

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-01

Проводов Д. 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 «Магазин»**

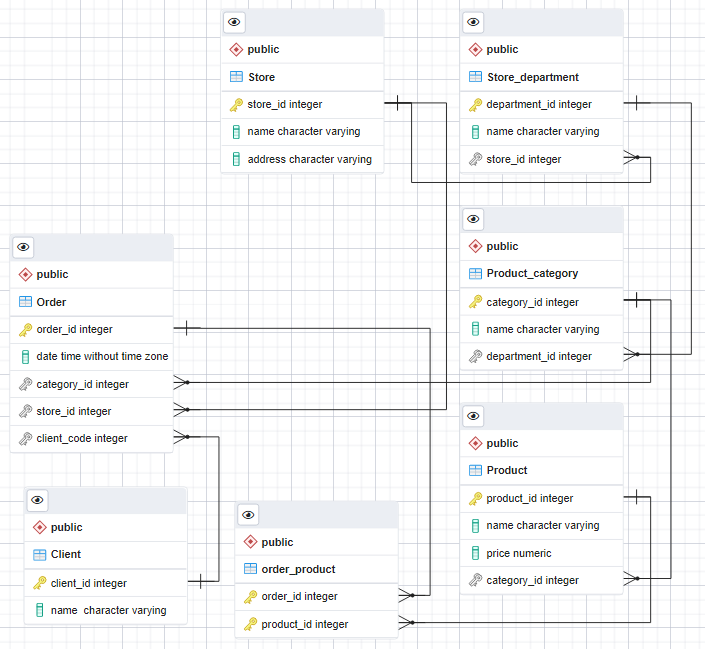


Рисунок 1. Схема бази даних, побудовано у [p](https://app.diagrams.net/)gAdmin 4.

**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.

**Структура програми та її опис**

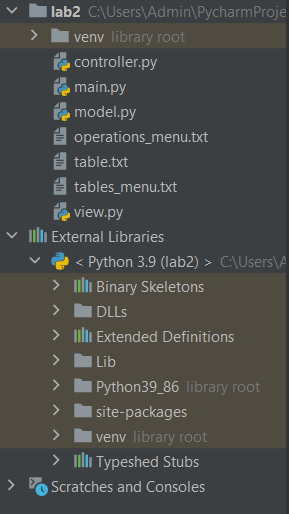


Рисунок 2. Структура програми.

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.

У файлах view.py описано клас View, що виводить результати виконання тієї чи іншої дії.

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**

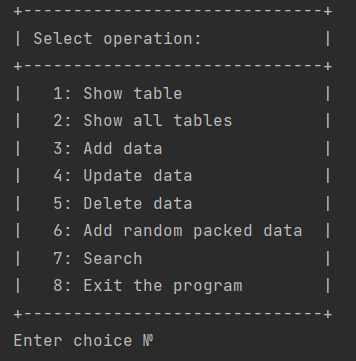


Рисунок 3. Меню для вибору операції над БД.

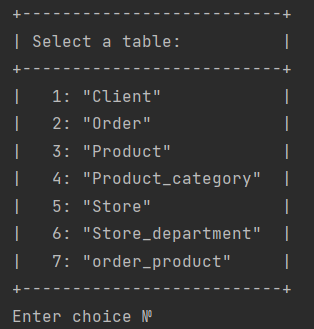


Рисунок 4. Меню для вибору таблиці, над якою проводитиметься обрана операція.

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

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

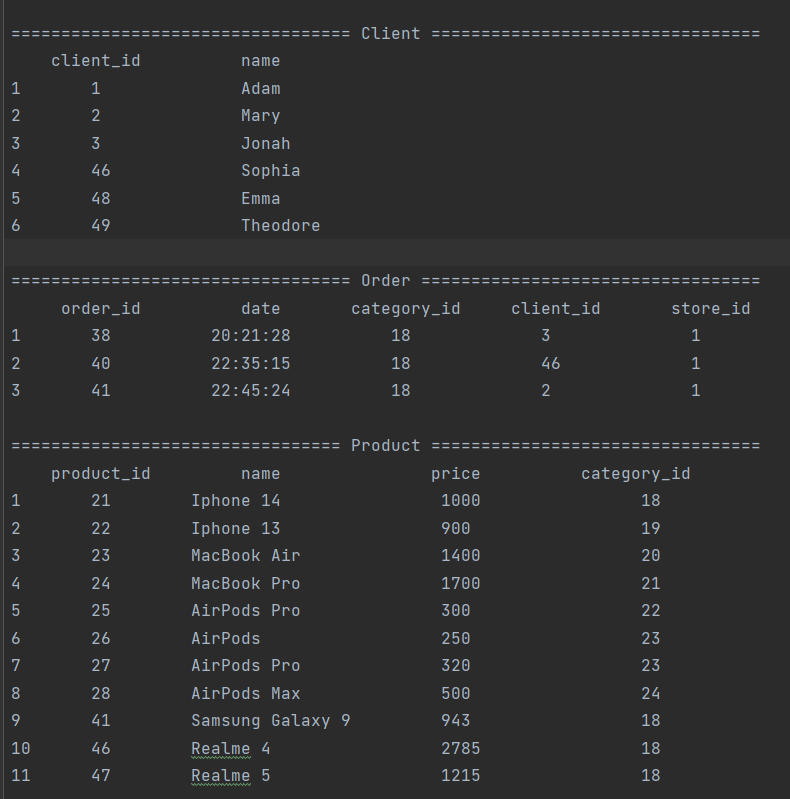
**Меню таблиць складається з 7 пунктів (Рисунок 3)**

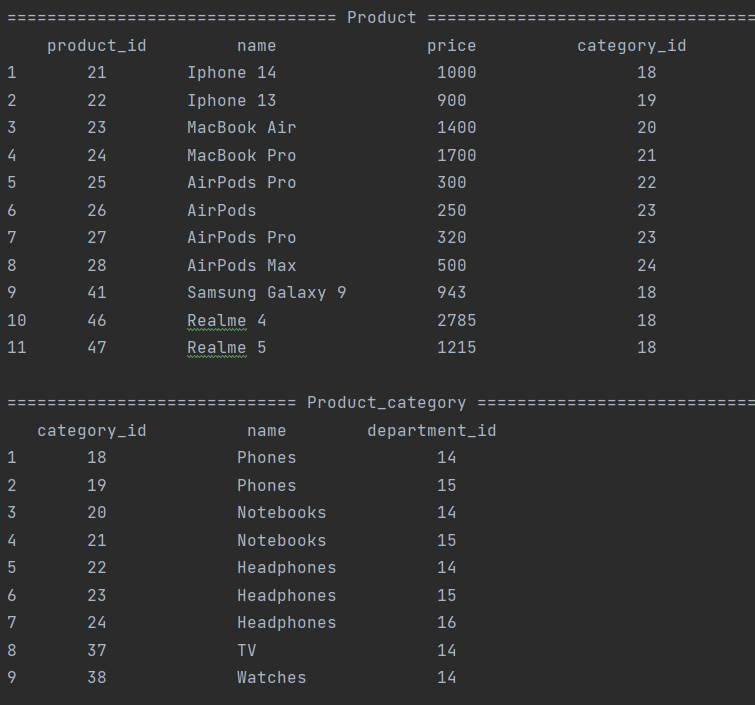
1. Виконати обрану операцію над таблицею “Client”.
2. Виконати обрану операцію над таблицею “Order”.
3. Виконати обрану операцію над таблицею “Product”.
4. Виконати обрану операцію над таблицею “Product\_category”.
5. Виконати обрану операцію над таблицею “Store”.
6. Виконати обрану операцію над таблицею “Store\_department”
7. Виконати обрану операцію над таблицею “order\_product”

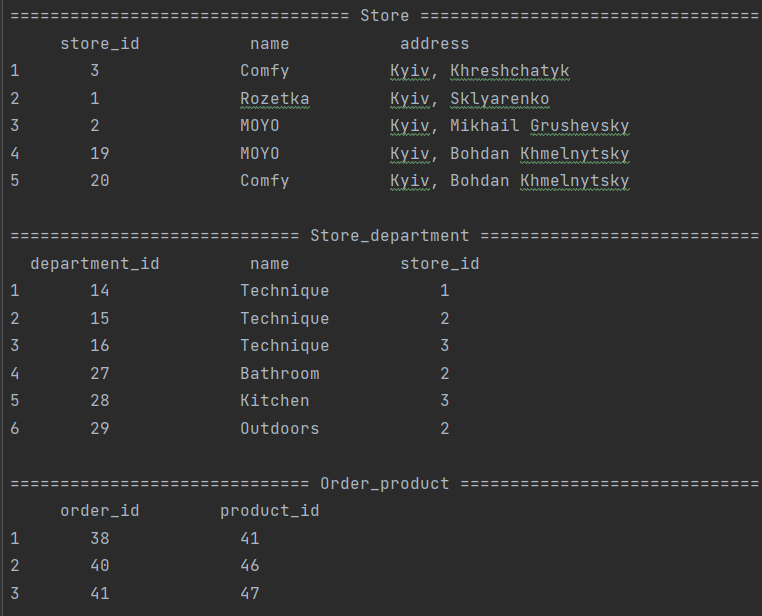
Слід також зауважити, що не всі операції потребують введення номеру таблиці, тому, при виборі операції номер 1, 6 та 8 меню таблиць не виводиться.

**Результати та виконання операцій**

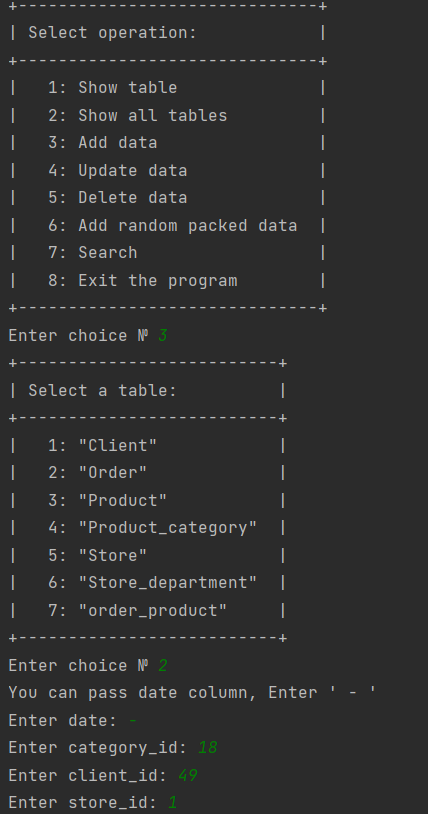
**Результат операції Show (для всіх таблиць)**

****

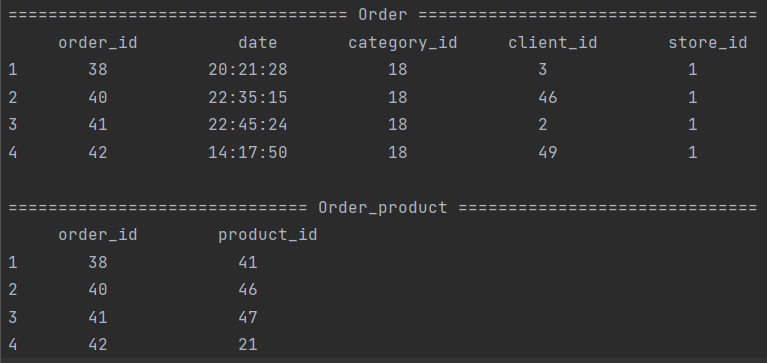
****

****

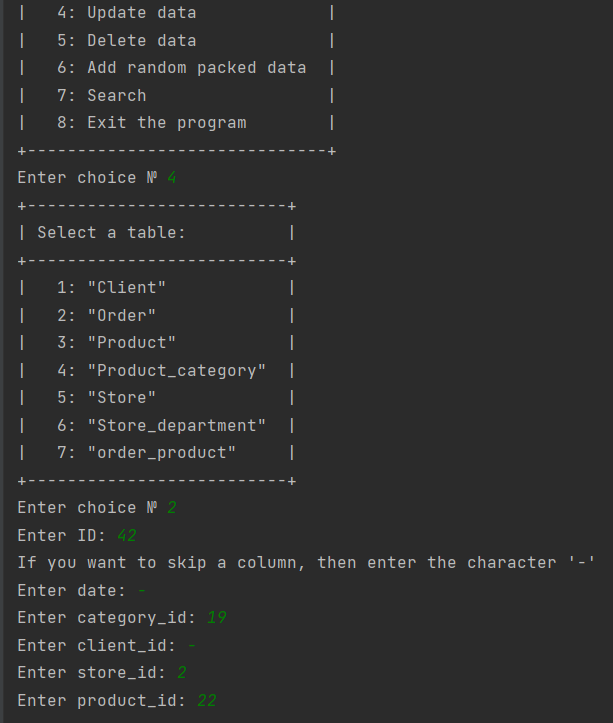
**Операція Add для таблиці “Order”**

****

**Результат (дані автоматично додались до зв’язаної таблиці Order\_product)**

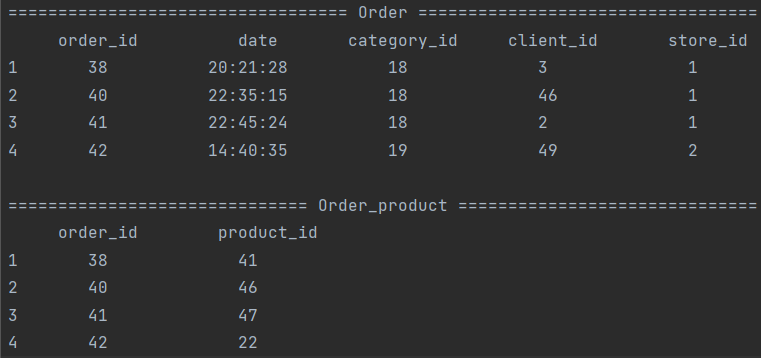
****

**Оновлення даних у таблиці “Order”**

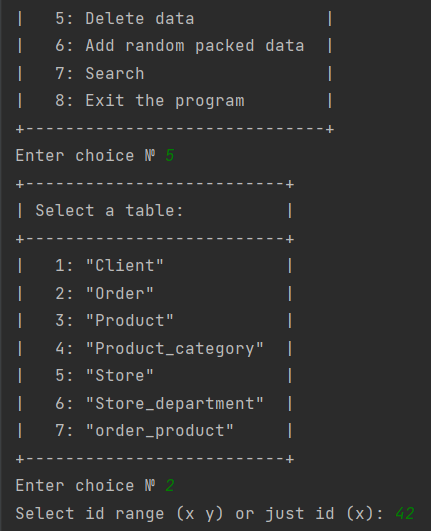
****

Клієнта залишаємо того ж самого, а час нового замовлення автоматично оновиться.

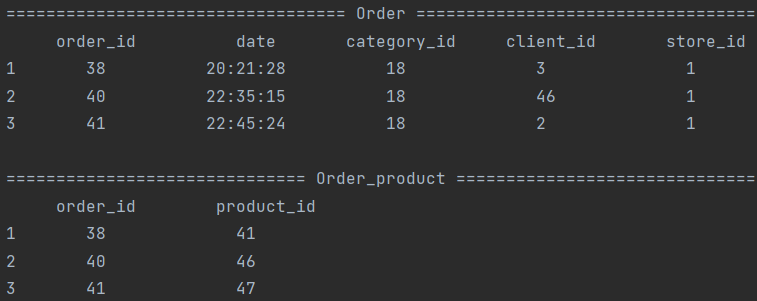
**Результат (дані автоматично оновились у зв’язаній таблиці Order\_product)**

****

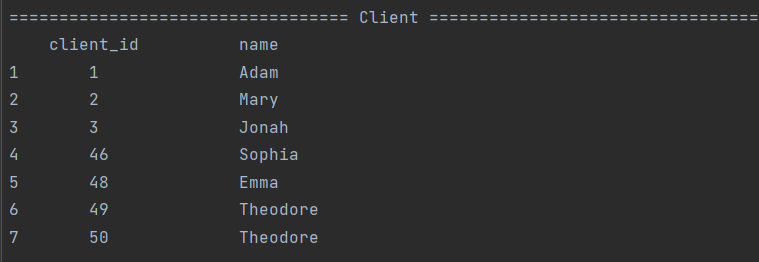
**Видалення замовлення з order\_id = 42**

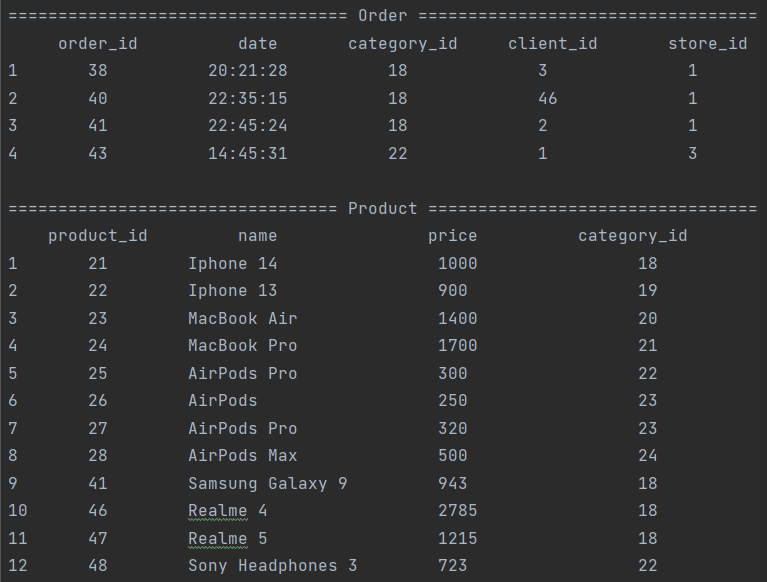
****

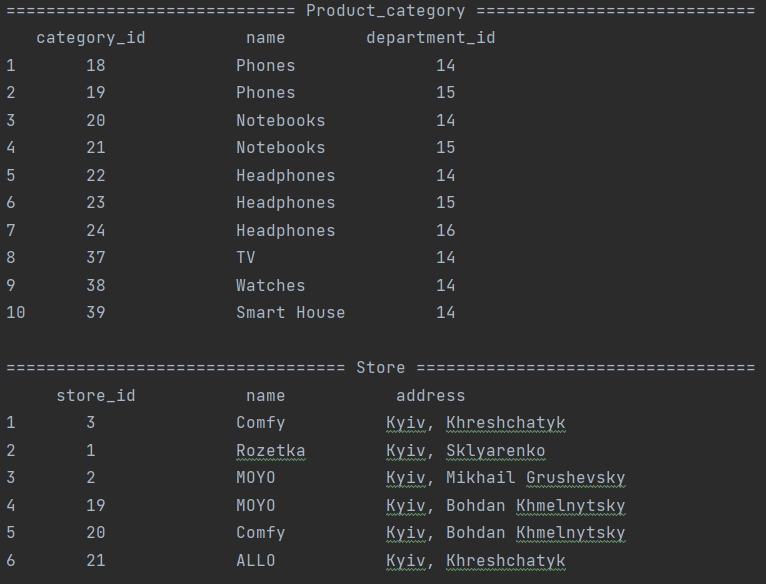
**Результат (дані автоматично видалено у зв’язаній таблиці Order\_product)**

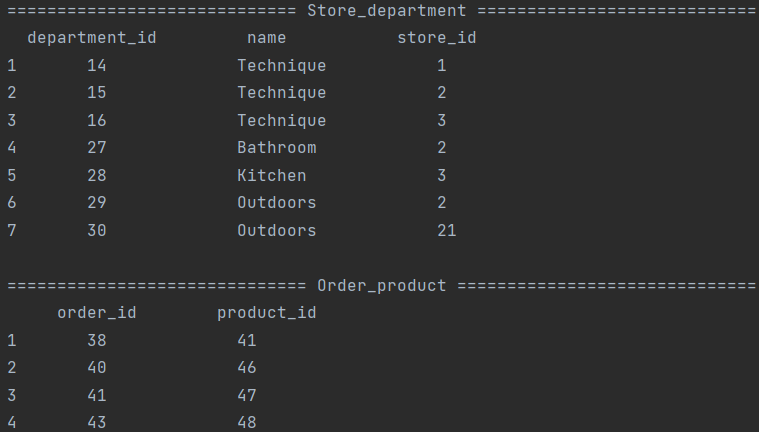
****

**Додавання випадкових даних до кожної таблиці (останні рядки)**

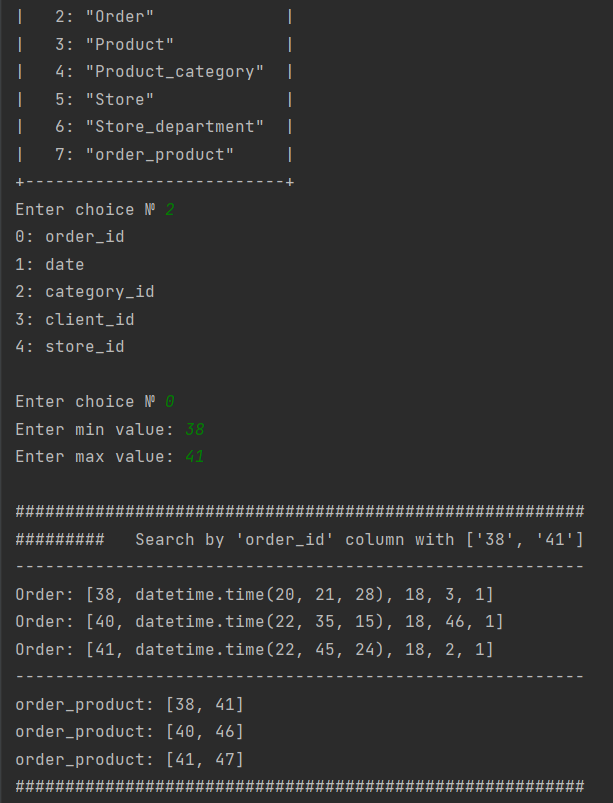
****

****

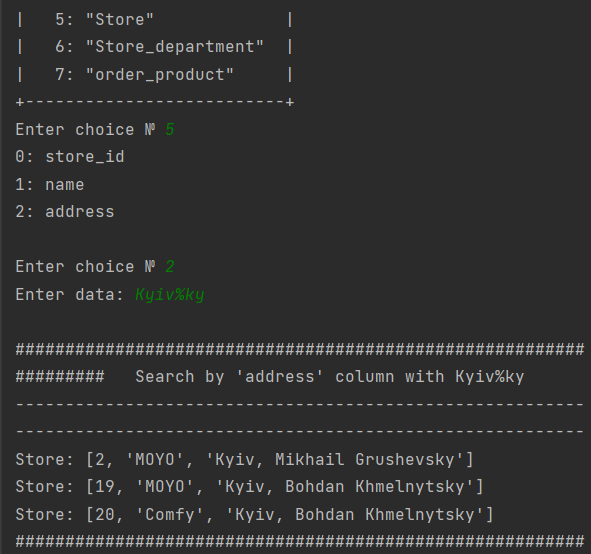
****

****

**Пошук та результати за параметром order\_id = [38, 41]**

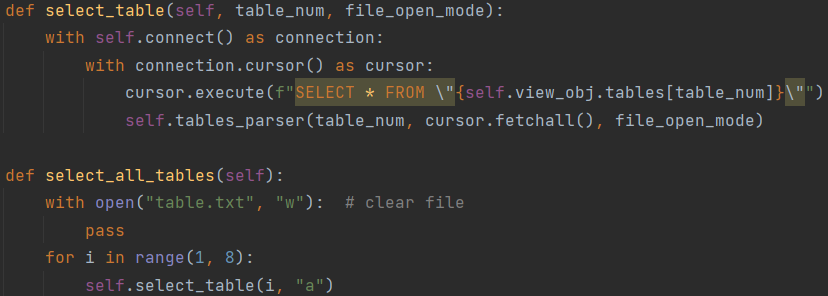
****

**Пошук за параметром Kyiv%ky**

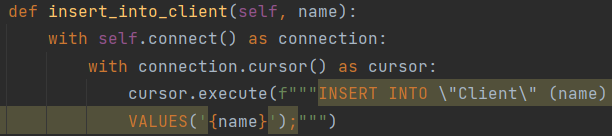
****

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

**Вибір даних з таблиць, та передання їх до контролеру парсерів, де буде викликана відповідна функція з модуля view.py та дані будуть записані у файл table.txt.**

****

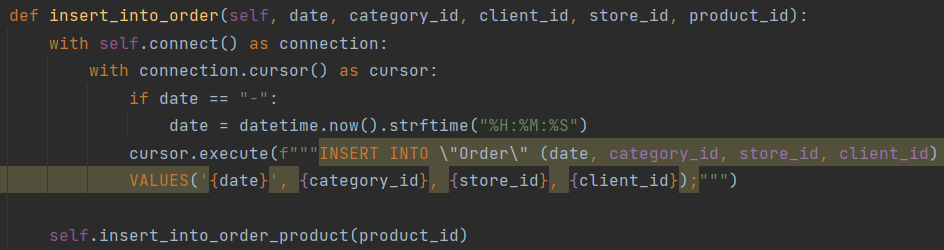
Кожна таблиця має окрему функцію для внесення даних.

****

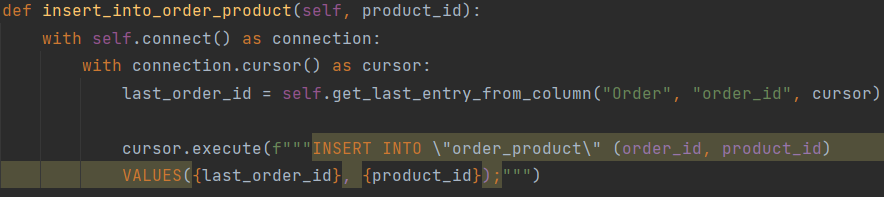
Але, у деяких з них є певні умови.

До таблиці “Order” можна не передавати час замовлення, тоді його буде обраховано автоматично..

Також, після внесення до БД нового замовлення, треба занести дані й до таблиці “Order\_product” (останній рядок).

****

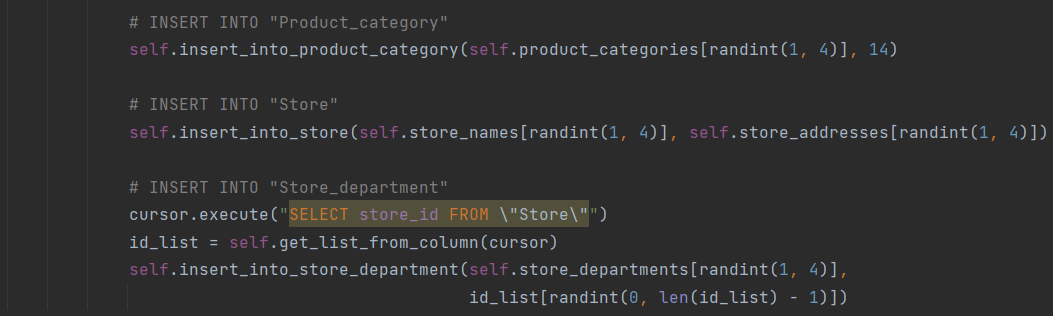
**Сама функція внесення даних до “Order\_product” виглядає наступним чином.**

****

Отримуємо останній order\_id та записуємо його в нашу таблицю.

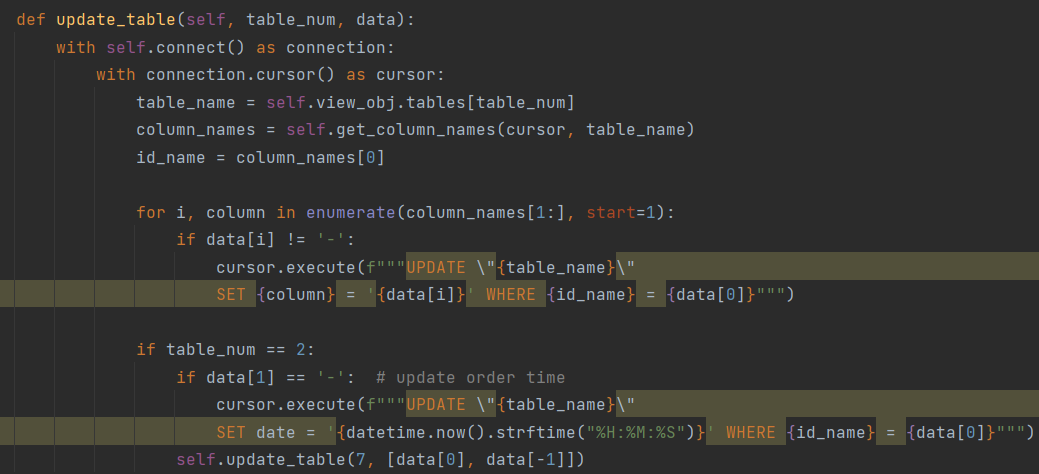
**Вставка пакетних «рандомізованих» даних має таку реалізацію.**

****

****

Більшість цих даних береться із словників, які було заповнено деякими значеннями.

**Оновлення даних рядка таблиці.**

****

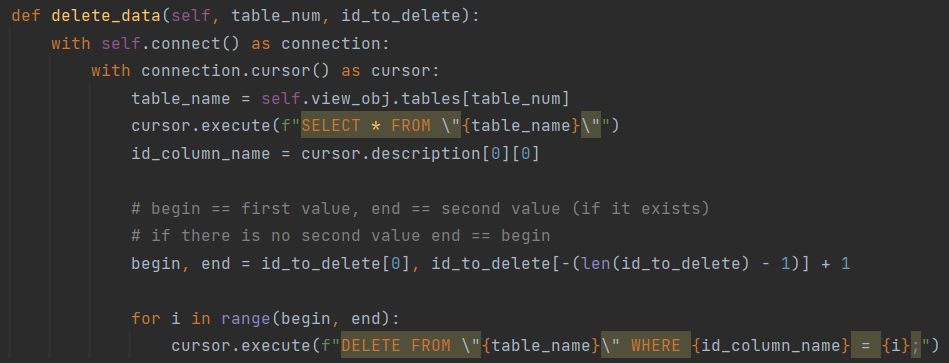
data – це список введених користувачем оновлених даних.

data[0] – значення обраного id рядку для оновлення.

У циклі ми проходимось по всіх колонках та записуємо нові дані.

Далі у if перевіряємо чи наша таблиця не є “Order”, якщо це вона, то ми викликаємо оновлення даних таблиці “Order\_product”, де [data[0], data[-1]] це [new\_order\_id, product\_id].

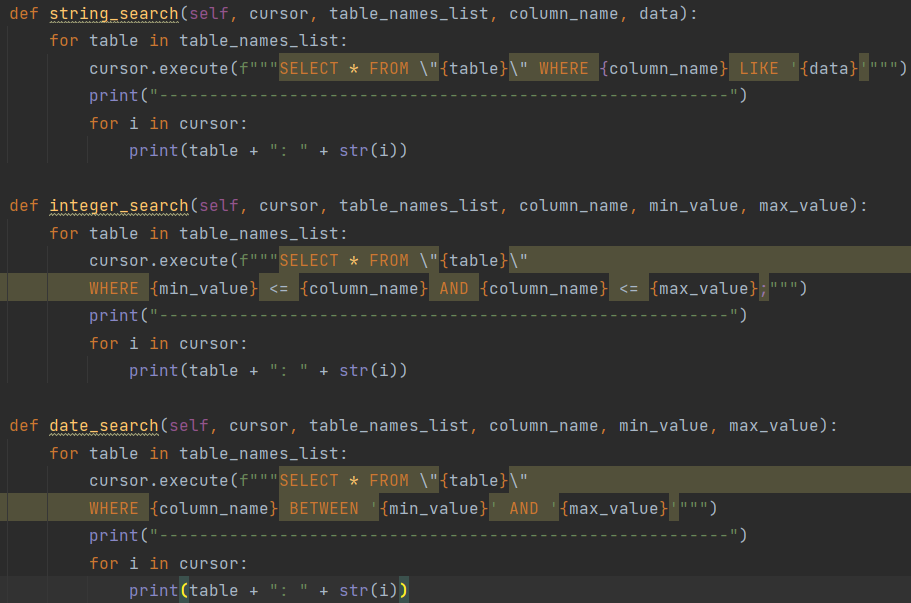
**Видалення даних з таблиці**

****

При зчитуванні id рядку для видалення, я реалізував можливість видалення одразу великої їх кількості (у діапазоні від Х до У).

При цьому було оброблена ситуація, коли користувач випадково вводить більше 2 значень, їх не враховуємо, беруться тільки перші значення. Проте, можна видалити й тільки один рядок, тоді значення begin == end.

**Пошук для рядкових, числових та часових форматів.**



**Код програми**

**main.py**

from controller import Controller  
  
  
def main():  
 obj = Controller()  
 obj.menu()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**model.py**

from view import View  
  
from random import choice  
from datetime import datetime  
  
import psycopg2  
import psycopg2.extras  
from psycopg2.extras import DictCursor  
  
  
class Model:  
 def \_\_init\_\_(self):  
 self.view\_obj = View()  
  
 def connect(self):  
 return psycopg2.connect(dbname='lab1', user='postgres', password='Sqlpr0v0d0k', host='localhost')  
  
 def generate\_random\_data(self, cursor, length):  
 uppercase\_letter = "chr(ascii('A') + (random() \* 25)::int)"  
 lowercase\_letter = "chr(ascii('a') + (random() \* 25)::int)"  
 cursor.execute(f"""SELECT ({uppercase\_letter}{(" || " + lowercase\_letter) \* (length - 1)})""")  
  
 return cursor.fetchone()[0]  
  
 def get\_last\_entry\_from\_column(self, table\_name, column\_name, cursor):  
 cursor.execute(f"""SELECT {column\_name} FROM \"{table\_name}\"  
 ORDER BY {column\_name} DESC LIMIT 1""")  
 return cursor.fetchone()[0]  
  
 def get\_column\_names(self, cursor, table\_name):  
 cursor.execute(f"SELECT \* FROM \"{table\_name}\"")  
 return [item[0] for item in cursor.description]  
  
 def get\_list\_from\_column(self, cursor):  
 # converting [(item, ), (item, ), ...] to [item, item, ...]  
 return [item[0] for item in cursor.fetchall()]  
  
 def get\_column\_names\_to\_view(self, table\_num):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 table\_name = self.view\_obj.tables[table\_num]  
 return self.get\_column\_names(cursor, table\_name)  
  
 def insert\_data\_controller(self, table\_num):  
 if table\_num == 1:  
 data = self.view\_obj.input\_client()  
 self.insert\_into\_client(\*data)  
 elif table\_num == 2:  
 print("You can pass date column, Enter ' - '")  
 data = self.view\_obj.input\_order()  
 self.insert\_into\_order(\*data)  
 elif table\_num == 3:  
 data = self.view\_obj.input\_product()  
 self.insert\_into\_product(\*data)  
 elif table\_num == 4:  
 data = self.view\_obj.input\_product\_category()  
 self.insert\_into\_product\_category(\*data)  
 elif table\_num == 5:  
 data = self.view\_obj.input\_store()  
 self.insert\_into\_store(\*data)  
 elif table\_num == 6:  
 data = self.view\_obj.input\_department()  
 self.insert\_into\_store\_department(\*data)  
 elif table\_num == 7:  
 print("You cannot insert data into this table")  
  
 def tables\_parser\_controller(self, table\_num, data, mode):  
 with open("table.txt", mode) as file:  
 if table\_num == 1:  
 self.view\_obj.client\_parser(data, file)  
 elif table\_num == 2:  
 self.view\_obj.order\_parser(data, file)  
 elif table\_num == 3:  
 self.view\_obj.product\_parser(data, file)  
 elif table\_num == 4:  
 self.view\_obj.product\_category\_parser(data, file)  
 elif table\_num == 5:  
 self.view\_obj.store\_parser(data, file)  
 elif table\_num == 6:  
 self.view\_obj.store\_department\_parser(data, file)  
 elif table\_num == 7:  
 self.view\_obj.order\_product\_parser(data, file)  
  
 def select\_table(self, table\_num, file\_open\_mode):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 cursor.execute(f"SELECT \* FROM \"{self.view\_obj.tables[table\_num]}\"")  
 self.tables\_parser\_controller(table\_num, cursor.fetchall(), file\_open\_mode)  
  
 def select\_all\_tables(self):  
 with open("table.txt", "w"): # clear file  
 pass  
 for i in range(1, 8):  
 self.select\_table(i, "a")  
  
 def insert\_into\_client(self, name):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 cursor.execute(f"""INSERT INTO \"Client\" (name)  
 VALUES('{name}');""")  
  
 def insert\_into\_order(self, date, category\_id, client\_id, store\_id, product\_id):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 if date == "-":  
 date = datetime.now().strftime("%H:%M:%S")  
 cursor.execute(f"""INSERT INTO \"Order\" (date, category\_id, store\_id, client\_id)  
 VALUES('{date}', {category\_id}, {store\_id}, {client\_id});""")  
  
 self.insert\_into\_order\_product(product\_id)  
  
 def insert\_into\_order\_product(self, product\_id):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 last\_order\_id = self.get\_last\_entry\_from\_column("Order", "order\_id", cursor)  
  
 cursor.execute(f"""INSERT INTO \"order\_product\" (order\_id, product\_id)  
 VALUES({last\_order\_id}, {product\_id});""")  
  
 def insert\_into\_product(self, name, price, category\_id):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 cursor.execute(f"""INSERT INTO \"Product\" (name, price, category\_id)  
 VALUES('{name}', {price}, {category\_id});""")  
  
 def insert\_into\_product\_category(self, name, department\_id):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 cursor.execute(f"""INSERT INTO \"Product\_category\" (name, department\_id)  
 VALUES('{name}', {department\_id});""")  
  
 def insert\_into\_store(self, name, address):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 cursor.execute(f"""INSERT INTO \"Store\" (name, address)  
 VALUES('{name}', '{address}');""")  
  
 def insert\_into\_store\_department(self, name, store\_id):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 cursor.execute(f"""INSERT INTO \"Store\_department\" (name, store\_id)  
 VALUES('{name}', {store\_id})""")  
  
 def insert\_random\_data\_packet(self):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 # INSERT INTO "Client"  
 self.insert\_into\_client(self.generate\_random\_data(cursor, 5))  
  
 # INSERT INTO "Product"  
 product\_name = self.generate\_random\_data(cursor, 7)  
 cursor.execute(f"""SELECT category\_id FROM \"Product\_category\"""")  
 category = choice(self.get\_list\_from\_column(cursor))  
 self.insert\_into\_product(product\_name, '(random() \* 3000)::int', category)  
  
 # INSERT INTO "Order" and "order\_product"  
 cursor.execute(f"""SELECT client\_id FROM \"Client\"""")  
 client\_id = choice(self.get\_list\_from\_column(cursor))  
 new\_product\_id = self.get\_last\_entry\_from\_column("Product", "product\_id", cursor)  
 self.insert\_into\_order('-', category, client\_id, 1, new\_product\_id)  
  
 # INSERT INTO "Product\_category"  
 self.insert\_into\_product\_category(self.generate\_random\_data(cursor, 7), 14)  
  
 # INSERT INTO "Store"  
 store\_name = self.generate\_random\_data(cursor, 5)  
 city = self.generate\_random\_data(cursor, 5)  
 street = self.generate\_random\_data(cursor, 8)  
 self.insert\_into\_store(store\_name, city + ", " + street)  
  
 # INSERT INTO "Store\_department"  
 cursor.execute("SELECT store\_id FROM \"Store\"")  
 store\_id = choice(self.get\_list\_from\_column(cursor))  
 self.insert\_into\_store\_department(self.generate\_random\_data(cursor, 5), store\_id)  
  
 def update\_table(self, table\_num, data):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 table\_name = self.view\_obj.tables[table\_num]  
 column\_names = self.get\_column\_names(cursor, table\_name)  
 id\_name = column\_names[0]  
  
 for i, column in enumerate(column\_names[1:], start=1):  
 if data[i] != '-':  
 cursor.execute(f"""UPDATE \"{table\_name}\"  
 SET {column} = '{data[i]}' WHERE {id\_name} = {data[0]}""")  
  
 if table\_num == 2:  
 if data[1] == '-': # update order time  
 cursor.execute(f"""UPDATE \"{table\_name}\"  
 SET date = '{datetime.now().strftime("%H:%M:%S")}' WHERE {id\_name} = {data[0]}""")  
 self.update\_table(7, [data[0], data[-1]])  
  
 def delete\_data(self, table\_num, id\_to\_delete):  
 with self.connect() as connection:  
 with connection.cursor() as cursor:  
 table\_name = self.view\_obj.tables[table\_num]  
 cursor.execute(f"SELECT \* FROM \"{table\_name}\"")  
 id\_column\_name = cursor.description[0][0]  
  
 # begin == first value, end == second value (if it exists)  
 # if there is no second value end == begin  
 begin, end = id\_to\_delete[0], id\_to\_delete[-(len(id\_to\_delete) - 1)] + 1  
  
 for i in range(begin, end):  
 cursor.execute(f"DELETE FROM \"{table\_name}\" WHERE {id\_column\_name} = {i};")  
  
 def string\_search(self, cursor, table\_names\_list, column\_name, data):  
 for table in table\_names\_list:  
 cursor.execute(f"""SELECT \* FROM \"{table}\" WHERE {column\_name} LIKE '{data}'""")  
 print("---------------------------------------------------------")  
 for i in cursor:  
 print(table + ": " + str(i))  
  
 def integer\_search(self, cursor, table\_names\_list, column\_name, min\_value, max\_value):  
 for table in table\_names\_list:  
 cursor.execute(f"""SELECT \* FROM \"{table}\"  
 WHERE {min\_value} <= {column\_name} AND {column\_name} <= {max\_value};""")  
 print("---------------------------------------------------------")  
 for i in cursor:  
 print(table + ": " + str(i))  
  
 def date\_search(self, cursor, table\_names\_list, column\_name, min\_value, max\_value):  
 for table in table\_names\_list:  
 cursor.execute(f"""SELECT \* FROM \"{table}\"  
 WHERE {column\_name} BETWEEN '{min\_value}' AND '{max\_value}'""")  
 print("---------------------------------------------------------")  
 for i in cursor:  
 print(table + ": " + str(i))  
  
 def search\_controller(self, cursor, table\_names\_list, column\_name, data\_type, data):  
 print("\n#########################################################")  
 print(f"#########\tSearch by '{column\_name}' column with {data}")  
 if data\_type == 'integer':  
 self.integer\_search(cursor, table\_names\_list, column\_name, \*data)  
 elif data\_type == 'character varying':  
 self.string\_search(cursor, table\_names\_list, column\_name, data)  
 elif data\_type == 'time without time zone':  
 self.date\_search(cursor, table\_names\_list, column\_name, \*data)  
 print("#########################################################\n\n")  
  
  
 def search\_into\_table(self, table\_num, search\_param):  
 with self.connect() as connection:  
 with connection.cursor(cursor\_factory=DictCursor) as cursor:  
 table\_name = self.view\_obj.tables[table\_num]  
 column\_names = self.get\_column\_names(cursor, table\_name)  
 column = column\_names[search\_param]  
  
 cursor.execute(f"""SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.COLUMNS   
 WHERE COLUMN\_NAME = '{column}'""")  
 table\_names\_list = self.get\_list\_from\_column(cursor)  
  
 cursor.execute(f"""SELECT \* FROM INFORMATION\_SCHEMA.COLUMNS  
 WHERE TABLE\_NAME = '{table\_name}' AND COLUMN\_NAME = '{column}'""")  
 data\_type = cursor.fetchone()['data\_type']  
  
 data\_to\_search = self.view\_obj.input\_data\_to\_search(data\_type)  
 self.search\_controller(cursor, table\_names\_list, column, data\_type, data\_to\_search)

**view.py**

import psycopg2  
  
  
class View:  
 def \_\_init\_\_(self):  
 self.tables = {  
 1: 'Client',  
 2: 'Order',  
 3: 'Product',  
 4: 'Product\_category',  
 5: 'Store',  
 6: 'Store\_department',  
 7: 'order\_product',  
 8: 'Exit the program',  
 }  
  
 def menu\_parser(self, func, file\_path, end):  
 with open(file\_path, "r") as file:  
 print(file.read())  
 param = int(input("Enter choice № "))  
  
 if param < 1 or param > end:  
 print(f"Enter the number from 1 to {end} <--------- warning")  
 return func()  
 else:  
 return param  
  
 def operations\_menu(self):  
 return self.menu\_parser(self.operations\_menu, "operations\_menu.txt", 8)  
  
 def tables\_menu(self):  
 return self.menu\_parser(self.tables\_menu, "tables\_menu.txt", 7)  
  
 def client\_parser(self, data, file):  
 file.write("\n" + " Client ".center(75, "=") + "\n")  
 file.write("client\_id".center(17, " ") + "name".center(15, " ") + "\n")  
  
 for i, item in enumerate(data, start=1):  
 file.write(str(i).ljust(8, " ")  
 + str(item[0]).ljust(15, " ")  
 + str(item[1]) + "\n")  
  
 def order\_parser(self, data, file):  
 file.write("\n" + " Order ".center(75, "=") + "\n")  
 file.write("order\_id".center(17, " ")  
 + "date".center(15, " ")  
 + "category\_id".center(15, " ")  
 + "client\_id".center(15, " ")  
 + "store\_id".center(15, " ") + "\n")  
  
 for i, item in enumerate(data, start=1):  
 file.write(str(i).ljust(8, " ")  
 + str(item[0]).ljust(12, " ")  
 + str(item[1]).ljust(18, " ")  
 + str(item[2]).ljust(15, " ")  
 + str(item[3]).ljust(15, " ")  
 + str(item[4]) + "\n")  
  
 def product\_parser(self, data, file):  
 file.write("\n" + " Product ".center(75, "=") + "\n")  
 file.write("product\_id".center(17, " ")  
 + "name".center(15, " ")  
 + "price".center(25, " ")  
 + "category\_id".center(10, " ") + "\n")  
  
 for i, item in enumerate(data, start=1):  
 file.write(str(i).ljust(8, " ")  
 + str(item[0]).ljust(10, " ")  
 + str(item[1]).ljust(25, " ")  
 + str(item[2]).ljust(20, " ")  
 + str(item[3]) + "\n")  
  
 def product\_category\_parser(self, data, file):  
 file.write("\n" + " Product\_category ".center(75, "=") + "\n")  
 file.write("category\_id".center(17, " ")  
 + "name".center(17, " ")  
 + "department\_id".center(17, " ") + "\n")  
  
 for i, item in enumerate(data, start=1):  
 file.write(str(i).ljust(8, " ")  
 + str(item[0]).ljust(15, " ")  
 + str(item[1]).ljust(20, " ")  
 + str(item[2]) + "\n")  
  
 def store\_parser(self, data, file):  
 file.write("\n" + " Store ".center(75, "=") + "\n")  
 file.write("store\_id".center(17, " ")  
 + "name".center(17, " ")  
 + "address".center(17, " ") + "\n")  
  
 for i, item in enumerate(data, start=1):  
 file.write(str(i).ljust(8, " ")  
 + str(item[0]).ljust(15, " ")  
 + str(item[1]).ljust(15, " ")  
 + str(item[2]) + "\n")  
  
 def store\_department\_parser(self, data, file):  
 file.write("\n" + " Store\_department ".center(75, "=") + "\n")  
 file.write("department\_id".center(17, " ")  
 + "name".center(17, " ")  
 + "store\_id".center(17, " ") + "\n")  
  
 for i, item in enumerate(data, start=1):  
 file.write(str(i).ljust(8, " ")  
 + str(item[0]).ljust(15, " ")  
 + str(item[1]).ljust(20, " ")  
 + str(item[2]) + "\n")  
  
 def order\_product\_parser(self, data, file):  
 file.write("\n" + " Order\_product ".center(75, "=") + "\n")  
 file.write("order\_id".center(17, " ")  
 + "product\_id".center(17, " ") + "\n")  
  
 for i, item in enumerate(data, start=1):  
 file.write(str(i).ljust(8, " ")  
 + str(item[0]).ljust(15, " ")  
 + str(item[1]) + "\n")  
  
 def input\_client(self):  
 return input("Enter name: ")  
  
 def input\_order(self):  
 date = input("Enter date: ")  
 category\_id = input("Enter category\_id: ")  
 client\_id = input("Enter client\_id: ")  
 store\_id = input("Enter store\_id: ")  
 product\_id = input("Enter product\_id: ")  
 return [date, category\_id, client\_id, store\_id, product\_id]  
  
 def input\_product(self):  
 product\_name = input("Enter product\_name: ")  
 price = input("Enter price: ")  
 category\_id = input("Enter category\_id: ")  
 return [product\_name, price, category\_id]  
  
 def input\_product\_category(self):  
 category\_name = input("Enter category\_name: ")  
 department\_id = input("Enter department\_id: ")  
 return [category\_name, department\_id]  
  
 def input\_store(self):  
 store\_name = input("Enter store\_name: ")  
 address = input("Enter address: ")  
 return [store\_name, address]  
  
 def input\_department(self):  
 department\_name = input("Enter department\_name: ")  
 store\_id = input("Enter store\_id: ")  
 return [department\_name, store\_id]  
  
 def update\_data(self, table\_num):  
 updated\_data = {  
 1: self.input\_client,  
 2: self.input\_order,  
 3: self.input\_product,  
 4: self.input\_product\_category,  
 5: self.input\_store,  
 6: self.input\_department,  
 }  
 row\_id = input("Enter ID: ")  
 print("If you want to skip a column, then enter the character '-'")  
 return [row\_id, \*(updated\_data[table\_num]())]  
  
 def delete(self):  
 id\_range = input("Select id range (x y) or just id (x): ").split()  
 return [int(i) for i in id\_range]  
  
 def search(self, column\_names):  
 for i, column in enumerate(column\_names):  
 print(str(i) + ": " + column)  
  
 return int(input("\nEnter choice № "))  
  
 def input\_data\_to\_search(self, data\_type):  
 if data\_type == 'integer':  
 min\_value = input("Enter min value: ")  
 max\_value = input("Enter max value: ")  
 return [min\_value, max\_value]  
 elif data\_type == 'character varying':  
 return input("Enter data: ")  
 elif data\_type == 'time without time zone':  
 min\_value = input("Enter HH:MM:SS min date: ")  
 max\_value = input("Enter HH:MM:SS max date: ")  
 return [min\_value, max\_value]  
 else:  
 print(f"Cannot find data with '{data\_type}' type")

**controller.py**

from model import Model  
from view import View  
  
  
class Controller:  
 def \_\_init\_\_(self):  
 self.view\_obj = View()  
 self.model\_obj = Model()  
  
 def menu(self):  
 table\_num = 0  
 while True:  
 operation\_num = self.view\_obj.operations\_menu()  
 if operation\_num == 8:  
 break  
 if operation\_num != 2 and operation\_num != 6:  
 table\_num = self.view\_obj.tables\_menu()  
  
 if operation\_num == 1:  
 self.model\_obj.select\_table(table\_num, "w")  
 elif operation\_num == 2:  
 self.model\_obj.select\_all\_tables()  
 elif operation\_num == 3:  
 self.model\_obj.insert\_data\_controller(table\_num)  
 elif operation\_num == 4:  
 data = self.view\_obj.update\_data(table\_num)  
 self.model\_obj.update\_table(table\_num, data)  
 elif operation\_num == 5:  
 id\_num = self.view\_obj.delete()  
 self.model\_obj.delete\_data(table\_num, id\_num)  
 elif operation\_num == 6:  
 self.model\_obj.insert\_random\_data\_packet()  
 elif operation\_num == 7:  
 column\_names = self.model\_obj.get\_column\_names\_to\_view(table\_num)  
 search\_param = self.view\_obj.search(column\_names)  
 self.model\_obj.search\_into\_table(table\_num, search\_param)

**operations\_menu.txt**

+------------------------------+  
| Select operation: |  
+------------------------------+  
| 1: Show table |  
| 2: Show all tables |  
| 3: Add data |  
| 4: Update data |  
| 5: Delete data |  
| 6: Add random packed data |  
| 7: Search |  
| 8: Exit the program |  
+------------------------------+

**tables\_menu.txt**

+--------------------------+  
| Select a table: |  
+--------------------------+  
| 1: "Client" |  
| 2: "Order" |  
| 3: "Product" |  
| 4: "Product\_category" |  
| 5: "Store" |  
| 6: "Store\_department" |  
| 7: "order\_product" |  
+--------------------------+