1. Introduction:
   1. Background

Hospital 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 hospital, a well-designed database is crucial.[1] Beyond its function for treatment services and records, the database also should perform efficient employee management. On one hand, hospital 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 of it.

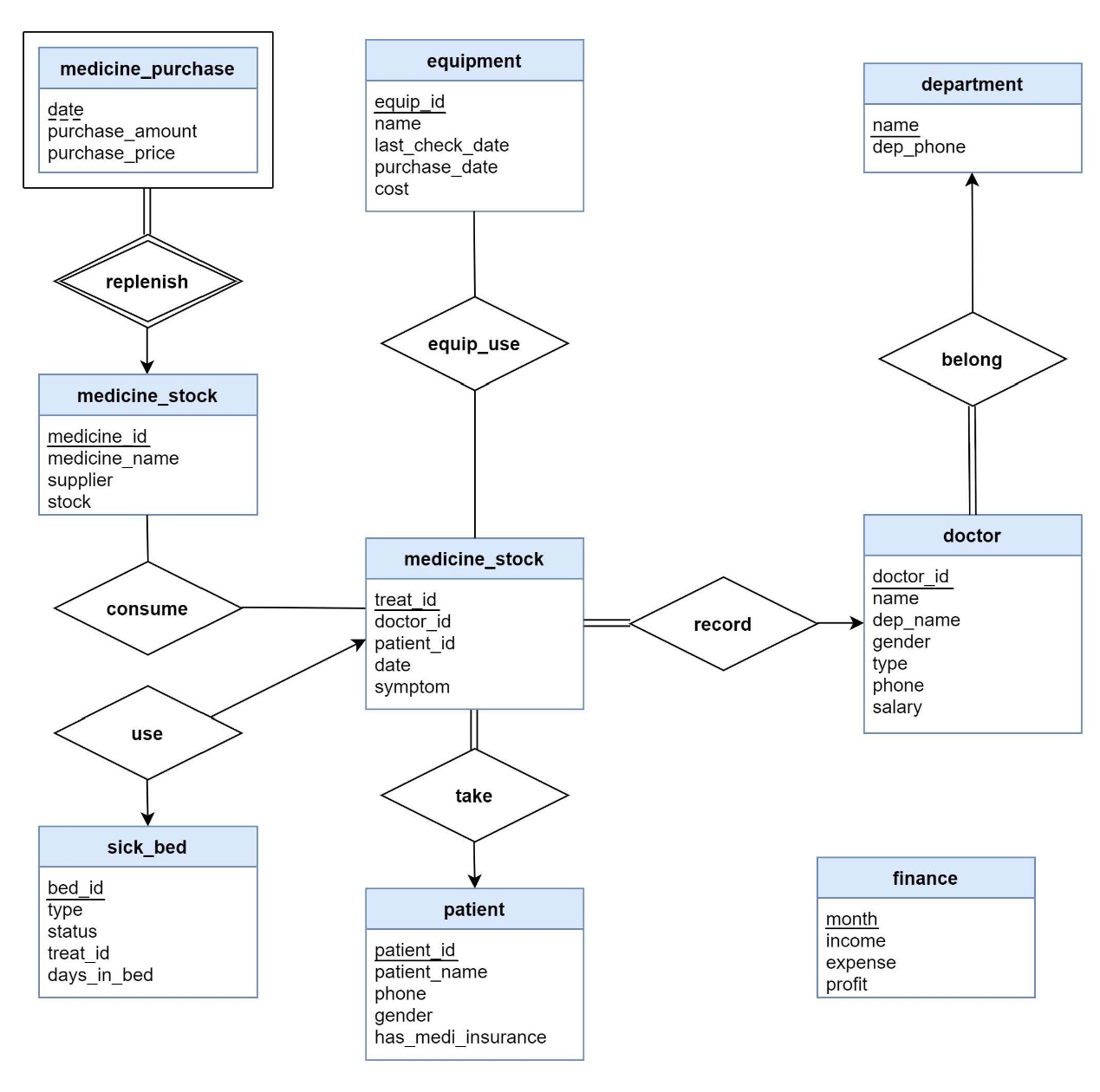
* 1. Motivation

Various database systems have been customized for hospital management. However, most of these systems are isolated instead of connected. For example, medicine management does not keep the treatment record. People can only check the prescription, but can never guarantee what exactly take from the inventory, the kind of medicine and its consumption amount. So, we decide to implement a well-connected hospital management system, which can transfer the queries directly to other parts in 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 the certain goal. Once the database is connected, people can easily use SQL queries to get data they want from different parts of hospital management system.

1. Design and Implementation:
   1. E-R Diagram

The ER diagram of our project is shown as follows:



As you can see, there are 9 entities and 8 relationships in total in our E-R diagram. We will talk about all the relevant entities, attributes and relationships together with some constraints and properties.

First, 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 medicine\_purchase entity include 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.

Second, 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 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 includes bed\_id (PK), type (normal or ICU), status (available or occupied), treat\_id (FK, towards the treatment\_reocrd entity) and days\_in\_bed (length of stay). Some treatments offered by doctors suggest patients to 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.

Third, 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 medicine 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.

Fourth, there are many kinds of equipment in every hospital. Therefore, we create an entity called equipment. The attributes of equipment entity are equip\_id (PK), name, purchase\_date, cost and last\_check\_date. As you can see, some treatments may use 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.

Fifth, 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 finance 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.

* 1. Relation Schema

**patient**(patient\_id, patient\_name, phone, gender, has\_medi\_insurance) 3NF

PK: patient\_id

**doctor**(doctor\_id, name, dep\_name, gender, type, phone, salary) 3NF

PK: doctor\_id

**department**(dep\_name, dep\_phone) 3NF

PK: dep\_name

**treatment\_record**(treat\_id, doctor\_id, patient\_id, date, symptom) 3NF

PK: treat\_id FK: doctor\_id, patient\_id

**program**(treat\_id, program\_id, equip\_id, program\_cost) 3NF

PK: (treat\_id, program\_id) FK: treat\_id, equip\_id

**equipment**(equip\_id, name, last\_check\_date) 3NF

PK: equip\_id

**equipment\_purchase**(equip\_id, date, cost) 3NF

PK: (equip\_id, date)

**medicine\_consumption**(treat\_id, medicine\_id, consumption\_amount, sell\_price) 3NF

PK: (treat\_id, medicine\_id) FK: treat\_id, medicine\_id

**medicine**(medicine\_id, name, supplier, stock) 3NF

PK: medicine\_id

**medicine\_purchase**(medicine\_id, date, purchase\_amount, purchase\_price) 3NF

PK: (medicine\_id, date) FK: medicine\_id

**sickbed**(bed\_id, type, status, treat\_id, days\_in\_bed) 2NF

PK: bed\_id FK: treat\_id 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) 3NF

PK: month

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 with real data download from Kaggle to fill our tables. For some simple SQL queries that just extract something from the tables, there is no need to use real data. However, 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 real data that download from Kaggle would be stored in positions that meet our design. We pick those meaningful data and drop others that does not meet our requirements.

2.4 Sample MySQL queries

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, VSS 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.

There are 6 tabs in the web page, which are Doctor MGT, Patient MGT, Outpatient, Bed MGT, Medicine MGT and Equipment MGT, respectively.

**Doctor MGT**

It is a tab regarding the `patient` entity in the database. The functions on this tab contains the basically CRUD (create, read, update, delete) operations on the doctors’ information.

**Query doctor info**

The input can be either entering **part of** a doctor’s name or selecting a particular department. The output will be the query result table selecting based on the input.

|  |  |  |
| --- | --- | --- |
| 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” |

**Doctor entry**

The input is the basic information of a doctor, which could be null. 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); |

**Doctor demission**

The input is the whole doctor name. After clicking the “process” button, the corresponding doctor will be eliminated.

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

**Doctor update**

**The update operation in our website can be subdivided into two parts**. One is query the doctor information by the whole doctor name and present the information in the corresponding input boxes. After the user makes some modification and clicks the “update” button, the system carry out an update operation.

|  |  |
| --- | --- |
| 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" |

* **Patient MGT**

It is a tab regarding the `patient` entity in the database. It contains the functions Patient Search, Add Patient, Update Patient, and Delete Patient. They are also related to simple CRUD MySQL queries and similar to those in the doctor part. Thus the implementation details are skipped here.

* **Outpatient**

It is the most complex tab in our implementation. It is a tab regarding the tables treatment\_record, medicine\_consumption, program and medicine in the **reverse engineering diagram**.

* + **Latest treatment**

The input is one 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; |

* + **Add program**

Given the id of a treatment record and the id of the equipment used as well as the program cost, the system will save the record into the program table. The program id of that record is obtained by the current maximum program id plus one.

|  |
| --- |
| 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); |

* + **Add medicine**

Given the id of a treatment record and the id and amount of the medicine consumed, the system will save the record into the medicine\_consumption table. In the meantime, the system will decrease the stock value in the medicine table by the amount value.

|  |
| --- |
| 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; |

* + **Outpatient history record**

Given the id of a patient, the system will output the query result of all his/her historical treatment records. It is a simple select query.

* **Bed MGT**

It is a tab regarding the sickbed entity in the database.

* + **Sick bed usage status**

It is a simple select all operation on the sickbed table. The implementation details are skipped.

* + **Hospitalized patient check out**

In practical, it is used for a recovered patient to check out. Given the bed id, the management system will locate the sickbed and check whether it is occupied. If so, the system update the status into “available” and set the hospitalization time to be 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 MGT**

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

* **Equipment MGT**

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

|  |
| --- |
| 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.

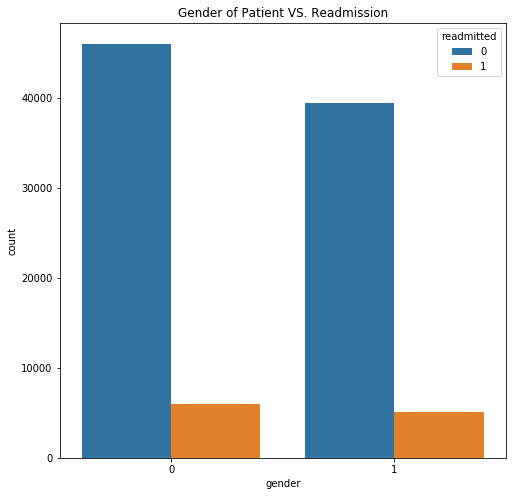
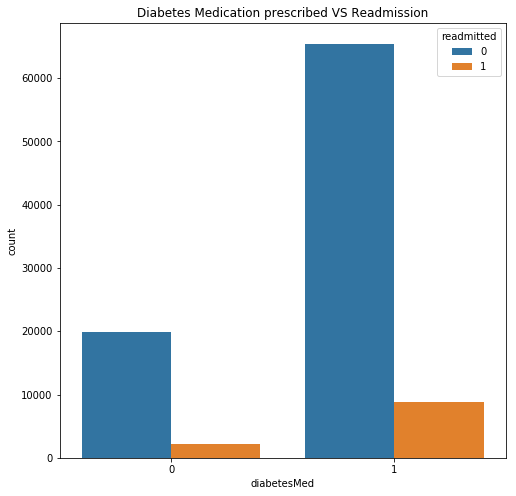
**Data Preparation**

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.

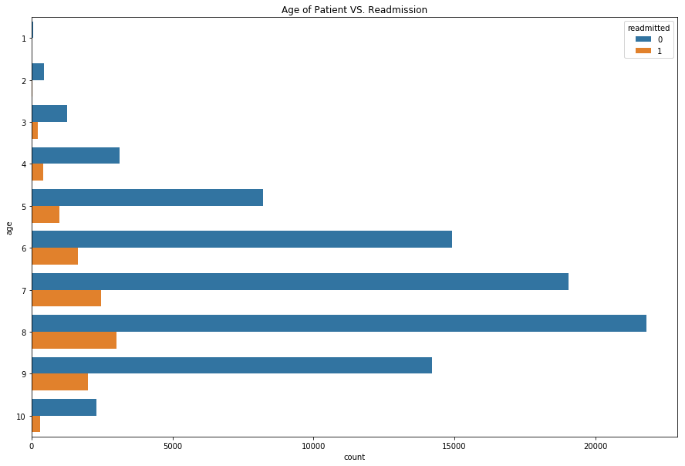
We dropped variables like weight, payer code, and medical specialty because they contain more than 40% missing values. Variables (drugs named citoglipton and examide) all contains the same value are also dropped since essentially they can not provide any interpretive or discriminatory information for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

**Data Visualization**

The relationship between three features and readmission are visualized below.

**Figure 1** Gender and Readmission **Figure 2** Age and Readmission

**Figure 3** Diabetes Medication Prescribed and Readmission

**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** Performance of Different Models

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 will a patient recover after treatment. Therefore, they can provide better health care services.

2.6 Index Data Fields

Before discussing the option on indexing or hashing, some general knowledge should be clarified. By discussing whether to add an index on the colomns, and what type of index should we add, we are talking about the intrinsic data characteristics for each column. Indexing uses B+ tree to store the data, while hashing uses hash table 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, there are pointers 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 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 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 value, 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 it.

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