Patient Relationships System

Database Systems/Databases and SQL
CSCI 6622/2215-01

Team "New"
(The Database Trio)

University of New Haven

TAGLIATELA COLLEGE OF ENGINEERING, West Haven, CT

Submitted To:

Dr. Reza Sadeghi

Spring 2021

Team Information

Team Name

New (The DatabaseTrio)

Team Members

1. Maria Kurisoo mkuri1@unh.newhaven.edu (Team Head)
2. Hayley Smith hsmit9@unh.newhaven.edu (Team Member)
3. Alexander Socha asoch1@unh.newhaven.edu (Team Member)

Roles of Team Members

1. **Maria Kurisoo**: Maria led the team and was an active contributor to *all* aspects of the project.

2. **Hayley Smith**: Hayley was an active contributor to *all* aspects of the project.

3. **Alex Socha**: Alex was an active and crucial asset to the design of our GUI and Python code. He also assisted in uploading the project to GitHub.

All Members: Our team met numerous times via Zoom to work on the project, and all team members were always present and punctual. All team members were immensely helpful and hardworking in creating this successful project. All team members actively presented the project on Tuesday, May 04, 2021 during our class lecture period.

GitHub Project URL:

https://github.com/Socha-Alex/databases_Project

Table of Contents

Table of Figures

Table of Tables

(no tables)

Table of Attachments

Please reference the following attachments related to this project:

Name	File Name	Purpose
Database Tables	"TABLES-GroupNew_PatientRelationshipsSystem.xlsx"	This file displays the details of the tables in our database.
SQL Code	"patient_relationships_db.sql"	This script has our SQL code that we wrote to create our database and insert entries.
GUI Code	"database_project_finalUpdate_04May2021.txt"	This file has the Python Tkinter code that we used to create our GUI and link with our database.
Project PowerPoint	"Database Final Presentation_GroupNew.pptx"	This PowerPoint is what was presented in class.

Introduction

For this project, we were tasked with creating a useful SQL database that could connect to a Graphical User Interface (GUI) so users can easily insert, delete, modify, and search the database. Creating this project involved: drafting an idea for our project, creating an Entity Relationship Model and our tables, writing SQL code using MySQL Workbench, inserting entries into our database, creating a GUI, and linking our SQL code with the GUI. This document serves to describe these steps in detail.

Project Description

1. **Project Topic:** Patient Relationships System
2. **Who will be the users of our database?**

   Small private primary care medical offices with only a few providers.

   What information is it expected to be store?

   The database will store information related to patient relationships, which includes basic patient information, what provider the patient sees at the medical office, their emergency contact, their pharmacy, and their legal guardian.

   What is the merit of using our database?

   Small private primary care medical offices, especially those with only a few healthcare providers, often do not have the funds for robust database systems. Consequently, the medical office may have trouble keeping track of relationships connected to the patient. Our Patient Relationships System database is small which allows it to be more affordable, and convenient since it stores information on the players in a patient's care (what provider they see, their emergency contact, their legal guardian, and their preferred pharmacy). The medical office or primary care provider can easily view this information and decide whether they need to reach out to anyone to proceed with the patient's care. For example, if a patient needs a medical procedure, the healthcare provider can view if this patient has a legal guardian that needs to be contacted for consent.

   Our Patient Relationships System is simple. It is does not include information beyond what is necessary to understand the relationships of the patient. Therefore, it does not include information on the patient's health history and treatment, or any other medical details. The simplicity will allow the healthcare provider to quickly access pertinent relationship information without being bombarded with extraneous, time-consuming details.

   What is our estimation for number of records?

Since the World Health Organization (WHO) recommends no more than 1000 patients per provider[1], this database may need to hold a few thousand patients, depending on how many providers are at the medical office. Some patients may have no relationships besides their primary care provider, while other patients may have many relationships, therefore it is hard to estimate the number of records for the other entities.

Normalization: Our database adheres to normalization up to at least 3NF.

3. **Tables:** Please see the included Database Tables document. In the document you will see our five entities, their attributes, their data types, their primary and foreign keys, their purpose, and examples.
4. **ER Model:** Our ER model is described and shown on pages 7 – 9. It provides a visual of our entities, attributes, primary keys, and the relationships, participations, and cardinalities between the entities.
5. **SQL:** Please see the included SQL Code document**.** Our SQL code is described on pages 10 - 11. Our code involved creating the database, creating the tables, creating primary and foreign keys, and inserting into all our tables.
6. **GUI:** Please see the included GUI Code document. Our GUI is described on pages 11 – 16. Our GUI code created a primitive GUI to allow users to interact with our database for inserts, deletes, modifications, and searching. Our GUI code was linked with our SQL code.
7. **Presentation:** Please see included Presentation PowerPoint document. The document shows what was presented to the class on Tuesday, May 04, 2021.

Entity Relationship Model

In Figure 1 on the next page, our Entity Relationship Model can be seen. Entities are shown in the square boxes, attributes are shown in the ovals, and primary keys are underlined. Additionally, relationships are shown with diamonds, cardinality is show with “N” or “1”, complete participation is represented with double lines, and partial participation is represented with single lines.

Every entity has a serial ID, which is used as their primary key. The entities’ respective primary keys are patientID, providerID, contactID, pharmacyID, and guardianID. All entities have a relationship with the Patient entity. These relationships are described below as well the resulting foreign keys:

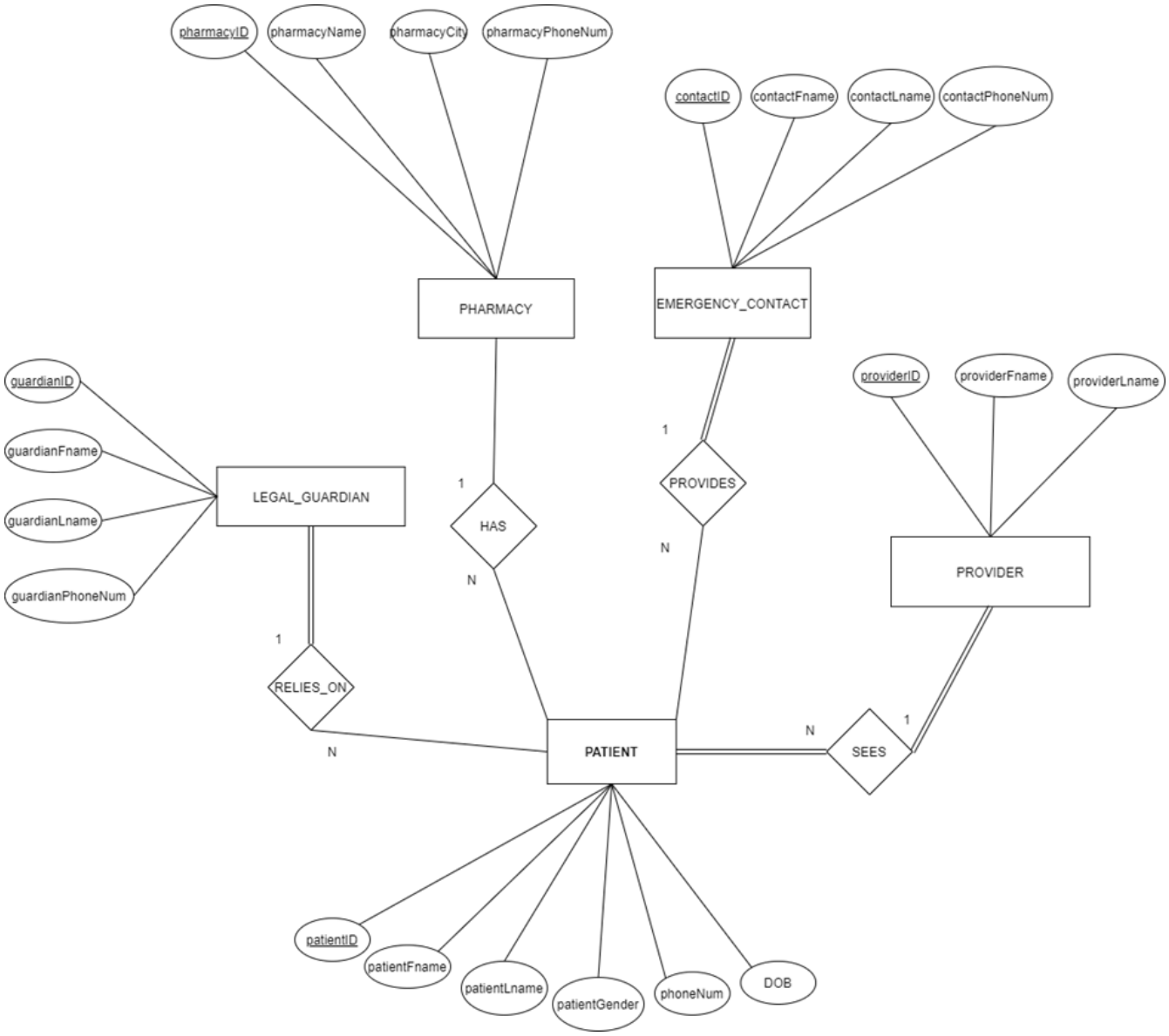
- **Patient-Provider:** The Patient sees the Provider, and both have complete participation. Every Patient must have one Provider, and one Provider may have many Patients. The Patient table has “providerID” as a foreign key to indicate what Provider they see.

- **Patient-Emergency_Contact**: The Patient provides the Emergency_Contact information. Not every Patient will have an Emergency_Contact, but every Emergency_Contact will be associated with a Patient. Many Patients will have one Emergency_Contact. The Patient table has “contactID” as a foreign key to indicate who their Emergency_Contact is. The Emergency_Contact table has “patientID” as a foreign key to indicate who they are the Emergency_Contact of.

- **Patient-Pharmacy:** The Patient has a Pharmacy. Not every Patient will have a Pharmacy. Each Pharmacy may be used by many patients. The Patient table has “pharmacyID” as a foreign key to indicate what pharmacy they have.

- **Patient-Legal Guardian:** The Patient relies on the Legal_Guardian. Not every Patient will have a Legal_Guardian, but every Legal_Guardian will be associated with a Patient. Many Patients will have one Legal_Guardian. The Patient table has “guardianID” as a foreign key to indicate who their Legal_Guardian is. The Legal_Guardian table has “patientID” as a foreign key to indicate who they are the Legal_Guardian of.

Figure 1: Entity Relationship Model



SQL Database

How we implanted our entity relationship model through MySQL and how we filled our tables:

Lines 2-4: We created our database and named it "patient_relationships_db". We then did the "show databases" command to ensure that our database was created. We then began the process of working in our database using the "use" keyword and the name of our database.

Lines 7-12: We created our "Provider" table, which involved the "create table" command, and listing out the attributes, their datatype, and if they were not null. We listed our primary key for the Provider table in this section as well.

Lines 16-20: Here we inserted data into our Provider table. We only did 5 entries for this section, rather than 20, because our database is meant for a small primary care medical office, so we thought more than 5 entries would be too much.

Lines 23-36: Here we created our "Patient" table and added the primary key and one of the foreign keys. We chose to create this table after the Provider table, because the ProviderID (which is the Provider primary key) is a not null foreign key within the Patient table (provider_is). We chose not null because every patient is assigned a provider in the primary care medical office, and there will never be a circumstance where a patient is in the system without being connected to a provider. Therefore, we needed the Provider table already in place so that it could later be referenced when we assigned it as a foreign key in the Patient table. The Patient table also has 3 additional foreign keys (pharmacy_is, contact_is, and guardian_is) which were not referenced here, because the other tables (Pharmacy, Emergency_Contact, and Legal_Guardian) need to be created first. Those foreign keys are default null because the patient is not required to have connections to those entities.

Lines 39-58: Here we inserted our 20 entries into the Patient table.

Lines 61-67: Here we created the "Pharmacy" table and assigned its primary key.

Lines 70-89: Here we inserted our 20 entries into the Pharmacy table.

Lines 92-100: Here we created the "Emergency_Contact table" and assigned its primary key and foreign key. We ensured this table was created after the Patient table, because the Patient table primary key (patientID) is the not null foreign key (contact_of) for the Emergency_Contact table. We set it up this way because we want every Emergency_Contact in the system to be assigned to a patient, because if it is not, the contact information would be useless in the system and there would be no way to determine what patient it belongs to.

Lines 103-122: Here we inserted our 20 entries into the Emergency_Contact table.

Lines 125-133: Here we created our "Legal_Guardian table" and assigned its primary key and foreign key (guardian_of). Like the Emergency _Contact table, the foreign key is not null and references the patientID because there should never be a circumstance where a legal guardian is in the system without being connected to a patient.

Lines 137-146: Here we inserted 10 entries into the Legal_Guardian table. We chose not to do 20 because it is unrealistic that every patient would have a legal guardian. We expect that only patients who are minors or have cognitive issues would have a guardian.

Lines 149-152: By this point, all of the tables have been created, so here we added the remaining foreign keys to the Patient table, using the "alter table" command.

Lines 156-175: Lastly, we updated the Patient table entries using the "update" and "set" commands to reflect the newly added foreign keys.

Graphical User Interface (GUI)

Our GUI was created using Python and Tkinter. Our GUI is primitive but functional. Our GUI provides instructions to the user to compensate for its limitations. Figures 2-8 showcase our GUI in action:

Figure 2: The welcome page of our GUI.

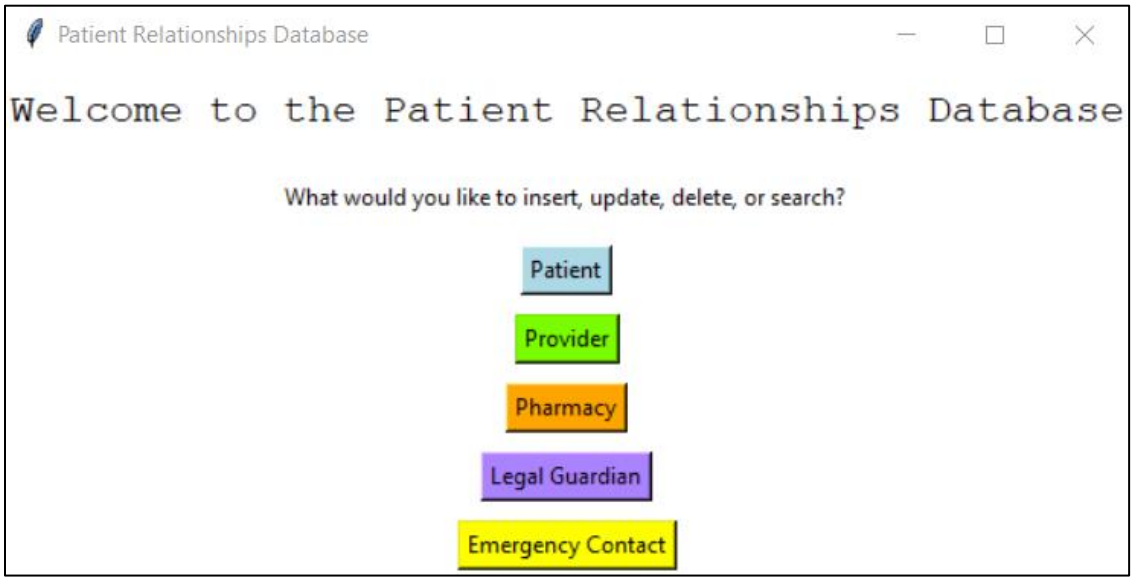


Figure 3: Emergency Contact Window (Search ContactID "3")

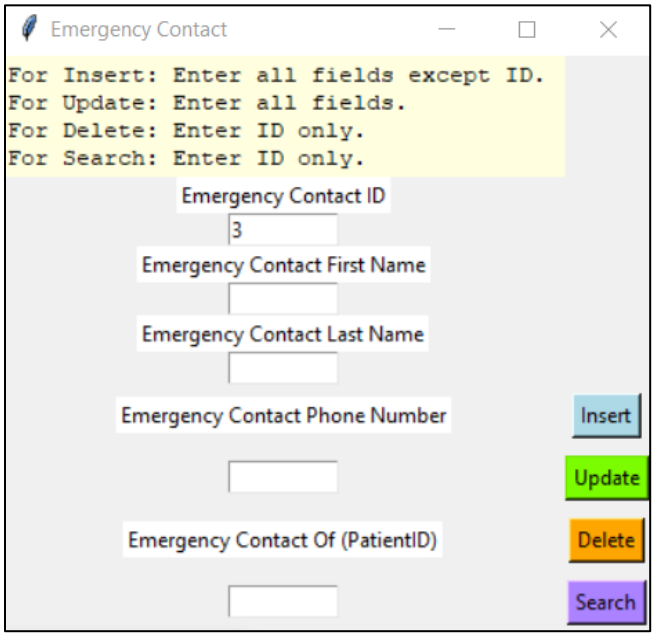
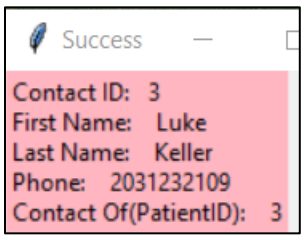


Figure 4: Search Success Window



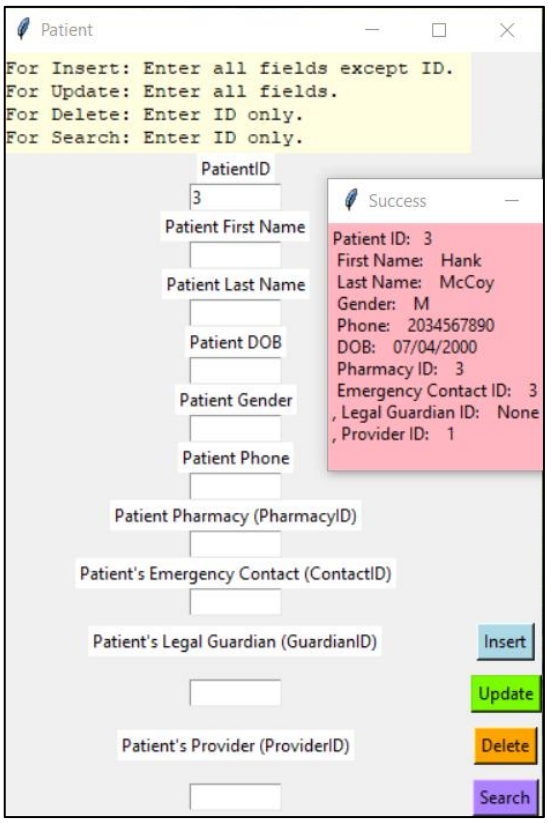


Figure 5: Patient Window (Search PatientID "3"), Search Success Window

Figure 6: Provider Window (Update ProviderID "1" with "Tim" and "Greer")

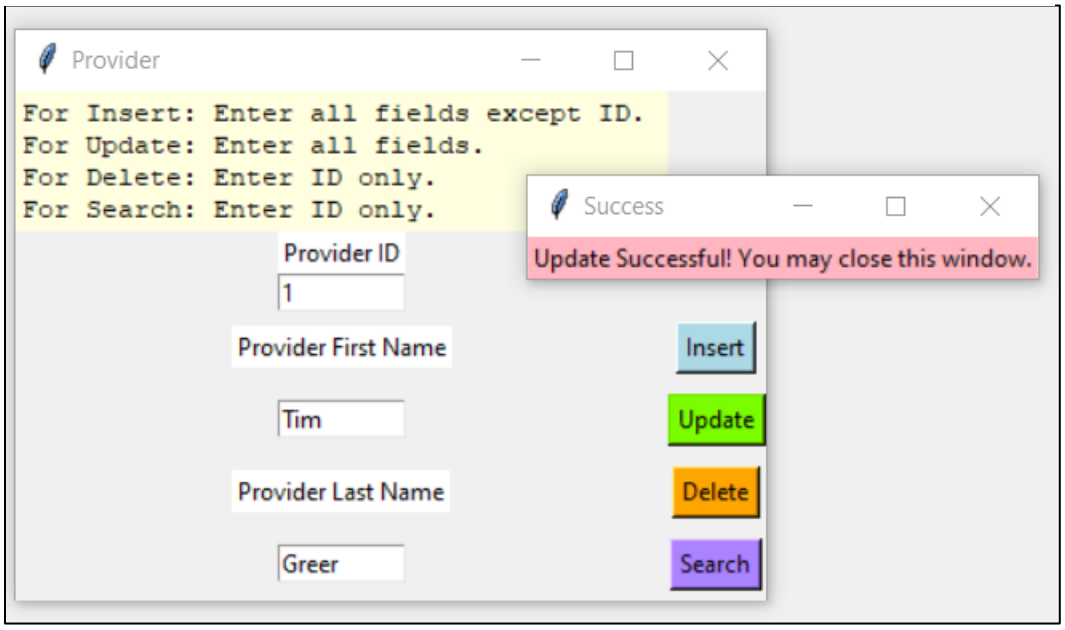


Figure 7: MySQL View (before Provider update)

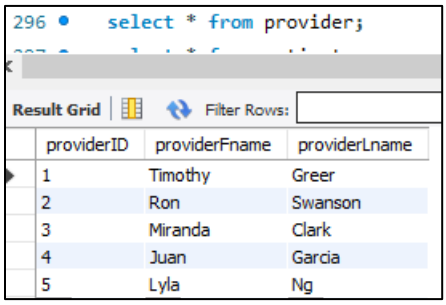
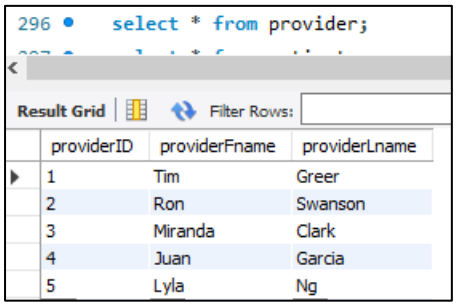


Figure 8: MySQL View (after Provider update)



Description of Our GUI Code:

Lines 1-33: MainWindow( ) is a function to create and populate our main window. After window was created, named, and geometry was set. A frame on the window was then created to house two labels: a welcome message and instructions to the user. For each label the following were specified: label name, font color, background color, font type and size, and geometry. The instructions ask the user what entity they would like to insert, delete, update, or search. Five buttons were then added to the frame, one for each entity. For each button, the following were specified: the frame, font color,

background color, button name, a command, and geometry. The command assigned to each button was a function for a new window to pop up, respective to the entity, PatientWindow( ), ProviderWindow( ), PharmacyWindow( ), GuardianWindow( ) or ContactWindow( ).

Lines 38-217: Patient-Related Functions and Window

- **Lines 38-61:** InsertPatient( ) is a function that is a component of the Patient Window, that is called when the user presses the Insert button. This function causes a patient to be inserted into our database. In this function, first the names of our database variables were assigned to be the global variables used in our GUI. These global variables were retrieved using the built-in get( ) function. The global variables come from what the user inputs in the Patient Window. After this step, an insertion SQL command was assigned into a variable as a string. The string receives the inputs from the user by use of the "%s" conversion specifier. The string was sent as an input to the database through the database function mycursor.execute( ). "mycursor" is a cursor we defined for our database in our main function. "db.commit( )" was then used to end the transaction in our database. "db" is what we defined our database as in our main function. Lastly, a success window was created to display to the user that the insert was successful. To create the success window, a window and label was created in a similar manner to how previous windows and labels were created.
- **Lines 65-89:** UpdatePatient( ) is also a component of the Patient Window. It is called when the user presses the Update button. This function updates a patient in the database. This function was designed almost identical to the InsertPatient ( ) function, except the string that is passed into the database is an update SQL command, rather than an insertion command.
- **Lines 93-108:** DeletePatient( ) is another component of the Patient Window. It is called when the user presses the Delete button. It causes a patient entry to be deleted from the database. This function was designed almost identical to the InsertPatient ( ) function, except the string that is passed into the database is a deletion SQL command, rather than an insertion command.
- **Lines 112-128:** SearchPatient( ) is another component of the Patient Window. It is called when the user presses the Search button. It causes a patient entry to be selected from the database and then the information is retrieved and displayed to the GUI user. This function was designed similarly to the InsertPatient ( ) function, except the string that is passed into the database is a search SQL command, rather than an insertion command. Additionally, after the search command the function fetchone( ) was used to retrieve all the data from the row so it could be brought back for the user to see.
- **Lines 132-217:** PatientWindow( ) is a function that is called from the Main Window when the user clicks the Patient button. This window allows the user to input the data they would like insert, update, delete or search. First, we created the window and an instruction label, in a similar manner as was done for previous windows and instruction labels in our code. We then made labels for every patient attribute, set the variable as global for use in other functions, and created an entry field for each respective attribute label for user input. Finally, we have four

buttons. Each button calls to one of the four patient functions previously described and results in a database change.

Lines 222-346: Provider-Related Functions and Window

This section of our code is identical to the Patient-related section, except respective to the Provider table in our database.

Lines 351-483: Pharmacy-Related Functions and Window

This section of our code is identical to the Patient-related section, except respective to the Pharmacy table in our database.

Lines 488-626: Emergency Contact-Related Functions and Window

This section of our code is identical to the Patient-related section, except respective to the Emergency Contact table in our database.

Lines 632-770: Legal Guardian-Related Functions and Window

This section of our code is identical to the Patient-related section, except respective to the Legal Guardian table in our database.

Lines 776-792: Main Function

In our main function we connect to our database, import tkinter, and call the MainWindow( ) function to display the main window to the user.

Note: Throughout the above code description the following were Tkinter widgets: windows, labels, entries, buttons.

Connection of Database and GUI

In Figure 9 on the next page, a screen shot of our code shows how we connected to our database. First, we imported the MySQL connector. Using the connector, we then connected to our database by listing our log-in credentials and the name of our database. This information was all stored into the variable “db”. This variable was used throughout our GUI code to connect to our database, as shown in Figure 10. In Figure 9, it can also be seen that we assigned a database cursor to the variable “mycursor”. This cursor allowed us to run the execute command shown in Figure 10. In Figure 10, an SQL command to perform a provider deletion is stored in a variable as a string. In the next line, the execute function previously described receives the string as input and the database receives this string and runs the SQL command. Lastly, “db.commit()” ends the SQL transaction. This type of procedure was done throughout our code every time an SQL command was needed.

Figure 9: Connecting to Database

```
775
776 if __name__ == '__main__':
777
778     import mysql.connector
779
780     db = mysql.connector.connect(
781         host = "localhost",
782         user = "root",
783         passwd = "77OILCsc!614",
784         database = "patient_relationships_db"
785         )
786
787     mycursor = db.cursor()
788
```

Figure 10: An example of passing an SQL command to the database.

```
242
243 def DeleteProvider():
244     #Link SQL attributes with GUI variables
245     providerID = entry_providerID.get()
246
247     #Send SQL statement to database as a string
248     delete_provider = "Delete from provider where providerID= '%s' ;" % (providerID)
249     mycursor.execute(delete_provider)
250     db.commit()
251
```

References

1. Kumar, R., & Pal, R. (2018). India achieves who recommended doctor POPULATION ratio: A call for paradigm shift in public Health discourse! Retrieved May 04, 2021, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6259525/