Hospital Database Management System 2023

DECEMBER 21

COMPANY NAME

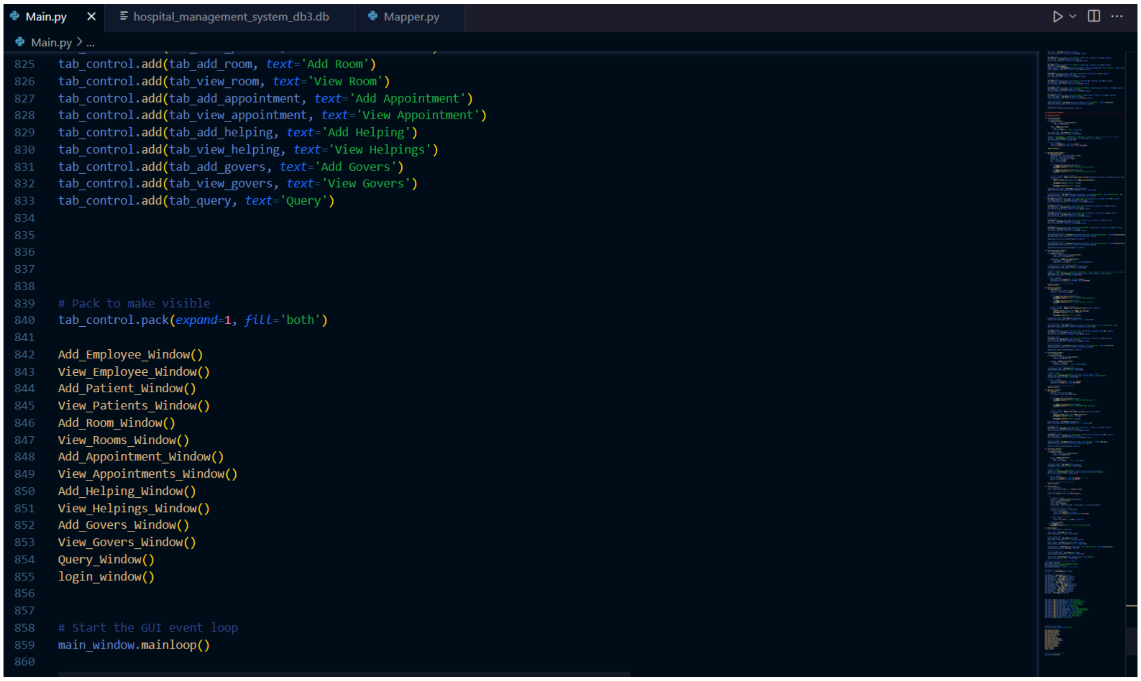
For Advanced Database Course

Participants Information

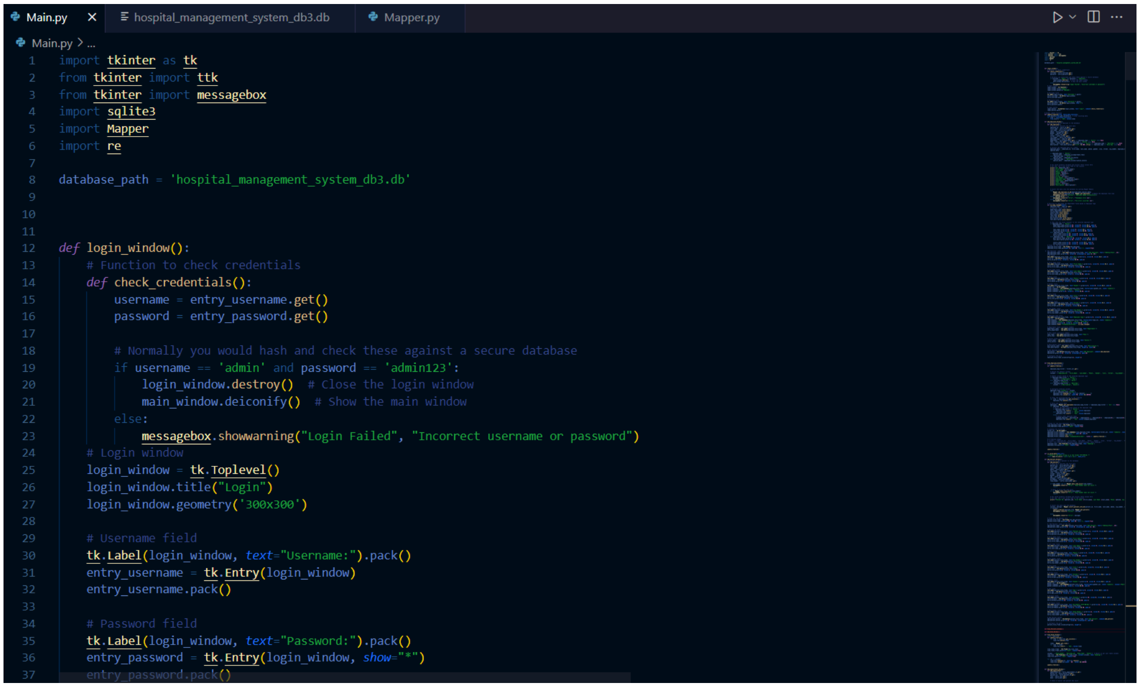
Name	ID
Marwan Amr Mohamed wageh	20201446971
Ahmed Emad Mohamed Ahmed	20201497546
Karim Hussam Al-Din El-Sayed Mohamed	20201446854
Abdelrahman Mohamed Ali Mohamed	20201446699

A Walk through The Process of Making the Project

Here is the GUI Code

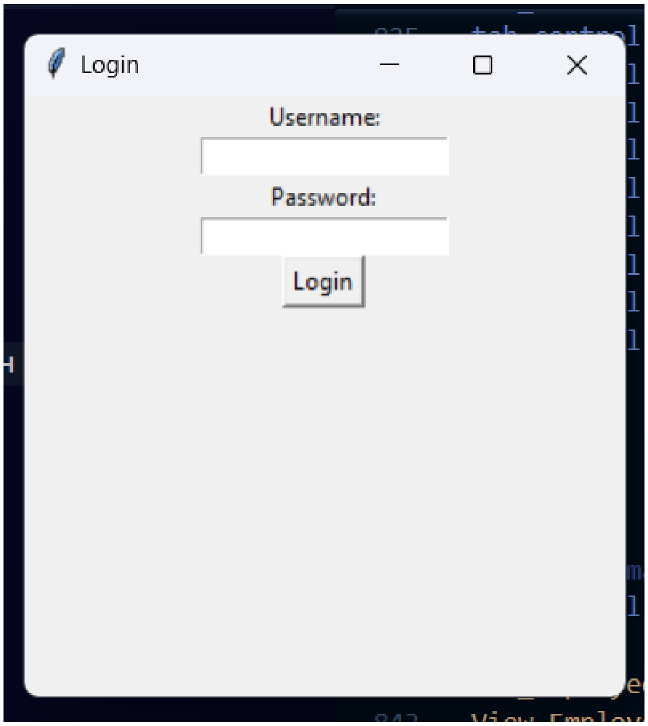


```python
tab_control.add(tab_add_room, text='Add Room')
tab_control.add(tab_view_room, text='View Room')
tab_control.add(tab_add_appointment, text='Add Appointment')
tab_control.add(tab_view_appointment, text='View Appointment')
tab_control.add(tab_add_helping, text='Add Helping')
tab_control.add(tab_view_helping, text='View Helpings')
tab_control.add(tab_add_govers, text='Add Govers')
tab_control.add(tab_view_govers, text='View Govers')
tab_control.add(tab_query, text='Query')





# Pack to make visible
tab_control.pack(expand=1, fill='both')

Add_Employee_Window()
View_Employee_Window()
Add_Patient_Window()
View_Patients_Window()
Add_Room_Window()
View_Rooms_Window()
Add_Appointment_Window()
View_Appointments_Window()
Add_Helping_Window()
View_Helpings_Window()
Add_Govers_Window()
View_Govers_Window()
Query_Window()
login_window()


# Start the GUI event loop
main_window.mainloop()

```

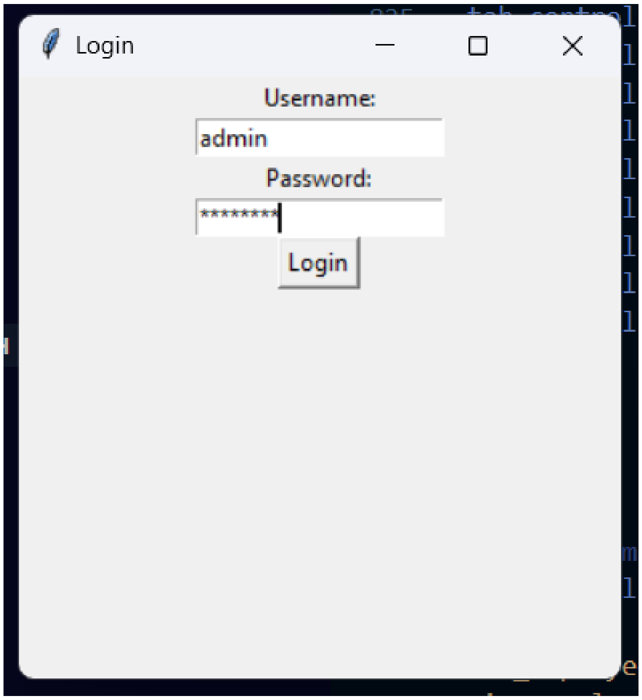


```python
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
import sqlite3
import Mapper
import re

database_path = 'hospital_management_system_db3.db'



def login_window():
    # Function to check credentials
    def check_credentials():
        username = entry_username.get()
        password = entry_password.get()

        # Normally you would hash and check these against a secure database
        if username == 'admin' and password == 'admin123':
            login_window.destroy()  # Close the login window
            main_window.deiconify()  # Show the main window
        else:
            messagebox.showwarning("Login Failed", "Incorrect username or password")
    # Login window
    login_window = tk.Toplevel()
    login_window.title("Login")
    login_window.geometry('300x300')

    # Username field
    tk.Label(login_window, text="Username:").pack()
    entry_username = tk.Entry(login_window)
    entry_username.pack()

    # Password field
    tk.Label(login_window, text="Password:").pack()
    entry_password = tk.Entry(login_window, show="*")
    entry_password.pack()
```

And here is the output.

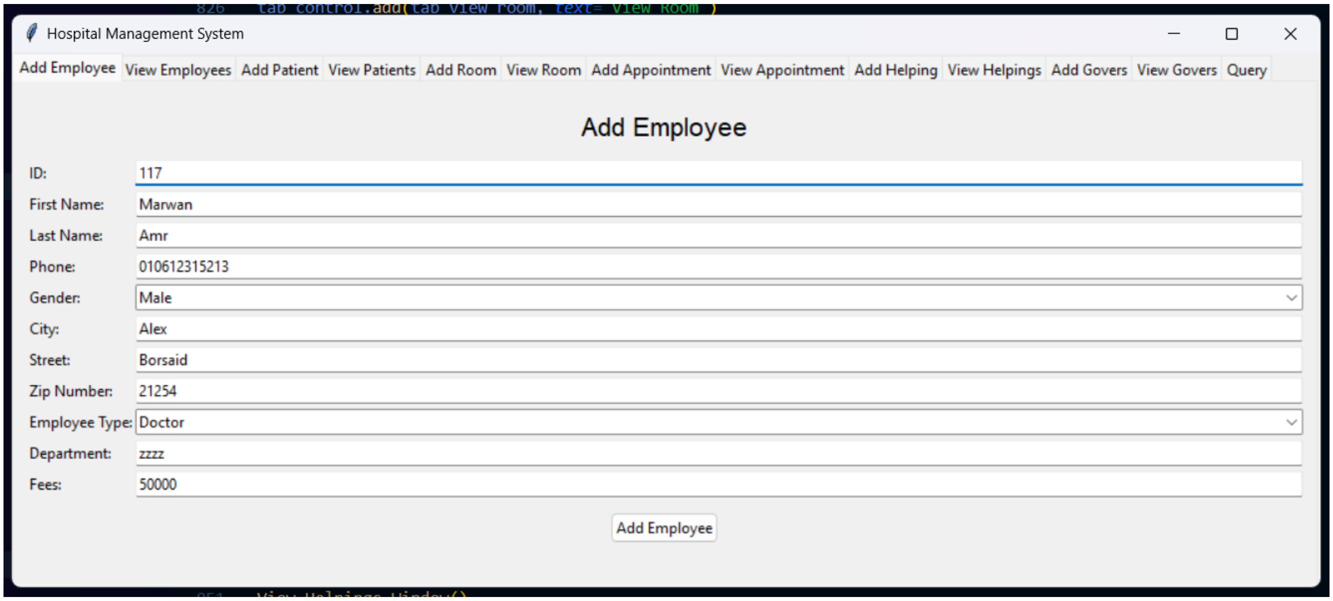


A login page for the GUI System



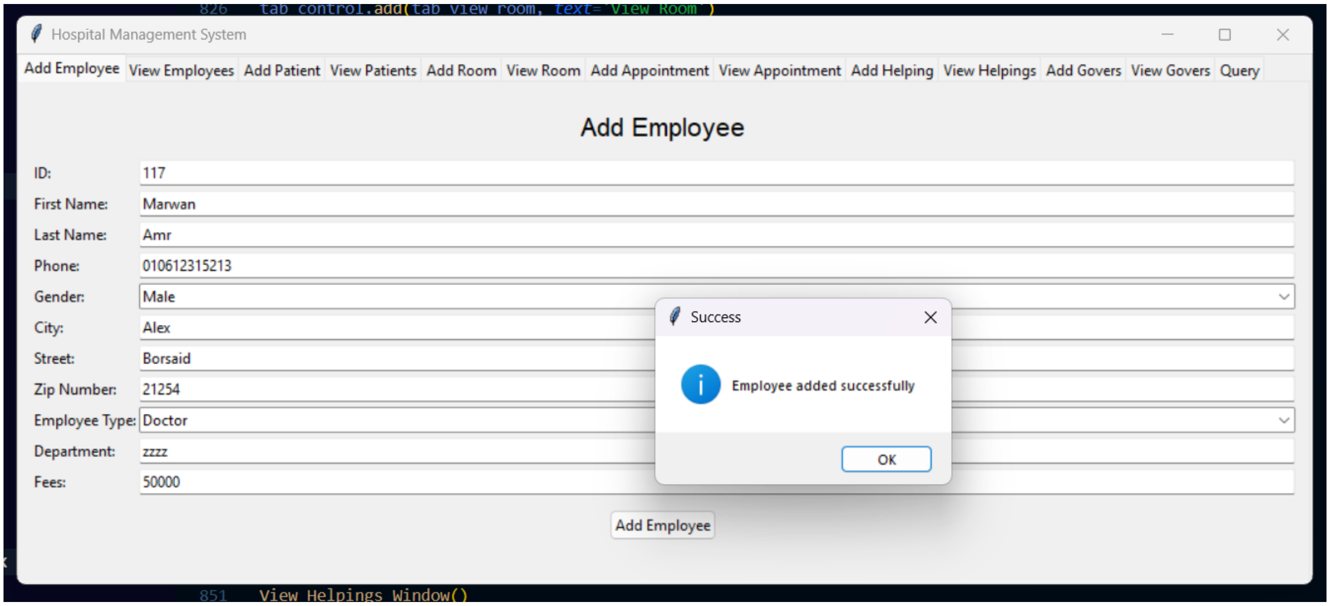
Username:	admin
Password:	********

After login … we got access to the GUI RDBM System.



Add Employee

ID:	117
First Name:	Marwan
Last Name:	Amr
Phone:	010612315213
Gender:	Male
City:	Alex
Street:	Borsaid
Zip Number:	21254
Employee Type:	Doctor
Department:	zzzz
Fees:	50000



```
tab_control.add(tab_view_room, text='View Room')
```

Add Employee

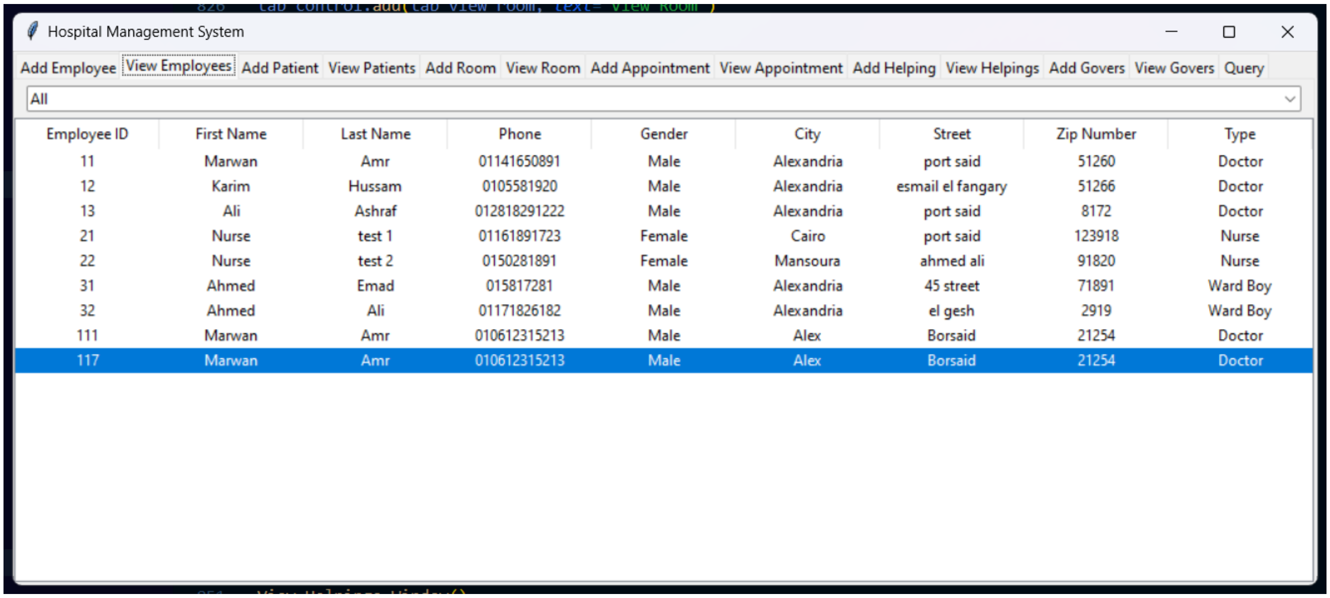
ID:	117
First Name:	Marwan
Last Name:	Amr
Phone:	010612315213
Gender:	Male
City:	Alex
Street:	Borsaid
Zip Number:	21254
Employee Type:	Doctor
Department:	zzzz
Fees:	50000

Success

Employee added successfully

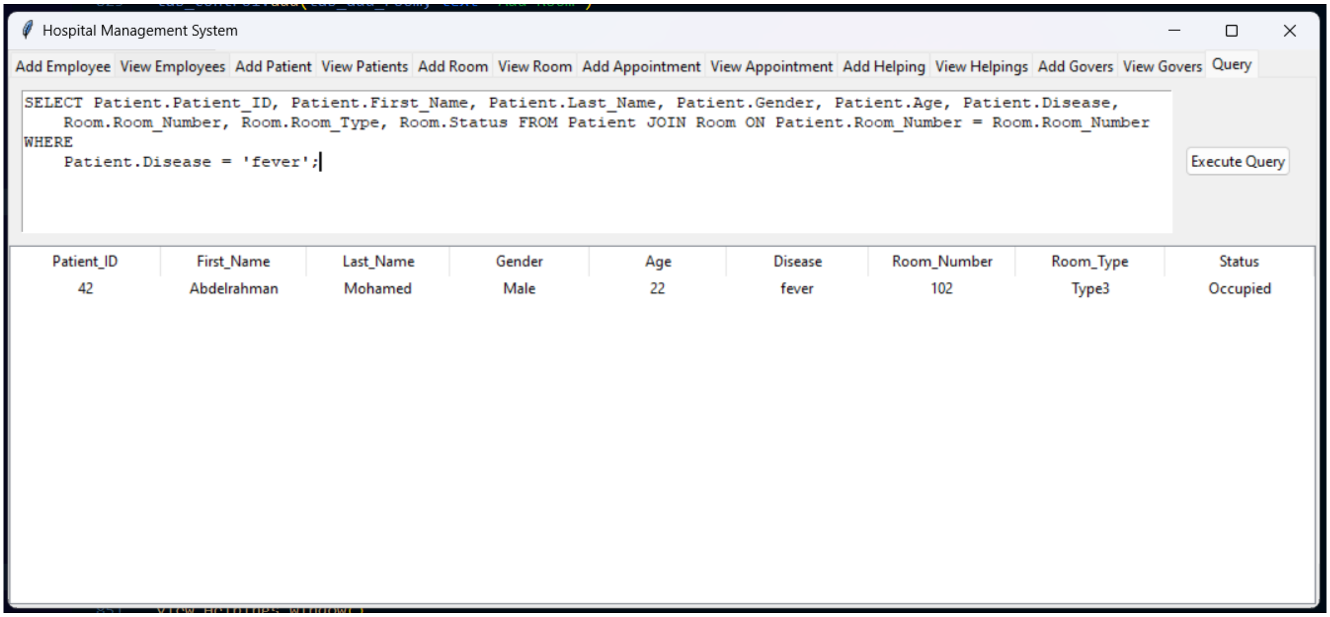
```
View_Helpings_Window()
```

We inserted some data to test the Database System. (A Doctor)



Employee ID	First Name	Last Name	Phone	Gender	City	Street	Zip Number	Type
11	Marwan	Amr	01141650891	Male	Alexandria	port said	51260	Doctor
12	Karim	Hussam	0105581920	Male	Alexandria	esmail el fangary	51266	Doctor
13	Ali	Ashraf	012818291222	Male	Alexandria	port said	8172	Doctor
21	Nurse	test 1	01161891723	Female	Cairo	port said	123918	Nurse
22	Nurse	test 2	0150281891	Female	Mansoura	ahmed ali	91820	Nurse
31	Ahmed	Emad	015817281	Male	Alexandria	45 street	71891	Ward Boy
32	Ahmed	Ali	01171826182	Male	Alexandria	el gesh	2919	Ward Boy
111	Marwan	Amr	010612315213	Male	Alex	Borsaid	21254	Doctor
117	Marwan	Amr	010612315213	Male	Alex	Borsaid	21254	Doctor

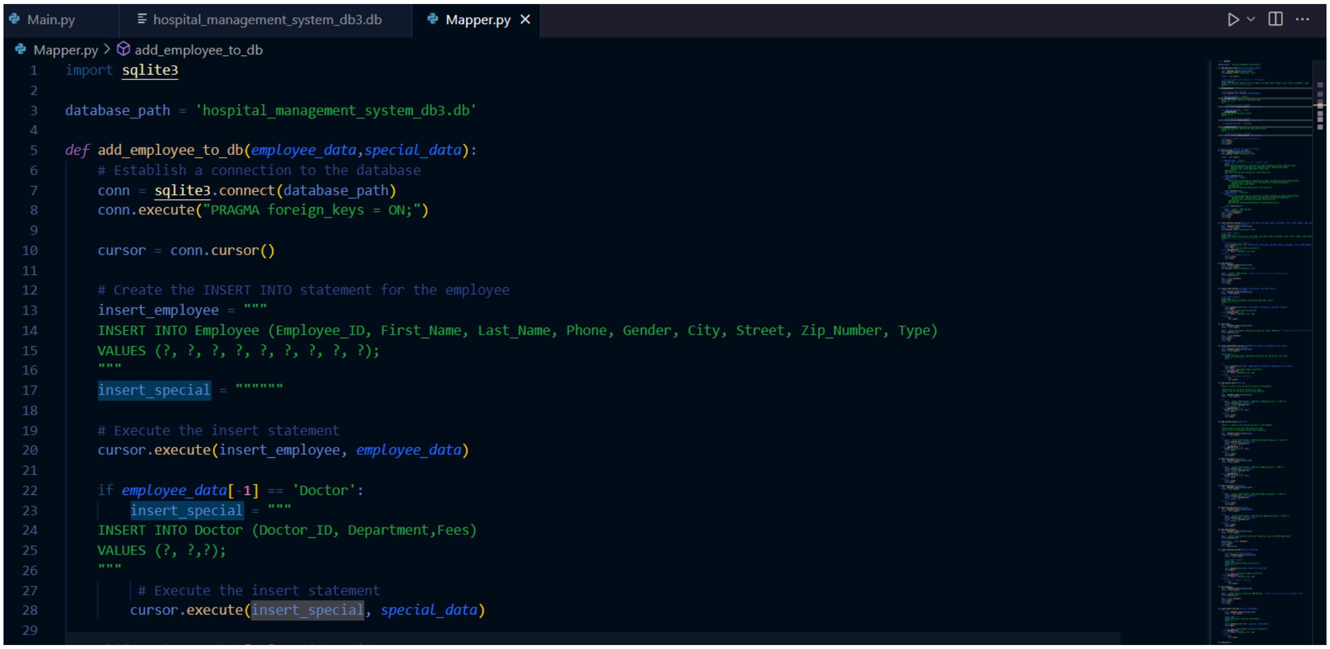
We have mainly 2 sections for each Entity one for insertion and one for viewing the tables. (tables viewing need to be refreshed after insertion to visualize the insertion process in the GUI)



```
SELECT Patient.Patient_ID, Patient.First_Name, Patient.Last_Name, Patient.Gender, Patient.Age, Patient.Disease,
    Room.Room_Number, Room.Room_Type, Room.Status FROM Patient JOIN Room ON Patient.Room_Number = Room.Room_Number
WHERE
    Patient.Disease = 'fever';
```

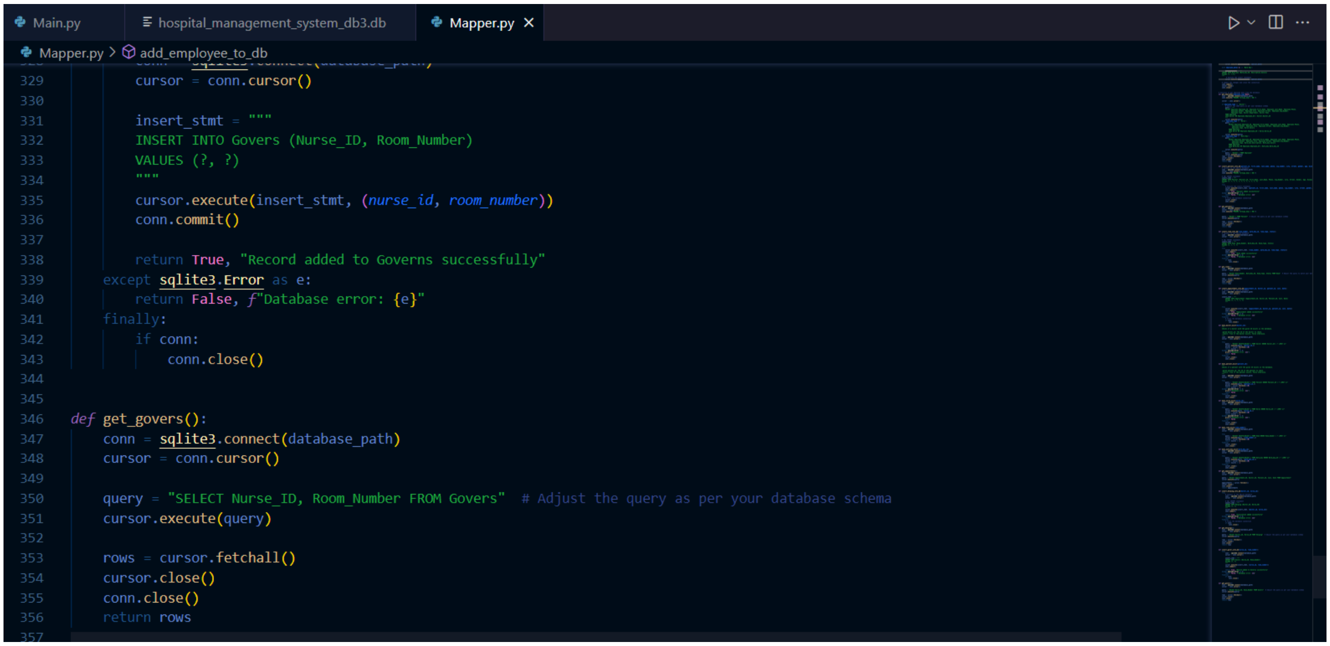
Patient_ID	First_Name	Last_Name	Gender	Age	Disease	Room_Number	Room_Type	Status
42	Abdelrahman	Mohamed	Male	22	fever	102	Type3	Occupied

We also added a section where we can execute separated queries from the preconfigured GUI for insertion and viewing.


Main.py
hospital_management_system_db3.db
Mapper.py
Mapper.py > add_employee_to_db

```python
import sqlite3

database_path = 'hospital_management_system_db3.db'

def add_employee_to_db(employee_data,special_data):
    # Establish a connection to the database
    conn = sqlite3.connect(database_path)
    conn.execute("PRAGMA foreign_keys = ON;")

    cursor = conn.cursor()

    # Create the INSERT INTO statement for the employee
    insert_employee = """
    INSERT INTO Employee (Employee_ID, First_Name, Last_Name, Phone, Gender, City, Street, Zip_Number, Type)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);
    """
    insert_special = """"""

    # Execute the insert statement
    cursor.execute(insert_employee, employee_data)

    if employee_data[-1] == 'Doctor':
        insert_special = """
    INSERT INTO Doctor (Doctor_ID, Department,Fees)
    VALUES (?, ?,?);
    """
        # Execute the insert statement
        cursor.execute(insert_special, special_data)

```

And for the back-end code using python and sqllite3 library.


Main.py
hospital_management_system_db3.db
Mapper.py ×
Mapper.py > add_employee_to_db

```python
        cursor = conn.cursor()

        insert_stmt = """
        INSERT INTO Govers (Nurse_ID, Room_Number)
        VALUES (?, ?)
        """
        cursor.execute(insert_stmt, (nurse_id, room_number))
        conn.commit()

        return True, "Record added to Governs successfully"
    except sqlite3.Error as e:
        return False, f"Database error: {e}"
    finally:
        if conn:
            conn.close()


def get_govers():
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    query = "SELECT Nurse_ID, Room_Number FROM Govers"  # Adjust the query as per your database schema
    cursor.execute(query)

    rows = cursor.fetchall()
    cursor.close()
    conn.close()
    return rows
```

Here is the end of the code. (Can be accessed in detail in the Python code).

Creation SQL Queries

```sql
-- Employee table
CREATE TABLE IF NOT EXISTS Employee (
    Employee_ID INTEGER PRIMARY KEY,
    First_Name TEXT,
    Last_Name TEXT,
    Phone TEXT,
    Zip_Number TEXT,
    City TEXT,
    Street TEXT,
    Gender TEXT,
    Type TEXT
);
```

```
-- Doctor table
CREATE TABLE IF NOT EXISTS Doctor (
   Doctor_ID INTEGER PRIMARY KEY,
   Department TEXT,
   Fees INTEGER,
   FOREIGN KEY (Doctor_ID) REFERENCES Employee (Employee_ID)
);

-- Nurse table
CREATE TABLE IF NOT EXISTS Nurse (
   Nurse_ID INTEGER PRIMARY KEY,
   Salary REAL,
   FOREIGN KEY (Nurse_ID) REFERENCES Employee (Employee_ID)
);

-- Ward boy table
CREATE TABLE IF NOT EXISTS Ward_boy (
   Ward_boy_ID INTEGER PRIMARY KEY,
   Description TEXT,
   Salary REAL,
   FOREIGN KEY (Ward_boy_ID) REFERENCES Employee (Employee_ID)
);

-- Room table
CREATE TABLE IF NOT EXISTS Room (
   Room_Number INTEGER PRIMARY KEY,
   Ward_boy_ID INTEGER,
   Room_Type TEXT,
   Status TEXT,
   FOREIGN KEY (Ward_boy_ID) REFERENCES Ward_boy (Ward_boy_ID)
);
-- Patient table
CREATE TABLE IF NOT EXISTS Patient (
   Patient_ID INTEGER PRIMARY KEY,
   First_Name TEXT,
   Last_Name TEXT,
   Phone TEXT,
   Zip_Number TEXT,
   City TEXT,
   Street TEXT,
   Gender TEXT,
   Age INTEGER,
   Disease TEXT,
   Birthdate TEXT
);

-- Appointment table
CREATE TABLE IF NOT EXISTS Appointment (
   Appointment_ID INTEGER PRIMARY KEY,
   Doctor_ID INTEGER,
   Patient_ID INTEGER,
   Cost REAL,
   Date TEXT,
   FOREIGN KEY (Doctor_ID) REFERENCES Doctor (Doctor_ID),
   FOREIGN KEY (Patient_ID) REFERENCES Patient (Patient_ID)
);
```

```
-- Helping table (Association table for Doctor and Nurse)
CREATE TABLE IF NOT EXISTS Helping (
    Doctor_ID INTEGER,
    Nurse_ID INTEGER,
    PRIMARY KEY (Doctor_ID, Nurse_ID),
    FOREIGN KEY (Doctor_ID) REFERENCES Doctor (Doctor_ID),
    FOREIGN KEY (Nurse_ID) REFERENCES Nurse (Nurse_ID)
);

-- Govers table (Association table for Nurse and Room)
CREATE TABLE IF NOT EXISTS Govers (
    Nurse_ID INTEGER,
    Room_Number INTEGER,
    PRIMARY KEY (Nurse_ID, Room_Number),
    FOREIGN KEY (Nurse_ID) REFERENCES Nurse (Nurse_ID),
    FOREIGN KEY (Room_Number) REFERENCES Room (Room_Number)
);
```

Indexing SQL Queries

```
CREATE INDEX idx_Appointment_Doctor_ID ON Appointment (Doctor_ID);
CREATE INDEX idx_Appointment_Patient_ID ON Appointment (Patient_ID);
CREATE INDEX idx_Helping_Doctor_ID ON Helping (Doctor_ID);
CREATE INDEX idx_Helping_Nurse_ID ON Helping (Nurse_ID);
CREATE INDEX idx_Govers_Nurse_ID ON Govers (Nurse_ID);
CREATE INDEX idx_Govers_Room_Number ON Govers (Room_Number);
```

Selection SQL Queries

```
SELECT Patient.Patient_ID,
       Patient.First_Name,
       Patient.Last_Name,
       Patient.Gender,
       Patient.Age,
       Patient.Disease,
       Room.Room_Number,
       Room.Room_Type,
       Room.Status
FROM Patient
JOIN Room ON Patient.Room_Number = Room.Room_Number
WHERE Patient.Disease = 'fever';
```

This SQL query retrieves information about patients who have fever. It combines data from two tables: `Patient` and `Room`. The query shows each patient's ID, name, gender, age, disease, and their room details like room number, type, and status. It only includes patients whose disease is listed as 'Fever' and shows their room.

```
SELECT Disease, AVG(Age) AS Average_Age
       FROM Patient GROUP BY Disease;
```

This SQL query calculates the average age of patients for each different disease listed in the `Patient` table. It shows two things: the disease and the corresponding average age of patients with that disease.

```
SELECT
    Patient.Patient_ID, Patient.First_Name AS Patient_First_Name,
    Patient.Last_Name AS Patient_Last_Name,
    Nurse.Nurse_ID, Nurse.Salary AS Nurse_Salary, Ward_boy.Ward_boy_ID,
    Ward_boy.Description AS Ward_boy_Description,
    Ward_boy.Salary AS Ward_boy_Salary FROM Patient
LEFT JOIN Room ON Patient.Room_Number = Room.Room_Number
LEFT JOIN Govers ON Govers.Room_Number = Room.Room_Number
LEFT JOIN Nurse ON Govers.Nurse_ID = Nurse.Nurse_ID
LEFT JOIN Ward_boy ON Ward_boy.Ward_boy_ID = Room.Ward_boy_ID;
```

This SQL query lists each patient's information along with the details of the nurse and ward boy assigned to their room. It includes patients even if they don't have an assigned nurse or ward boy. You get the patient's name and ID, the nurse's ID and salary, and the ward boy's ID, description, and salary.

```
SELECT
    Doctor.Doctor_ID, Employee.First_Name, Employee.Last_Name,
    COUNT(Appointment.Appointment_ID) AS Appointment_Count FROM
    Doctor JOIN Employee ON Doctor.Doctor_ID = Employee.Employee_ID
LEFT JOIN Appointment ON Doctor.Doctor_ID = Appointment.Doctor_ID
GROUP BY Doctor.Doctor_ID
HAVING COUNT(Appointment.Appointment_ID) > 2;
```

This SQL query finds doctors with more than two appointments. It shows each doctor's ID, first name, and last name, along with their total number of appointments. The data is combined from the Doctor, Employee, and Appointment tables. Only doctors who have over two appointments are listed.

```
SELECT
        Appointment_ID,
        Doctor_ID,
        Patient_ID,
        Cost,
        Date
FROM
        Appointment
WHERE
        Date BETWEEN DATE(‘now’) AND DATE(‘now’, ‘+7 days’);
```

This SQL query lists appointments scheduled for the next 7 days. It shows the appointment ID, doctor ID, patient ID, cost, and date for each appointment from the `Appointment` table. Only appointments between today and the next 7 days are included.

We Started by making the business rule to assist us in making the ERD diagram then we quickly realized the need of an enhancement as it got so complicated with all the relations and Entities we wanted to add, we thought about implementing the Generalization/Specialization part from chapter 4.

After that we started working on the EERD of the hospital management system and the relational model (all illustrations using Draw.io).

Last thing we did was Implement our work in Oracle and insert data into the created tables to test the accuracy and make sure data is inserted properly in each table with no errors.

If there is one thing, we learnt from this learning experience it is that.

> ***“Many ideas grow better when transplanted into another mind than the one where they sprang up”***