

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра інженерії програмного забезпечення**

**Лабораторна робота №2**

з дисципліни

**«**Бази даних і засоби управління**»**

Виконав:

студент ІII курсу

ФПМ групи КП-71

Гришко Ю.І.

Перевірив: Петрашенко А. В.

Київ 2019

**Ознайомлення з базовими операціями СУБД PostgreSQL**

*Завдання* роботи полягає у наступному:

1. Виконати нормалізацію бази даних, яка була створена у лабораторній роботі №1, до третьої нормальної форми (3НФ);
2. Реалізувати функціональні вимоги, наведені нижче.

*Функціональні вимоги*

1. Реалізувати внесення, редагування та вилучення даних у базі засобами консольного інтерфейсу;
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі;
3. Забезпечити реалізацію пошуку за двома-трьома атрибутами з двох сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як перелічення, для логічного типу – значення True/False, для дат – у рамках діапазону дат;
4. Забезпечити реалізацію повнотекстового пошуку за будь-яким текстовим атрибутом бази даних засобами PostgreSQL з виділенням знайденого фрагменту.

*Вимоги до інтерфейсу користувача*

1. Використовувати консольний інтерфейс користувача.

**Варіант 2**

Пошук за атрибутами – діапазон дат та логічний тип.

Повнотекстовий пошук – слово не входить, ціла фраза.

**Нормалізована модель даних БД «Магазин»**

Усі таблиці (відношення) знаходяться в 3 НФ тому, що у кожній із них:

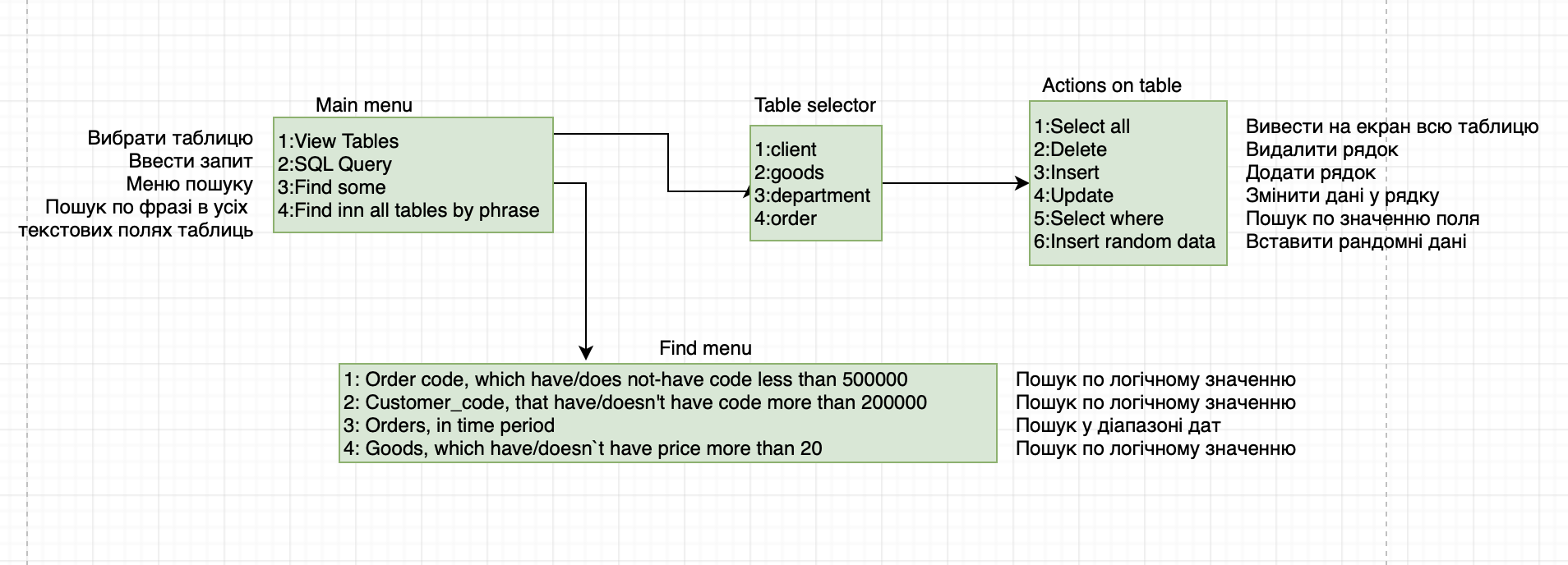
1. Всі атрибути є атомарними та відсутні повторення рядків (1НФ);
2. Первинний ключ складається лише з одного атрибуту (2НФ);
3. Кожний не первинний атрибут не є транзитивно залежним від первинного ключа (не залежить від іншого непервинного атрибуту)(3НФ).

****

**Загальний опис програми**

Програма написана з дотриманням патерну MVC (Model, View, Controller). Така архітектура дозволяє відокремити маніпуляцію з даними та ввід/вивід інформації. Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL. Програма складається з 5 модулів:

1. main.py – точка входу до програми, тут створюєтся Сontroller, Model, та View;
2. utils.py – містить утилітарні функції для роботи з базою даних(підключення, різні запити, генерація випадкових значень, т.д.)
3. model.py – клас Model, який містить методи для роботи з БД;
4. view.py – клас View, який містить методи для виводу результатів роботи програми на екран;
5. controller.py – клас Controller, який містить логіку роботи програми(обробка вводу користувача, визови методів з View та Model).

**Опис меню програми**

**Посилання на основні класи та методи програми**

1. [Main](#main)
2. [Utils](#utils)
3. [Model](#model)
   1. Завдання 1
      1. [Метод, який дістає інформацію з таблиці](#select_some)
      2. [Метод, який додає інформацію до БД](#insert_data)
      3. [Метод, який видаляє інформацію з БД](#delete_data)
      4. [Метод, який оновлює інформацію в БД](#update)
   2. Завдання 2
      1. [Метод, який створює випадкову інформацію і додає її до таблиці](#insert_random)
   3. [Завдання 3](#Task3)
      1. [Метод, який реалізує вибір інформації за логічним значенням](#find_bool)
      2. [Метод, який реалізує вибір інформації за наявності слова у трьох таблицях одночасно](#find_word)
      3. [Метод, який реалізує вибір інформації за діапазоном дат у двох таблицях одночасно](#find_time)
      4. [Метод, який реалізує вибір інформації за відсутності слова у двох таблицях одночасно](#find_no_word)
   4. Завдання 4
      1. [Метод, який запускає пошук по фразі у БД](#find_by_phraze)
4. [View](#view)
   1. [Метод, який виводить на екран](#print_table) таблицю
5. [Controller](#Controller)

**Лістинг програми**

**main.py**

from controller import Controller  
import utils  
from model import Model  
from view import View  
  
  
def main():  
 controller = Controller(Model("localhost", "1"), View())  
 controller.start()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**utils.py**

import psycopg2  
import random  
from psycopg2 import sql  
  
  
# Connect to psql database  
def open\_connection(phost, pport, pdatabase, puser, ppassword):  
 try:  
 return psycopg2.connect(host=phost, port=pport, database=pdatabase, user=puser, password=ppassword)  
 except (Exception, psycopg2.Error) as error:  
 print("Error: connection with PostgreSQL\n\t", error)  
  
  
# Execute query and get result. If error - print exception  
def query(cursor, q, query\_params):  
 try:  
 cursor.execute(q, query\_params)  
 retval = cursor.fetchall()  
 if retval:  
 return retval  
 except BaseException as e:  
 print("ERROR: ", str(e))  
  
  
# Get list of tables in database  
def list\_tables(cursor):  
 data = query(cursor, """  
 SELECT table\_name FROM information\_schema.tables   
 WHERE table\_schema = 'public';  
 """, ())  
 if data:  
 return [x[0] for x in data]  
  
  
# Get list of columns for some table in database  
def list\_table\_columns(cursor, table):  
 data = query(cursor, """  
 SELECT column\_name  
 FROM information\_schema.columns  
 WHERE table\_schema = 'public'  
 AND table\_name = %s;  
 """, (table,))  
 if data:  
 return [x[0] for x in data]  
  
  
# Get type of column(integer, text, etc.)  
def get\_column\_type(cursor, table, column):  
 data = query(cursor, """  
 SELECT data\_type  
 FROM information\_schema.columns  
 WHERE table\_schema = 'public'  
 AND table\_name = %s  
 AND column\_name = %s;  
 """, (table, column))  
 if data:  
 return data[0][0]  
  
  
# Insert some data to table. NO TYPE CHECKS!!!  
def insert\_data(connection, cursor, table, data):  
 q = sql.SQL("""  
 INSERT INTO {} VALUES %s;  
 """).format(sql.Identifier(table))  
  
 try:  
 cursor.execute(q, (data,))  
 connection.commit()  
 except BaseException as e:  
 print("ERROR: ", str(e))  
 return False  
 return True  
  
  
# Update item in table. Selection of item is done with selection of some value in some field.  
# New data passed through list with strings. If string is empty - don't update that field.  
def update\_item(connection, cursor, table\_name, column\_to\_check, expected\_value, new\_data):  
 columns = list\_table\_columns(cursor, table\_name)  
 insert\_str = ""  
 for i in range(0, len(columns)):  
 # '' means default value  
 if new\_data[i] != '':  
 if insert\_str != "":  
 insert\_str += ', '  
 insert\_str += "%s='%s'" % (columns[i], new\_data[i])  
  
 q = sql.SQL("""  
 UPDATE {} SET """ + insert\_str + """ WHERE {}=%s;  
 """).format(sql.Identifier(table\_name), sql.Identifier(column\_to\_check))  
  
 try:  
 cursor.execute(q, (expected\_value, ))  
 connection.commit()  
 except BaseException as e:  
 print("ERROR: ", str(e))  
 return False  
 return True  
  
  
# Insert some data to table. Selection of item is done with selection of some value in some field.  
def delete\_data(connection, cursor, table, column\_name, expected\_value):  
 q = sql.SQL("""  
 DELETE FROM {} WHERE {}=%s;  
 """).format(sql.Identifier(table), sql.Identifier(column\_name))  
  
 try:  
 cursor.execute(q, (expected\_value,))  
 connection.commit()  
 except BaseException as e:  
 print("ERROR: ", str(e))  
 return False  
 return True  
  
  
# Just random string with random length  
def random\_string():  
 random\_str = ""  
 for i in range(0, random.randint(5, 10)):  
 random\_str += str(random.choice("0123456789abcdefghijklmnopqrstuvwxyz"))  
 return random\_str  
  
  
# Get random value for some SQL type.  
# Unknown type => None  
def gen\_random(type\_v):  
 switcher = {  
 'integer':  
 lambda: random.randint(0, 16387),  
 'text':  
 lambda: random\_string(),  
 'bigint':  
 lambda: random.randint(0, 16387),  
 'boolean':  
 lambda: random.choice(['true', 'false']),  
 'timestamp with time zone':  
 # 2019-08-21 08:30:00+03:00  
 lambda: "%04d-%02d-%02d %02d:%02d:%02d+%02d:00" %  
 (random.randint(1970, 2037), # year  
 random.randint(1, 12), # month  
 random.randint(1, 28), # day  
 random.randint(0, 23), # hour  
 random.randint(0, 59), # minute  
 random.randint(0, 59), # second  
 random.randint(0, 11), # timezone  
 ),  
 }  
 return (switcher.get(type\_v, lambda: None))()  
  
  
# Do nothing  
def do\_nothing():  
 return None # do nothing

**view.py**

from prettytable import PrettyTable  
import getch  
import sys  
  
  
class View(object):  
  
 # This message is displayed on program start  
 def hello\_message(self):  
 print("Hello. You can always type 'back' and 'exit'. Enjoy.")  
  
 # Print start menu entries  
 def start\_menu(self):  
 print("\n::::::::::::::::::::::::::::::::::::::::")  
 print("\t1: View Tables")  
 print("\t2: SQL Query")  
 print("\t3: Find some")  
 print("\t4: Find in all tables by phraze")  
 print("\t5: Exit")  
  
 # Print some data. Data - list of tuples(table). No columns names specified.  
 def print\_data(self, table\_data, on\_none\_message=None):  
 if table\_data:  
 x = PrettyTable()  
 for row in table\_data:  
 x.add\_row(row)  
 print(x)  
 elif on\_none\_message:  
 print(on\_none\_message)  
  
 # Print menu entries for selection of some column from column list  
 def select\_column\_menu(self, table\_name, columns):  
 print("Select some field of table '", table\_name, "':", sep="")  
 counter = 1  
 for column in columns:  
 print("\t%d: %s" % (counter, column))  
 counter += 1  
  
 # Like print data, but with printing column names. There are in first element of 'table\_data'  
 def print\_table(self, table\_data, on\_none\_message=None):  
 if table\_data:  
 x = PrettyTable()  
 x.field\_names = table\_data[0]  
  
 for row in table\_data[1]:  
 x.add\_row(row)  
 print(x)  
 elif on\_none\_message:  
 print(on\_none\_message)  
  
 # Request some input from user. Validation can be done with list of valid cases or with lambda.  
 # 'back' and 'exit' are allways valid.  
 # 'exit' means exit from program  
 def request\_input(self,  
 message,  
 valid\_cases=[],  
 validator=None,  
 message\_on\_wrong="Wrong input, try again(or enter 'back'):"):  
 if validator is None:  
 validator = (lambda x: (any(x is s for s in valid\_cases))) if valid\_cases else (lambda \_: True)  
  
 print(message, end=" ")  
 while True:  
 try:  
 retval = input()  
 if retval == 'back' or validator(retval):  
 return retval  
 else:  
 if retval == 'exit':  
 print("Bye")  
 sys.exit(0)  
 else:  
 print(message\_on\_wrong, end=" ")  
 except KeyboardInterrupt:  
 print("\nInterrupted by user")  
 sys.exit()  
 except Exception as e:  
 print("Error on input:", e)  
  
 # Print tables list (menu entries)  
 def print\_tables(self, tables\_list):  
 counter = 1  
 for table in tables\_list:  
 print("\t", counter, ": ", table, sep="")  
 counter += 1  
  
 # Print some message and call 'getch'.  
 # User may have some time to rest before menu loop will continue.  
 def print\_and\_getch(self, message):  
 print(message)  
 return getch.getch()  
  
 # Print Ok if 'is\_all\_ok', else print 'FAIL'  
 # Then call 'getch'  
 def after\_action\_message(self, is\_all\_ok):  
 return self.print\_and\_getch("Ok" if is\_all\_ok else "FAIL")  
  
 # Print table actions menu entries  
 def print\_table\_menu(self, table\_name):  
 print("TABLE:", table\_name)  
 print("\t1: Select all")  
 print("\t2: Delete")  
 print("\t3: Insert")  
 print("\t4: Update")  
 print("\t5: Select where")  
 print("\t6: Insert random data")  
  
 # Print find menu entries  
 def find\_menu(self):  
 print("FIND MENU")  
 print("\t1: Order code, which have/does not-have code less than 500000")  
 print("\t2: Customer\_code, that have/doesn't have code more than 200000")  
 print("\t3: Orders, in time period")  
 print("\t4: Goods, which have/doesn`t have price more than 20")

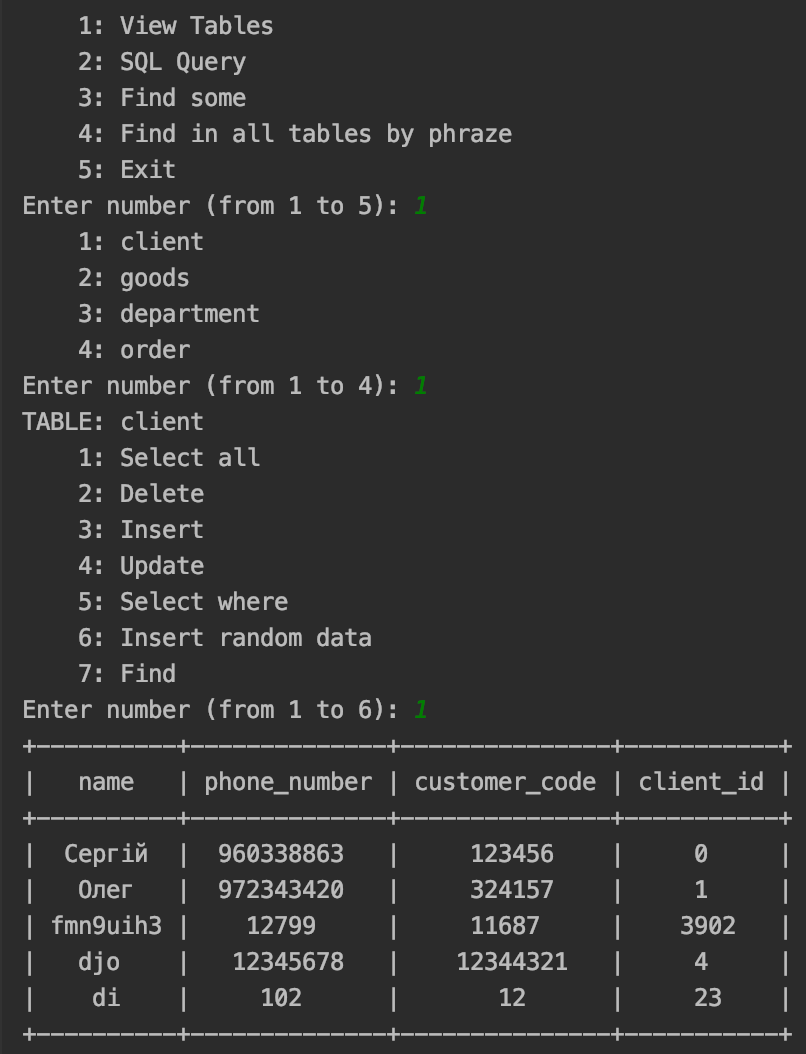
**model.py**

import utils as utils  
import psycopg2  
from psycopg2 import sql  
  
  
class Model(object):  
  
 def \_\_init\_\_(self, host, password):  
 self.open\_connection = lambda: utils.open\_connection(host, 5432, "Shop", "postgres", "9459298")  
  
 # Select some from some table  
 def select\_some(self, table\_name, column, value):  
 with self.open\_connection() as conn:  
 with conn.cursor() as cursor:  
 data = utils.query(cursor, sql.SQL("""  
 SELECT \* FROM {} WHERE {}=%s;  
 """).format(sql.Identifier(table\_name), sql.Identifier(column)), (value,))  
 if data:  
 return [utils.list\_table\_columns(cursor, table\_name), data]  
  
 # Update some in some table  
 # Details in 'utils'  
 def update(self, table\_name, column\_to\_check, exp\_value, new\_data):  
 with self.open\_connection() as conn:  
 with conn.cursor() as cursor:  
 return utils.update\_item(conn, cursor, table\_name, column\_to\_check, exp\_value, new\_data)  
  
 # Search phraze in all text columns of table  
 def find\_by\_phraze(self, table\_name, phraze):  
 with self.open\_connection() as conn:  
 with conn.cursor() as cursor:  
 def find\_by\_phraze(column):  
 return val  
  
 columns = utils.list\_table\_columns(cursor, table\_name)  
 columns = list(filter((lambda x: utils.get\_column\_type(cursor, table\_name, x) == 'text'), columns))  
 data = []  
 for column in columns:  
 val = utils.query(cursor, sql.SQL("""  
 SELECT \* FROM {} WHERE {} LIKE %s;  
 """).format(sql.Identifier(table\_name), sql.Identifier(column)),  
 ("%{}%".format(phraze),))  
 if val:  
 data.extend(val)  
 if data:  
 return [utils.list\_table\_columns(cursor, table\_name), list(dict.fromkeys(data))]  
  
 # Look in 'utils'  
 def list\_tables(self):  
 with self.open\_connection() as conn:  
 with conn.cursor() as cursor:  
 return utils.list\_tables(cursor)  
  
 # Look in 'utils'  
 def list\_columns(self, table\_name):  
 with self.open\_connection() as conn:  
 with conn.cursor() as cursor:  
 return utils.list\_table\_columns(cursor, table\_name)  
  
 # Get full table from database.  
 def get\_full\_table(self, table\_name):  
 with self.open\_connection() as conn:  
 with conn.cursor() as cursor:  
 columns = utils.list\_table\_columns(cursor, table\_name)  
 data = utils.query(cursor, sql.SQL("""  
 SELECT \* FROM {};  
 """).format(sql.Identifier(table\_name)), ())  
  
 if columns and data:  
 return [columns, data]  
  
 # Insert some random value to some table.  
 def insert\_random(self, table\_name):  
 with self.open\_connection() as connection:  
 with connection.cursor() as cursor:  
 def gen\_rand\_data(x):  
 return utils.gen\_random(utils.get\_column\_type(cursor, table\_name, x))  
  
 random\_data = tuple(gen\_rand\_data(x) for x in (utils.list\_table\_columns(cursor, table\_name)))  
 return utils.insert\_data(connection, cursor, table\_name, random\_data)  
  
 # Get column data from table.  
 # Returns list of tuples, where first element is column name and second - type of column  
 def get\_table\_columns\_data(self, table\_name):  
 with self.open\_connection() as connection:  
 with connection.cursor() as cursor:  
 columns = utils.list\_table\_columns(cursor, table\_name)  
 return list(zip(columns, map((lambda x: utils.get\_column\_type(cursor, table\_name, x)), columns)))  
  
 # Insert some data to some table  
 # Details in 'utils'  
 def insert\_data(self, table\_name, data):  
 with self.open\_connection() as connection:  
 with connection.cursor() as cursor:  
 return utils.insert\_data(connection, cursor, table\_name, data)  
  
 # Delete some data in database.  
 # Details in 'utils'  
 def delete\_data(self, table\_name, column\_name, expected\_value):  
 with self.open\_connection() as connection:  
 with connection.cursor() as cursor:  
 return utils.delete\_data(connection, cursor, table\_name, column\_name, expected\_value)  
  
 # Details in 'utils'  
 def query(self, q):  
 with self.open\_connection() as connection:  
 with connection.cursor() as cursor:  
 return utils.query(cursor, q, ())  
  
 # "Find menu" options  
  
 # Find all order codes which have/does not-have code less than 500000  
 def find\_1(self, using\_1):  
 with self.open\_connection() as connection:  
 if using\_1 is True:  
 with connection.cursor() as cursor:  
 data = utils.query(cursor, """  
 SELECT \* FROM "order"   
 WHERE order\_code < 500000;  
 """, (using\_1,))  
 if data:  
 return [utils.list\_table\_columns(cursor, 'order'), data]  
 elif using\_1 is False:  
 with connection.cursor() as cursor:  
 data = utils.query(cursor, """  
 SELECT \* FROM "order"   
 WHERE order\_code > 500000;  
 """, (using\_1,))  
 if data:  
 return [utils.list\_table\_columns(cursor, 'order'), data]  
  
 # Find client code, that have/doesn't have code more than 200000.  
 def find\_2(self, using\_2):  
 with self.open\_connection() as connection:  
 if using\_2 is True:  
 with connection.cursor() as cursor:  
 data = utils.query(cursor, """  
 SELECT \* FROM "client"   
 WHERE customer\_code > 200000  
 """, (using\_2,))  
 if data:  
 return [utils.list\_table\_columns(cursor, 'client'), data]  
 elif using\_2 is False:  
 with connection.cursor() as cursor:  
 data = utils.query(cursor, """  
 SELECT \* FROM "client"   
 WHERE customer\_code < 200000  
 """, (using\_2,))  
 if data:  
 return [utils.list\_table\_columns(cursor, 'client'), data]  
  
 # Find orders with time between two timestamps  
 def find\_3(self, start\_time, end\_time):  
 with self.open\_connection() as connection:  
 with connection.cursor() as cursor:  
 data = utils.query(cursor, """  
 SELECT \* FROM "order"   
 WHERE order\_data BETWEEN %s AND %s;  
 """, (start\_time, end\_time))  
 if data:  
 return [utils.list\_table\_columns(cursor, 'order'), data]  
  
 # Find all goods, that does not contains some word in registration place(country, for example)  
 def find\_4(self, using\_4):  
 with self.open\_connection() as connection:  
 if using\_4 is True:  
 with connection.cursor() as cursor:  
 data = utils.query(cursor, """  
 SELECT \* FROM "goods"   
 WHERE goods\_price > 20;  
 """, (using\_4, ))  
 if data:  
 return [utils.list\_table\_columns(cursor, 'goods'), data]  
 elif using\_4 is False:  
 with connection.cursor() as cursor:  
 data = utils.query(cursor, """  
 SELECT \* FROM "goods"   
 WHERE goods\_price < 20;  
 """, (using\_4, ))  
 if data:  
 return [utils.list\_table\_columns(cursor, 'goods'), data]

**controller.py**

import utils as utils  
  
  
class Controller(object):  
  
 # Very strange and not typical code  
 def \_\_init\_\_(self, model, view):  
 self.model = model  
 self.view = view  
  
 def start(self):  
 self.view.hello\_message()  
 self.show\_main\_menu()  
  
 # Main menu handler  
 def show\_main\_menu(self):  
 self.view.start\_menu()  
  
 input\_v = self.view.request\_input("Enter number (from 1 to 5):",  
 valid\_cases=['1', '2', '3', '4', '5'])  
  
 if input\_v == "1":  
 self.print\_tables()  
 self.show\_main\_menu()  
  
 if input\_v == "2":  
 sql\_query = self.view.request\_input("Enter query:")  
 self.view.print\_data(self.model.query(sql\_query), on\_none\_message="NOTHING")  
 self.show\_main\_menu()  
  
 if input\_v == "3":  
 self.find\_menu()  
  
 if input\_v == "4":  
 tables = self.model.list\_tables()  
 phraze = self.view.request\_input("Enter phraze:")  
 for table in tables:  
 print("On table %s:" % (table,))  
 self.view.print\_table(self.model.find\_by\_phraze(table, phraze), "NOTHING")  
 self.show\_main\_menu()  
  
 # Ask user to enter fields values of some object. Return None if problems  
 def request\_input\_object(self, table\_name, message="You can enter nothing for random value.", random\_on\_none=True):  
 data\_list = []  
 print(message)  
 for column\_data in self.model.get\_table\_columns\_data(table\_name):  
 data = self.view.request\_input("\tField '" + column\_data[0] + "'(" + column\_data[1] + "):")  
 if data == 'back':  
 return  
 elif random\_on\_none and data == '':  
 data\_list.append(utils.gen\_random(column\_data[1]))  
 else:  
 data\_list.append(data)  
 return data\_list  
  
 # Select table menu  
 def print\_tables(self):  
 tables = self.model.list\_tables()  
 self.view.print\_tables(tables)  
 input\_v = self.view.request\_input("Enter number (from 1 to " + str(len(tables)) + "):",  
 validator=lambda x: x.isdigit() and 0 < int(x) <= len(tables))  
 if input\_v == "back":  
 return None  
 self.table\_menu(tables[int(input\_v) - 1])  
  
 # Ask user to select some field of table and expected value on this field(for SELECT's)  
 def select\_obj\_menu(self, table\_name):  
 columns = self.model.list\_columns(table\_name)  
 self.view.select\_column\_menu(table\_name, columns)  
 input\_v = self.view.request\_input("Enter number (from 1 to " + str(len(columns)) + "):",  
 validator=lambda x: x.isdigit() and 0 < int(x) <= len(columns))  
 if input\_v != "back":  
 column = columns[int(input\_v) - 1]  
 expected\_value = self.view.request\_input("\tEnter expected value on field %s:" % (column,))  
 if expected\_value != "back":  
 return column, expected\_value  
 return None, None  
  
 # What can you do with table?  
 def table\_menu(self, table\_name):  
 self.view.print\_table\_menu(table\_name)  
 input\_v = self.view.request\_input("Enter number (from 1 to 6):",  
 valid\_cases=["1", "2", "3", "4", "5", "6"])  
  
 if input\_v != "back":  
 # SELECT ALL  
 if input\_v == "1":  
 data = self.model.get\_full\_table(table\_name)  
 self.view.print\_table(data)  
 self.view.after\_action\_message(data)  
 self.table\_menu(table\_name)  
  
 # DELETE  
 if input\_v == "2":  
 column, value = self.select\_obj\_menu(table\_name)  
 if column and value:  
 self.view.after\_action\_message(self.model.delete\_data(table\_name, column, value))  
 self.table\_menu(table\_name)  
  
 # INSERT  
 if input\_v == "3":  
 def insert():  
 self.view.after\_action\_message(self.model.insert\_data(table\_name,  
 tuple(self.request\_input\_object(table\_name))))  
  
 insert()  
 self.table\_menu(table\_name)  
  
 # UPDATE  
 if input\_v == "4":  
 self.update\_menu(table\_name)  
 self.table\_menu(table\_name)  
  
 # SELECT  
 if input\_v == "5":  
 column, value = self.select\_obj\_menu(table\_name)  
 if column and value:  
 data = self.model.select\_some(table\_name, column, value)  
 self.view.print\_table(data, on\_none\_message="NOTHING FOUND")  
 self.view.after\_action\_message(data)  
 self.table\_menu(table\_name)  
  
 # INSERT RANDOM  
 if input\_v == "6":  
 self.view.after\_action\_message(self.model.insert\_random(table\_name))  
 self.table\_menu(table\_name)  
  
 # Update menu  
 def update\_menu(self, table\_name):  
 print("SELECT OBJECT TO UPDATE:")  
 c, v = self.select\_obj\_menu(table\_name)  
 if c and v:  
 obj = self.request\_input\_object(table\_name, "Enter nothing for not touching field", False)  
 if obj:  
 self.view.after\_action\_message(self.model.update(table\_name, c, v, obj))  
  
  
 # TABLE DEPENDENCY ZONE  
  
 # Just find menu  
 def find\_menu(self):  
 self.view.find\_menu()  
 input\_v = self.view.request\_input("Enter number (from 1 to 4):",  
 valid\_cases=["1", "2", "3", "4"])  
 if input\_v != 'back':  
 if input\_v == '1':  
 data = self.view.request\_input("Enter param(boolean):")  
 if data != 'back' and data:  
 data = data.lower()  
 if data != 'true' and data != 'false' and data != 'none' and data != 'null':  
 print(data, "isn't bool")  
 else:  
 val = self.model.find\_1((data == 'true') if data == 'true' or data == 'false' else None)  
 self.view.print\_table(val, on\_none\_message="NOTHING FOUND")  
 self.view.after\_action\_message(val)  
  
 if input\_v == '2':  
 data = self.view.request\_input("Enter param(boolean):")  
 if data != 'back' and data:  
 data = data.lower()  
 if data != 'true' and data != 'false' and data != 'none' and data != 'null':  
 print(data, "isn't bool")  
 else:  
 val = self.model.find\_2((data == 'true') if data == 'true' or data == 'false' else None)  
 self.view.print\_table(val, on\_none\_message="NOTHING FOUND")  
 self.view.after\_action\_message(val)  
  
 if input\_v == '3':  
 data\_s = self.view.request\_input("Enter start\_time param(timestamp with time zone):")  
 if data\_s != 'back' and data\_s:  
 data\_e = self.view.request\_input("Enter end\_time param(timestamp with time zone):")  
 if data\_e != 'back' and data\_e:  
 val = self.model.find\_3(data\_s, data\_e)  
 self.view.print\_table(val, on\_none\_message="NOTHING FOUND")  
 self.view.after\_action\_message(val)  
  
 if input\_v == '4':  
 data = self.view.request\_input("Enter param(boolean):")  
 if data != 'back' and data:  
 data = data.lower()  
 if data != 'true' and data != 'false' and data != 'none' and data != 'null':  
 print(data, "isn't bool")  
 else:  
 val = self.model.find\_4((data == 'true') if data == 'true' or data == 'false' else None)  
 self.view.print\_table(val, on\_none\_message="NOTHING FOUND")  
 self.view.after\_action\_message(val)  
  
 self.find\_menu()

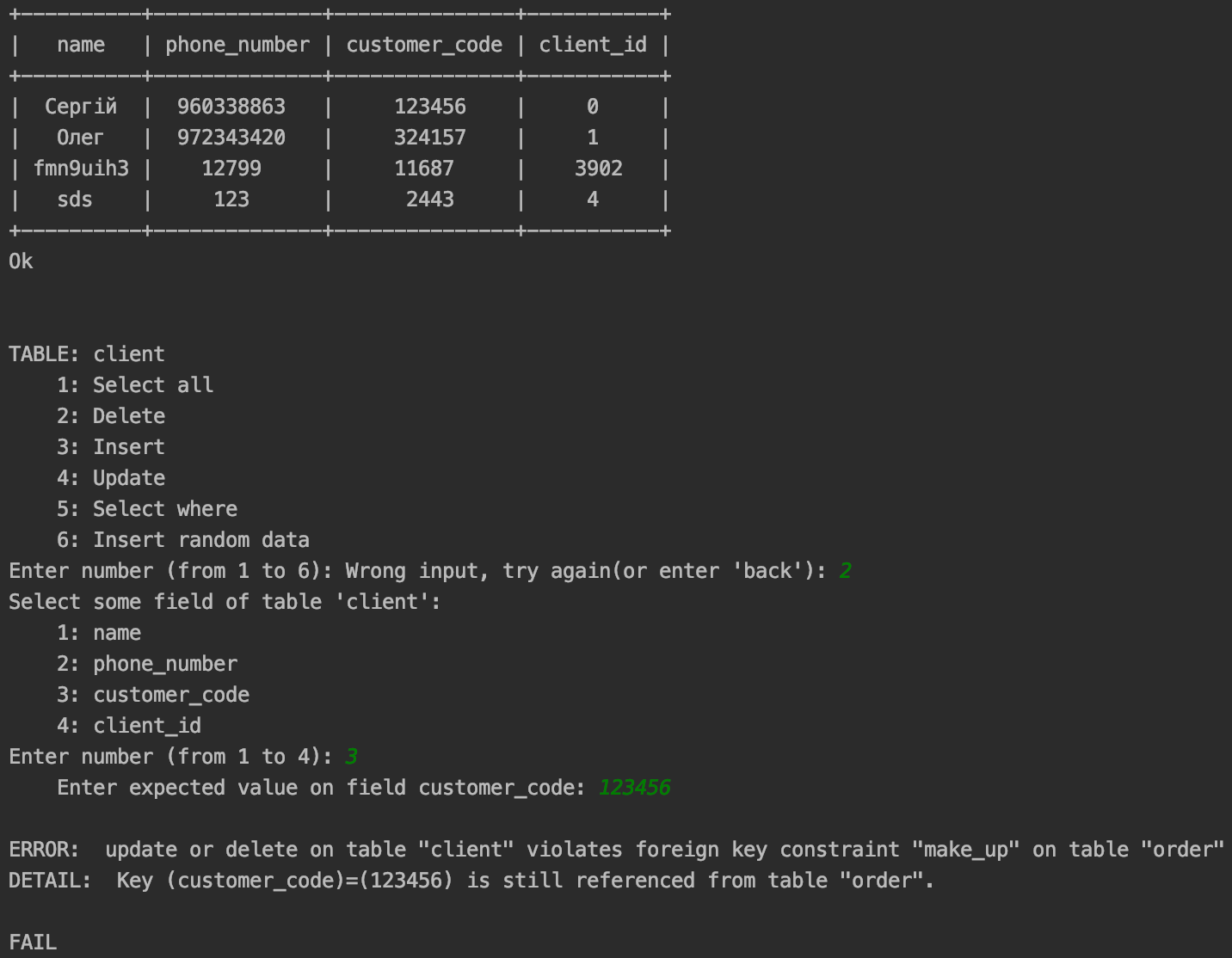
**Приклад роботи програми**

****

**\**

**Скріншот результатів виконання операції вилучення**

Спроба вилучення кортежу на який є посилання



Вдале видалення і демонстрація результату вилучення

