Final Report

CompSci 316 Fall 2015
Amanda Duffy, Guan-Wun Hao, Jiahao Pei and Chen Yang

Project Description

People who need to take their medications on a regular basis forget about it from time to time. They may not remember to take the medicine, but there is a lower chance for them to forget their cell phones at the same time. Our goal in this course project is to develop a reminder application which runs on an Android phone and notifies the users the need of medicines on time. It includes the following features:

- A simple UI that allows the user to enter medication information, including name, dose, and frequency to take.
- General information about the drugs: name, medical description, and side effects (toxicity).
- Warnings of drugs' interactions/conflicts.
- Pharmacy information: load the names, phone numbers and addresses of local pharmacies into the database, which makes the refilling of medicine much easier.

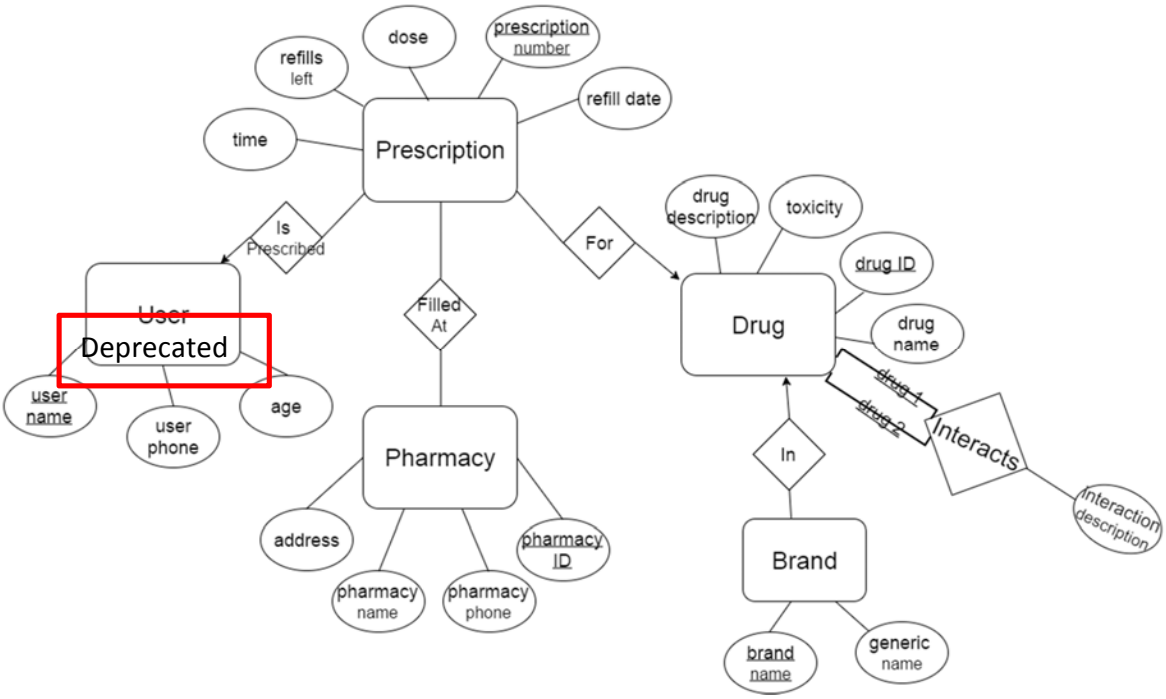
This application would be useful for those people who have to manage several medications. It could be a simple way to help users stay healthier.

We found some existing applications that have similar functions, but none of them can accomplish everything we proposed with our app.

	Remind To Take Medicine	**Remind To Refill**	**Drug Database**	**Prescription Info**	**Contact Pharmacy**	**Checklist**
Pill Reminder	Y	N	N	N	N	N
Dosecast	Y	Y	Y	N	N	N
Medisafe	Y	Y	N	N	N	N
Medicine Time!	Y	N	N	N	N	N
Med Helper	Y	Y	N	N	Y	Y

For ease of development, we made the Android version of this app only. All the drugs information was downloaded from drugs.com database and then parsed into our format.

System Design



E/R Model

(all pointers in the diagram means many-to-one relationship)

1. The user has unique user name (the name which he/she registers in the app). Users can choose to link the account with their Google/Facebook account, instead of creating a new one. We also request phone number and age from each user. Eventually, we did not have enough time to figure out a way to use a third-party account to log into our app, so we decided to make each user to keep his/her own reminder data locally.
2. Users are given a prescription for a specific drug. A user can have multiple prescriptions. Prescription holds the personal information associated with the drug and specific user. The prescription number is unique for the specific user using the specific type of drug. We also add refill date and refills left (how many refills remain for the given prescription before a doctor needs to prescribe more) as Prescription attributes to remind the user. Moreover, "dose" and "time" save more detailed information of how much dose to take at what particular time, for example, 9AM every morning.
3. Drugs are stored as generic information which are not associated with personal information. A drug can have multiple prescriptions for different users but a prescription is for a generic drug. Each drug has a unique ID and drug name. We store drug's ID as primary key and set the name as unique. The intention is to use primary key, drug ID, in other tables when we want to access drug information without the need to extract its name. However, because we want to specify different brand names for a specific

generic drug and for clarification of interactions between two drugs, we create two tables to store the information and access them using drug's unique key, drug name, without duplicate drug IDs. In addition, we have each drug's description and toxicity stored inside the table.

4. Interaction table specifies the interaction between two drugs. The two drug names inside the table are from drug's table and we put additional attribute of interaction description to store the information. The E/R diagram above shows the multiplicity of the relationship. We can translate it as "drug1 interacts with drug2".
5. As mentioned earlier in drug's table, brand table stores the drug information of name difference existing in the market. The table is not used when we only want to get drug's information, like toxicity or description. But this table is useful as a dictionary for looking up the various brand names in a generic drug.
6. Prescriptions are filled at pharmacies and each pharmacy can fill many drug prescriptions. Pharmacies may have the same name but we specify that their Pharmacy ID as unique. We also request every pharmacy's phone number and address.
7. Since a lot of medications from different manufacturers or with different amount in the bottles differ with each other but share the same effective constituent will have the exactly the same drug information in Drug table. By introducing the DrugBrand table, we removed the duplicated drug in formation and successfully reduced the number of tuples in Drug table by a huge factor.

E/R Model Translation

(There is a slight naming difference between the translation and the simplified-name E/R model, for the purpose of readability in the diagram. But the name in the translation is the same as in CreateTable.sql)

[Deprecated]Users(<u>userName</u>, userPhone, age) // deprecated
Prescription(<u>prescriptionNumber</u>, dose, time, refillDate, refillsLeft)
Pharmacy(<u>pharmacyID</u>, pharmacyName, pharmacyPhone, address)
Drugs(<u>drugID</u>, drugName, drugDescription, toxicity)
Interaction(<u>drugName</u>, <u>Interaction_drugName</u>, interactionDescription)
DrugBrand(<u>brandName</u>, genericName)
PrescriptionIsPrescribedbyUser(<u>prescriptionNumber</u>, userName)
PrescriptionForDrug(<u>prescriptionNumber</u>, drugID)
PrescriptionFilledAtPharmacy(<u>prescriptionNumber</u>, <u>pharmacyID</u>)
DrugInBrand(drugID, drugName, <u>brandName</u>)

(Further modification:

1. merge *PrescriptionIsPrescribedbyUser* and *PrescriptionForDrug* and *PrescriptionFilledAtPharmacy* into *Prescription*
2. merge *DrugInBrand* into *DrugBrand*

)
[Deprecated]Users(<u>userName</u>, userPhone, age)

Prescription(prescriptionNumber, userName, drugID, pharmacyID, dose, time, refillDate, refillsLeft)
(Although there would be duplicates in prescriptionNumber and pharmacyID because they are multiple-to-multiple relationship, we still put them in one table to simplify the relationship, i.e. get rid of *PrescriptionFilledAtPharmacy* table)
Pharmacy(pharmacyID, pharmacyName, pharmacyPhone, address)
Drugs(drugID, drugName, drugDescription, toxicity)
Interaction(drugName, Interaction_drugName, interactionDescription)
DrugBrand(brandName, genericName)
(Because drugName is unique, genericName can reference drugName. Also, *DrugBrand* table has the information for the relationship between *Drug* and *Brand*, we thus drop *DrugInBrand* table)
----------All relations are in BCNF----------

Design Choices

From Milestone 1 to Milestone 2, we put interaction and toxicity information in one table. The design didn't satisfy BCNF, because if we list all the interaction mapping pairs of drugs, there will be a lot of duplicates about the drugs' toxicity description. Thus, we changed our design in the second milestone to put them in separate table as weak entity sets.

From Milestone 2 to our final design version, we have made several big adjustments. Most changes are related to drug table and partly because of data parsing process. The drug information which we retrieved has drug descriptions and side effects within one database while interaction between drugs are stored in another database. Therefore, it would be quite inefficient and meaningless to separate *Drug* table, with only one attribute drugID, and leave *Toxicity* in another table as weak entity set to *Drug*. Since toxicity and drug description contain basic information about a certain drug, we combine them in one table (Drug table) to describe each drug.

In *Drug* table, we put drug names along with drug IDs. It is thus more readable and more convenient for connecting with other tables, which are *Prescription*, *Interaction* and *DrugBrand*. *Interaction* table is no longer a weak entity set but rather reference the unique key(drug name) from *Drug* table and only describe two drugs' interaction. Originally, we would like to reference drug's primary key, which is the ID. But using name in the table helps us better understand which drug has interaction with another one.

Also, we decided to remove the Users table out of a consideration of safety and better user experience. Storing the user drug taking information on server/cloud has a potential risk of privacy leakage; on the other hand, storing the data locally will allow the users to use the app offline.

Moreover, *DrugBrand* table is introduced in our final design. The consideration is based on the fact that there are many existing brand names associated with the same generic drug in the market. Storing the comparison information can help us identify each drug prescription, especially for refilling drugs in different pharmacies. For the similar concern with designing *Interaction* table, we can compare the name difference directly by referencing unique key(drug name) from *Drug* table. In this sense, there is no need to search for the drug's name associated with the id if we were to use id's instead.

Evaluation

The application reached the basic goal of reminding users of taking medicine on a regular time basis. The use of it is easy enough for most people. One major issue currently is the smoothness of the application running depends a lot on the device it is installed on. It performs much better on a newer and faster machine. Since the functions it is realizing are not very complicated, we probably need to evaluate the code quality and make optimization.

At the first time of installing and opening the application, it will load the database into the software. Thus it takes a little while for the loading process, but saves the time for future use. If it fetches data from an online server every time, it will take less time at first loading but more later. We need to conduct some experiments comparing these two approaches on performances later.

Open Issues and Future Work

- The drugs' information is downloaded from an online medicine database and then loaded into our application. It is not happening automatically. As a result, every time the online database makes an update, we will have to re-download and import it into the app. We need to find a method that allows to link our app to the online database and read directly from it.
- Server end account would be a good start for future improvements of the application. First, it would be a backup of the app data in case the system/app crashes. Second, it would make it possible for us to generate health report based on the medicines a user takes and misses and thus offer fitness suggestions. Third, an extension function can be implemented to establish a connection between the patient and his/her doctor.
- Scheduled reminders for doctor appointments.
- Daily/weekly/monthly checklist, with summary and alerts of medicine taking and missing.
- Add a Substitution table. Based on the interaction table, if two medicine conflict with each other, possible substitutes will be suggested and sent to doctors for evaluation.
- Add a feature for the elder patients, with "read-out-loud" reminder, besides the simple alarming.
- One possible reason for the app to be running slow is that we loaded the xml into the software as SQL tables rather than using it directly. This was because it was easier to

implement when we first tried it. We want to try to read and process the xml data directly.

- Due to lack of time in the end, we did not get the Pharmacy table working in our proposed way. In the future, texting/calling to the pharmacies from the app to get them ready the medication to be refilled would be good features to add.