Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ОТЧЕТ

по лабораторной работе №6

на тему

Создание приложения для базы данных

Больница

Студент: Д.И. Ковальчук

Преподаватель: Д.В. Куприянова

МИНСК 2024

1 Цель работы

Создание прикладной программы для работы с базой данных и выполняющей заданные транзакции, а также реализация механизма работы с базой данных (добавление новых данных в таблицу, удаление, обновление) посредством любой среды и языка программирования.

2 Порядок выполнения работы

1) Реализовать механизм работы с базой данных (добавление новых данных в таблицу, удаление, обновление).

2) Реализовать интерфейс вывода запросов из 4 и 5 лабораторных работ.

3) Реализовать интерфейс пользователя.

4) Оформить *отчет.*

3 Выполнение работы

3.1 Используемые технологии

Язык программирования: Python.

Для работы с базой данных использовалась библиотека “psycopg2”.

Для реализации графического интерфейса использовалась библиотека “Tkinter”.

3.2 Системные требования

Операционная система: Windows 11, СУБД: PostgreSQL, интерпретатор Python версии 3.11 и выше

3.3 Руководство пользователя

3.3.1 Запуск приложения

Для запуска приложения необходимо задействовать интерпретатор, написав в командной строке:

*$ python hospital.py*

3.3.2 Начало работы

После запуска приложения будет отображено окно, приведённое на рисунке 3.1.

Окно разделено на две части. Слева расположено окно таблицы, содержащее информацию, получаемую из базы данных.

На панели справа расположены меню выбора таблицы, кнопки взаимодействия с таблицей, а также кнопка вывода списка запросов.



Рисунок 3.1 – Основное окно программы

3.3.3 Выбор таблицы

Выбор таблицы осуществляется посредством выбора пункта выпадающего меню в правой верхней части окна (рис. 3.2).



Рисунок 3.2 – Выбор таблицы

На рисунке 3.3 отображён результат выбора таблицы service. Название выбранной таблицы теперь отображается в заголовке выпадающего меню.

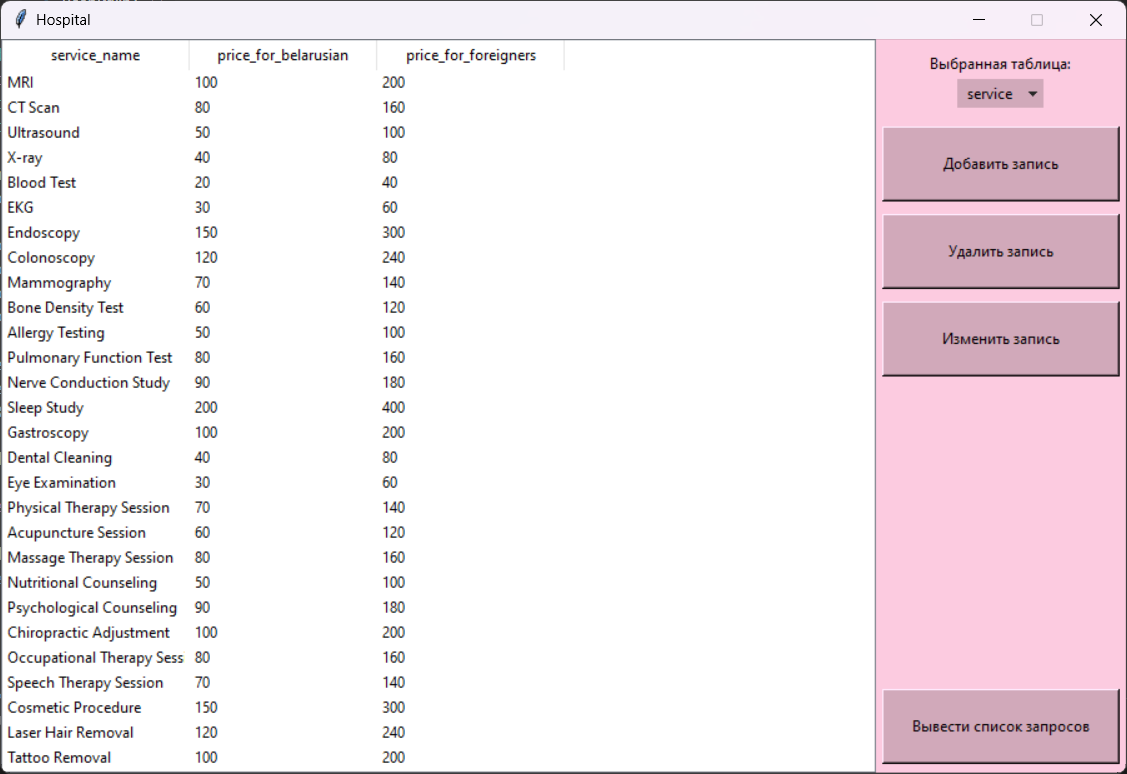


Рисунок 3.3 – Результат выбора таблицы service

3.3.4 Добавление записи

Добавление записи осуществляется посредством нажатия на соответствующую кнопку. После нажатия будет открыто новое окно с полями для ввода, оно представлено на рисунке 3.4.

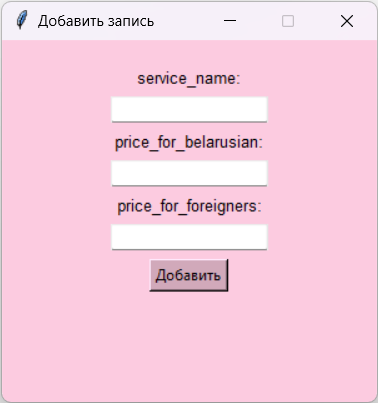


Рисунок 3.4 – Окно добавления записи

После заполнения полей в новом окне, необходимо нажать на кнопку «Добавить». В качестве примера добавлена новая запись об услуге: ДНК-тест, цена для белорусских граждан на которую 330 рублей и цена для иностранцев 785 рублей. Результат изображён на рисунке 3.5.

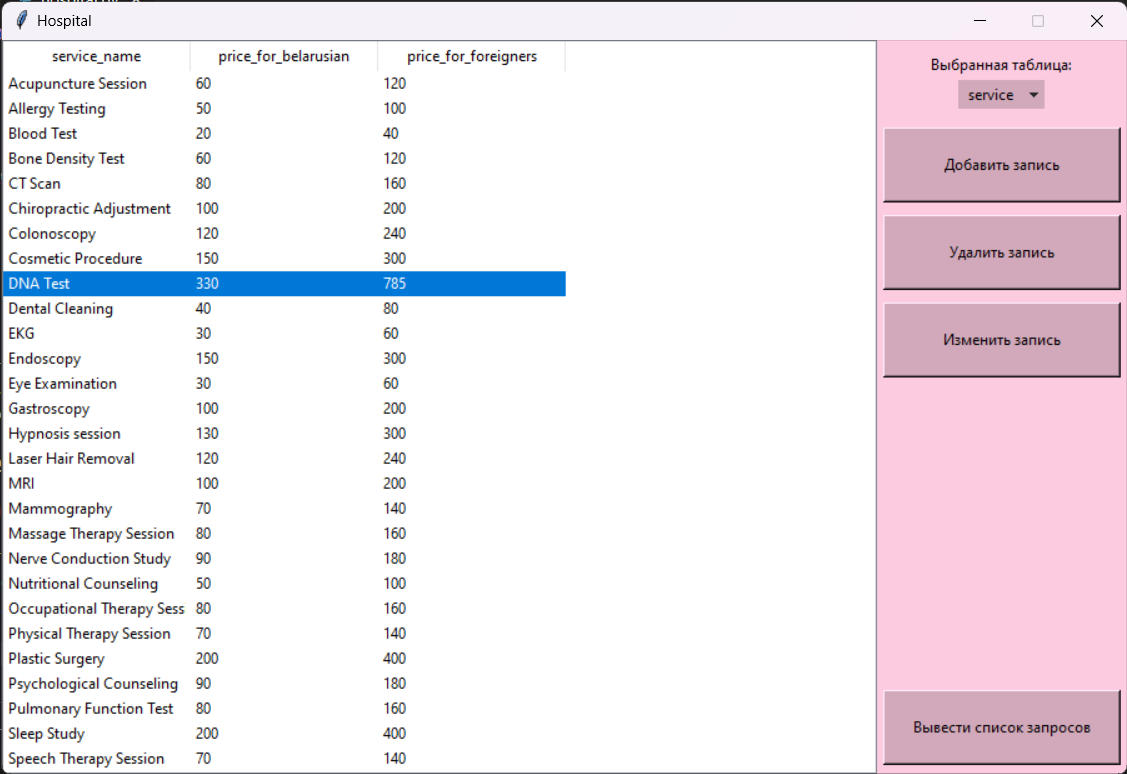


Рисунок 3.5 – Результат добавления записи

3.3.5 Удаление записи

Удаление записи осуществляется посредством её выделения и нажатия на соответствующую кнопку. В качестве примера в таблице doctor\_service удаляется запись о том, что доктор с doctor\_id = 4 предоставляет услугу МРТ.

На рисунках 3.6 и 3.7 изображена таблица до и после удаления записи соответственно.

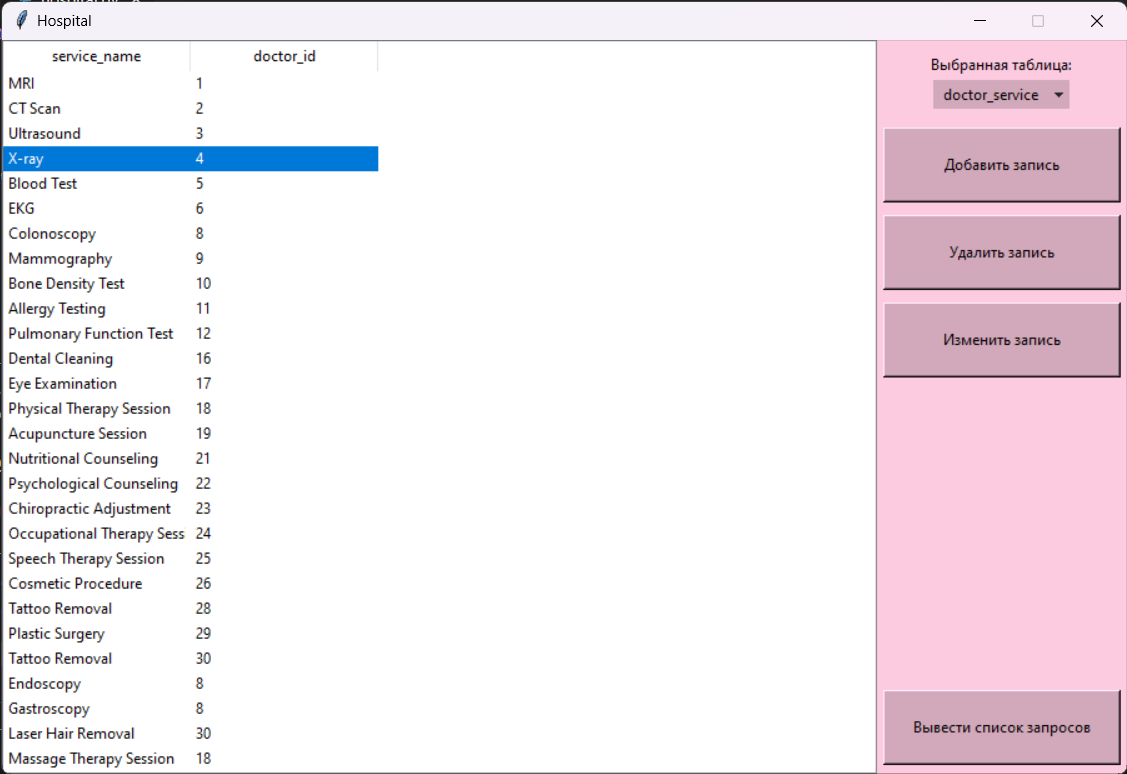


Рисунок 3.6 – Таблица doctor\_service до удаления записи

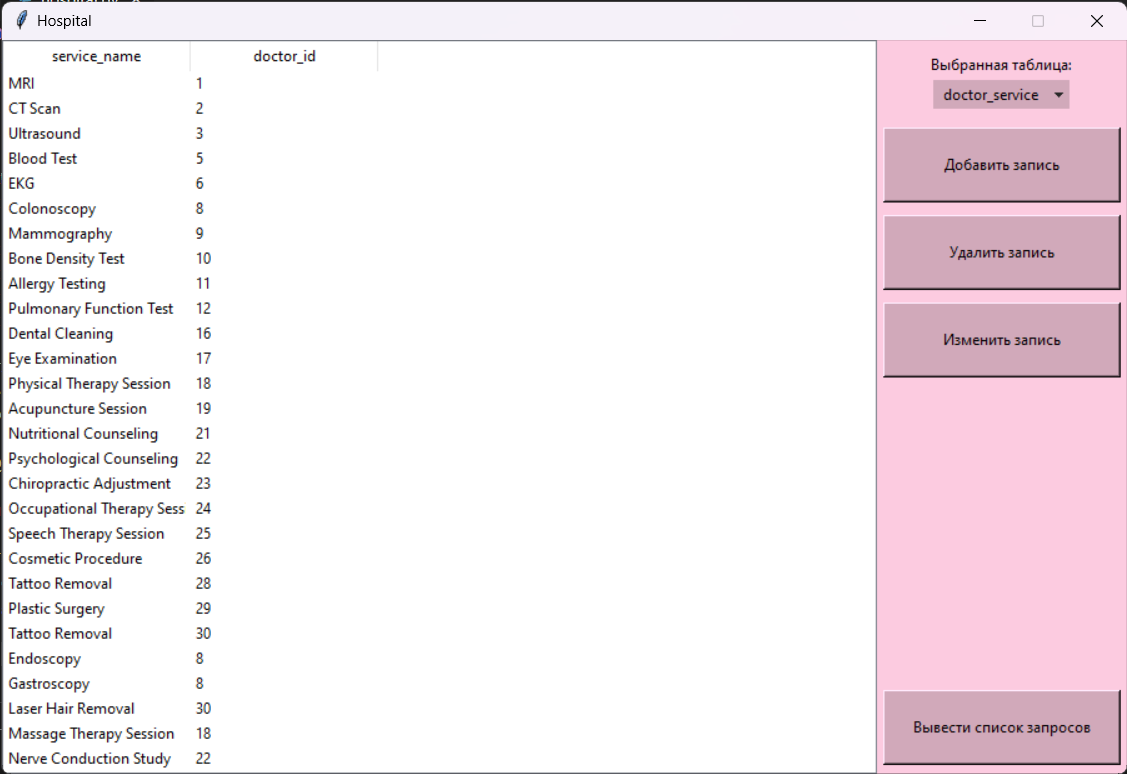


Рисунок 3.7 – Таблица doctor\_service после удаления записи

3.3.6 Изменение записи

Изменение записи осуществляется посредством её выделения и нажатия на соответствующую кнопку. После нажатия будет открыто новое окно с полями ввода, предварительно заполненными исходными значениями изменяемой записи. Данное окно приведено на рисунке 3.8.

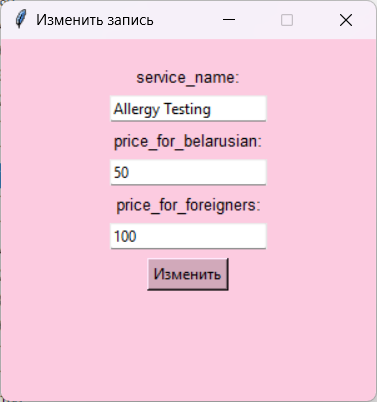


Рисунок 3.8 – Таблица doctor\_service после удаления записи

В качестве примера для изменения в таблице service выбрана запись с service\_name = “Allergy Testing”. На рисунках 3.9 и 3.10 изображена таблица до и после изменения записи соответственно.

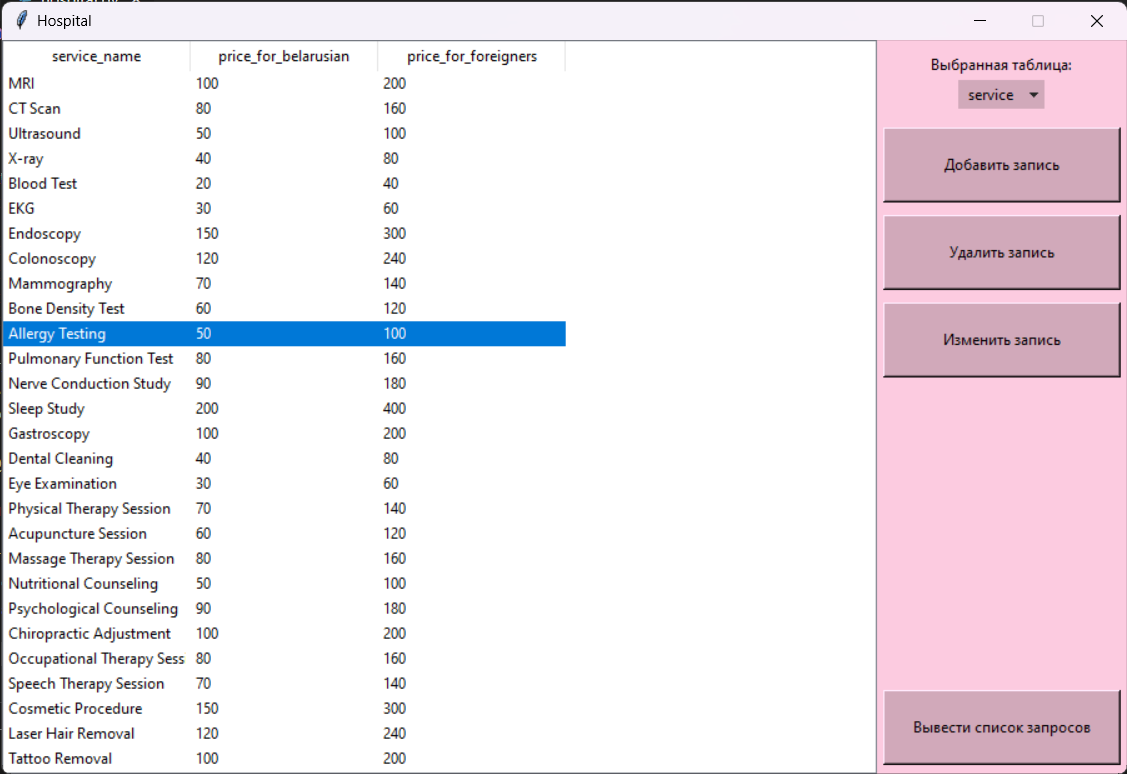


Рисунок 3.9 – Таблица service до изменения записи

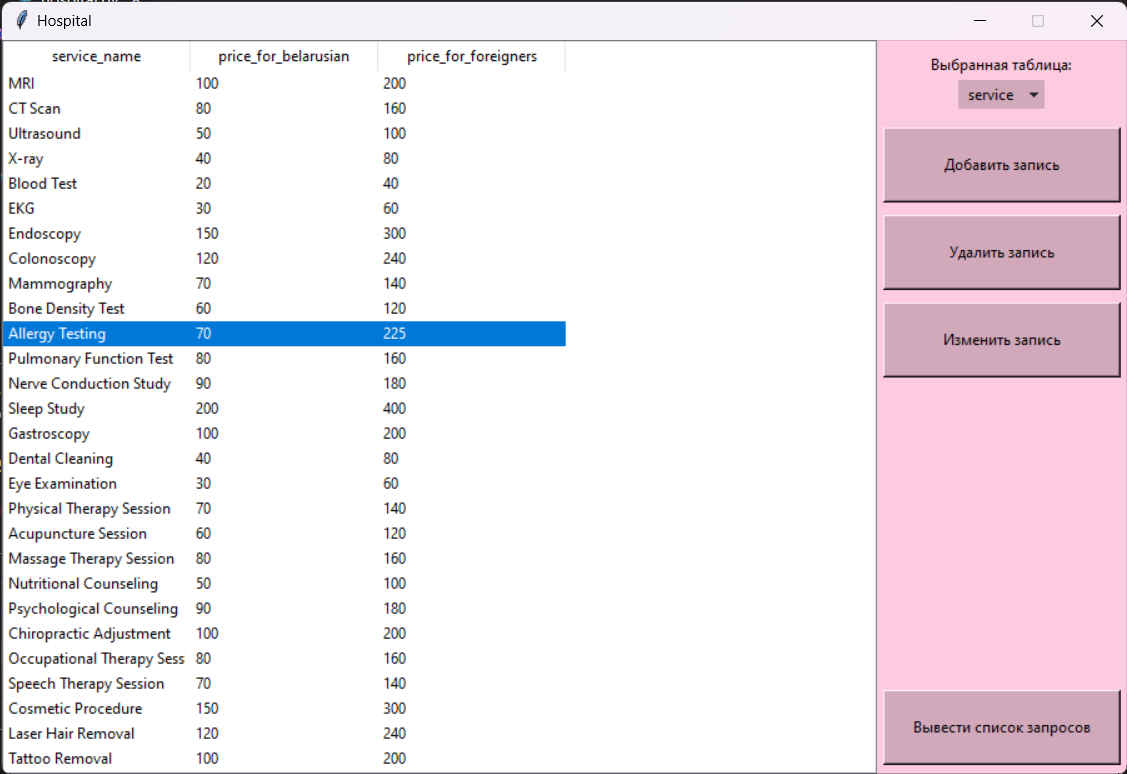


Рисунок 3.10 – Таблица service после изменения записи

3.3.7 Список запросов

При нажатии кнопку “Вывести список запросов” будет открыто новое окно со списком запросов из лабораторных работ №4 и №5 в виде кнопок. При нажатии на одну из кнопок запросов таблица результата будет отображёна в основном окне.

Окно списка запросов изображено на рисунке 3.11.

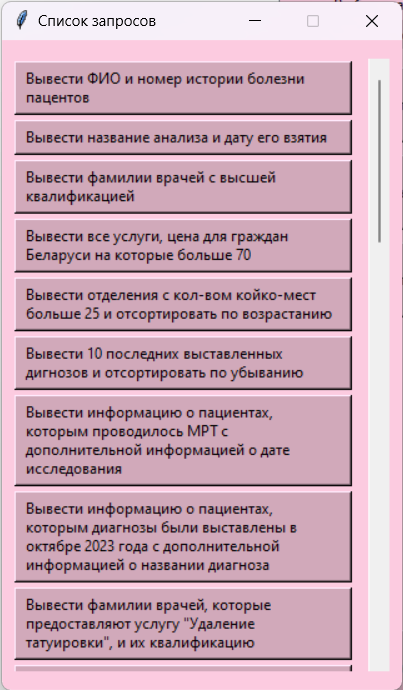


Рисунок 3.11 – Окно списка запросов

В качестве примера был выбран запрос «Вывести информацию об отделениях с наименьшим количеством койко-мест». Результат нажатия на соответствующую кнопку запроса в окне списка запросов изображён на рисунке 3.12.

Стоит также обратить внимание на то, что после вывода результата запроса, кнопки для работы с записями таблицы «Добавить запись», «Удалить запись», «Изменить запись» становятся неактивными.

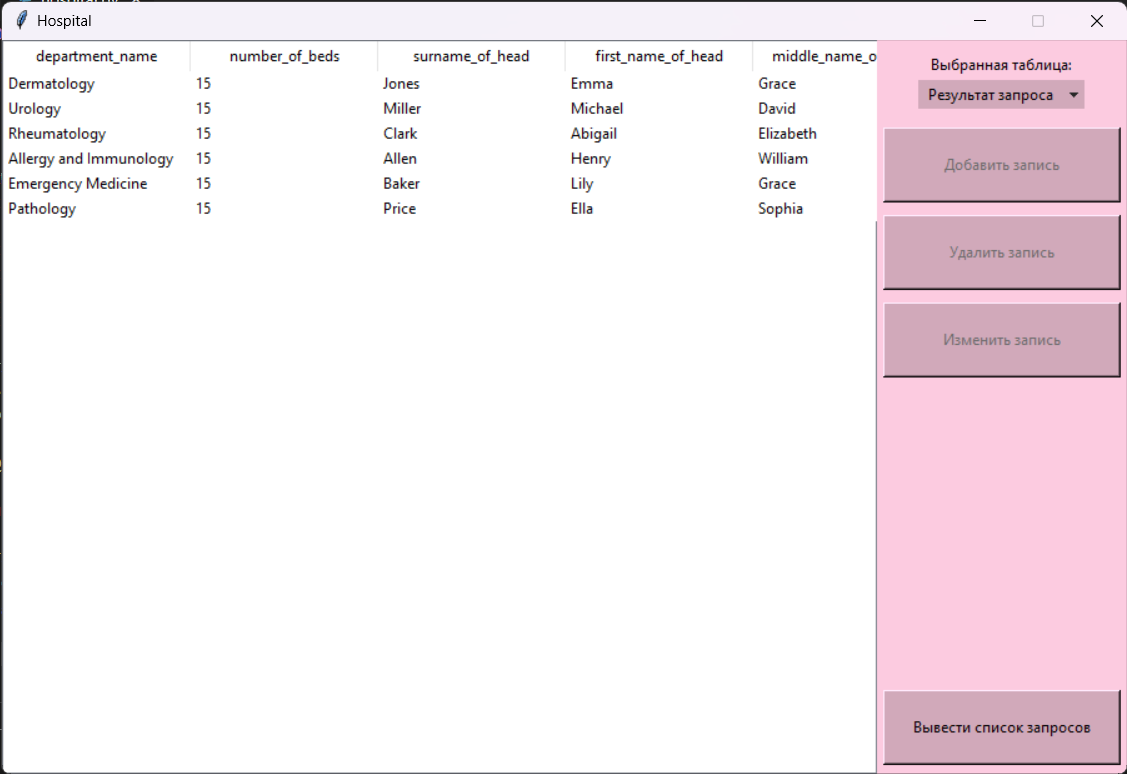


Рисунок 3.12 – Результат выполнения запроса

3.4 Листинг кода

Листинг кода приведён в приложении А.

4 Вывод

В ходе лабораторной работы были разработана прикладная программа для работы с базой данных, где реализованы основной механизм работы с БД (добавление, удаление, обновление данных в таблице), интерфейс вывода запросов из 4 и 5 лабораторных работ, а также графический интерфейс для удобного использования приложения рядовым пользователем.

**ПРИЛОЖЕНИЕ А**

Листинг кода

Файл *database.py*:

import psycopg2

def initConnection(username = 'postgres', password = 'root', hostname =

'localhost', port = '5432', dbname = 'hospital'):

    try:

        conn = psycopg2.connect(host = hostname, port = port, dbname = dbname,

user = username, password = password, client\_encoding = 'utf-8')

    except:

        pass

    conn.autocommit = True

    return conn

def runQuery(conn, query):

    with conn.cursor() as cursor:

        cursor.execute(query)

        data = cursor.fetchall()

        colnames = [desc[0] for desc in cursor.description]

        return (data, colnames)

def getAllTables(conn):

    with conn.cursor() as cursor:

        cursor.execute('SELECT table\_name FROM information\_schema.tables WHERE table\_schema=\'public\'')

        data = cursor.fetchall()

        return data

def getTableData(conn, tableName):

    with conn.cursor() as cursor:

        cursor.execute(f'SELECT \* FROM {tableName}')

        data = cursor.fetchall()

        colnames = [desc[0] for desc in cursor.description]

        return (data, colnames)

def addRecord(conn, tableName, keys, values):

    with conn.cursor() as cursor:

        try:

            cursor.execute(f'INSERT INTO {tableName} {keys} VALUES {values}')

            return None

        except:

            return "Убедитесь, что все поля заполнены правильно!"

def deleteRecord(conn, tableName, parameters):

    with conn.cursor() as cursor:

        try:

            cursor.execute(f'DELETE FROM {tableName} WHERE {parameters}')

            return None

        except:

            return "На данную запись существуют ссылки в других таблицах!"

def updateRecord(conn, tableName, setString, parameters):

    with conn.cursor() as cursor:

        try:

            cursor.execute(f'UPDATE {tableName} SET {setString} WHERE {parameters}')

            return None

        except:

            return "Убедитесь, что все поля заполнены правильно!"

def getButtonInfo():

    return (

        ['Вывести ФИО и номер истории болезни пацентов',

         'SELECT history\_number, first\_name, middle\_name, surname FROM patient'],

        ['Вывести название анализа и дату его взятия',

         'SELECT test\_name, date\_ FROM test'],

        ['Вывести фамилии врачей с высшей квалификацией',

         'SELECT surname, qualification FROM doctor WHERE qualification LIKE \'Higher\''],

        ['Вывести все услуги, цена для граждан Беларуси на которые больше 70',

         'SELECT \* FROM service WHERE price\_for\_belarusian > 70'],

        ['Вывести отделения с кол-вом койко-мест больше 25 и отсортировать по возрастанию',

         'SELECT \* FROM department WHERE number\_of\_beds > 25 ORDER BY number\_of\_beds ASC'],

        ['Вывести 10 последних выставленных дигнозов и отсортировать по убыванию',

         'SELECT \* FROM diagnosis ORDER BY date\_ DESC LIMIT 10'],

        ['Вывести информацию о пациентах, которым проводилось МРТ с дополнительной информацией о дате исследования',

         'SELECT patient.history\_number, type\_of\_treatment, surname, first\_name, middle\_name, department\_name, test\_name, date\_ FROM patient CROSS JOIN test WHERE patient.history\_number = test.history\_number AND test\_name LIKE \'MRI\''],

        ['Вывести информацию о пациентах, которым диагнозы были выставлены в октябре 2023 года с дополнительной информацией о названии диагноза',

         'SELECT patient.history\_number, type\_of\_treatment, surname, first\_name, middle\_name, department\_name, diagnosis\_name, date\_ FROM patient CROSS JOIN diagnosis WHERE patient.history\_number = diagnosis.history\_number AND date\_ >= \'2023-10-01\' AND date\_ <= \'2023-10-31\''],

        ['Вывести фамилии врачей, которые предоставляют услугу "Удаление татуировки", и их квалификацию',

         'SELECT doctor.doctor\_id, surname, qualification, service\_name FROM doctor INNER JOIN doctor\_service ON (doctor.doctor\_id = doctor\_service.doctor\_id) AND service\_name LIKE \'Tattoo Removal\''],

        ['Вывести информацию о пациентах, которые лежат в отделениях, где меньше 25 койко-мест',

         'SELECT history\_number, surname, type\_of\_treatment, patient.department\_name, number\_of\_beds FROM patient INNER JOIN department ON (patient.department\_name = department.department\_name) WHERE number\_of\_beds < 25'],

        ['Вывести информацию о пациентах, которым предоставлена консультация и поставлен диагноз (LEFT OUTER JOIN)',

         'SELECT patient.history\_number, surname, type\_of\_treatment, department\_name, diagnosis\_name FROM patient LEFT OUTER JOIN diagnosis ON (patient.history\_number = diagnosis.history\_number) WHERE type\_of\_treatment LIKE \'Consultation\''],

        ['Вывести информацию о врачах и платных услугах, которые эти врачи предоставляют (LEFT OUTER JOIN)',

         'SELECT doctor.doctor\_id, surname, qualification, service\_name FROM doctor LEFT OUTER JOIN doctor\_service ON (doctor.doctor\_id = doctor\_service.doctor\_id)'],

        ['Вывести информацию о пациентах, которым предоставлена консультация и поставлен диагноз (RIGHT OUTER JOIN)',

         'SELECT \* FROM diagnosis RIGHT OUTER JOIN patient ON (diagnosis.history\_number = patient.history\_number) WHERE type\_of\_treatment LIKE \'Consultation\''],

        ['Вывести информацию о врачах с высшей квалификацией и платных услугах, которые эти врачи предоставляют (RIGHT OUTER JOIN)',

         'SELECT \* FROM doctor\_service RIGHT OUTER JOIN doctor ON (doctor\_service.doctor\_id = doctor.doctor\_id) WHERE qualification LIKE \'Higher\''],

        ['Вывести список пациентов с дополнительной информацией об осмотре пациента врачом в период с 01.01.2023 по 15.01.2023',

         'SELECT \* FROM patient FULL OUTER JOIN doctor\_patient ON (patient.history\_number = doctor\_patient.history\_number) WHERE date\_ >= \'2023-01-01\' AND date\_ <= \'2023-01-15\''],

        ['Вывести список пациентов, которым были выставлены последние 10 диагнозов с дополнительной информацией о диагнозе',

         'SELECT \* FROM patient FULL OUTER JOIN diagnosis ON (patient.history\_number = diagnosis.history\_number) WHERE diagnosis\_id IS NOT NULL ORDER BY date\_ DESC LIMIT 10'],

        ['Вывести информации об общей стоимости услуг для граждан Беларуси, предоставляемых врачами с высшей категорией',

         'SELECT SUM(price\_for\_belarusian) AS total\_price FROM doctor INNER JOIN doctor\_service ON (doctor.doctor\_id = doctor\_service.doctor\_id) INNER JOIN service ON (doctor\_service.service\_name = service.service\_name) WHERE qualification LIKE \'Higher\''],

        ['Вывести среднюю стоимость платных услуг для иностранных граждан',

         'SELECT AVG(price\_for\_foreigners) AS average\_price FROM service'],

        ['Вывести кол-во анализов крови, выполненных в период с \'01.01.2024\' по \'20.01.2024\'',

         'SELECT COUNT(test\_name) AS count\_ FROM test WHERE test\_name LIKE \'Blood Test\' AND date\_ >= \'2024-01-01\' AND date\_ <= \'2024-01-20\''],

        ['Вывести информацию об отделениях с наименьшим кол-вом койко-мест',

         'SELECT \* FROM department WHERE number\_of\_beds = (SELECT MIN(number\_of\_beds) FROM department)'],

        ['Вывести информацию об отделениях с наибольшим кол-вом койко-мест',

         'SELECT \* FROM department WHERE number\_of\_beds = (SELECT MAX(number\_of\_beds) FROM department)'],

        ['Вывести информацию о количестве того, сколько раз был выполнен каждый из возможных анализов',

         'SELECT test\_name, COUNT(\*) AS number\_ FROM test GROUP BY test\_name'],

        ['Вывести информацию о кол-ве врачей каждой из категорий, при том, что кол-во врачей в каждой из выведенных категорий должно быть больше 5',

         'SELECT COUNT(doctor\_id), qualification FROM doctor GROUP BY qualification HAVING COUNT(doctor\_id) > 5'],

        ['Вывести пациентов, относящихся к отделениям кардиологии и педиатрии',

         'SELECT patient.history\_number, patient.surname, patient.department\_name FROM patient WHERE department\_name IN (\'Cardiology\', \'Pediatrics\')'],

        ['Вывести информацию об анализах, сделанных лаборантами с фамилией Johnson',

         'SELECT \* FROM test WHERE name\_of\_nurse = ANY (SELECT name\_of\_nurse FROM test WHERE name\_of\_nurse LIKE \'Johnson\')'],

        ['Вывести информацию об анализах, взятых у пацентов, чья история болезни имеет номер больше 1020',

         'SELECT \* FROM test WHERE history\_number <> ALL (SELECT history\_number FROM test WHERE history\_number < 1020)'],

        ['Вывести информацию о врачах, которые являются главными врачами отделений',

         'SELECT surname, speciality FROM doctor WHERE EXISTS (SELECT surname\_of\_head FROM department WHERE department.surname\_of\_head = doctor.surname)'],

        ['Вывести врачей, которые имеют вторую категорию, или проводят анализ крови, или осматривали пациентов 01.01.2024 (без повторений)',

         'SELECT doctor\_id FROM doctor WHERE qualification LIKE \'Second\' UNION SELECT doctor\_id FROM doctor\_service WHERE service\_name LIKE \'Blood Test\' UNION SELECT doctor\_id FROM doctor\_patient WHERE date\_ = \'2024-01-01\''],

        ['Вывести врачей, которые имеют вторую категорию, или проводят анализ крови, или осматривали пациентов 01.01.2024 (с повторениями)',

         'SELECT doctor\_id FROM doctor WHERE qualification LIKE \'Second\' UNION ALL SELECT doctor\_id FROM doctor\_service WHERE service\_name LIKE \'Blood Test\' UNION ALL SELECT doctor\_id FROM doctor\_patient WHERE date\_ = \'2024-01-01\''],

        ['Вывести врачей, которые имеют высшую категорию, выполняют удаление татуировок и проводили осмотр пациента 28.01.2023',

         'SELECT doctor\_id FROM doctor WHERE qualification LIKE \'Higher\' INTERSECT SELECT doctor\_id FROM doctor\_service WHERE service\_name LIKE \'Tattoo Removal\' INTERSECT SELECT doctor\_id FROM doctor\_patient WHERE date\_ > \'2023-01-01\''],

        ['Вывести номер истории болезни пациентов, которым делали анализ крови, которые не обращались за консультацией, и которых не осматривали после 15.01.2023',

         'SELECT history\_number FROM test WHERE test\_name LIKE \'Blood Test\' EXCEPT SELECT history\_number FROM patient WHERE type\_of\_treatment LIKE \'Consultation\' EXCEPT SELECT history\_number FROM doctor\_patient WHERE date\_ > \'2023-01-15\'']

    )

Файл *hospital.py*:

from database import \*

import tkinter as tk

from tkinter import ttk

from tkinter import messagebox

def initWindow():

    root = tk.Tk()

    root.title('Hospital')

    root.geometry("900x587+100+80")

    root.resizable(False, False)

    root.configure(background='#FCCBE0')

    newTable = addTable(root)

    selectedTable = addTableChoice(root)

    CRUDButtons = addCRUDButtons(root)

    addQueryChoice(root)

    return (root, newTable, CRUDButtons, selectedTable)

def addTable(parent):

    def onSelectElement(event):

        global selectedElement, selectedTable

        if selectedTable.get() != 'Результат запроса' and table.selection():

            selectedElement = table.selection()[0]

    def keyboardScroll(event):

        if event.keysym == 'Up':

            table.yview\_scroll(-1, "units")

        elif event.keysym == 'Down':

            table.yview\_scroll(1, "units")

        elif event.keysym == 'Left':

            table.xview\_scroll(-10, "units")

        elif event.keysym == 'Right':

            table.xview\_scroll(10, "units")

    table = ttk.Treeview(parent, show='headings', selectmode='browse')

    table.place(height=587, width=700, x=0, y=0)

    table.bind('<ButtonRelease-1>', onSelectElement)

    table.bind("<KeyPress>", keyboardScroll)

    return table

def addTableChoice(parent):

    def onSelect(event):

        global colnames

        data, colnames = getTableData(conn, selectedTable.get())

        updateTable(data, colnames)

        blockCRUD(False)

    frame = tk.Frame(parent, background='#FCCBE0')

    label = ttk.Label(frame, text='Выбранная таблица:', background='#FCCBE0')

    label.pack()

    selectedTable = tk.StringVar(parent)

    style = ttk.Style()

    style.configure('TMenubutton', background='#D1A9BA', borderwidth=2)

    options = getAllTables(conn)

    options.append(options[0])

    drop = ttk.OptionMenu(frame, selectedTable, \*options, command=onSelect)

    drop.pack(pady=3)

    frame.place(height=100, width=200, x=700, y=10)

    selectedTable.set('Выбор')

    return selectedTable

def addCRUDButtons(parent):

    frame = tk.Frame(parent, background='#FCCBE0', pady=5)

    addButton = tk.Button(frame, text="Добавить запись", height=80, background='#D1A9BA',

                          command= lambda: createAddWindow(parent), state=tk.DISABLED)

    removeButton = tk.Button(frame, text="Удалить запись", height=80, background='#D1A9BA',

                             command= lambda: createRemoveWindow(parent), state=tk.DISABLED)

    updateButton = tk.Button(frame, text="Изменить запись", height=80, background='#D1A9BA',

                             command=lambda: createUpdateWindow(parent), state=tk.DISABLED)

    frame.place(height=220, width=200, x=700, y=65)

    addButton.place(height=60, width=190, x=5, y=0)

    removeButton.place(height=60, width=190, x=5, y=70)

    updateButton.place(height=60, width=190, x=5, y=140)

    return (addButton, removeButton, updateButton)

def createAddWindow(parent):

    window = tk.Toplevel(parent, background='#FCCBE0', pady=20)

    window.title("Добавить запись")

    height = len(colnames) \* 80 + 50

    window.geometry(f'300x{height}+1000+100')

    window.resizable(False, False)

    def addValues():

        values = [textField.get() for textField in textFields]

        keyString = ''

        for column in colnames[:-1]:

            keyString += f'{column}, '

        keyString += colnames[-1]

        keyString = f"({keyString})"

        result = addRecord(conn, selectedTable.get(), keyString, tuple(values))

        if result is None:

            table.insert('', tk.END, values=values)

            sortRecords(0, False)

            window.destroy()

        else:

            messagebox.showerror("Ошибка", result)

    mainFrame = tk.Frame(window, background='#FCCBE0')

    mainFrame.pack()

    textFields = []

    for field in colnames:

        frame = tk.Frame(mainFrame, background='#FCCBE0')

        frame.pack()

        label = ttk.Label(frame, text=f'{field}:', background='#FCCBE0', font=('Helvetica', 10))

        label.pack()

        textField = ttk.Entry(frame, background='#FCCBE0')

        textField.pack(pady=5)

        textFields.append(textField)

    confirm = tk.Button(window, text="Добавить", command=addValues, background='#D1A9BA')

    confirm.pack(pady=2)

def createRemoveWindow(parent):

    if selectedElement is None:

        return

    values = table.item(selectedElement, 'values')

    paramString = ''

    for i in range(len(colnames) - 1):

        paramString += f'{colnames[i]} = \'{values[i]}\' AND '

    paramString += f'{colnames[-1]} = \'{values[-1]}\''

    result = deleteRecord(conn, selectedTable.get(), paramString)

    if result is None:

        table.delete(selectedElement)

    else:

        messagebox.showerror("Ошибка", result)

def createUpdateWindow(parent):

    if selectedElement is None:

        return

    window = tk.Toplevel(parent, background='#FCCBE0', pady=20)

    window.title("Изменить запись")

    height = len(colnames) \* 80 + 50

    window.geometry(f'300x{height}+1000+100')

    window.resizable(False, False)

    values = table.item(selectedElement, 'values')

    paramString = ''

    for i in range(len(colnames) - 1):

        paramString += f'{colnames[i]} = \'{values[i]}\' AND '

    paramString += f'{colnames[-1]} = \'{values[-1]}\''

    def updateValues():

        newValues = [textField.get() for textField in textFields]

        setString = ''

        for i in range(len(colnames) - 1):

            setString += f'{colnames[i]} = \'{newValues[i]}\', '

        setString += f'{colnames[-1]} = \'{newValues[-1]}\''

        result = updateRecord(conn, selectedTable.get(), setString, paramString)

        if result is None:

            table.item(selectedElement, value=newValues)

            table.update\_idletasks()

            window.destroy()

        else:

            messagebox.showerror("Ошибка", result)

    mainFrame = tk.Frame(window, background='#FCCBE0')

    mainFrame.pack()

    textFields = []

    count = 0

    for field in colnames:

        frame = tk.Frame(mainFrame, background='#FCCBE0')

        frame.pack()

        label = ttk.Label(frame, text=f'{field}:', background='#FCCBE0', font=('Helvetica', 10))

        label.pack()

        textField = ttk.Entry(frame, background='#FCCBE0')

        textField.pack(pady=5)

        textField.insert(0, values[count])

        textFields.append(textField)

        count += 1

    confirm = tk.Button(window, text="Изменить", command=updateValues, background='#D1A9BA')

    confirm.pack(pady=2)

def addQueryChoice(parent):

    choiceButton = tk.Button(parent, text="Вывести список запросов", background='#D1A9BA',

                             command= lambda: queryChoiceWindow(parent))

    choiceButton.place(height=60, width=190, x=705, y=520)

def queryChoiceWindow(parent):

    def createCallback(query):

        def callback():

            global colnames

            data, colnames = runQuery(conn, query)

            updateTable(data, colnames)

            blockCRUD(True)

            selectedTable.set('Результат запроса')

            window.destroy()

        return callback

    def configScroll(event):

        canvas.configure(scrollregion=canvas.bbox("all"))

    def onScroll(event):

        if 'canvas' in str(event.widget):

            canvas.yview\_scroll(int(-1\*(event.delta/120)), "units")

    window = tk.Toplevel(parent, background='#FCCBE0')

    window.title("Список запросов")

    window.geometry(f'320x520+1000+100')

    window.resizable(False, False)

    canvas = tk.Canvas(window, background='#FCCBE0')

    frame = tk.Frame(canvas, background='#FCCBE0')

    scroll = ttk.Scrollbar(canvas, orient=tk.VERTICAL, command=canvas.yview)

    canvas.configure(yscrollcommand=scroll.set, highlightthickness=0)

    scroll.pack(side=tk.RIGHT, fill=tk.Y)

    btnsInfo = getButtonInfo()

    for entry in btnsInfo:

        callback = createCallback(entry[1])

        btn1 = tk.Button(frame, text=entry[0], padx=5, pady=2, background='#D1A9BA',

                         cursor='hand2', width=260, justify='left',

                         anchor='w', wraplength=250, command=callback)

        btn1.pack(pady=2)

    canvas.create\_window((0,0), window=frame, width=270)

    frame.bind("<Configure>", configScroll)

    canvas.configure(scrollregion=canvas.bbox('all'), background='#FCCBE0',)

    canvas.place(height=490, width=300, x=10, y=15)

    canvas.update\_idletasks()

    canvas.yview\_moveto(0)

    canvas.bind\_all("<MouseWheel>", onScroll)

def updateTable(data, columns):

    global table, selectedElement

    clearTable()

    selectedElement = None

    table['columns'] = list(range(0, len(columns)))

    count = 0

    for column in columns:

        table.column(str(count), minwidth=50, width=150, stretch=tk.NO)

        table.heading(str(count), text=column, command=lambda count=count: sortRecords(count, False))

        count += 1

    for record in data:

        table.insert('', tk.END, values=record)

def clearTable():

    global table

    for item in table.get\_children():

        table.delete(item)

def sortRecords(column, rev):

    global table

    records = [(table.set(k, column), k) for k in table.get\_children("")]

    try:

        records = [(float(record[0]), record[1]) for record in records]

    except ValueError:

        pass

    if all(isinstance(record[0], (int, float)) for record in records):

        records.sort(key=lambda x: float(x[0]), reverse=rev)

    else:

        records.sort(key=lambda x: str(x[0]), reverse=rev)

    for index,  (\_, k) in enumerate(records):

        table.move(k, "", index)

    table.heading(column, command=lambda: sortRecords(column, not rev))

def blockCRUD(block):

    global CRUDbuttons

    for button in CRUDbuttons:

        button.config(state=tk.DISABLED if block else tk.NORMAL)

# ----------------------------------

selectedElement = None

conn = initConnection()

root, table, CRUDbuttons, selectedTable = initWindow()

colnames = None

root.mainloop()