

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***»

Виконав: студент 3 курсу

ФПМ групи КВ-83

Гломозда Дмитро

Перевірив: Павловський В.І.

Київ – 2020

**Ознайомлення з базовими операціями СУБД PostgreSQL**

***Метою роботи*** є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

***Загальне*** ***завдання*** роботи полягає у наступному:

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з 2-х та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

***Вимоги до інтерфейсу користувача***

1. Використовувати консольний інтерфейс користувача.

***Варіант (предметна область):*** база даних ресторан

**Звіт**

**1. Нормалізована логічна модель даних БД**

Нижче наведено схему нормалізованої бази даних спроектованої в Лабораторній роботі №1.

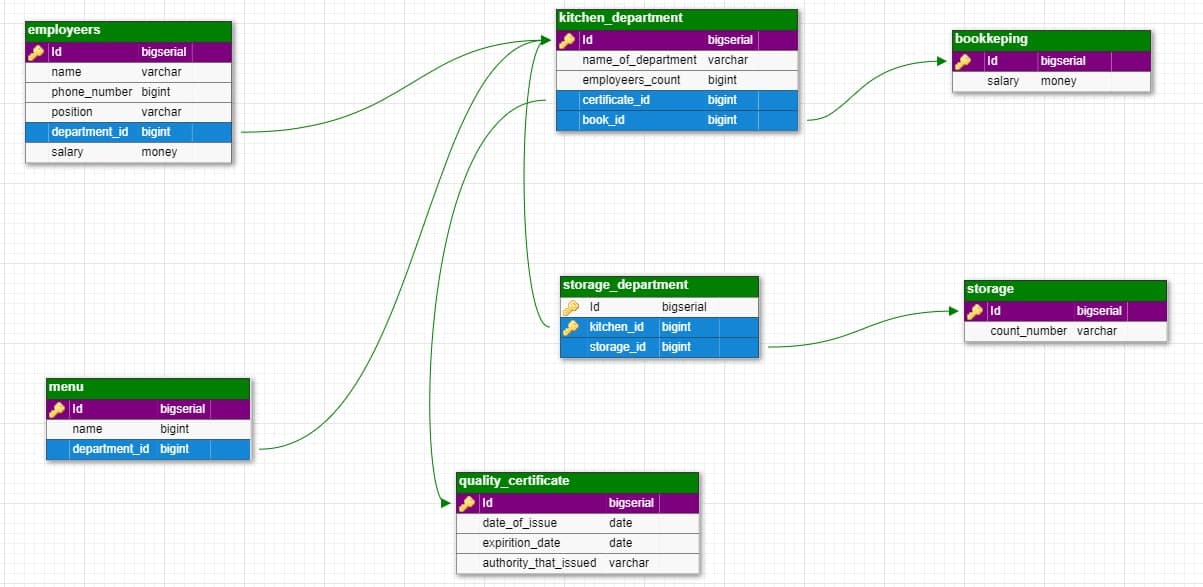


Рисунок 1.1 — Схема нормалізованої бази даних PostgreSQL на основі   
ER-моделі предметної області "Ресторан".

*Примітка*. При побудові схеми БД використано сервіс DBdesigner

Середовище розробки – Visual Studio Code. Мова програмування – Python 3.8. Бібліотека роботи з БД – psychopg2

**2.Опис програми**

Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL та реалізовує функціональні вимоги, що наведені у завданні. Програма складається з 5 модулів:

1. main.py — точка входу програми. Створення головного меню та підменю, яке являє собою відповідний контролер БД;
2. model.py — модуль містить опис класу Model, в якому виконується управління даними, логікою та правилами програми;
3. view.py — модуль містить опис класу View, який відповідає за обробку та виведення даних, отриманих в результаті запитів до БД;
4. controller.py — модуль містить опис класу Controller, який приймає введені користувачем дані і делегує представлення даних у View та обробку даних у Model;
5. backend.py — модуль, в якому виконується більшість бізнес-логіки за допомогою SQL-запитів;

**3.Структура меню програми**

Нижче наведено структуру меню програми.



Рисунок 3.1 — Головне меню програми.

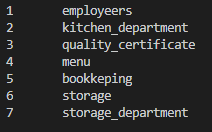


Рисунок 3.2 — Підменю для вибору таблиці

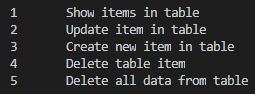
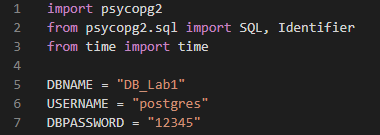


Рисунок 3.3 — Підменю для вибору типу запиту

**Лістинг програми для підключення до БД за допомогою бібліотек psycopg2**



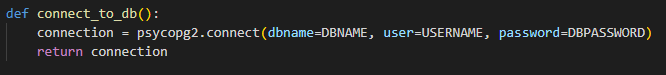
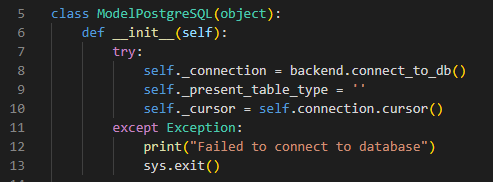


Рисунок 3.4 — Підключення до БД

1. **Лістинги програм з директивами внесення, редагування та вилучення даних у базі даних та результати виконання цих директив**

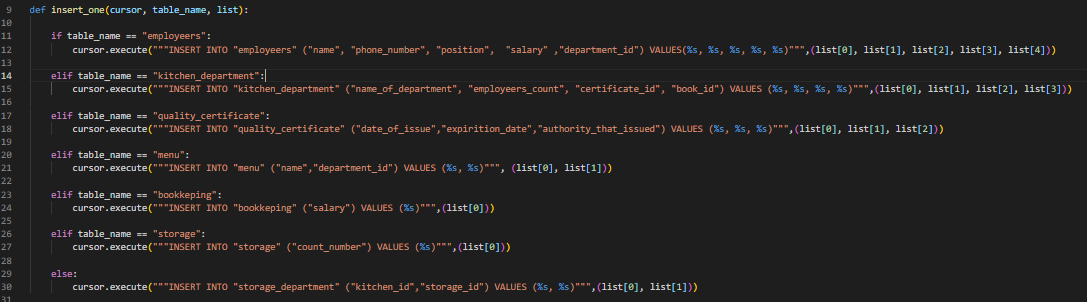


Рисунок 4.1 — Функція додавання запису у таблицю

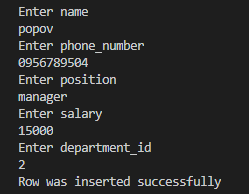
****

Рисунок 4.2 — Результат успішного внесення запису у таблицю "employeers"

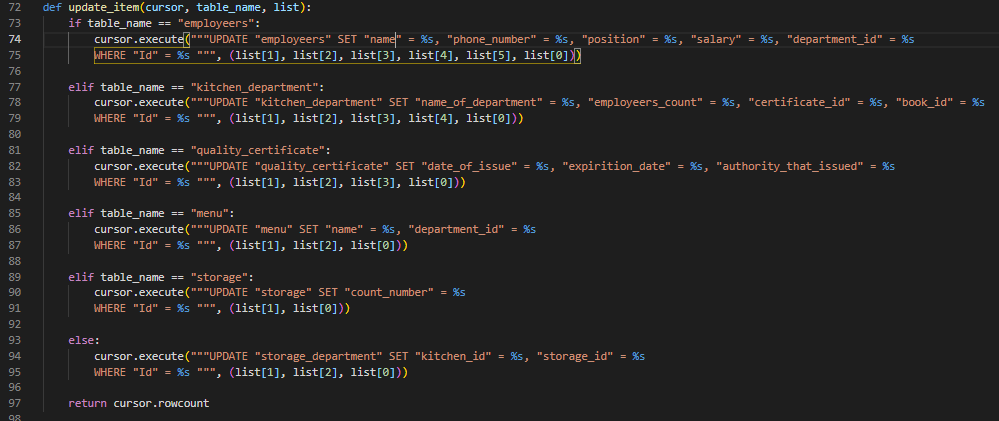


Рисунок 4.3 — Функція оновлення запису в таблиці

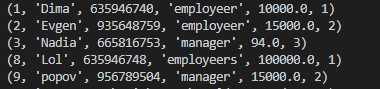
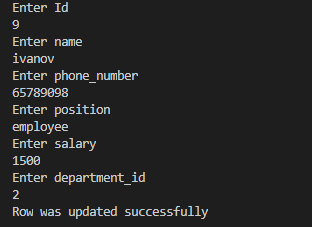


Рисунок 4.4 — Результат оновлення запису в таблиці "employeers"

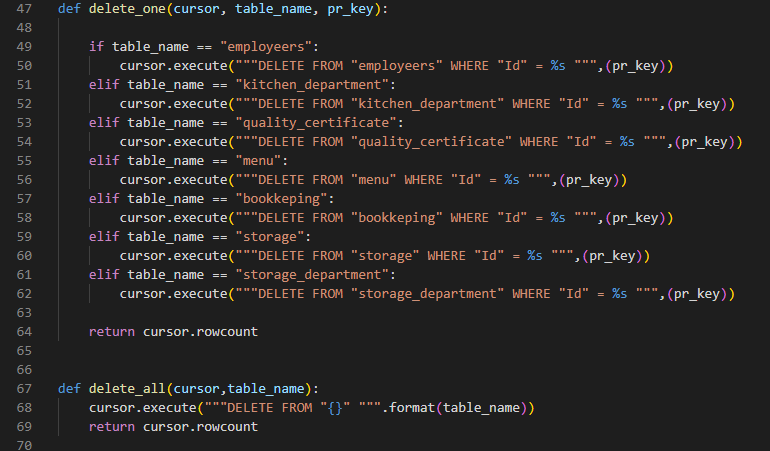
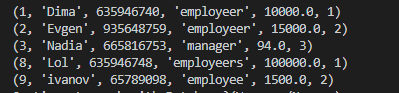
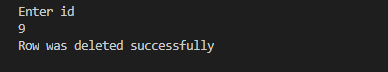


Рисунок 4.5 — Функція видалення запису в таблиці





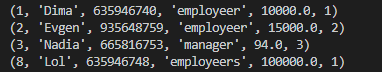


Рисунок 4.6 — Результат видалення запису в таблиці "Passenger"

**5.Лістинг програми зі статичним запитом пошуку**

Для демонстрації роботи пошуку :

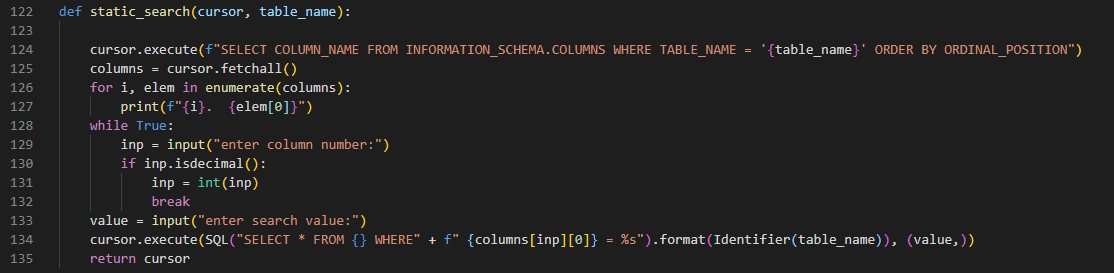


Рисунок 5.1 — Функція статичного запиту пошуку записів в таблиці

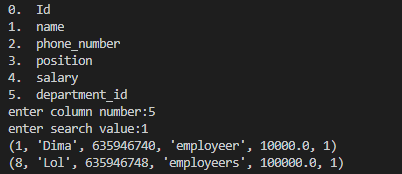


Рисунок 5.2 — Результат запиту пошуку записів в таблиці

**6.Лістинг програми генерування «рандомізованих» даних**

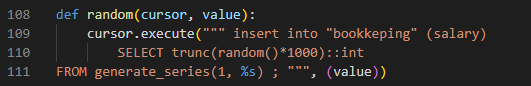


Рисунок 6.1 — Функція генерування «рандомізованих» даних



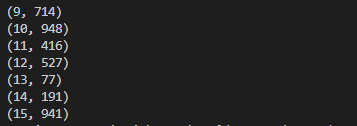


Рисунок 6.2 — Результат генерування 7 «рандомізованих» записів у таблицю "bookkeping"

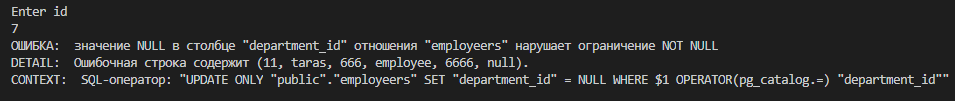
**7.Дослідження режимів обмеження ON DELETE**

*1. Режим CASCADE*

При видаленні запису з таблиці kitchlen\_department, запис з таблиці employeers видаляється.

*2. Режим SET NULL*

При видаленні запису з таблиці kitchen\_department, department\_id запис в таблиці employeers встановлюється в null. Якщо в налаштуваннях таблиці вказати, що department\_id не може бути null, то перехоплюємо повідомлення про помилку.

****

*3. Режим SET DEFAULT (значення за замовчуванням = 0)*

При видаленні запису з таблиці kitchen\_department, перехоплюємо повідомлення про помилку, так як kitchen\_department з id = 0 не існує.

****

*4. Режим NO ACTION*

При видаленні запису з таблиці kitchen\_department, виникає помилка; це поведінка за замовчуванням.



*5.Режим RESTRICT*

При видаленні запису з таблиці kitchen\_department, виникає помилка; це пояснюється тим, що режим RESTRICT не дає можливості видалити батьківський рядок, якщо в нього є дочірні.

****

**8.Структура програми**

Нижче наведено структуру програми та взаємодію окремих модулів.

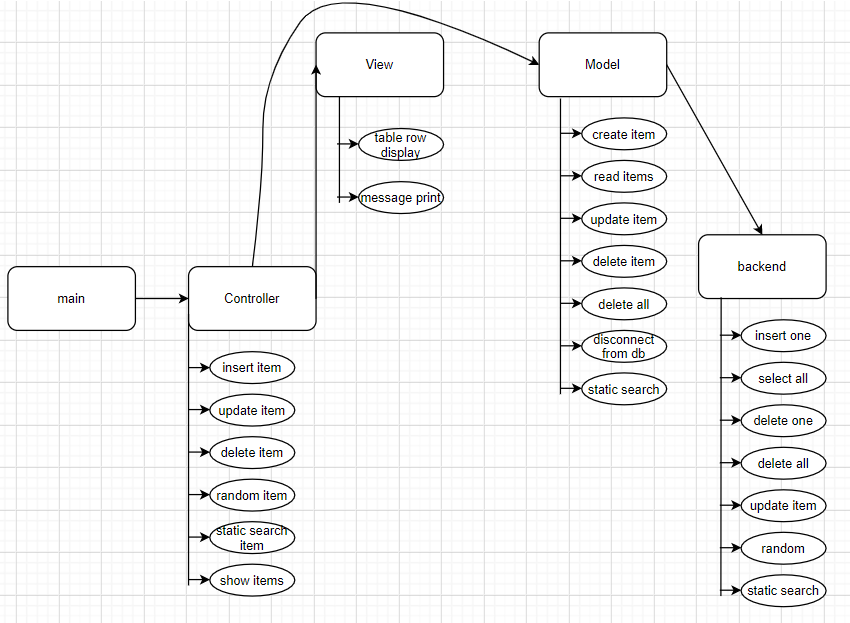
****

Рисунок 8.1 — Діаграма зв’язків модулів програми

**9.Код програми**

**main.py**

from controller import Controller

from model import ModelPostgreSQL

from view import View

if \_\_name\_\_ == '\_\_main\_\_':

    c = Controller(ModelPostgreSQL(),View())

    while True:

        type = c.action\_type\_select()

        if type == "1":

            c.table\_type\_select()

            c.action\_select()

        elif type == "2":

            c.static\_search()

        else:

            c.random\_insert()

        if not c.question\_about\_end():

            break

    c.disconnect\_from\_db()

**backend.py**

import psycopg2

from psycopg2.sql import SQL, Identifier

from time import time

DBNAME = "DB\_Lab1"

USERNAME = "postgres"

DBPASSWORD = "12345"

def insert\_one(cursor, table\_name, list):

    if table\_name == "employeers":

        cursor.execute("""INSERT INTO "employeers" ("name", "phone\_number", "position",  "salary" ,"department\_id") VALUES(%s, %s, %s, %s, %s)""",(list[0], list[1], list[2], list[3], list[4]))

    elif table\_name == "kitchen\_department":

        cursor.execute("""INSERT INTO "kitchen\_department" ("name\_of\_department", "employeers\_count", "certificate\_id", "book\_id") VALUES (%s, %s, %s, %s)""",(list[0], list[1], list[2], list[3]))

    elif table\_name == "quality\_certificate":

        cursor.execute("""INSERT INTO "quality\_certificate" ("date\_of\_issue","expirition\_date","authority\_that\_issued") VALUES (%s, %s, %s)""",(list[0], list[1], list[2]))

    elif table\_name == "menu":

        cursor.execute("""INSERT INTO "menu" ("name","department\_id") VALUES (%s, %s)""", (list[0], list[1]))

    elif table\_name == "bookkeping":

        cursor.execute("""INSERT INTO "bookkeping" ("salary") VALUES (%s)""",(list[0]))

    elif table\_name == "storage":

        cursor.execute("""INSERT INTO "storage" ("count\_number") VALUES (%s)""",(list[0]))

    else:

        cursor.execute("""INSERT INTO "storage\_department" ("kitchen\_id","storage\_id") VALUES (%s, %s)""",(list[0], list[1]))

def select\_all(cursor, table\_name):

    if table\_name == "employeers":

        cursor.execute(""" SELECT 'Id', name, phone\_number, position, concat(salary::numeric, '$') as salary, department\_id from "employeers" """)

    if table\_name == "quality\_certificate":

        cursor.execute(""" SELECT 'Id', to\_char(date\_of\_issue, 'YYYY-MM-DD'), to\_char(expirition\_date, 'YYYY-MM-DD'), authority\_that\_issued from "quality\_certificate" """)

    else:

        cursor.execute(""" SELECT \* FROM "{}" """.format(table\_name))

    return cursor

def delete\_one(cursor, table\_name, pr\_key):

    if table\_name == "employeers":

        cursor.execute("""DELETE FROM "employeers" WHERE "Id" = %s """,(pr\_key))

    elif table\_name == "kitchen\_department":

        cursor.execute("""DELETE FROM "kitchen\_department" WHERE "Id" = %s """,(pr\_key))

    elif table\_name == "quality\_certificate":

        cursor.execute("""DELETE FROM "quality\_certificate" WHERE "Id" = %s """,(pr\_key))

    elif table\_name == "menu":

        cursor.execute("""DELETE FROM "menu" WHERE "Id" = %s """,(pr\_key))

    elif table\_name == "bookkeping":

        cursor.execute("""DELETE FROM "bookkeping" WHERE "Id" = %s """,(pr\_key))

    elif table\_name == "storage":

        cursor.execute("""DELETE FROM "storage" WHERE "Id" = %s """,(pr\_key))

    elif table\_name == "storage\_department":

        cursor.execute("""DELETE FROM "storage\_department" WHERE "Id" = %s """,(pr\_key))

    return cursor.rowcount

def delete\_all(cursor,table\_name):

    cursor.execute("""DELETE FROM "{}" """.format(table\_name))

    return cursor.rowcount

def update\_item(cursor, table\_name, list):

    if table\_name == "employeers":

        cursor.execute("""UPDATE "employeers" SET "name" = %s, "phone\_number" = %s, "position" = %s, "salary" = %s, "department\_id" = %s

        WHERE "Id" = %s """, (list[1], list[2], list[3], list[4], list[5], list[0]))

    elif table\_name == "kitchen\_department":

        cursor.execute("""UPDATE "kitchen\_department" SET "name\_of\_department" = %s, "employeers\_count" = %s, "certificate\_id" = %s, "book\_id" = %s

        WHERE "Id" = %s """, (list[1], list[2], list[3], list[4], list[0]))

    elif table\_name == "quality\_certificate":

        cursor.execute("""UPDATE "quality\_certificate" SET "date\_of\_issue" = %s, "expirition\_date" = %s, "authority\_that\_issued" = %s

        WHERE "Id" = %s """, (list[1], list[2], list[3], list[0]))

    elif table\_name == "menu":

        cursor.execute("""UPDATE "menu" SET "name" = %s, "department\_id" = %s

        WHERE "Id" = %s """, (list[1], list[2], list[0]))

    elif table\_name == "storage":

        cursor.execute("""UPDATE "storage" SET "count\_number" = %s

        WHERE "Id" = %s """, (list[1], list[0]))

    else:

        cursor.execute("""UPDATE "storage\_department" SET "kitchen\_id" = %s, "storage\_id" = %s

        WHERE "Id" = %s """, (list[1], list[2], list[0]))

    return cursor.rowcount

def connect\_to\_db():

    connection = psycopg2.connect(dbname=DBNAME, user=USERNAME, password=DBPASSWORD)

    return connection

def disconnect\_from\_db(connection,cursor):

    cursor.close()

    connection.close()

    print("Connection with Data Base is closed")

def random(cursor, value):

    cursor.execute(""" insert into "bookkeping" (salary)

        SELECT trunc(random()\*1000)::int

FROM generate\_series(1, %s) ; """, (value))

def static\_search(cursor, table\_name):

    cursor.execute(f"SELECT COLUMN\_NAME FROM INFORMATION\_SCHEMA.COLUMNS WHERE TABLE\_NAME = '{table\_name}' ORDER BY ORDINAL\_POSITION")

    columns = cursor.fetchall()

    for i, elem in enumerate(columns):

        print(f"{i}.  {elem[0]}")

    while True:

        inp = input("enter column number:")

        if inp.isdecimal():

            inp = int(inp)

            break

    value = input("enter search value:")

    cursor.execute(SQL("SELECT \* FROM {} WHERE" + f" {columns[inp][0]} = %s").format(Identifier(table\_name)), (value,))

    return cursor

**view.py**

class View(object):

    @staticmethod

    def action\_type\_select\_display():

        print("Choose an action:\n")

        print("1\tDefinite action in definite table\n2\tStatic search\n3\tRandom")

    @staticmethod

    def table\_name\_select\_display():

        print("Choose table name:\n")

        print("1\temployeers\n2\tkitchen\_department\n3\tquality\_certificate\n4\tmenu\n5\tbookkeping\n6\tstorage\n7\tstorage\_department\n")

    @staticmethod

    def action\_select\_display():

        print("Choose an action with table:\n")

        print("1\tShow items in table\n2\tUpdate item in table\n3\tCreate new item in table\n"

        "4\tDelete table item\n5\tDelete all data from table\n")

    @staticmethod

    def enter\_cortege\_item\_display(item):

        print("Enter {}".format(item))

    @staticmethod

    def table\_rows\_display(items):

        cursor = items

        row = items.fetchone()

        while row is not None:

            print(row)

            row = cursor.fetchone()

    @staticmethod

    def question\_about\_end\_display():

        print("Continue to work with Database?(Yes - y/No - n)\n")

    @staticmethod

    def message\_print(message):

        print(message)

**model.py**

import backend

import sys

class ModelPostgreSQL(object):

    def \_\_init\_\_(self):

        try:

            self.\_connection = backend.connect\_to\_db()

            self.\_present\_table\_type = ''

            self.\_cursor = self.connection.cursor()

        except Exception:

            print("Failed to connect to database")

            sys.exit()

    @property

    def connection(self):

        return self.\_connection

    @property

    def cursor(self):

        return self.\_cursor

    @property

    def present\_table\_type(self):

        return self.\_present\_table\_type

    @present\_table\_type.setter

    def present\_table\_type(self,new\_present\_table\_type):

        self.\_present\_table\_type = new\_present\_table\_type

    def create\_item(self,cortage):

        backend.insert\_one(self.cursor,self.present\_table\_type,cortage)

    def read\_items(self):

        return backend.select\_all(self.cursor,self.present\_table\_type)

    def update\_item(self, list):

        backend.update\_item(self.cursor, self.present\_table\_type, list)

    def delete\_item(self,pr\_key):

        return backend.delete\_one(self.cursor,self.present\_table\_type,pr\_key)

    def delete\_all(self):

        return backend.delete\_all(self.cursor,self.present\_table\_type)

    def disconnect\_from\_db(self):

        backend.disconnect\_from\_db(self.connection,self.cursor)

    def random(self, value):

        backend.random(self.cursor, value)

    def static\_search(self, table\_name):

        return backend.static\_search(self.cursor, table\_name)

**controller.py**

import random

class Controller(object):

    def \_\_init\_\_(self, model, view):

        self.model = model

        self.view = view

    def show\_items(self):

        items = self.model.read\_items()

        if items.rowcount:

            self.view.table\_rows\_display(items)

            return

        self.view.message\_print("This table was already empty\n")

    def enter\_items(self, table\_item\_names):

        return\_array = []

        for name in table\_item\_names:

            self.view.enter\_cortege\_item\_display(name)

            inp = str(input())

            return\_array.append(inp)

        return return\_array

    def table\_type\_select(self):

        self.view.table\_name\_select\_display()

        while True:

            table\_type = str(input())

            if 0 < int(table\_type) < 7:

                if table\_type == "1":

                    self.model.present\_table\_type = "employeers"

                elif table\_type == "2":

                    self.model.present\_table\_type = "kitchen\_department"

                elif table\_type == "3":

                    self.model.present\_table\_type = "quality\_certificate"

                elif table\_type == "4":

                    self.model.present\_table\_type = "menu"

                elif table\_type == "5":

                    self.model.present\_table\_type = "bookkeping"

                elif table\_type == "6":

                    self.model.present\_table\_type = "storage"

                elif table\_type == "7":

                    self.model.present\_table\_type = "storage\_department"

                return

            self.view.message\_print("Error:enter number from 1 to 7\n")

    def action\_type\_select(self):

        self.view.action\_type\_select\_display()

        while True:

            action\_type = str(input())

            if action\_type != "1" and action\_type != "2" and action\_type != "3":

                self.view.message\_print("Error:enter number from 1 to 3\n")

                continue

            break

        return action\_type

    def action\_select(self):

        self.view.action\_select\_display()

        while True:

            action = str(input())

            if action == "1":

                self.show\_items()

            elif action == "2":

                self.update\_item()

            elif action == "3":

                self.insert\_item()

            elif action == "4":

                self.delete\_item()

            elif action == "5":

                self.delete\_all()

            else:

                self.view.message\_print("Error:Enter number from 1-5\n")

                continue

            break

        self.model.connection.commit()

    def question\_about\_end(self):

        self.view.question\_about\_end\_display()

        while True:

            inp = str(input())

            if inp == "Y" or inp == "y":

                return True

            elif inp == "N" or inp == "n":

                return False

            else:

                self.view.message\_print("""Error:enter "Y" or "N"\n """)

    def disconnect\_from\_db(self):

        self.model.disconnect\_from\_db()

    def insert\_item(self):

        while True:

            if self.model.present\_table\_type == 'employeers':

                list = self.enter\_items(("name", "phone\_number", "position", "salary",  "department\_id"))

            elif self.model.present\_table\_type == 'kitchen\_department':

                list = self.enter\_items(("name\_of\_department", "employeers\_count", "certificate\_id", "book\_id"))

            elif self.model.present\_table\_type == 'quality\_certificate':

                list = self.enter\_items(("date\_of\_issue","expirition\_date","authority\_that\_issued"))

            elif self.model.present\_table\_type == 'menu':

                list = self.enter\_items(("name","department\_id"))

            elif self.model.present\_table\_type == 'bookkeping':

                list = self.enter\_items(("salary",))

            elif self.model.present\_table\_type == 'storage':

                list = self.enter\_items(("count\_number",))

            else:

                list = self.enter\_items(("kitchen\_id","storage\_id"))

            try:

                self.model.create\_item(list)

                self.view.message\_print("Row was inserted successfully\n")

                break

            except Exception as error:

                print(error)

                break

            finally:

                self.model.connection.commit()

    def update\_item(self):

        while True:

            if self.model.present\_table\_type == 'employeers':

                list = self.enter\_items(("Id", "name", "phone\_number", "position", "salary", "department\_id"))

            elif self.model.present\_table\_type == 'kitchen\_department':

                list = self.enter\_items(("Id", "name\_of\_department", "employeers\_count", "certificate\_id", "book\_id"))

            elif self.model.present\_table\_type == 'quality\_certificate':

                list = self.enter\_items(("Id", "date\_of\_issue","expirition\_date","authority\_that\_issued"))

            elif self.model.present\_table\_type == 'menu':

                list = self.enter\_items(("Id", "name","department\_id"))

            elif self.model.present\_table\_type == 'bookkeping':

                list = self.enter\_items(("Id", "salary",))

            elif self.model.present\_table\_type == 'storage':

                list = self.enter\_items(("Id", "count\_number",))

            else:

                list = self.enter\_items(("Id", "kitchen\_id","storage\_id"))

            try:

                self.model.update\_item(list)

                self.view.message\_print("Row was updated successfully\n")

                break

            except Exception as error:

                print(error)

            finally:

                self.model.connection.commit()

    def delete\_item(self):

        id = self.enter\_items(["id"])

        try:

            if self.model.delete\_item(id):

                self.view.message\_print("Row was deleted successfully\n")

            else:

                self.view.message\_print("There isn't row for deleting with such attribute value\n")

        except Exception as error:

                print(error)

        finally:

                self.model.connection.commit()

    def delete\_all(self):

        if self.model.delete\_all():

            self.view.message\_print("All rows in table were deleted successfully\n")

        else:

            self.view.message\_print("Table was already empty\n")

    def random\_insert(self):

        self.view.message\_print("How many random records do you want to enter?\n")

        value = str(input())

        self.model.random(value)

        self.model.connection.commit()

    def static\_search(self):

        self.view.table\_name\_select\_display()

        while True:

            table\_type = str(input())

            if 0 < int(table\_type) < 7:

                if table\_type == "1":

                    name = "employeers"

                    break

                elif table\_type == "2":

                    name = "kitchen\_department"

                    break

                elif table\_type == "3":

                    name = "quality\_certificate"

                    break

                elif table\_type == "4":

                    name = "menu"

                    break

                elif table\_type == "5":

                    name = "bookkeping"

                    break

                elif table\_type == "6":

                    name = "storage"

                    break

                elif table\_type == "7":

                    name = "storage\_department"

                    break

        items = self.model.static\_search(name)

        if items.rowcount:

            self.view.table\_rows\_display(items)

            return

        self.view.message\_print("This table was already empty\n")