

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»**

Тема: «Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL»

Виконала: студентка ІII курсу

ФПМ групи КВ-83

Мягкова Дар’я Андріївна

Перевірив: Павловський В.І.

Київ – 2020

**Зміст**

[Завдання 3](#_Toc58697540)

[Нормалізована модель даних БД «Сервіс доставки їжі» 6](#_Toc58697541)

[Завдання №1 7](#_Toc58697542)

[Завдання №2 13](#_Toc58697543)

[Завдання №3 14](#_Toc58697544)

[Код програми 16](#_Toc58697545)

[Файл controller.py 16](#_Toc58697546)

[Файл main.py 19](#_Toc58697547)

[Файл model.py 23](#_Toc58697548)

[Файл mvc\_exceptions.py 25](#_Toc58697549)

[Файл postgresql\_backend.py 25](#_Toc58697550)

[Файл time\_decor.py 29](#_Toc58697551)

[Файл view.py 29](#_Toc58697552)

# Завдання

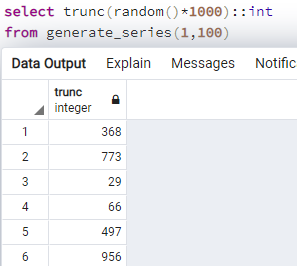
*Загальне завдання* роботи полягає у наступному:

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 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).

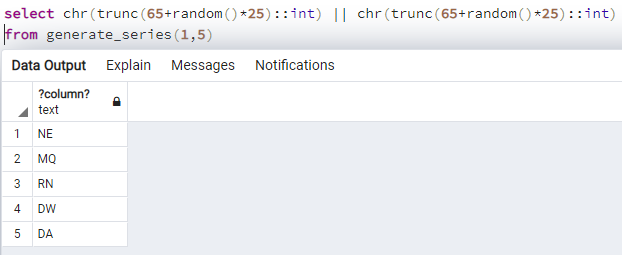
*Деталізоване завдання*:

1. 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.
2. 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**!

Приклад генерації 100 псевдовипадкових чисел:



Приклад генерації 5 псевдовипадкових рядків:



Приклад генерації псевдовипадкової мітки часу з діапазону [доступний за посиланням](https://stackoverflow.com/questions/22964272/postgresql-get-a-random-datetime-timestamp-between-two-datetime-timestamp/22965061).

Кількість даних для генерування має вводити користувач з клавіатури. Для тесту взяти 100 000 записів для однієї-двох таблиць.

Особливу увагу слід звернути на відповідність даних вимогам зовнішніх ключів з метою уникнення помилок порушення обмежень цілісності (foreign key).

1. 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.
2. Програмний код організувати згідно шаблону Model-View-Controller(MVC).  Приклад організації коду згідно шаблону доступний [за даним посиланням](https://www.giacomodebidda.com/mvc-pattern-in-python-introduction-and-basicmodel/). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

Рекомендована бібліотека взаємодії з PostgreSQL Psycopg2: http://initd.org/psycopg/docs/usage.html)

*Вимоги до інтерфейсу користувача*

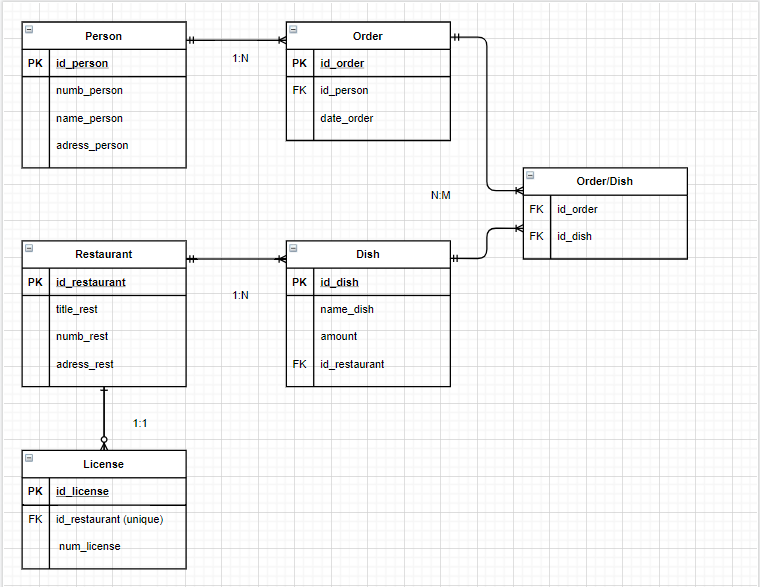
Використовувати консольний інтерфейс користувача.

*Вимоги до інструментарію*

 Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

# Нормалізована модель даних БД «Сервіс доставки їжі»

На рисунку зображена нормалізована модель даних БД ‘Тестування вакцини’, розроблена на першій лабораторній роботі.



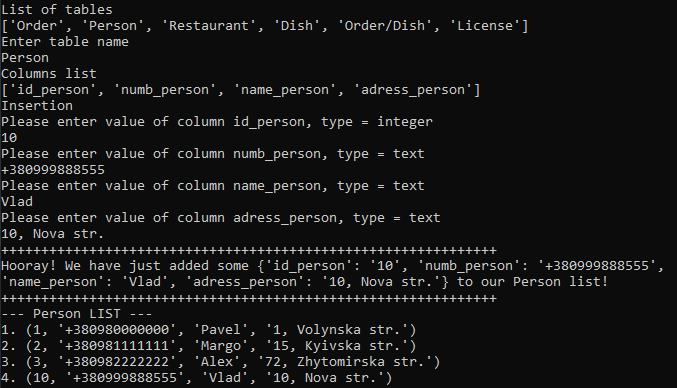
**Опис програми**

Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL і мови програмування Python, та реалізовує функціональні вимоги, що наведені у завданні. Додаток використовує шаблон проектування MVC.

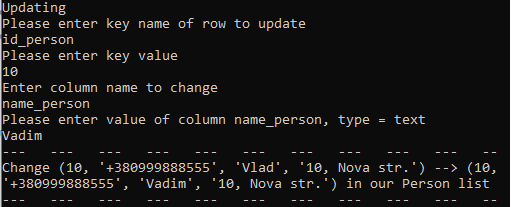
# Завдання №1

Приклад роботи програми, що виконує вставку, оновлення та видалення даних. При успішному виконанні команди програма сповіщає про це у вигляді повідомлення.

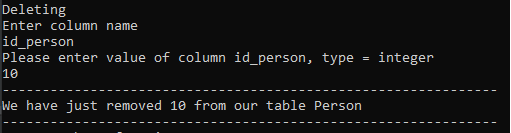
Вставка



Оновлення



Видалення



Обробка помилок. При введенні даних програма сповіщає користувачу інформацію про назву колонки та тип даних, що необхідно ввести. Якщо програма запрошує введення назви стовпчика та користувач виконує введення неіснуючого стовпчика, то програма виконує запрошення ввести назву стовпця знову, доки не отримає коректний результат. Також при виконанні вставки з вже існуючим ключем програма виконує обробку виключення і виводить на екран текст з помилкою. При оновленні, або видаленні програма перевіряє наявність запису з відповідним ключем, якщо ключ не було знайдено програма сповіщає про це.

Програма обробляє виключення, що генерують функції бібліотеки Psycopg2, а також для випадків, коли функції Psycopg2 не генерують виключень, наприклад видалення, відновлення по ключу, якого немає в базі, генерується виключення ItemNotStored.

Код функцій для основних операцій, що використовує модель (фрагмент файлу postgresql\_backend.py):

def **insert\_one**(conn, table\_name, field\_values):  
 cur = conn.cursor()  
 keys = field\_values.keys()  
 columns = ','.join(keys)  
 values = ','.join(['%({})s'.format(k) for k in keys])  
 insert = 'INSERT INTO "{}" ({}) values ({})'.format(table\_name, columns, values)  
 cur.mogrify(insert, field\_values)  
  
 cur.execute(insert, field\_values)  
 conn.commit()  
  
  
def **select\_one**(conn, table\_name, key\_name, key\_value):  
 cur = conn.cursor()  
 cur.execute('SELECT \* FROM "{}" WHERE {} = %s LIMIT 1'.format(table\_name, key\_name), (key\_value,))  
 result = cur.fetchone()  
 if result is not None:  
 return result  
 else:  
 **raise mvc\_exc.ItemNotStored**(  
 'Can\'t read row with {} = "{}" because it\'s not stored in table "{}"'  
 .format(key\_name, key\_value, table\_name))  
  
  
def **update\_one**(conn, table\_name, key\_change, new\_val, key, key\_val):  
 cur = conn.cursor()  
 sql\_check = 'SELECT EXISTS(SELECT 1 FROM "{}" WHERE {} = %s LIMIT 1)' \  
 .format(table\_name, key)  
 cur.execute(sql\_check, (key\_val,))  
 result = cur.fetchone()  
 if result is not None:  
 cur.execute('UPDATE "{}" Set {} = %s WHERE {} = %s;'.format(table\_name, key\_change, key), (new\_val, key\_val))  
 conn.commit()  
 else:  
 **raise mvc\_exc.ItemNotStored**(  
 'Can\'t update {} = {} because it\'s not stored in table "{}"'  
 .format(key, key\_val, table\_name))  
  
  
def **delete\_one**(conn, table\_name, key\_name, key\_val):  
 cur = conn.cursor()  
 sql\_check = 'SELECT EXISTS(SELECT 1 FROM "{}" WHERE {} = %s LIMIT 1)' \  
 .format(table\_name, key\_name)  
 cur.execute(sql\_check, (key\_val,))  
 result = cur.fetchone()  
  
 if result is not None:  
 cur.execute('DELETE FROM "{}" WHERE {} = %s'.format(table\_name, key\_name), (key\_val,))  
 conn.commit()  
 else:  
 **raise mvc\_exc.ItemNotStored**(  
 'Can\'t delete {} = {} because it\'s not stored in table "{}"'  
 .format(key\_name, key\_val, table\_name))

На рівні контролеру виконується виклик функцій моделі, виведення результатів роботи, або помилок. Контролер дозволяє обробку будь-яких виключень, що можуть виникнути під час роботи програми в тому числі і посилальної цілісністю двох пов'язаних таблиць. Далі наведемо фрагмент коду контролера, що забезпечує обробку і виведення помилок.

def **insert\_item**(self, table\_name, field\_values):

try:

self.model.create\_item(table\_name, field\_values)

self.view.display\_item\_stored(table\_name, field\_values)

except mvc\_exc.ItemAlreadyStored as e:

self.view.display\_item\_already\_stored\_error(table\_name, e)

except psycopg2.Error as e:

***self.model.connection.rollback()***

***print('Insert error. \nType: {} \nText: {}'.format(type(e), e))***

def **update\_item**(self, table\_name, key\_change, new\_val, key\_name, key\_val):

try:

old\_item = self.model.read\_item(table\_name, key\_name, key\_val)

self.model.update\_item(table\_name, key\_change, new\_val, key\_name, key\_val)

new\_item = self.model.read\_item(table\_name, key\_name, key\_val)

self.view.display\_item\_updated(table\_name, new\_item, old\_item)

except mvc\_exc.ItemNotStored as e:

self.view.display\_item\_not\_yet\_stored\_error(key\_val, table\_name, e)

except psycopg2.Error as e:

***self.model.connection.rollback()***

***print('Update error. \nType: {} \nText: {}'.format(type(e), e))***

def **delete\_item**(self, table\_name, key\_name, key\_val):

try:

self.model.delete\_item(table\_name, key\_name, key\_val)

self.view.display\_item\_deletion(table\_name, key\_val)

except mvc\_exc.ItemNotStored as e:

self.view.display\_item\_not\_yet\_stored\_error(key\_val, table\_name, e)

except psycopg2.Error as e:

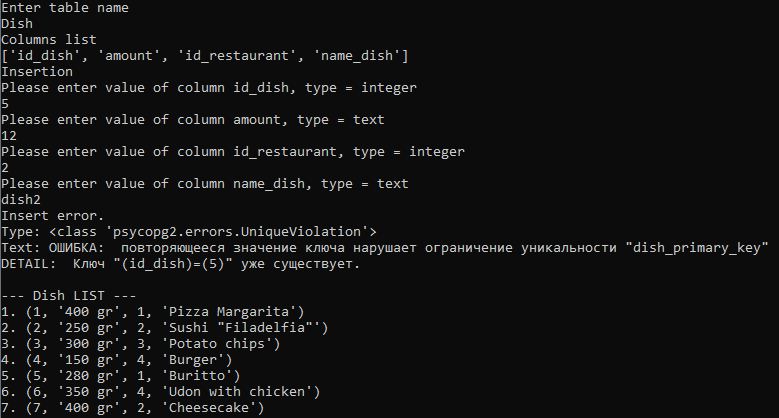
***self.model.connection.rollback()***

***print('Delete error. \nType: {} \nText: {}'.format(type(e), e))***

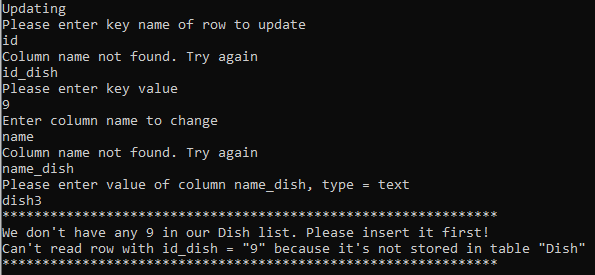
Вставка. Обробка виключення, що стосується обмеження зовнішнього ключа



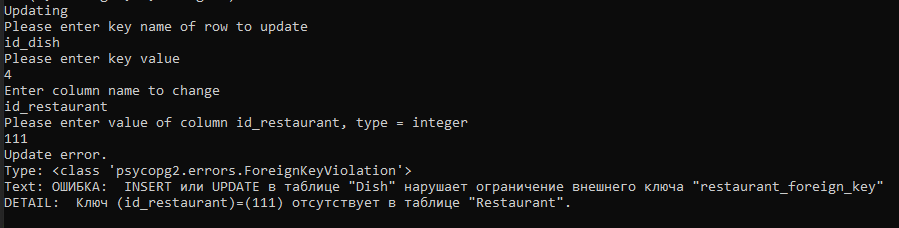
Вставка. Повторне значення первинного ключа



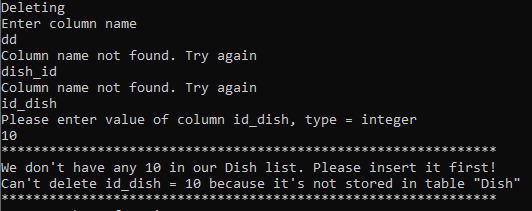
Оновлення. Перевірка коректності введення назви колонки, а також введення значення ключа якого не існує для таблиці в базі даних.



Оновлення. Обробка виключення, що стосується обмеження зовнішнього ключа

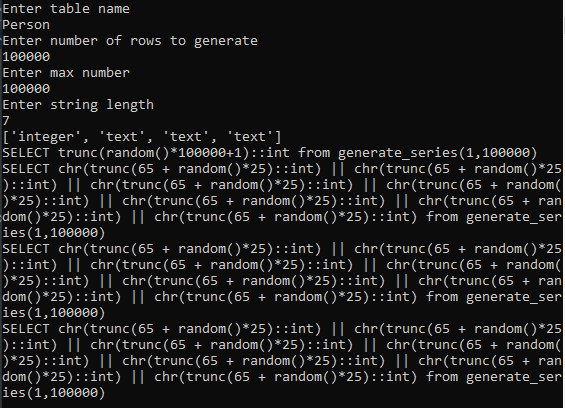


Видалення. Некоректне введення назви стовпчика та значення ключа, що не існує.

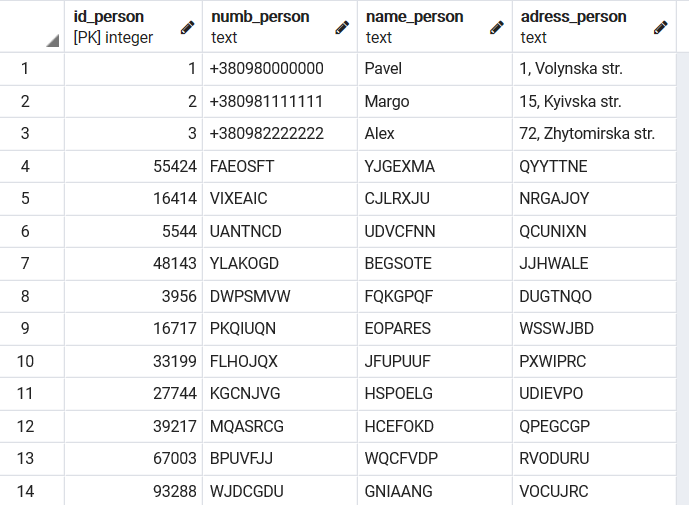


# Завдання №2

Приклад роботи генерації псевдовипадкових рядків в таблицях Person:



Результат



Запити, що було створено для таблиці. Для кожного стовпця по одному запиту.

SELECT trunc(random()\*100000+1)::int from generate\_series(1,100000)

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) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) from generate\_series(1,100000)

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) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) from generate\_series(1,100000)

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) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) from generate\_series(1,100000)

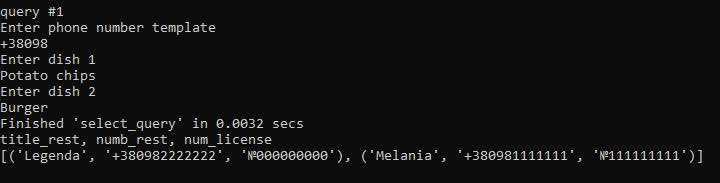
# Завдання №3

В ході роботи було реалізовано три запити:

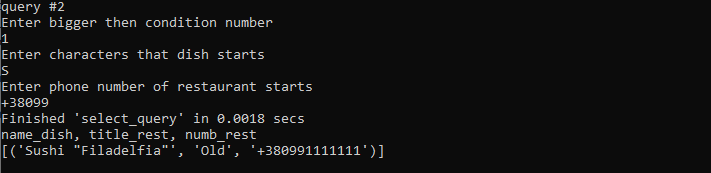
1. Шукає заклади з телефонами, що починаються на деякі числа, також заклад повинен мати одну з двох заданих страв.
2. Шукає ресторани в яких будь-яку страву було заказано більше за задане число та назва страву повинна починатися з заданих літер та телефон закладу повинен починатися з заданої послідовності чисел.
3. Обробляє замовлення, що було заказано після заданої дати, веде підрахунок замовлених страв для окремого замовника, кількість яких повинно бути у заданих межах та шукає імена серед замовників, що повинно мати задану літеру, або послідовність літер.

Для розглянемо приклади вводу необхідних даних та виконання запитів.

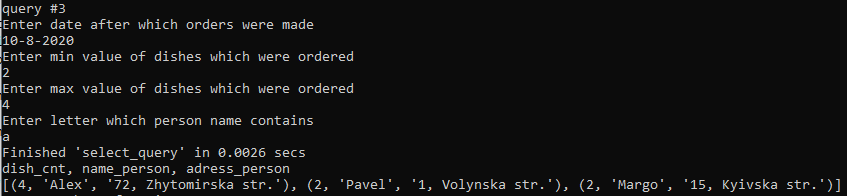
Запит №1



Запит №2



Запит №3



Текст запитів:

*Запит №1*

JOIN "License"

ON "Restaurant".id\_restaurant = "License".id\_restaurant

JOIN "Dish"

ON "Restaurant".id\_restaurant = "Dish".id\_restaurant

WHERE numb\_rest LIKE '{0}%'

AND (name\_dish = '{1}' OR name\_dish = '{2}')

*Запит №2*

SELECT name\_dish, title\_rest, numb\_rest FROM

(SELECT id\_dish, count(id\_dish) AS dish\_cnt FROM "Order/Dish"

GROUP BY id\_dish) AS "DishCont"

JOIN "Dish"

ON "Dish".id\_dish = "DishCont".id\_dish

JOIN "Restaurant"

ON "Restaurant".id\_restaurant = "Dish".id\_restaurant

WHERE dish\_cnt > {0}

AND name\_dish LIKE '{1}%'

AND numb\_rest LIKE '{2}%'

*Запит №3*

SELECT dish\_cnt, name\_person, adress\_person FROM

(SELECT count(id\_dish) AS dish\_cnt, name\_person, "Person".id\_person, adress\_person FROM "Person"

JOIN "Order"

ON "Order".id\_person = "Person".id\_person

JOIN "Order/Dish"

ON "Order/Dish".id\_order = "Order".id\_order

WHERE date\_order > '{0}'

GROUP BY name\_person, "Person".id\_person) AS "DishCount"

WHERE "DishCount".dish\_cnt BETWEEN {1} AND {2}

AND name\_person ILIKE '%{3}%'

# Код програми

## Файл controller.py

import psycopg2  
  
import mvc\_exceptions as mvc\_exc  
  
  
class Controller(object):  
  
 def \_\_init\_\_(self, model, view):  
 self.model = model  
 self.view = view  
  
 def show\_items(self, table\_name, bullet\_points=False):  
 items = self.model.read\_items(table\_name)  
  
 if bullet\_points:  
 self.view.show\_bullet\_point\_list(table\_name, items)  
 else:  
 self.view.show\_number\_point\_list(table\_name, items)  
  
 def insert\_item(self, table\_name, field\_values):  
 try:  
 self.model.create\_item(table\_name, field\_values)  
 self.view.display\_item\_stored(table\_name, field\_values)  
 except mvc\_exc.ItemAlreadyStored as e:  
 self.view.display\_item\_already\_stored\_error(table\_name, e)  
 except psycopg2.Error as e:  
 self.model.connection.rollback()  
 print('Insert error. \nType: {} \nText: {}'.format(type(e), e))  
  
 def update\_item(self, table\_name, key\_change, new\_val, key\_name, key\_val):  
 try:  
 old\_item = self.model.read\_item(table\_name, key\_name, key\_val)  
 self.model.update\_item(table\_name, key\_change, new\_val, key\_name, key\_val)  
 new\_item = self.model.read\_item(table\_name, key\_name, key\_val)  
 self.view.display\_item\_updated(table\_name, new\_item, old\_item)  
 except mvc\_exc.ItemNotStored as e:  
 self.view.display\_item\_not\_yet\_stored\_error(key\_val, table\_name, e)  
 except psycopg2.Error as e:  
 self.model.connection.rollback()  
 print('Update error. \nType: {} \nText: {}'.format(type(e), e))  
  
 def delete\_item(self, table\_name, key\_name, key\_val):  
 try:  
 self.model.delete\_item(table\_name, key\_name, key\_val)  
 self.view.display\_item\_deletion(table\_name, key\_val)  
 except mvc\_exc.ItemNotStored as e:  
 self.view.display\_item\_not\_yet\_stored\_error(key\_val, table\_name, e)  
 except psycopg2.Error as e:  
 self.model.connection.rollback()  
 print('Delete error. \nType: {} \nText: {}'.format(type(e), e))  
  
 def get\_columns(self, table\_name):  
 return self.model.get\_columns(table\_name)  
  
 def check\_for\_present(self, column\_name, table\_name):  
 return self.model.check\_for\_present(column\_name, table\_name)  
  
 def get\_list\_of\_tables(self):  
 return self.model.get\_list\_of\_tables()  
  
 def get\_column\_type(self, table, column\_name):  
 return self.model.get\_column\_type(table, column\_name)  
  
 def column\_data(self, table\_name, column\_name):  
 return self.model.column\_data(table\_name, column\_name)  
  
 def auto\_gen\_int(self, rows\_number, max\_val=100):  
 return self.model.auto\_gen\_int(rows\_number, max\_val)  
  
 def auto\_gen\_char(self, rows\_number, str\_len=5):  
 return self.model.auto\_gen\_char(rows\_number, str\_len)  
  
 def search1(self, search\_tmpl, dish1, dish2):  
 sql = 'SELECT title\_rest, numb\_rest, num\_license ' \  
 'FROM "Restaurant" ' \  
 'JOIN "License" ' \  
 'ON "Restaurant".id\_restaurant = "License".id\_restaurant ' \  
 'JOIN "Dish" ' \  
 'ON "Restaurant".id\_restaurant = "Dish".id\_restaurant ' \  
 'WHERE numb\_rest LIKE \'{}%\' '\  
 'AND (name\_dish = \'{}\' OR name\_dish = \'{}\')'.format(search\_tmpl, dish1, dish2)  
  
 res = self.model.select\_query(sql)  
 print("title\_rest, numb\_rest, num\_license")  
 print(res)  
  
 def search2(self, dish\_amount, name\_dish\_start, numb\_start):  
 sql = 'SELECT name\_dish, title\_rest, numb\_rest FROM ' \  
 '(SELECT id\_dish, count(id\_dish) AS dish\_cnt FROM "Order/Dish" ' \  
 'GROUP BY id\_dish) AS "DishCont" ' \  
 'JOIN "Dish" ' \  
 'ON "Dish".id\_dish = "DishCont".id\_dish ' \  
 'JOIN "Restaurant" ' \  
 'ON "Restaurant".id\_restaurant = "Dish".id\_restaurant ' \  
 'WHERE dish\_cnt > {} ' \  
 'AND name\_dish LIKE \'{}%\' ' \  
 'AND numb\_rest LIKE \'{}%\''.format(dish\_amount, name\_dish\_start, numb\_start)  
 res = self.model.select\_query(sql)  
 print("name\_dish, title\_rest, numb\_rest")  
 print(res)  
  
 def search3(self, after\_date, dish\_cnt\_min, dish\_cnt\_max, name\_contains):  
 sql = 'SELECT dish\_cnt, name\_person, adress\_person FROM ' \  
 '(SELECT count(id\_dish) AS dish\_cnt, name\_person, "Person".id\_person, adress\_person ' \  
 'FROM "Person" ' \  
 'JOIN "Order" ON "Order".id\_person = "Person".id\_person ' \  
 'JOIN "Order/Dish" ON "Order/Dish".id\_order = "Order".id\_order ' \  
 'WHERE date\_order > \'{}\' ' \  
 'GROUP BY name\_person, "Person".id\_person) AS "DishCount" ' \  
 'WHERE "DishCount".dish\_cnt BETWEEN {} AND {} ' \  
 'AND name\_person ILIKE \'%{}%\''.format(after\_date, dish\_cnt\_min, dish\_cnt\_max, name\_contains)  
 res = self.model.select\_query(sql)  
 print("dish\_cnt, name\_person, adress\_person")  
 print(res)

## Файл main.py

import psycopg2  
  
from controller import Controller  
from model import ModelPostgreSQL  
from view import View  
  
  
def task1(ctrl):  
 print("List of tables")  
 print(ctrl.get\_list\_of\_tables())  
 print("Enter table name")  
 table\_name = input()  
 columns = ctrl.get\_columns(table\_name)  
 if columns is None:  
 return  
 print('Columns list')  
 print(ctrl.get\_columns(table\_name))  
  
 ins\_values = {}  
 print("Insertion")  
 for col in columns:  
 col\_type = ctrl.get\_column\_type(table\_name, col)  
 print('Please enter value of column {}, type = {}'.format(col, col\_type))  
 val = input()  
 ins\_values[col] = val  
 ctrl.insert\_item(table\_name, ins\_values)  
 ctrl.show\_items(table\_name)  
  
 print("Updating")  
 print('Please enter key name of row to update')  
 key\_name = input()  
 while ctrl.check\_for\_present(key\_name, table\_name) is False:  
 print('Column name not found. Try again')  
 key\_name = input()  
 print('Please enter key value')  
 key\_val = input()  
  
 print('Enter column name to change')  
 column\_name = input()  
 while ctrl.check\_for\_present(column\_name, table\_name) is False:  
 print('Column name not found. Try again')  
 column\_name = input()  
  
 col\_type = ctrl.get\_column\_type(table\_name, column\_name)  
 print('Please enter value of column {}, type = {}'.format(column\_name, col\_type))  
 column\_value = input()  
  
 ctrl.update\_item(table\_name, column\_name, column\_value, key\_name, key\_val)  
  
 print("Deleting")  
 print('Enter column name')  
 column\_name = input()  
 while ctrl.check\_for\_present(column\_name, table\_name) is False:  
 print('Column name not found. Try again')  
 column\_name = input()  
 col\_type = ctrl.get\_column\_type(table\_name, column\_name)  
 print('Please enter value of column {}, type = {}'.format(column\_name, col\_type))  
 column\_value = input()  
 ctrl.delete\_item(table\_name, column\_name, column\_value)  
  
  
def task2(ctrl):  
 print("Enter table name")  
 table\_name = input()  
 columns = ctrl.get\_columns(table\_name)  
 if columns is None:  
 return  
  
 print("Enter number of rows to generate")  
 rows\_number = int(input())  
  
 print("Enter max number")  
 max\_val = int(input())  
 print("Enter string length")  
 str\_len = int(input())  
  
 columns\_type = [ctrl.get\_column\_type(table\_name, col) for col in columns]  
 print(columns\_type)  
  
 elements = []  
 for i, column in enumerate(columns):  
  
 if columns\_type[i] == 'text':  
 val = ctrl.auto\_gen\_char(rows\_number, str\_len)  
 else:  
 val = ctrl.auto\_gen\_int(rows\_number, max\_val)  
 elements.append(val)  
  
 for j in range(rows\_number):  
 item = []  
 for k in range(len(columns)):  
 item.append(elements[k][j])  
 values = dict(zip(columns, item))  
 try:  
 ctrl.model.create\_item(table\_name, values)  
 except (psycopg2.DatabaseError, Exception) as error:  
 pass  
  
  
def task3(ctrl):  
  
 print('query #1')  
 print('Enter phone number template')  
 phone = input()  
 print('Enter dish 1')  
 dish1 = input()  
 print('Enter dish 2')  
 dish2 = input()  
  
 ctrl.search1(phone, dish1, dish2)  
  
 print()  
 print('query #2')  
 print('Enter bigger then condition number')  
 num = input()  
 print('Enter characters that dish starts')  
 dish\_start = input()  
 print('Enter phone number of restaurant starts')  
 res\_start = input()  
  
 ctrl.search2(num, dish\_start, res\_start)  
  
 print()  
 print('query #3')  
 print('Enter date after which orders were made')  
 date\_after = input()  
 print('Enter min value of dishes which were ordered ')  
 dish\_min = input()  
 print('Enter max value of dishes which were ordered')  
 dish\_max = input()  
 print('Enter letter which person name contains')  
 letter = input()  
 ctrl.search3(date\_after, dish\_min, dish\_max, letter)  
  
  
def menu():  
 print("Enter number of action")  
 print("1. Task #1")  
 print("2. Task #2")  
 print("3. Task #3")  
 print("4. Exit")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 c = Controller(ModelPostgreSQL(), View())  
  
 task = 0  
 while task != '4':  
 menu()  
 task = input()  
 if task == '1':  
 task1(c)  
 elif task == '2':  
 task2(c)  
 elif task == '3':  
 task3(c)

## Файл model.py

import psycopg2 as psycopg2  
import postgresql\_backend  
from time\_decor import timer  
  
  
class ModelPostgreSQL(object):  
  
 def \_\_init\_\_(self):  
 self.\_connection = psycopg2.connect(database='delivery', user='postgres',  
 password='qwerty', host='localhost')  
  
 @property  
 def connection(self):  
 return self.\_connection  
  
 @timer  
 def select\_query(self, query):  
 return postgresql\_backend.select\_query(self.connection, query)  
  
 def create\_item(self, table\_name, field\_values):  
 return postgresql\_backend.insert\_one(self.connection, table\_name, field\_values)  
  
 def read\_item(self, table\_name, key\_name, key\_val):  
 return postgresql\_backend.select\_one(self.connection, table\_name, key\_name, key\_val)  
  
 def read\_items(self, table\_name):  
 return postgresql\_backend.select\_all(self.connection, table\_name)  
  
 def update\_item(self, table\_name, key\_change, new\_val, key\_name, key\_val):  
 return postgresql\_backend.update\_one(self.connection, table\_name, key\_change, new\_val, key\_name, key\_val)  
  
 def delete\_item(self, table\_name, key\_name, key\_val):  
 return postgresql\_backend.delete\_one(self.connection, table\_name, key\_name, key\_val)  
  
 def get\_columns(self, table\_name):  
 return postgresql\_backend.get\_columns(self.connection, table\_name)  
  
 def check\_for\_present(self, column\_name, table\_name):  
 return postgresql\_backend.check\_for\_present(self.connection, column\_name, table\_name)  
  
 def get\_list\_of\_tables(self):  
 return postgresql\_backend.get\_list\_of\_tables(self.connection)  
  
 def get\_column\_type(self, table, column\_name):  
 return postgresql\_backend.get\_column\_type(self.connection, table, column\_name)  
  
 def column\_data(self, table\_name, column\_name):  
 return postgresql\_backend.column\_data(self.connection, table\_name, column\_name)  
  
 def auto\_gen\_int(self, rows\_number, max\_val):  
 return postgresql\_backend.auto\_gen\_int(self.connection, max\_val, rows\_number)  
  
 def auto\_gen\_char(self, rows\_number, str\_len):  
 return postgresql\_backend.auto\_gen\_char(self.connection, str\_len, rows\_number)  
  
 def \_\_del\_\_(self):  
 self.connection.close()

## Файл mvc\_exceptions.py

class ItemAlreadyStored(Exception):

pass

class ItemNotStored(Exception):

pass

## Файл postgresql\_backend.py

import psycopg2  
from psycopg2 import IntegrityError  
import mvc\_exceptions as mvc\_exc  
  
  
def select\_query(conn, query):  
 cur = conn.cursor()  
 cur.execute(query)  
 return cur.fetchall()  
  
  
def select\_all(conn, table\_name):  
 cur = conn.cursor()  
 cur.execute('SELECT \* FROM "{}"'.format(table\_name))  
 return cur.fetchall()  
  
  
def insert\_one(conn, table\_name, field\_values):  
 cur = conn.cursor()  
 keys = field\_values.keys()  
 columns = ','.join(keys)  
 values = ','.join(['%({})s'.format(k) for k in keys])  
 insert = 'INSERT INTO "{}" ({}) values ({})'.format(table\_name, columns, values)  
 cur.mogrify(insert, field\_values)  
  
 cur.execute(insert, field\_values)  
 conn.commit()  
  
  
def select\_one(conn, table\_name, key\_name, key\_value):  
 cur = conn.cursor()  
 cur.execute('SELECT \* FROM "{}" WHERE {} = %s LIMIT 1'.format(table\_name, key\_name), (key\_value,))  
 result = cur.fetchone()  
 if result is not None:  
 return result  
 else:  
 raise mvc\_exc.ItemNotStored(  
 'Can\'t read row with {} = "{}" because it\'s not stored in table "{}"'  
 .format(key\_name, key\_value, table\_name))  
  
  
def update\_one(conn, table\_name, key\_change, new\_val, key, key\_val):  
 cur = conn.cursor()  
 sql\_check = 'SELECT EXISTS(SELECT 1 FROM "{}" WHERE {} = %s LIMIT 1)' \  
 .format(table\_name, key)  
 cur.execute(sql\_check, (key\_val,))  
 result = cur.fetchone()  
 if result is not None:  
 cur.execute('UPDATE "{}" Set {} = %s WHERE {} = %s;'.format(table\_name, key\_change, key), (new\_val, key\_val))  
 conn.commit()  
 else:  
 raise mvc\_exc.ItemNotStored(  
 'Can\'t update {} = {} because it\'s not stored in table "{}"'  
 .format(key, key\_val, table\_name))  
  
  
def delete\_one(conn, table\_name, key\_name, key\_val):  
 cur = conn.cursor()  
 sql\_check = 'SELECT EXISTS(SELECT 1 FROM "{}" WHERE {} = %s LIMIT 1)' \  
 .format(table\_name, key\_name)  
 cur.execute(sql\_check, (key\_val,))  
 result = cur.fetchone()  
  
 if result is not None:  
 cur.execute('DELETE FROM "{}" WHERE {} = %s'.format(table\_name, key\_name), (key\_val,))  
 conn.commit()  
 else:  
 raise mvc\_exc.ItemNotStored(  
 'Can\'t delete {} = {} because it\'s not stored in table "{}"'  
 .format(key\_name, key\_val, table\_name))  
  
  
def get\_columns(conn, table\_name):  
 cur = conn.cursor()  
 value = []  
 cur.execute("SELECT column\_name FROM INFORMATION\_SCHEMA.COLUMNS" \  
 " WHERE TABLE\_NAME = '{}'".format(table\_name))  
 for column in cur.fetchall():  
 value.append(\*column)  
 return value  
  
  
def check\_for\_present(conn, column\_name, table\_name):  
 columns = get\_columns(conn, table\_name)  
 if column\_name not in columns:  
 return False  
 return True  
  
  
def get\_list\_of\_tables(con):  
 cur = con.cursor()  
 cur.execute("SELECT table\_name FROM information\_schema.tables "  
 "WHERE table\_schema = 'public'")  
 return [col[0] for col in cur.fetchall()]  
 # for table in cur.fetchall():  
 # print(table)  
  
  
def get\_column\_type(conn, table, column\_name):  
 cur = conn.cursor()  
 cur.execute("SELECT column\_name, data\_type FROM information\_schema.columns "  
 "WHERE table\_name = '{}'".format(table))  
  
 for col in cur.fetchall():  
 if column\_name in col:  
 return col[1]  
  
  
def column\_data(con, table\_name, column\_name):  
 try:  
 cur = con.cursor()  
 cur.execute('SELECT {} FROM "{}"'.format(column\_name, table\_name))  
 values = []  
 for val in cur.fetchall():  
 values.append(\*val)  
 return values  
 except(psycopg2.DatabaseError, Exception) as error:  
 print(error)  
  
  
def auto\_gen\_int(con, max\_val, rows\_number):  
 try:  
 query = 'SELECT trunc(random()\*{}+1)::int from generate\_series(1,{})'.format(max\_val, rows\_number)  
 print(query)  
 cur = con.cursor()  
 cur.execute(query)  
 # numbers = [num[0] for num in cursor.fetchall()]  
 return cur.fetchall()  
 except(psycopg2.DatabaseError, Exception) as error:  
 print(error)  
  
  
def auto\_gen\_char(con, str\_len, rows\_number):  
 try:  
 part = 'chr(trunc(65 + random()\*25)::int)'  
 if str\_len > 0:  
 param = part + (' || ' + part) \* (str\_len - 1)  
  
 query = 'SELECT {} from generate\_series(1,{})'.format(param, rows\_number)  
 print(query)  
 cur = con.cursor()  
 cur.execute(query)  
 return cur.fetchall()  
 except(psycopg2.DatabaseError, Exception) as error:  
 print(error)

## Файл time\_decor.py

import time  
import functools  
  
  
def timer(func):  
 """Print the runtime of func"""  
  
 @functools.wraps(func)  
 def wrapper\_timer(\*args, \*\*kwargs):  
 start\_time = time.perf\_counter()  
 value = func(\*args, \*\*kwargs)  
 end\_time = time.perf\_counter()  
 run\_time = end\_time - start\_time  
 print(f"Finished {func.\_\_name\_\_!r} in {run\_time:.4f} secs")  
 return value  
  
 return wrapper\_timer

## Файл view.py

class View(object):  
  
 @staticmethod  
 def show\_bullet\_point\_list(item\_type, items):  
 print('--- {} LIST ---'.format(item\_type.upper()))  
 for item in items:  
 print('\* {}'.format(item))  
  
 @staticmethod  
 def show\_number\_point\_list(table\_name, items):  
 print('--- {} LIST ---'.format(table\_name))  
 for i, item in enumerate(items):  
 print('{}. {}'.format(i+1, item))  
  
 @staticmethod  
 def show\_item(item\_type, item, item\_info):  
 print('//////////////////////////////////////////////////////////////')  
 print('Good news, we have some {}!'.format(item.upper()))  
 print('{} INFO: {}'.format(item\_type.upper(), item\_info))  
 print('//////////////////////////////////////////////////////////////')  
  
 @staticmethod  
 def display\_missing\_item\_error(item, err):  
 print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
 print('We are sorry, we have no {}!'.format(item.upper()))  
 print('{}'.format(err.args[0]))  
 print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
  
 @staticmethod  
 def display\_item\_already\_stored\_error(table\_name, err):  
 print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
 print('Hey! We already have this key in our {} list!'  
 .format(table\_name))  
 print('{}'.format(err.args[0]))  
 print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
  
 @staticmethod  
 def display\_item\_not\_yet\_stored\_error(item, table\_name, err):  
 print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
 print('We don\'t have any {} in our {} list. Please insert it first!'  
 .format(item, table\_name))  
 print('{}'.format(err.args[0]))  
 print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
  
 @staticmethod  
 def display\_item\_stored(table\_name, item):  
 print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')  
 print('Hooray! We have just added some {} to our {} list!'  
 .format(item, table\_name))  
 print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')  
  
 @staticmethod  
 def display\_item\_updated(table\_name, new\_item, old\_item):  
 print('--- --- --- --- --- --- --- --- --- --- --')  
 print('Change {} --> {} in our {} list'.format(old\_item, new\_item, table\_name))  
 print('--- --- --- --- --- --- --- --- --- --- --')  
  
 @staticmethod  
 def display\_item\_deletion(table\_name, name):  
 print('--------------------------------------------------------------')  
 print('We have just removed {} from our table {}'.format(name, table\_name))  
 print('--------------------------------------------------------------')