supporter S, assistsS A, Patient P, PatientDisease D
      where P.UserID=A.P_UserID
      AND S.UserID=A.S_UserID
      AND P.UserID=D.UserID
      AND D.did=1
      AND A.since=to_date('09/2016','mm/yyyy');
      2      3      4      5      6      7
      COUNT(*)
      -----
      0
```

Figure 5.1: Query 1 snapshot

5.2 Give the number of patients who were not complying with the recommended frequency of recording observations.

SQL Query:

```
SELECT count(distinct UserID)
FROM Alert A
WHERE A.Atype = 1;
```

```

SQL> SELECT count(distinct UserID)
      2 FROM Alert A
      3 WHERE A.Atype = 1;

COUNT(DISTINCTUSERID)
-----
1

```

Figure 5.2: Query 2 snapshot

5.3 List the health supporters who themselves are patients.

SQL Query:

```

SELECT S_UserID FROM AssistsS, Patient
WHERE S_UserID=UserID AND sickornot=1;

```

```

SQL> SELECT S_UserID FROM AssistsS, Patient
WHERE S_UserID=UserID AND sickornot=1;
      2
S_USERID
-----
P2

```

Figure 5.3: Query 3 snapshot

5.4 List the patients who are not sick.

SQL Query:

```

SELECT UserID from Patient WHERE sickornot=0;

```

```

SQL> SQL Query:
SELECT UserID from Patient WHERE sickornot=0;
SP2-0042: unknown command "SQL Query:" - rest of line ignored.
SQL>
USERID
-----
P3
P4

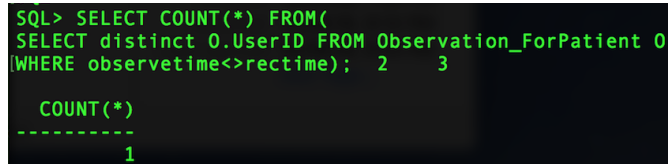
```

Figure 5.4: Query 4 snapshot

5.5 How many patients have different observation time and recording time (of the observation).

SQL Query:

```
SELECT COUNT(*) FROM(
SELECT distinct O.UserID FROM Observation_ForPatient O
WHERE observetime<>rectime);
```



```
SQL> SELECT COUNT(*) FROM(
SELECT distinct O.UserID FROM Observation_ForPatient O
WHERE observetime<>rectime); 2    3

COUNT(*)
-----
1
```

Figure 5.5: Query 5 snapshot

6 Files and source code

The application and database system includes four JAVA files:

CreateTables.java	//Create schemas in database
InsertValues.java	//Initialize records in each schema
QueriesV2.java	//All JDBC queries to database
Query.java	//A query class for each returned query result
Connect.java	//A connect class for transitions
Menu.java	//A menu class for show menus
InitModule.java	//A class to init menus and connects
WolfHealth.java	//Main application driver class

7 Readme

7.1 Members

Table 7.1 lists group members and responsibilities.

Name	UnityID	Work
Mengying Wang	mwang22	database design, query design, sample data design
Ran tan	rtan2	database design, CMD UI, framework design
Shuai Chen	schen35	database design,query, transaction and constraints design,testing
Jingjuan Deng	jdeng8	database design, query, trigger and SQL function design

Table 7.1: Members and responsibilities

7.2 How to run

The application has a command line interface based on JAVA. Before running it, all the five java files need to be compiled: `javac QueriesV2.java javac Query.java javac Connect.java javac Menu.java javac InitModule.java javac WolfHealth.java`

After compiling them, the two files to initiate database schemas and records should be executed at first in terminal: `java CreateTables java InsertValues` After that, the application interface file can be executed for test: `java classDriver` Then the interface will be showed in the terminal. First it shows a login/sign up view, in witch you can create an account or sign in an existing account:

```
=====
Start Page:
Login or Sign up a new account to get access to our services!
-----
0. Back
1. LogIn
2. SignUp
3. Exit
Please enter your choice here: 2
=====
```

Figure 7.1: Application login view

After logging in, you can choose to act as a patient or a supporter. Different roles have different responsibilities.

As a patient, you can use functions listed below:


```
-----
0. Back
1. As a Patient
2. As a Supporter
3. Exit
Please enter your choice here: █
```

Figure 7.2: Choosing role

```
=====
Well Patient Page:
Keep Healthy and Strong!
-----
0. Back
1. Show My Profile
2. Edit My Profile
3. Show My Health Supporters
4. Edit My Health Supporters
5. See Recommendations for Me
6. Update My Recommendations
7. Show My Observations
8. Type in New Observations
9. Show Active Alerts
10. Acknowledge and Deactivate Alerts
11. Exit
Please enter your choice here: █
```

Figure 7.3: Patient view

As a supporter, you can use functions listed below:

```
=====
Supporter Page:
Be Responsible as a Health Supporter!
-----
0. Back
1. Show My Profile
2. Edit My Profile
3. Show Patients under My Support
4. Exit
Please enter your choice here: █
```

Figure 7.4: Supporter view

Enjoy our system!