

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ

ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних систем**

**Лабораторна робота №2**

з дисципліни **Бази даних і засоби управління**

*на тему: “Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL”*

Виконала:

студентка ІII курсу

групи КВ-91

Кузьмич А.А.

Перевірив:

Павловський В. І.

Київ – 2021

# Постановка задачі

*Метою роботи* є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

*Загальне завдання* роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових, – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання- контролер).

**Інформація про програму**

Використана мова програмування: Python 3.10

Використані бібліотеки: psycopg2 (для зв’язку з СУБД), time (для виміру часу запиту пошуку для завдання 3)

Використаний шаблон проектування: MVC.

# Відомості про обрану предметну галузь з лабораторної роботи №1

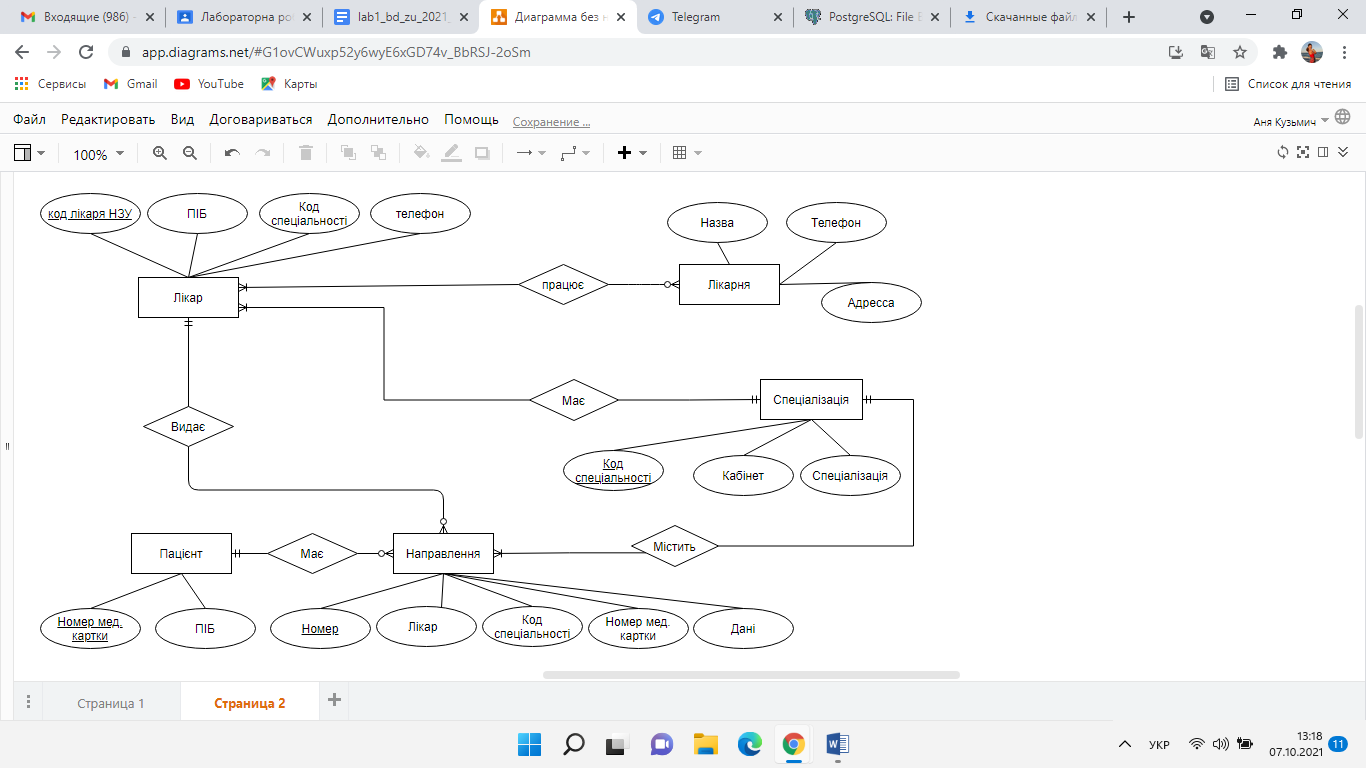


Рисунок 1 - ER-діаграма побудована за нотацією “Пташиної лапки (Crow’s foot)”

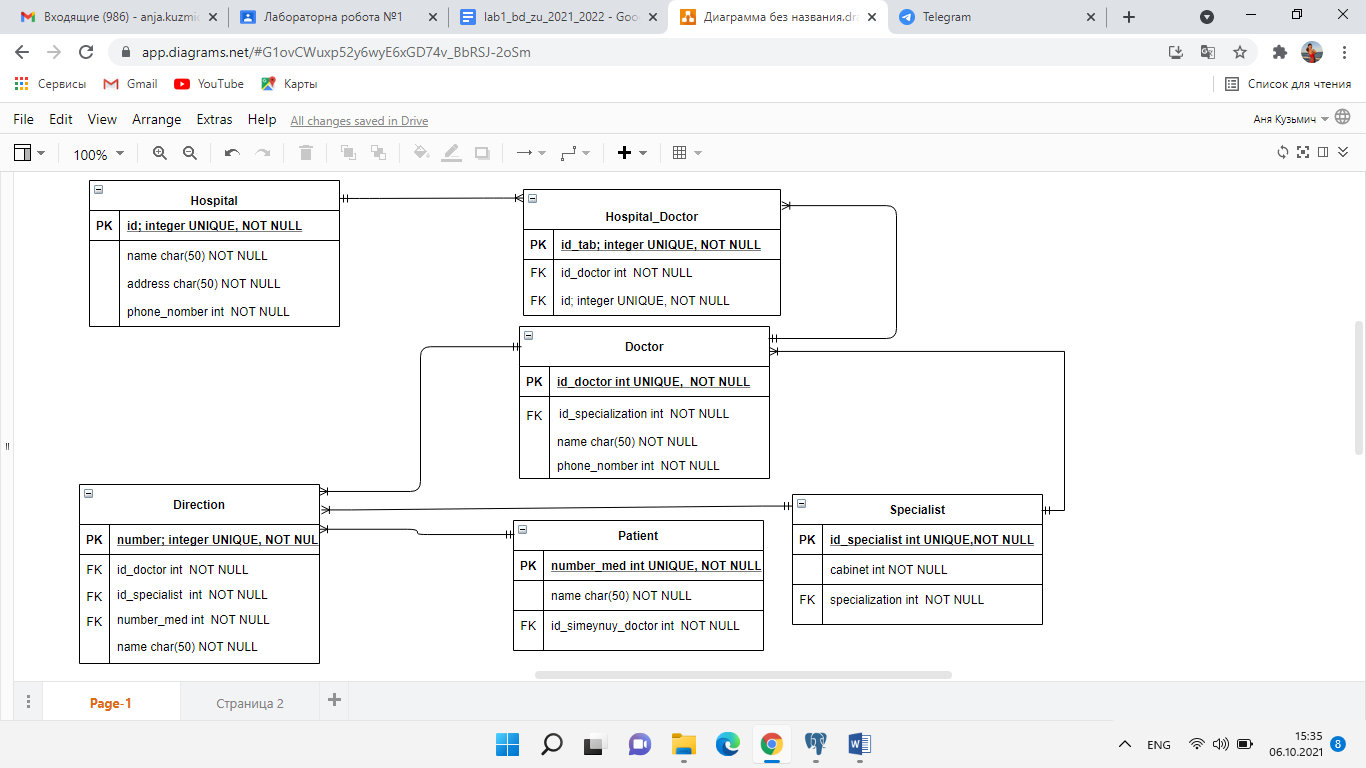


Рисунок 2 - Схема бази даних

Таблиця 1 - Опис структури БД.

|  |  |  |
| --- | --- | --- |
| Сутність | Атрибут | Тип атрибуту |
| **Hospital** - *містить дані про лікарню* | **id** *- унікальний ідентифікатор лікарні*  **name** *- назва лікарні*  **phone** *– телефонний номер лікарні*  **address** *- фізична адреса лікарні* | **integer** *(числовий)*  **character varying** *(рядок)*  **integer** *(числовий)*  **character varying** *(рядок)* |
| **Doctor** - *містить у собі всі дані лікарів* | **id\_doctor** *- унікальний ідентифікатор лікаря*  **name** *– ПІБ лікаря*  **id\_specialist** *- ідентифікатор спеціальності*  **phone** *- телефонний номер лікаря* | **integer** *(числовий)*  **character varying** *(рядок)*  **integer** *(числовий)*  **integer** *(числовий)* |
| **Patient** - *зберігає дані про пацієнтів* | **Number\_med** *- унікальний номер медичної картки*  **Name\_p** *– ПІБ пацієнта* | **integer** *(числовий)*  **character varying** *(рядок)* |
| **Direction** - *містить інформацію про обстеження пацієнтів.* | **number** *- унікальний ідентифікатор направлення*  **Number\_med** *- унікальний номер медичної картки*  **id\_doctor** *- унікальний ідентифікатор лікаря*  **id\_specialist -** *унікальний ідентифікатор спеціальності*  **data** *– дані обстеження* | **integer** *(числовий)*  **integer** *(числовий)*  **integer** *(числовий)*  **integer** *(числовий)*  **character varying** *(рядок* |
| **Specialist** - *містить інформацію про спеціальності лікарів* | **id\_specialist -** *унікальний ідентифікатор спеціальності*  **cabinet –** *номер кабінету*  **specialization –** *назва спеціальності* | **integer** *(числовий)*  **integer** *(числовий)*  **character varying** *(рядок* |
| **Hospital\_ Doctor**  *містить інформацію про відповідність лікаря і лікарні, де він працює.* | **id\_tab - -** *унікальний ідентифікатор таблиці відповідності*  **id\_doctor***- унікальний ідентифікатор лікаря*  **id***- унікальний ідентифікатор лікарні* | **integer** *(числовий)*  **integer** *(числовий)*  **integer** *(числовий)* |

**Структура програми**

За шаблоном проектування MVC, додаток складається з таких модулей:

*model.py* — виконує операції з БД – SELECT, INSERT, DELETE, UPDATE та, більш складні операції з БД

*controller.py* — головне та допоміжні меню, для зручного керуванням БД

*config.py* — файл з вхідними даними до БД

*main.py* — точка входу в програму, та підключення до БД

# Схема меню користувача

# 

На знімку екрану термінала продемонстовано початкове меню, де можна побачити функції які можна виконати з БД. Кожна команда запускає відповідну функцію з файлу Controller, яка в подальшому передає аргументи у функцію файлу Model, яка в свою чергу сформовує і здійснює запит до бази даних

***Методи реалізовані до пункту 1 завдання лабораторної роботи:***

1. Insert data in table — викликає функцію вставки даних у таблицю;
2. Edit data in table — викликає функцію редагування даних у таблиці;
3. Delete data from table — викликає функцію видалення даних у таблиці;
4. Print rows — викликає функцію виводу даних з таблиці.

Кожна з функцій викликає допоміжну функцію select\_table() яка дозволяє користувачеві вибрати таблицю

## Метод реалізований до пункту 2 завдання лабораторної роботи:

1. Generate random data — дозволяє користувачеві заповнити таблицю або всю базу даних випадково згенереваними данними.

## Метод реалізований до пункту 3 завдання лабораторної роботи:

1. Search from tables — викликає функцію пошуку даних у таблицях за атрибутами та поєднання таблиць за ключем.

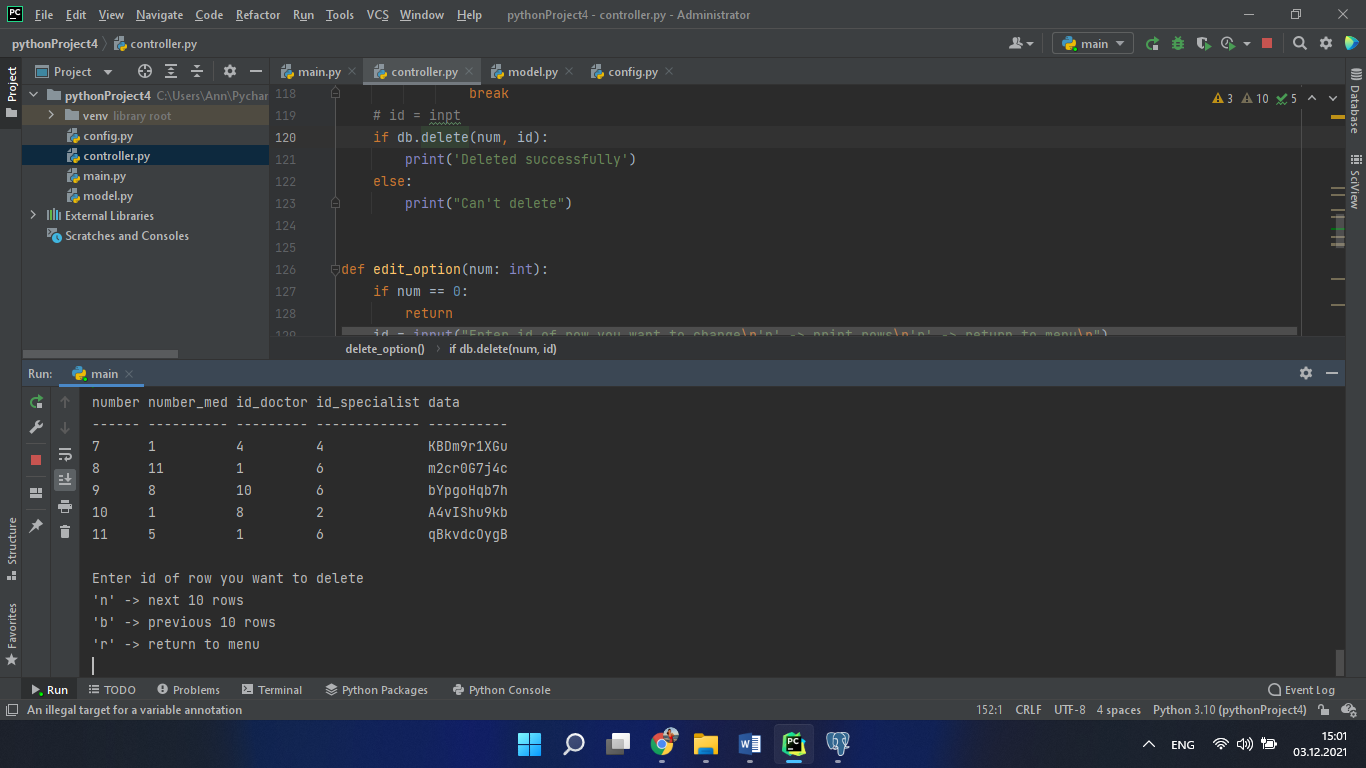
# Завдання 1

## Запит на видалення

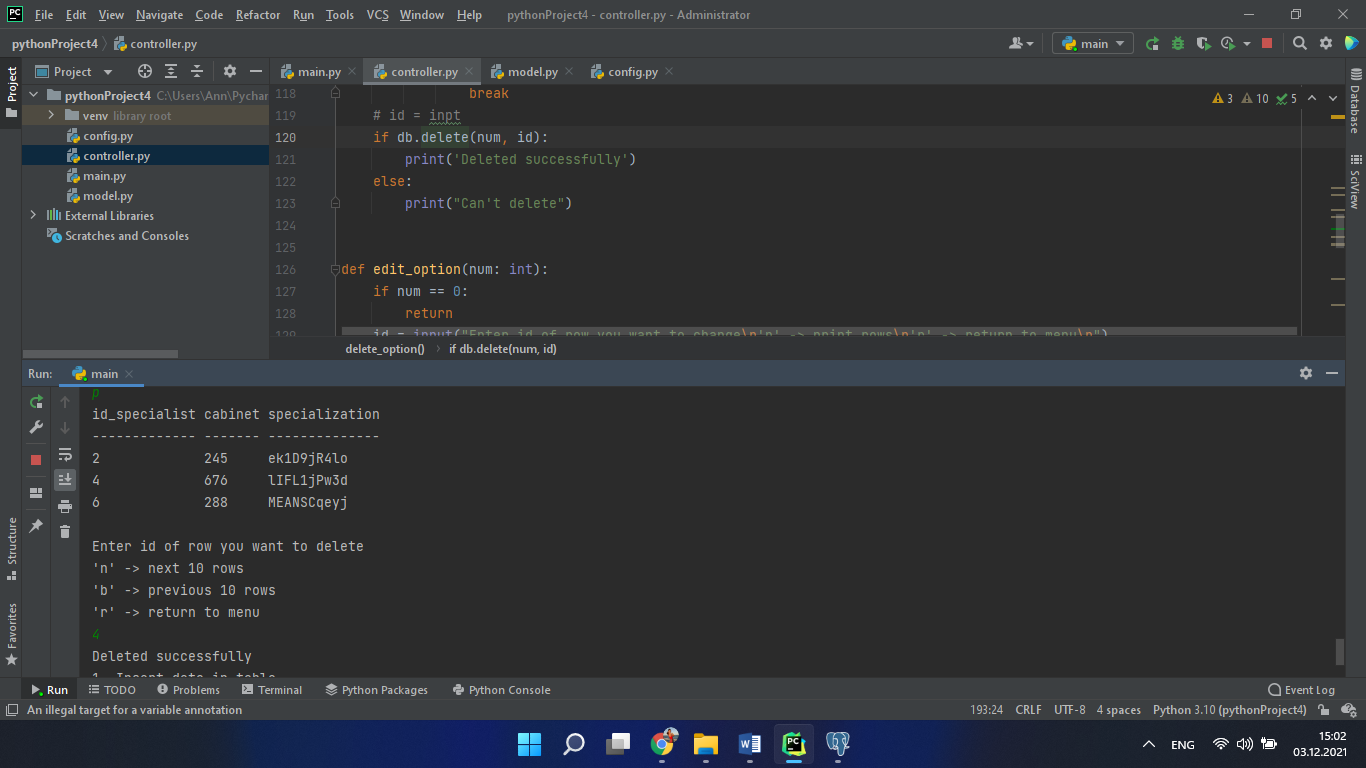
## 

Для перевірки роботи розглянемо запити на видалення даних з батьківської таблиці Specialist і дочірньої таблиці Direction.

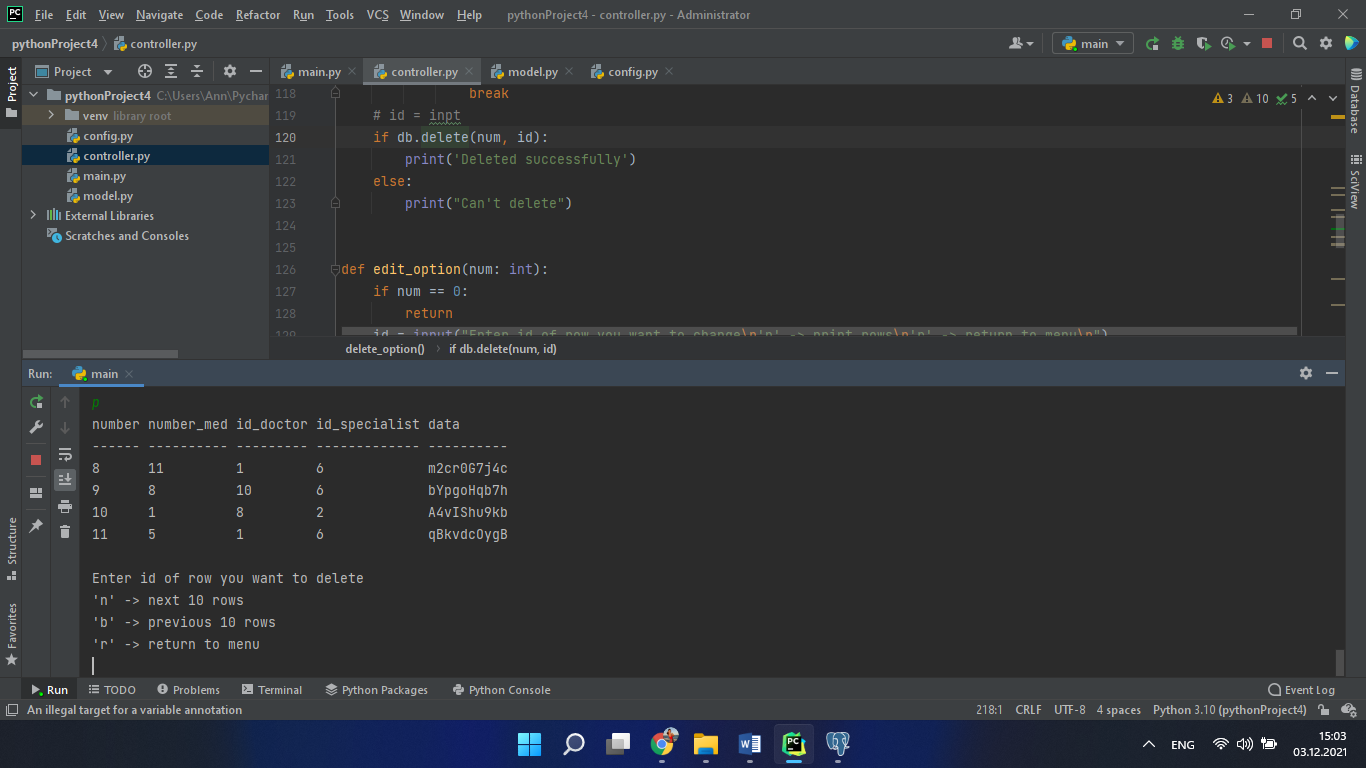
Таблиця Direction до видалення:



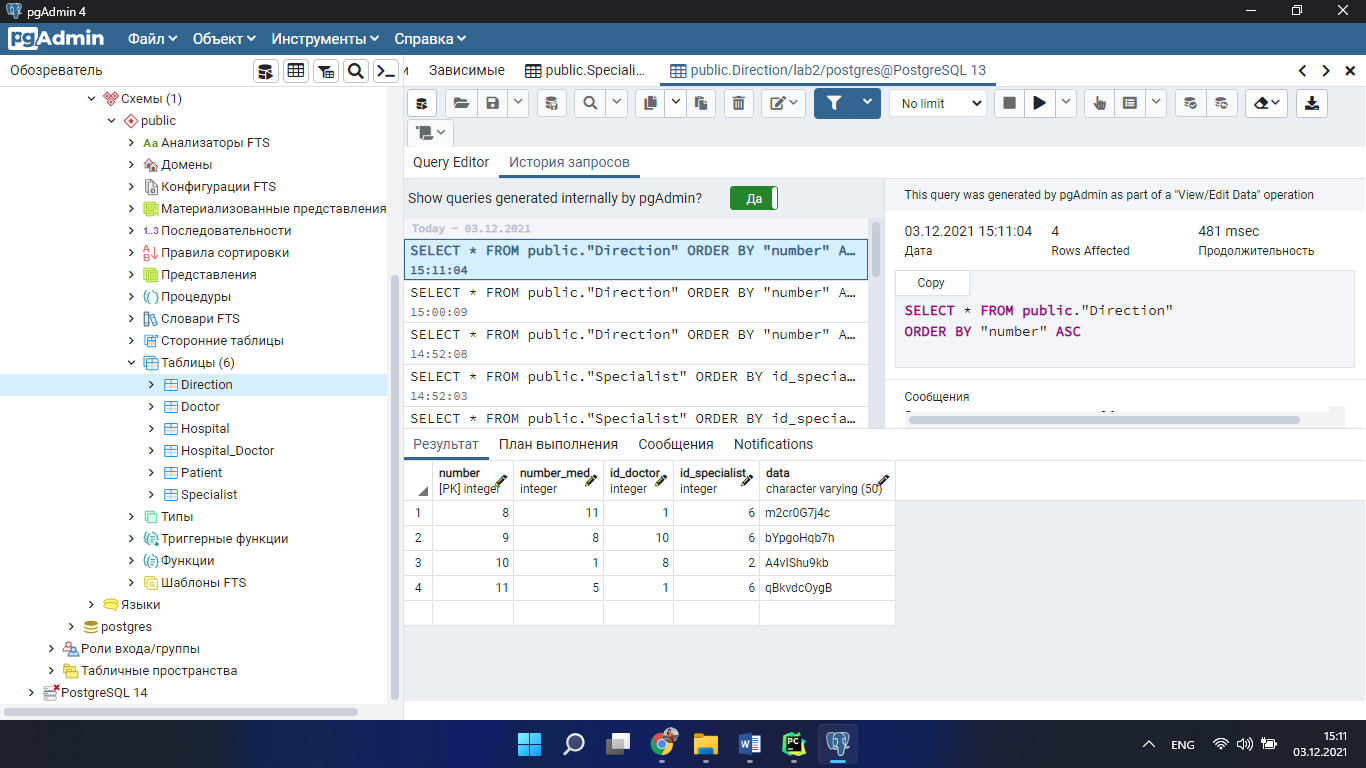
Таблиця Specialist до видалення:



Таблиця Direction після видалення:



Таблиця Direction в PgAdmin4 після видалення поля з id 4

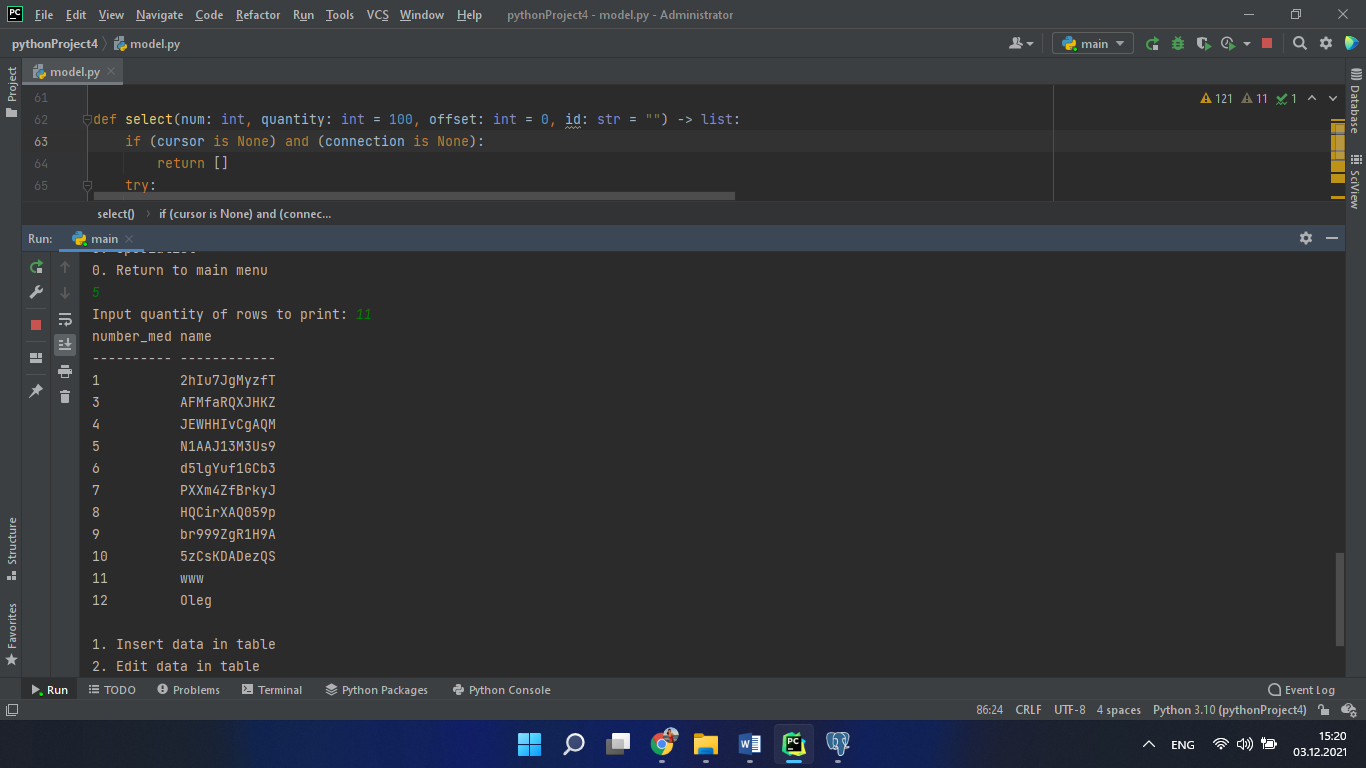


## Запит на вставку поля

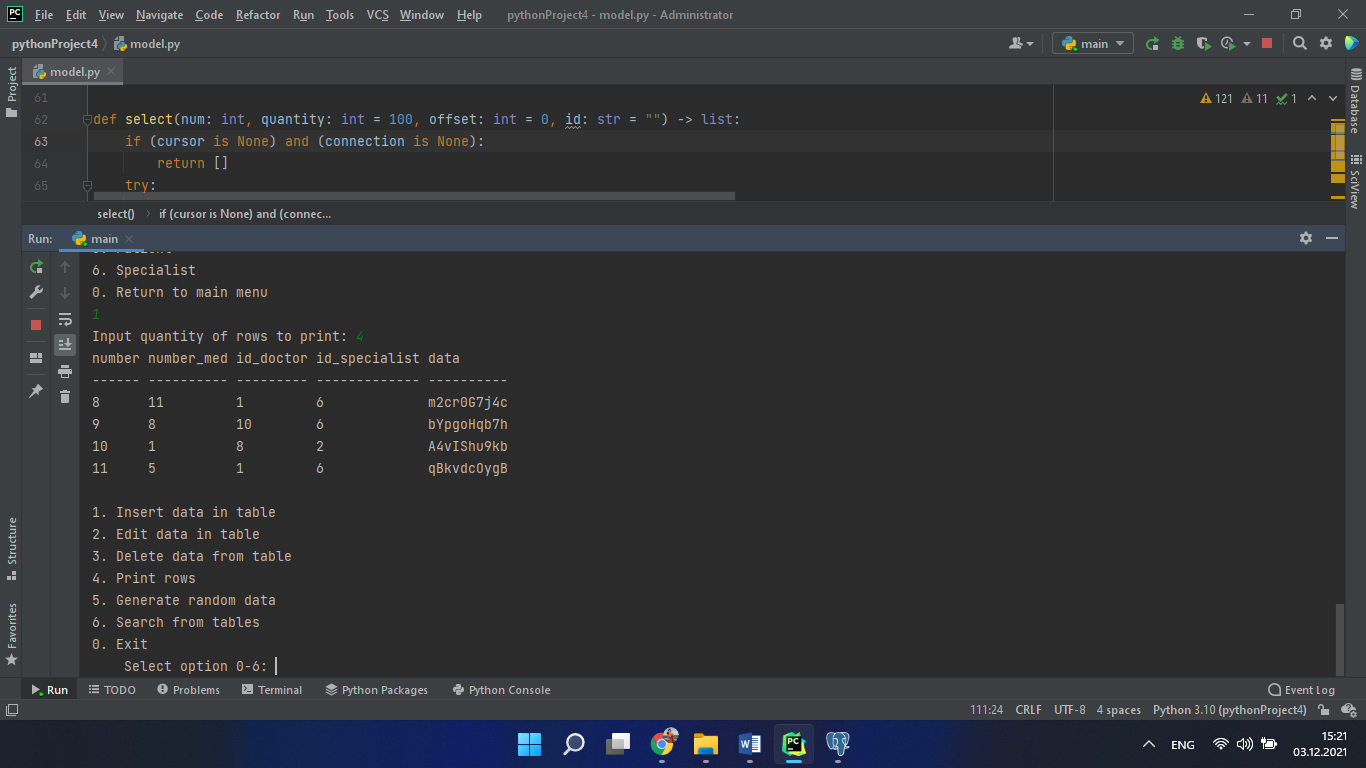
def insert(num: int, col: list) -> bool:  
 if (cursor is None) and (connection is None):  
 return False  
 try:  
 match num:  
 case 1:  
 cursor.execute("""INSERT INTO PUBLIC."Direction" (number\_med, id\_doctor, id\_specialist, data) \  
 VALUES (%s, %s, %s, %s)""", col)  
 case 2:  
 cursor.execute("""INSERT INTO PUBLIC."Doctor" (id\_specialist, name\_doc, phone\_num) \  
 VALUES (%s, %s, %s)""", col)  
 case 3:  
 cursor.execute("""INSERT INTO PUBLIC."Hospital" (name, address, phone) \  
 VALUES (%s, %s, %s)""", col)  
 case 4:  
 cursor.execute("""INSERT INTO PUBLIC."Hospital\_Doctor" (id\_doctor, id) \  
 VALUES (%s, %s)""", col)  
 case 5:  
 cursor.execute("""INSERT INTO PUBLIC."Patient" (name) \  
 VALUES (%s)""", col)  
 case 6:  
 cursor.execute("""INSERT INTO PUBLIC."Specialist" (cabinet, specialization) \  
 VALUES (%s, %s)""", col)  
 connection.commit()  
 except (Exception, psycopg2.Error) as error:  
 print("Can't insert into table", error)  
 cursor.execute('rollback')  
 return False  
 return True

Для перевірки роботи розглянемо запити на вставки в таблицю Patient. Спочатку коректний, потім з неіснуючим значенням зовнішнього ключа таблиці Direction.

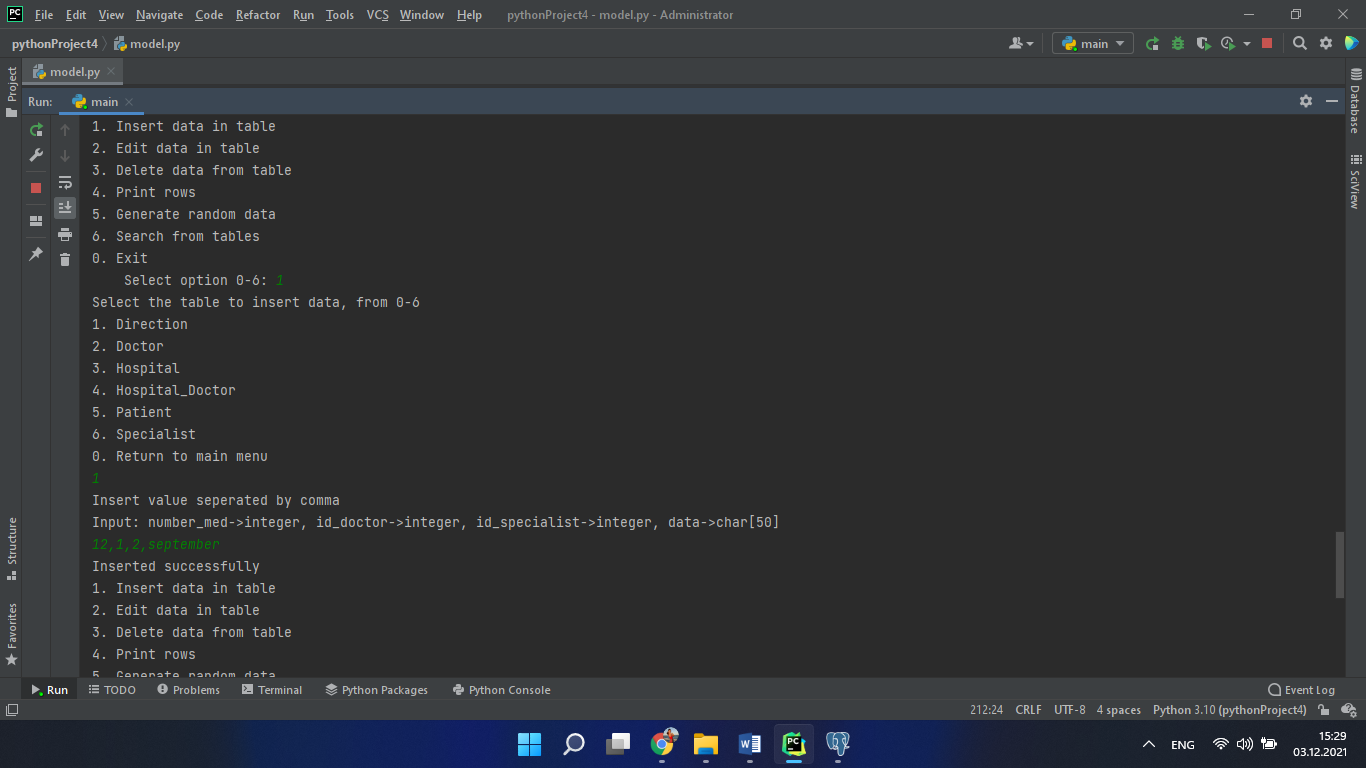
Таблиця Patient до вставки:



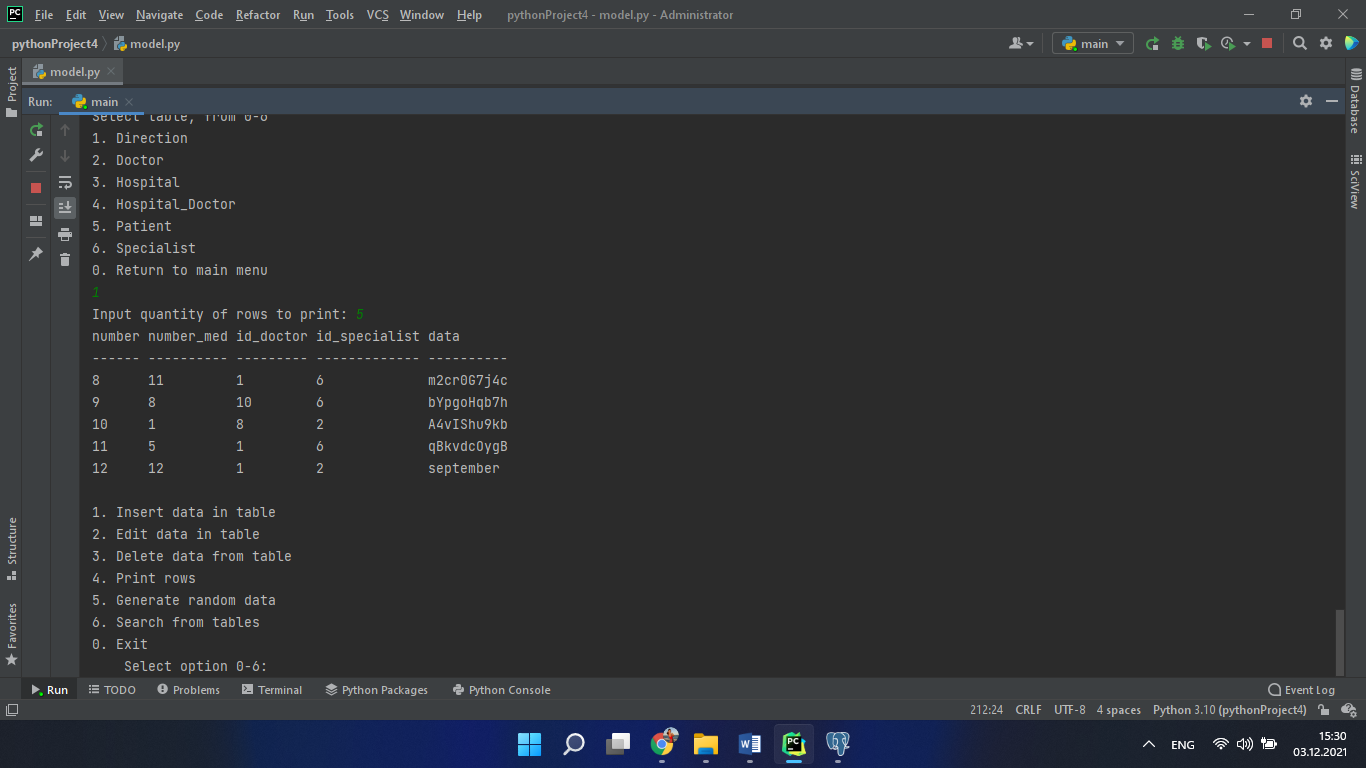
Таблиця Direction до вставки:



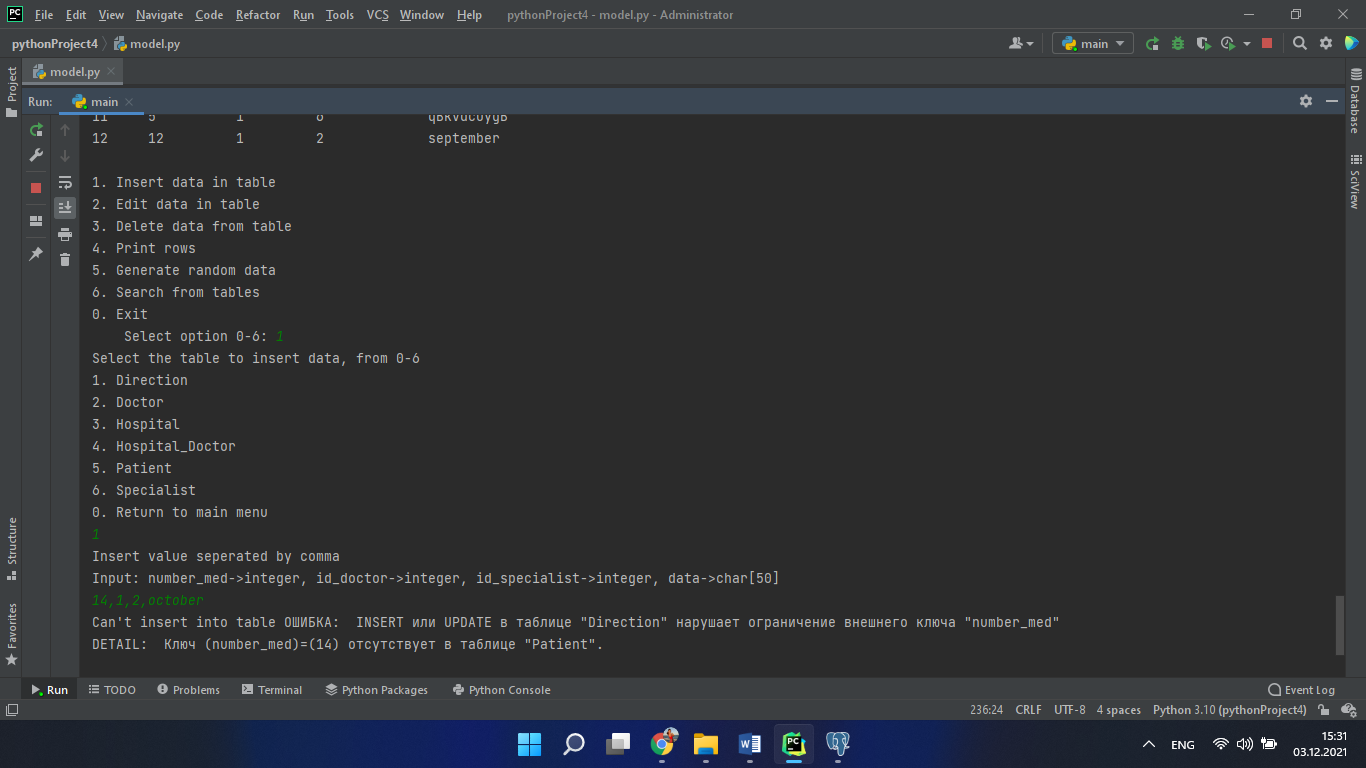
Виконаємо вставку



Таблиця Direction після вставки:



Як бачимо помилок не виникло, тепер спробуємо додати з некоректним зовнішнім ключем.



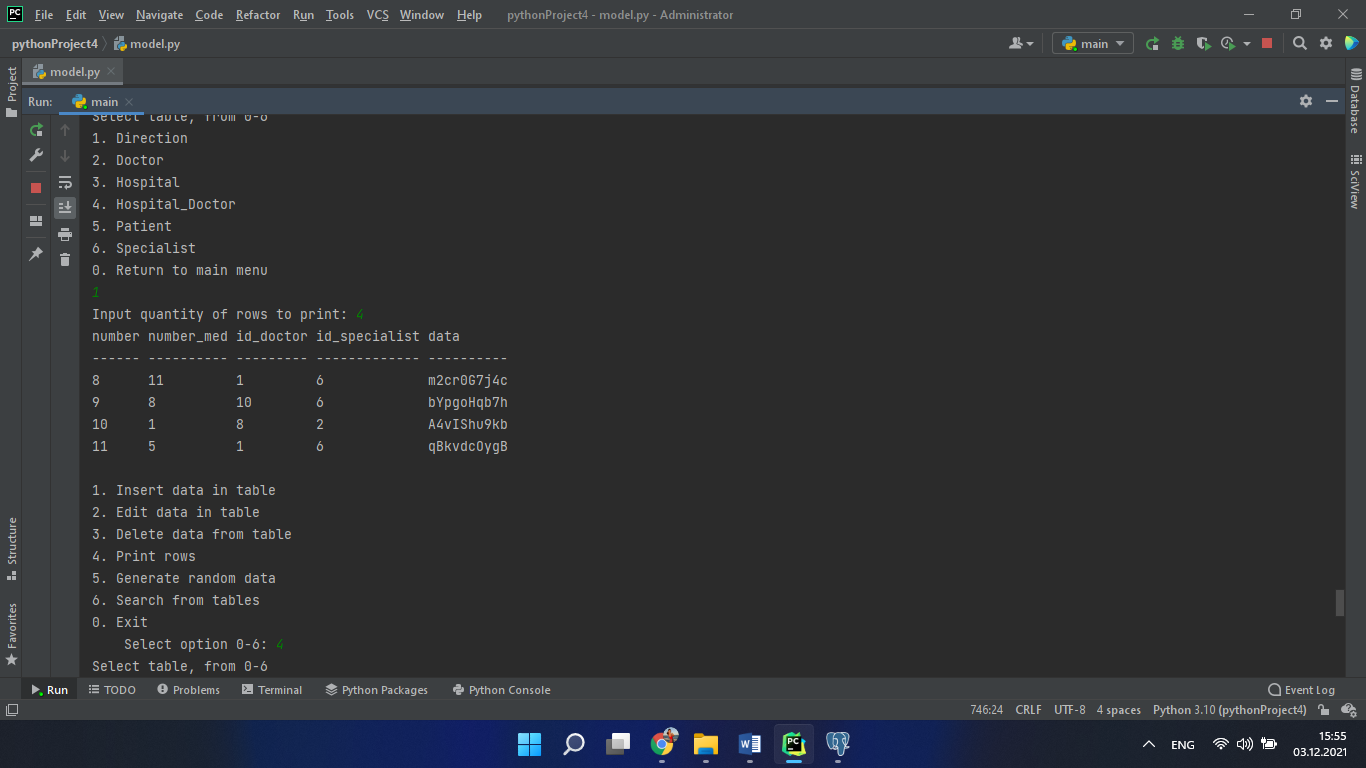
Можемо побачити помилку.

## Запит на зміну полів

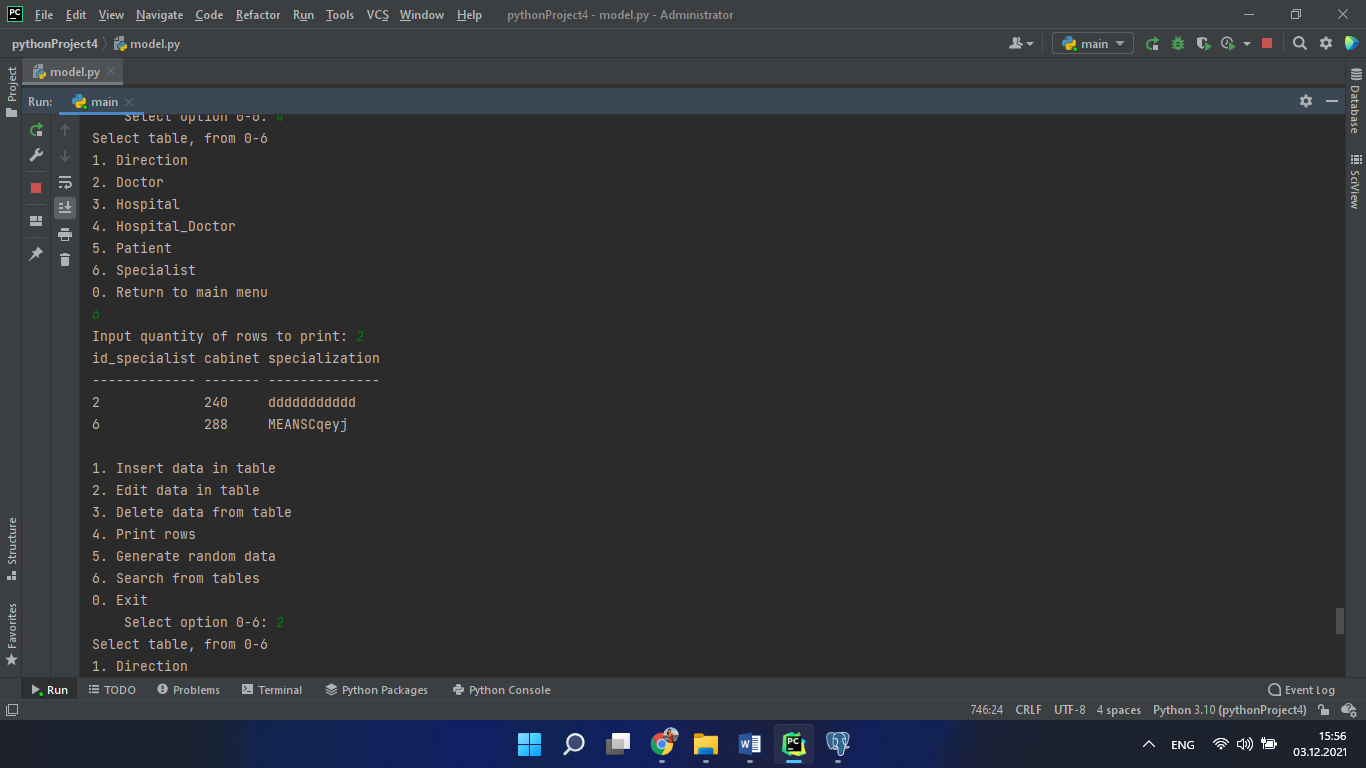
def update(num: int, col: list, id: int) -> bool:  
 if (cursor is None) and (connection is None):  
 return False  
 try:  
 match num:  
 case 1:  
 cursor.execute(  
 """UPDATE PUBLIC."Direction" SET id\_doctor = %s, id\_specialist = %s, number\_med = %s, data = %s WHERE number=%s;""",  
 (\*col, id,))  
 case 2:  
 cursor.execute(  
 """UPDATE PUBLIC."Doctor" SET id\_specialist = %s, name\_doc = %s, phone\_num = %s WHERE id\_doctor = %s;""",  
 (\*col, id,))  
 case 3:  
 cursor.execute(  
 """UPDATE PUBLIC."Hospital" SET name = %s, address = %s, phone = %s WHERE id = %s;""",  
 (\*col, id,))  
 case 4:  
 cursor.execute(  
 """UPDATE PUBLIC."Hospital\_Doctor" SET id\_doctor = %s, id = %s WHERE id\_tab = %s;""",  
 (\*col, id,))  
 case 5:  
 cursor.execute(  
 """UPDATE PUBLIC."Patient" SET name = %s WHERE number\_med = %s;""",  
 (\*col, id,))  
 case 6:  
 cursor.execute(  
 """UPDATE PUBLIC."Specialist" SET cabinet = %s, specialization = %s WHERE id\_specialist = %s;""",  
 (\*col, id,))  
 connection.commit()  
 except (Exception, psycopg2.Error) as error:  
 print(f"Can't update the table row = {id}, {col}\n", error)  
 cursor.execute('rollback')  
 return False  
 return True

Для перевірки роботи розглянемо запити на зміну значенння в таблиці Specialist. Спочатку коректний, потім з неіснуючим значенням зовнішнього ключа таблиці Direction.

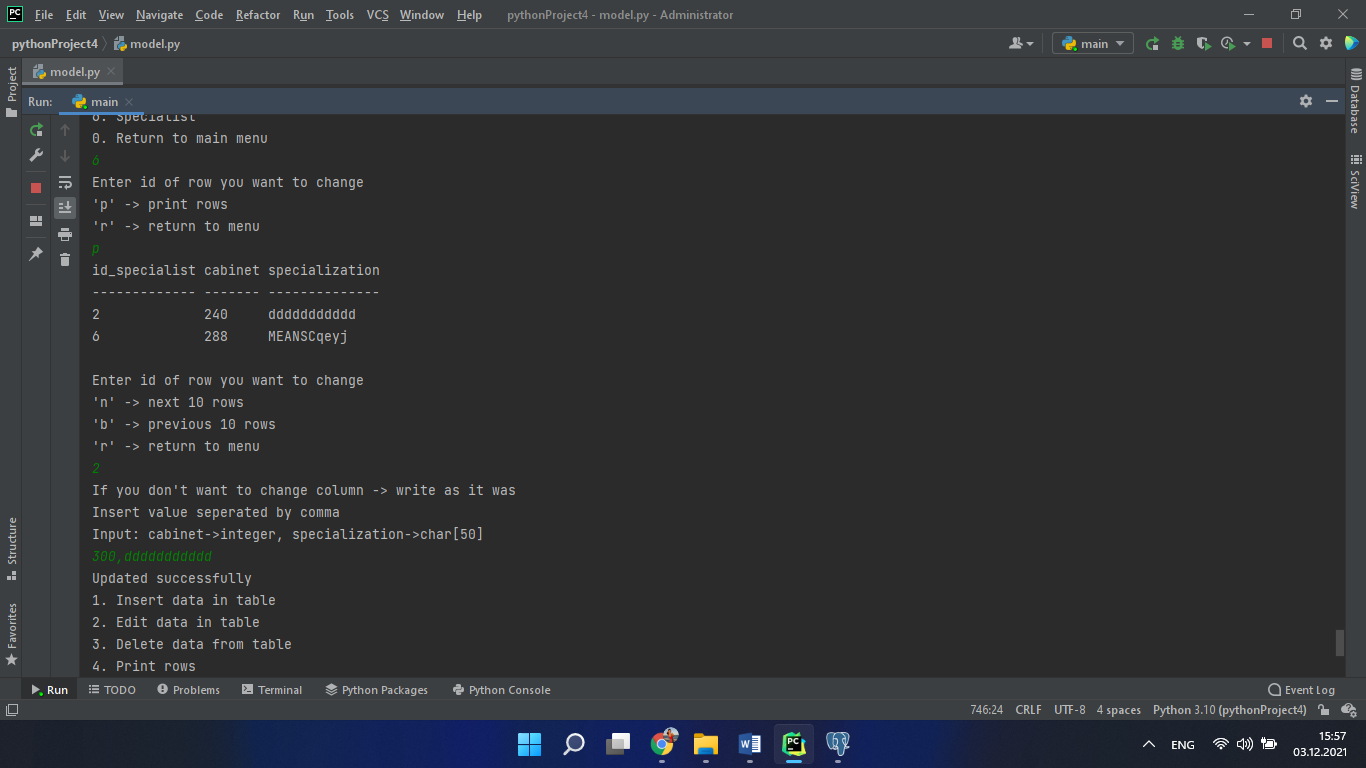
Таблиця Direction до зміни:



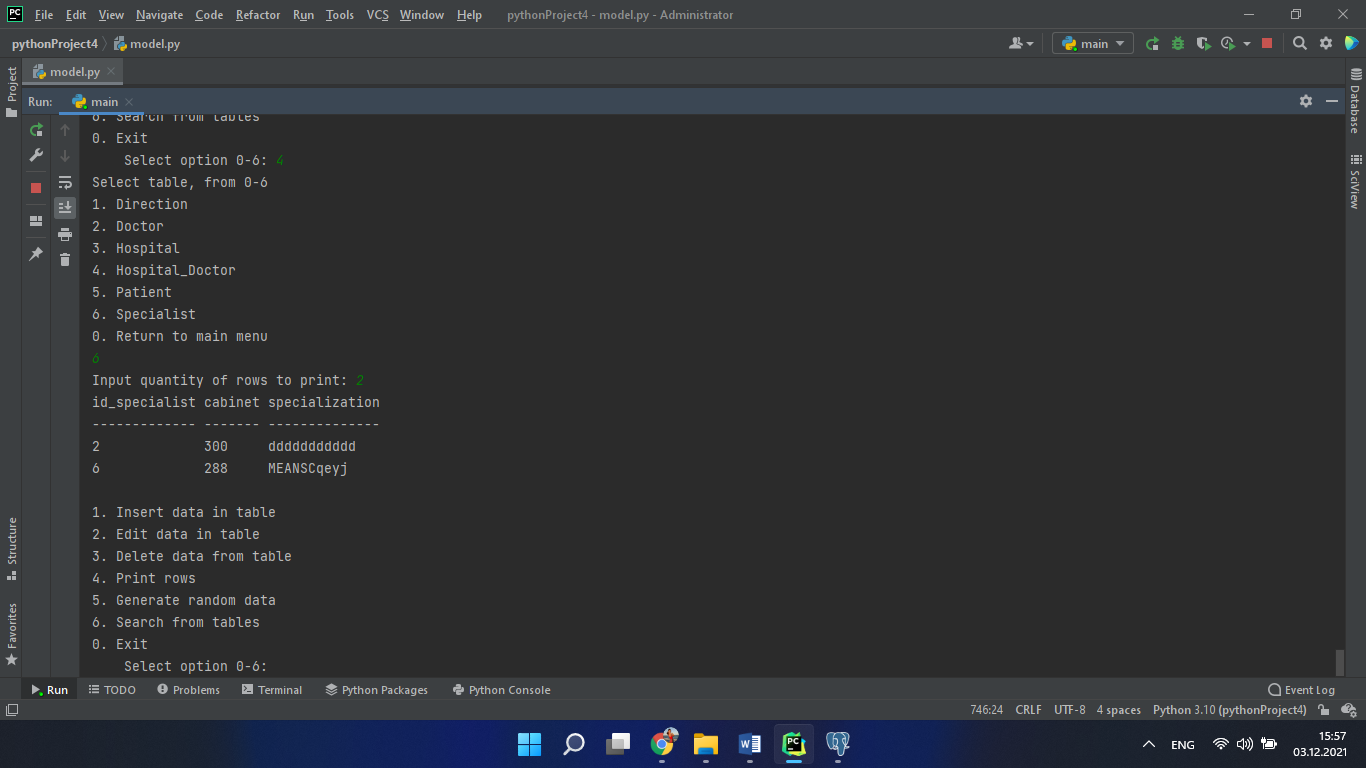
Таблиця Specialist до зміни:



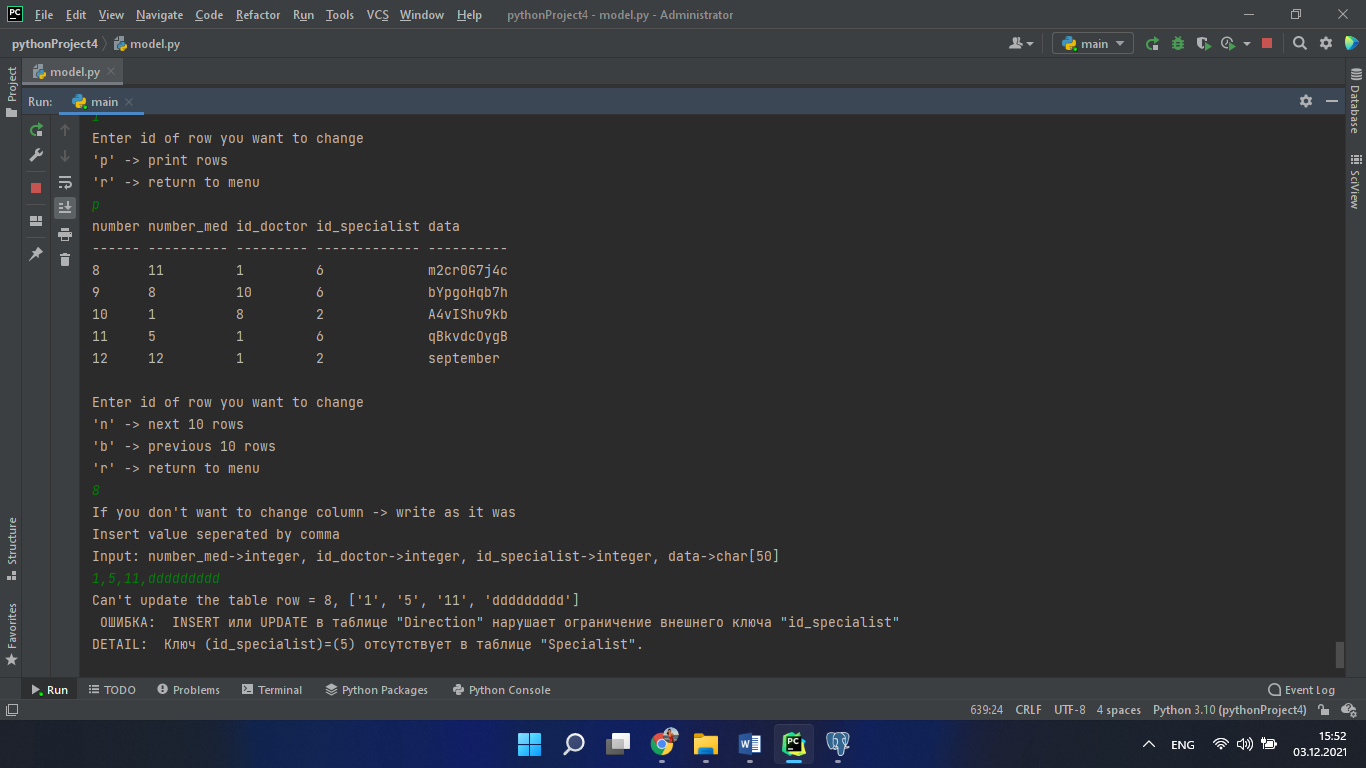
Змінимо номер кабінету.



Таблиця Specialist після зміни



Спробуємо змінити в таблиці Specialist.



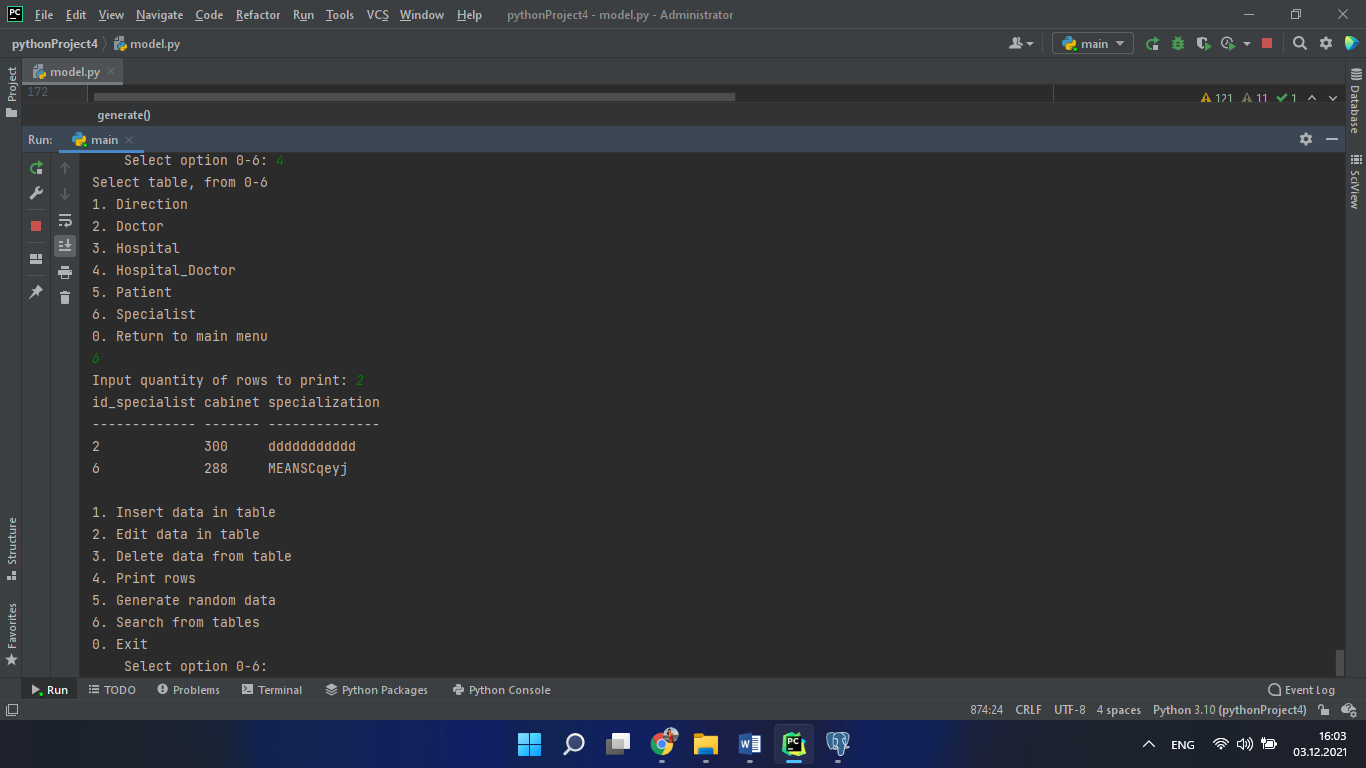
Можемо бачити помилку, про відсутність ключа == 5.

# Завдання 2

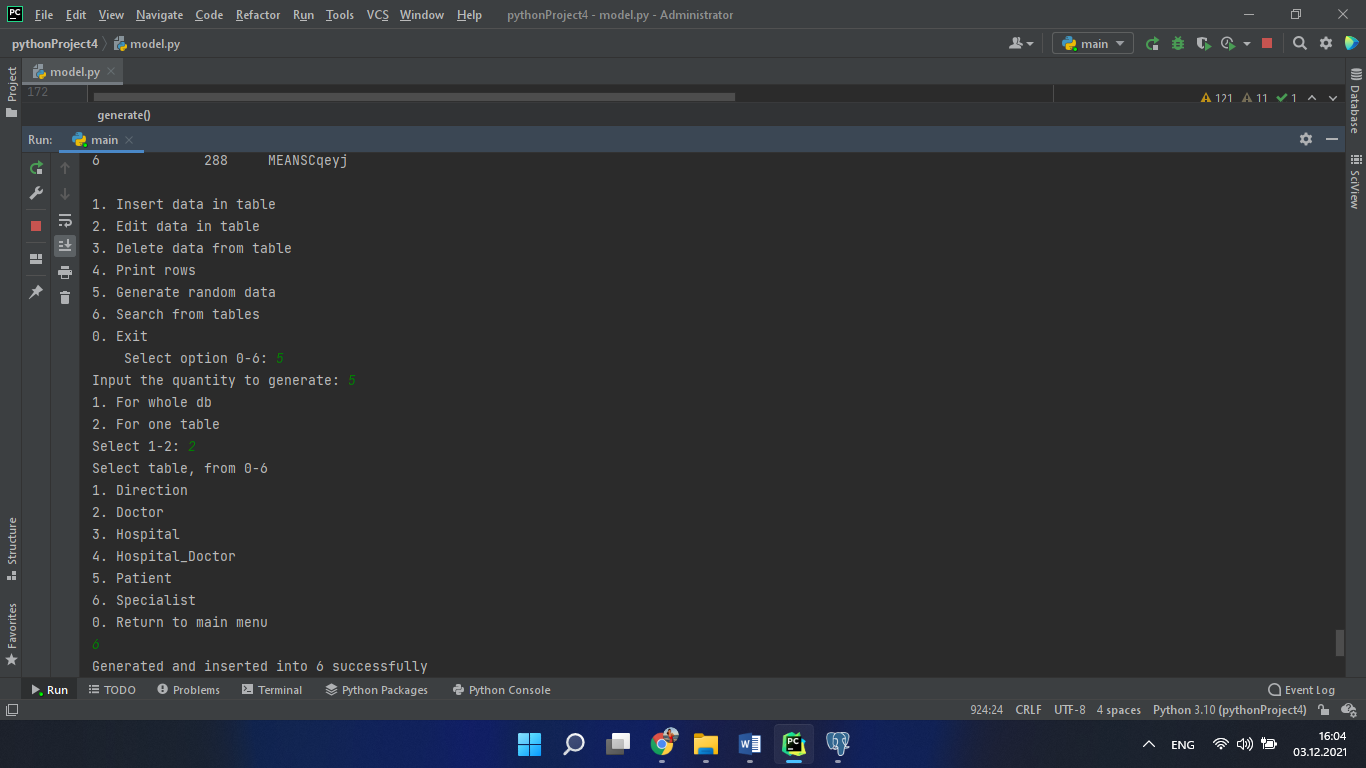
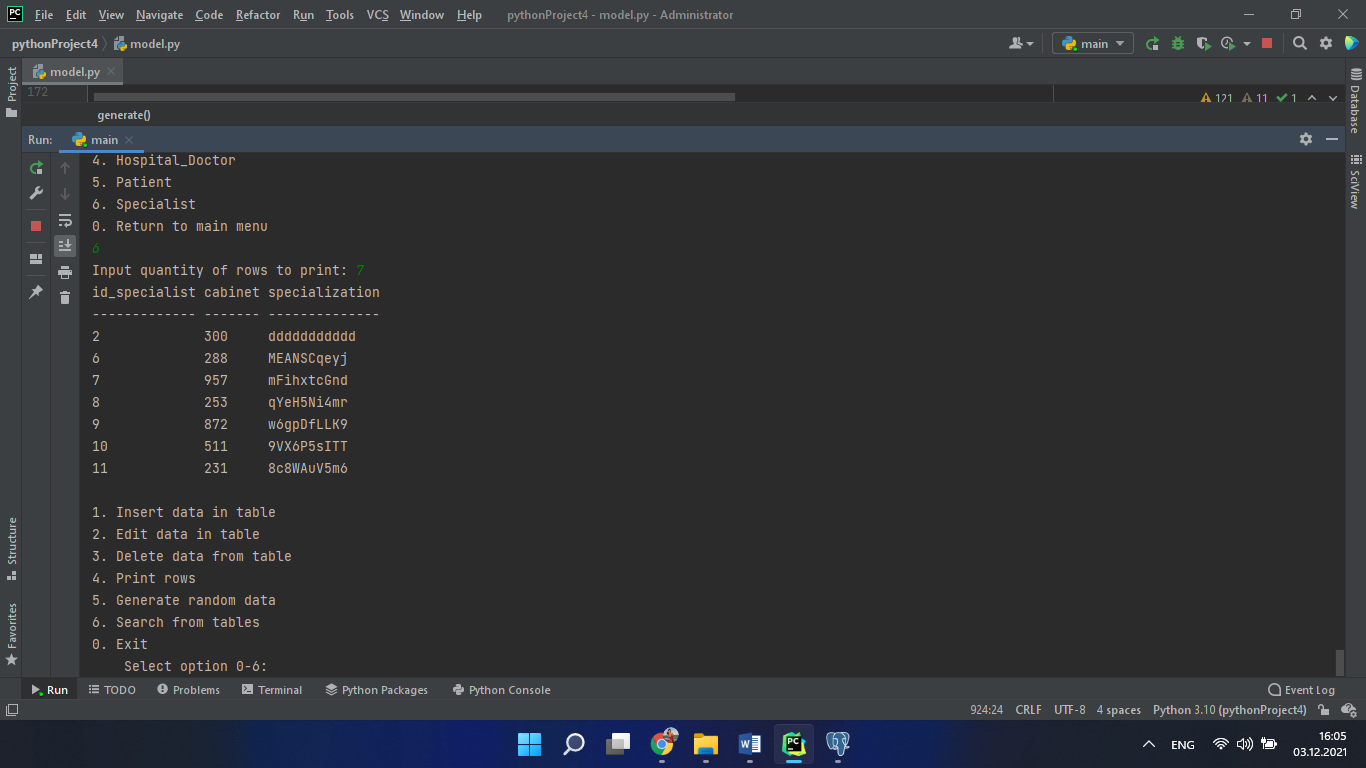
def generate(num: int, quant: int):  
 if (cursor is None) and (connection is None):  
 return False  
 try:  
 for i in range(quant):  
 match num:  
 case 1:  
 cursor.execute(  
 """INSERT INTO public."Direction"(data, number\_med, id\_doctor, id\_specialist) Select random\_string(10), number\_med, id\_doctor, public."Specialist".id\_specialist From public."Doctor" cross join public."Specialist" cross join public."Patient" order by random() limit 1;""")  
 case 2:  
 cursor.execute(  
 """INSERT INTO public."Doctor"(name\_doc, phone\_num, id\_specialist) Select random\_string(8), random\_between(1000,100000), id\_specialist From public."Specialist" order by random() limit 1; """)  
 case 3:  
 cursor.execute(  
 """INSERT INTO public."Hospital"(name, address, phone) VALUES (random\_string(10), random\_string(10), random\_between(1000,100000));""")  
 case 4:  
 cursor.execute(  
 """INSERT INTO public."Hospital\_Doctor"(id\_doctor, id) Select id\_doctor, id From public."Hospital" cross join public."Doctor" order by random() limit 1;""")  
 case 5:  
 cursor.execute(  
 """INSERT INTO public."Patient"(name) Select random\_string(12);""")  
 case 6:  
 cursor.execute(  
 """INSERT INTO public."Specialist"(cabinet, specialization) Select random\_between(10,1000), random\_string(10);""")  
 connection.commit()  
 except (Exception, psycopg2.Error) as error:  
 print(f"Can't insert in the table row = {num}\n", error)  
 cursor.execute('rollback')  
 return False  
 return True

Вставка 5 псевдорандомізованих записів у кожну з таблиць.

Таблиця Specialist перед вставкою записів:



Таблиці після вставки згенерованих даних

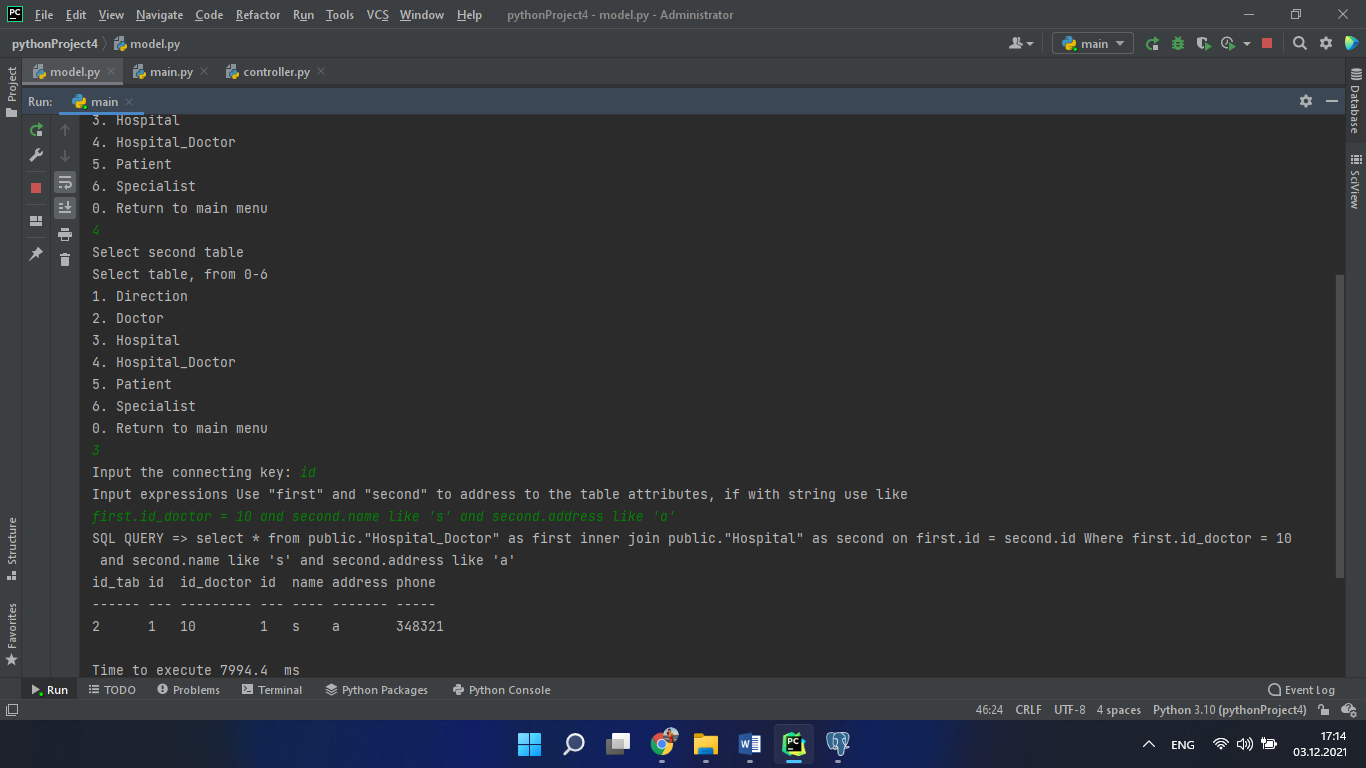
 

# Завдання 3

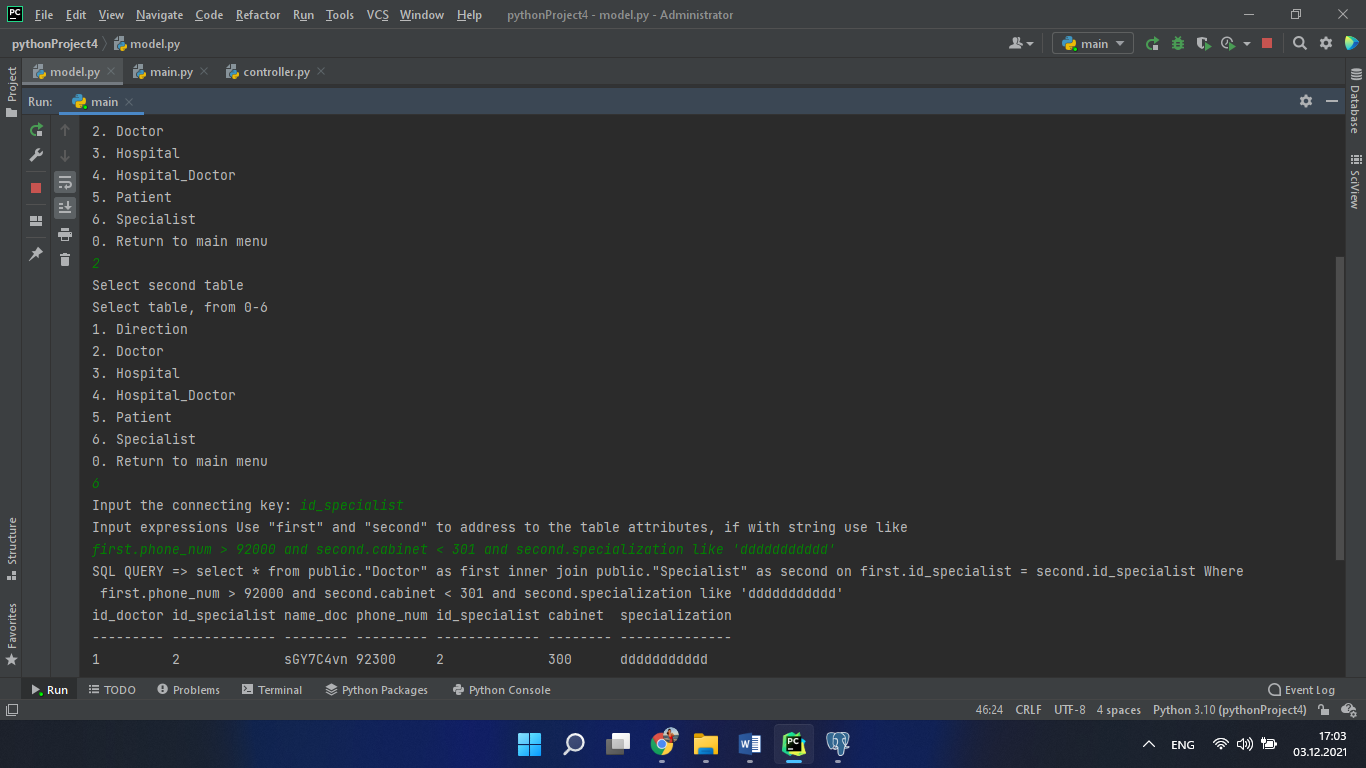
def search(tables: list[int], key: str, expression: str):  
 sql\_query = "select \* from "  
 match tables[0]:  
 case 1:  
 sql\_query += 'public."Direction" as first'  
 case 2:  
 sql\_query += 'public."Doctor" as first'  
 case 3:  
 sql\_query += 'public."Hospital" as first'  
 case 4:  
 sql\_query += 'public."Hospital\_Doctor" as first'  
 case 5:  
 sql\_query += 'public."Patient" as first'  
 case 6:  
 sql\_query += 'public."Specialist" as first'  
  
 sql\_query += ' inner join '  
 match tables[1]:  
 case 1:  
 sql\_query += 'public."Direction" as second'  
 case 2:  
 sql\_query += 'public."Doctor" as second'  
 case 3:  
 sql\_query += 'public."Hospital" as second'  
 case 4:  
 sql\_query += 'public."Hospital\_Doctor" as second'  
 case 5:  
 sql\_query += 'public."Patient" as second'  
 case 6:  
 sql\_query += 'public."Specialist" as second'  
 sql\_query += f' on first.{key} = second.{key} Where {expression}'  
 print('SQL QUERY =>', sql\_query)  
 global cursor  
 try:  
 timer = time.time\_ns()  
 cursor.execute(sql\_query)  
 except (Exception, psycopg2.Error) as error:  
 print("Can't execute search\n", error)  
 return []  
 rows = cursor.fetchall()  
 timer = time.time\_ns() - timer  
 return rows, timer

Об’єднання двох таблиць і пошук в них за 3 атрибутами:

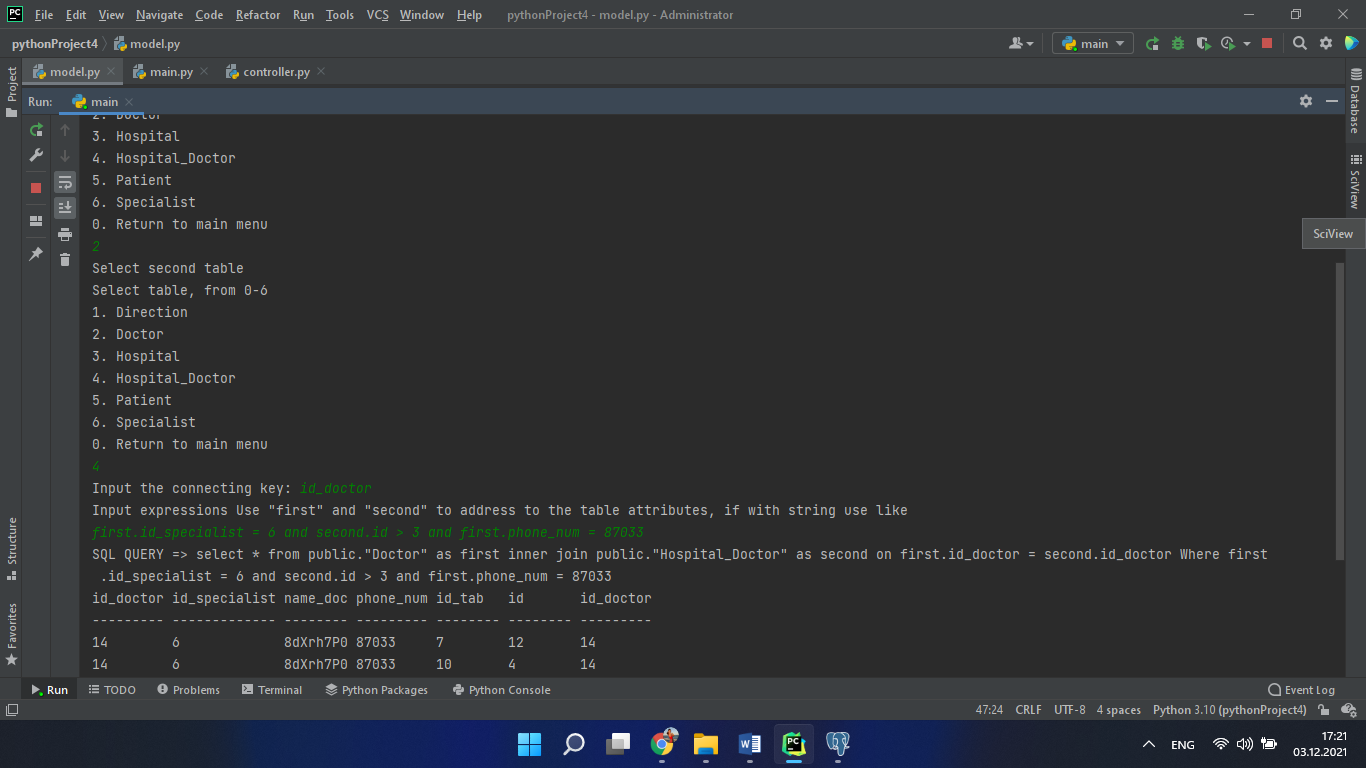
Перший пошук



Другий пошук



Третій пошук



# Завдання 4

Код програмного модулю “model.py”

import time  
  
from config import \*  
import psycopg2  
  
cursor = None  
connection = None  
  
  
def connect():  
 try:  
 global connection, cursor  
 connection = psycopg2.connect(user=user, password=password, host='localhost', port="5432", database=database)  
  
 cursor = connection.cursor()  
  
 except (Exception, psycopg2.Error) as error:  
 print("Can't connect to DB", error)  
 disconnect()  
  
  
def disconnect():  
 if connection:  
 cursor.close()  
 connection.close()  
 print("Successfully disconnected from DB")  
 else:  
 print("Can't disconnect")  
  
  
def insert(num: int, col: list) -> bool:  
 if (cursor is None) and (connection is None):  
 return False  
 try:  
 match num:  
 case 1:  
 cursor.execute("""INSERT INTO PUBLIC."Direction" (number\_med, id\_doctor, id\_specialist, data) \  
 VALUES (%s, %s, %s, %s)""", col)  
 case 2:  
 cursor.execute("""INSERT INTO PUBLIC."Doctor" (id\_specialist, name\_doc, phone\_num) \  
 VALUES (%s, %s, %s)""", col)  
 case 3:  
 cursor.execute("""INSERT INTO PUBLIC."Hospital" (name, address, phone) \  
 VALUES (%s, %s, %s)""", col)  
 case 4:  
 cursor.execute("""INSERT INTO PUBLIC."Hospital\_Doctor" (id\_doctor, id) \  
 VALUES (%s, %s)""", col)  
 case 5:  
 cursor.execute("""INSERT INTO PUBLIC."Patient" (name) \  
 VALUES (%s)""", col)  
 case 6:  
 cursor.execute("""INSERT INTO PUBLIC."Specialist" (cabinet, specialization) \  
 VALUES (%s, %s)""", col)  
 connection.commit()  
 except (Exception, psycopg2.Error) as error:  
 print("Can't insert into table", error)  
 cursor.execute('rollback')  
 return False  
 return True  
  
  
def select(num: int, quantity: int = 100, offset: int = 0, id: str = "") -> list:  
 if (cursor is None) and (connection is None):  
 return []  
 try:  
 match num:  
 case 1:  
 if id:  
 cursor.execute("""SELECT \* FROM public."Direction" WHERE number=%s""", id)  
 else:  
 cursor.execute("""SELECT \* FROM public."Direction" ORDER BY number ASC limit %s offset %s""",  
 (quantity, offset,))  
 case 2:  
 if id:  
 cursor.execute("""SELECT \* FROM public."Doctor" WHERE id\_doctor=%s""", id)  
 else:  
 cursor.execute(  
 """SELECT \* FROM public."Doctor" ORDER BY id\_doctor ASC limit %s offset %s""",  
 (quantity, offset,))  
 case 3:  
 if id:  
 cursor.execute("""SELECT \* FROM public."Hospital" WHERE id=%s""", id)  
 else:  
 cursor.execute("""SELECT \* FROM public."Hospital" ORDER BY id ASC limit %s offset %s""",  
 (quantity, offset,))  
 case 4:  
 if id:  
 cursor.execute("""SELECT \* FROM public."Hospital\_Doctor" WHERE id\_tab=%s""", id)  
 else:  
 cursor.execute(  
 """SELECT \* FROM public."Hospital\_Doctor" ORDER BY id\_tab ASC limit %s offset %s""",  
 (quantity, offset,))  
 case 5:  
 if id:  
 cursor.execute("""SELECT \* FROM public."Patient" WHERE number\_med=%s""", id)  
 else:  
 cursor.execute("""SELECT \* FROM public."Patient" ORDER BY number\_med ASC limit %s offset %s""",  
 (quantity, offset,))  
 case 6:  
 if id:  
 cursor.execute("""SELECT \* FROM public."Specialist" WHERE id\_specialist=%s""", id)  
 else:  
 cursor.execute("""SELECT \* FROM public."Specialist" ORDER BY id\_specialist ASC limit %s offset %s""",  
 (quantity, offset,))  
 return cursor.fetchall()  
 except(Exception, psycopg2.Error) as error:  
 print(f"Can't select from a table {num}, with {id=}\n", error)  
 return []  
  
  
def delete(num: int, id: str) -> bool:  
 if (cursor is None) and (connection is None):  
 return False  
 try:  
 match num:  
 case 1:  
 cursor.execute("""DELETE FROM public."Direction" WHERE number=%s""", (id,))  
 case 2:  
 cursor.execute("""DELETE FROM public."Doctor" WHERE id\_doctor like %s""", (id,))  
 case 3:  
 cursor.execute("""DELETE FROM public."Hospital" WHERE id like %s""", (id,))  
 case 4:  
 cursor.execute("""DELETE FROM public."Hospital\_Doctor" WHERE id\_tab = %s""", (id,))  
 case 5:  
 cursor.execute("""DELETE FROM public."Patient" WHERE number\_med = %s""", (id,))  
 case 6:  
 cursor.execute("""DELETE FROM public."Specialist" WHERE id\_specialist = %s""", (id,))  
 connection.commit()  
 return True  
 except(Exception, psycopg2.Error) as error:  
 print(f"Can't delete from a table {num}, {id=}\n", error)  
 cursor.execute('rollback')  
 return False  
  
  
def update(num: int, col: list, id: int) -> bool:  
 if (cursor is None) and (connection is None):  
 return False  
 try:  
 match num:  
 case 1:  
 cursor.execute(  
 """UPDATE PUBLIC."Direction" SET id\_doctor = %s, id\_specialist = %s, number\_med = %s, data = %s WHERE number=%s;""",  
 (\*col, id,))  
 case 2:  
 cursor.execute(  
 """UPDATE PUBLIC."Doctor" SET id\_specialist = %s, name\_doc = %s, phone\_num = %s WHERE id\_doctor = %s;""",  
 (\*col, id,))  
 case 3:  
 cursor.execute(  
 """UPDATE PUBLIC."Hospital" SET name = %s, address = %s, phone = %s WHERE id = %s;""",  
 (\*col, id,))  
 case 4:  
 cursor.execute(  
 """UPDATE PUBLIC."Hospital\_Doctor" SET id\_doctor = %s, id = %s WHERE id\_tab = %s;""",  
 (\*col, id,))  
 case 5:  
 cursor.execute(  
 """UPDATE PUBLIC."Patient" SET name = %s WHERE number\_med = %s;""",  
 (\*col, id,))  
 case 6:  
 cursor.execute(  
 """UPDATE PUBLIC."Specialist" SET cabinet = %s, specialization = %s WHERE id\_specialist = %s;""",  
 (\*col, id,))  
 connection.commit()  
 except (Exception, psycopg2.Error) as error:  
 print(f"Can't update the table row = {id}, {col}\n", error)  
 cursor.execute('rollback')  
 return False  
 return True  
  
  
def generate(num: int, quant: int):  
 if (cursor is None) and (connection is None):  
 return False  
 try:  
 for i in range(quant):  
 match num:  
 case 1:  
 cursor.execute(  
 """INSERT INTO public."Direction"(data, number\_med, id\_doctor, id\_specialist) Select random\_string(10), number\_med, id\_doctor, public."Specialist".id\_specialist From public."Doctor" cross join public."Specialist" cross join public."Patient" order by random() limit 1;""")  
 case 2:  
 cursor.execute(  
 """INSERT INTO public."Doctor"(name\_doc, phone\_num, id\_specialist) Select random\_string(8), random\_between(1000,100000), id\_specialist From public."Specialist" order by random() limit 1; """)  
 case 3:  
 cursor.execute(  
 """INSERT INTO public."Hospital"(name, address, phone) VALUES (random\_string(10), random\_string(10), random\_between(1000,100000));""")  
 case 4:  
 cursor.execute(  
 """INSERT INTO public."Hospital\_Doctor"(id\_doctor, id) Select id\_doctor, id From public."Hospital" cross join public."Doctor" order by random() limit 1;""")  
 case 5:  
 cursor.execute(  
 """INSERT INTO public."Patient"(name) Select random\_string(12);""")  
 case 6:  
 cursor.execute(  
 """INSERT INTO public."Specialist"(cabinet, specialization) Select random\_between(10,1000), random\_string(10);""")  
 connection.commit()  
 except (Exception, psycopg2.Error) as error:  
 print(f"Can't insert in the table row = {num}\n", error)  
 cursor.execute('rollback')  
 return False  
 return True  
5  
  
  
def search(tables: list[int], key: str, expression: str):  
 sql\_query = "select \* from "  
 match tables[0]:  
 case 1:  
 sql\_query += 'public."Direction" as first'  
 case 2:  
 sql\_query += 'public."Doctor" as first'  
 case 3:  
 sql\_query += 'public."Hospital" as first'  
 case 4:  
 sql\_query += 'public."Hospital\_Doctor" as first'  
 case 5:  
 sql\_query += 'public."Patient" as first'  
 case 6:  
 sql\_query += 'public."Specialist" as first'  
  
 sql\_query += ' inner join '  
 match tables[1]:  
 case 1:  
 sql\_query += 'public."Direction" as second'  
 case 2:  
 sql\_query += 'public."Doctor" as second'  
 case 3:  
 sql\_query += 'public."Hospital" as second'  
 case 4:  
 sql\_query += 'public."Hospital\_Doctor" as second'  
 case 5:  
 sql\_query += 'public."Patient" as second'  
 case 6:  
 sql\_query += 'public."Specialist" as second'  
 sql\_query += f' on first.{key} = second.{key} Where {expression}'  
 print('SQL QUERY =>', sql\_query)  
 global cursor  
 try:  
 timer = time.time\_ns()  
 cursor.execute(sql\_query)  
 except (Exception, psycopg2.Error) as error:  
 print("Can't execute search\n", error)  
 return []  
 rows = cursor.fetchall()  
 timer = time.time\_ns() - timer  
 return rows, timer

Даний модуль є точкою доступу до бази даних з програми. Саме в ньому реалізуються всі запити. Для цього в ньому використовується бібліотека – psycopg2.

Функція connect() – відповідає за підключення до БД.

Функція disconnect() – відповідає за відключення від БД.

Функція insert(num, col) –вставка в таблицю num, дані col.

Функція delete(num, id) – видалення з таблиці num, рядок з первинним ключем == id.

Функція update(num, col, id) – заміна в таблиці num, рядка з первинним ключем == id, данними col.

Функцію select(num, quantity, offset, id) – витягує дані з таблиці num та відсортовує їх, а якщо вказано id, то витягує тільки рядок з первинним ключем == id.

Функція generate(num, quant) – генерує дані і вставляє в таблицю num, quant разів ці даніазрр.

Функція search(tables, key, expression) ­–об’єднує дані і витягує дані з таблиць