

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**»**

Виконав студент ІII курсу

ФПМ групи КВ-83

Пащенко Антон КВ-83

Перевірив: Павловський В.І.

Київ – 2020

**Ознайомлення з базовими операціями СУБД PostgreSQL**

*Загальне* *завдання* роботи полягає у наступному:

1. Реалізувати функції 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 2-х 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.

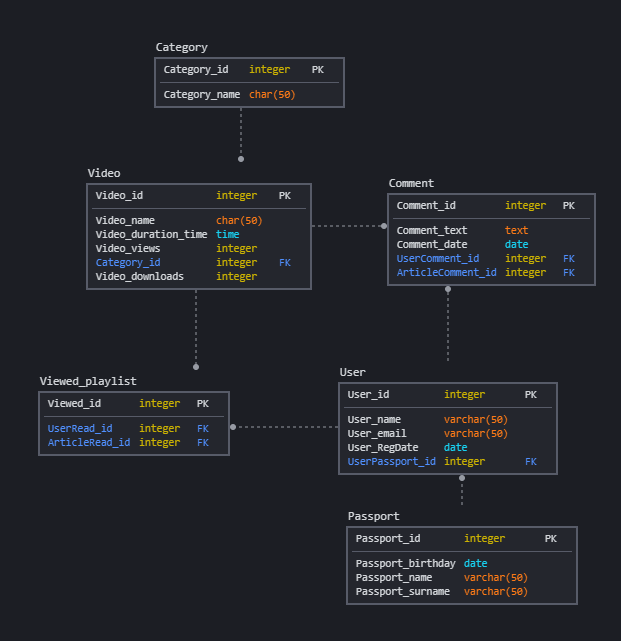
Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

*Вимоги до інтерфейсу користувача*

1. Використовувати консольний інтерфейс користувача.

**Нормалізована логічна модель даних БД «Відеохостінг»**

На Рис. Наведено нормалізовану логічну модель даних БД «Відеохостінг»



**Рис 2.1 Нормалізована логічна модель даних БД «Відеохостінг»**

**Опис програми**

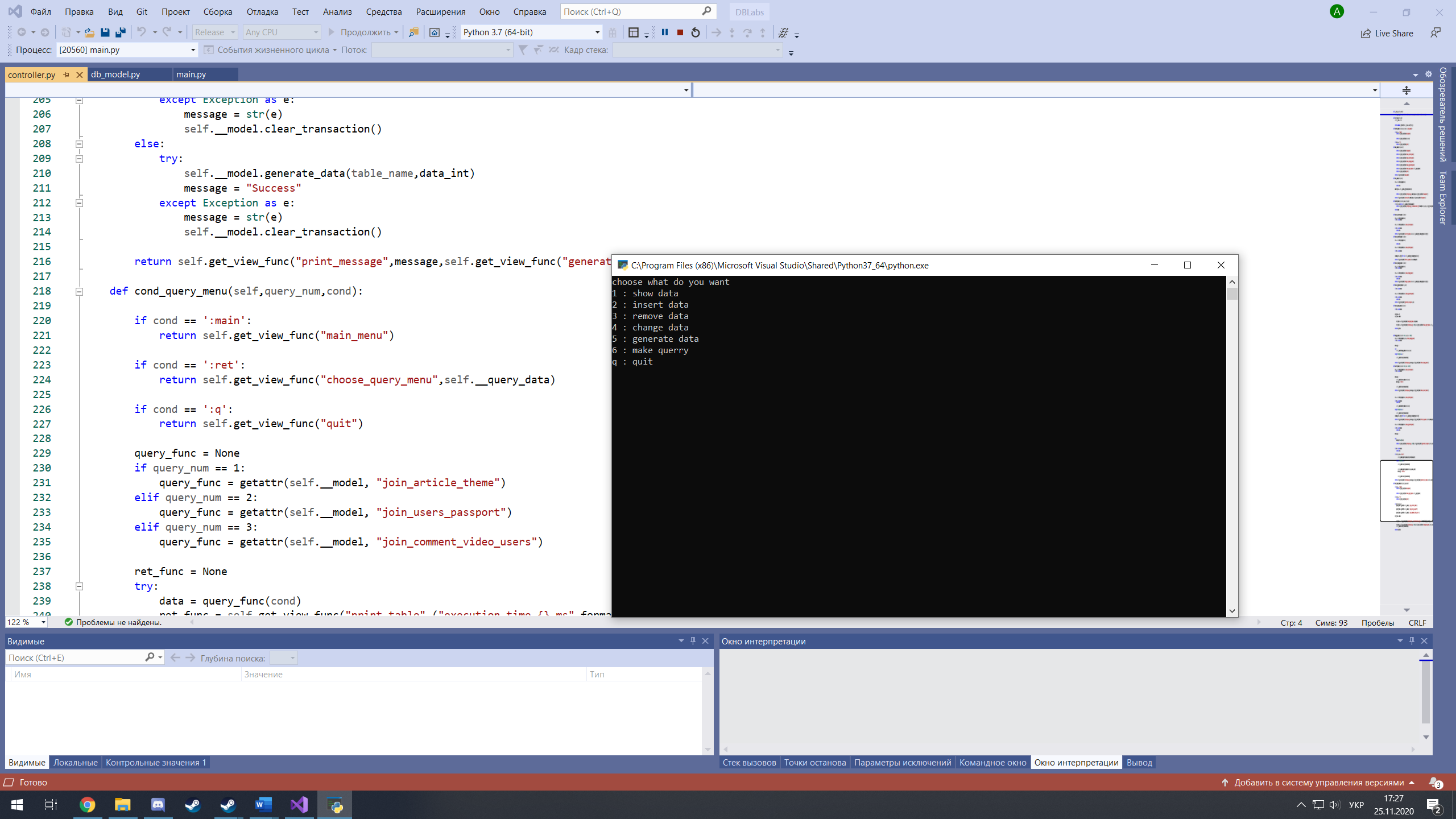
Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL та реалізовує функціональні вимоги, що наведені у завданні. Програма складається з 3 модулів:

1. console\_view – клас що відповідає за відображення даних та обробку вводу користувача.
2. controller.py – клас що обробляє запити від console\_view та викликає методи моделі.
3. db\_model – клас що виконує запити до PostgresSQL та повертає результат.

Використані сторонні бібліотеки psycopg2, prettytable.

**Структура меню програми**

Меню програми



**Рис 2.2** **Меню програми**

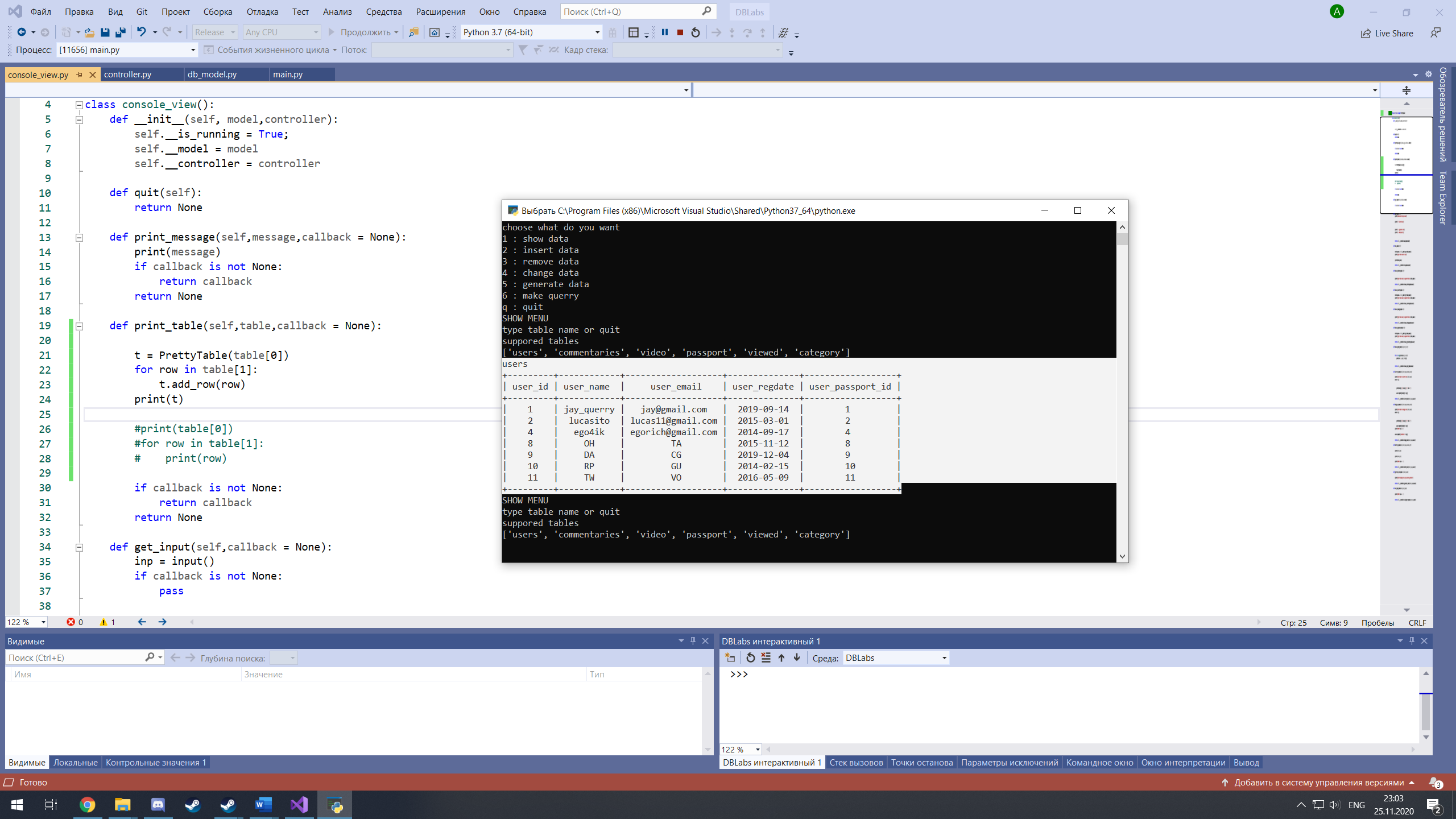
**Функціонал**

Кожне меню запитує таблицю з якою буде проведена якась дія та додаткову інформацію в залежності від дії. При введенні неіснуючої таблиці програма повідомить про помилку та дасть можливість ввести коректні данні.

1. Show data виводить стовпці таблиці.
2. Insert data запрошує данні для конкретної таблиці, при введенні некоректних даних (за типом) програма повідомить про помилку та дасть можливість ввести коректні данні. Після успішного вводу кортеж додається до таблиці.
3. Remove data запрошує умову за якою будуть видалятися дані, при введенні некоректної умови програма повідомить про помилку та дасть можливість ввести коректну умову. Після успішного вводу данні що підходять під умову видаляються з таблиці.
4. Change data запрошує данні для конкретної таблиці на які треба змінити існуючі стовпці. Далі запрошується умова за якою будуть вибиратися кортежі що мають змінитися. При введенні некоректних даних (за типом або за умовою) програма повідомить про помилку та дасть можливість ввести коректні данні.
5. Generate data запрошує кількість кортежів, що треба згенерувати. При успішній генерації виводиться повідомлення.
6. Make query запрошує в якій комбінації таблиць буде проведений пошук video + category, users + passport, comment + video + users. Потім запрошується умова за якою буде проведений пошук. Якщо всі данні коректні виводиться результат інакше виводиться помилка і пропонується ввести нові данні.

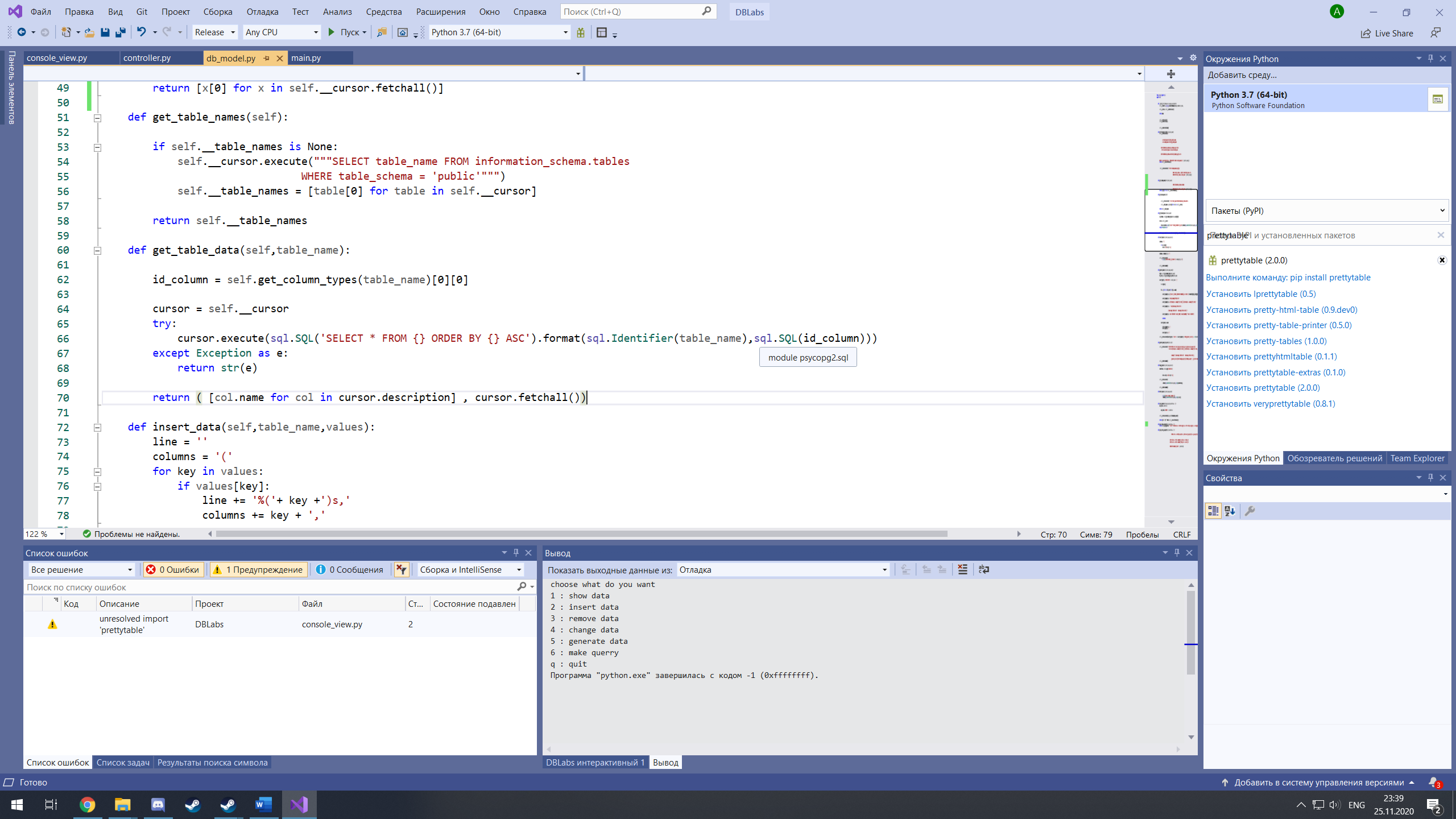
**Вибірка елементів з БД**

Приклад вибірки елементів з таблиці users:



**Рис 2.3 таблиця users**

Метод що повертає вміст таблиці

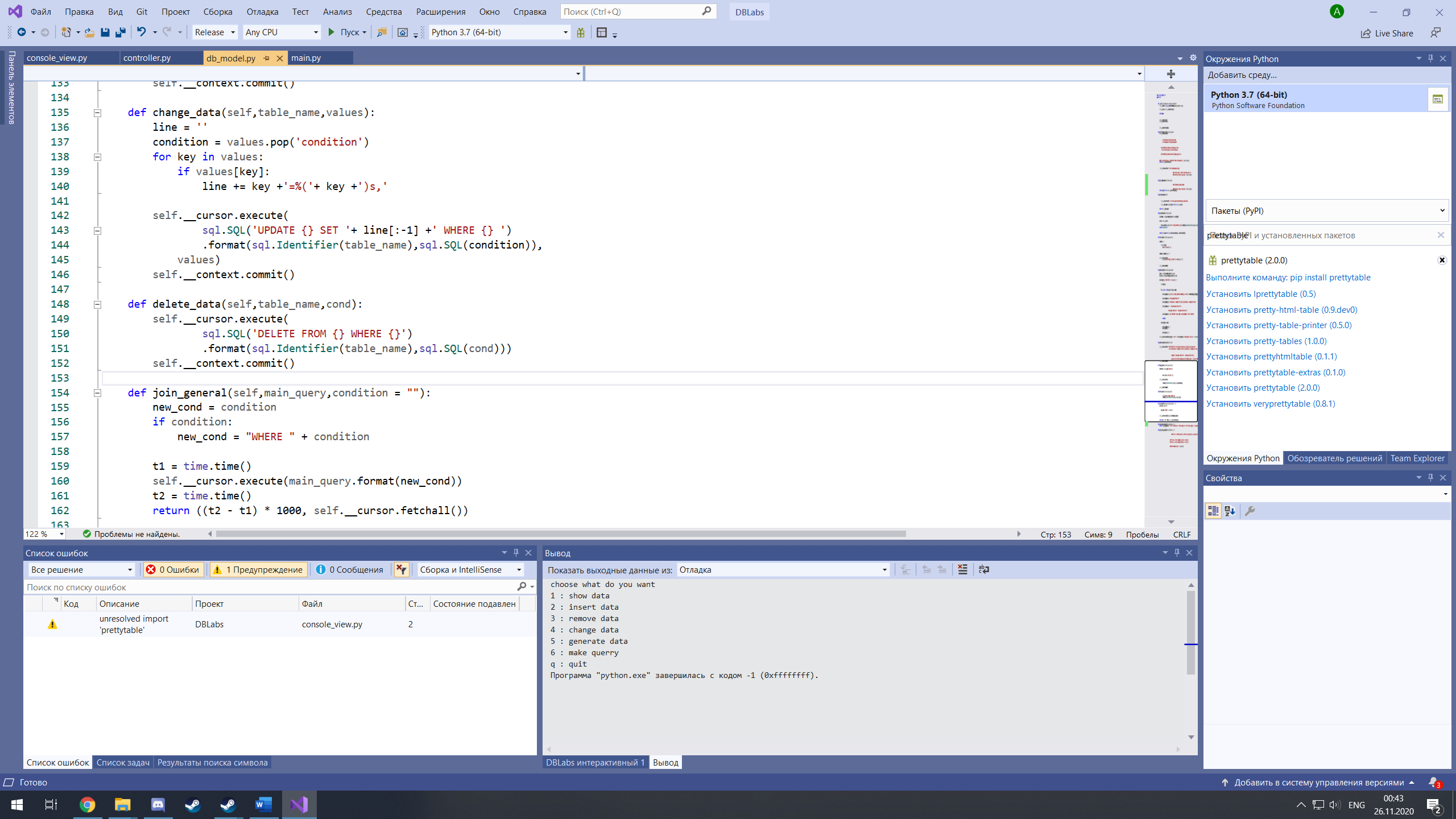


**Рис 2.4** **Метод get\_table\_data**

**Видалення зв’язаних між собою даних**

Наприклад таблиця Users з’єднана з таблицею Passport і Video то при видаленні запису з users будуть видаленні всі пов'язані записи з таблиць passport та video. Це реалізується засобами PostgresSQL за допомогою каскадного видалення.

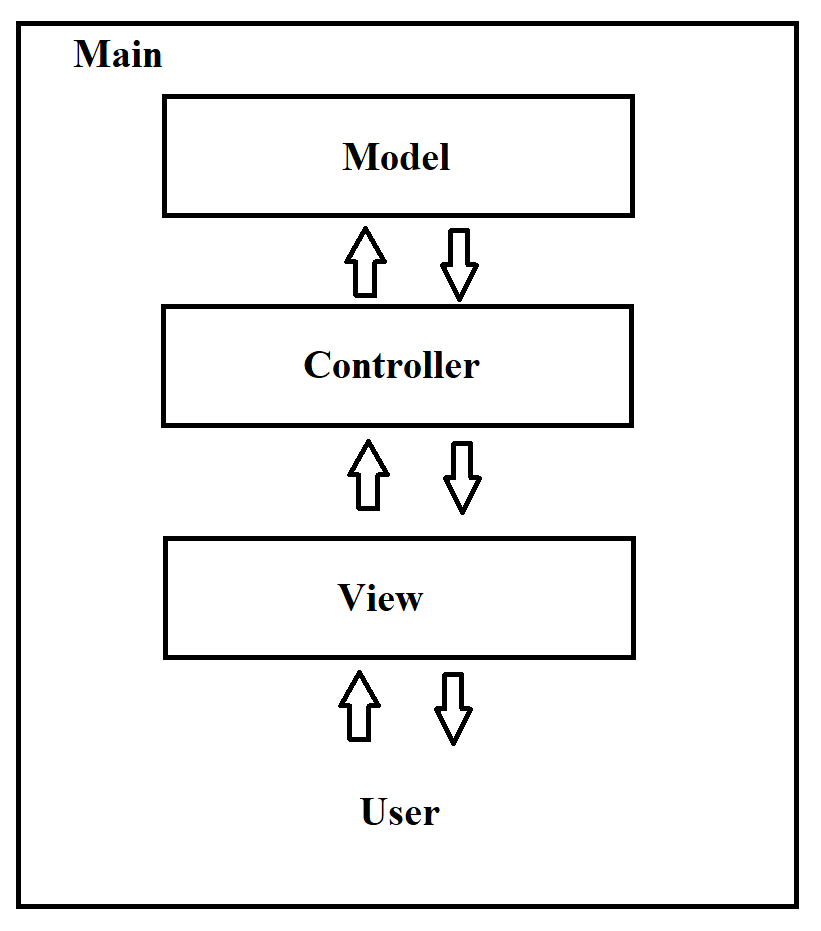
Метод що видаляє данні з таблиці



**Рис 2.4** **Метод delete\_data**

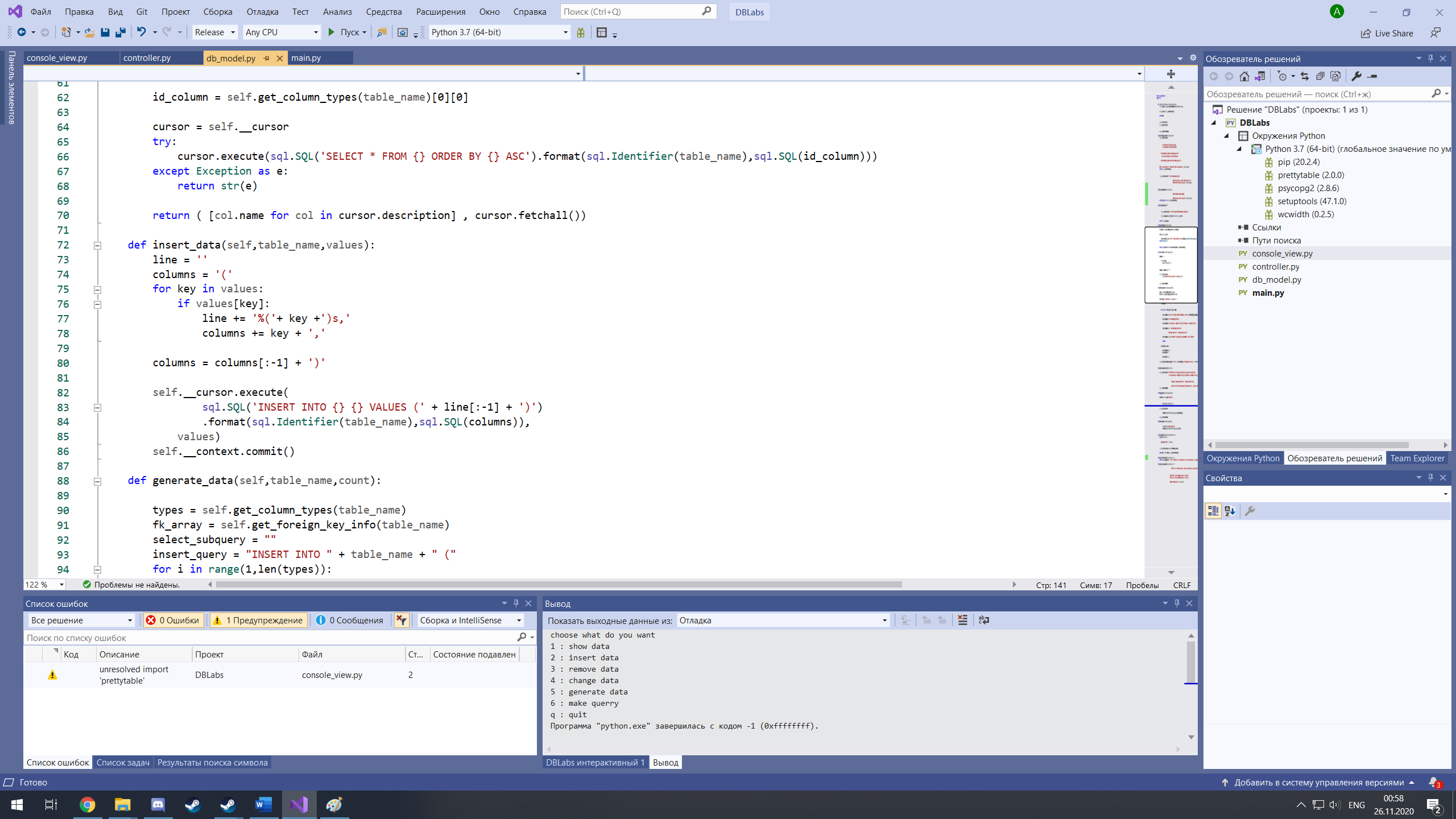
**Структура програми**

Програма створена за патерном MVC (Model-View-Controller). Складається з модулів Controller(controller.py), Model(db\_model.py) View(console\_view.py).

****

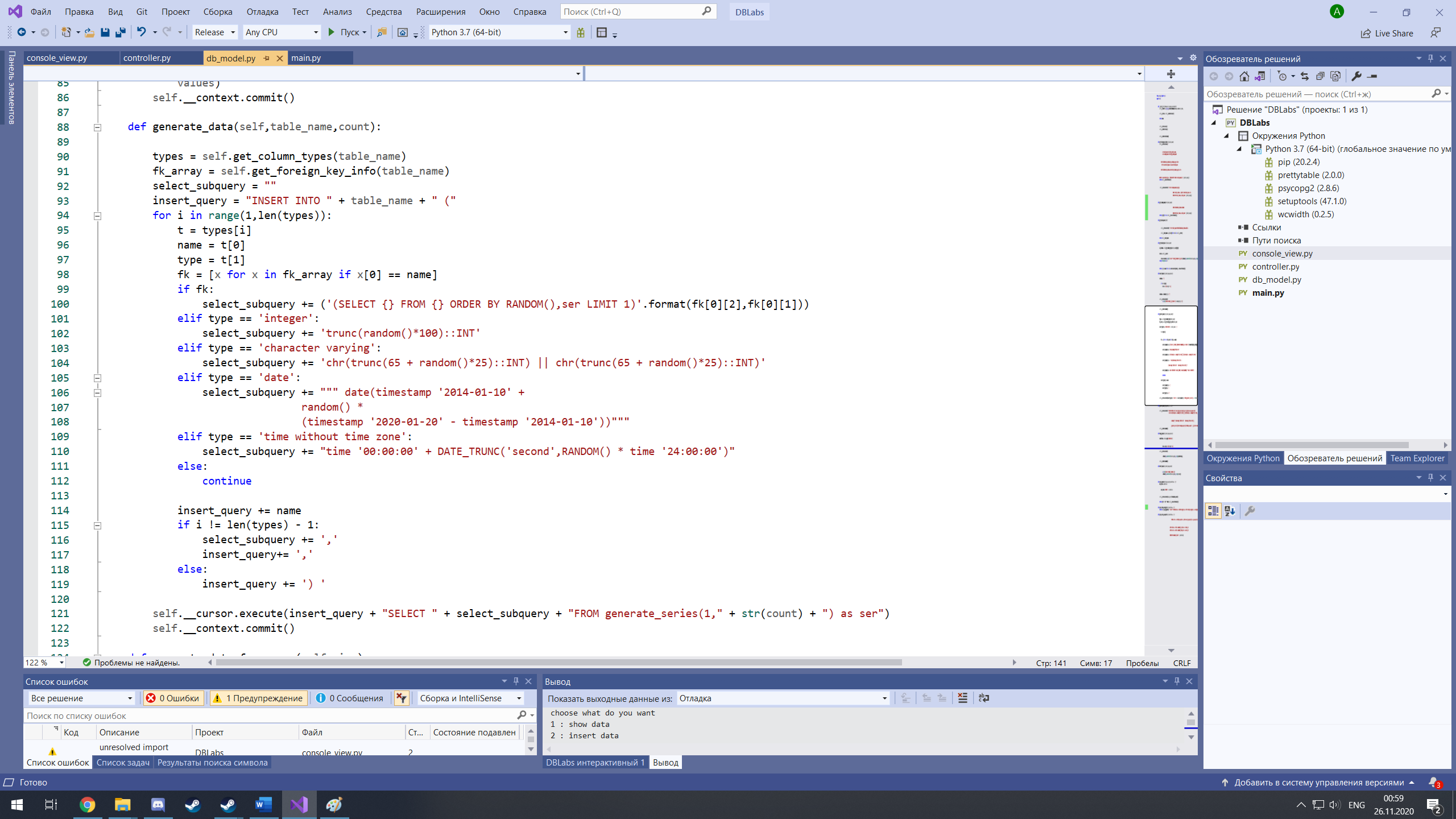
**Рис 2.5 Структура програми**

**Додавання запису в таблицю**



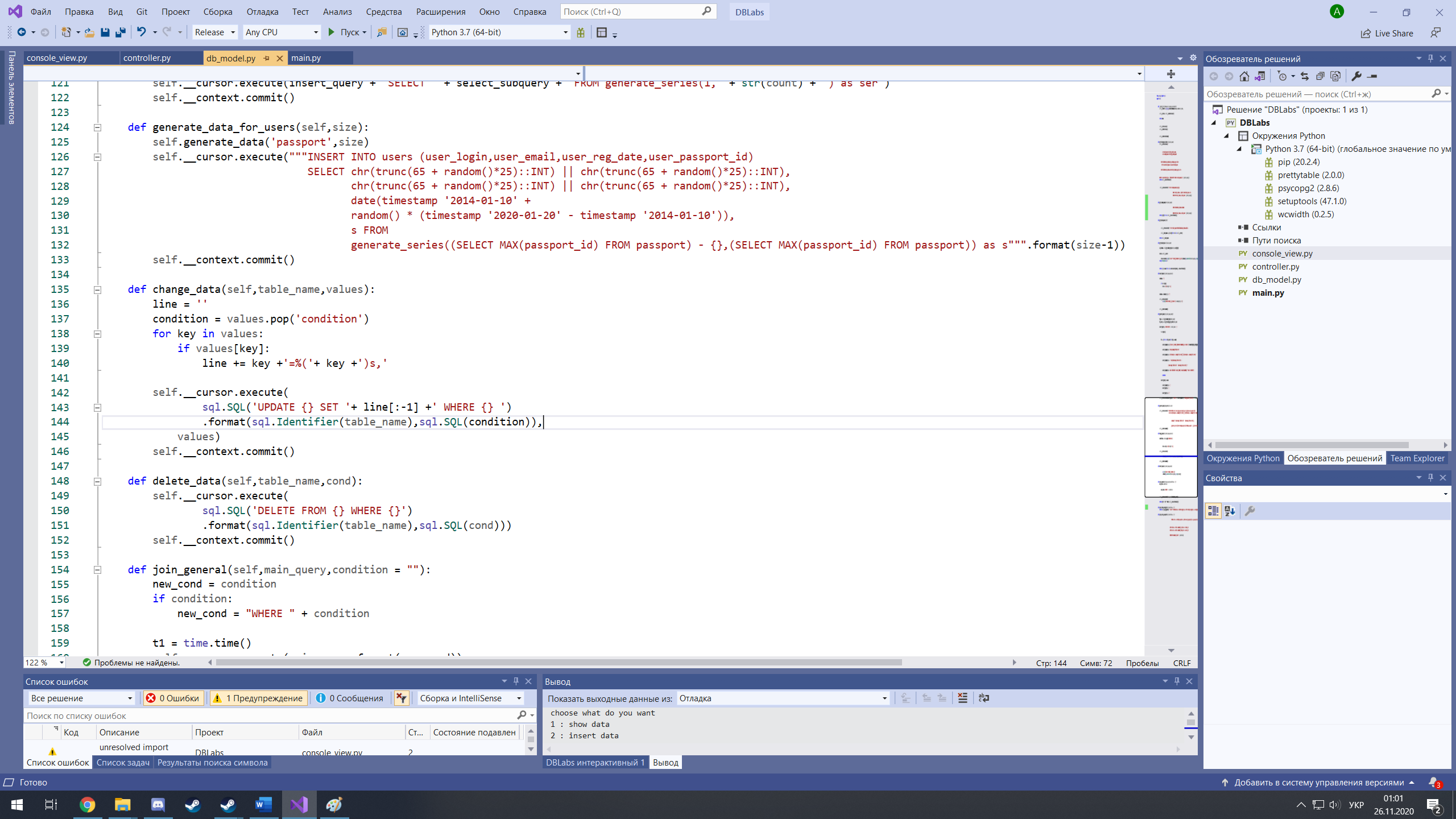
**Рис 2.6** **Метод insert\_data**

**Додавання N згенерованих записів в таблицю**



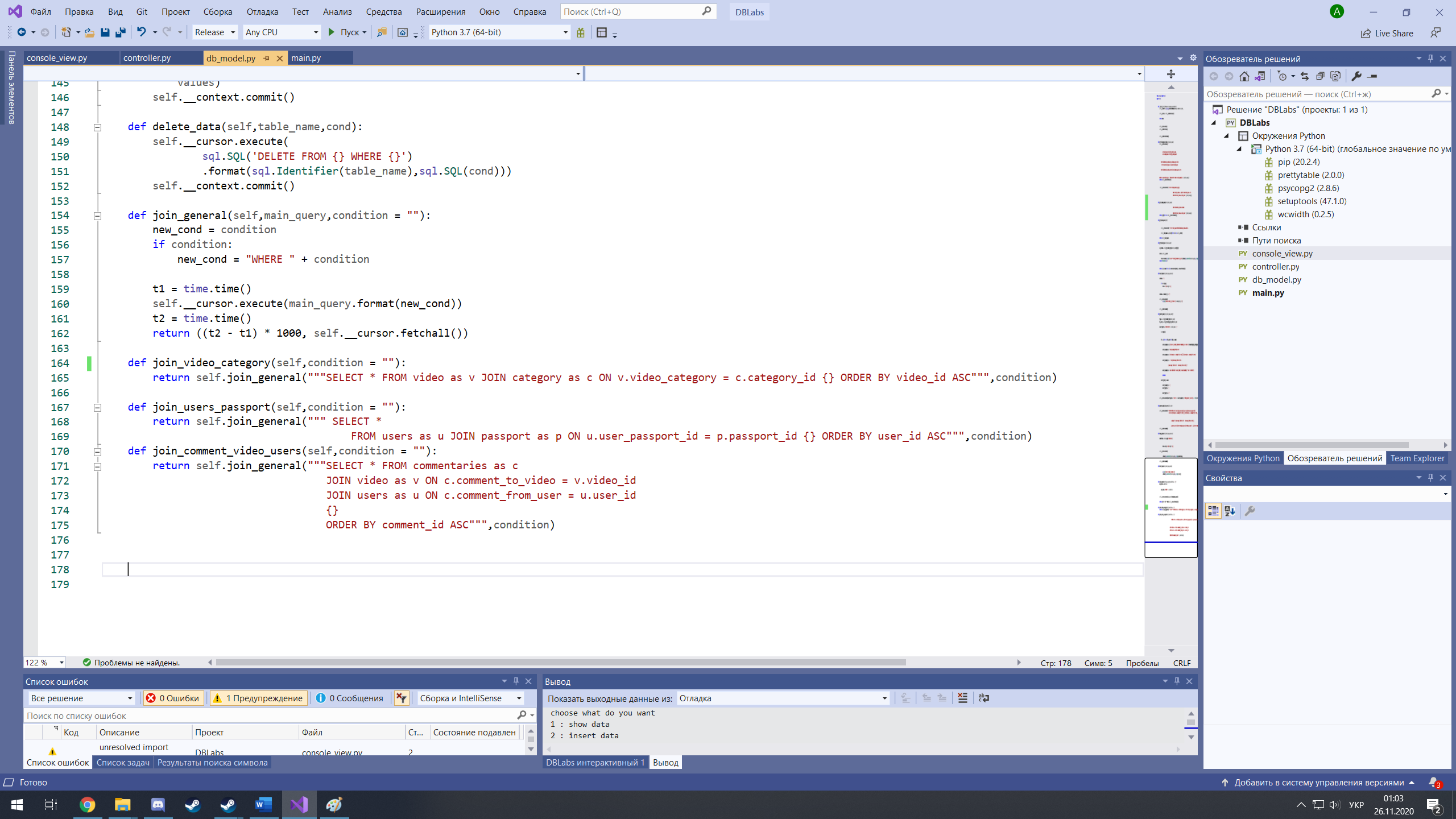
**Рис 2.7** **Метод generate\_data**

**Зміна даних в таблиці**



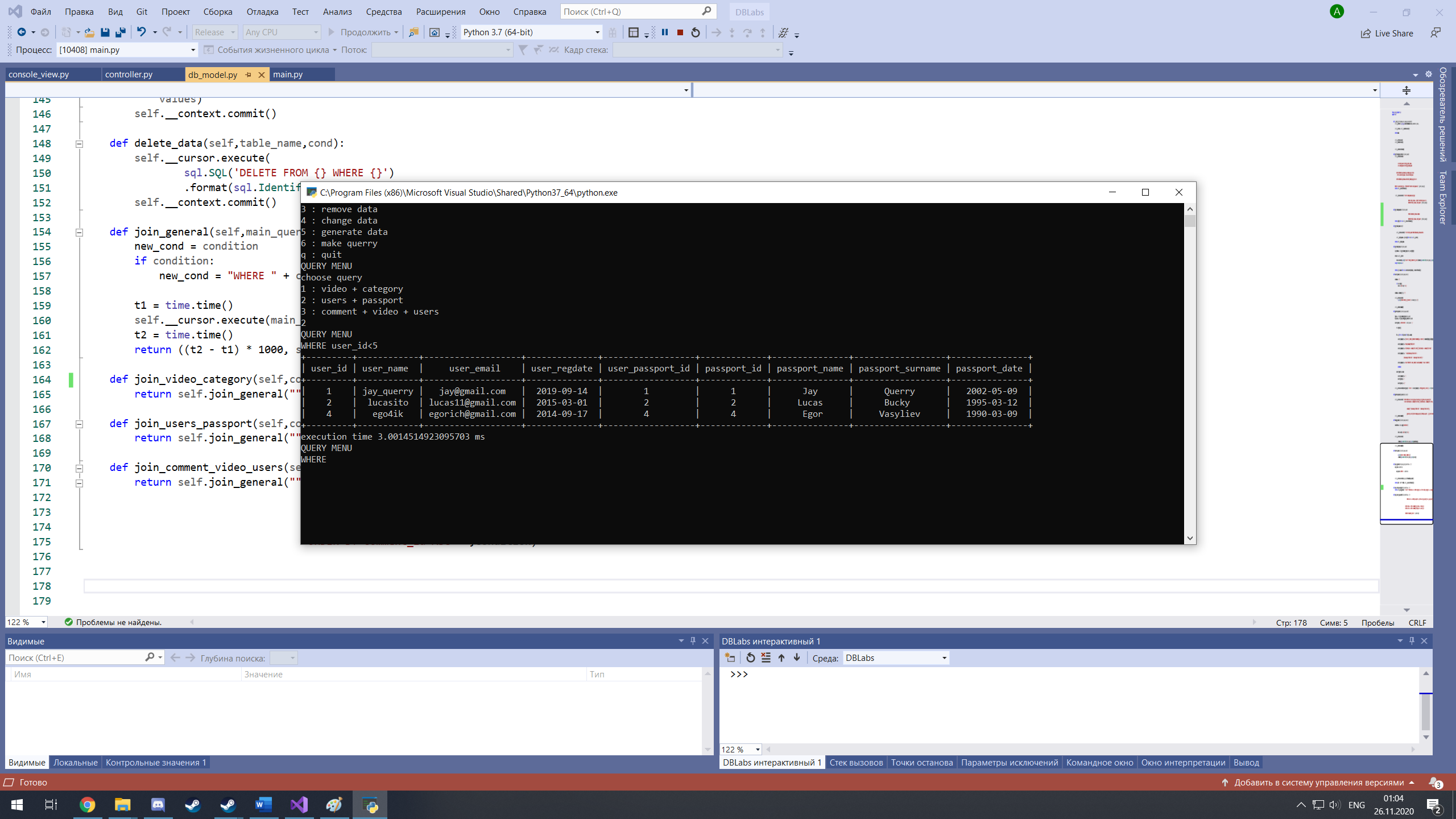
**Рис 2.8** **Метод change\_data**

**Пошук в декількох таблицях**



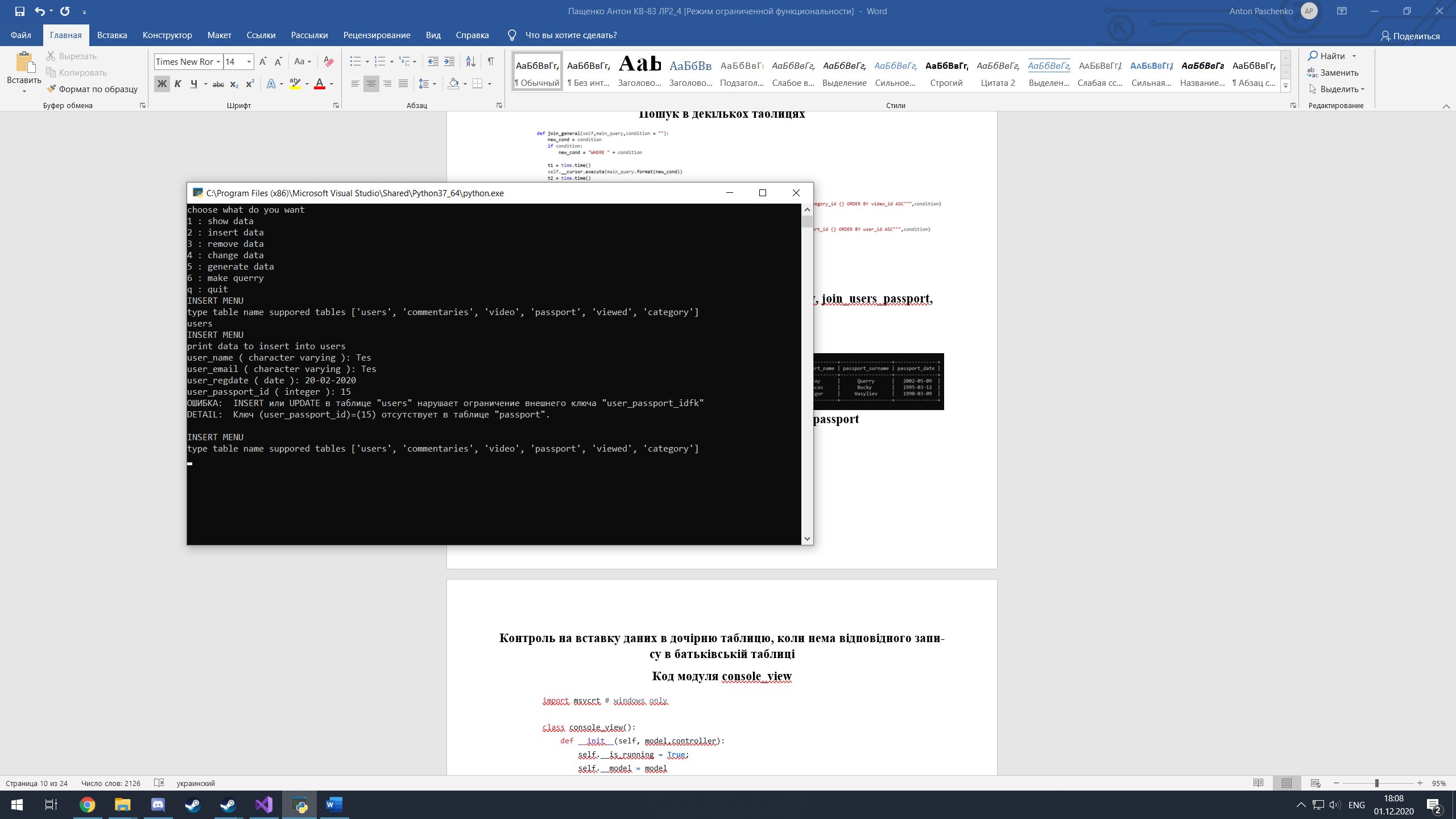
**Рис 2.9** **Методи join\_general, join\_video\_category, join\_users\_passport, join\_comment\_video\_users**

**Приклад пошуку**



**Рис 2.8** **Об'єднані таблиці users та passport**

**Контроль на вставку даних в дочірню таблицю, коли нема відповідного запису в батьківській таблиці**



**Рис 2.9** **Помилка при введення некоректного зовнішнього ключа**

**Код модуля console\_view**

|  |  |
| --- | --- |
|  | import msvcrt # windows only |
|  |  |
|  | class console\_view(): |
|  | def \_\_init\_\_(self, model,controller): |
|  | self.\_\_is\_running = True; |
|  | self.\_\_model = model |
|  | self.\_\_controller = controller |
|  |  |
|  | def quit(self): |
|  | return None |
|  |  |
|  | def print\_message(self,message,callback = None): |
|  | print(message) |
|  | if callback is not None: |
|  | return callback |
|  | return None |
|  |  |
|  | def print\_table(self,table,callback = None): |
|  |  |
|  | t = PrettyTable(table[0]) |
|  | for row in table[1]: |
|  | t.add\_row(row) |
|  | print(t) |
|  |  |
|  | if callback is not None: |
|  | return callback |
|  | return None |
|  |  |
|  | def get\_input(self,callback = None): |
|  | inp = input() |
|  | if callback is not None: |
|  | pass |
|  |  |
|  | def main\_menu(self): |
|  | print('choose what do you want') |
|  | print('1 : show data') |
|  | print('2 : insert data') |
|  | print('3 : remove data') |
|  | print('4 : change data') |
|  | print('5 : generate data') |
|  | print('6 : make querry') |
|  | print('q : quit') |
|  | char = chr(msvcrt.getch()[0]) |
|  | return self.\_\_controller.main\_menu(char) |
|  |  |
|  | def show\_menu(self): |
|  | print("SHOW MENU") |
|  | table\_names = self.\_\_model.get\_table\_names() |
|  | print('type table name or quit') |
|  | print('suppored tables') |
|  | print(table\_names) |
|  | answer = input() |
|  | return self.\_\_controller.show\_menu(answer) |
|  |  |
|  | def choose\_delete\_menu(self): |
|  | print("DELETE MENU") |
|  | table\_names = self.\_\_model.get\_table\_names() |
|  | print('type table name','suppored tables',table\_names) |
|  | answer = input() |
|  | return self.\_\_controller.choose\_delete\_menu(answer) |
|  |  |
|  | def choose\_insert\_menu(self): |
|  | print("INSERT MENU") |
|  | table\_names = self.\_\_model.get\_table\_names() |
|  | print('type table name','suppored tables',table\_names) |
|  | answer = input() |
|  | return self.\_\_controller.choose\_insert\_menu(answer) |
|  |  |
|  | def choose\_change\_menu(self): |
|  | print("CHANGE MENU") |
|  | table\_names = self.\_\_model.get\_table\_names() |
|  | print('type table name','suppored tables',table\_names) |
|  | answer = input() |
|  | return self.\_\_controller.choose\_change\_menu(answer) |
|  |  |
|  | def choose\_generate\_menu(self): |
|  | print("GENERATE MENU") |
|  | table\_names = self.\_\_model.get\_table\_names() |
|  | print('type table name','suppored tables',table\_names) |
|  | answer = input() |
|  | return self.\_\_controller.choose\_generate\_menu(answer) |
|  |  |
|  | def choose\_query\_menu(self,query\_list): |
|  | print("QUERY MENU") |
|  | print("choose query") |
|  | for i in range(0,len(query\_list)): |
|  | print(i + 1,":",query\_list[i]) |
|  | answer = input() |
|  |  |
|  | return self.\_\_controller.choose\_query\_menu(answer) |
|  |  |
|  | def insert\_row\_menu(self,table\_name,columns\_data): |
|  | print("INSERT MENU") |
|  | print('print data to insert into',table\_name) |
|  | answer = {} |
|  |  |
|  | for data in columns\_data: |
|  | print(data[0],'(',data[1],'):',end=' ') |
|  | inp = input() |
|  | answer.update({data[0] : inp}) |
|  |  |
|  | return self.\_\_controller.insert\_data(table\_name,answer) |
|  |  |
|  | def change\_row\_menu(self,table\_name,columns\_data): |
|  | print("CHANGE MENU") |
|  | print('print data to change into',table\_name) |
|  | answer = {} |
|  |  |
|  | for data in columns\_data: |
|  | print(data[0],'(',data[1],'):',end=' ') |
|  | inp = input() |
|  | answer.update({data[0] : inp}) |
|  | print('WHERE',end =' ') |
|  | inp = input() |
|  | answer.update({'condition' : inp}) |
|  |  |
|  | return self.\_\_controller.change\_data(table\_name,answer) |
|  |  |
|  | def delete\_row\_menu(self,table\_name,column\_arr): |
|  | print("DELETE MENU") |
|  | print(table\_name) |
|  | print("columns") |
|  | print(column\_arr) |
|  | #self.\_\_model.get |
|  | print("WHERE",end = " ") |
|  | answer = input() |
|  | return self.\_\_controller.delete\_data(table\_name,answer) |
|  |  |
|  | def generate\_size\_menu(self,table\_name): |
|  | print("GENERATE MENU") |
|  | print('print how many rows you want to generate') |
|  | answer = input() |
|  | return self.\_\_controller.generate\_data(table\_name,answer) |
|  |  |
|  | def cond\_query\_menu(self,query\_num): |
|  | print("QUERY MENU") |
|  | print("WHERE",end = " ") |
|  | cond = input() |
|  | return self.\_\_controller.cond\_query\_menu(query\_num,cond) |
|  |  |

**Код модуля controller**

|  |
| --- |
|  |
|  | class controller():  def \_\_init\_\_(self, model): |
|  | self.\_\_model = model |
|  | self.\_\_query\_data = ["video + category","users + passport","comment + video + users"] |
|  |  |
|  | def set\_view(self,view): |
|  | self.\_\_view = view |
|  |  |
|  | def get\_view\_func(self,name,\*args): |
|  | return lambda : getattr(self.\_\_view, name)(\*args) |
|  |  |
|  | def check\_input(self,data,prev\_menu = "main\_menu"): |
|  |  |
|  | if data == ':main': |
|  | return self.get\_view\_func("main\_menu") |
|  |  |
|  | if data == ':ret': |
|  | return self.get\_view\_func(prev\_menu) |
|  |  |
|  | if data == ':q': |
|  | return self.get\_view\_func("quit") |
|  |  |
|  | def main\_menu(self,answer): |
|  | if answer == '1': |
|  | return self.get\_view\_func("show\_menu") |
|  | if answer == '2': |
|  | return self.get\_view\_func("choose\_insert\_menu") |
|  | if answer == '3': |
|  | return self.get\_view\_func("choose\_delete\_menu") |
|  | if answer == '4': |
|  | return self.get\_view\_func("choose\_change\_menu") |
|  | if answer == '5': |
|  | return self.get\_view\_func("choose\_generate\_menu") |
|  | if answer == '6': |
|  | return self.get\_view\_func("choose\_query\_menu",self.\_\_query\_data) |
|  | if answer == 'q': |
|  | return self.get\_view\_func("quit") |
|  |  |
|  | return self.get\_view\_func("main\_menu") |
|  |  |
|  | def show\_menu(self,data): |
|  |  |
|  | res = self.check\_input(data) |
|  | if res is not None: |
|  | return res |
|  |  |
|  | model\_data = self.\_\_model.get\_table\_data(data) |
|  |  |
|  | if type(model\_data) == str: |
|  | return self.get\_view\_func("print\_message",model\_data,self.get\_view\_func("show\_menu")) |
|  |  |
|  | return self.get\_view\_func("print\_table",model\_data,self.get\_view\_func("show\_menu")) |
|  |  |
|  | def check\_table(self,table\_name,curr\_menu): |
|  |  |
|  | if table\_name not in self.\_\_model.get\_table\_names(): |
|  | return self.get\_view\_func("print\_message","unknown table {0}".format(table\_name),self.get\_view\_func(curr\_menu)) |
|  |  |
|  | return None |
|  |  |
|  |  |
|  | def choose\_insert\_menu(self,data): |
|  |  |
|  | res = self.check\_input(data) |
|  | if res is not None: |
|  | return res |
|  |  |
|  | res = self.check\_table(data,"choose\_insert\_menu") |
|  |  |
|  | if res is not None: |
|  | return res |
|  |  |
|  | return self.get\_view\_func("insert\_row\_menu",data,self.\_\_model.get\_column\_types(data)[1:]) |
|  |  |
|  | def choose\_delete\_menu(self,data): |
|  |  |
|  | res = self.check\_input(data) |
|  | if res is not None: |
|  | return res |
|  |  |
|  | res = self.check\_table(data,"choose\_delete\_menu") |
|  |  |
|  | if res is not None: |
|  | return res |
|  |  |
|  | column\_arr = self.\_\_model.get\_column\_names(data) |
|  |  |
|  | return self.get\_view\_func("delete\_row\_menu",data,column\_arr) |
|  |  |
|  | def choose\_change\_menu(self,data): |
|  |  |
|  | res = self.check\_input(data) |
|  | if res is not None: |
|  | return res |
|  |  |
|  | res = self.check\_table(data,"choose\_change\_menu") |
|  |  |
|  | if res is not None: |
|  | return res |
|  |  |
|  | return self.get\_view\_func("change\_row\_menu",data,self.\_\_model.get\_column\_types(data)[1:]) |
|  |  |
|  | def choose\_generate\_menu(self,data): |
|  | res = self.check\_input(data) |
|  | if res is not None: |
|  | return res |
|  |  |
|  | res = self.check\_table(data,"choose\_generate\_menu") |
|  |  |
|  | if res is not None: |
|  | return res |
|  |  |
|  | return self.get\_view\_func("generate\_size\_menu",data) |
|  |  |
|  | def choose\_query\_menu(self,data): |
|  | res = self.check\_input(data) |
|  | if res is not None: |
|  | return res |
|  |  |
|  | int\_data = 0 |
|  | res\_func = None |
|  | try: |
|  | int\_data = int(data) |
|  | if int\_data <= len(self.\_\_query\_data) and int\_data > 0: |
|  | res\_func = self.get\_view\_func("cond\_query\_menu",int\_data) |
|  | else: |
|  | self.get\_view\_func("print\_message","uknown index {}".format(int\_data),self.get\_view\_func("choose\_query\_menu",self.\_\_query\_data)) |
|  | except Exception as e: |
|  | res\_func = self.get\_view\_func("print\_message",str(e),self.get\_view\_func("choose\_query\_menu",self.\_\_query\_data)) |
|  |  |
|  | return res\_func |
|  |  |
|  |  |
|  |  |
|  | def change\_data(self,table :str ,new\_data : dict): |
|  |  |
|  | res = self.check\_input(new\_data,"choose\_change\_menu") |
|  | if res is not None: |
|  | return res |
|  |  |
|  | message = " " |
|  |  |
|  | try: |
|  | self.\_\_model.change\_data(table,new\_data) |
|  | message = "Success" |
|  | except Exception as e: |
|  | message = str(e) |
|  | self.\_\_model.clear\_transaction() |
|  |  |
|  | column\_data = self.\_\_model.get\_column\_types(table) |
|  | return self.get\_view\_func("print\_message",message,self.get\_view\_func("choose\_change\_menu")) |
|  |  |
|  | def insert\_data(self,table :str ,data : dict): |
|  |  |
|  | res = self.check\_input(data,"choose\_insert\_menu") |
|  | if res is not None: |
|  | return res |
|  |  |
|  | message = " " |
|  | try: |
|  | self.\_\_model.insert\_data(table,data) |
|  | message = "Success" |
|  | except Exception as e: |
|  | message = str(e) |
|  | self.\_\_model.clear\_transaction() |
|  |  |
|  | return self.get\_view\_func("print\_message",message,self.get\_view\_func("choose\_insert\_menu")) |
|  |  |
|  |  |
|  | def delete\_data(self,table :str ,data : str): |
|  | res = self.check\_input(data,"choose\_delete\_menu") |
|  | message = " " |
|  | if res is not None: |
|  | return res |
|  | try: |
|  | self.\_\_model.delete\_data(table,data) |
|  | message = "Success" |
|  | except Exception as e: |
|  | message = str(e) |
|  | self.\_\_model.clear\_transaction() |
|  |  |
|  | column\_arr = self.\_\_model.get\_column\_names(table) |
|  |  |
|  | return self.get\_view\_func("print\_message",message,self.get\_view\_func("delete\_row\_menu",table,column\_arr)) |
|  |  |
|  | def generate\_data(self,table\_name,data): |
|  | res = self.check\_input(data,"choose\_generate\_menu") |
|  |  |
|  | if res is not None: |
|  | return res |
|  |  |
|  | message = " " |
|  | data\_int = 0 |
|  |  |
|  | try: |
|  | data\_int = int(data) |
|  | except Exception as e: |
|  | return self.get\_view\_func("print\_message",str(e),self.get\_view\_func("generate\_size\_menu",table\_name)) |
|  |  |
|  |  |
|  | if res is not None: |
|  | return res |
|  |  |
|  | if table\_name == 'users': |
|  | try: |
|  | self.\_\_model.generate\_data\_for\_users(data\_int) |
|  | message = "Success" |
|  | except Exception as e: |
|  | message = str(e) |
|  | self.\_\_model.clear\_transaction() |
|  | else: |
|  | try: |
|  | self.\_\_model.generate\_data(table\_name,data\_int) |
|  | message = "Success" |
|  | except Exception as e: |
|  | message = str(e) |
|  | self.\_\_model.clear\_transaction() |
|  |  |
|  | return self.get\_view\_func("print\_message",message,self.get\_view\_func("generate\_size\_menu",table\_name)) |
|  |  |
|  | def cond\_query\_menu(self,query\_num,cond): |
|  |  |
|  | if cond == ':main': |
|  | return self.get\_view\_func("main\_menu") |
|  |  |
|  | if cond == ':ret': |
|  | return self.get\_view\_func("choose\_query\_menu",self.\_\_query\_data) |
|  |  |
|  | if cond == ':q': |
|  | return self.get\_view\_func("quit") |
|  |  |
|  | query\_func = None |
|  | column\_arr = [] |
|  | if query\_num == 1: |
|  | query\_func = getattr(self.\_\_model, "join\_video\_category") |
|  | column\_arr = self.\_\_model.get\_column\_names('video') + self.\_\_model.get\_column\_names('category') |
|  | elif query\_num == 2: |
|  | query\_func = getattr(self.\_\_model, "join\_users\_passport") |
|  | column\_arr = self.\_\_model.get\_column\_names('users') + self.\_\_model.get\_column\_names('passport') |
|  | elif query\_num == 3: |
|  | query\_func = getattr(self.\_\_model, "join\_comment\_video\_users") |
|  | column\_arr = self.\_\_model.get\_column\_names('commentaries') |
|  | + self.\_\_model.get\_column\_names('video') |
|  | + self.\_\_model.get\_column\_names('users') |
|  |  |
|  | ret\_func = None |
|  | try: |
|  | data = query\_func(cond) |
|  | ret\_func = self.get\_view\_func("print\_table",(column\_arr,data[1]), |
|  | self.get\_view\_func("print\_message","execution time {} ms".format(data[0]),self.get\_view\_func("cond\_query\_menu",query\_num))) |
|  | except Exception as e: |
|  | ret\_func = self.get\_view\_func("print\_message",str(e),self.get\_view\_func("cond\_query\_menu",query\_num)) |
|  | self.\_\_model.clear\_transaction() |
|  |  |
|  | return ret\_func |

**Код модуля db\_model**

|  |  |
| --- | --- |
|  | import psycopg2  from psycopg2 import sql  import time |
|  |  |
|  | class db\_model(): |
|  | def \_\_init\_\_(self,dbname,user\_name,password,host): |
|  | self.\_\_context = psycopg2.connect(dbname=dbname, user=user\_name, |
|  | password=password, host=host) |
|  | self.\_\_cursor = self.\_\_context.cursor() |
|  | self.\_\_table\_names = None |
|  | return None |
|  |  |
|  | def \_\_del\_\_(self): |
|  | self.\_\_cursor.close() |
|  | self.\_\_context.close() |
|  |  |
|  | def clear\_transaction(self): |
|  | self.\_\_context.rollback() |
|  |  |
|  | def get\_foreign\_key\_info(self,table\_name): |
|  | self.\_\_cursor.execute(""" |
|  | SELECT |
|  | kcu.column\_name, |
|  | ccu.table\_name AS foreign\_table\_name, |
|  | ccu.column\_name AS foreign\_column\_name |
|  | FROM |
|  | information\_schema.table\_constraints AS tc |
|  | JOIN information\_schema.key\_column\_usage AS kcu |
|  | ON tc.constraint\_name = kcu.constraint\_name |
|  | AND tc.table\_schema = kcu.table\_schema |
|  | JOIN information\_schema.constraint\_column\_usage AS ccu |
|  | ON ccu.constraint\_name = tc.constraint\_name |
|  | AND ccu.table\_schema = tc.table\_schema |
|  | WHERE tc.constraint\_type = 'FOREIGN KEY' AND tc.table\_name=%s;""",(table\_name,)) |
|  | return self.\_\_cursor.fetchall() |
|  |  |
|  | def get\_column\_types(self,table\_name): |
|  | self.\_\_cursor.execute("""SELECT column\_name, data\_type |
|  | FROM information\_schema.columns |
|  | WHERE table\_schema = 'public' AND table\_name = %s |
|  | ORDER BY table\_schema, table\_name""",(table\_name,)) |
|  | return self.\_\_cursor.fetchall() |
|  |  |
|  | def get\_column\_names(self,table\_name): |
|  | self.\_\_cursor.execute("""SELECT column\_name |
|  | FROM information\_schema.columns |
|  | WHERE table\_schema = 'public' AND table\_name = %s |
|  | ORDER BY table\_schema, table\_name""",(table\_name,)) |
|  | return [x[0] for x in self.\_\_cursor.fetchall()] |
|  |  |
|  | def get\_table\_names(self): |
|  |  |
|  | if self.\_\_table\_names is None: |
|  | self.\_\_cursor.execute("""SELECT table\_name FROM information\_schema.tables |
|  | WHERE table\_schema = 'public'""") |
|  | self.\_\_table\_names = [table[0] for table in self.\_\_cursor] |
|  |  |
|  | return self.\_\_table\_names |
|  |  |
|  | def get\_table\_data(self,table\_name): |
|  |  |
|  | id\_column = self.get\_column\_types(table\_name)[0][0] |
|  |  |
|  | cursor = self.\_\_cursor |
|  | try: |
|  | cursor.execute(sql.SQL('SELECT \* FROM {} ORDER BY {} ASC').format(sql.Identifier(table\_name),sql.SQL(id\_column))) |
|  | except Exception as e: |
|  | return str(e) |
|  |  |
|  | return ( [col.name for col in cursor.description] , cursor.fetchall()) |
|  |  |
|  | def insert\_data(self,table\_name,values): |
|  | line = '' |
|  | columns = '(' |
|  | for key in values: |
|  | if values[key]: |
|  | line += '%('+ key +')s,' |
|  | columns += key + ',' |
|  |  |
|  | columns = columns[:-1] + ')' |
|  |  |
|  | self.\_\_cursor.execute( |
|  | sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')') |
|  | .format(sql.Identifier(table\_name),sql.SQL(columns)), |
|  | values) |
|  | self.\_\_context.commit() |
|  |  |
|  | def generate\_data(self,table\_name,count): |
|  |  |
|  | types = self.get\_column\_types(table\_name) |
|  | fk\_array = self.get\_foreign\_key\_info(table\_name) |
|  | select\_subquery = "" |
|  | insert\_query = "INSERT INTO " + table\_name + " (" |
|  | for i in range(1,len(types)): |
|  | t = types[i] |
|  | name = t[0] |
|  | type = t[1] |
|  | fk = [x for x in fk\_array if x[0] == name] |
|  | if fk: |
|  | select\_subquery += ('(SELECT {} FROM {} ORDER BY RANDOM(),ser LIMIT 1)'.format(fk[0][2],fk[0][1])) |
|  | elif type == 'integer': |
|  | select\_subquery += 'trunc(random()\*100)::INT' |
|  | elif type == 'character varying': |
|  | select\_subquery += 'chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + random()\*25)::INT)' |
|  | elif type == 'date': |
|  | select\_subquery += """ date(timestamp '2014-01-10' + |
|  | random() \* |
|  | (timestamp '2020-01-20' - timestamp '2014-01-10'))""" |
|  | elif type == 'time without time zone': |
|  | select\_subquery += "time '00:00:00' + DATE\_TRUNC('second',RANDOM() \* time '24:00:00')" |
|  | else: |
|  | continue |
|  |  |
|  | insert\_query += name |
|  | if i != len(types) - 1: |
|  | select\_subquery += ',' |
|  | insert\_query+= ',' |
|  | else: |
|  | insert\_query += ') ' |
|  |  |
|  | self.\_\_cursor.execute(insert\_query + "SELECT " + select\_subquery + "FROM generate\_series(1," + str(count) + ") as ser") |
|  | self.\_\_context.commit() |
|  |  |
|  | def generate\_data\_for\_users(self,size): |
|  | self.generate\_data('passport',size) |
|  | self.\_\_cursor.execute("""INSERT INTO users (user\_login,user\_email,user\_reg\_date,user\_passport\_id) |
|  | SELECT chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + random()\*25)::INT), |
|  | chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + random()\*25)::INT), |
|  | date(timestamp '2014-01-10' + |
|  | random() \* (timestamp '2020-01-20' - timestamp '2014-01-10')), |
|  | s FROM |
|  | generate\_series((SELECT MAX(passport\_id) FROM passport) - {},(SELECT MAX(passport\_id) FROM passport)) as s""".format(size-1)) |
|  | self.\_\_context.commit() |
|  |  |
|  | def change\_data(self,table\_name,values): |
|  | line = '' |
|  | condition = values.pop('condition') |
|  | for key in values: |
|  | if values[key]: |
|  | line += key +'=%('+ key +')s,' |
|  |  |
|  | self.\_\_cursor.execute( |
|  | sql.SQL('UPDATE {} SET '+ line[:-1] +' WHERE {} ') |
|  | .format(sql.Identifier(table\_name),sql.SQL(condition)), |
|  | values) |
|  | self.\_\_context.commit() |
|  |  |
|  | def delete\_data(self,table\_name,cond): |
|  | self.\_\_cursor.execute( |
|  | sql.SQL('DELETE FROM {} WHERE {}') |
|  | .format(sql.Identifier(table\_name),sql.SQL(cond))) |
|  | self.\_\_context.commit() |
|  |  |
|  | def join\_general(self,main\_query,condition = ""): |
|  | new\_cond = condition |
|  | if condition: |
|  | new\_cond = "WHERE " + condition |
|  |  |
|  | t1 = time.time() |
|  | self.\_\_cursor.execute(main\_query.format(new\_cond)) |
|  | t2 = time.time() |
|  | return ((t2 - t1) \* 1000, self.\_\_cursor.fetchall()) |
|  |  |
|  | def join\_video\_category(self,condition = ""): |
|  | return self.join\_general("""SELECT \* FROM video as v JOIN category as c ON v.video\_category = c.category\_id {} ORDER BY video\_id ASC""",condition) |
|  |  |
|  | def join\_users\_passport(self,condition = ""): |
|  | return self.join\_general(""" SELECT \* |
|  | FROM users as u JOIN passport as p ON u.user\_passport\_id = p.passport\_id {} ORDER BY user\_id ASC""",condition) |
|  | def join\_comment\_video\_users(self,condition = ""): |
|  | return self.join\_general("""SELECT \* FROM commentaries as c |
|  | JOIN video as v ON c.comment\_to\_video = v.video\_id |
|  | JOIN users as u ON c.comment\_from\_user = u.user\_id |
|  | {} |
|  | ORDER BY comment\_id ASC""",condition) |

**Код файла main.py**

|  |  |
| --- | --- |
|  | import psycopg2 |
|  | from db\_model import db\_model |
|  | from controller import controller |
|  | import os |
|  |  |
|  | def main(): |
|  | model = db\_model('anton','postgres','821083','127.0.0.1') |
|  | cont = controller(model) |
|  | view = console\_view(model,cont) |
|  | cont.set\_view(view) |
|  | func = view.main\_menu() |
|  | while func is not None: |
|  | func = func() |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | main() |