iDigiHealth-Patient Care Information System Design and Implementation

Project Report

Roll No. PGDBDB202002012
Health Informatics (Elective - PGDBD-105B)

Dr. Ashalatha Sreshty
P.G. Diploma in Big Data Biology
Institute of Bioinformatics and Applied Biotechnology
Electronic City, Bangalore

iDigiHealth-Patient Care Information System - Design and Implementation

INTRODUCTION

Patient care is the prime focus of many clinical disciplines like medicine, nursing, pharmacy, nutrition, therapies such as respiratory, physical, and occupational, and others. Patient care information systems (PCISs) has become the core building blocks for a safer health care system [1]. PCISs are applications that support the health care process by allowing health care professionals or patients direct access to order entry systems, medical record systems, radiology information systems, patient information systems, and so on [2]. The potential of PCIS to overcome current breakdowns and inefficiencies in patient information processes has leveraged information in several ways. Patient health records (PHRs) are an extension of traditional electronic health records (EHRs) and emerged in the early 1970s to increase patient empowerment and engagement. PHRs created a patient-centeric platform to enable continuity of care, error reduction, treatment choice, and patient-provider partnership building [3, 4]. Recent years has evidenced as surge in the research and implementation of patient-centered PHR systems to enable information sharing and collaboration, with the goal of improving health outcomes and reducing costs.

The meaningful use of PHR encouraged the integration of technology into medical practice, making vast amounts of patient data available electronically. In later stages, the program focused on empowering patients by providing them with online access to their heath data. An increasing stride in the usage of PHRs can be attributed to the rise in mobile computing and patient's technical aptitude. The EHR data comprised laboratory results and summary of care and patient-generated data such as symptoms. Tremendous amounts of patient data are now available through PHR systems. With patients' permission, these data, along with the application of advanced data mining and machine learning, can provide significant new

opportunities in research. For instance, models in areas such as disease prediction, patient risk assessment, and early symptom detection can now be improved, leading to major advances in health outcomes and cost optimization. However, along with new opportunities provided by PHR systems come data and user-related challenges. Data-related issues such as quality, privacy, and security pertain to collection, safe storage, and processing of large quantities of patient data from distributed information systems. Implementation of PCIS is to solve the issue of poor practice or to improve the quality of health care.

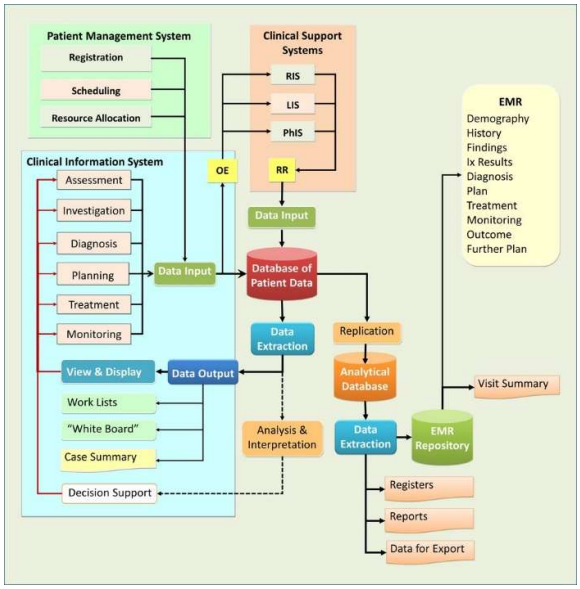
The major components of the PCIS is the easy networking of patient related data from appointments in the care centers to storing the laboratory results, analysing the data and making it accessible to care givers, patients and other third-party users for societal benefit. Recent evolution of technology has also included patient-centered decision support system that combined a comprehensive individual patient information and an aggregation of patient records to provide personalized patient recommendations for an improved and efficient diagnosis [5].

Although Patient Care Information System is attractive and acceptable, unfortunately is less advocated. Hence, this project is an effort to design and model patient-centered information systems, especially focussed on patients suffering with leukemia. Leukemia is a malignancy caused due to abnormal alterations in the normal cell regulatory processes cause uncontrolled proliferation of hematopoietic stem cells in the bone marrow. Leukemia is unregulated and rapid proliferation of leukemic cells results in replacements of normal hematopoietic precursor cells of erythroid, megakaryocytic, myeloid or lymphoid lineage by proliferating leukemic cells in bone marrow. Leukemia is classified by the type of white blood cells affected and by how quickly the disease progresses. In terms of how quickly it develops or gets worse, leukemia is classified as either acute (fast-growing) or chronic (slow-growing). Acute leukemia is rapidly progressing and results in the accumulation of immature, functionless blood cells in the bone marrow. With this type of leukemia, cells reproduce and

build up in the marrow, decreasing the marrow's ability to produce enough healthy blood cells. Chronic leukemia progresses more slowly and results in the accumulation of relatively mature, but still abnormal, white blood cells. Different types of leukemia are : (i) Acute myeloid leukemia, (ii) Acute lymphocytic leukemia, (ii) Acute myeloid leukemia and (iv) Chronic lymphocytic leukemia. Monitoring of the disease symptoms, treatment plan are key to diagnosis and patient-centered care respects and responds to individual differences in patient preferences, needs, and values.

The patient care information system consists of (i) Patient Information System that keeps the record of demographic data and other patient related personal information, (ii) Clinical Information system that includes clinical decision system, electronic medical record (EMR) and (iii) Clinical Support system that constitutes laboratory support system, blood banking information system, radiology information system and other support systems and (iv) patient information database management system.

In this project, a prototype of patient-centeric information system (PCIS) was designed and developed to store, manage and access the patient data during various requirements. The relationships of the systems in this application is depicted as chart below:



*Source: Derivation of EMR from Various Systems of HIS and the Database of Patient Data (Patient Information Database) (*https://drdollah.com/hospital-information-system-his/)

OBJECTIVES

(i) Design and development of relational database management system to store the electronic health records of the patient

(ii) Design and develop graphical user interface of the PCIS application

(iii) Develop Clinical information system to enable data visualization of the laboratory results as graphs

METHODOLOGY

1) Design of PCIS system

Based on the literature, the various components and contents of the PCIS application was designed. A framework of the prototype of the application was drafted and its possibility of connectivity between different other departments of the information systems was checked to construct the relational database.

2) Development of back-end relational database management system

Relational database system to store, manage and access patient-centeric information system was developed using MySQL, MySQL workbench developed and provided by ORACLE. MySQL is the world's most popular open source database for cost-effectively delivering reliable, high performance and scalable e-commerce, online transaction processing, and embedded database applications. It is an integrated, transaction safe, ACID-compliant database with full commit, rollback, crash recovery, and row-level locking capabilities. MySQL delivers the ease of use, scalability, and high performance, as well as a full suite of database drivers and visual tools to help developers and DBAs build and manage their MySQL applications.

MySQL Workbench is a unified visual development and administration platform that includes advanced tools for database modeling and design, query development and testing, server

configuration and monitoring, user and security administration, backup and recovery automation, audit data inspection, and wizard-driven database migrations.

MySQL Connector Python is installed to establish connection between the database and the python script to interface with the database management system. MySQL Connector Python is written in pure Python, and it is self-sufficient to execute database queries through python.

3) Development of Front-end Graphical User Interface (GUI)

The front-end GUI for easy interface with relational database was developed using Python programming language and tkinter module. Python is open source software. It is a dynamic language, and it has all the support to build large and complex enterprise database-centric applications using MySQL. Python integrated with several third-party modules facilitate the integration of MySQL with the tkinter GUI. Sublime Text and Git Bash editor was used to write and execute the code.

4) Computational system configuration

To design and develop the PCIS application, SONY VAIO laptop (Model: SVF14N25CXB) was used. The application was developed on Microsoft Windows 10 Pro Operating system, Intel® Core™ i5-4200U CPU @ 1.6 GHz, 8 GB RAM and 600 GB Hard Disk Space.

RESULTS AND DISCUSSION

Many healthcare centres have their own separated systems leading to the lack of communications and the inefficient data sharing. For instance, finance department uses simple EXCEL spreadsheets to record the payment information, in the clinic department, the doctors write prescriptions for the patients and keep paper documents, do not have any information about the patients' insurance plans, the medicine department has to keep the prescription and inventory records on their own computer system. While each system serves a distinctive purpose, there is no coordinating, assimilating and representing of data.

In view of these disadvantages of the current system, a healthcare management system is proposed. Healthcare management system is a database management system (DBMS), which is based on computer networks, using the advanced database technology to construct, maintain, and manipulate various kinds of data in a database system (DBS). The DBMS can track and update all the information of recorded patients in the healthcare center during a particular time span. The major advantages of the DBMS are easy to retrieve and update information, efficient data sharing and communication, and reliable backup and security

A prototype of the PCIS application which provides a graphical user interface to store, manage and access the patient information in the relational database is developed. iDigiHealth is a patient centered care system, which facilitates the health care center to retrieve, update, and report the patient information efficiently, in turn helping the doctors make timely, effective diagnoses of blood cancer patients. The major framework of the PCIS application includes Patient information system, Clinical support system and Clinical information systems as discussed earlier. A schematic representation the iDigiHealth Application is provided in Figure 2.

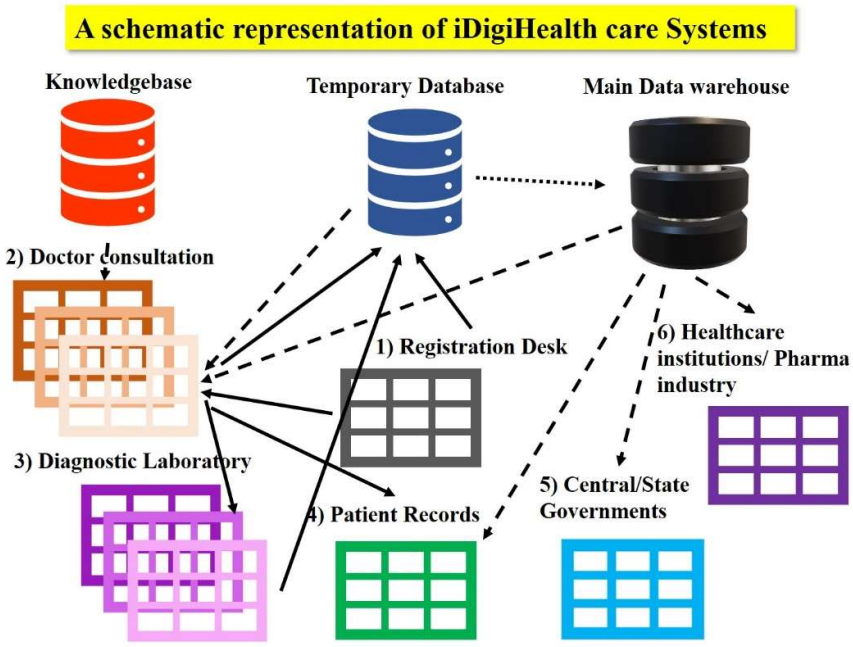


Figure 2: Schematic representation of the relational database management system of iDigiHealth care system for the leukemia patients.

iDigiHealth application interconnects and facilitates transfer of data between different departments at the health care providing centres. The schema shows the storing of data generated on daily basis to a temporary database, which then transfer the data to analytical database which facilitate the access of patient data to all the departments like registration, finance, medical consultation and other third party user like government surveillance system and health and pharma industries.

Development of relational database management system

A relational database management system was designed and constructed using the MySQL and MySQL workbench. The relational database is interfaced to python script thorugh the MySQL connector python. Figure 3 shows the schema of database-connector-python network that will facilitate the execution of queries and perform operations.

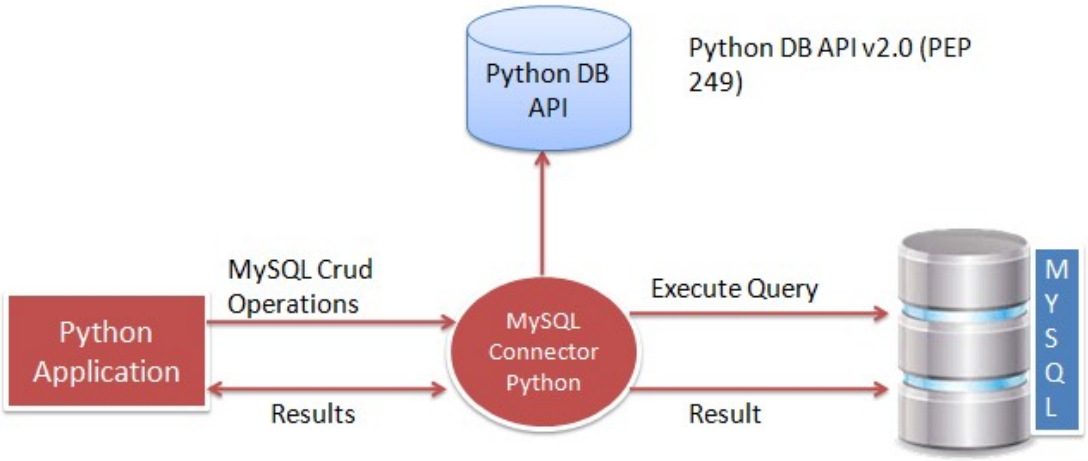


Figure 3: Diagrammatic representation of the Python MySQL database programming

Two different databases were generated, one for storing the day-to-day patient information temporarily and then transferring the data to an analytical database. The registration department, which registers the patient information and provides appointments with the medical consultants constitute the patient information system. This department registers the personal details of the new patient and in case of already existing patient record, fixes the appointment with concerned medical consultant for diagnosis. Figure 4 is the snapshot of the relational database designed and generated. The primary or temporary database has several separate tables to store the information input from patient information system (patient details, appointment, doctor details), clinical support system (haematology lab, imaging lab and molecular genetics laboratory) and clinical information system (medical consultation). The secondary of analytical database is a permanent storage of the patients health records (PHR), which further facilitates access to the medical consultants, patients themselves, government and health industry for specific retrieval of the patient information. Figure 5 show the schema of the database design and development of the analytical database.

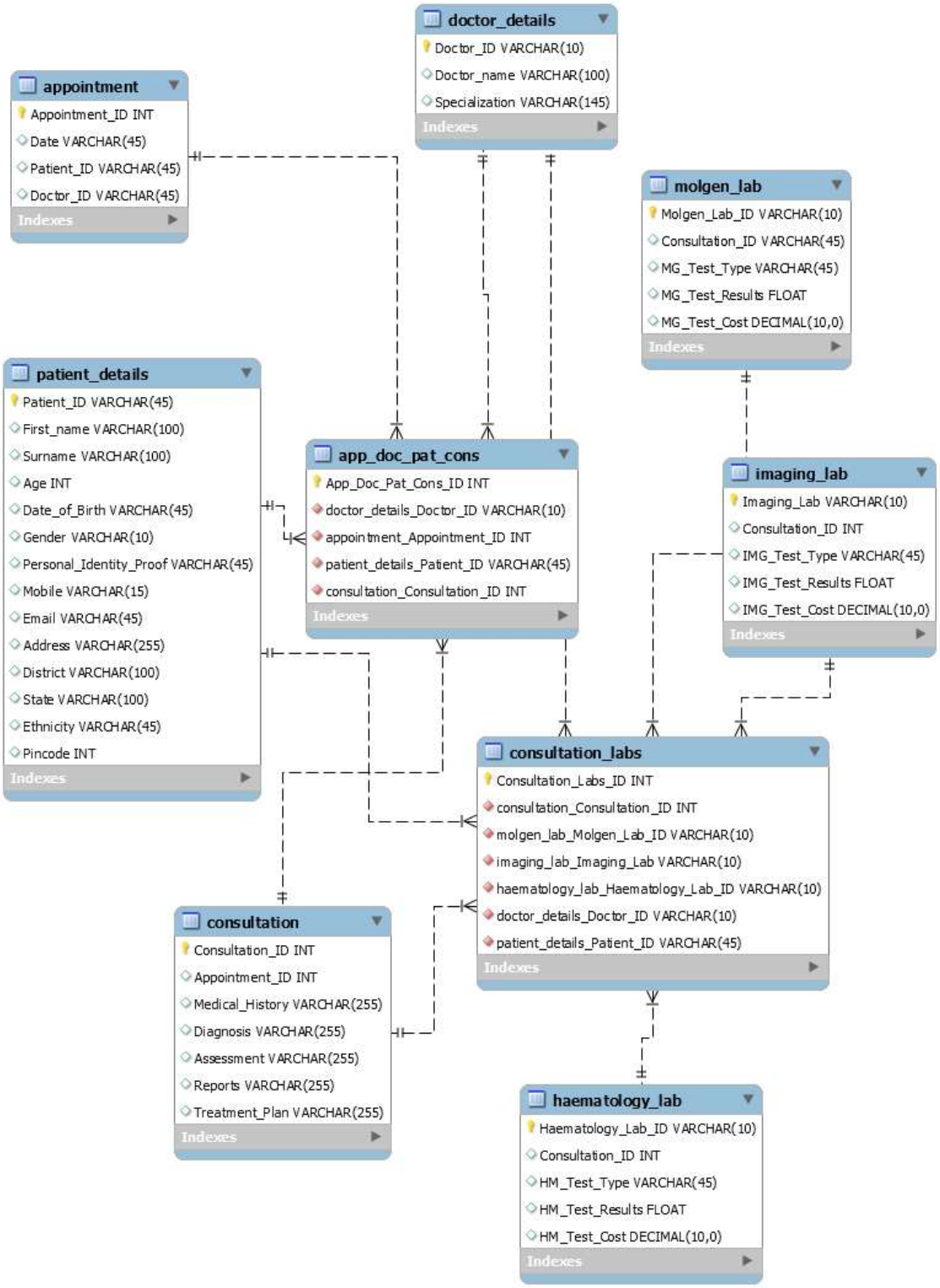


Figure 4: Schema of the relational database management systems developed in MySQL workbench for the primary or temporary database

Figure 5: Schema of the relational database management systems developed in MySQL workbench for the analytical database for permanent storage of PHRs

Patient Information System (PIS)

PIS manages the registry of clients or customers of a health care facility through The Patient Registration Application. This application software is used to enlist a new person as a client (patient) of the healthcare institution. The functions include to capture identification and demographic data, create a new entity with a unique identification number in the Database of Patients. maintain a single Medical Record for the patient and maintain a master list of patients as a permanent register i.e. the Client Register also known as "Patient Register" or the "Master Patient Index" (PMI). The identification number is a unique person identifier and which may also act as the Medical Record Number (MRN). It allows data regarding a single patient to be constituted as a single record, shared between systems and used for subsequent visits and encounters without the need for repeated data acquisition and entry of static data. Before giving the number it is essential that the patient is properly identified i.e. the patient is actually the person he/she claims to be. This is usually done by comparing with the identification data already available on his/her identity card/passport.

PIS inputs the patient's personal details, fixes appointments with the available doctors in duty. Figure 6 is the graphical user interface of PIS which facilitates the data input of the newly registered patient. The GUI interacts with the primary database through python script and tkinter module. The python code is provided in the appendix section of this report. The registration of patient personal details include name, surname, age, date of birth, gender, contact information, address, ethnicity and personal identification proof. A unique patient ID is created to identify the patient and store the details of medical consultation and diagnosis reports from various laboratories, which are linked by defining relations between the tables.

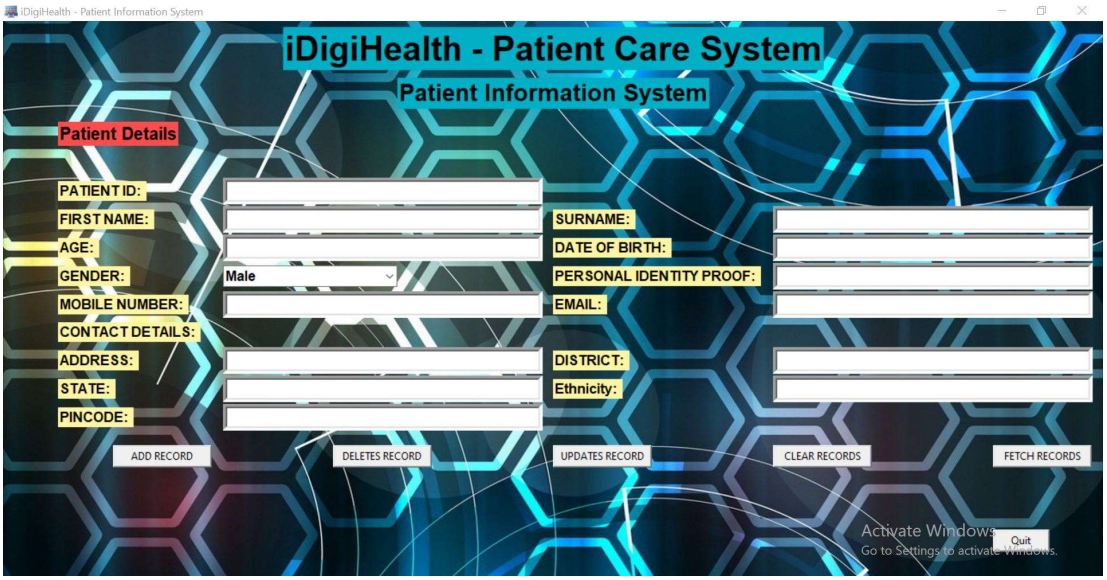


Figure 6: Snapshot of the graphical user interface to facilitate the input of patient personal and sensitive information by the patient information system

After registration of the patient details appointment with the duty doctor is fixed through a graphical interface which connects to the appointment table of the primary database (Figure 5). The snapshot of the GUI for appointment fixing was provided in Figure 7 and the entering the doctor details in Figure 8.

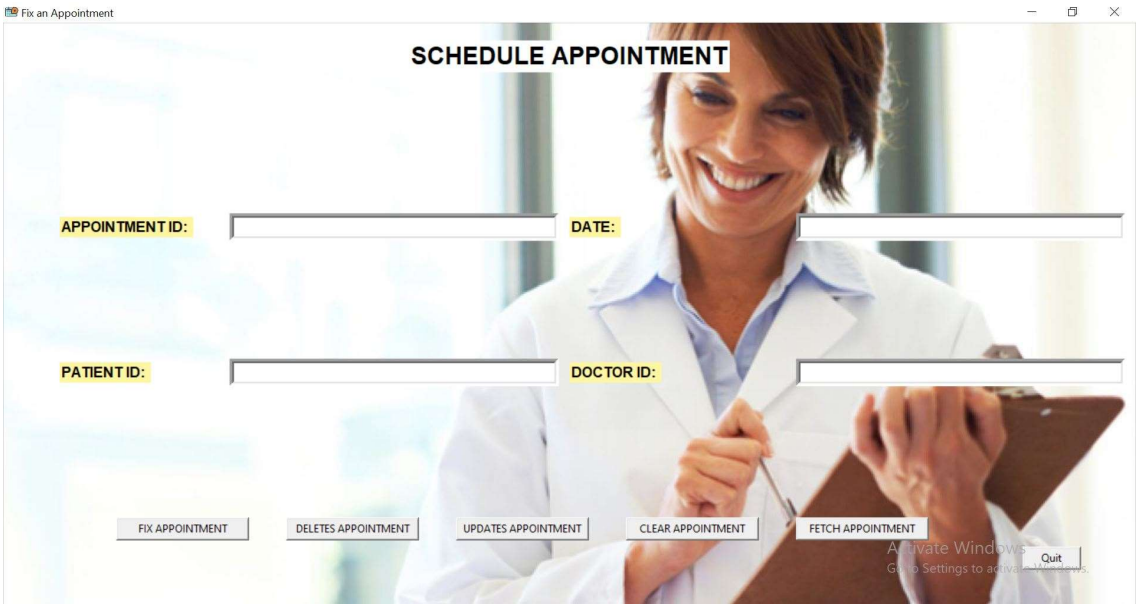


Figure 7: Screen show the GUI to fix appointment for the patients with the duty doctors

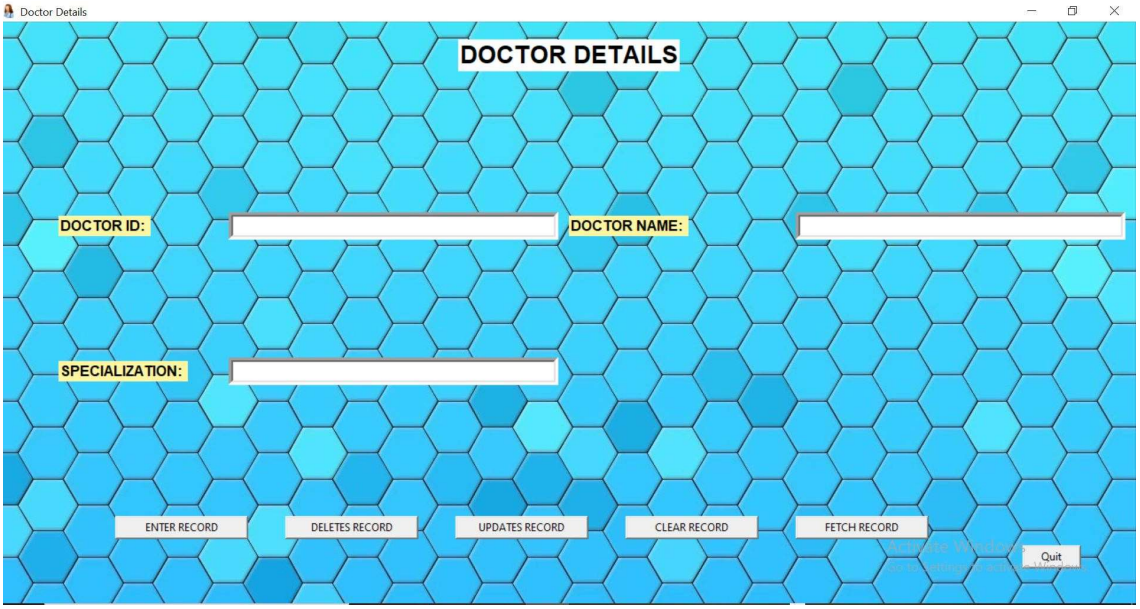


Figure 8: Screenshot of the GUI interface to enter doctor details and retrieve records

Clinical Information System (CIS)

The functions of Clinical Information System (CIS) is a part of the Hospital Information System (HIS) which facilitates direct patient care i.e. activities where care providers. CIS interacts face to face with patients and perform procedures that affect them physically, physiologically or psychologically. The clinical information system integrates the Laboratory Information System (LIS). An integral feature of LIS is the capability of sending messages and getting results from machines that perform tests automatically (analysers). The Clinical Information applications extract this data from the Patient Information Database and display them as individual results or charts for viewing by users. The functions of a laboratory in the context of the Patient Care Information System can be receiving orders, performing tests and provide results. There are variations in the tests and can be classified into biochemistry, immunology, haematology, anatomical pathology and microbiology. In this prototype of iDigiHealth care system, I have designed and developed GUI for haematology and imaging laboratory that are crucial for performing blood tests for the patients in regular basis and maintain their EMRs.

Figure 9 shows the snapshot the GUI for the haematology laboratory to input the results of various blood tests. Similarly, the imaging laboratory, where microscopy, x-ray, CTScan, etc facilities are provided for the patients, the results and image files are uploaded to the primary database through the graphical user interface, the screenshot of which is shown in Figure 10.

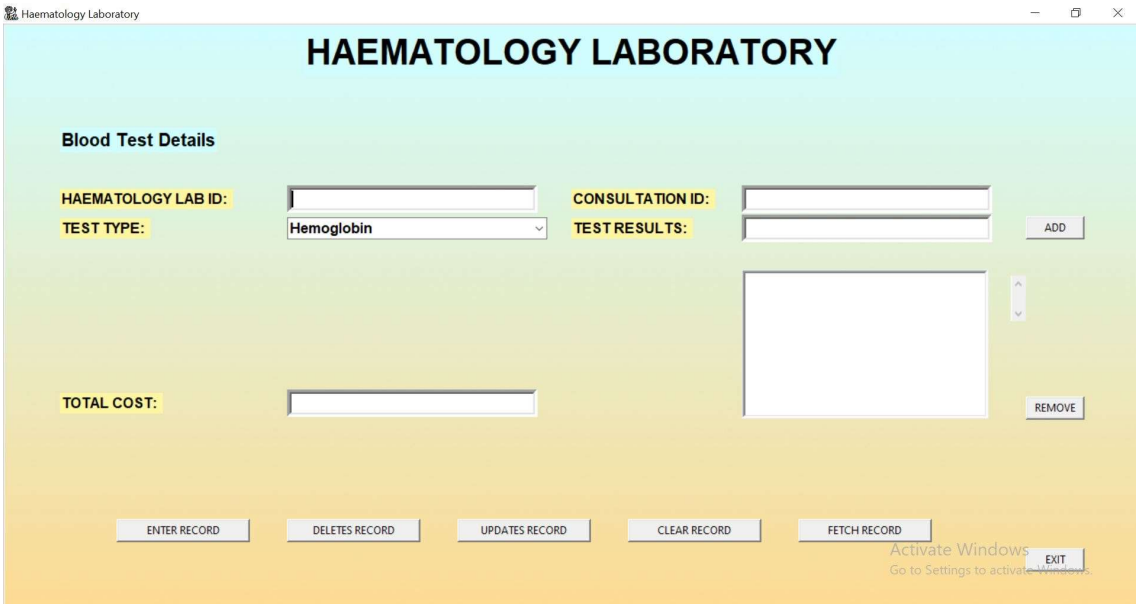


Figure 9: Screenshot of the GUI to enable data input for the haematology laboratory

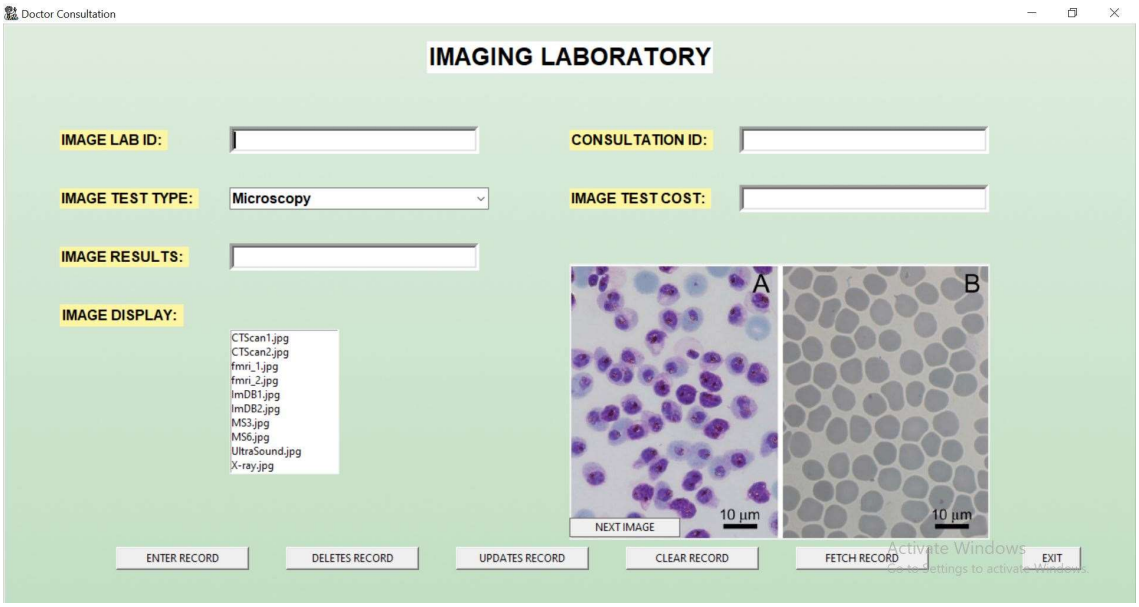


Figure 10: Screenshot the GUI for the Imaging laboratory

Clinical Support System (CSS)

Clinical support is envisaged not as a single system or application but as built-in functions within the whole patient care application or its components. CSS provides directions through care plans and the computer analysis and interpretation of results (normal, abnormal, scoring, stratification, grading, staging, comparison with standards for quality control). This mainly includes the doctors prescription, treatment plan based on the data analysis and visualization of the results obtained from various diagnostic test laboratories. Diagnosis is made by considering certain variables including signs, symptoms, clinical test results, investigation findings (laboratory, imaging, and endoscopy), monitoring parameters, clinical progress and response to treatment. Through research and experience the medical profession has identified sets of variables i.e. the criteria that predict a diagnosis. This knowledge can be presented to care providers to assist them in making a diagnosis. For this purpose, iDigiHealth facilitates assigning values to each criterion, calculate the score and present it in a calculated data field. A comparison is made with the accepted scores to determine the likelihood of the diagnosis. The doctor records the medical history of the patients, records their symptoms and prescribes several tests which are directed to the respective laboratories and retrieves the results and based on which formulates the treatment plan for the patient. Figure 11 show the GUI developed as a component of iDigiHealth care system.

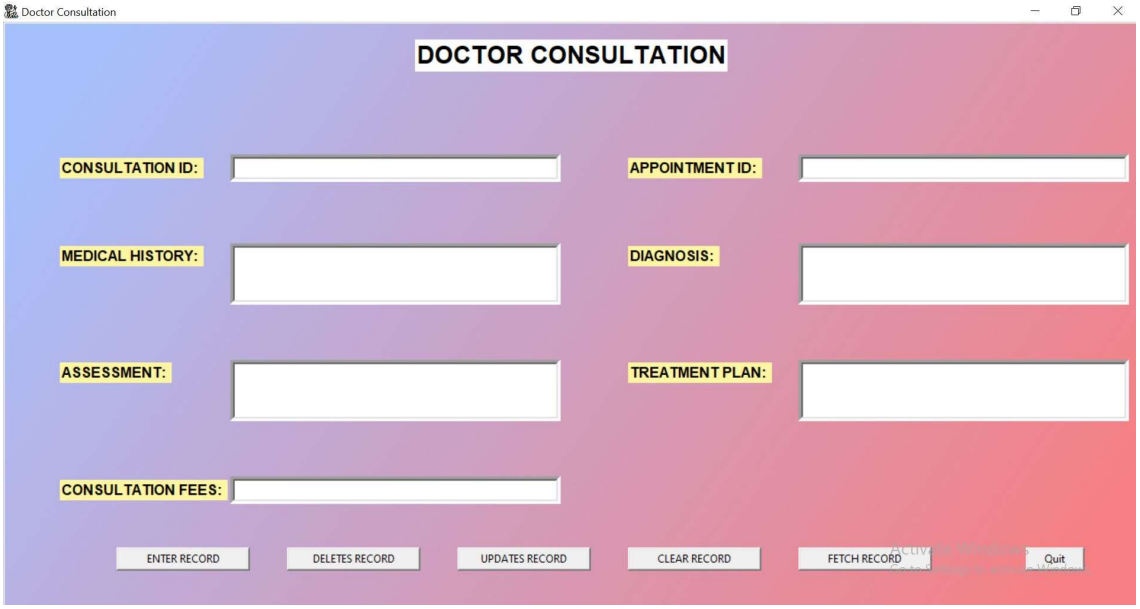


Figure 11: Screenshot of the GUI for doctor consultation to facilitate input diagnosis and treatment information specific to the patient.

iDigiHealth care system also facilitates the visualization of the laboratory results as charts. The GUI provides entry boxes to input the readings and plot along with the standard values to see the variations in the results in comparison with standard normal values as well as the previous laboratory tests (Figure 12).

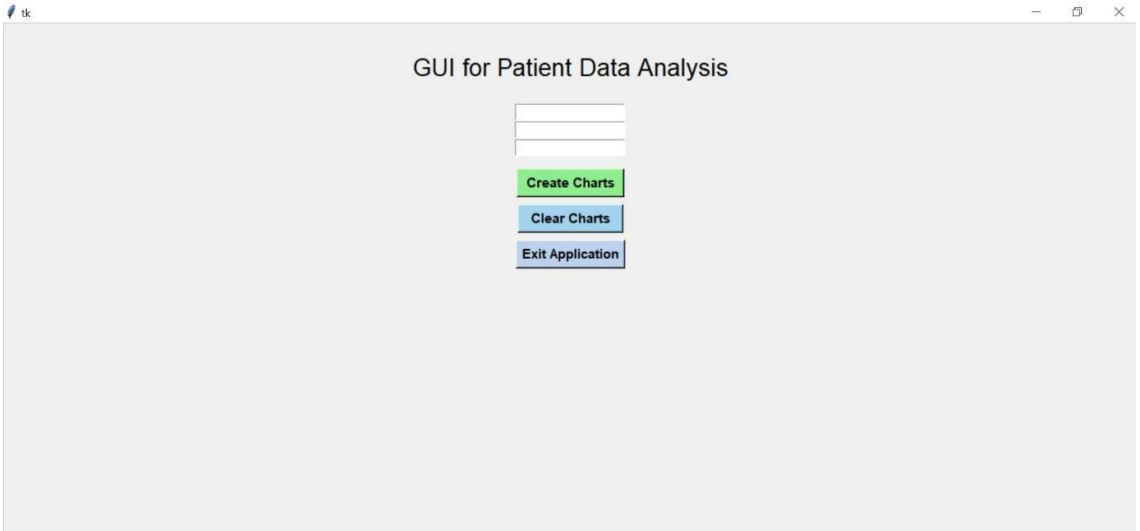


Figure 12: Screenshot of the GUI to enable data visualization of the laboratory results to generate bar and pie charts

Data transfer to the permanent analytical database of iDigiHealth system

iDigiHealth care system also integrates permanent database for analytical purpose. The daily data generated into this database - analytical-idigihealth-db, every end of the day for providing access to the patient's record to the doctor to study the progress in the patient's health by comparing the previous consultation report, plot the data, and formulate the treatment plan. Even, the patient is provided access to the database to access all his HER information to keep track of his health and personal information. The third party users are also provided access to this database to facilitate various purposes after masking the critical and sensitive data of the patients under the compliance of HIPAA. Figure 13 show the screenshot the analytical database, after transferring the data from the tables of primary database for permanent and safe storage of patient health records.

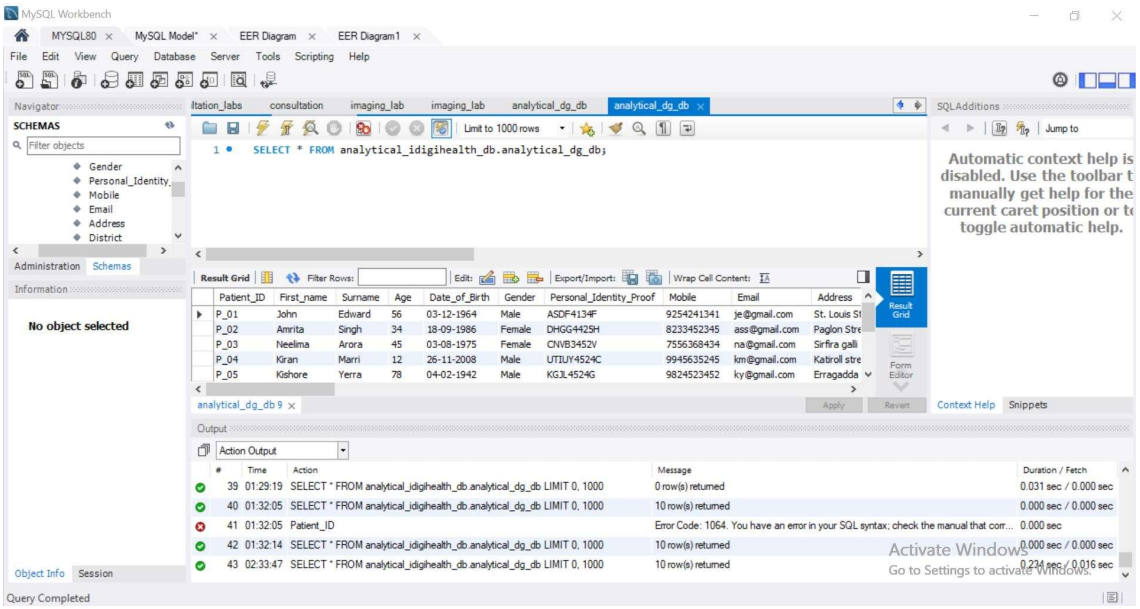


Figure 13: The screenshot show the data transferred to the analytical database for permanent storage of PHRs

Note: All the python codes used to develop iDigiHealth application is provided under the appendix section.

SUMMARY

iDigihealth care system is a prototype of the health information system centered for patient care, especially for the blood cancer patients. This system constitutes all the essential components of health information management system like patient information system to enable record patient details and fix appointment with the available duty doctors. The clinical information system integrates with the various diagnostic laboratories to record patient specific data like blood tests, imaging and molecular genetic analysis. The third component is the clinical support system which enables the doctor to access the laboratory results of the patient and plot the results as bar and pie charts to compare with the standard values and define a treatment plan. The doctor also records the medical history, symptoms, diagnosis and treatment plan prescribed for the patient into the database for future access of the data, even for the patient and other third party researchers.

In summary iDigiHealth is a patient centered care system that enables maintaining the EHRs, reduces the usage of paper-based documentation, quick and easy transfer of data between the departments and stores patients records to facilitate personalized therapy and research.

REFERENCES

1) Committee on Quality of Health Care in America Crossing the Quality Chasm: A New Health System for the 21st Century. Washington, DC: National Academy Press, 2001

2) Ash, J. S., Berg, M., & Coiera, E. (2004). Some unintended consequences of information technology in health care: the nature of patient care information system-related errors. *Journal of the American Medical Informatics Association : JAMIA*, *11*(2), 104–112.

3) Shenkin BN, Warner DC. Sounding board. Giving the patient his medical record: a proposal to improve the system. N Engl J Med. 1973 Sep 27;289(13):688–92.

4) Hinman E, Holloway J. The patient carried personal health record: a tool to increase patient participation in the treatment process. J Clin Comput. 1977;6(4):9.

5) Bouayad L, Ialynytchev A, Padmanabhan B. Patient Health Record Systems Scope and Functionalities: Literature Review and Future Directions [published correction appears in J Med Internet Res. 2019 Sep 05;21(9):e15796]. *J Med Internet Res*. 2017;19(11):e388.

APPENDIX

Python scripts for the GUIs

1) Python code for the Patient Information System

```
# Patient Information System
# Registration of Patient and Appointment fixing dashboard
# Developed using Python and Tkinter
# Author - Dr. Ashalatha Sreshty Mamidi

import tkinter as tk
import mysql
import mysql.connector
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from PIL import ImageTk, Image
import os
import __main__


mydb = mysql.connector.connect(host = "localhost", user = "root", passwd = "alsy", database
= "idigihealth_db")
mycursor = mydb.cursor()

# Defining functions

#def populate_list():
#         patients_list.delete(0, END)
#         for row in mycursor.fetch():
#                   patients_list.insert(END, row)

def select_record(event):
    try:
        global selected_item
        index = patients_list.curselection()[0]
        selected_item = patients_list.get(index)

        patientID_entrybox.delete(0, END)
        patientID_entrybox.insert(END, selected_item[1])
        firstname_entrybox.delete(0, END)
        firstname_entrybox.insert(END, selected_item[2])
        surname_entrybox.delete(0, END)
        surname_entrybox.insert(END, selected_item[3])
        age_entrybox.delete(0, END)
```

```
        age_entrybox.insert(END, selected_item[4])
        dob_entrybox.delete(0, END)
        dob_entrybox.insert(END, selected_item[5])
        gender_combobox.delete(0, END)
        gender_combobox.insert(END, selected_item[6])
        pin_entrybox.delete(0, END)
        pin_entrybox.insert(END, selected_item[7])
        mobile_entrybox.delete(0, END)
        mobile_entrybox.insert(END, selected_item[8])
        email_entrybox.delete(0, END)
        email_entrybox.insert(END, selected_item[9])
        address_entrybox.delete(0, END)
        address_entrybox.insert(END, selected_item[10])
        district_entrybox.delete(0, END)
        district_entrybox.insert(END, selected_item[11])
        state_entrybox.delete(0, END)
        state_entrybox.insert(END, selected_item[12])
        ethnicity_entrybox.delete(0, END)
        ethnicity_entrybox.insert(END, selected_item[13])
        pincode_entrybox.delete(0, END)
        pincode_entrybox.insert(END, selected_item[14])
    except IndexError:
        pass

def Register():
    Patient_ID=patientID_entrybox.get()
    dbPatient_ID=""
    Select="select Patient_ID from patient_details where Patient_ID='%s'" %(Patient_ID)
    mycursor.execute(Select)
    result=mycursor.fetchall()
    for i in result:
        dbPatient_ID=i[0]
    if(Patient_ID == dbPatient_ID):
        messagebox.askokcancel("Information","Record Already exists")
    else:
        Insert="Insert into patient_details(Patient_ID, First_name, Surname, Age, Date_of_Birth,
Gender, Personal_Indentity_Proof, Mobile, Email, Address, District, State, Ethnicity, Pincode)
values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"
        Patient_ID = patientID_entrybox.get()
        First_name = firstname_entrybox.get()
        Surname = surname_entrybox.get()
        Age = age_entrybox.get()
        Date_of_Birth = dob_entrybox.get()
        Gender = gender_combobox.get()
        Personal_Indentity_Proof = pin_entrybox.get()
        Mobile = mobile_entrybox.get()
```

```
        Email = email_entrybox.get()
        Address = address_entrybox.get()
        District = district_entrybox.get()
        State = state_entrybox.get()
        Ethnicity = ethnicity_entrybox.get()
        Pincode = pincode_entrybox.get()

        if (First_name !="" and Surname !="" and Age !="" and Date_of_Birth !="" and Gender
!="" and Personal_Indentity_Proof !="" and Mobile !="" and Email !="" and Address !="" and
District !="" and State !="" and Ethnicity !="" and Pincode !=""):
            Value=(Patient_ID, First_name, Surname, Age, Date_of_Birth, Gender,
Personal_Indentity_Proof, Mobile, Email, Address, District, State, Ethnicity, Pincode)
            mycursor.execute(Insert,Value)
            mydb.commit()
            messagebox.askokcancel("Information","Record inserted Successfully")
            patientID_entrybox.delete(0, END)
            firstname_entrybox.delete(0, END)
            surname_entrybox.delete(0, END)
            age_entrybox.delete(0, END)
            dob_entrybox.delete(0, END)
            gender_combobox.delete(0, END)
            pin_entrybox.delete(0, END)
            mobile_entrybox.delete(0, END)
            email_entrybox.delete(0, END)
            address_entrybox.delete(0, END)
            district_entrybox.delete(0, END)
            state_entrybox.delete(0, END)
            ethnicity_entrybox.delete(0, END)
            pincode_entrybox.delete(0, END)
        else:
            if (First_name =="" and Surname =="" and Age =="" and Date_of_Birth =="" and
Gender =="" and Personal_Indentity_Proof =="" and Mobile =="" and Email =="" and Address
=="" and District =="" and State =="" and Ethnicity =="" and Pincode =""):
             messagebox.askokcancel("Information","New Entry Fill All Details")
            else:
             messagebox.askokcancel("Information", "Some fields left blank")
#       populate_list()

def Fetchrecord():
    if(patientID_entrybox.get() == ""):
        messagebox.showinfo("Fetch status", "Patient ID is compulsory to fetch record")
    else:
        mycursor.execute("select * from patient_details where Patient_ID = '"+
patientID_entrybox.get() +"'")
        rows = mycursor.fetchall()
```

```
        for row in rows:
#           patientID_entrybox.insert(0, row[0])
            firstname_entrybox.insert(0, row[1])
            surname_entrybox.insert(0, row[2])
            age_entrybox.insert(0, row[3])
            dob_entrybox.insert(0, row[4])
            gender_combobox.insert(0, row[5])
            pin_entrybox.insert(0, row[6])
            mobile_entrybox.insert(0, row[7])
            email_entrybox.insert(0, row[8])
            address_entrybox.insert(0, row[9])
            district_entrybox.insert(0, row[10])
            state_entrybox.insert(0, row[11])
            ethnicity_entrybox.insert(0, row[12])
            pincode_entrybox.insert(0, row[13])
#       populate_list()

def Delete():
    Patient_ID=patientID_entrybox.get()
    Delete="delete from patient_details where Patient_ID='%s'" %(Patient_ID)
    mycursor.execute(Delete)
    mydb.commit()
    messagebox.showinfo("Information","Record Deleted")
    patientID_entrybox.delete(0, END)
    firstname_entrybox.delete(0, END)
    surname_entrybox.delete(0, END)
    age_entrybox.delete(0, END)
    dob_entrybox.delete(0, END)
    gender_combobox.delete(0, END)
    pin_entrybox.delete(0, END)
    mobile_entrybox.delete(0, END)
    email_entrybox.delete(0, END)
    address_entrybox.delete(0, END)
    district_entrybox.delete(0, END)
    state_entrybox.delete(0, END)
    ethnicity_entrybox.delete(0, END)
    pincode_entrybox.delete(0, END)
#   populate_list()

def Update():
    Patient_ID = patientID_entrybox.get()
    First_name = firstname_entrybox.get()
    Surname = surname_entrybox.get()
    Age = age_entrybox.get()
    Date_of_Birth = dob_entrybox.get()
    Gender = gender_combobox.get()
```

```
    Personal_Indentity_Proof = pin_entrybox.get()
    Mobile = mobile_entrybox.get()
    Email = email_entrybox.get()
    Address = address_entrybox.get()
    District = district_entrybox.get()
    State = state_entrybox.get()
    Ethnicity = ethnicity_entrybox.get()
    Pincode = pincode_entrybox.get()
    Update="Update patient_details set First_name='%s', Surname='%s', Age='%s', Date_of_Birth='%s', Gender='%s', Personal_Indentity_Proof='%s', Mobile='%s', Email='%s', Address='%s', District='%s', State='%s', Ethnicity ='%s', Pincode ='%s' where Patient_ID='%s'" %(First_name, Surname, Age, Date_of_Birth, Gender, Personal_Indentity_Proof, Mobile, Email, Address, District, State, Ethnicity, Pincode, Patient_ID)
    mycursor.execute(Update)
    mydb.commit()
 #   populate_list()
    messagebox.showinfo("Information","Record Update Successfully")

def Clear():
    patientID_entrybox.delete(0, END)
    firstname_entrybox.delete(0, END)
    surname_entrybox.delete(0, END)
    age_entrybox.delete(0, END)
    dob_entrybox.delete(0, END)
    gender_combobox.delete(0, END)
    pin_entrybox.delete(0, END)
    mobile_entrybox.delete(0, END)
    email_entrybox.delete(0, END)
    address_entrybox.delete(0, END)
    district_entrybox.delete(0, END)
    state_entrybox.delete(0, END)
    ethnicity_entrybox.delete(0, END)
    pincode_entrybox.delete(0, END)

# Create window object
PIS = tk.Tk()
PIS.title('iDigiHealth - Patient Information System')
PIS.iconbitmap('Hosp1.ico')
PIS.geometry('800x800')
PIS.configure(bg='#00b0c6')

# Background Image
background_image = ImageTk.PhotoImage(Image.open('RegPic3.jpg'))
background_label = Label(PIS, image = background_image)
background_label.place(relwidth = 1, relheight = 1)
```

```
# Title Frame and Label
heading_PIS = ttk.Label(PIS, text = "iDigiHealth - Patient Care System", font = "Helvetica 30 bold", background = '#00b0c6')
heading_PIS.place(relx=.5, rely=.01, anchor="n")

# Sub-Title
heading2_PIS = ttk.Label(PIS, text = "Patient Information System", font = "Helvetica 20 bold", background = '#00b0c6')
heading2_PIS.place(relx=.5, rely=.1, anchor="n")

# Patient Details
heading2_PIS = ttk.Label(PIS, text = "Patient Details", font = "Helvetica 15 bold", background = '#ff4d4d')
heading2_PIS.place(relx=.05, rely=.2, anchor="w")

# Create Labels
PatientID =  ttk.Label(PIS, text = 'PATIENT ID: ', font = "Helvetica 12 bold", background = '#fff6a4')
PatientID.place(relx=.05, rely=.30, anchor="w")

First_name = ttk.Label(PIS, text = 'FIRST NAME: ', font = "Helvetica 12 bold", background = '#fff6a4')
First_name.place(relx=.05, rely=.35, anchor="w")

Surname = ttk.Label(PIS, text = 'SURNAME: ', font = "Helvetica 12 bold", background = '#fff6a4')
Surname.place(relx=.50, rely=.35, anchor="w")

Age = ttk.Label(PIS, text = 'AGE: ', font = "Helvetica 12 bold", background = '#fff6a4')
Age.place(relx=.05, rely=.40, anchor="w")

Date_of_Birth = ttk.Label(PIS, text = 'DATE OF BIRTH: ', font = "Helvetica 12 bold", background = '#fff6a4')
Date_of_Birth.place(relx=.50, rely=.40, anchor="w")

Gender = ttk.Label(PIS, text = 'GENDER: ', font = "Helvetica 12 bold", background = '#fff6a4')
Gender.place(relx=.05, rely=.45, anchor="w")

Personal_Indentity_Proof = ttk.Label(PIS, text = 'PERSONAL IDENTITY PROOF: ', font = "Helvetica 12 bold", background = '#fff6a4')
Personal_Indentity_Proof.place(relx=.50, rely=.45, anchor="w")

Mobile = ttk.Label(PIS, text = 'MOBILE NUMBER: ', font = "Helvetica 12 bold", background = '#fff6a4')
Mobile.place(relx=.05, rely=.50, anchor="w")
```

```
Email = ttk.Label(PIS, text = 'EMAIL: ', font = "Helvetica 12 bold", background = '#fff6a4')
Email.place(relx=.50, rely=.50, anchor="w")

ContactDetails =  ttk.Label(PIS, text = 'CONTACT DETAILS: ', font = "Helvetica 12 bold",
background = '#fff6a4')
ContactDetails.place(relx=.05, rely=.55, anchor="w")

Address = ttk.Label(PIS, text = 'ADDRESS: ', font = "Helvetica 12 bold", background =
'#fff6a4')
Address.place(relx=.05, rely=.60, anchor="w")

District = ttk.Label(PIS, text = 'DISTRICT: ', font = "Helvetica 12 bold", background =
'#fff6a4')
District.place(relx=.50, rely=.60, anchor="w")

State = ttk.Label(PIS, text = 'STATE: ', font = "Helvetica 12 bold", background = '#fff6a4')
State.place(relx=.05, rely=.65, anchor="w")

Ethnicity = ttk.Label(PIS, text = 'Ethnicity: ', font = "Helvetica 12 bold", background =
'#fff6a4')
Ethnicity.place(relx=.50, rely=.65, anchor="w")

Pincode = ttk.Label(PIS, text = 'PINCODE: ', font = "Helvetica 12 bold", background =
'#fff6a4')
Pincode.place(relx=.05, rely=.70, anchor="w")

# Create Entry Box
patientID_entrybox = Entry(PIS, width = 40, bd = 5, font = 'Helvetica 12 bold')
patientID_entrybox.place(relx=.20, rely=.30, anchor="w")
patientID_entrybox.focus()

firstname_entrybox = Entry(PIS, width = 40, bd = 5, font = 'Helvetica 12 bold')
firstname_entrybox.place(relx=.20, rely=.35, anchor="w")

surname_entrybox = Entry(PIS, width = 40, bd = 5, font = 'Helvetica 12 bold')
surname_entrybox.place(relx=.70, rely=.35, anchor="w")

age_entrybox = Entry(PIS, width = 40, bd = 5, font = 'Helvetica 12 bold')
age_entrybox.place(relx=.20, rely=.40, anchor="w")

dob_entrybox = Entry(PIS, width = 40, bd = 5, font = 'Helvetica 12 bold')
dob_entrybox.place(relx=.70, rely=.40, anchor="w")

# create combobox
gender_combobox = ttk.Combobox(PIS, width = 20, font = 'Helvetica 12 bold')
```

```
gender_combobox['values'] = ('Male', 'Female', 'Other')
gender_combobox.current(0)
gender_combobox.place(relx=.20, rely=.45, anchor="w")

# Create Entry Box
pin_entrybox = Entry(PIS, width = 40, bd = 5, font = 'Helvetica 12 bold')
pin_entrybox.place(relx=.70, rely=.45, anchor="w")

mobile_entrybox = Entry(PIS, width = 40, bd = 5, font = 'Helvetica 12 bold')
mobile_entrybox.place(relx=.20, rely=.50, anchor="w")

email_entrybox = Entry(PIS, width = 40, bd = 5, font = 'Helvetica 12 bold')
email_entrybox.place(relx=.70, rely=.50, anchor="w")

address_entrybox = Entry(PIS, width = 40, bd = 5, font = 'Helvetica 12 bold')
address_entrybox.place(relx=.20, rely=.60, anchor="w")

district_entrybox = Entry(PIS, width = 40, bd = 5, font = 'Helvetica 12 bold')
district_entrybox.place(relx=.70, rely=.60, anchor="w")

state_entrybox = Entry(PIS, width = 40, bd = 5, font = 'Helvetica 12 bold')
state_entrybox.place(relx=.20, rely=.65, anchor="w")

ethnicity_entrybox = Entry(PIS, width = 40, bd = 5, font = 'Helvetica 12 bold')
ethnicity_entrybox.place(relx=.70, rely=.65, anchor="w")

pincode_entrybox = Entry(PIS, width = 40, bd = 5, font = 'Helvetica 12 bold')
pincode_entrybox.place(relx=.20, rely=.70, anchor="w")

# Create Buttons
add_btn = Button(PIS, text='ADD RECORD', width=15, command=Register)
add_btn.place(relx=.10, rely=.75)

remove_btn = Button(PIS, text='DELETES RECORD', width=15, command=Delete)
remove_btn.place(relx=.30, rely=.75)

update_btn = Button(PIS, text='UPDATES RECORD', width=15, command=Update)
update_btn.place(relx=.50, rely=.75)

clear_btn = Button(PIS, text='CLEAR RECORDS', width=15, command=Clear)
clear_btn.place(relx=.70, rely=.75)

fetch_btn = Button(PIS, text='FETCH RECORDS', width=15, command=Fetchrecord)
fetch_btn.place(relx=.90, rely=.75)

# Appointment window
```

```
#appointment_btn = Button(PIS, text = 'Take an Appointment', width=20, command=popup_Appointment)
#appointment_btn.place(relx=.40, rely=.90)

# Quit Button
quit_btn = Button(PIS, text="Quit", width=8, command=PIS.quit)
quit_btn.place(relx = .90, rely = .90)

# Start program
PIS.mainloop()
```

2) Python code for fixing appointment

```
# Patient Information System
# Dashboard for Appointment fixing of Patient
# Developed using Python and Tkinter
# Database - MySQL, MySQL Workbench, mysql-connector-python
# Author - Dr. Ashalatha Sreshty Mamidi



import tkinter as tk
import mysql
import mysql.connector
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from PIL import ImageTk, Image

mydb = mysql.connector.connect(host = "localhost", user = "root", passwd = "alsy", database = "idigihealth_db")
mycursor = mydb.cursor()

#Appointment_top = tk.Toplevel(PIS)
Appointment_top = tk.Tk()

Appointment_top.title("Fix an Appointment")
Appointment_top.iconbitmap('app1.ico')
Appointment_top.geometry('800x800')

#    Appointment_top.configure(bg='#00b0c6')

# Background Image
app_bg_image = ImageTk.PhotoImage(Image.open('AppPic.jpg'))
app_bg_label = Label(Appointment_top, image = app_bg_image)
```

```
app_bg_label.place(relwidth = 1, relheight = 1)
app_bg_label.image = app_bg_image

# Defining functions

def FixApp():
   App_ID=AppointmentID_App_EB.get()
   dbAppointment_ID=""
   Select="select Appointment_ID from appointment where Appointment_ID='%s'"
%(App_ID)
   mycursor.execute(Select)
   result=mycursor.fetchall()
   for i in result:
      dbAppointment_ID=i[0]
   if(App_ID == dbAppointment_ID):
      messagebox.askokcancel("Information","Appointment already taken")
   else:
      Insert="Insert into appointment (Appointment_ID, Date, Patient_ID, Doctor_ID)
values(%s,%s,%s,%s)"
      Appointment_ID = AppointmentID_App_EB.get()
      Date = Date_App_EB.get()
      Patient_ID = PatientID_App_EB.get()
      Doctor_ID = DoctorID_App_EB.get()

      if (Date !="" and Patient_ID !="" and Doctor_ID !=""):
         Value=(Appointment_ID, Date, Patient_ID, Doctor_ID)
         mycursor.execute(Insert,Value)
         mydb.commit()
         messagebox.askokcancel("Information","Appointment fixed successfully")
         AppointmentID_App_EB.delete(0, END)
         Date_App_EB.delete(0, END)
         PatientID_App_EB.delete(0, END)
         DoctorID_App_EB.delete(0, END)
      else:
         if (Date =="" and Patient_ID =="" and Doctor_ID =""):
          messagebox.askokcancel("Information","New appointment fill all details")
         else:
          messagebox.askokcancel("Information", "Some fields left blank")
#       populate_list()

def FetApp():
   if(AppointmentID_App_EB.get() == ""):
      messagebox.showinfo("Fetch status", "Appointment ID is compulsory to fetch details")
   else:
      mycursor.execute("select * from appointment where Appointment_ID = '"+
AppointmentID_App_EB.get() +"'")
```

```
        rows = mycursor.fetchall()

        for row in rows:
#           patientID_entrybox.insert(0, row[0])
            Date_App_EB.insert(0, row[1])
            PatientID_App_EB.insert(0, row[2])
            DoctorID_App_EB.insert(0, row[3])

#       populate_list()

def DelApp():
    App_ID = AppointmentID_App_EB.get() # Appointment_ID here is a variable
    Delete="delete from appointment where Appointment_ID='%s'" %(App_ID)
    mycursor.execute(Delete)
    mydb.commit()
    messagebox.showinfo("Information","Appointment record deleted")
    AppointmentID_App_EB.delete(0, END)
    Date_App_EB.delete(0, END)
    PatientID_App_EB.delete(0, END)
    DoctorID_App_EB.delete(0, END)


#    populate_list()

def UpdApp():
    App_ID = AppointmentID_App_EB.get()
    Date = Date_App_EB.get()
    Pat_ID = PatientID_App_EB.get()
    Doc_ID = DoctorID_App_EB.get()
    Update="Update appointment set Appointment_ID='%s', Date='%s', Patient_ID='%s',
Doctor_ID='%s'" %(App_ID, Date, Pat_ID, Doc_ID)
    mycursor.execute(Update)
    mydb.commit()
 #   populate_list()
    messagebox.showinfo("Information","Record Updated successfully")

def ClrApp():
    AppointmentID_App_EB.delete(0, END)
    Date_App_EB.delete(0, END)
    PatientID_App_EB.delete(0, END)
    DoctorID_App_EB.delete(0, END)


# Title Frame and Label
app_PIS = ttk.Label(Appointment_top, text = "SCHEDULE APPOINTMENT", font =
"Helvetica 20 bold", background = '#ffffff')
```

```
app_PIS.place(relx=.50, rely=.03, anchor="n")

# Labels

AppointmentID_App = ttk.Label(Appointment_top, text = 'APPOINTMENT ID: ', font = "Helvetica 12 bold", background = '#fff6a4')
AppointmentID_App.place(relx=.05, rely=.35, anchor="w")

Date_APP = ttk.Label(Appointment_top, text = 'DATE: ', font = "Helvetica 12 bold", background = '#fff6a4')
Date_APP.place(relx=.50, rely=.35, anchor="w")

PatientID_App =  ttk.Label(Appointment_top, text = 'PATIENT ID: ', font = "Helvetica 12 bold", background = '#fff6a4')
PatientID_App.place(relx=.05, rely=.60, anchor="w")

DoctorID_App =  ttk.Label(Appointment_top, text = 'DOCTOR ID: ', font = "Helvetica 12 bold", background = '#fff6a4')
DoctorID_App.place(relx=.50, rely=.60, anchor="w")


# Entry Buttons

AppointmentID_App_EB = Entry(Appointment_top, width = 40, bd = 5, font = 'Helvetica 12 bold')
AppointmentID_App_EB.place(relx=.20, rely=.35, anchor="w")

Date_App_EB = Entry(Appointment_top, width = 40, bd = 5, font = 'Helvetica 12 bold')
Date_App_EB.place(relx=.70, rely=.35, anchor="w")

PatientID_App_EB = Entry(Appointment_top, width = 40, bd = 5, font = 'Helvetica 12 bold')
PatientID_App_EB.place(relx=.20, rely=.60, anchor="w")

DoctorID_App_EB = Entry(Appointment_top, width = 40, bd = 5, font = 'Helvetica 12 bold')
DoctorID_App_EB.place(relx=.70, rely=.60, anchor="w")


# Create Buttons
fixApp_btn = Button(Appointment_top, text='FIX APPOINTMENT', width=20, command=FixApp)
fixApp_btn.place(relx=.10, rely=.85)

delApp_btn = Button(Appointment_top, text='DELETES APPOINTMENT', width=20, command=DelApp)
delApp_btn.place(relx=.25, rely=.85)
```

```
updApp_btn = Button(Appointment_top, text='UPDATES APPOINTMENT', width=20, command=UpdApp)
updApp_btn.place(relx=.40, rely=.85)

clrApp_btn = Button(Appointment_top, text='CLEAR APPOINTMENT', width=20, command=ClrApp)
clrApp_btn.place(relx=.55, rely=.85)

fetApp_btn = Button(Appointment_top, text='FETCH APPOINTMENT', width=20, command=FetApp)
fetApp_btn.place(relx=.70, rely=.85)

# Quit Button

quit_btn = Button(Appointment_top, text="Quit", width=8, command=Appointment_top.quit)
quit_btn.place(relx = .90, rely = .90)

# Start program

Appointment_top.mainloop()
```

3) Python code for entering doctor information

```
# iDigiHealth - Patient Information System
# Dashboard for entering Doctor details
# Developed using Python and Tkinter
# Database - MySQL, MySQL Workbench, mysql-connector-python
# Author - Dr. Ashalatha Sreshty Mamidi



import tkinter as tk
import mysql
import mysql.connector
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from PIL import ImageTk, Image

mydb = mysql.connector.connect(host = "localhost", user = "root", passwd = "alsy", database = "idigihealth_db")
mycursor = mydb.cursor()

#Appointment_top = tk.Toplevel(PIS)
DocDetails = tk.Tk()
```

```
DocDetails.title("Doctor Details")
DocDetails.iconbitmap('doc.ico')
DocDetails.geometry('800x800')

# Appointment_top.configure(bg='#00b0c6')

# Background Image
doc_bg_image = ImageTk.PhotoImage(Image.open('RegPic2.jpg'))
doc_bg_label = Label(DocDetails, image = doc_bg_image)
doc_bg_label.place(relwidth = 1, relheight = 1)
doc_bg_label.image = doc_bg_image

# Defining functions

def Enter():
   Doc_ID=DoctorID_EB.get()
   dbDoc_ID=""
   Select="select Doctor_ID from doctor_details where Doctor_ID='%s'" %(Doc_ID)
   mycursor.execute(Select)
   result=mycursor.fetchall()
   for i in result:
      dbDoc_ID=i[0]
   if(Doc_ID == dbDoc_ID):
      messagebox.askokcancel("Information","Record already taken")
   else:
      Insert="Insert into doctor_details (Doctor_ID, Doctor_name, Specialization)
values(%s,%s,%s)"
      Doc_ID = DoctorID_EB.get()
      Docname = Doctor_name_EB.get()
      Docspl = DocSpecialization_EB.get()

      if (Docname !="" and Docspl !=""):
         Value=(Doc_ID, Docname, Docspl)
         mycursor.execute(Insert,Value)
         mydb.commit()
         messagebox.askokcancel("Information","Record inserted successfully")
         DoctorID_EB.delete(0, END)
         Doctor_name_EB.delete(0, END)
         DocSpecialization_EB.delete(0, END)

      else:
         if (Doctor_name =="" and Specialization =""):
          messagebox.askokcancel("Information","New Doctor - fill all details")
         else:
          messagebox.askokcancel("Information", "Some fields left blank")
```

```
#       populate_list()

def Fetch():
   if(DoctorID_EB.get() == ""):
      messagebox.showinfo("Fetch status", "Doctor ID is compulsory to fetch details")
   else:
      mycursor.execute("select * from doctor_details where Doctor_ID = '"+
DoctorID_EB.get() +"'")
      rows = mycursor.fetchall()

      for row in rows:
#         patientID_entrybox.insert(0, row[0])
         Doctor_name_EB.insert(0, row[1])
         DocSpecialization_EB.insert(0, row[2])

#       populate_list()

def Remove():
   Doc_ID=DoctorID_EB.get() # Appointment_ID here is a variable
   Delete="delete from doctor_details where Doctor_ID='%s'" %(Doc_ID)
   mycursor.execute(Delete)
   mydb.commit()
   messagebox.showinfo("Information","Doctor details record removed")
   DoctorID_EB.delete(0, END)
   Doctor_name_EB.delete(0, END)
   DocSpecialization_EB.delete(0, END)

#   populate_list()

def Update():
   Doc_ID = DoctorID_EB.get()
   Docname = Doctor_name_EB.get()
   Docspl = DocSpecialization_EB.get()
   Update="Update doctor_details set Doctor_name='%s', Specialization='%s',
Doctor_ID='%s'" %(Docname, Docspl, Doc_ID)
   mycursor.execute(Update)
   mydb.commit()
 #  populate_list()
   messagebox.showinfo("Information","Record updated successfully")

def Clear():
   DoctorID_EB.delete(0, END)
   Doctor_name_EB.delete(0, END)
   DocSpecialization_EB.delete(0, END)
```

```
# Title Frame and Label
doc_PIS = ttk.Label(DocDetails, text = "DOCTOR DETAILS", font = "Helvetica 20 bold",
background = '#ffffff')
doc_PIS.place(relx=.50, rely=.03, anchor="n")

# Labels

DoctorID = ttk.Label(DocDetails, text = 'DOCTOR ID: ', font = "Helvetica 12 bold",
background = '#fff6a4')
DoctorID.place(relx=.05, rely=.35, anchor="w")

Doctor_name = ttk.Label(DocDetails, text = 'DOCTOR NAME: ', font = "Helvetica 12 bold",
background = '#fff6a4')
Doctor_name.place(relx=.50, rely=.35, anchor="w")

DocSpecialization =  ttk.Label(DocDetails, text = 'SPECIALIZATION: ', font = "Helvetica 12
bold", background = '#fff6a4')
DocSpecialization.place(relx=.05, rely=.60, anchor="w")


# Entry Buttons

DoctorID_EB = Entry(DocDetails, width = 40, bd = 5, font = 'Helvetica 12 bold')
DoctorID_EB.place(relx=.20, rely=.35, anchor="w")

Doctor_name_EB = Entry(DocDetails, width = 40, bd = 5, font = 'Helvetica 12 bold')
Doctor_name_EB.place(relx=.70, rely=.35, anchor="w")

DocSpecialization_EB = Entry(DocDetails, width = 40, bd = 5, font = 'Helvetica 12 bold')
DocSpecialization_EB.place(relx=.20, rely=.60, anchor="w")


# Create Buttons
enter_btn = Button(DocDetails, text='ENTER RECORD', width=20, command=Enter)
enter_btn.place(relx=.10, rely=.85)

fetch_btn = Button(DocDetails, text='FETCH RECORD', width=20, command=Fetch)
fetch_btn.place(relx=.70, rely=.85)

remove_btn = Button(DocDetails, text='DELETES RECORD', width=20, command=Remove)
remove_btn.place(relx=.25, rely=.85)

update_btn = Button(DocDetails, text='UPDATES RECORD', width=20, command=Update)
update_btn.place(relx=.40, rely=.85)

clear_btn = Button(DocDetails, text='CLEAR RECORD', width=20, command=Clear)
```

```
clear_btn.place(relx=.55, rely=.85)

# Quit Button

quit_btn = Button(DocDetails, text="Quit", width=8, command=DocDetails.quit)
quit_btn.place(relx = .90, rely = .90)

# Start program

DocDetails.mainloop()
```

4) Python code for haematology laboratory

```
# iDigiHealth - Patient Information System
# GUI for Haematology laboratory
# Developed using Python and Tkinter
# Database - MySQL, MySQL Workbench, mysql-connector-python
# Author - Dr. Ashalatha Sreshty Mamidi

import tkinter as tk
import mysql
import mysql.connector
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from PIL import ImageTk, Image

mydb = mysql.connector.connect(host = "localhost", user = "root", passwd = "alsy", database
= "idigihealth_db")
mycursor = mydb.cursor()

#Appointment_top = tk.Toplevel(PIS)
HmLab = tk.Tk()

HmLab.title("Haematology Laboratory")
HmLab.iconbitmap('Lab.ico')
HmLab.geometry('800x800')
#photo=PhotoImage(file='031Blessing.png')
#HmLab.configure(bg='#00b0c6')

def resize_image(event):
    new_width = event.width
    new_height = event.height
    image = copy_of_image.resize((new_width, new_height))
    photo = ImageTk.PhotoImage(image)
```

```
    label.config(image = photo)
    label.image = photo #avoid garbage collection

image = Image.open('031Blessing.png')
copy_of_image = image.copy()
photo = ImageTk.PhotoImage(image)
label = ttk.Label(HmLab, image = photo)
label.bind('<Configure>', resize_image)
label.pack(fill=BOTH, expand = YES)



# Defining functions
#def populate_list():
#    Test_TypeRes_Listbox.delete(0, END)
#    for row in mycursor.fetchall():
#        Test_TypeRes_Listbox.insert(END, row)

def add():

    if TestType_CB.get() == '' or TestResults_EB.get() == '':
        messagebox.showerror('Error Message', 'Please include all fields')
        return

    Test_TypeRes_Listbox.insert(END, (TestType_CB.get(), TestResults_EB.get()))

def removentry():

    Test_TypeRes_Listbox.delete(0, END)

def select_TypeRes(event):
    try:
        global selected_item
        index = (Test_TypeRes_Listbox.curselection())#[0]
        selected_item = Test_TypeRes_Listbox.get(index)
        TestType_CB.delete(0, END)
        TestType_CB.insert(END, selected_item[1])
        TestResults_EB.delete(0, END)
        TestResults_EB.insert(END, selected_item[2])
    except IndexError:
        pass

TestRes_List = []
TestType_List = []

def Enter():
```

```
    # process Listobox entries
    TTL = Test_TypeRes_Listbox.get(0, END)
    for item in TTL:
        TestType_List.append(item[0])
        TestRes_List.append(item[1])
    #print(TestType_List)
    #print(TestRes_List)
    HMTestType = str(",".join(TestType_List)) # Converting list to string
    HMTestResults =  str(",".join(TestRes_List)) # Converting list to string
#    print(HMTestType)
#    print(HMTestResults)

    # Entering data into database table
    Hm_ID=HmLabID_EB.get()
    dbHm_ID=""
    Select="SELECT Haematology_Lab_ID FROM haematology_lab WHERE Haematology_Lab_ID='%s'" %(Hm_ID)
    mycursor.execute(Select)
    result=mycursor.fetchall()
    for i in result:
        dbHm_ID=i[0]
    if(Hm_ID == dbHm_ID):
        messagebox.askokcancel("Information","Record already taken")
    else:
        Insert="INSERT INTO haematology_lab (Haematology_Lab_ID, Consultation_ID, HM_Test_Type, HM_Test_Results, HM_Test_Cost) values(%s,%s,%s,%s,%s)"
        Haematology_Lab_ID = HmLabID_EB.get()
        Consultation_ID = ConsultID_EB.get()
        HM_Test_Type = HMTestType
        HM_Test_Results = HMTestResults
        HM_Test_Cost = TotalCost_EB.get()
        print(Haematology_Lab_ID, Consultation_ID, HM_Test_Type, HM_Test_Results, HM_Test_Cost)

        if (Consultation_ID !="" and HM_Test_Cost !=""):
            Value=(Haematology_Lab_ID, Consultation_ID, HM_Test_Type, HM_Test_Results, HM_Test_Cost)
            mycursor.execute(Insert,Value)
            mydb.commit()
            messagebox.askokcancel("Information","Record inserted successfully")
            HmLabID_EB.delete(0, END)
            ConsultID_EB.delete(0, END)
            Test_TypeRes_Listbox.delete(0, END)
            TotalCost_EB.delete(0, END)
```

```
        else:
            if (Consultation_ID =="" and HM_Test_Type =="" and HM_Test_Results =="" and
HM_Test_Cost =""):
             messagebox.askokcancel("Information","New Patient - fill all details")
            else:
             messagebox.askokcancel("Information", "Some fields left blank")

def Fetch():
    if(HmLabID_EB.get() == ""):
        messagebox.showinfo("Fetch status", "Haematology Lab ID is compulsory to fetch
details")
    else:
        mycursor.execute("SELECT * FROM haematology_lab WHERE Haematology_Lab_ID
= '"+ HmLabID_EB.get() +"'")
        rows = mycursor.fetchall()

        for row in rows:
            Test_TypeRes_Listbox.insert(0, END)
#           ConsultID_EB.insert(0, row[1])
#           TestType_CB.insert(0, row[2])
#           TestResults_EB.insert(0, row[3])
#           TotalCost_EB.insert(0, row[4])
#       populate_list()

def Remove():
    Hm_ID=HmLabID_EB.get() # Hm_Lab_ID here is a variable
    Delete="DELETE FROM haematology_lab WHERE Haematology_Lab_ID ='%s'"
%(Hm_ID)
    mycursor.execute(Delete)
    mydb.commit()
    messagebox.showinfo("Information","Record removed")
    HmLabID_EB.delete(0, END)
    ConsultID_EB.delete(0, END)
    Test_TypeRes_Listbox.delete(0, END)
    TotalCost_EB.delete(0, END)
#   populate_list()

def Update():

    # process Listobox entries
    TTL = Test_TypeRes_Listbox.get(0, END)
    for item in TTL:
        TestType_List.append(item[0])
        TestRes_List.append(item[1])
    #print(TestType_List)
    #print(TestRes_List)
```

```
    HMTestType = str(",".join(TestType_List)) # Converting list to string
    HMTestResults =  str(",".join(TestRes_List)) # Converting list to string
    print(HMTestType)
    print(HMTestResults)

    Haematology_Lab_ID = HmLabID_EB.get()
    Consultation_ID = ConsultID_EB.get()
    HM_Test_Type = HMTestType
    HM_Test_Results = HMTestResults
    HM_Test_Cost = TotalCost_EB.get()

    Update="UPDATE haematology_lab SET Haematology_Lab_ID='%s' Consultation_ID='%s', HM_Test_Type='%s', HM_Test_Results='%s', HM_Test_Cost='%s'" %(Consultation_ID, HM_Test_Type, HM_Test_Results, HM_Test_Cost, Haematology_Lab_ID)
    mycursor.execute(Update)
    mydb.commit()
    messagebox.showinfo("Information","Record updated successfully")
#    populate_list()

def Clear():
    HmLabID_EB.delete(0, END)
    ConsultID_EB.delete(0, END)
    Test_TypeRes_Listbox.delete(0, END)
    TotalCost_EB.delete(0, END)

# Labels

# Title Frame and Label
HmLab_Title = ttk.Label(HmLab, text = "HAEMATOLOGY LABORATORY", font = "Helvetica 30 bold", background = '#d1fdff', borderwidth = 5)
HmLab_Title.place(relx=.5, rely=.01, anchor="n")

# Patient Details
heading2_PIS = ttk.Label(HmLab, text = "Blood Test Details", font = "Helvetica 15 bold", background = '#d1fdff')
heading2_PIS.place(relx=.05, rely=.20, anchor="w")

# Create Labels
HmLab_ID =  ttk.Label(HmLab, text = 'HAEMATOLOGY LAB ID: ', font = "Helvetica 12 bold", background = '#fff6a4')
HmLab_ID.place(relx=.05, rely=.30, anchor="w")

Consultation_ID = ttk.Label(HmLab, text = 'CONSULTATION ID: ', font = "Helvetica 12 bold", background = '#fff6a4')
Consultation_ID.place(relx=.50, rely=.30, anchor="w")
```

```
Test_Type = ttk.Label(HmLab, text = 'TEST TYPE: ', font = "Helvetica 12 bold", background = '#fff6a4')
Test_Type.place(relx=.05, rely=.35, anchor="w")

Test_Results = ttk.Label(HmLab, text = 'TEST RESULTS: ', font = "Helvetica 12 bold", background = '#fff6a4')
Test_Results.place(relx=.50, rely=.35, anchor="w")

Total_Cost = ttk.Label(HmLab, text = 'TOTAL COST: ', font = "Helvetica 12 bold", background = '#fff6a4')
Total_Cost.place(relx=.05, rely=.65, anchor="w")

# Create Entry Boxes

# Create Entry Box
HmLabID_EB = Entry(HmLab, width = 30, bd = 5, font = 'Helvetica 12 bold')
HmLabID_EB.place(relx=.25, rely=.30, anchor="w")
HmLabID_EB.focus()

ConsultID_EB = Entry(HmLab, width = 30, bd = 5, font = 'Helvetica 12 bold')
ConsultID_EB.place(relx=.65, rely=.30, anchor="w")

# create combobox
TestType_CB = ttk.Combobox(HmLab, width = 30, font = 'Helvetica 12 bold', state = 'readonly')
TestType_CB['values'] = ('Hemoglobin', 'RBC_Count', 'WBC_Count', 'Granulocytes', 'Lymphocytes', 'Monocytes', 'Eosinophils', 'Basophils', 'Platelets')
TestType_CB.current(0)
TestType_CB.place(relx=.25, rely=.35, anchor="w")

TestResults_EB = Entry(HmLab, width = 30, bd = 5, font = 'Helvetica 12 bold')
TestResults_EB.place(relx=.65, rely=.35, anchor="w")

TotalCost_EB = Entry(HmLab, width = 30, bd = 5, font = 'Helvetica 12 bold')
TotalCost_EB.place(relx=.25, rely=.65, anchor="w")

# Buttons

add_btn = Button(HmLab, text="ADD", width=8, command = add)
add_btn.place(relx = .90, rely = .33)

rmventry_btn = Button(HmLab, text="REMOVE", width=8, command = removentry)
rmventry_btn.place(relx = .90, rely = .64)

enter_btn = Button(HmLab, text='ENTER RECORD', width=20, command=Enter)
```

```
enter_btn.place(relx=.10, rely=.85)

fetch_btn = Button(HmLab, text='FETCH RECORD', width=20, command=Fetch)
fetch_btn.place(relx=.70, rely=.85)

remove_btn = Button(HmLab, text='DELETES RECORD', width=20, command=Remove)
remove_btn.place(relx=.25, rely=.85)

update_btn = Button(HmLab, text='UPDATES RECORD', width=20, command=Update)
update_btn.place(relx=.40, rely=.85)

clear_btn = Button(HmLab, text='CLEAR RECORD', width=20, command=Clear)
clear_btn.place(relx=.55, rely=.85)

# ListBox

Test_TypeRes_Listbox = Listbox(HmLab, height=10, width=45, border=3)
Test_TypeRes_Listbox.place(relx=.65, rely=.55, anchor="w")

# Create scrollbar
scrollbar = Scrollbar(HmLab)
scrollbar.place(relx=.90, rely=.47, anchor="e")
# Set scroll to listbox
Test_TypeRes_Listbox.configure(yscrollcommand=scrollbar.set)
scrollbar.configure(command=Test_TypeRes_Listbox.yview)
# Bind select
Test_TypeRes_Listbox.bind('<<ListboxSelect>>', select_TypeRes)

# Populate data
#populate_list()

# Quit Button
exit_btn = Button(HmLab, text="EXIT", width=8, command=HmLab.quit)
exit_btn.place(relx = .90, rely = .90)

# Start program
HmLab.mainloop()
```

5) Python code for imaging laboratory

```
# iDigiHealth - Patient Information System
# GUI for Imaging laboratory
# Developed using Python and Tkinter
# Database - MySQL, MySQL Workbench, mysql-connector-python
# Author - Dr. Ashalatha Sreshty Mamidi
```

```
import tkinter as tk
import mysql
import mysql.connector
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from PIL import ImageTk, Image
import glob
from tkinter import filedialog
import base64

mydb = mysql.connector.connect(host = "localhost", user = "root", passwd = "alsy", database
= "idigihealth_db")
mycursor = mydb.cursor()

#Appointment_top = tk.Toplevel(PIS)
ImageLab = tk.Tk()
ImageLab.title("Doctor Consultation")
ImageLab.iconbitmap('Lab.ico')
ImageLab.geometry('800x800')
#photo=PhotoImage(file='031Blessing.png')
#HmLab.configure(bg='#00b0c6')

def resize_image(event):
    new_width = event.width
    new_height = event.height
    image = copy_of_image.resize((new_width, new_height))
    photo = ImageTk.PhotoImage(image)
    label.config(image = photo)
    label.image = photo #avoid garbage collection

image = Image.open('053SoftGrass.png')
copy_of_image = image.copy()
photo = ImageTk.PhotoImage(image)
label = ttk.Label(ImageLab, image = photo)
label.bind('<Configure>', resize_image)
label.pack(fill=BOTH, expand = YES)

# Browse and show images

def show(event):
        n = ImgDisplayBox.curselection()
```

```
        fname = ImgDisplayBox.get(n)
        img = tk.PhotoImage(file = fname)
        lab.config(image = img)
        lab.image = img
        print(fname)
        return(fname)

def convertToBinaryData(filename):
    # Convert digital data to binary format
    with open(filename, 'rb') as file:
        binaryData = file.read()
    return binaryData

def Enter():

    # Entering data into database table
    Img_ID=ImgLabID_EB.get()
    dbImg_ID=""
    Select="SELECT Imaging_Lab FROM imaging_lab WHERE Imaging_Lab='%s'" %(Img_ID)
    mycursor.execute(Select)
    result=mycursor.fetchall()

    photo = show(fname)
    print(convertToBinaryData(photo))
    encodestring = base64.b64encode(photo)

    for i in result:
        dbImg_ID=i[0]
    if(Img_ID == dbImg_ID):
        messagebox.askokcancel("Information","Record already taken")
    else:
        Insert="INSERT INTO imaging_lab (Imaging_Lab, Consultation_ID, IMG_Test_Type, IMG_Test_Results, IMG_Test_Cost) values(%s,%s,%s,%s,%s)"
        Imaging_Lab = ImgLabID_EB.get()
        Consultation_ID = ConsultID_EB.get()
        IMG_Test_Type = ImgTestType_CB.get()
        IMG_Test_Results = encodestring
        IMG_Test_Cost = ImgCost_EB.get()

        if (Consultation_ID !="" and IMG_Test_Type !="" and IMG_Test_Cost !=""):
            Value=(Imaging_Lab, Consultation_ID, IMG_Test_Type, IMG_Test_Results, IMG_Test_Cost)
            mycursor.execute(Insert,Value)
            mydb.commit()
            messagebox.askokcancel("Information","Record inserted successfully")
```

```
            ImgLabID_EB.delete(0, END)
            ConsultID_EB.delete(0, END)
            ImgTestType_CB.delete(0, END)
            TestResults_EB.delete(0, END)
            ImgCost_EB.delete(0, END)

        else:
            if (Consultation_ID =="" and IMG_Test_Type =="" and IMG_Test_Results =="" and
IMG_Test_Cost =""):
             messagebox.askokcancel("Information","New Patient - fill all details")
            else:
             messagebox.askokcancel("Information", "Some fields left blank")

def Fetch():
    if(ImgLabID_EB.get() == ""):
        messagebox.showinfo("Fetch status", "Imaging Lab ID is compulsory to fetch details")
    else:
        mycursor.execute("SELECT * FROM imaging_lab WHERE Imaging_Lab = '"+
ImgLabID_EB.get() +"'")
        rows = mycursor.fetchall()

        for row in rows:
            print("Image lab Id = ", row[0], )
            print("Consultation ID = ", row[1])
            print("Test type - ", row[2])
            image = row[2]
            print("Test cost - ", row[4])
            print("Storing image on disk \n")
            write_file(image, photo)


def Remove():
    Img_Lab = ImgLabID_EB.get() # Img_Lab_ID here is a variable
    Delete="DELETE FROM imaging_lab WHERE Imaging_Lab ='%s'" %(Img_Lab)
    mycursor.execute(Delete)
    mydb.commit()
    messagebox.showinfo("Information","Record removed")
    ImgLabID_EB.delete(0, END)
    ConsultID_EB.delete(0, END)
    ImgTestType_CB.delete(0, END)
    ImgCost_EB.delete(0, END)
#    populate_list()

def Update():

    Imaging_Lab = ImgLabID_EB.get()
```

```
    Consultation_ID = ConsultID_EB.get()
    IMG_Test_Type = ImgTestType_CB.get()
    IMG_Test_Results = convertToBinaryData(photo)
    IMG_Test_Cost = ImgCost_EB.get()

    Update="UPDATE imaging_lab SET Consultation_ID='%s', IMG_Test_Type='%s', IMG_Test_Results='%s', IMG_Test_Cost='%s' WHERE Imaging_Lab='%s'" %(Consultation_ID, HM_Test_Type, HM_Test_Results, HM_Test_Cost, Haematology_Lab_ID)
    mycursor.execute(Update)
    mydb.commit()
    messagebox.showinfo("Information","Record updated successfully")
#    populate_list()

def Clear():
    ImgLabID_EB.delete(0, END)
    ConsultID_EB.delete(0, END)
    ImgTestType_CB.delete(0, END)
    ImgCost_EB.delete(0, END)


# Title Frame and Label
Img_PIS = ttk.Label(ImageLab, text = "IMAGING LABORATORY", font = "Helvetica 20 bold", background = '#ffffff')
Img_PIS.place(relx=.50, rely=.03, anchor="n")

# Create Labels
ImageLab_ID =  ttk.Label(ImageLab, text = 'IMAGE LAB ID: ', font = "Helvetica 12 bold", background = '#fff6a4')
ImageLab_ID.place(relx=.05, rely=.20, anchor="w")

Consulation_ID = ttk.Label(ImageLab, text = 'CONSULTATION ID: ', font = "Helvetica 12 bold", background = '#fff6a4')
Consulation_ID.place(relx=.50, rely=.20, anchor="w")

Img_Test_Type = ttk.Label(ImageLab, text = 'IMAGE TEST TYPE: ', font = "Helvetica 12 bold", background = '#fff6a4')
Img_Test_Type.place(relx=.05, rely=.30, anchor="w")

Img_Test_Cost = ttk.Label(ImageLab, text = 'IMAGE TEST COST: ', font = "Helvetica 12 bold", background = '#fff6a4')
```

```
Img_Test_Cost.place(relx=.50, rely=.30, anchor="w")

Image_Test_Results = ttk.Label(ImageLab, text = 'IMAGE RESULTS: ', font = "Helvetica 12
bold", background = '#fff6a4')
Image_Test_Results.place(relx=.05, rely=.40, anchor="w")

# Images List Box
Disp_label = tk.Label(ImageLab, text = 'IMAGE DISPLAY: ', font = "Helvetica 12 bold",
background = '#fff6a4')
Disp_label.place(relx=.05, rely=.50, anchor="w")

ImgDisplayBox = Listbox(ImageLab)
ImgDisplayBox.place(relx=.20, rely=.65, anchor="w")
list_images = [i for i in glob.glob("*.jpg")]

for fname in list_images:
        ImgDisplayBox.insert(tk.END, fname)

ImgDisplayBox.bind("<<ListboxSelect>>", show)
img = tk.PhotoImage(file = "bloodcells.png")
lab = tk.Label(ImageLab, image = img)
lab.place(relx=.50, rely=.65, anchor="w")

# Entry Buttons

# Create Entry Box
ImgLabID_EB = Entry(ImageLab, width = 30, bd = 5, font = 'Helvetica 12 bold')
ImgLabID_EB.place(relx=.20, rely=.20, anchor="w")
ImgLabID_EB.focus()

ConsultID_EB = Entry(ImageLab, width = 30, bd = 5, font = 'Helvetica 12 bold')
ConsultID_EB.place(relx=.65, rely=.20, anchor="w")

# create combobox
ImgTestType_CB = ttk.Combobox(ImageLab, width = 30, font = 'Helvetica 12 bold', state =
'readonly')
ImgTestType_CB['values'] = ('Microscopy', 'CT Scan', 'Ultrasound', 'MRI Scan', 'X-ray')
ImgTestType_CB.current(0)
ImgTestType_CB.place(relx=.20, rely=.30, anchor="w")

ImgCost_EB = Entry(ImageLab, width = 30, bd = 5, font = 'Helvetica 12 bold')
ImgCost_EB.place(relx=.65, rely=.30, anchor="w")

TestResults_EB = Entry(ImageLab, width = 30, bd = 5, font = 'Helvetica 12 bold')
TestResults_EB.place(relx=.20, rely=.40, anchor="w")
```

```
# Buttons

#add_btn = Button(ImageLab, text="ADD", width=8)#, command = add)
#add_btn.place(relx = .90, rely = .33)

#rmventry_btn = Button(ImageLab, text="REMOVE", width=8)#, command = removentry)
#rmventry_btn.place(relx = .90, rely = .64)

Next_img = Button(ImageLab, text="NEXT IMAGE", width=17, default=ACTIVE, borderwidth=0)#, command=Read_image)
Next_img.place(relx=.50, rely=.85)

enter_btn = Button(ImageLab, text='ENTER RECORD', width=20)#, command=Enter)
enter_btn.place(relx=.10, rely=.90)

fetch_btn = Button(ImageLab, text='FETCH RECORD', width=20)#, command=Fetch)
fetch_btn.place(relx=.70, rely=.90)

remove_btn = Button(ImageLab, text='DELETES RECORD', width=20)#, command=Remove)
remove_btn.place(relx=.25, rely=.90)

update_btn = Button(ImageLab, text='UPDATES RECORD', width=20)#, command=Update)
update_btn.place(relx=.40, rely=.90)

clear_btn = Button(ImageLab, text='CLEAR RECORD', width=20)#, command=Clear)
clear_btn.place(relx=.55, rely=.90)

# ListBox

#Test_TypeRes_Listbox = Listbox(ImageLab, height=15, width=140, border=3)
#Test_TypeRes_Listbox.place(relx=.20, rely=.67, anchor="w")

# Create scrollbar
#scrollbar = Scrollbar(ImageLab)
#scrollbar.place(relx=.90, rely=.47, anchor="e")
# Set scroll to listbox
#Test_TypeRes_Listbox.configure(yscrollcommand=scrollbar.set)
#scrollbar.configure(command=Test_TypeRes_Listbox.yview)
# Bind select
#Test_TypeRes_Listbox.bind('<<ListboxSelect>>')#, select_TypeRes)

# Populate data
#populate_list()
```

```
# Quit Button
exit_btn = Button(ImageLab, text="EXIT", width=8, command=ImageLab.quit)
exit_btn.place(relx = .90, rely = .90)

# Start program
ImageLab.mainloop()
```

6) Python code for doctor consultation

```
#iDigiHealth - Patient Information System
# Dashboard for Doctor Consultation of Patient
# Developed using Python and Tkinter
# Database - MySQL, MySQL Workbench, mysql-connector-python
# Author - Dr. Ashalatha Sreshty Mamidi
import tkinter as tk
import mysql
import mysql.connector
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from PIL import ImageTk, Image

mydb = mysql.connector.connect(host = "localhost", user = "root", passwd = "alsy", database
= "idigihealth_db")
mycursor = mydb.cursor()

#Appointment_top = tk.Toplevel(PIS)
Consultn = tk.Tk()
Consultn.title("Doctor Consultation")
Consultn.iconbitmap('Lab.ico')
Consultn.geometry('800x800')
#photo=PhotoImage(file='031Blessing.png')
#HmLab.configure(bg='#00b0c6')

def resize_image(event):
    new_width = event.width
    new_height = event.height
    image = copy_of_image.resize((new_width, new_height))
    photo = ImageTk.PhotoImage(image)
    label.config(image = photo)
    label.image = photo #avoid garbage collection

image = Image.open('014AmyCrisp.png')
copy_of_image = image.copy()
```

```
photo = ImageTk.PhotoImage(image)
label = ttk.Label(Consultn, image = photo)
label.bind('<Configure>', resize_image)
label.pack(fill=BOTH, expand = YES)

# Defining functions

def Enter():
   Con_ID=ConsultnID_EB.get()
   dbCon_ID=""
   Select="SELECT Consultation_ID FROM consultation WHERE Consultation_ID='%s'" %(Con_ID)
   mycursor.execute(Select)
   result=mycursor.fetchall()
   for i in result:
      dbCon_ID=i[0]
   if(Con_ID == dbCon_ID):
      messagebox.askokcancel("Information","Record already taken")
   else:
      Insert="INSERT INTO consultation (Consultation_ID, Appointment_ID, Medical_History, Diagnosis, Assessment, Treatment_Plan, Consultation_Fees) values(%s,%s,%s,%s,%s,%s,%s)"
      Con_ID = ConsultnID_EB.get()
      App_ID = AppointmentID_EB.get()
      MedHis = MedHistory_EB.get("1.0","end-1c")
      Diags = Diagnosis_EB.get("1.0","end-1c")
      Assess = Assessment_EB.get("1.0","end-1c")
      TrtPln = TreatPlan_EB.get("1.0","end-1c")
      ConFee = ConsultFees_EB.get()

      if (App_ID !="" and MedHis !="" and Diags !="" and Assess !="" and TrtPln !="" and ConFee !=""):
         Value=(Con_ID, App_ID, MedHis, Diags, Assess,TrtPln, ConFee)
         mycursor.execute(Insert,Value)
         mydb.commit()
         messagebox.askokcancel("Information","Record inserted successfully")
         ConsultnID_EB.delete(0, END)
         AppointmentID_EB.delete(0, END)
         MedHistory_EB.delete(0, END)
         Diagnosis_EB.delete(0, END)
         Assessment_EB.delete(0, END)
         TreatPlan_EB.delete(0, END)
         ConsultFees_EB.delete(0, END)

      else:
```

```
        if (Appointment_ID =="" and Medical_History =="" and Diagnosis =="" and
Assessment =="" and Treatment_Plan =="" and Consultation_Fees =""):
         messagebox.askokcancel("Information","New Doctor - fill all details")
        else:
         messagebox.askokcancel("Information", "Some fields left blank")
#       populate_list()

def Fetch():
   if(ConsultnID_EB.get() == ""):
      messagebox.showinfo("Fetch status", "Consultation ID is compulsory to fetch details")
   else:
      mycursor.execute("SELECT * FROM consultation WHERE Consultation_ID = '"+
ConsultnID_EB.get() +"'")
      rows = mycursor.fetchall()

      for row in rows:
         AppointmentID_EB.insert(0, row[1])
         MedHistory_EB.insert(0, row[2])
         Diagnosis_EB.insert(0, row[3])
         Assessment_EB.insert(0, row[4])
         TreatPlan_EB.insert(0, row[5])
         ConsultFees_EB.insert(0, row[6])
         populate_list()

def Remove():
   Con_ID=ConsultnID_EB.get() # Appointment_ID here is a variable
   Delete="DELETE FROM consultation WHERE Consultation_ID='%s'" %(Con_ID)
   mycursor.execute(Delete)
   mydb.commit()
   messagebox.showinfo("Information","Record removed")
   ConsultnID_EB.delete(0, END)
   AppointmentID_EB.delete(0, END)
   MedHistory_EB.delete(0, END)
   Diagnosis_EB.delete(0, END)
   Assessment_EB.delete(0, END)
   TreatPlan_EB.delete(0, END)
   ConsultFees_EB.delete(0, END)

#   populate_list()

def Update():
   Con_ID = ConsultnID_EB.get()
   App_ID = AppointmentID_EB.get()
   MedHis = MedHistory_EB.get("1.0","end-1c")
   Diags = Diagnosis_EB.get("1.0","end-1c")
   Assess = Assessment_EB.get("1.0","end-1c")
```

```
    TrtPln = TreatPlan_EB.get("1.0","end-1c")
    ConFee = ConsultFees_EB.get()
    Update="UPDATE consultation SET Consultation_ID ='%s', Appointment_ID ='%s', Medical_History ='%s', Diagnosis = '%s', Assessment = '%s', Treatment_Plan = '%s', Consultation_Fees = '%s'" %(Con_ID, App_ID, MedHis, Diags, Assess, TrtPln, ConFee)
    mycursor.execute(Update)
    mydb.commit()
 #   populate_list()
    messagebox.showinfo("Information","Record updated successfully")

def Clear():
    ConsultnID_EB.delete(0, END)
    AppointmentID_EB.delete(0, END)
    MedHistory_EB.delete(0, END)
    Diagnosis_EB.delete(0, END)
    Assessment_EB.delete(0, END)
    TreatPlan_EB.delete(0, END)
    ConsultFees_EB.delete(0, END)

# Title Frame and Label
Cons_PIS = ttk.Label(Consultn, text = "DOCTOR CONSULTATION", font = "Helvetica 20 bold", background = '#ffffff')
Cons_PIS.place(relx=.50, rely=.03, anchor="n")

# Create Labels
Consultn_ID =  ttk.Label(Consultn, text = 'CONSULTATION ID: ', font = "Helvetica 12 bold", background = '#fff6a4')
Consultn_ID.place(relx=.05, rely=.25, anchor="w")

Appointment_ID = ttk.Label(Consultn, text = 'APPOINTMENT ID: ', font = "Helvetica 12 bold", background = '#fff6a4')
Appointment_ID.place(relx=.55, rely=.25, anchor="w")

Med_History = ttk.Label(Consultn, text = 'MEDICAL HISTORY: ', font = "Helvetica 12 bold", background = '#fff6a4')
Med_History.place(relx=.05, rely=.40, anchor="w")

Diagnosis = ttk.Label(Consultn, text = 'DIAGNOSIS: ', font = "Helvetica 12 bold", background = '#fff6a4')
Diagnosis.place(relx=.55, rely=.40, anchor="w")

Assessment = ttk.Label(Consultn, text = 'ASSESSMENT: ', font = "Helvetica 12 bold", background = '#fff6a4')
Assessment.place(relx=.05, rely=.60, anchor="w")
```

```
Treat_Plan = ttk.Label(Consultn, text = 'TREATMENT PLAN: ', font = "Helvetica 12 bold",
background = '#fff6a4')
Treat_Plan.place(relx=.55, rely=.60, anchor="w")

Consult_Fees = ttk.Label(Consultn, text = 'CONSULTATION FEES: ', font = "Helvetica 12
bold", background = '#fff6a4')
Consult_Fees.place(relx=.05, rely=.80, anchor="w")


# Entry Buttons

ConsultnID_EB = Entry(Consultn, width = 40, bd = 5, font = 'Helvetica 12 bold')
ConsultnID_EB.place(relx=.20, rely=.25, anchor="w")

AppointmentID_EB = Entry(Consultn, width = 40, bd = 5, font = 'Helvetica 12 bold')
AppointmentID_EB.place(relx=.70, rely=.25, anchor="w")

MedHistory_EB = Text(Consultn, width = 40, height = 3, bd = 5, font = 'Helvetica 12 bold')
MedHistory_EB.place(relx=.20, rely=.43, anchor="w")

Diagnosis_EB = Text(Consultn, width = 40, height = 3, bd = 5, font = 'Helvetica 12 bold')
Diagnosis_EB.place(relx=.70, rely=.43, anchor="w")

Assessment_EB = Text(Consultn, width = 40, height = 3, bd = 5, font = 'Helvetica 12 bold')
Assessment_EB.place(relx=.20, rely=.63, anchor="w")

TreatPlan_EB = Text(Consultn, width = 40, height = 3, bd = 5, font = 'Helvetica 12 bold')
TreatPlan_EB.place(relx=.70, rely=.63, anchor="w")

ConsultFees_EB = Entry(Consultn, width = 40, bd = 5, font = 'Helvetica 12 bold')
ConsultFees_EB.place(relx=.20, rely=.80, anchor="w")

# Create Buttons
enter_btn = Button(Consultn, text='ENTER RECORD', width=20, command=Enter)
enter_btn.place(relx=.10, rely=.90)

fetch_btn = Button(Consultn, text='FETCH RECORD', width=20, command=Fetch)
fetch_btn.place(relx=.70, rely=.90)

remove_btn = Button(Consultn, text='DELETES RECORD', width=20, command=Remove)
remove_btn.place(relx=.25, rely=.90)

update_btn = Button(Consultn, text='UPDATES RECORD', width=20, command=Update)
update_btn.place(relx=.40, rely=.90)

clear_btn = Button(Consultn, text='CLEAR RECORD', width=20, command=Clear)
```

```
clear_btn.place(relx=.55, rely=.90)

# Quit Button

quit_btn = Button(Consultn, text="Quit", width=8, command=Consultn.quit)
quit_btn.place(relx = .90, rely = .90)

# Start program

Consultn.mainloop()
```