香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

CSC3170
Database System

Group Project Report
Hospital Information System

Li Huayue 118010138

Fang Xinyu 118010061

Shi Guodong 118010257

Feng Yuhao 118010068

Li Longxiang 118010144

May 1, 2021

Contents

1. Introduction

1.1 Background

Hospitals plays an important role in people's daily life. It not only provides patients with treatment services but also stores the records for further usages. These usages include inventory checking, accounting auditing, anamnesis analyzing and others. To meet the mounting requirements of hospitals, a well-designed database is crucial. (Griffith, J.R. & White, K. R., 2005) Beyond its function for treatment services and records, the database also should perform efficient employee management. On one hand, hospitals need to store employee information which includes his or her salary to determine the payload. On the other hand, considering the treatment record of anamnesis, it is necessary to find which doctor offers the treatment and which department should take responsibility for it.

1.2 Motivation

Basically, a database needs to realize functions of insertion, deletion, selection, and updating. All tables in the database including patient, doctor, treatment record, medicine and equipment should be applied to these four basic types of queries.

Various database systems have been customized for hospital management, but most of these systems are isolated instead of connected. We designed a well-connected hospital management system, which can transfer the queries directly to other parts of the system.

In further application, the efficient method of data collecting is the basis of machine learning and bigdata-based application. We need to extract attributes from different tables and check their influences on a certain goal. Once the database is connected, people can easily use SQL queries to get data they want from different parts of the hospital management system.

2. Design and Implementation

2.1 E-R Diagram

There are 9 entities and 8 relationships in total in our E-R diagram. In every entity, there are some attributes to represent the information of corresponding object. Some entities have something to do with adjacent entities. According to the actual requirements of the hospital system, there are some constraints on attributes and relationships. We will talk about all the relevant entities, attributes, and relationships together with some constraints and properties.

Every hospital should purchase and stock medicine. We assume that we need to know the date, amount, and price when the hospital purchases medicine, so the attributes of the medicine_purchase entity include the date, purchase_amount and purchase_price. However, we cannot identify every entity of medicine purchase using these attributes, which means the medicine_purchase entity is a weak entity. The identifying entity of it is medicine_stock. The medicine_stock entity records all the information of medicine of the hospital system including medicine ids, medicine names, suppliers and stock. Therefore, the attributes of medicine_stock are medicine_id (PK), medicine_name, supplier, and stock. We use the medicine_id as the discriminator attribute to uniquely identify the weak entity, medicine_purchase. We assume that all the medicine purchased must be stocked but the medicine stocked may get from other ways like donating instead of purchasing.

Based on this constraint and the property of weak entity, the relationship between the medicine_purchase and medicine_stock is "replenish" and the relationship is total-to-partial and many-to-one.

When doctors give treatment to patients, the record of the treatment will be saved. Therefore, we create an entity called treatment_record. The attributes of it are treat_id (PK), doctor_id (FK, towards the doctor entity), patient_id (FK, towards the patient entity), date and symptom. These attributes include all the information needed for treatments. Some treatments may need to consume many kinds of medicine but some may not use any medicine. And not all the medicine in the stock will already be used in the treatment and many medicines are used in many treatments. Therefore, the relationship between the medicine_stock and treatment_record is "consume" and the relationship is many-to-many and partial-to-partial. What's more, there must exist sick beds in every hospital. To manage these sick beds, we create an entity called sick_bed. The attributes of it include bed_id (PK), type (normal or ICU), status (available or occupied), treat_id (FK, towards the treatment_record entity), and days_in_bed (length of stay). Some treatments offered by doctors suggest patients be hospitalized, which means one sick bed will be used for that treatment. Not all the sick beds have already been used and not all the treatments will use the sick beds, so the relationship between the sick_bed and treatment_record is "use" and the relationship is one-to-one and partial-to-partial.

It is obvious that there exist patients and doctors in every hospital. We create two entities: patient and doctor to manage the information of them. The attributes of doctor entities are doctor_id (PK), name, dep_name (FK towards department entity), gender, type (nurse or doctor), phone, and salary. The attributes of patient entities are patient_id (PK), patient_name, phone, gender, has_medi_insurance (whether the patient has medical insurance or not). These two entities are connected by using the treatment_record entity. Treatments must be offered by doctors and some doctors have already offered a lot of treatments. However, some doctors may not offer any treatment for some reasons like they just come to the hospital. Therefore, the relationship between the treatment_record entity and doctor entity is "record" and the relationship is many-to-one and total-to-partial. For patients, all the treatments must be offered to patients. Some patients may take treatments many times and some patients may not agree to take any treatment. Therefore, the relationship between the treatment_record and the patient is "take" and the relationship is many-to-one and total-to-partial.

There are many kinds of equipment in every hospital. Therefore, we create an entity called equipment. The attributes of the equipment entity are equip_id (PK), name, purchase_date, cost, and last_check_date. As you can see, some treatments may use the equipment. Some equipment may be used in many treatments. And there still exist some treatments which do not need to use any equipment. And some equipment is not already be used in any treatment. Therefore, the relationship between the equipment and treatment_record is "equip_use" and the relationship is many-to-many and partial-to-partial.

All the doctors must belong to some departments in the hospital. Therefore, we create an entity called department. The attributes of it are name (PK) and dep_phone. There are a lot of doctors in many departments. And there are some departments where no doctors exist. Therefore, the relationship between the department and the doctor is "belong" and the relationship is one-to-many and partial-to-total.

Last but not least, there is a special entity called finance to store the financial information of the hospital. Because it has something to do with a lot of entities but we want to make the E-R diagram of our system clear and concise, we make this entity alone. The attributes of it are month (PK), income, expense, and profit.

These are all the contents of our ER diagram and analysis of relevant entities, attributes, and relationships together with some constraints and properties.

The ER diagram of our project is shown as follows:

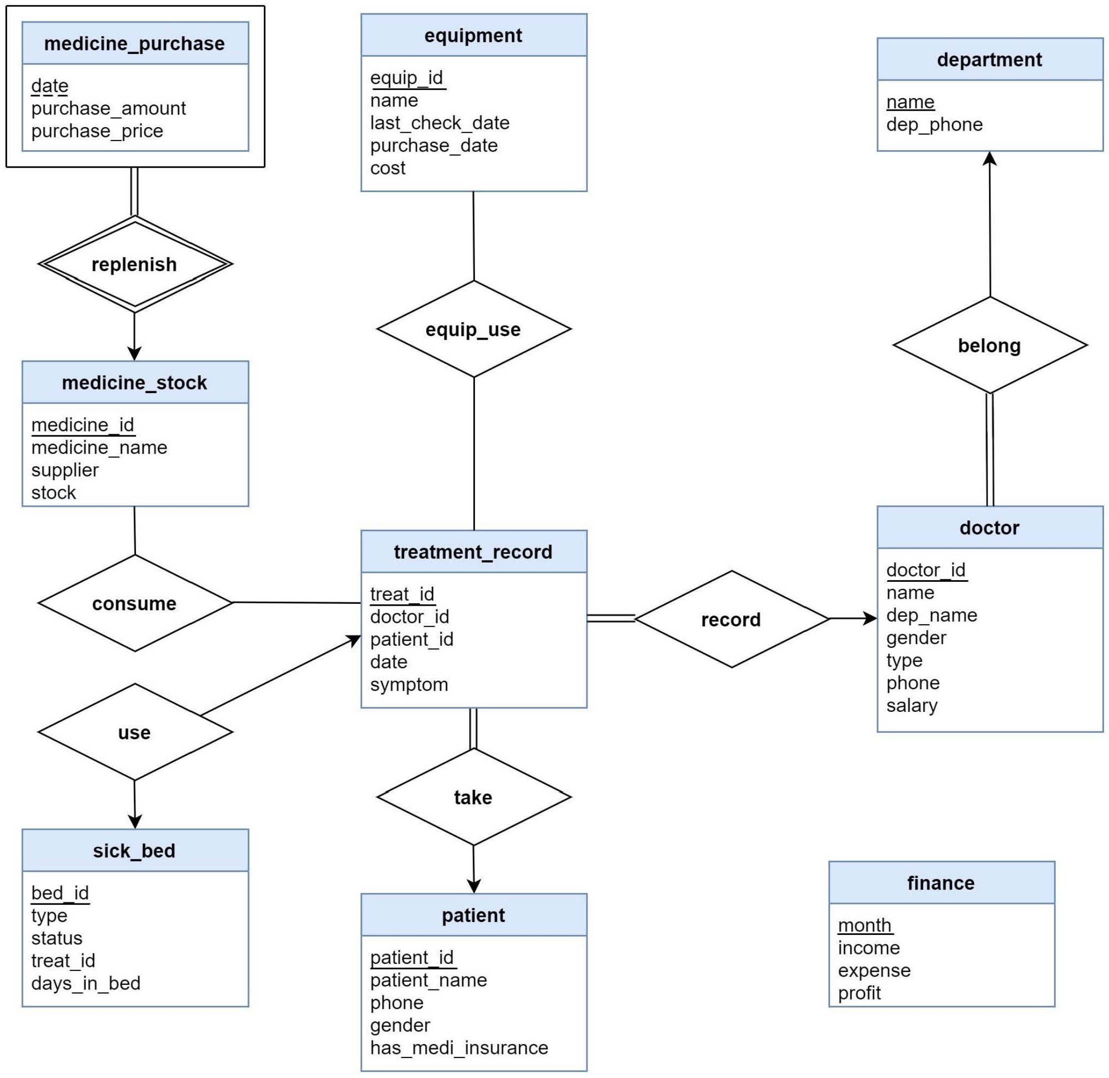


Figure 1 E-R Diagram

2.2 Relation Schema

patient(patient_id, patient_name, phone, gender, has_medi_insurance)		
PK: patient_id	FK: Null	3NF

doctor(doctor_id, name, dep_name, gender, type, phone, salary)		
PK: doctor_id	FK: dep_name	3NF

department(dep_name, dep_phone)		
PK: dep_name	FK: Null	3NF

treatment_record(treat_id, doctor_id, patient_id, date, symptom)		
PK: treat_id	FK: doctor_id, patient_id	3NF

equip_use(treat_id, program_id, equip_id, program_cost)		
PK: treat_id, program_id	FK: treat_id, equip_id	3NF

Transformed from the intermediate relation between treatment_record and equipment.

equipment(equip_id, name, last_check_date, purchase_date, cost)		
PK: equip_id	FK: Null	3NF

medicine_consumption(treat_id, medicine_id, consumption_amount, sell_price)		
PK: treat_id, medicine_id	FK: treat_id, medicine_id	3NF

medicine_stock(medicine_id, name, supplier, stock)		
PK: medicine_id	FK: Null	3NF

medicine_purchase(date, purchase_amount, purchase_price)		
PK: date	FK: Null	3NF

sickbed(bed_id, type, status, treat_id, days_in_bed)		
PK: bed_id	FK: treat_id	2NF

Transitive dependency: treat_id → days_in_bed This table is designed in 2NF for simplicity, because one treat_id only appears once in this table.

finance(month, income, expense, profit)		
PK: month	FK: Null	3NF

2.3 Data

Since the data fields of our database are specifically designed, it is impossible to find real data that completely meet our requirement. So, we randomly generate some data and combined it with real data download from UCI to fill our tables. Tables like doctors are generated by Python code, because they are related to simple SQL queries that do not need to be further applied. Since we plan to perform some advanced machine learning based predictions, we need to apply complex SQL queries that provide analytic functions of selecting data. Thus, the information of medicine and some patient records are collected from UCI, which helps us perform the prediction.

2.4 Sample MySQL queries

2.4.1 Graphic User Interface

Reflecting the aim of the project is to implement a hospital management system, the next step is to simulate a graphical user interface utilizing the well-designed database with data loaded. The interface is mainly presented by Bokeh, which is a Python library for creating JavaScript-powered interactive visualizations for modern web browsers. This approach neatly sidesteps the heavy front-end framework of Html, CSS, and JavaScript. The back end is based on Python and is connected to our cloud database with PyMySQL.

The web page of our hospital management system contains 6 tabs. Each tab corresponds to one main table in the database and contains several functions which are possibly used in practical. Besides information delivery and presentation, one function is equivalent to execute one or a set of MySQL select, update, insert or delete queries. In the next section, the functions in each tab and their corresponding MySQL queries will be introduced.

2.4.2 Sample Queries

There are 6 tabs on the web page, which are Doctor Management, Patient Management, Outpatient, Bed Management, Medicine Management, and Equipment Management, respectively. The sample output of all the functions with implementation details presented will be shown in the appendix.

Doctor Management

(i) Query Doctor Info Users can search the information of doctors by name or department. The input can be a prefix or suffix.

Input type	Sample input	MySQL query statement
Part of doctor name	“Charles”	Select * from doctor where name = “%Charles%”
Department name	“Burns”	Select * from doctor where dep_name = “Burns”

(ii) Doctor Entry To insert a doctor into the database, the user can input basic information about the doctor. The doctor id is valued as the current maximum id plus one.

input(docname=”Charles”, dep_name=”Burns”, gender=”male”, type=”doctor”, phonenum=”13970631951”, salary=”10000”)

Declare curmax_id int;
select max(doctor_id) into curmax_id from doctor;
insert into doctor values (curmax_id+1, "Charles", "Burns", "male", "doctor", "13970631951", 10000);

(iii) Doctor Dismission User can implement doctor dismission by deleting the information

Input (docname="Seaman")	Delete from doctor where name="Seaman"

(iv)Doctor Update The update operation can be subdivided into two parts. First, query doctors' information by the doctor's name and present the information in the corresponding input boxes. After the user makes some modifications, the system updates the information in the database.

Part1	Input(docname="Seaman")
	Select * from doctor where name="Seaman";
Part2	Input(dep_name, gender, type, phone,salary)

	update doctor set name="Seaman", dep_name="Burns",
	gender="male", type="nurse", phone="13835959782",
	salary=9000 where name="Seaman"

Outpatient

This part is related to multiple tables including treatment_record, medicine_consumption, program, and medicine.

(i) Latest Treatment User can search by patient_id. The underlying logic will search the treatment record with the largest date, and present all the treatment programs and consumed medicines of that treatment record.

```
Input (patient_id = 1008621214)
select treat_id from treatment_record where patient_id = 1008621214 order by
date desc limit 0,1;
select * from program where treat_id = 1849794466;
select * from medicine_consumption where treat_id=1849794466;
```

(ii) Add program Given the ID of a treatment record, the ID of the equipment used and the program cost, the system will save the record into the table.

```
Input (treat_id=1849794466, equip_id=19388, cost=200)
Declare i curmax_pid nt;
select max(program_id) into curmax_pid from program;
insert into program values (1849794466,curmax_pid+1,19388,200);
```

(iii) Add Medicine Given the id of a treatment record, the medicine ID, and the amount of the medicine consumed, the system will save the record into the medicine_consumption table. The system will decrease the stock value in the medicine table by the amount value at the same time.

```
Input (treat_id=1849794466, medicine_id=100178, amount=5)
declare curstock int;
select stock into curstock from medicine where medicine_id=100178;
insert into medicine_consumption values (1849794466,100178,5,165);
update medicine set stock=(curstock-5) where medicine_id=100178;
```

(iv) Outpatient history record Given the patient ID, the system will output the result of all his or her historical treatment records.

Bed Management

It is a tab regarding the sickbed entity in the database. It is used when a recovered patient checks out. Given the bed ID, the management system will locate the sickbed and check whether it is occupied. If so, the system updates the status to "available" and sets the hospitalization time to zero.

```
Input (bed_id=10055)
select count(*) from sickbed where bed_id=10055 and status="occupied";
update sickbed set status="available", days_in_bed=0 where bed_id=10055;
```

Medicine Management

The tab is about the medicine entity in the database. Its functions are equivalent to simple CRUD operations in MySQL queries so the implementation details are skipped.

Equipment Management

This tab is about the equipment entity in the database. The functions Equipment Search and Add Equipment are select and insert queries respectively in MySQL. The function Equipment Maintenance is a variant of update queries. Given the equipment ID, the system will update its last maintenance date to the current date.

Input (equip_id = 10003)
update equipment set last_check_date=curdate() where equip_id=10003;

2.5 Data Analysis

Data stored in the hospital database can be extracted and analyzed to acquire more information. In this section, relationships between attributes will be visualized and prediction on hospital readmission for diabetes will be done using three machine learning models.

2.5.1 Data Preparation

The source of data is UCI diabetes 130-US hospitals for the years 1999-2008 data set. We customized the patient entity, adding extra attributes to it so that the original dataset can be completely loaded into the hospital database. Then, the following MySQL command is used to select the data we need.

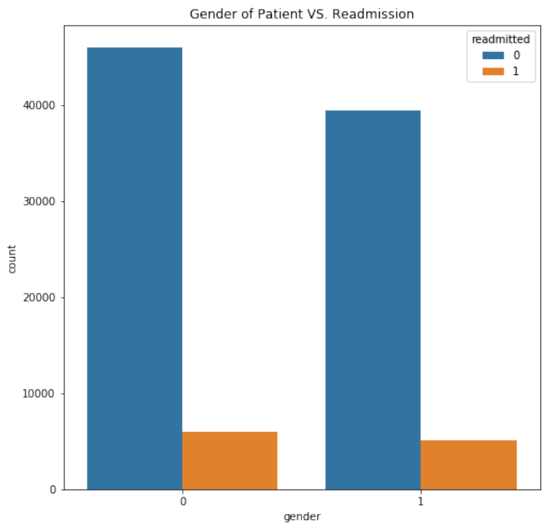
*select * from patient inner join treatment_record on treatment_record. patient_id = patient. patient_id;*

We dropped variables like weight, payer code, and medical specialty because they contain more than 40% missing values. Variables (drugs named citoglipton and examide) all contain the same value are also dropped since essentially they can not provide any interpretive or discriminatory information for predicting readmission. Moreover, columns like patient_id, treat_id, and doctor_id are dropped because they are not related to the readmission.

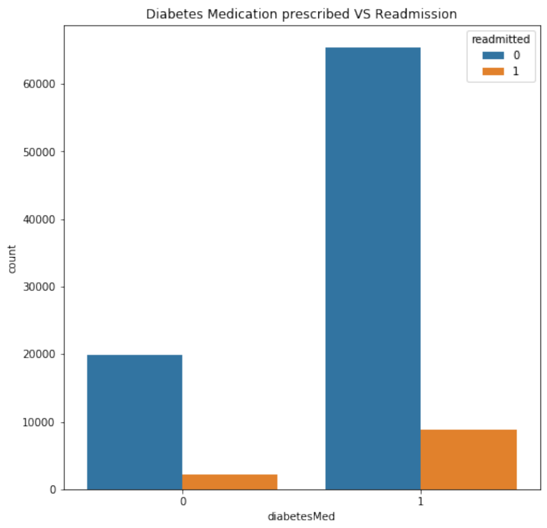
The original dataset used string values for gender, race, medication change, and each of the 23 drugs used. Therefore, we use a label-encoder to better fit those variables into our models. For example, we encode the gender from "female" and "male" into "0" and "1". Finally, we shuffle the dataset, then split the data, using 50000 data for training and 20000 for testing.

2.5.2 Data Visualization

The relationship between the three features and readmission is visualized below.



(a) Gender and Readmission



(b) Prescription and Readmission

Figure 2

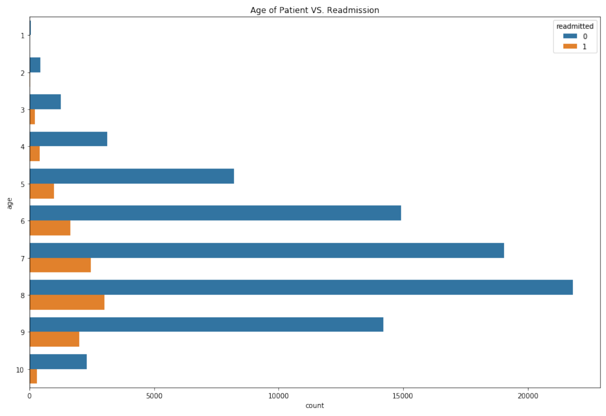


Figure 3 Age and Readmission

As can be seen from Figure 3, the age distribution of diabetes cases basically conforms to the normal distribution.

2.5.3 Modeling

We imported three models from scikit-learn to do the prediction, which are logistic regression, decision tree classifier, and random forest. The accuracy, precision, and recall are shown in the following figure.

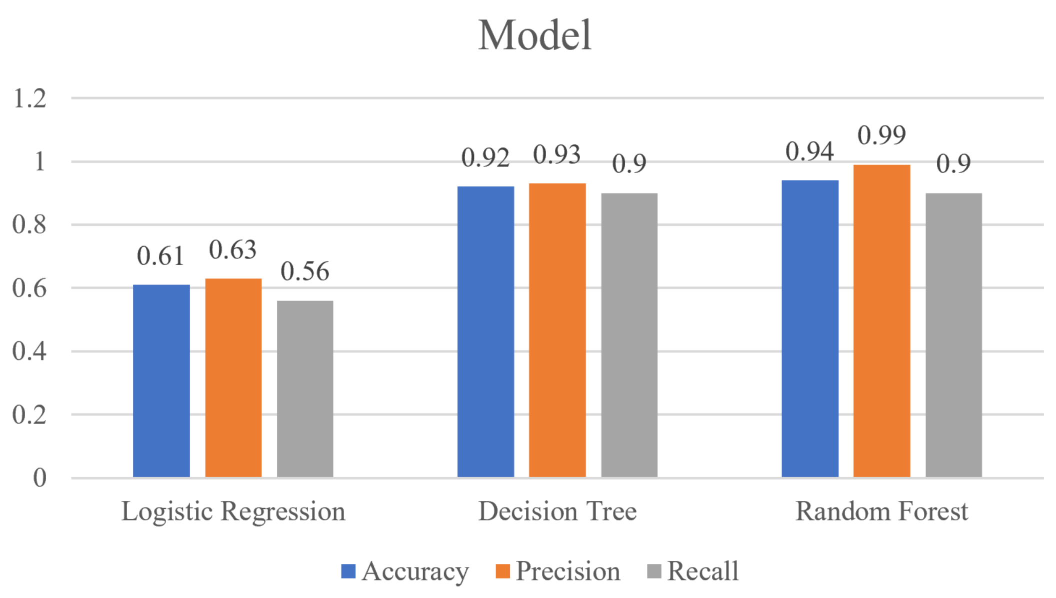


Figure 4

The random forest model gives the best performance and the accuracy reached 94%. For the random forest model, we use Grid Search to adjust n_estimators and max_features, searching through the parameter space to find the best combination. Meanwhile, the hyper-parameter cv (cross_validation) is set to 5 to avoid over-fitting and under-fitting.

With the model we trained and data in the hospital database, doctors can better predict how well a patient recovers after treatment to provide better health care services.

2.6 Index Data Fields

Before discussing the option of indexing or hashing, some general knowledge should be clarified. By discussing whether to add an index on the columns and what type of index should we add, we are talking about the intrinsic data characteristics for each column. Indexing uses B+ trees to store the data, while hashing uses hash tables to store the data. Hence, the comparison is namely the comparison between the B+ tree and the hash table. Their storage structure and ways of data retrieval are taken into consideration when deciding what to use in the tables.

B+ tree uses a special tree structure to store the entries. Each leave node contains multiple entries and they are places in a sorted way. Moreover, pointers are pointing from one leaf to another. This feature enables the ability to an efficient range search with symbols ¿ ¡ in the SELECT query. Since the data is already sorted, B+ tree is also convenient for sorting, especially for those queries with ORDER BY. Nonetheless, although the B+ tree is efficient in average data retrieval, it is not the most efficient in retrieving data, because it needs to go through several levels of tree to find the data. In most cases,the hash table only requires one I/O operation.

Hash table, on the other hand, is poor for SELECT in range, because the data stored are discrete, and it does not support efficient sorting. However, if the most frequent operation on the column is to retrieve one entry, then hash table is the most efficient. Furthermore, if the column contains many replicated values, then hash collision will be a significant problem, and thus decline the efficiency. For the columns with many repeating data, it is better not to create an index on them.

The detailed implementation of our database is shown below, by applying the above rules.

patient B+ tree: patient_id Hash: patient_name, phone None: gender, has_medi_insurance
doctor B+ tree: doctor_id Hash: name, phone None: dep_name, gender, type, salary
department Hash: dep_name, dep_phone
treatment_record B+ tree: date, doctor_id, patient_id Hash: treat_id, None: symptom
program B+ tree: treat_id None: program_id, equip_id, program_cost
equipment B+ tree: equip_id, name, last_check_date
equipment_purchase B+ tree: equip_id, date None: cost
medicine_consumption B+ tree: treat_id, medicine_id None: consumption_amount, sell_price
medicine B+ tree: medicine_id, supplier Hash: name None: stock
medicine_purchase B+ tree: medicine_id, date None: purchase_amount, purchase_price
sickbed B+ tree: bed_id, treat_id None: type, status, days_in_bed
finance B+ tree: month None: income, expense, profit

3. Conclusion

3.1 Self-evaluation

We have learned how to design the database according to real-world requirements. We create tables ground on the proper choices of normal form. The biggest challenge we encountered is data collection. We find it is hard to get real data that completely fits the design of our database. Therefore, we generated some data and then combined these data with some real-world data downloaded from UC Irvine (UCI) Machine Learning Repository. In practice, we have studied how to manage the database efficiently using MySQL workbench like loading data, setting foreign keys, and creating indexes.

In order to visualize our database, we learned to use the bokeh library of Python to design a Graphical User Interface (GUI) for our database. One problem we have encountered is that using Python to retrieve data from the MySQL database requires the preinstallation of our database in local computers. To avoid repeating the installation process on several computers and to make our collaboration process smooth, we moved our database to Alibaba Cloud, a cloud server, so that everyone can connect to it via the Internet. We also designed API for our database, which facilitated the design phase of the GUI. We implemented a function for each type of query (like SELECT/INSERT). In this way, executing queries in the backend requires no more than calling a function.

In a nutshell, we have gained much practical experience in the process of building the project, which will be beneficial in future works.

3.2 Contribution

Member	Work	Contribution
Li Huayue 118010138	Data collection & E-R diagram analysis	21%
Fang Xinyu 118010061	Backend & Further application	21%
Shi Guodong 118010257	Database implementation	21%
Feng Yuhao 118010068	Frontend & GUI design	21%
Li LongXiang 118010144	API & Server connection	16%

4. References

Griffith, J. R., White, K. R. (2005). *The revolution in hospital management. Journal of Healthcare Management, 50(3).*

Beata Strack, Jonathan P. DeShazo, Chris Gennings, Juan L. Olmo, Sebastian Ventura, Krzysztof J. Cios, and John N. Clore (2014) *"Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records," BioMed Research International, vol. 2014, Article ID 781670, 11 pages.*

5. Appendix

The related files are attached with this report, including GUI SQL samples(corresponding to Part 2.4.2), source code of GUI, and database.