A black and white drawing of a building

Description automatically generated

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського» ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Розрахунково-графічна робота**

з дисципліни **Бази даних і засоби управління**

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

Виконав:

студент ІІI курсу

групи КВ-12

Гусельніков А.О.

Перевірив:

Павловский В. І.

Київ – 2023

**Мета:** здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

**Завдання:**

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. 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).

**Виконання роботи**

**Сутності предметної області**

Згідно цієї області для побудови бази даних було виділено наступні сутності:

1. Організатори (Оrganizers).

Атрибути: ідентифікатор організатора, ім’я організатора, прізвище організатора, електронна пошта організатора.

Призначення: збереження даних щодо організаторів заходу.

1. Туристи (Tourists).

Атрибути: ідентифікатор туриста, ім’я туриста, прізвище туриста, електронна пошта туриста.

Призначення: збереження даних щодо туристів.

1. Заходи (Events).

Атрибути: ідентифікатор заходу, назва заходу, тип заходу, дата коли відбудеться, ідентифікатор адреси.

Призначення: збереження даних щодо заходів.

1. Номери телефонів (Phone\_nums).

Атрибути: ідентифікатор номеру телефону, номер телефону, ідентифікатор туриста, якому він належить.

Призначення: збереження даних щодо контактних номерів телефонів туристів.

**Зв’язки між сутностями предметної області**

1. Зв’язок «Оrganizers» - «Events» є зв’язком N:M. Організатор може мати багато створених заходів, а може й не мати, у одного заходу може бути один або більше організаторів.
2. Зв’язок «Tourists» - «Events» є зв’язком N:M. Турист може забронювати місця у багатьох заходах, а може й не забронювати, захід може мати багато заброньованих місць, а може й не мати.
3. Зв’язок «Tourists» - «Phone\_nums» є зв’язком 1:N. Кожен турист має ввести хоча б один контактний номер телефону та номер телефону має належати тільки одній людині.

Графічне подання концептуальної моделі «Сутність-зв’язок» зображено на рисунку 1.

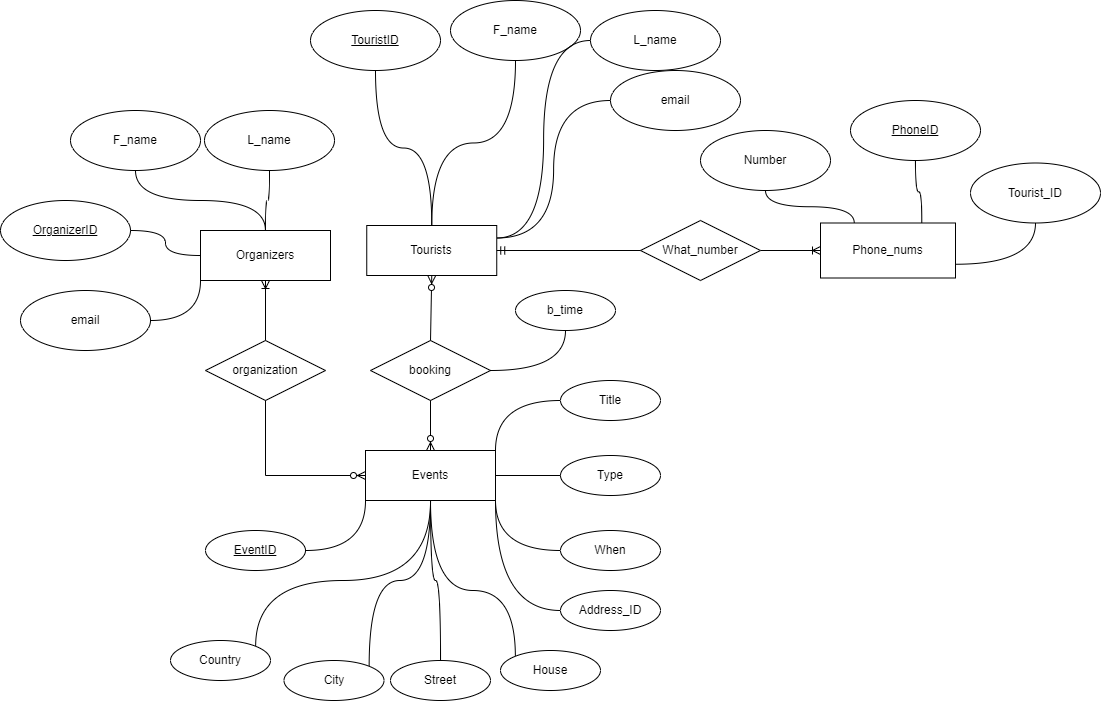


Рисунок 1 – ER-діаграма, побудована за нотацією Чена

Графічне подання логічної моделі «Сутність-зв’язок» зображено на рисунку 2.

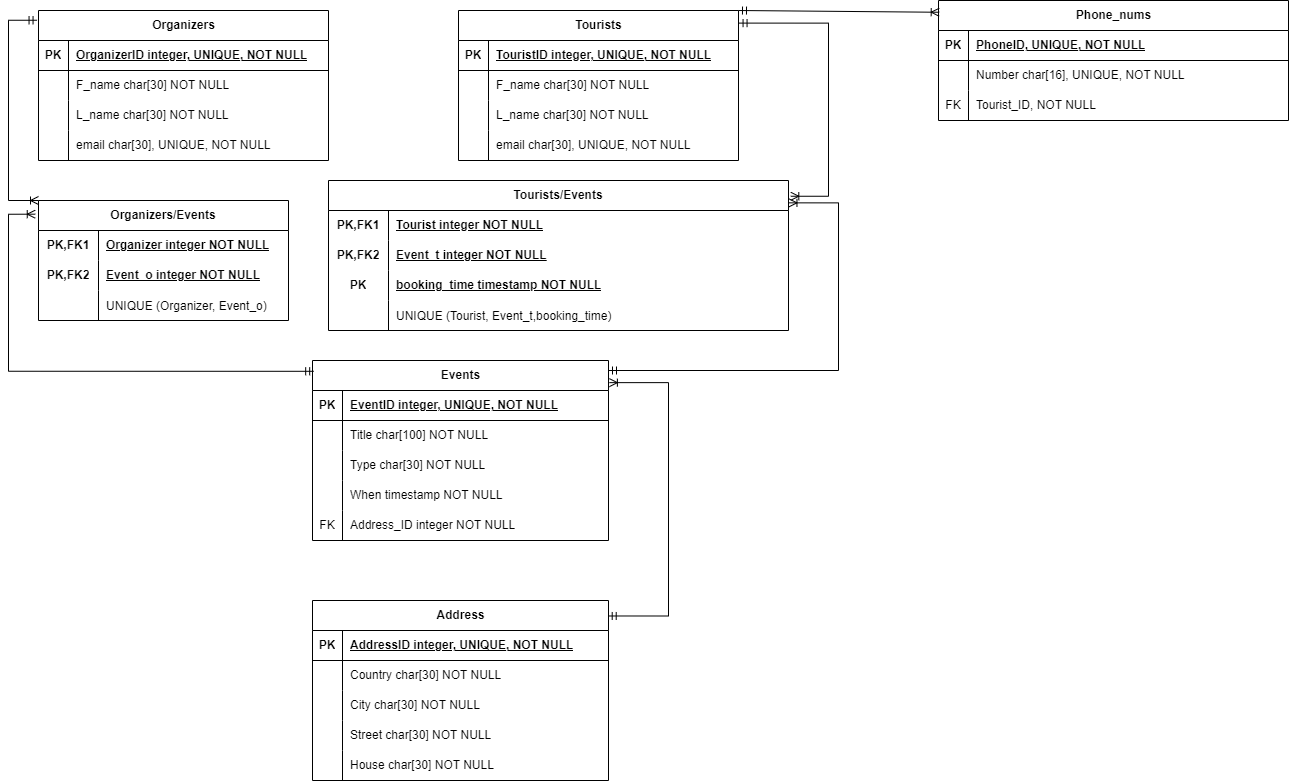


Рисунок 2 – Схема бази даних

**Середовище та компоненти розробки**

Для розробки використовувалась мова програмування Python, середовище розробки PyCharm, а також стороння бібліотека, що надає API для доступу до PostgreSQL – psycopg2.

**Шаблон проектування**

MVC - Шаблон проектування, який використаний у програмі.

Model – представляє клас, що описує логіку використовуваних даних. Згідно компоненту моделі, у моїй програмі відповідають всі компоненти які знаходять у папці Models.

View – в нашому випадку консольний інтерфейс з яким буде взаємодіяти наш користувач. Згідно компоненту представлення, то їй відповідають такі компоненти, згідно яким користувач бачить необхідні дані, що є представленням даних у вигляді консольного інтерфейсу.

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує вводяться користувачем дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок, наприклад, у вигляді подання.

**Структура програми та її опис**

****

Рисунок 3 - Структура програми

Програма умовно поділена на 3 модулі:

1. файл Controller.py;
2. файл Model.py;
3. файл View.py.

Та файл Main.py, який запускає програму

Модулі, як видно з їх назв, повністю відповідають використаному патерну MVC.

У файлі Model (.py) описаний клас моделі, що займається регулюванням підключення до бази даних, та виконанням низькорівневих запитів до неї.

У файлі Controller (.py) описаний інтерфейс взаємодії з користувачем, запит бажаної дії, виконання пошуку, тощо.

У файлі View (.py) описаний клас, що виводить результати виконання тієї чи іншої дії на екран консолі.

**Структура меню програми**

Меню для користувача складається з 8-ми пунктів (Рисунок 4).

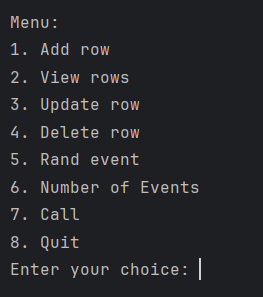


Рисунок 4 - Меню програми

**Фрагмент коду (файл main.py), в якому наведено виклик функції run()**

1. **from** controller **import** Controller
3. **if** \_\_name\_\_ **==** "\_\_main\_\_":
4. controller **=** Controller()
5. controller.run()

**Фрагмент коду (файл model.py), який відповідає за валідацію введених користувачем даних та перевірку на існування потрібних ідентифікаторів**

1. **def** is\_int(self, value):
2. **return** value.isdigit()
4. **def** is\_date(self, input\_string):
5. **try**:
6. datetime.strptime(input\_string, '%Y-%m-%d %H:%M:%S')
7. **return** True
8. **except** ValueError:
9. **return** False
10. **def** check\_str(self, value, length):
11. **if** len(value) <**=** length **and** len(value) >**=** 1:
12. **return** True
13. **else**:
14. **return** False

17. **def** is\_email(self,email):
18. pattern **=** r"^[-\w\.]+@([-\w]+\.)+[-\w]{2,4}$"
19. **if** re.match(pattern, email) **is** **not** None:
20. **return** True
21. **else**:
22. **return** False
23. **def** is\_correct\_address\_id(self, id):
24. c **=** self.conn.cursor()
25. **if** self.is\_int(id):
26. c.execute('SELECT COUNT(\*) FROM "Address" WHERE "AddressID" = %s', (id,))
27. r **=** c.fetchone()
28. **if** r[0] **==** 0:
29. **return** False
30. **elif** r[0] **==** 1:
31. **return** True
32. **else**:
33. **return** False
35. **def** is\_correct\_organizer\_id(self, id):
36. c **=** self.conn.cursor()
37. **if** self.is\_int(id):
38. c.execute('SELECT COUNT(\*) FROM "Оrganizers" WHERE "ОrganizerID" = %s', (id,))
39. r **=** c.fetchone()
40. **if** r[0] **==** 0:
41. **return** False
42. **elif** r[0] **==** 1:
43. **return** True
44. **else**:
45. **return** False
47. **def** is\_correct\_event\_id(self, id):
48. c **=** self.conn.cursor()
49. **if** self.is\_int(id):
50. c.execute('SELECT COUNT(\*) FROM "Events" WHERE "EventID" = %s', (id,))
51. r **=** c.fetchone()
52. **if** r[0] **==** 0:
53. **return** False
54. **elif** r[0] **==** 1:
55. **return** True
56. **else**:
57. **return** False
58. **def** is\_correct\_tourist\_id(self, id):
59. c **=** self.conn.cursor()
60. **if** self.is\_int(id):
61. c.execute('SELECT COUNT(\*) FROM "Tourists" WHERE "TouristID" = %s', (id,))
62. r **=** c.fetchone()
63. **if** r[0] **==** 0:
64. **return** False
65. **elif** r[0] **==** 1:
66. **return** True
67. **else**:
68. **return** False
69. **def** is\_correct\_phone\_id(self, id):
70. c **=** self.conn.cursor()
71. **if** self.is\_int(id):
72. c.execute('SELECT COUNT(\*) FROM "Phone\_nums" WHERE "PhoneID" = %s', (id,))
73. r **=** c.fetchone()
74. **if** r[0] **==** 0:
75. **return** False
76. **elif** r[0] **==** 1:
77. **return** True
78. **else**:
79. **return** False

**Фрагмент коду (файл model.py), який відповідає за додавання рядку в таблицю Organizers**

1. **def** org\_tour\_check(self,f\_name, l\_name, email):
2. **if** self.check\_str(f\_name,30):
3. **if** self.check\_str(l\_name,30):
4. **if** self.check\_str(email,30):
5. **if** self.is\_email(email):
6. **return** True, 'Ok'
7. **else**:
8. **return** False, 'ERROR: wrong email style'
9. **else**:
10. **return** False, 'ERROR: wrong email length'
11. **else**:
12. **return** False, 'ERROR: wrong l\_name length'
13. **else**:
14. **return** False, 'ERROR: wrong f\_name length'
15. **def** add\_organizer(self, f\_name, l\_name, email):
16. c **=** self.conn.cursor()
17. b, strr **=** self.org\_tour\_check(f\_name, l\_name, email)
18. **if** b:
19. c.execute('INSERT INTO "Оrganizers" ("F\_name", "L\_name", email) VALUES (%s, %s, %s)',(f\_name, l\_name, email))
20. self.conn.commit()
21. **return** b,strr

**Фрагмент коду (файл model.py), який відповідає за додавання рядку в таблицю Tourists**

1. **def** add\_tourist(self, f\_name, l\_name, email):
2. c **=** self.conn.cursor()
3. b, strr **=** self.org\_tour\_check(f\_name, l\_name, email)
4. **if** b:
5. c.execute('INSERT INTO "Tourists" ("F\_name", "L\_name", email) VALUES (%s, %s, %s)', (f\_name, l\_name, email))
6. self.conn.commit()
7. **return** b, strr

**Фрагмент коду (файл model.py), який відповідає за додавання рядку в таблицю Events**

1. **def** event\_check(self,title, type\_, date, address\_id):
2. **if** self.is\_correct\_address\_id(address\_id):
3. **if** self.check\_str(title,100):
4. **if** self.check\_str(type\_,30):
5. **if** self.is\_date(date):
6. **return** True, 'Ok'
7. **else**:
8. **return** False, 'ERROR: wrong date style'
9. **else**:
10. **return** False, 'ERROR: wrong type length'
11. **else**:
12. **return** False, 'ERROR: wrong title length'
13. **else**:
14. **return** False, 'ERROR: wrong Address\_ID'
16. **def** add\_event(self, title, type\_, date, address\_id):
17. c **=** self.conn.cursor()
18. b, strr **=** self.event\_check(title, type\_, date, address\_id)
19. **if** b:
20. c.execute('INSERT INTO "Events" ("Title", "Type", "Date", "Address\_ID") VALUES (%s, %s, %s, %s)', (title, type\_, date, address\_id))
21. self.conn.commit()
22. **return** b,strr

**Фрагмент коду (файл model.py), який відповідає за додавання рядку в таблицю Phone\_nums**

1. **def** phone\_check(self, tourist\_id, number):
2. **if** self.is\_correct\_tourist\_id(tourist\_id):
3. **if** self.check\_str(number,16):
4. **return** True, 'Ok'
5. **else**:
6. **return** False, 'ERROR: wrong number length'
7. **else**:
8. **return** False, 'ERROR: wrong Tourist\_ID'
9. **def** add\_phone\_num(self, tourist\_id, number):
10. c **=** self.conn.cursor()
11. b, strr **=** self.phone\_check(tourist\_id, number)
12. **if** b:
13. c.execute('INSERT INTO "Phone\_nums" ("Tourist\_ID", "Number") VALUES (%s, %s)',(tourist\_id, number))
14. self.conn.commit()
15. **return** b,strr

**Фрагмент коду (файл model.py), який відповідає оновлення рядку в таблицях**

1. **def** update\_tour(self, id, f\_name, l\_name, email):
2. c **=** self.conn.cursor()
3. **if** self.is\_correct\_tourist\_id(id):
4. b,strr **=** self.org\_tour\_check(f\_name, l\_name, email)
5. **if** b:
6. c.execute('UPDATE "Tourists" SET "F\_name"=%s, "L\_name"=%s, email=%s WHERE "TouristID"=%s', (f\_name, l\_name, email, id))
7. self.conn.commit()
8. **return** b,strr
9. **else**:
10. **return** False, 'ERROR: wrong TouristID'
12. **def** update\_org(self, id, f\_name, l\_name, email):
13. c **=** self.conn.cursor()
14. **if** self.is\_correct\_organizer\_id(id):
15. b, strr **=** self.org\_tour\_check(f\_name, l\_name, email)
16. **if** b:
17. c.execute('UPDATE "Оrganizers" SET "F\_name"=%s, "L\_name"=%s, email=%s WHERE "ОrganizerID"=%s', (f\_name, l\_name, email, id))
18. self.conn.commit()
19. **return** b,strr
20. **else**:
21. **return** False, 'ERROR: wrong ОrganizerID'
23. **def** update\_event(self, id, title, type\_, date, address\_id):
24. c **=** self.conn.cursor()
25. **if** self.is\_correct\_event\_id(id):
26. b, strr **=** self.event\_check(title, type\_, date, address\_id)
27. **if** b:
28. c.execute('UPDATE "Events" SET "Title"=%s, "Type"=%s,"Date"=%s, "Address\_ID"=%s WHERE "EventID"=%s', (title, type\_, date, address\_id, id))
29. self.conn.commit()
30. **return** b,strr
31. **else**:
32. **return** False, 'ERROR: wrong EventID'
34. **def** update\_phone(self, id, tourist\_id, number):
35. c **=** self.conn.cursor()
36. **if** self.is\_correct\_phone\_id(id):
37. b, strr **=** self.phone\_check(tourist\_id, number)
38. **if** b:
39. c.execute('UPDATE "Phone\_nums" SET "Tourist\_ID"=%s, "Number"=%s WHERE "PhoneID"=%s', (tourist\_id, number, id))
40. self.conn.commit()
41. **return** b, strr
42. **else**:
43. **return** False, 'ERROR: wrong PhoneID'

**Фрагмент коду (файл model.py), який відповідає видалення рядку в таблицях**

1. **def** delete\_p(self, id):
2. c **=** self.conn.cursor()
3. **if** self.is\_correct\_phone\_id(id):
4. c.execute('DELETE FROM "Phone\_nums" WHERE "PhoneID"=%s', (id,))
5. self.conn.commit()
6. **return** True
7. **else**:
8. **return** False
9. **def** cnt\_t\_fk(self,id):
10. c **=** self.conn.cursor()
11. c.execute('SELECT COUNT(\*) FROM "Tourists/Events" WHERE "Tourist" = %s', (id,))
12. t\_e **=** c.fetchone()
13. c.execute('SELECT COUNT(\*) FROM "Phone\_nums" WHERE "Tourist\_ID" = %s', (id,))
14. p **=** c.fetchone()
15. **if** t\_e[0] > 0 **or** p[0] > 0:
16. **return** True, t\_e[0], p[0]
17. **else**:
18. **return** False,t\_e[0],p[0]
19. **def** get\_t\_e(self, id):
20. c **=** self.conn.cursor()
21. c.execute('SELECT \* FROM "Tourists/Events" WHERE "Tourist" = %s',  (id,))
22. **return** c.fetchall()
24. **def** get\_p(self, id):
25. c **=** self.conn.cursor()
26. c.execute('SELECT \* FROM "Phone\_nums" WHERE "Tourist\_ID" = %s',  (id,))
27. **return** c.fetchall()
28. **def** delete\_t(self, id):
29. c **=** self.conn.cursor()
30. **if** self.is\_correct\_tourist\_id(id):
31. c.execute('DELETE FROM "Tourists" WHERE "TouristID"=%s', (id,))
32. self.conn.commit()
33. **return** True
34. **else**:
35. **return** False


39. **def** cnt\_o\_fk(self,id):
40. c **=** self.conn.cursor()
41. c.execute('SELECT COUNT(\*) FROM "Оrganizers/Events" WHERE "Оrganizer" = %s', (id,))
42. o\_e **=** c.fetchone()
44. **if** o\_e[0] > 0:
45. **return** True, o\_e[0]
46. **else**:
47. **return** False,o\_e[0]
48. **def** get\_o\_e(self, id):
49. c **=** self.conn.cursor()
50. c.execute('SELECT \* FROM "Оrganizers/Events" WHERE "Оrganizer" = %s',  (id,))
51. **return** c.fetchall()
52. **def** delete\_o(self,id):
53. c **=** self.conn.cursor()
54. **if** self.is\_correct\_organizer\_id(id):
55. c.execute('DELETE FROM "Оrganizers" WHERE "ОrganizerID"=%s', (id,))
56. self.conn.commit()
57. **return** True
58. **else**:
59. **return** False
61. **def** cnt\_e\_fk(self,id):
62. c **=** self.conn.cursor()
63. c.execute('SELECT COUNT(\*) FROM "Оrganizers/Events" WHERE "Event\_o" = %s', (id,))
64. e1 **=** c.fetchone()
65. c.execute('SELECT COUNT(\*) FROM "Tourists/Events" WHERE "Event\_t" = %s', (id,))
66. e2 **=** c.fetchone()
67. **if** e1[0] > 0 **or** e2[0] > 0:
68. **return** True, e1[0],e2[0]
69. **else**:
70. **return** False,e1[0],e2[0]
72. **def** get\_e1(self, id):
73. c **=** self.conn.cursor()
74. c.execute('SELECT \* FROM "Оrganizers/Events" WHERE "Event\_o" = %s',  (id,))
75. **return** c.fetchall()
77. **def** get\_e2(self, id):
78. c **=** self.conn.cursor()
79. c.execute('SELECT \* FROM "Tourists/Events" WHERE "Event\_t" = %s',  (id,))
80. **return** c.fetchall()
82. **def** delete\_e(self,id):
83. c **=** self.conn.cursor()
84. **if** self.is\_correct\_event\_id(id):
85. c.execute('DELETE FROM "Events" WHERE "EventID"=%s', (id,))
86. self.conn.commit()
87. **return** True
88. **else**:
89. **return** False

**Фрагмент коду (файл model.py), який відповідає за створення потрібної кількості рандомних рядків в таблиці Events**

1. **def** rand\_event(self,n):
2. c **=** self.conn.cursor()
3. **for** i **in** range(n):
4. c.execute('INSERT INTO "Events" ("Title", "Type", "Date", "Address\_ID") VALUES ((select chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int)), (select chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int)), (select timestamp \'2023-01-10 20:00:00\' + random() \* (timestamp \'2023-12-20 20:00:00\' - timestamp \'2023-01-10 10:00:00\')), (select "AddressID" from "Address" order by random() limit 1))')
5. self.conn.commit()

**Повний текст програми:**

main.py

1. **from** controller **import** Controller
3. **if** \_\_name\_\_ **==** "\_\_main\_\_":
4. controller **=** Controller()
5. controller.run()

Модуль model.py

1. **import** psycopg2
2. **import** re
3. **from** datetime **import** datetime

6. **class** Model:
7. **def** \_\_init\_\_(self):
8. self.conn **=** psycopg2.connect(
9. dbname**=**'postgres',
10. user**=**'postgres',
11. password**=**'55krotvkaske',
12. host**=**'localhost',
13. port**=**5432
14. )
16. **def** is\_int(self, value):
17. **return** value.isdigit()
19. **def** is\_date(self, input\_string):
20. **try**:
21. datetime.strptime(input\_string, '%Y-%m-%d %H:%M:%S')
22. **return** True
23. **except** ValueError:
24. **return** False
25. **def** check\_str(self, value, length):
26. **if** len(value) <**=** length **and** len(value) >**=** 1:
27. **return** True
28. **else**:
29. **return** False

32. **def** is\_email(self,email):
33. pattern **=** r"^[-\w\.]+@([-\w]+\.)+[-\w]{2,4}$"
34. **if** re.match(pattern, email) **is** **not** None:
35. **return** True
36. **else**:
37. **return** False
38. **def** is\_correct\_address\_id(self, id):
39. c **=** self.conn.cursor()
40. **if** self.is\_int(id):
41. c.execute('SELECT COUNT(\*) FROM "Address" WHERE "AddressID" = %s', (id,))
42. r **=** c.fetchone()
43. **if** r[0] **==** 0:
44. **return** False
45. **elif** r[0] **==** 1:
46. **return** True
47. **else**:
48. **return** False
50. **def** is\_correct\_organizer\_id(self, id):
51. c **=** self.conn.cursor()
52. **if** self.is\_int(id):
53. c.execute('SELECT COUNT(\*) FROM "Оrganizers" WHERE "ОrganizerID" = %s', (id,))
54. r **=** c.fetchone()
55. **if** r[0] **==** 0:
56. **return** False
57. **elif** r[0] **==** 1:
58. **return** True
59. **else**:
60. **return** False
62. **def** is\_correct\_event\_id(self, id):
63. c **=** self.conn.cursor()
64. **if** self.is\_int(id):
65. c.execute('SELECT COUNT(\*) FROM "Events" WHERE "EventID" = %s', (id,))
66. r **=** c.fetchone()
67. **if** r[0] **==** 0:
68. **return** False
69. **elif** r[0] **==** 1:
70. **return** True
71. **else**:
72. **return** False
73. **def** is\_correct\_tourist\_id(self, id):
74. c **=** self.conn.cursor()
75. **if** self.is\_int(id):
76. c.execute('SELECT COUNT(\*) FROM "Tourists" WHERE "TouristID" = %s', (id,))
77. r **=** c.fetchone()
78. **if** r[0] **==** 0:
79. **return** False
80. **elif** r[0] **==** 1:
81. **return** True
82. **else**:
83. **return** False
84. **def** is\_correct\_phone\_id(self, id):
85. c **=** self.conn.cursor()
86. **if** self.is\_int(id):
87. c.execute('SELECT COUNT(\*) FROM "Phone\_nums" WHERE "PhoneID" = %s', (id,))
88. r **=** c.fetchone()
89. **if** r[0] **==** 0:
90. **return** False
91. **elif** r[0] **==** 1:
92. **return** True
93. **else**:
94. **return** False
96. **def** org\_tour\_check(self,f\_name, l\_name, email):
97. **if** self.check\_str(f\_name,30):
98. **if** self.check\_str(l\_name,30):
99. **if** self.check\_str(email,30):
100. **if** self.is\_email(email):
101. **return** True, 'Ok'
102. **else**:
103. **return** False, 'ERROR: wrong email style'
104. **else**:
105. **return** False, 'ERROR: wrong email length'
106. **else**:
107. **return** False, 'ERROR: wrong l\_name length'
108. **else**:
109. **return** False, 'ERROR: wrong f\_name length'
110. **def** add\_organizer(self, f\_name, l\_name, email):
111. c **=** self.conn.cursor()
112. b, strr **=** self.org\_tour\_check(f\_name, l\_name, email)
113. **if** b:
114. c.execute('INSERT INTO "Оrganizers" ("F\_name", "L\_name", email) VALUES (%s, %s, %s)',(f\_name, l\_name, email))
115. self.conn.commit()
116. **return** b,strr

119. **def** add\_tourist(self, f\_name, l\_name, email):
120. c **=** self.conn.cursor()
121. b, strr **=** self.org\_tour\_check(f\_name, l\_name, email)
122. **if** b:
123. c.execute('INSERT INTO "Tourists" ("F\_name", "L\_name", email) VALUES (%s, %s, %s)', (f\_name, l\_name, email))
124. self.conn.commit()
125. **return** b, strr
127. **def** event\_check(self,title, type\_, date, address\_id):
128. **if** self.is\_correct\_address\_id(address\_id):
129. **if** self.check\_str(title,100):
130. **if** self.check\_str(type\_,30):
131. **if** self.is\_date(date):
132. **return** True, 'Ok'
133. **else**:
134. **return** False, 'ERROR: wrong date style'
135. **else**:
136. **return** False, 'ERROR: wrong type length'
137. **else**:
138. **return** False, 'ERROR: wrong title length'
139. **else**:
140. **return** False, 'ERROR: wrong Address\_ID'
142. **def** add\_event(self, title, type\_, date, address\_id):
143. c **=** self.conn.cursor()
144. b, strr **=** self.event\_check(title, type\_, date, address\_id)
145. **if** b:
146. c.execute('INSERT INTO "Events" ("Title", "Type", "Date", "Address\_ID") VALUES (%s, %s, %s, %s)', (title, type\_, date, address\_id))
147. self.conn.commit()
148. **return** b,strr
150. **def** phone\_check(self, tourist\_id, number):
151. **if** self.is\_correct\_tourist\_id(tourist\_id):
152. **if** self.check\_str(number,16):
153. **return** True, 'Ok'
154. **else**:
155. **return** False, 'ERROR: wrong number length'
156. **else**:
157. **return** False, 'ERROR: wrong Tourist\_ID'
158. **def** add\_phone\_num(self, tourist\_id, number):
159. c **=** self.conn.cursor()
160. b, strr **=** self.phone\_check(tourist\_id, number)
161. **if** b:
162. c.execute('INSERT INTO "Phone\_nums" ("Tourist\_ID", "Number") VALUES (%s, %s)',(tourist\_id, number))
163. self.conn.commit()
164. **return** b,strr

167. **def** get\_all(self, strr, strr1):
168. c **=** self.conn.cursor()
169. c.execute('SELECT \* FROM %s ORDER BY %s ASC' **%**(strr, strr1))
170. **return** c.fetchall()
172. **def** update\_tour(self, id, f\_name, l\_name, email):
173. c **=** self.conn.cursor()
174. **if** self.is\_correct\_tourist\_id(id):
175. b,strr **=** self.org\_tour\_check(f\_name, l\_name, email)
176. **if** b:
177. c.execute('UPDATE "Tourists" SET "F\_name"=%s, "L\_name"=%s, email=%s WHERE "TouristID"=%s', (f\_name, l\_name, email, id))
178. self.conn.commit()
179. **return** b,strr
180. **else**:
181. **return** False, 'ERROR: wrong TouristID'
183. **def** update\_org(self, id, f\_name, l\_name, email):
184. c **=** self.conn.cursor()
185. **if** self.is\_correct\_organizer\_id(id):
186. b, strr **=** self.org\_tour\_check(f\_name, l\_name, email)
187. **if** b:
188. c.execute('UPDATE "Оrganizers" SET "F\_name"=%s, "L\_name"=%s, email=%s WHERE "ОrganizerID"=%s', (f\_name, l\_name, email, id))
189. self.conn.commit()
190. **return** b,strr
191. **else**:
192. **return** False, 'ERROR: wrong ОrganizerID'
194. **def** update\_event(self, id, title, type\_, date, address\_id):
195. c **=** self.conn.cursor()
196. **if** self.is\_correct\_event\_id(id):
197. b, strr **=** self.event\_check(title, type\_, date, address\_id)
198. **if** b:
199. c.execute('UPDATE "Events" SET "Title"=%s, "Type"=%s,"Date"=%s, "Address\_ID"=%s WHERE "EventID"=%s', (title, type\_, date, address\_id, id))
200. self.conn.commit()
201. **return** b,strr
202. **else**:
203. **return** False, 'ERROR: wrong EventID'
205. **def** update\_phone(self, id, tourist\_id, number):
206. c **=** self.conn.cursor()
207. **if** self.is\_correct\_phone\_id(id):
208. b, strr **=** self.phone\_check(tourist\_id, number)
209. **if** b:
210. c.execute('UPDATE "Phone\_nums" SET "Tourist\_ID"=%s, "Number"=%s WHERE "PhoneID"=%s', (tourist\_id, number, id))
211. self.conn.commit()
212. **return** b, strr
213. **else**:
214. **return** False, 'ERROR: wrong PhoneID'
216. **def** delete\_p(self, id):
217. c **=** self.conn.cursor()
218. **if** self.is\_correct\_phone\_id(id):
219. c.execute('DELETE FROM "Phone\_nums" WHERE "PhoneID"=%s', (id,))
220. self.conn.commit()
221. **return** True
222. **else**:
223. **return** False
224. **def** cnt\_t\_fk(self,id):
225. c **=** self.conn.cursor()
226. c.execute('SELECT COUNT(\*) FROM "Tourists/Events" WHERE "Tourist" = %s', (id,))
227. t\_e **=** c.fetchone()
228. c.execute('SELECT COUNT(\*) FROM "Phone\_nums" WHERE "Tourist\_ID" = %s', (id,))
229. p **=** c.fetchone()
230. **if** t\_e[0] > 0 **or** p[0] > 0:
231. **return** True, t\_e[0], p[0]
232. **else**:
233. **return** False,t\_e[0],p[0]
234. **def** get\_t\_e(self, id):
235. c **=** self.conn.cursor()
236. c.execute('SELECT \* FROM "Tourists/Events" WHERE "Tourist" = %s',  (id,))
237. **return** c.fetchall()
239. **def** get\_p(self, id):
240. c **=** self.conn.cursor()
241. c.execute('SELECT \* FROM "Phone\_nums" WHERE "Tourist\_ID" = %s',  (id,))
242. **return** c.fetchall()
243. **def** delete\_t(self, id):
244. c **=** self.conn.cursor()
245. **if** self.is\_correct\_tourist\_id(id):
246. c.execute('DELETE FROM "Tourists" WHERE "TouristID"=%s', (id,))
247. self.conn.commit()
248. **return** True
249. **else**:
250. **return** False


254. **def** cnt\_o\_fk(self,id):
255. c **=** self.conn.cursor()
256. c.execute('SELECT COUNT(\*) FROM "Оrganizers/Events" WHERE "Оrganizer" = %s', (id,))
257. o\_e **=** c.fetchone()
259. **if** o\_e[0] > 0:
260. **return** True, o\_e[0]
261. **else**:
262. **return** False,o\_e[0]
263. **def** get\_o\_e(self, id):
264. c **=** self.conn.cursor()
265. c.execute('SELECT \* FROM "Оrganizers/Events" WHERE "Оrganizer" = %s',  (id,))
266. **return** c.fetchall()
267. **def** delete\_o(self,id):
268. c **=** self.conn.cursor()
269. **if** self.is\_correct\_organizer\_id(id):
270. c.execute('DELETE FROM "Оrganizers" WHERE "ОrganizerID"=%s', (id,))
271. self.conn.commit()
272. **return** True
273. **else**:
274. **return** False
276. **def** cnt\_e\_fk(self,id):
277. c **=** self.conn.cursor()
278. c.execute('SELECT COUNT(\*) FROM "Оrganizers/Events" WHERE "Event\_o" = %s', (id,))
279. e1 **=** c.fetchone()
280. c.execute('SELECT COUNT(\*) FROM "Tourists/Events" WHERE "Event\_t" = %s', (id,))
281. e2 **=** c.fetchone()
282. **if** e1[0] > 0 **or** e2[0] > 0:
283. **return** True, e1[0],e2[0]
284. **else**:
285. **return** False,e1[0],e2[0]
287. **def** get\_e1(self, id):
288. c **=** self.conn.cursor()
289. c.execute('SELECT \* FROM "Оrganizers/Events" WHERE "Event\_o" = %s',  (id,))
290. **return** c.fetchall()
292. **def** get\_e2(self, id):
293. c **=** self.conn.cursor()
294. c.execute('SELECT \* FROM "Tourists/Events" WHERE "Event\_t" = %s',  (id,))
295. **return** c.fetchall()
297. **def** delete\_e(self,id):
298. c **=** self.conn.cursor()
299. **if** self.is\_correct\_event\_id(id):
300. c.execute('DELETE FROM "Events" WHERE "EventID"=%s', (id,))
301. self.conn.commit()
302. **return** True
303. **else**:
304. **return** False
306. **def** rand\_event(self,n):
307. c **=** self.conn.cursor()
308. **for** i **in** range(n):
309. c.execute('INSERT INTO "Events" ("Title", "Type", "Date", "Address\_ID") VALUES ((select chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int)), (select chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int)), (select timestamp \'2023-01-10 20:00:00\' + random() \* (timestamp \'2023-12-20 20:00:00\' - timestamp \'2023-01-10 10:00:00\')), (select "AddressID" from "Address" order by random() limit 1))')
310. self.conn.commit()
311. **def** cnt\_events(self):
312. c **=** self.conn.cursor()
313. c.execute('SELECT COUNT(\*) FROM "Events"')
314. r **=** c.fetchone()
315. **return** r[0]
317. **def** call(self):
318. c **=** self.conn.cursor()
319. c.execute('select \* from (select "Address\_ID" as "AddressID", count (\*) as cnt from "Events" where "Date" > Now() group by "Address\_ID") as address\_id inner join "Address"  on "Address"."AddressID" = address\_id."AddressID"')
320. **return** c.fetchall()

Модуль controller.py

1. **from** model **import** Model
2. **from** view **import** View

5. **class** Controller:
6. **def** \_\_init\_\_(self):
7. self.model **=** Model()
8. self.view **=** View()
10. **def** run(self):
11. **while** True:
12. choice **=** self.view.show\_menu()
13. **if** choice **==** '1':
14. c1 **=** self.view.show\_extra\_menu()
15. **if** c1 **==** '1':
16. self.add\_new\_tourist()
17. **elif** c1 **==** '2':
18. self.add\_new\_organizer()
19. **elif** c1 **==** '3':
20. self.add\_new\_event()
21. **elif** c1 **==** '4':
22. self.add\_new\_phone\_num()

25. **elif** choice **==** '2':
26. c1 **=** self.view.show\_extra\_menu()
27. **if** c1 **==** '1':
28. self.view\_t()
29. **elif** c1 **==** '2':
30. self.view\_o()
31. **elif** c1 **==** '3':
32. self.view\_e()
33. **elif** c1 **==** '4':
34. self.view\_p()
35. **elif** choice **==** '3':
36. c1 **=** self.view.show\_extra\_menu()
37. **if** c1 **==** '1':
38. self.update\_t()
39. **elif** c1 **==** '2':
40. self.update\_o()
41. **elif** c1 **==** '3':
42. self.update\_e()
43. **elif** c1 **==** '4':
44. self.update\_p()
45. **elif** choice **==** '4':
46. c1 **=** self.view.show\_extra\_menu()
47. **if** c1 **==** '1':
48. self.delete\_t()
49. **elif** c1 **==** '2':
50. self.delete\_o()
51. **elif** c1 **==** '3':
52. self.delete\_e()
53. **elif** c1 **==** '4':
54. self.delete\_p()
55. **elif** choice **==** '5':
56. self.rand\_event()
57. **elif** choice **==** '6':
58. self.event\_cnt()
59. **elif** choice **==** '7':
60. self.view\_call()
61. **elif** choice **==** '8':
62. **break**







71. **def** add\_new\_tourist(self):
72. f\_name,l\_name,email **=** self.view.get\_tourist\_organizer\_input()
73. b,msg **=** self.model.add\_tourist(f\_name,l\_name,email)
74. **if** b:
75. self.view.show\_message("Tourist added successfully!")
76. **else**:
77. self.view.show\_message(msg)
79. **def** add\_new\_organizer(self):
80. f\_name,l\_name,email **=** self.view.get\_tourist\_organizer\_input()
81. b,msg **=** self.model.add\_organizer(f\_name,l\_name,email)
82. **if** b:
83. self.view.show\_message("Organizer added successfully!")
84. **else**:
85. self.view.show\_message(msg)
87. **def** add\_new\_event(self):
88. title, type\_, date, address\_id **=** self.view.get\_event\_input()
89. b,msg **=** self.model.add\_event(title, type\_, date, address\_id)
90. **if** b:
91. self.view.show\_message("Event added successfully!")
92. **else**:
93. self.view.show\_message(msg)
95. **def** add\_new\_phone\_num(self):
96. tourist\_id, number **=** self.view.get\_phone\_input()
97. b,msg **=** self.model.add\_phone\_num(tourist\_id, number)
98. **if** b:
99. self.view.show\_message("Event added successfully!")
100. **else**:
101. self.view.show\_message(msg)
103. **def** view\_t(self):
104. t **=** self.model.get\_all('"Tourists"', '"TouristID"')
105. self.view.show\_tourists(t)
107. **def** view\_o(self):
108. t **=** self.model.get\_all('"Оrganizers"', '"ОrganizerID"')
109. self.view.show\_organizers(t)
111. **def** view\_e(self):
112. t **=** self.model.get\_all('"Events"', '"EventID"')
113. self.view.show\_events(t)
115. **def** view\_p(self):
116. t **=** self.model.get\_all('"Phone\_nums"', '"PhoneID"')
117. self.view.show\_nums(t)
119. **def** update\_t(self):
120. id **=** self.view.get\_id()
121. f\_name, l\_name, email **=** self.view.get\_tourist\_organizer\_input()
122. b, strr **=** self.model.update\_tour(id, f\_name, l\_name, email)
123. **if** b:
124. self.view.show\_message(f"Tourist id = {id} updated successfully!")
125. **else**:
126. self.view.show\_message(strr)
128. **def** update\_o(self):
129. id **=** self.view.get\_id()
130. f\_name, l\_name, email **=** self.view.get\_tourist\_organizer\_input()
131. b, strr **=** self.model.update\_org(id, f\_name, l\_name, email)
132. **if** b:
133. self.view.show\_message(f"Organizer id = {id} updated successfully!")
134. **else**:
135. self.view.show\_message(strr)
137. **def** update\_e(self):
138. id **=** self.view.get\_id()
139. title, type\_, date, address\_id **=** self.view.get\_event\_input()
140. b, strr **=** self.model.update\_event(id, title, type\_, date, address\_id)
141. **if** b:
142. self.view.show\_message(f"Event id = {id} updated successfully!")
143. **else**:
144. self.view.show\_message(strr)
146. **def** update\_p(self):
147. id **=** self.view.get\_id()
148. tourist\_id, number **=** self.view.get\_phone\_input()
149. b, strr **=** self.model.update\_phone(id, tourist\_id, number)
150. **if** b:
151. self.view.show\_message(f"Phone id = {id} updated successfully!")
152. **else**:
153. self.view.show\_message(strr)
155. **def** delete\_p(self):
156. id **=** self.view.get\_id()
157. b **=** self.model.delete\_p(id)
158. **if** b:
159. self.view.show\_message("Phone deleted successfully!")
160. **else**:
161. self.view.show\_message("ERROR: wrong id")
163. **def** delete\_t(self):
164. id **=** self.view.get\_id()
165. b, t\_e, p **=** self.model.cnt\_t\_fk(id)
166. **if** **not** b:
167. bb **=** self.model.delete\_t(id)
168. **if** bb:
169. self.view.show\_message("Tourist deleted successfully!")
170. **else**:
171. self.view.show\_message("ERROR: wrong id")
172. **else**:
173. self.view.show\_message("\nWARNING:\n")
174. self.view.show\_message("You need to manually delete these rows in the tables, due to the fact that they contain references to the id of the row being deleted, and then try to delete again")
175. **if** t\_e > 0:
176. j **=** self.model.get\_t\_e(id)
177. self.view.show\_t\_e(j)
178. **if** p > 0:
179. j **=** self.model.get\_p(id)
180. self.view.show\_nums(j)
182. **def** delete\_o(self):
183. id **=** self.view.get\_id()
184. b, o\_e **=** self.model.cnt\_o\_fk(id)
185. **if** **not** b:
186. bb **=** self.model.delete\_o(id)
187. **if** bb:
188. self.view.show\_message("Organizer deleted successfully!")
189. **else**:
190. self.view.show\_message("ERROR: wrong id")
191. **else**:
192. self.view.show\_message("\nWARNING:\n")
193. self.view.show\_message(
194. "You need to manually delete these rows in the tables, due to the fact that they contain references to the id of the row being deleted, and then try to delete again")
195. **if** o\_e > 0:
196. j **=** self.model.get\_o\_e(id)
197. self.view.show\_o\_e(j)
199. **def** delete\_e(self):
200. id **=** self.view.get\_id()
201. b, e1,e2 **=** self.model.cnt\_e\_fk(id)
202. **if** **not** b:
203. bb **=** self.model.delete\_e(id)
204. **if** bb:
205. self.view.show\_message("Event deleted successfully!")
206. **else**:
207. self.view.show\_message("ERROR: wrong id")
208. **else**:
209. self.view.show\_message("\nWARNING:\n")
210. self.view.show\_message(
211. "You need to manually delete these rows in the tables, due to the fact that they contain references to the id of the row being deleted, and then try to delete again")
212. **if** e1 > 0:
213. j **=** self.model.get\_o\_e(id)
214. self.view.show\_o\_e(j)
215. **if** e2 > 0:
216. j **=** self.model.get\_t\_e(id)
217. self.view.show\_t\_e(j)
219. **def** rand\_event(self):
220. n **=** int(self.view.get\_cnt())
221. self.model.rand\_event(n)
222. self.view.show\_message("%s rows were created in the Events table" **%**n)
224. **def** event\_cnt(self):
225. self.view.show\_message("Number of Events: " **+** str(self.model.cnt\_events()))
227. **def** view\_call(self):
228. t **=** self.model.call()
229. self.view.show\_call(t)

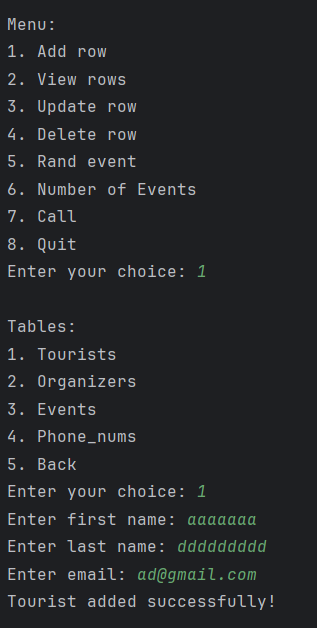
Модуль view.py

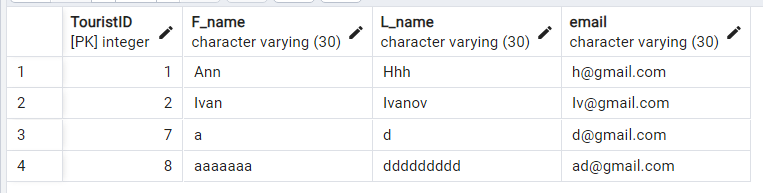
1. **class** View:
3. **def** show\_menu(self):
4. self.show\_message("\nMenu:")
5. self.show\_message("1. Add row")
6. self.show\_message("2. View rows")
7. self.show\_message("3. Update row")
8. self.show\_message("4. Delete row")
9. self.show\_message("5. Rand event")
10. self.show\_message("6. Number of Events")
11. self.show\_message("7. Call")
12. self.show\_message("8. Quit")
13. **return** input("Enter your choice: ")

16. **def** show\_extra\_menu(self):
17. self.show\_message("\nTables: ")
18. self.show\_message("1. Tourists")
19. self.show\_message("2. Organizers")
20. self.show\_message("3. Events")
21. self.show\_message("4. Phone\_nums")
22. self.show\_message("5. Back")
23. **return** input("Enter your choice: ")
25. **def** show\_tourists(self, j):
26. print("Tourists: ")
27. **for** i **in** j:
28. print(f"TouristID: {i[0]}, F\_name: {i[1]}, L\_name: {i[2]}, email: {i[3]}")
30. **def** show\_organizers(self, j):
31. print("Organizers: ")
32. **for** i **in** j:
33. print(f"OrganizerID: {i[0]}, F\_name: {i[1]}, L\_name: {i[2]}, email: {i[3]}")
35. **def** show\_events(self, j):
36. print("Events: ")
37. **for** i **in** j:
38. print(f"EventID: {i[0]}, Title: {i[1]}, Type: {i[2]}, Date: {i[3]}, Address\_ID: {i[4]}")
40. **def** show\_nums(self, j):
41. print("Phone\_nums: ")
42. **for** i **in** j:
43. print(f"PhoneID: {i[0]}, Tourist\_ID: {i[1]}, Number: {i[2]}")
45. **def** show\_t\_e(self, j):
46. print("Tourists/Events: ")
47. **for** i **in** j:
48. print(f"Tourist: {i[0]}, Event\_t: {i[1]}, booking\_time: {i[2]}")
50. **def** show\_o\_e(self, j):
51. print("Оrganizers/Events: ")
52. **for** i **in** j:
53. print(f"Оrganizer: {i[0]}, Event\_o: {i[1]}")
55. **def** show\_call(self, j):
56. print("Number of events that will take place starting tomorrow: ")
57. **for** i **in** j:
58. print(f"AddressID: {i[0]}, cnt: {i[1]}, AddressID: {i[2]}, Country: {i[3]}, City: {i[4]}, Street: {i[5]}, House: {i[6]}")
60. **def** get\_tourist\_organizer\_input(self):
61. f\_name **=** input("Enter first name: ")
62. l\_name **=** input("Enter last name: ")
63. email **=** input("Enter email: ")
64. **return** f\_name,l\_name,email
65. **def** get\_event\_input(self):
66. title **=** input("Enter event title: ")
67. type\_ **=** input("Enter event type: ")
68. date **=** input("Enter event date: ")
69. address\_id **=** input("Enter event address\_id: ")
70. **return** title, type\_, date, address\_id
72. **def** get\_phone\_input(self):
73. tourist\_id **=** input("Enter tourist\_id: ")
74. number **=** input("Enter number: ")
75. **return** tourist\_id, number
77. **def** get\_id(self):
78. **return** input("Enter ID: ")
80. **def** get\_cnt(self):
81. **return** input("Enter enter number of lines to create: ")
83. **def** show\_message(self, message):
84. print(message)

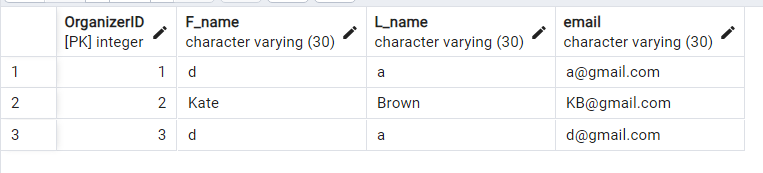
**Скріншоти, які демонструють роботу програми**

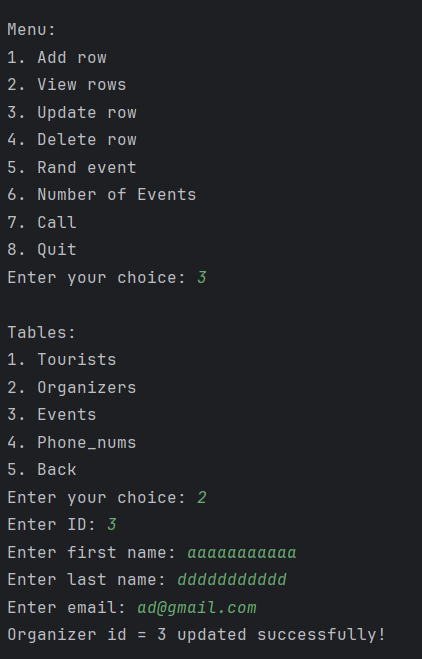
Додавання рядку до таблиці Tourists

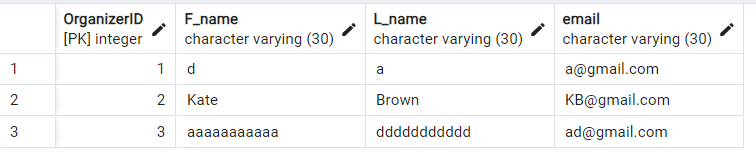
****

****

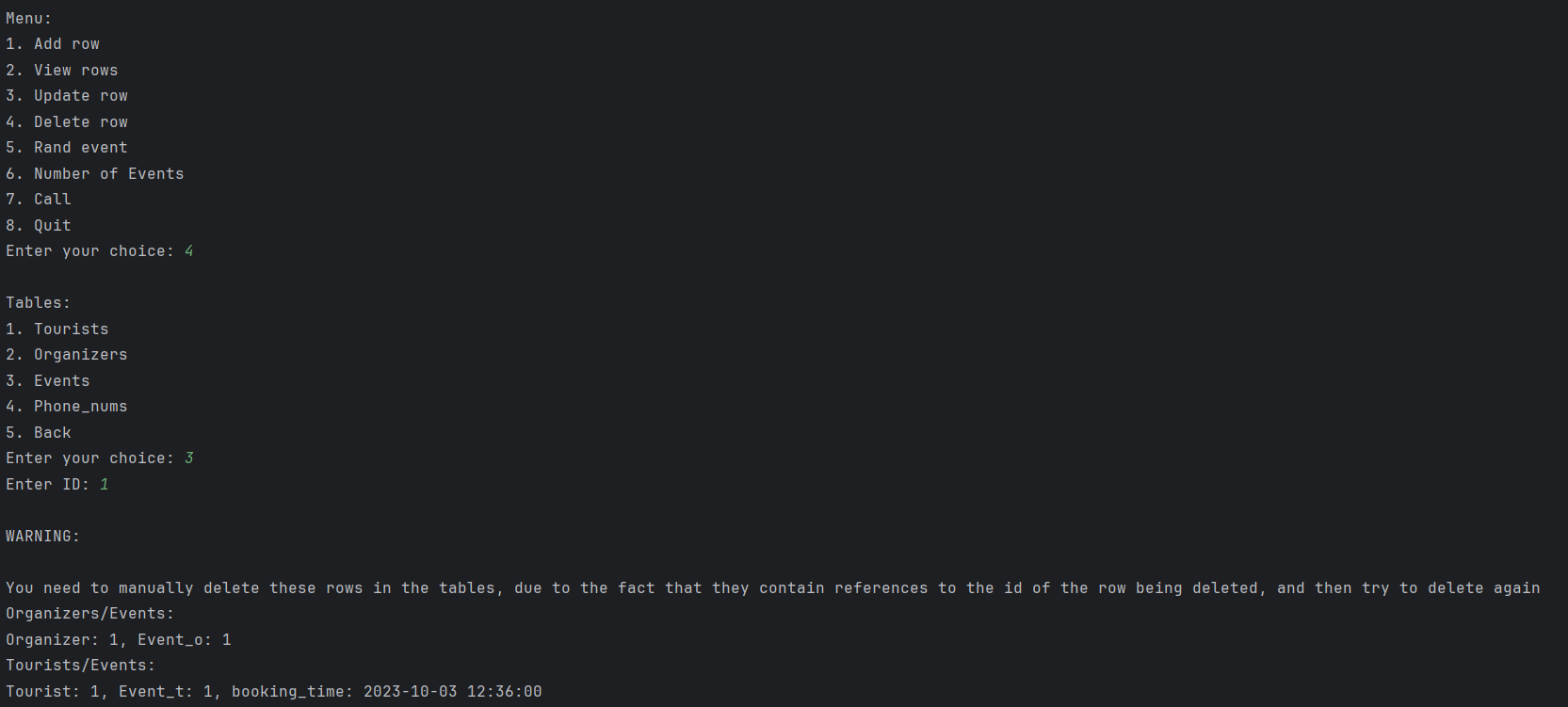
Оновлення рядку з таблиці Organizers



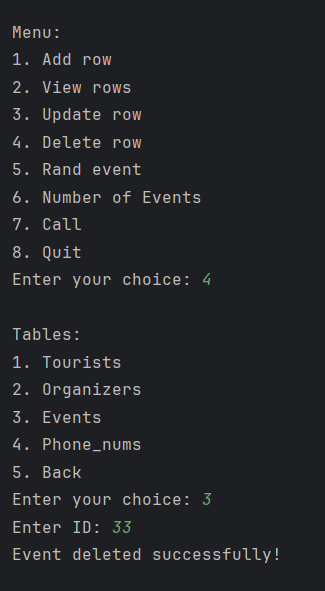




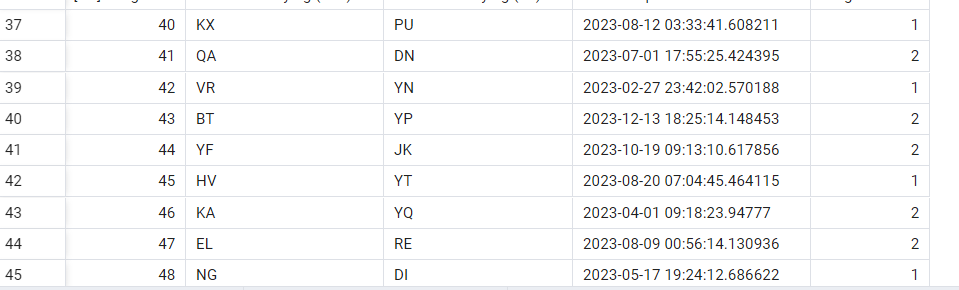
Видалення рядку Events, ідентифікатор якого використовується як зовнішній ключ

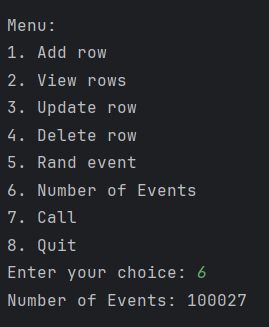


Видалення рядку Events, ідентифікатор якого НЕ використовується як зовнішній ключ



Створення рандомних рядків у таблиці Events





Запит який показує кількість заходів, які будуть відбуватись з завтрашнього дня, згруповані по адресам та додана інформація з іншої таблиці, що саме це за адреси

