RAJA MOHAMED AARIF UIN 928000635

CSCE 608 Project: Hospital Database Management System

Project Description:

The project is implementing a Hospital Database Management system. This project implementation and user interface gels well with real world hospital management system in use.

In this project, we have a database which has multiple tables to store various records of different entitie . We have various tables which maintain records of various departments, the doctors who work at the hospital ,the patient details and nurse details ,room etc.

The project works on real world scenarios ,In our project , the assumptions are that a department can have many doctors, a doctor can treat many patients, a patient can have multiple medical records, a room can hold multiple patients , a nurse can attend many patients and a patient can be attended by many nurses.

In this project, we developed an user interface for hospital admin to access various functionalities provided.

Here is a brief overview of various functionalities provided to the user

Searching a patient by patient-ID (pid):

This pulls up the details of the patient like the name,address, contact etc.

Advanced search for Patient:

Here the user is given a choice to enter any detail of patient you remember such as name, address, room_number in the hospital or phone number , and the system will pull up all the matching records and user can parse through them.

Searching a doctor by Doctor-ID (did):

This pulls up the details of the doctor like the address, contact, department etc.

Creating a new entry for a patient:

This provides the user to get a new patient into the hospital and generate an entry for that patient.

The system also offers interesting statistics as below:

Total Income generated by hospital a given month in a given year:

The user can enter month and the system returns an output.

Most frequent disease in a given year:

The user can enter a year and the system returns the most frequent disease

The user also has an option to see the most frequent disease for which patients have been admitted across all the years in total.

Total income generated by a doctor:

The user can give the doctor ID(did) to find out the total income generated by the doctor till now.

Patients Treated by a Department:

The patients treated by a particular department, user can give the department as the input.

Departments Nurse has been working:

The patient also can enter nurse_ id to find the departments of the doctors treating her patients.

Most frequent Medicine prescribed by a doctor:

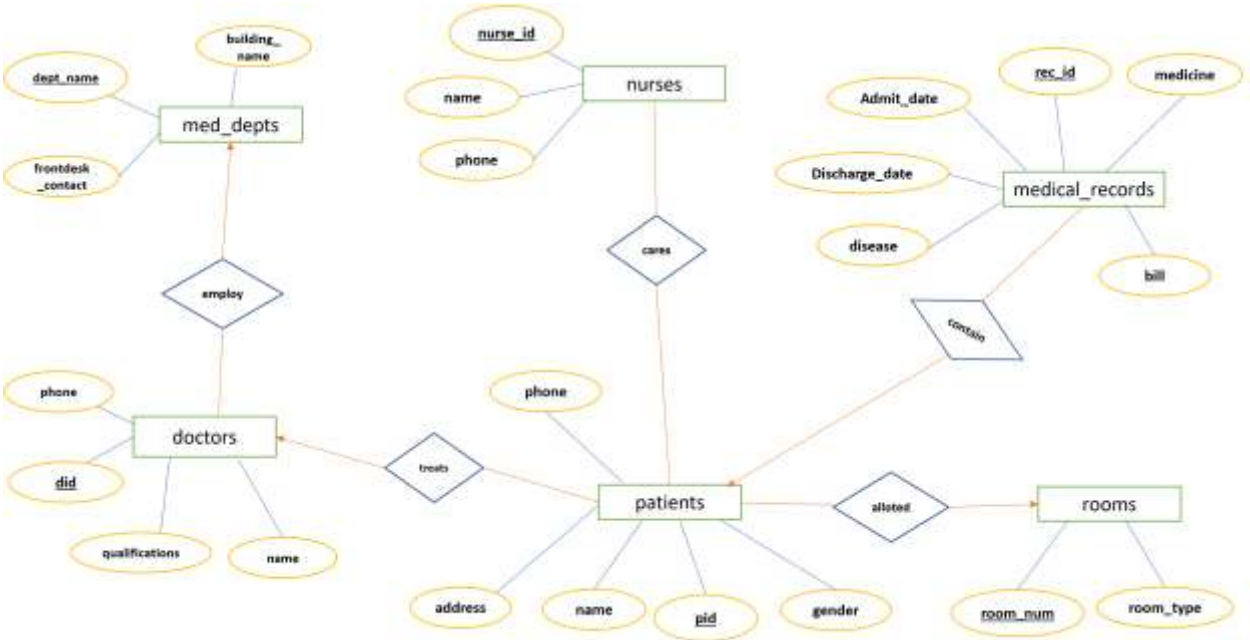
The user can enter the did to get the most frequently prescribed medicine by the doctor.

ER Diagrams:

The major entities are departments, doctors, patients,medical records and nurses.

So my ER diagram contains these as entity sets and the I have created various relationships between them based on real world scenarios like a doctor can treat many patients, a patient can have many medical records , department can have many doctors, a doctor can treat many patients, , a room can hold multiple patients , a nurse can attend many patients and a patient can be attended by many nurses.

The ER diagram is as below:



Tables and Normalization

ER Diagram to Relations:

We will use the following rules to convert ER diagrams to relations

- **Rule1:** Each entity set along with its attributes can be represented as a relation with attributes
- **Rule2**: Each relationship can be represented as a relation with keys of the connected sets and attributes of the relationship itself
- **Rule3**: For Many-one relationship , we can combine the relationship and the entity set on the "many" side of the relationship to be represented as one relation. In our scenario, we can achieve this by including the keys of entity set on the "one" (the entity set with the arrow head)side of the relationship into the relation of the "many" side of the relation(in our case, we don't have attributes on the relationship, so they are ignored)

Consider the **med_depts** entity set, using rule 1 : it can be represented as below:

med_depts: med_depts (dept_name, building_name, frontdesk_contact)

Now see the relationship **employ**, it's a many-one relationship between **doctors** and **med_depts** , where a med_dept can have many doctors, now using the Rule3 as discussed above, we can combine the employ and doctors entity set into one relation by including the key of the med_depts in doctors realtion as attribute and the **employ** has no attributes, so it need not be included.

So now entity set doctors and the employ relationship can be represented as one relation as below:

doctors(did ,name ,phone , qualifications , dept_name)

Now see the relationship **treats**, it's a many-one relationship between **doctors** and **patients**, where a doctor can have many patients, now using the Rule3 as discussed above, we can combine the treats relationship and patients entity set into one relation by including the key of the doctors relation in patients relation as an attribute and the **treats** has no attributes, so it need not be included.

Also relationship **alloted**, it's a many-one relationship between **rooms** and **patients**, where a room can have many patients, now using the Rule3 as discussed above, we can combine the allotted relationship and patients entity set into one relation by including the key of the rooms relation in patients relation as an attribute and the **allotted** has no attributes, so it need not be included.

So now entity set patients and the **treats** relationship and the **alloted** relationship can be represented as one relation as below:

Patients(pid, name, gender, address, phone ,did, roomnum)

Consider the rooms entity set, using rule 1 : it can be represented as below:

rooms(room_num, room_type)

consider the relationship **contain**, it's a many-one relationship between **medical_records** and **patients**, where a patient can have many medical_record entries , now using the Rule3 as discussed above, we can combine the **contain** relationship and **medical_records** entity set into one relation by including the

key of the patients relation as an attribute in medical_records and the **contain** has no attributes, so it need not be included.

medical_records(rec_id, medicine, disease, bill, admit_date, discharge_date, pid)

Consider the nurses entity set, using rule 1 : it can be represented as below:

nurses(nurse_id,name,phone)

Consider the **cares** relationship , it is many-many relationship, it can be represented as relation with keys of nurses and the patients relation

cares(nurse_id, patient_id)

Functional dependencies:

Lets consider the relations one by one to get the functional dependencies:

1.med_depts:

The relation is as follows: med_depts (dept_name, building_name, frontdesk_contact)

The FD' are as below:

- Dept_name → (building_name, frontdesk_contact)
- frontdesk_contact → (building_name, Dept_name)

Assumption : same building can have multiple departments

2. doctors (did ,name ,phone , qualifications , dept_name)

The FD' s are as below:

- Did → (name ,phone , qualifications , dept_name)
- (name ,phone) → (did, qualifications , dept_name)

The assumptions are doctors can have the same names and they can have the same qualifications and departments also, also two doctors can share the same number like as in home number ., people with same name don't share a common phone

So only the above two fd's are present

3. Patients(pid, name, gender, address, phone ,did, room_num)

FD's are as below:

- Pid → (name, gender, address, phone ,did, room_num)
- (name,phone) -> (pid, gender, address ,did, room_num)
- (name, address) -> (pid, gender, phone ,did, room_num)

The assumptions are patients can have the same names ,multiple patients can have the same phone number, multiple patients can have the same address and and multiple patients can have the same room_num, people with same name are not at the same address or don't share a common phone

4. Rooms(Room_num, room_type):

- Room_num → room_type

There can be multiple rooms with same room_type, so the above is the only fd

5.medical_records(rec_id, medicine, disease, bill, admit_date, discharge_date, pid)

- rec_id →(medicine, disease, bill, admit_date, discharge_date, pid)

Assumptions are each pid can have multiple entries, medicine and disease can be the same for many entries , patient can get admit and discharge the same day and also a possibility that he might get admitted again the same day.

6. Nurses(nurse_id,name,phone)

- Nurse_id → (name, phone)
- Name,phone->nurse_id

Here assumptions are two nurses can have the same name and two nurses can have the same phone number

7.cares(nurse_id,pid)

Here there are no fd' as each pid can have multiple nurse_id's and viceversa

BCNF Check

Let's consider each relation one by one to check if they are in BCNF

A relation R is in Boyce-Codd Normal Form (BCNF) if every nontrivial FD X→A (i.e., A∉X) has its left side X a superkey.

Algorithm Normalization(R, T):

Input: A relation R with attribute set Z and FD set T.

1. For each subset Y of Z Do construct Y^+;

2. Record the superkeys and keys Y for R (Y^+=Z);

3. If there is a subset Y of Z such that

   $Y^+ \neq Z$ and $Y^+ \neq Y$ \\ Y → A is a bad FD for some A

   Then

3.1 Call Decomposition(Y) to decompose the

relation R into two smaller relations R_1 and R_2;

3.2 Call Decomposed-FDs(R_k) for k = 1, 2 to

construct the FD sets T_1 for R_1 and T_2 for R_2;

1. med_depts:

The realtion is as follows: med_depts (dept_name, building_name, frontdesk_contact)

Lets say Z is the set of all attributes

The FD' are as below:

- Dept_name → (building_name, frontdesk_contact)
- frontdesk_contact → (building_name, Dept_name)

Lets apply the normalization algorithm to see if there any bad fds

- Lets say Z is the set of all attributes
- consider building_name, now closure of building_name is itself from the fds, so closure is same as attribute from the given fds, so there are no derived or bad fds.
- Now closure of dept_name is Z from the fds, so dept_name is superkey , same applies to frontdesk_contact, so its also superkey so no bad fds deroved till now.
- Now all other subsets of Z are supersets of either dept_name or frontdesk_contact, so their closure is also Z, so no derived fds or bad

So there are no bad fd's and all fd's have superkey on left side of an FD.**Hence med_depts is in BCNF**

2. doctors(did ,name ,phone , qualifications , dept_name)

Lets apply the normalization algorithm to doctors.

The FD' s are as below:

- did → (name ,phone , qualifications , dept_name)
- (name ,phone) → (did, qualifications , dept_name)

Lets say Z is the set of all attributes

- Closure of **did** is Z, so **did** is superkey and closure of all supersets of **did** are also Z.
- Now closure of name , closure of phone and closure of remaining subsets which have single attributes other than did is subset itself as there is no FD for single attribute other than **did**, so y+= y in all those cases, so no derived or bad fd here.
- Now lets consider set of two attributes,From second fd, closure of (name, phone) is Z, so it's a super key , so no derived fds here
- Now, for the remaining subsets of Z of size which contain **did** the closure is Z, as closure of **did is Z.** and the remaining subsets of size2, there is are no fd's corresponding to them from the above points, so the closure of that set is the same set , so no derived or bad fds.
- For the subsets of Z of size 3, the subsets which contain (name, phone) or (did), the closure is Z, so no derived fds in those cases, for the remaining cases, there is no fd for any of their subset as we have considered subsets of size 2and 1 above. So no derived fds or bad fds.
- The same above logic applies to sets of 4,5 and 6 attributes, so no derived or bad fds in all the caes

So there are only two fds as mentioned at the start , both of them have superkey on the left side, so **the relation is in BCNF form**

3. Patients(pid, name, gender, address, phone ,did, roomnum

- Pid → (name, gender, address, phone ,did, room_num)
- Name,phone -> (pid, gender, address ,did, room_num)
- Name, address -> (pid, gender, phone ,did, room_num)

Lets apply the normalization algorithm to patinets.

Lets say Z is the set of all attributes

- Lets consider single attribute subsets,Closure of **pid** is Z, so **pid** is superkey and closure of all supersets of **did** are also Z.
- Now closure of other single attribure subsets like name , closure of phone and closure of remaining is subset itself as there is no FD for single attribute other than **pid**, so y+= y in all those cases, so no derived or bad fd here.
- Now ets consider two attributes subsets ,from second fd, closure of (name, phone) is Z, so it's a super key , so no derived fds here, same applies to (name, address)
- Now, from the remaining subsets containing 2 attributes which contain **pid** the closure is Z, as closure of **pid is Z.**For the remaining, there is are no fd's corresponding to them from the above points or given fds , so the closure of that set is the same set , so no derived or bad fs.
- For the subsets conatining three attributes, the subets which are superset (name, phone) or (name, address) or (pid), the closure is Z, so no derived non trivial fds or bad fds in those cases, for the remaining cases, there is no fd for any of their subsets or themselves as we have considered subsets of two and one attributes above. So no derived fds or bad fds.
- The same above logic applies to subsets of 4,5 ,7and 6 attributes, so no derived bad fds in all those .

So there are only3 fds as mentioned at the start , all of them have superkey on the left side, so **the relation is in BCNF form**

4. **Rooms:**

- Room_num → room_type

There can be multiple rooms with same room_type, so the above is the only fd

Lets apply the normalization algorithm to rooms.

Lets say Z is the set of all attributes

- Lets consider single attribute subsets,Closure of **room_num** is Z, so **room_num** is superkey and closure of all supersets of **room_num** i.e (room_num, room_type)are also Z.
- The only remaining subset is room_type whose closure is itself , so no derived or bad fds here

So only one fd which has superkey on the left side, so the **relation is in BCNF form**

5.medical_records(rec_id, medicine, disease, bill, admit_date, discharge_date, pid)

FDs are as below:

rec_id →(medicine, disease, bill, admit_date, discharge_date, pid)

Lets say Z is the set of all attributes

- Lets consider single attribute subsets,Closure of **rec_id** is Z, so **rec_id** is superkey and closure of all supersets of **rec_id** are also Z.
- Now closure of other single subsets like disease, closure of medicine and closure of remaining is subset itself as there is no FD for single attribute other than **rec_id**, so y+= y in all those cases, so no derived or bad fd here.
- Consider the subsets containing 2 attributes which contain **rec_id** , their closure is Z as the rec_id is superkey , so all those subsets are superkeys as the closure is Z,**.** closure of any superset of thesee subset is also Z
- For the remaining, subsets of 2 attributes there is are no fd's corresponding to them from the above points or given fds , so the closure of that set is the same set , so no derived or bad fs.
- The same above logic applies to subsets of 3,4,5 ,6.and 7 attributes, so no derived bad fds in all those .

So there is only single fd as mentioned at the start which superkey on the left side, so the relation is in BCNF form

6. nurses(<u>nurse_</u>id,name,phone)

- nurse_id → (name, phone)
- name,phone->nurse_id

Here two nurses can have the same name and tww nurses can have the same phone number, so that os the only fd.

Lets apply the normalization algorithm to nurses.

Lets say Z is the set of all attributes

- Lets consider single attribute subsets,Closure of **nurse_id** is Z, so **nurse_id** is superkey and closure of all supersets of **nurs_id** are also Z.
- Now closure of other single subsets like name, closure of phone is subset itself as there is no FD for single attribute other than**nurse_id**, so y+= y in all those cases, so no derived or bad fd here.
- Consider the subsets containing 2 attributes which contain **nurse_id** , their closure is Z as the nurse_id is superkey , so all those subsets are superkeys as the closure is Z,**.**
- Now from the remaining two attributes subsets ,from second fd, closure of (name, phone) is Z, so it's a super key , so no derived fds here, closure of any superset of this subset is also Z
- For the remaining, subsets of 2 attributes there is are no fd's corresponding to them from the above points or given fds , so the closure of that set is the same set , so no derived or bad fs.
- The same above logic applies to subsets of 3attributes, so no derived bad fds in all those .

So there are only2 fds as mentioned at the start , all of them have superkey on the left side, so the relation **is in BCNF form**

7.**cares(nurse_id,pid)**

Here ther are no fd' as each pid can have multiple nurse_id's and viceversa

Closure of (nurse_id), (pid) is themselves as no fds.,Closure of (nurse_id, pid) is Z , so no derived or bad fds **so the relation is in BCNF form**

BCNF removes redundancy as it removes the FD's which lead to redundancy

Data Generation and Duplicate Avoidance:

MySQL is being used as backedn for theproject.I have used python's **mysql.connector** module to connect to the MySQL database and populate my tables using a python script.

The below code is the general method where the script enters into desired table in database and insert many entries fetching it from a list. This method is used to populate all the tables

```
conn = mysql.connector.connect(host='localhost', database='raja', user='root',password='raja123')

cursor =conn.cursor()

for i in l:

    sql_query = """INSERT INTO raja.med_depts (dept_name, building_name, frontdesk_contact) VALUES (%s, %s, %s)"""

    val= (i[0],i[1],i[2])

    result = cursor.execute(sql_query,val)

    conn.commit()

conn.close()
```

Lets consider each relation one by one to see how I populated the table and avoided duplicates

1.**med_depts (dept_name, building_name, frontdesk_contact):**

The dept_names have taken from https://www.sgu.edu/blog/medical/ultimate-list-of-medical-specialties/. And these are mades as a list using py

The building names and frontdesk contacts are randomly created using a python library using **pydbgen** Which returns randomly generated names and phone nums.

The primary key is the dept_name in the table , I have removed duplicates in that list before using them using python set .Also, sql throws an error if we use

2. **doctors(did ,name ,phone , qualifications , dept_name)**

The names and phone field are randomly created using a python library using **pydbgen**

The qualifications is fetched from the wikipedia page and dept_name is fetched from the above relation. I have created a list dept_name from dept_name attribute in med_depts.The dept_name in medical_records is populated by randomly selecting from that list by using **random** python library and

indexing the randomly generated value to that list . So only valid dept_names values are populated and so interesting joins can be done

The key **did** is sequentially generated and is the primary ke , so no duplicates for the key, it returns an error if we still try to insert a duplicate key value. Also dep_name is made foreign key in order to avoid dept_names entry which are not in med_depts table.

3. Rooms(Room_num, room_type):

The room_num is key and sequentially generated , so no possiblilty of duplicates

Room type is randomly selected from a list of fixed room types.

4. **Patients(pid, name, gender, address, phone ,did, room_num)**

The names , gender, address and phone field are randomly created using a python library using **pydbgen.**pid is sequentially generated and is the primary key , so no dupliciates for the key, it returns an error if we still try to insert a duplicate value

A list of did and room_num from room_num attribute in rooms and did in doctors is created and the did and room_num in patients fileds are populated by randomly selecting room_num from the room_num list and did in did list by using **random** python library and indexing the randomly generated value to these lists . So only valid values are populated and so interesting joins can be done.

Also room_num and did are made foreign keys in onrder to avoid invalid values being populated in the patients table.

5. medical_records(rec_id, medicine, disease, bill, admit_date, discharge_date, pid)

Here rec_id is the key and is sequentially generated, so no duplicates for key.

The medicine information is obtained from https://www.drugs.com/international/ and the disease info is obtained from https://www.cdc.gov/diseasesconditions/az/g.html and I have used python to convert these information into lists to be used.

I have created a list of pid from pid attribute in patients.The pid in medical_records is populated by randomly selecting pid from the list by using **random** python library and indexing the randomly generated value to that list . So only valid pid values are populated and so interesting joins can be done, also [id is made foreign key

The bill, admit_date, discharge_date filed are radomly generated by python script and random,randrange module.

6. **Nurses(nurse_id,name,phone)**

Nurse_id is key and is sequentially generated , so no duplicate values.

Name and phone are generated by pydbgen module

7.**cares (nurse_id, pid)**

Both attributed combined form the key , they are randomly slected from nurse_id column in nurses and pid column in patients ,Both combined are keys, so no duplicate entries are allowed as it throws a error.

Code Repository:

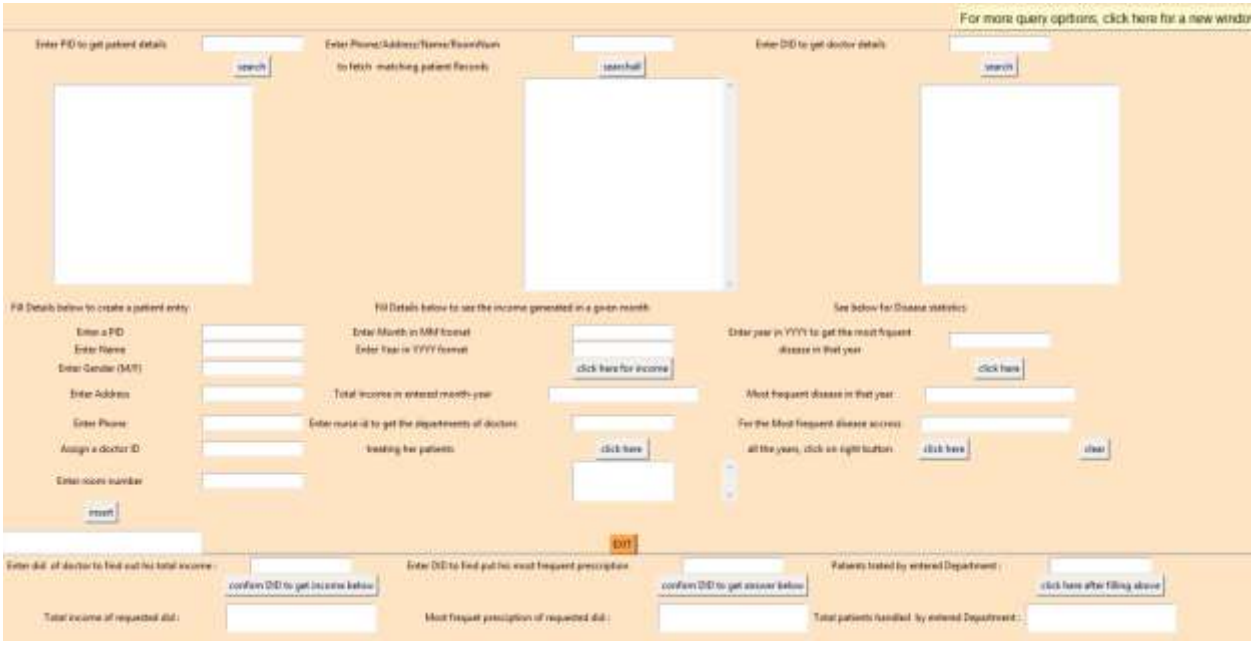
Code is present in https://github.com/aarifraja/608-Project-Hospital-Database.

User Interface and Functions

In this project , MYSQL is used for backend database and front end is made using Python **Tkinter** Module.The interaction between front end and backend is also coded using Python.

I have used **mysql connector** to allow python to interact with MySQL.

This the homepage of the our UI



Below are the functions provided by the user

Searching a patient by patient-ID (pid):

The user needs to enter the PID as in below screenshot to get the patient details

The query used is as below

select name,gender,address,phone,roomnum from patients where pid = %s

Advanced search for Patient:

Here the user is given a choice to enter any detail of patient in highlighted portion you remember such as name, address, room_number in the hospital or phone number , and the system will pull up all the matching records

The query used is as below:

select pid,name,gender,address,phone,roomnum from patients where name = %s or address =%s or phone= %s or roomnum= %s"

Searching a doctor by Doctor-ID (did):

The user needs to give the DID for the details

The query used is

Select Name, Phone,Qualifications,Dept_name from doctors where did = %s

Creating a new entry for a patient:

The user can fill the details below and create a new record

The query is as below:

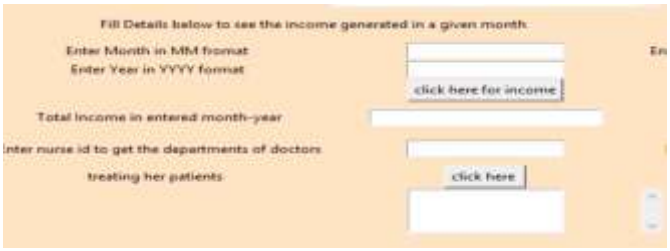
INSERT INTO patients (pid,name,gender,address,phone,did,roomnum) values (%s,%s,%s,%s,%s,%s,%s)

The system also checks if we are creating a duplcaite key and if did and roomnum are vald entries by comparing if they are present in the other tables.

Total Income generated by hospital a given month in a given year:

The user can enter month and the system returns ouput

Sql query: select sum(bill) from medical_records where discharge_date>= %s and discharge_date < %s



Here discharge_Date is created using the month and year to cover all the dates in a given month

Most frequent disease in a given year:

The user can enter a year and the system returns the most frequent disease

*select disease as freq from (select * from medical_records where admit_date>= %s and discharge_date < %s) as t2 group by disease order by freq desc limit 1*;

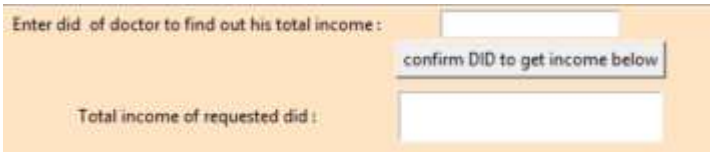
The user also has an option to see the most frequent disease for which patients have been admitted across all the years in total.

Sql query : *select disease , count(disease) as freq from medical_records group by disease order by freq desc limit 1*

Total income generated by a doctor:

The user can give the doctor ID(did) to find out the total income generated by the doctor till now.

Sql query: *select income from (select did,SUM(bill) as income from patients join medical_records on medical_records.pid = patients.pid group by did) as t1 where did = %s*



Patients Treated by a Department:

The, user can give the department as the input and the system returns patients treated by a particular department

Sql query : *select count(*) from (patients join doctors on doctors.did = patients.did) where dept_name = %s*

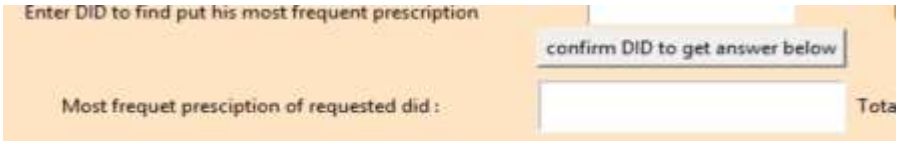
Departments Nurse has been working:

The patient also can enter nurse id to find the departments of the doctors treating her patients.

Sql query is "*select dept_name from (select t1.pid,t1.nurse_id,patients.did from (select * from cares where nurse_id = %s) as t1 join patients on t1.pid= patients.pid) as t2 join doctors where t2.did=doctors.did*";

Most frequent Medicine prescribed by a doctor:

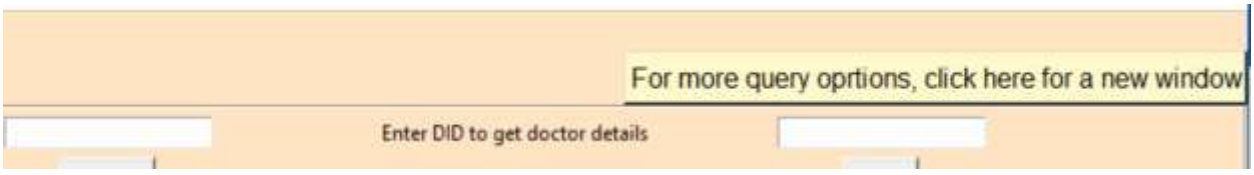
The user can enter the did to get the most frequently prescribed medicine by the doctor.



Sql query is *select medicine from (patients join medical_records on medical_records.pid = patients.pid and did = %s) group by medicine order by count(*) DESC limit 1 "*;

Exit Button:

Also an exit button has been given provided to exit the current window.



click on yellow button on top left button on top right as shown above to get a new window as below:

You can fetch the record details ot nurse details by using the rec_id and nurse-id in the above winows respectively.

select Name, Phone from nurses where nurse_id = %s and *select medicine,disease, bill, admit_date, discharge_date,bill from medical_records where rec_id = %s* are the queries

Discussion

The project has been a good learning experiences

First to speak of the challenges:

- One major challenge is Tkinter which I have used for front end , maintaining various modules becomes disruptive as some times when we add a widget, it displaces other widgets, needed to go through a lot of documentation and also stack overflow to get a proper layout .
- Data generation is a challenge because we have to keep track of duplicate key entries and foreign key validity

Learnings:

- This project has given a good hands on experience on how to approach when designing a database
- Writing SQL queries for the project has given a good coding experience in SQL
- Gave a refresher to all the concepts we have learnt for designing a database and how we use those concepts to build one.
- Learnt new front end GUI using Python Tknter