

**Національний технічний університет України  
“Київський політехнічний інститут імені Ігоря Сікорського”**

**Факультет прикладної математики  
Кафедра системного програмування і спеціалізованих  
комп’ютерних систем**

**Розрахунково-графічна робота**

*з дисципліни*

*“Бази даних та засоби управління”*

**ТЕМА: “ Створення додатку бази даних, орієнтованого на взаємодію з СУБД  
PostgreSQL”**

**Група: КВ-12**

**Виконала: Гнатюк С.В.**

**Оцінка:**

**Київ – 2023**

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

*Загальне завдання роботи полягає у наступному:*

1. Реалізувати функції перегляду, внесення, редагування та видалення 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).

### **Предметна галузь: Онлайн-сервіс для бронювання квитків на транспорт**

#### **Опис сутностей та зв'язків між ними**

Сутність User - користувач, який бронює квитки. Має char атрибути name (ім'я), surname (прізвище), phone (номер телефону) та key attribute user\_id (атрибут, значення якого не може повторюватися, оскільки він використовується для ідентифікації).

Сутність Vehicle - транспорт, на якому здійснюється поїздка. Має char атрибут vehicle\_type (тип транспорту, наприклад, автобус, поїзд тощо) та key attribute vehicle\_id.

Сутність Booking - бронювання. Має integer атрибут price (ціна поїздки), атрибут time типу time without timezone (час бронювання), key attribute booking\_id. Дана сутність пов'язана з двома іншими сутностями зв'язками 1:N, тобто один до багатьох. Один користувач може здійснювати багато бронювань, а один транспортний засіб можна бронювати багато разів. З іншого боку, одне бронювання може бути здійснене лише одним користувачем, і може стосуватися лише одного транспортного засобу.

Сутність staff – це працівник. Має ключовий атрибут staff\_id, character varying атрибути name (ім'я) та position (посада). Має 1:N зв'язок з сутністю Vehicle (транспорт), оскільки на одному транспортному засобі можуть працювати кілька

людей (наприклад, водій, провідник тощо), але один працівник працює лише на 1 транспортному засобі.

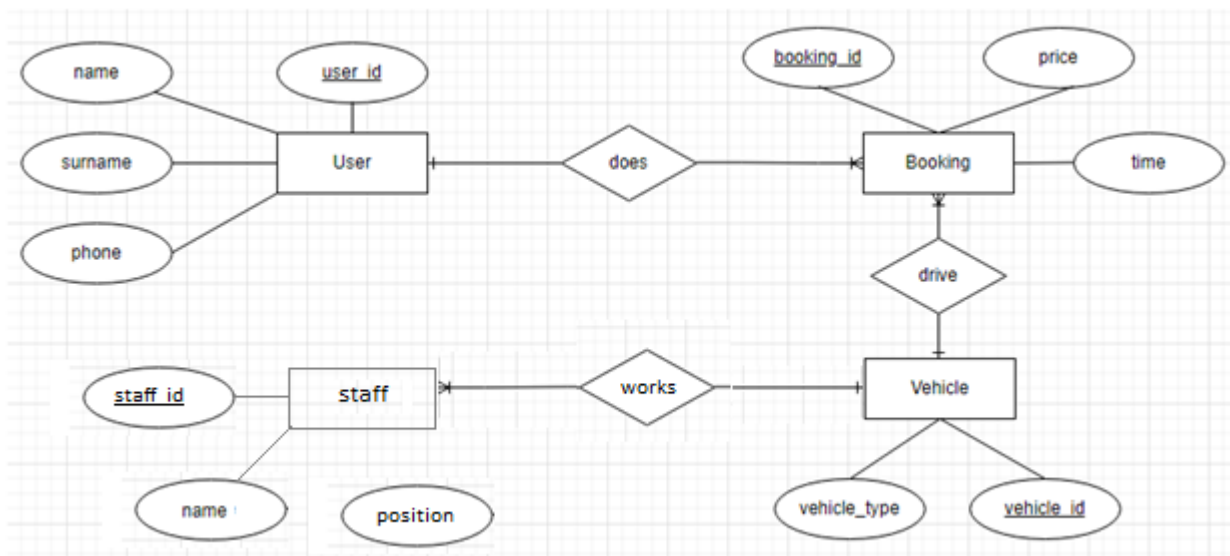


Рисунок 1 - ER-діаграма за нотацією Crow's foot

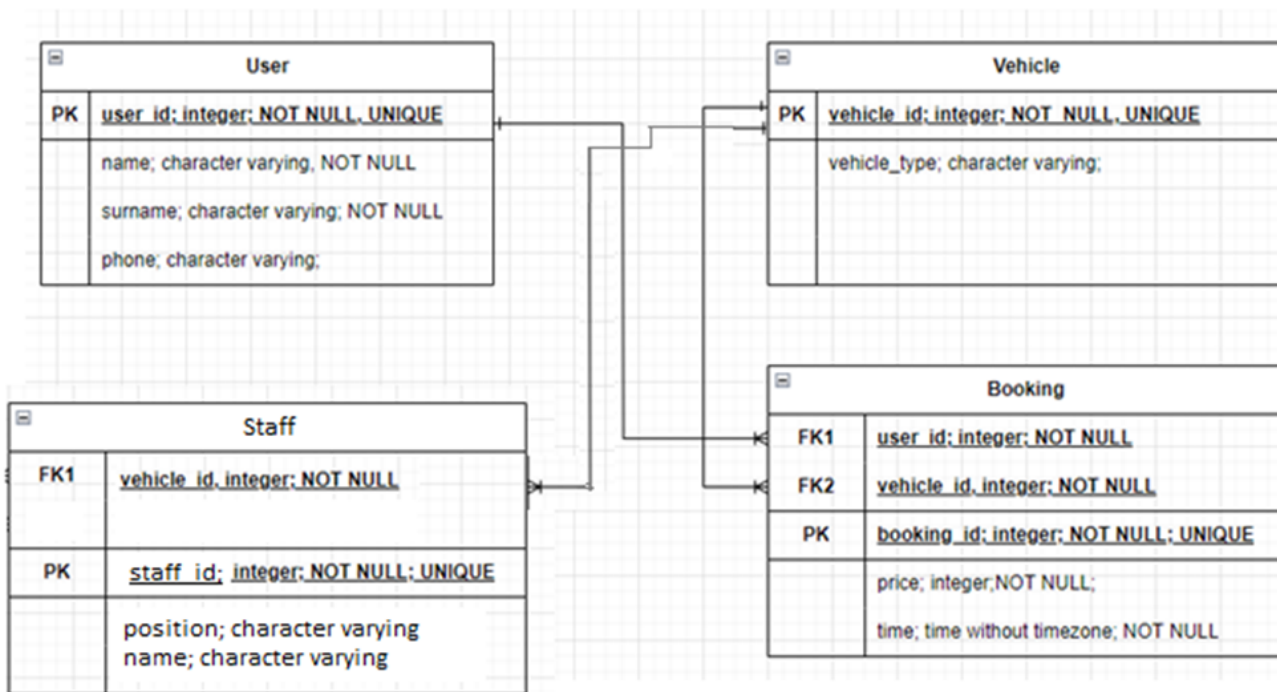


Рисунок 2 - Схема бази даних PostgreSQL на основі ER-моделі предметної галузі  
“Онлайн-сервіс бронювання квитків на транспорт”

## Мова програмування, середовище та компоненти розробки

В даній роботі використано мову програмування Python, середовище розробки програмного забезпечення PyCharm Community Edition, середовище для відлагодження SQL-запитів до бази даних pgAdmin4, стороння бібліотека psycopg2 для роботи з базами даних PostgreSQL в Python.

## Шаблон проєктування

У програмі використовується шаблон проєктування MVC (model, view, controller).

Model представляє логіку використання даних. View - інтерфейс, з яким буде взаємодіяти користувач. Controller забезпечує зв'язок між користувачем і системою; отримує введені користувачем дані і обробляє їх, а тоді надає користувачу певну інформацію.

## Файли програми

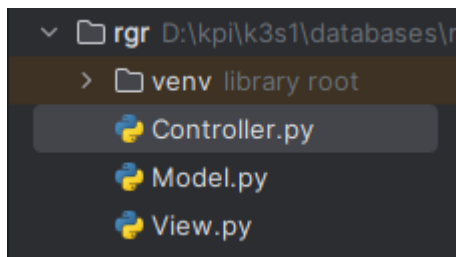


Рисунок 3 - структура файлів програми

У програмі використано 3 файли: Controller.py, Model.py, View.py. У файлі Model.py описано модель, яка виконує низькорівневі запити до бази даних. У файлі View.py описано інтерфейс користувача, зокрема меню і виведення інформації на

екран. У файлі Controller.py описана взаємодія між даними, які ввів користувач та запитами з файлу Model.py.

### Меню програми

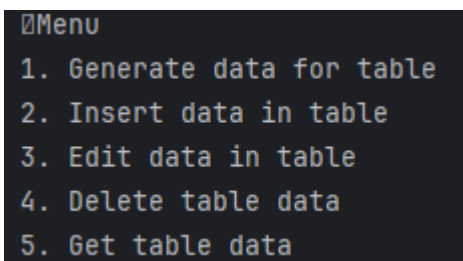


Рисунок 4 - меню програми

У меню програми є 5 опцій: генерувати дані таблиці, вставити дані в таблицю, змінити дані, видалити дані, отримати дані таблиці.

### Фрагмент програми для генерації даних

```

def generate_data(self, table_name, count):
    types = self.get_column_types(table_name)
    fk_array = self.get_foreign_key_info(table_name)
    select_subquery = ""
    insert_query = "INSERT INTO " + table_name + " ("
    for i in range(1, len(types)):
        t = types[i]
        name = t[0]
        type = t[1]
        fk = [x for x in fk_array if x[0] == name]
        if fk:
  
```

```

        select_subquery += '(SELECT {} FROM {} ORDER BY RANDOM(), ser LIMIT
1)'.format(fk[0][2], fk[0][1]))

        elif type == 'integer':

            select_subquery += 'trunc(random()*100)::INT'

        elif type == 'character varying':

            select_subquery += 'chr(trunc(65 + random()*25)::INT) || chr(trunc(65 +
random()*25)::INT)'

        elif type == 'time without time zone':

            select_subquery += "time '00:00:00' + DATE_TRUNC('second',RANDOM() * time
'24:00:00')"
```

```

        else:

            continue

        insert_query += name

        if i != len(types) - 1:

            select_subquery += ','

            insert_query += ','

        else:

            insert_query += ') '

        self.__cursor.execute(

            insert_query + "SELECT " + select_subquery + "FROM generate_series(1," +
str(count) + ") as ser")

        self.__context.commit()

```

### Фрагмент програми для редагування даних

```

def change_data(self, table_name, values):

    line = "

```

```

condition = values.pop('condition')
for key in values:
    if values[key]:
        line += key + '%(' + key + ')s,'
        self.__cursor.execute(
            sql.SQL('UPDATE {} SET ' + line[:-1] + ' WHERE {} ')
            .format(sql.Identifier(table_name), sql.SQL(condition)),
            values)
        self.__context.commit()

```

### Фрагмент программы для вставки данных

```

def insert_data(self, table_name, values):
    line = ""
    columns = '('
    for key in values:
        if values[key]:
            line += '%(' + key + ')s,'
            columns += key + ','
    columns = columns[:-1] + ')'

    self.__cursor.execute(sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')')
        .format(sql.Identifier(table_name), sql.SQL(columns)),
        values)
    self.__context.commit()

```

### Фрагмент програми для видалення даних

```
def delete_data(self, table_name, value, cond):
    self.__cursor.execute(
        sql.SQL('DELETE FROM {} WHERE {} = {}')
        .format(sql.Identifier(table_name), sql.Identifier(value), sql.SQL(cond)))
    self.__context.commit()
```

### Результати генерування даних таблиці

	vehicle_id [PK] integer	vehicle_type character varying (20)
1	1	boat
2	2	train
3	3	car

```
1. Generate data for table
2. Insert data in table
3. Edit data in table
4. Delete table data
5. Get table data
Choose an option: 1
Enter table name: vehicle
Enter count: 1
```

	vehicle_id [PK] integer	vehicle_type character varying (20)
1	1	boat
2	2	train
3	3	car
4	4	JQ

### Результати вставки даних в таблицю



	vehicle_id [PK] integer	vehicle_type character varying (20)
1	1	bus
2	2	train
3	3	car

```

Menu
1. Generate data for table
2. Insert data in table
3. Edit data in table
4. Delete table data
5. Get table data
Choose an option: 2
Enter table name: vehicle
Enter column names: vehicle_id
Enter values: 4

```

	vehicle_id [PK] integer	vehicle_type character varying (20)
1	1	bus
2	2	train
3	3	car
4	4	[null]



## Результати зміни даних таблиці

	vehicle_id [PK] integer	vehicle_type character varying (20)
1	1	bus
2	2	train
3	3	car
4	4	[null]



```

1. Generate data for table
2. Insert data in table
3. Edit data in table
4. Delete table data
5. Get table data
Choose an option: 3
Enter table name: vehicle
Enter column names: vehicle_type condition
Enter values: boat vehicle_id=1
['vehicle_id', 'vehicle_type']
[(1, 'boat'), (2, 'train'), (3, 'car'), (4, None)]

```

	vehicle_id [PK] integer 	vehicle_type character varying (20) 
1	1	boat
2	2	train
3	3	car
4	4	[null]

### Результати видалення даних таблиці

	vehicle_id [PK] integer 	vehicle_type character varying (20) 
1	1	boat
2	2	train
3	3	car
4	4	[null]

```

1. Generate data for table
2. Insert data in table
3. Edit data in table
4. Delete table data
5. Get table data
Choose an option: 4
Enter table name: vehicle
Enter column names: vehicle_id
Object id: 4
['vehicle_id', 'vehicle_type']
[(1, 'boat'), (2, 'train'), (3, 'car')]

```

	vehicle_id [PK] integer	vehicle_type character varying (20)
1	1	boat
2	2	train
3	3	car

### Код програми

View.py

```
import os
```

```
clear = lambda: os.system('cls')
```

```
def start():
```

```
    clear()
```

```
    print("Menu")
```

```
    print("1. Generate data for table")
```

```
    print("2. Insert data in table")
```

```
    print("3. Edit data in table")
```

```
print("4. Delete table data")
```

```
print("5. Get table data")
```

```
def output(data):
```

```
    for i in data:
```

```
        print(i)
```

Controller.py

```
import View
```

```
import Model
```

```
import time
```

```
while 1 == 1:
```

```
    View.start()
```

```
    choice = input("Choose an option: ")
```

```
    model = Model.db_model("lab", "postgres", "111", "")
```

```
    match choice:
```

```
        case "1":
```

```
            table = input("Enter table name: ")
```

```
            count = input("Enter count: ")
```

```
            model.generate_data(table, count)
```

```
            data = model.get_table_data(table)
```

```
            View.output(data)
```

```
            time.sleep(4)
```

```
        case "2":
```

```
            table = input("Enter table name: ")
```

```

columns = input("Enter column names: ").split(' ')
val = input("Enter values: ").split(' ')
values = {key: value for (key, value) in zip(columns, val)}
model.insert_data(table, values)
print("result:\n")
data = model.get_table_data(table)
View.output(data)
time.sleep(4)
case "3":
    table = input("Enter table name: ")
    columns = input("Enter column names: ").split(' ')
    val = input("Enter values: ").split(' ')
    values = {key: value for (key, value) in zip(columns, val)}
    model.change_data(table, values)
    data = model.get_table_data(table)
    View.output(data)
    time.sleep(4)
case "4":
    table = input("Enter table name: ")
    column = input("Enter column names: ")
    element = input("Object id: ")
    model.delete_data(table, column, element)
    data = model.get_table_data(table)
    View.output(data)
    time.sleep(4)
case "5":

```

```

table = input("Enter table name: ")
data = model.get_table_data(table)
View.output(data)
time.sleep(4)

```

Model.py

```

import psycopg2
from psycopg2 import sql
import time

```

```

class db_model():
    def __init__(self, dbname, user_name, password, host):
        self.__context = psycopg2.connect(dbname='postgres', user='postgres', password='2004',
        host='localhost')
        self.__cursor = self.__context.cursor()
        self.__table_names = None

    def __del__(self):
        self.__cursor.close()
        self.__context.close()

    def clear_transaction(self):
        self.__context.rollback()

```

```

def get_column_types(self, table_name):
    self.__cursor.execute("""SELECT column_name, data_type
FROM information_schema.columns
WHERE table_schema = 'public' AND table_name = %s
ORDER BY table_schema, table_name""", (table_name,))

    return self.__cursor.fetchall()

def get_foreign_key_info(self, table_name):
    self.__cursor.execute("""SELECT kcu.column_name, ccu.table_name AS
foreign_table_name,
ccu.column_name AS foreign_column_name
FROM information_schema.table_constraints AS tc
JOIN information_schema.key_column_usage AS kcu
ON tc.constraint_name = kcu.constraint_name
AND tc.table_schema = kcu.table_schema
JOIN information_schema.constraint_column_usage AS ccu
ON ccu.constraint_name = tc.constraint_name
AND ccu.table_schema = tc.table_schema
WHERE tc.constraint_type = 'FOREIGN KEY' AND
tc.table_name=%s;""", (table_name,))

    return self.__cursor.fetchall()

def get_table_data(self, table_name):
    id_column = self.get_column_types(table_name)[0][0]
    cursor = self.__cursor

    try:

```

```

        cursor.execute(
            sql.SQL('SELECT * FROM {} ORDER BY {} ASC').format(sql.Identifier(table_name),
            sql.SQL(id_column)))
    except Exception as e:
        return str(e)
    return ([col.name for col in cursor.description], cursor.fetchall())

```

```

def insert_data(self, table_name, values):

```

```

    line = ""
    columns = '('
    for key in values:
        if values[key]:
            line += '%(' + key + ')s,'
            columns += key + ','
    columns = columns[:-1] + ')'

```

```

    self.__cursor.execute(sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')')
        .format(sql.Identifier(table_name), sql.SQL(columns)),
        values)
    self.__context.commit()

```

```

def generate_data(self, table_name, count):

```

```

    types = self.get_column_types(table_name)
    fk_array = self.get_foreign_key_info(table_name)
    select_subquery = ""
    insert_query = "INSERT INTO " + table_name + " ("
    for i in range(1, len(types)):

```



```

t = types[i]
name = t[0]
type = t[1]
fk = [x for x in fk_array if x[0] == name]
if fk:
    select_subquery += '(SELECT {} FROM {} ORDER BY RANDOM(), ser LIMIT
1)'.format(fk[0][2], fk[0][1]))
    elif type == 'integer':
        select_subquery += 'trunc(random()*100)::INT'
    elif type == 'character varying':
        select_subquery += 'chr(trunc(65 + random()*25)::INT) || chr(trunc(65 +
random()*25)::INT)'
    elif type == 'time without time zone':
        select_subquery += "time '00:00:00' + DATE_TRUNC('second',RANDOM() * time
'24:00:00')"
    else:
        continue
insert_query += name
if i != len(types) - 1:
    select_subquery += ','
    insert_query += ';'
else:
    insert_query += ')'
self.__cursor.execute(
    insert_query + "SELECT " + select_subquery + "FROM generate_series(1," +
str(count) + ") as ser")
self.__context.commit()

```

```

def change_data(self, table_name, values):
    line = ""
    condition = values.pop('condition')
    for key in values:
        if values[key]:
            line += key + '=%( ' + key + ')s,'
            self.__cursor.execute(
                sql.SQL('UPDATE {} SET ' + line[:-1] + ' WHERE {} ')
                .format(sql.Identifier(table_name), sql.SQL(condition)),
                values)
            self.__context.commit()

def delete_data(self, table_name, value, cond):
    self.__cursor.execute(
        sql.SQL('DELETE FROM {} WHERE {} = {}')
        .format(sql.Identifier(table_name), sql.Identifier(value), sql.SQL(cond)))
    self.__context.commit()

def join_general(self, main_query, condition=""):
    new_cond = condition
    if condition:
        new_cond = "WHERE " + condition
    t1 = time.time()
    self.__cursor.execute(main_query.format(new_cond))
    t2 = time.time()

    return ((t2 - t1) * 1000, self.__cursor.fetchall())

```