

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***”*

Виконав:

студент ІII курсу

групи КВ-02

Лунгов О.В.

Перевірив:

Павловський В. І.

Київ – 2022

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. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі. 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).

Деталізоване завдання:

Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв’язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати вилучення рядків за умови наявності даних у підлеглій таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні внесення нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.

Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими не мовою програмування, а відповідним SQL-запитом!

Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.

Програмний код організувати згідно шаблону Model-View-Controller(MVC). Приклад організації коду згідно шаблону доступний за даним посиланням. При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати лише мову SQL.

**Логічна модель предметної області «Бар»**

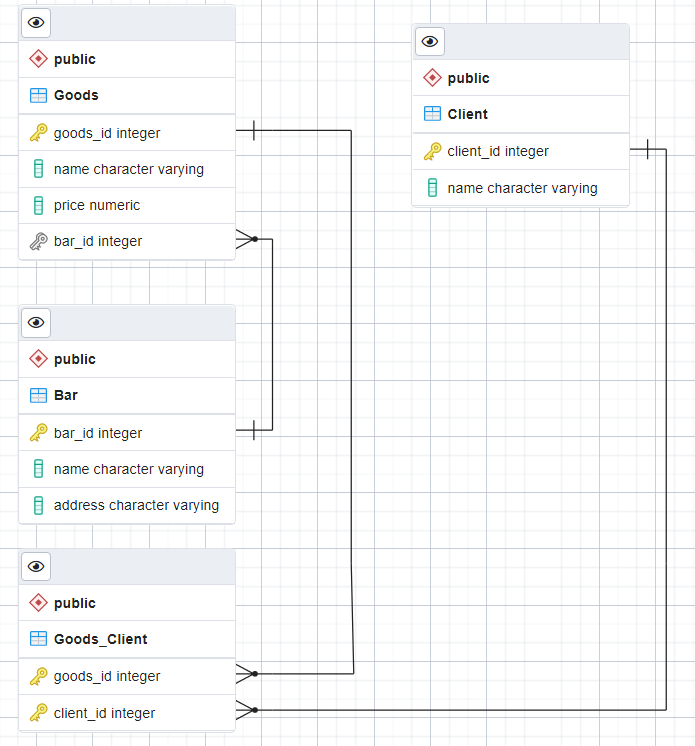


Рисунок 1. Схема бази даних, побудовано у [p](https://app.diagrams.net/)gAdmin 4.

**Середовище та компоненти розробки**

Для розробки використовувалась мова програмування Python, середовище розробки PyCharm, а також стороння бібліотека, що надає API для доступу до PostgreSQL – psycopg2.

**Шаблон проектування**

MVC - Шаблон проектування, який використаний у програмі.

Model – представляє клас, що описує логіку використовуваних даних. Згідно компоненту моделі, у моїй програмі відповідають всі компоненти які знаходять у файлі model.py.

View – в нашому випадку консольний інтерфейс з яким буде взаємодіяти наш користувач. Згідно компоненту представлення, то їй відповідають такі компоненти, згідно яким користувач бачить необхідні дані, що є представленням даних у вигляді консольного інтерфейсу.

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує введені користувачем дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок.

Task – представляє клас, котрий виконує таски

**Структура програми та її опис**

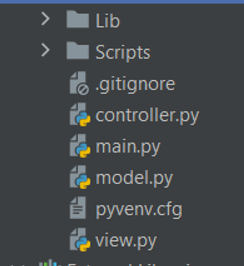


Рисунок 2. Структура програми.

Програма умовно поділена на 4 модулі: main.py, model.py, view.py, controller.py. Класи, як видно з їх назв, повністю відповідають використаному патерну MVC.

У файлі model.py описано клас Model, що займається регулювання підключення до бази даних, та виконанням низькорівневих запитів до неї клас Task.

У файлах view.py описано клас View, що виводить меню та приймає вхідні данні.

У файлі controller.py описано клас Controller, які оброблюються завдяки взаємодії з користувачем, запит бажаної дії, виконання пошуку, тощо.

У файлі main.py створюється екземпляр класу Controller, якому користувач передає вказівки щодо виконання програми.

**Структура меню програми**

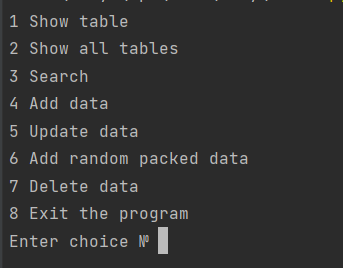


Рисунок 3. Меню для вибору операції над БД.

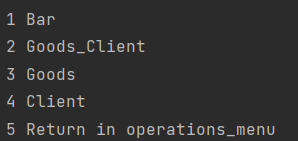


Рисунок 4. Меню для вибору таблиці, над якою проводитиметься обрана операція.

**Меню операцій складається з восьми пунктів (Рисунок 3)**

1. Виведення таблиці.
2. Виведення всіх таблиць.
3. Пошук усіх рядків з певної таблиці, які мають певні параметри, введені користувачем
4. Додавання нового рядку даних до таблиці.
5. Оновлення рядку даних у таблиці.
6. Додавання нових випадкових даних до кожної таблиці БД.
7. Видалення рядку даних з таблиці.
8. Завершення програми.

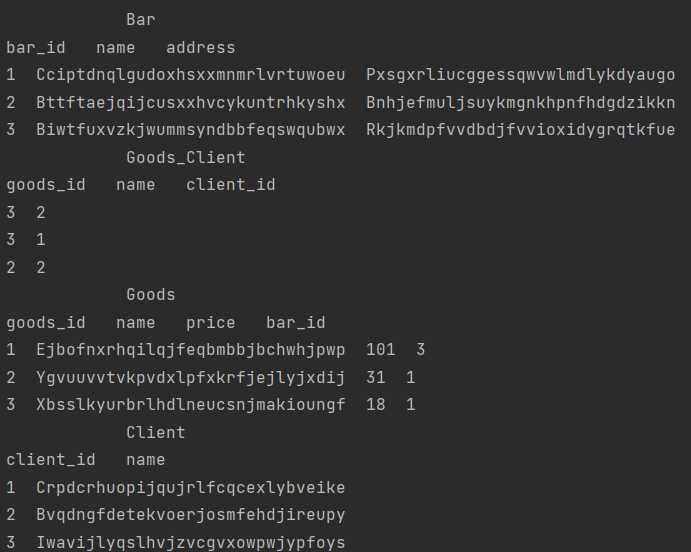
**Меню таблиць складається з 7 пунктів (Рисунок 3)**

1. Виконати обрану операцію над таблицею “Bar”.
2. Виконати обрану операцію над таблицею “Goods\_Client”.
3. Виконати обрану операцію над таблицею “Goods”.
4. Виконати обрану операцію над таблицею “Client”.
5. Повернутися до меню операцій

Слід також зауважити, що не всі операції потребують введення номеру таблиці, тому, при виборі операції номер 2 та 8 меню таблиць не виводиться.

**Результати та виконання операцій**

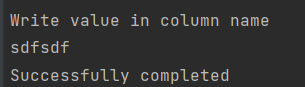
**Результат операції Show (для всіх таблиць)**

****

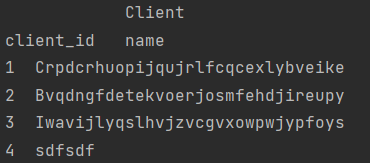
**Пошук інформації для “Client”**

****

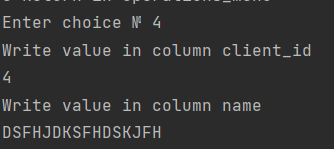
**Операція Add для таблиці “Client”**

****

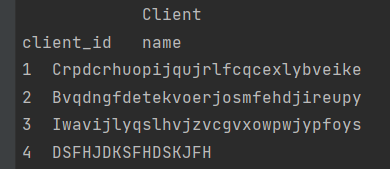
**Результат**

****

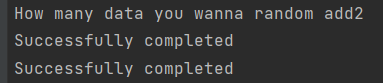
**Оновлення даних у таблиці “Client”**

****

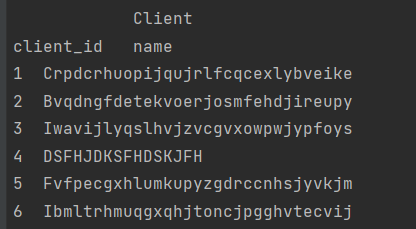
**Результат**

****

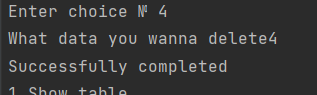
**Додаємо рандомних значень у таблицю “Client”**

****

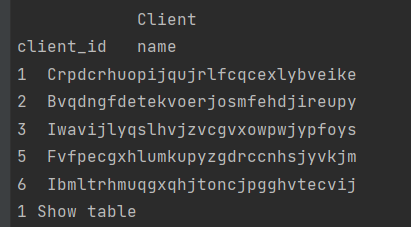
**Результат**

****

**Видалимо данні з таблиці “Client”**

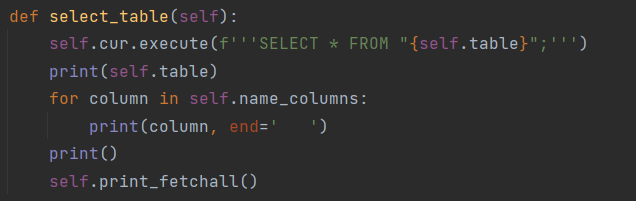
****

**Результат**

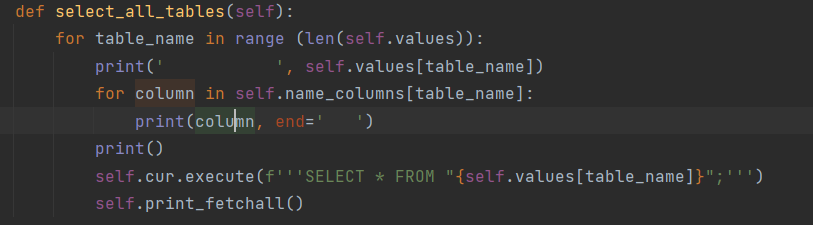
****

**Ілюстрації програмного коду модуля “model.py”**

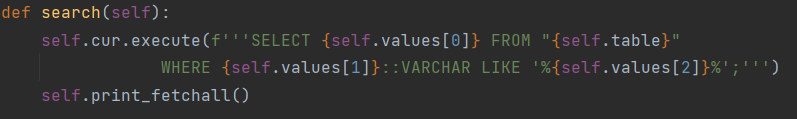
**Вибір даних з таблиць**

****

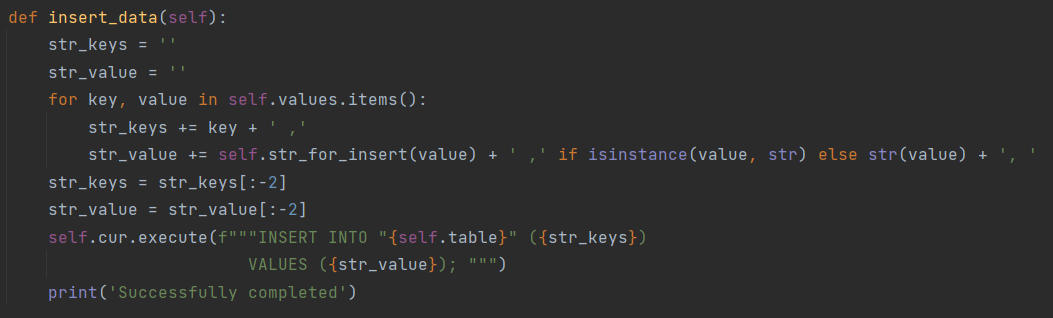
**Вибір даних з усіх таблиць**



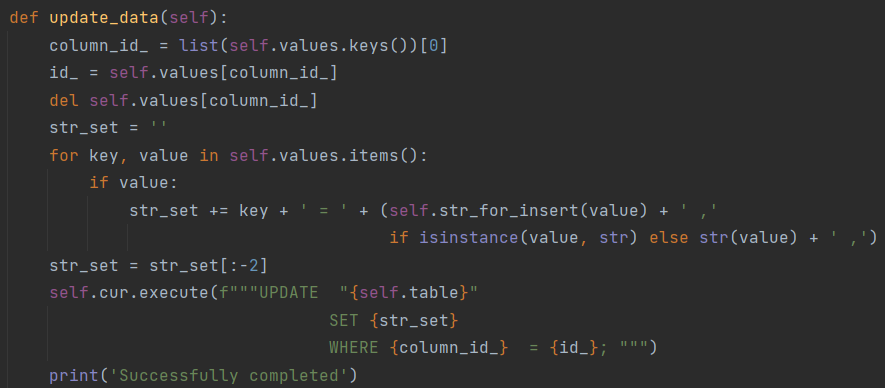
**Пошук даних у таблиці за необхідним полем**

****

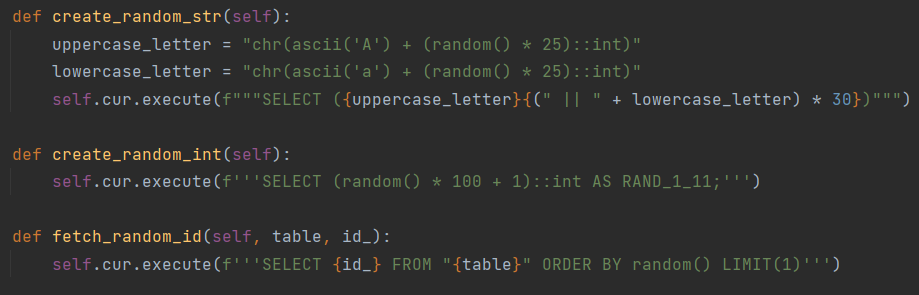
**Добавлення данних у таблицю**

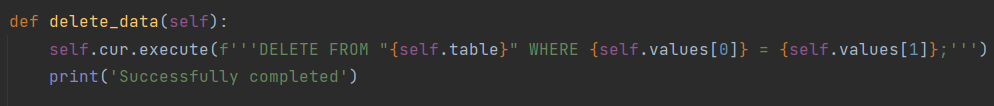
****

**Оновлення данних у таблиці**

****

**Додавання рандомних значень у таблицю**

** Видалення запису за таблиці**

****

**Код програми**

**main.py**

from controller import Controller  
  
def main():  
 dict\_table = {  
 'Bar': ['bar\_id', 'name', 'address'],  
 'Goods\_Client': ['goods\_id', 'name', 'client\_id'],  
 'Goods': ['goods\_id', 'name', 'price', 'bar\_id'],  
 'Client': ['client\_id', 'name']  
 }  
 dbname = 'Bar'  
 user = 'postgres'  
 password = '1234'  
 host = 'localhost'  
 port = '5432'  
 control = Controller(dict\_table, dbname, user, password, host, port)  
 control.menu()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**model.py**

import psycopg2  
from abc import abstractmethod  
from datetime import datetime  
  
  
class Model:  
 def \_\_init\_\_(self, database, user, password, host, port):  
 self.database = database  
 self.user = user  
 self.password = password  
 self.host = host  
 self.port = port  
 self.returns = None  
 self.cursor = None  
  
 def connect(self):  
 return psycopg2.connect(dbname=self.database, user=self.user, password=self.password,  
 host=self.host, port=self.port)  
  
 def execute(self, operation\_num, table=None, values=None, name\_columns=None):  
 with self.connect() as con:  
 try:  
 self.cursor = con.cursor()  
 task = Task(self.cursor, table, values, name\_columns)  
 self.returns = task.get\_task(operation\_num)  
 con.commit()  
 except (Exception, psycopg2.Error) as e:  
 print("Check you value", e)  
 finally:  
 if self.cursor:  
 self.cursor.close()  
 if self.returns:  
 return self.returns  
  
  
class Task:  
 def \_\_init\_\_(self, cursor, table, values, name\_columns):  
 self.dict\_foo = None  
 self.cur = cursor  
 self.table = table  
 self.values = values  
 self.name\_columns = name\_columns  
  
 def get\_task(self, task\_index):  
 self.dict\_foo = {  
 1: self.select\_table,  
 2: self.select\_all\_tables,  
 3: self.search,  
 4: self.insert\_data,  
 5: self.update\_data,  
 6: self.insert\_random,  
 7: self.delete\_data,  
 }  
 return self.dict\_foo[task\_index]()  
  
 def print\_fetchall(self):  
 tuple\_data = self.cur.fetchall()  
 if tuple\_data:  
 for columns in tuple\_data:  
 for column in columns:  
 print(column, end=' ')  
 print()  
 return tuple\_data  
 else:  
 print('Table is empty')  
  
 def create\_random\_str(self):  
 uppercase\_letter = "chr(ascii('A') + (random() \* 25)::int)"  
 lowercase\_letter = "chr(ascii('a') + (random() \* 25)::int)"  
 self.cur.execute(f"""SELECT ({uppercase\_letter}{(" || " + lowercase\_letter) \* 30})""")  
  
 def create\_random\_int(self):  
 self.cur.execute(f'''SELECT (random() \* 100 + 1)::int AS RAND\_1\_11;''')  
  
 def fetch\_random\_id(self, table, id\_):  
 self.cur.execute(f'''SELECT {id\_} FROM "{table}" ORDER BY random() LIMIT(1)''')  
  
 @staticmethod  
 def str\_for\_insert(our\_data):  
 return "'" + str(our\_data) + "'"  
  
 def select\_table(self):  
 self.cur.execute(f'''SELECT \* FROM "{self.table}";''')  
 print(self.table)  
 for column in self.name\_columns:  
 print(column, end=' ')  
 print()  
 self.print\_fetchall()  
  
 def select\_all\_tables(self):  
 for table\_name in range (len(self.values)):  
 print(' ', self.values[table\_name])  
 for column in self.name\_columns[table\_name]:  
 print(column, end=' ')  
 print()  
 self.cur.execute(f'''SELECT \* FROM "{self.values[table\_name]}";''')  
 self.print\_fetchall()  
  
 def search(self):  
 self.cur.execute(f'''SELECT {self.values[0]} FROM "{self.table}"   
 WHERE {self.values[1]}::VARCHAR LIKE '%{self.values[2]}%';''')  
 self.print\_fetchall()  
  
 def insert\_data(self):  
 str\_keys = ''  
 str\_value = ''  
 for key, value in self.values.items():  
 str\_keys += key + ' ,'  
 str\_value += self.str\_for\_insert(value) + ' ,' if isinstance(value, str) else str(value) + ', '  
 str\_keys = str\_keys[:-2]  
 str\_value = str\_value[:-2]  
 self.cur.execute(f"""INSERT INTO "{self.table}" ({str\_keys})  
 VALUES ({str\_value}); """)  
 print('Successfully completed')  
  
 def update\_data(self):  
 column\_id\_ = list(self.values.keys())[0]  
 id\_ = self.values[column\_id\_]  
 del self.values[column\_id\_]  
 str\_set = ''  
 for key, value in self.values.items():  
 if value:  
 str\_set += key + ' = ' + (self.str\_for\_insert(value) + ' ,'  
 if isinstance(value, str) else str(value) + ' ,')  
 str\_set = str\_set[:-2]  
 self.cur.execute(f"""UPDATE "{self.table}"  
 SET {str\_set}  
 WHERE {column\_id\_} = {id\_}; """)  
 print('Successfully completed')  
  
 def insert\_random(self):  
 id\_ = 1 if self.table != 'Goods\_Client' else 0  
 self.values = {}  
 self.cur.execute(f'''SELECT column\_name, data\_type FROM information\_schema.columns   
 WHERE table\_name = '{self.table}';''')  
 tuple\_data = [self.cur.fetchall()]  
 self.cur.execute(f'''SELECT  
 tc.table\_schema,   
 tc.constraint\_name,   
 tc.table\_name,   
 kcu.column\_name,   
 ccu.table\_schema AS foreign\_table\_schema,  
 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 (select row\_number() over (partition by table\_schema, table\_name,   
 constraint\_name order by row\_num) ordinal\_position,  
 table\_schema, table\_name, column\_name, constraint\_name  
 from (select row\_number() over (order by 1) row\_num, table\_schema,   
 table\_name, column\_name, constraint\_name  
 from information\_schema.constraint\_column\_usage  
 ) t  
 ) AS ccu  
 ON ccu.constraint\_name = tc.constraint\_name  
 AND ccu.table\_schema = tc.table\_schema  
 AND ccu.ordinal\_position = kcu.ordinal\_position  
 WHERE tc.constraint\_type = 'FOREIGN KEY' AND tc.table\_name = '{self.table}';''')  
 tuple\_data.append(self.cur.fetchall())  
 for list\_data in tuple\_data[0][id\_:]:  
 for data\_column in tuple\_data[1]:  
 if list\_data[0] == data\_column[3]:  
 self.fetch\_random\_id(data\_column[5], list\_data[0])  
 break  
 else:  
 if list\_data[1] == 'integer' or list\_data[1] == 'numeric':  
 self.create\_random\_int()  
 elif list\_data[1] == 'date':  
 value = datetime.now().strftime('%Y-%m-%d')  
 self.values[list\_data[0]] = value  
 continue  
 else:  
 self.create\_random\_str()  
 self.values[list\_data[0]] = self.cur.fetchall()[0][0]  
 return self.values  
  
 def delete\_data(self):  
 self.cur.execute(f'''DELETE FROM "{self.table}" WHERE {self.values[0]} = {self.values[1]};''')  
 print('Successfully completed')

**view.py**

from datetime import datetime  
from pydantic import ValidationError  
  
  
class View:  
 def \_\_init\_\_(self):  
 self.tables = {  
 1: 'Bar',  
 2: 'Goods\_Client',  
 3: 'Goods',  
 4: 'Client',  
 5: 'Return in operations\_menu',  
 }  
 self.sql\_operations = {  
 1: 'Show table',  
 2: 'Show all tables',  
 3: 'Search',  
 4: 'Add data',  
 5: 'Update data',  
 6: 'Add random packed data',  
 7: 'Delete data',  
 8: 'Exit the program',  
 }  
  
 def menu\_parser(self, menu, end):  
 while True:  
 for number, value in menu.items():  
 print(number, value)  
 value = input("Enter choice № ")  
 try:  
 value = int(value)  
 except Exception:  
 self.menu\_parser(menu, end)  
 else:  
 if 1 <= value <= end:  
 return value  
 else:  
 print(f"Enter the number from 1 to {end}")  
 self.menu\_parser(menu, end)  
  
 def operations\_menu(self):  
 value = self.menu\_parser(self.sql\_operations, 8)  
 return value  
  
 def tables\_menu(self):  
 value = self.menu\_parser(self.tables, 5)  
 if value == 5:  
 return 5  
 return self.tables[value]  
  
 def search(self, columns):  
 values = ['']  
 for count in range(len(columns)):  
 values[0] += columns[count] + ', '  
 print(count + 1, columns[count])  
 values.append(int(input('Enter number')))  
 if 0 < values[1] <= len(columns):  
 values[0] = values[0][:-2]  
 values[1] = columns[values[1] - 1]  
 values.append(input('write what you want search'))  
 return values  
 print('write correct')  
 return self.search(columns)  
  
 @staticmethod  
 def create\_data(columns, id\_):  
 values = {}  
 for column in columns[id\_:]:  
 if column == 'data':  
 value = datetime.now().strftime('%Y-%m-%d')  
 values[column] = value  
 continue  
 print('Write value in column', column)  
 value = input()  
 values[column] = value  
 return values  
  
 def add\_random(self):  
 value = input('How many data you wanna random add')  
 try:  
 value = int(value)  
 except Exception:  
 self.add\_random()  
 else:  
 if 1 <= value:  
 return value  
 else:  
 print("Enter the number from 1")  
 self.add\_random()  
  
 def delete\_id(self):  
 value = input('What data you wanna delete')  
 try:  
 value = int(value)  
 except Exception:  
 self.add\_random()  
 else:  
 if 1 <= value:  
 return value  
 else:  
 print("Enter the number from 1")  
 self.add\_random()

**controller.py**

from model import Model  
from view import View  
  
  
class Controller:  
 def \_\_init\_\_(self, dict\_table, dbname, user, password, host, port):  
 self.view\_obj = View()  
 self.model\_obj = Model(dbname, user, password, host, port)  
 self.dict\_table = dict\_table  
  
 def menu\_table(self, operation\_num):  
 table\_name = self.view\_obj.tables\_menu()  
 if isinstance(table\_name, str):  
 if operation\_num == 1:  
 self.model\_obj.execute(operation\_num, table=table\_name,  
 name\_columns=list(self.dict\_table[table\_name]))  
 elif operation\_num == 3:  
 data = self.view\_obj.search(self.dict\_table[table\_name])  
 self.model\_obj.execute(operation\_num, table\_name, data)  
 elif operation\_num == 4:  
 id\_ = 0 if table\_name == 5 else 1  
 data = self.view\_obj.create\_data(self.dict\_table[table\_name], id\_=id\_)  
 self.model\_obj.execute(operation\_num, table\_name, data)  
 elif operation\_num == 5:  
 data = self.view\_obj.create\_data(self.dict\_table[table\_name], id\_=0)  
 self.model\_obj.execute(operation\_num, table\_name, data)  
 elif operation\_num == 6:  
 number = self.view\_obj.add\_random()  
 for \_ in range(number):  
 data = self.model\_obj.execute(operation\_num, table=table\_name)  
 self.model\_obj.execute(4, table\_name, data)  
 else:  
 id\_ = self.view\_obj.delete\_id()  
 values = [self.dict\_table[table\_name][0], id\_]  
 self.model\_obj.execute(operation\_num, table\_name, values)  
  
 def menu(self):  
 while True:  
 operation\_num = self.view\_obj.operations\_menu()  
 if operation\_num == 2:  
 self.model\_obj.execute(operation\_num, values=list(self.dict\_table.keys()),  
 name\_columns=list(self.dict\_table.values()))  
 elif operation\_num != 8:  
 self.menu\_table(operation\_num)  
 else:  
 exit()