Изображение выглядит как текст

Автоматически созданное описание

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**

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

Виконала:

студентка ІII курсу

групи КВ-94,

Калюжна Марина

Перевірив:

Петрашенко А. В.

Київ – 2023

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

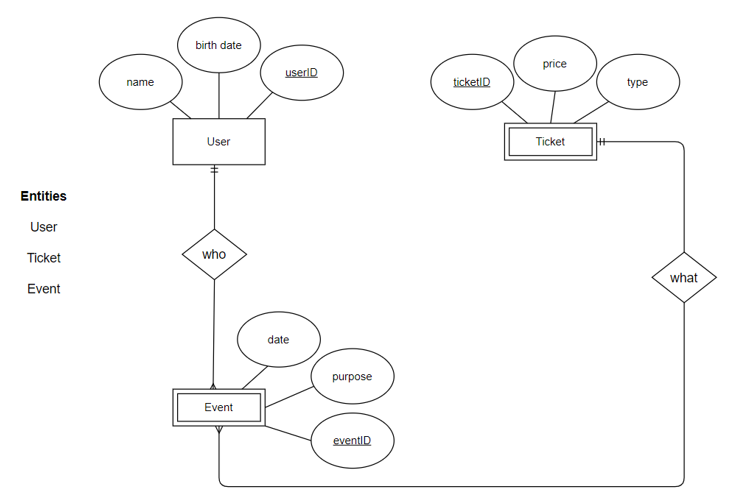
***Загальне завдання*** роботи полягає у наступному:

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

***Деталізоване завдання***:

1. Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв’язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати **вилучення** рядків за умови наявності даних у підлеглій таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.
2. Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими **не мовою програмування, а відповідним SQL-запитом**!
3. Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.
4. Програмний код організувати згідно шаблону Model-View-Controller(MVC).  Приклад організації коду згідно шаблону доступний [за даним посиланням](https://www.giacomodebidda.com/mvc-pattern-in-python-introduction-and-basicmodel/). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

<https://github.com/marinka09/marinka09> посилання на репозиторій

**Модель «сутність-зв’язок» системи резервувань квитків на заходи** ****

*Рисунок 1. ER-діаграма, побудована за нотацією "Пташиної лапки"*

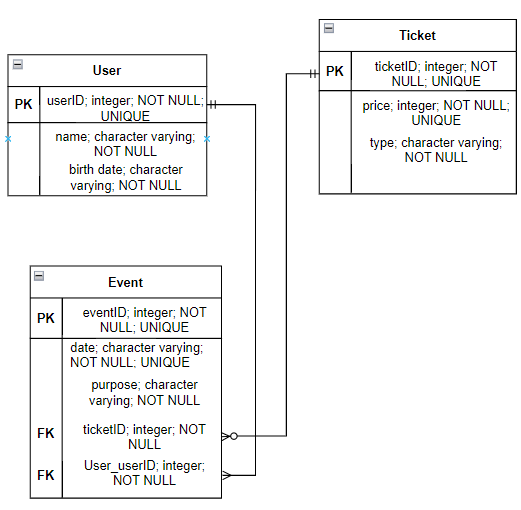
**Сутності, їх призначення та опис зв'язків:**

Маємо три сутності: ***User***, ***Ticket***, ***Event***.

**Сутність User** описує користувачів системи. Кожен користувач має ім’я, дату народження та свій ID та кожен з користувачів може відвідати декілька івентів.

**Сутність Ticket** – це квитки, які придбали користувачі. Тобто кожен окремий користувач може придбати багато квитків. Кожний квиток також має свій ID, тип та ціну, і в кожному квитку може бути прописано декілька подій (Event).

**Третя сутність – це Event**. Це заходи, кожний має свою певну дату, ID та мету. Захід має два зв’язки: хто виконує цей захід (користувач) та що це за захід (тобто в якому саме білеті є номер з назвою ). Один користувач може брати участь в багатьох заходах, а в одному білеті може бути прописано декілька заходів.



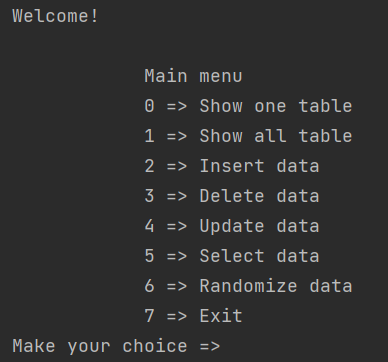
*Рисунок 2. Схема бази даних у графічному вигляді*

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

Мова програмування – Python 3.11

Середовище розробки програмного забезпечення – PyCharm Community Edition.

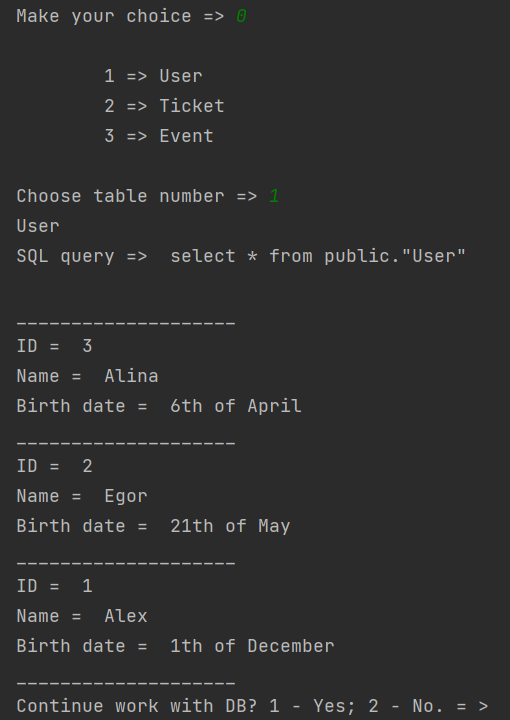
**Структура меню програми**



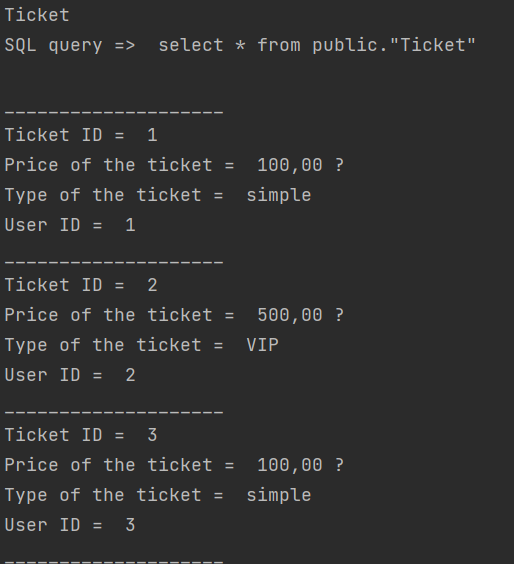
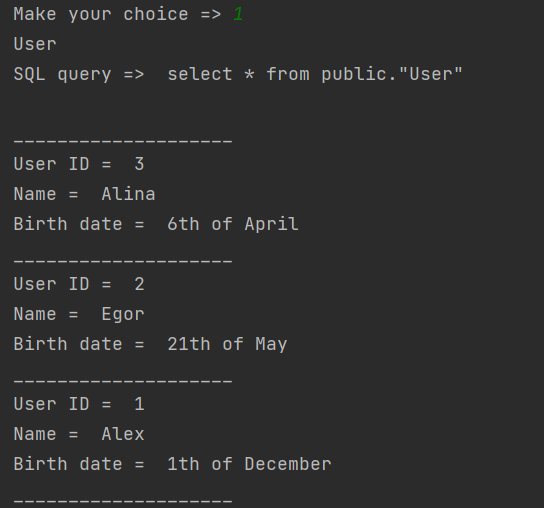
**Завдання 1**

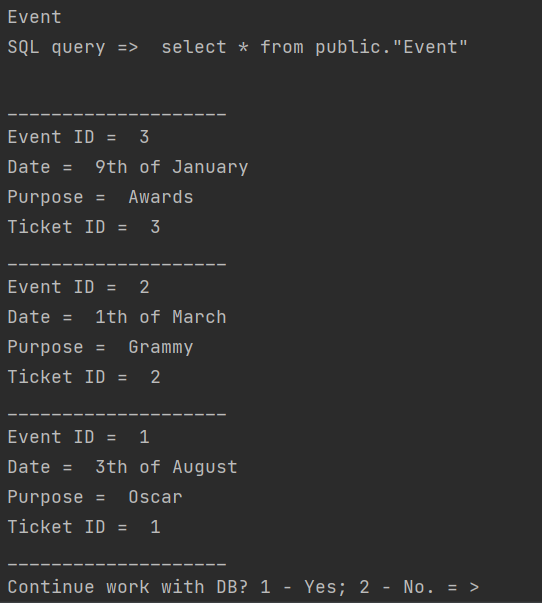
Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.

**Перегляд однієї таблиці**



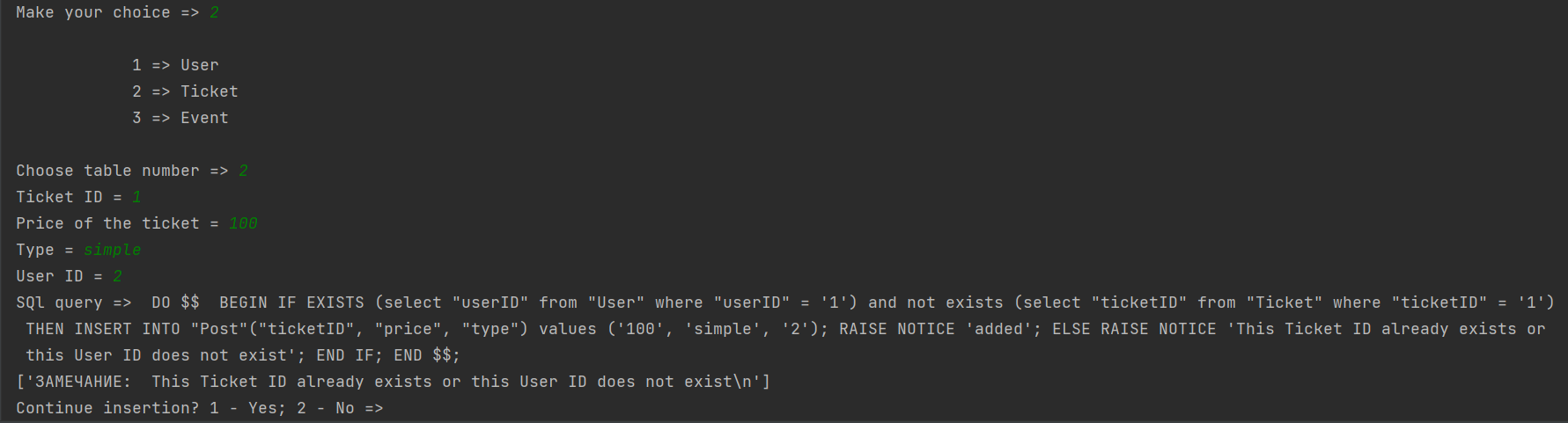
**Перегляд всіх таблиць**



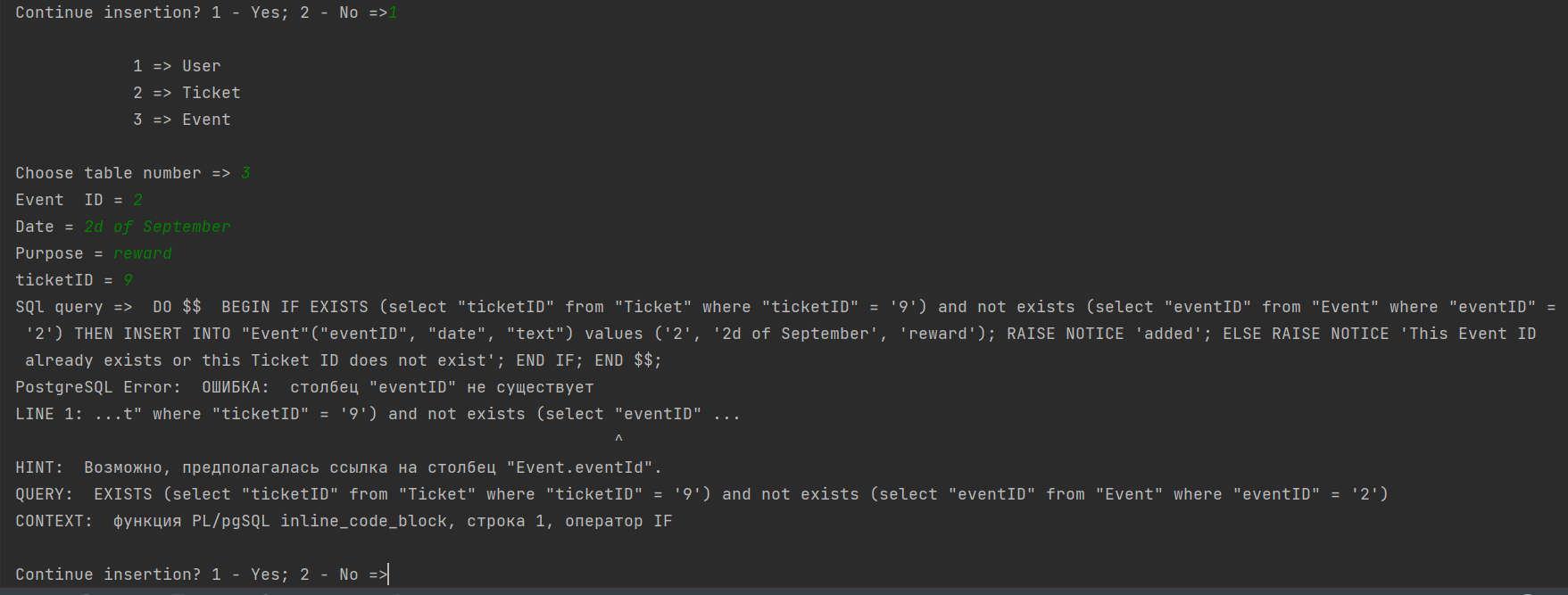


**Внесення даних**

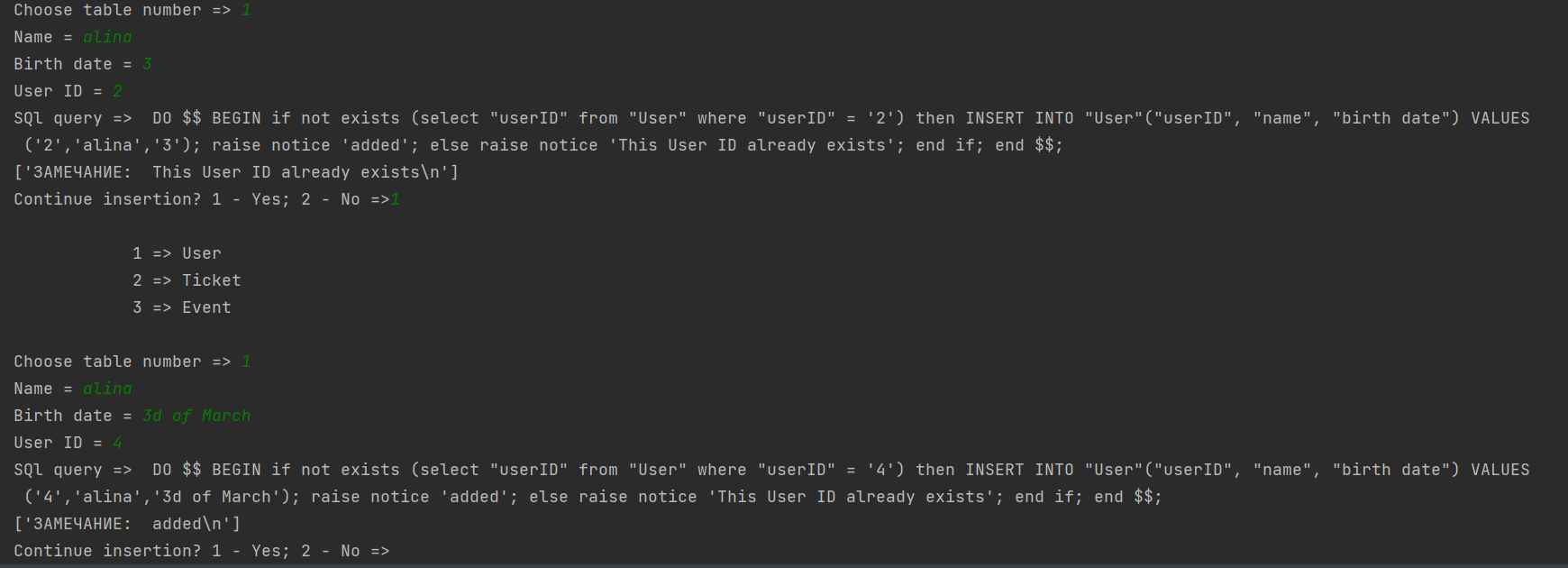
**Якщо такий первинний ключ вже є, виводиться помилка, яка каже користувачу про це:**

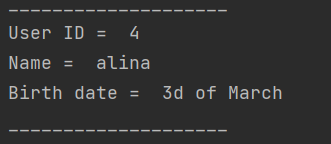


**Якщо такого первічного ключа немає, але також немає такого вторинного ключа, то також виводиться помилка, яка каже користувачеві про це (після виведення помилок програма не зупиняється):**



**Працює без помилок:**



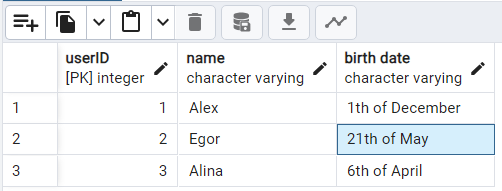


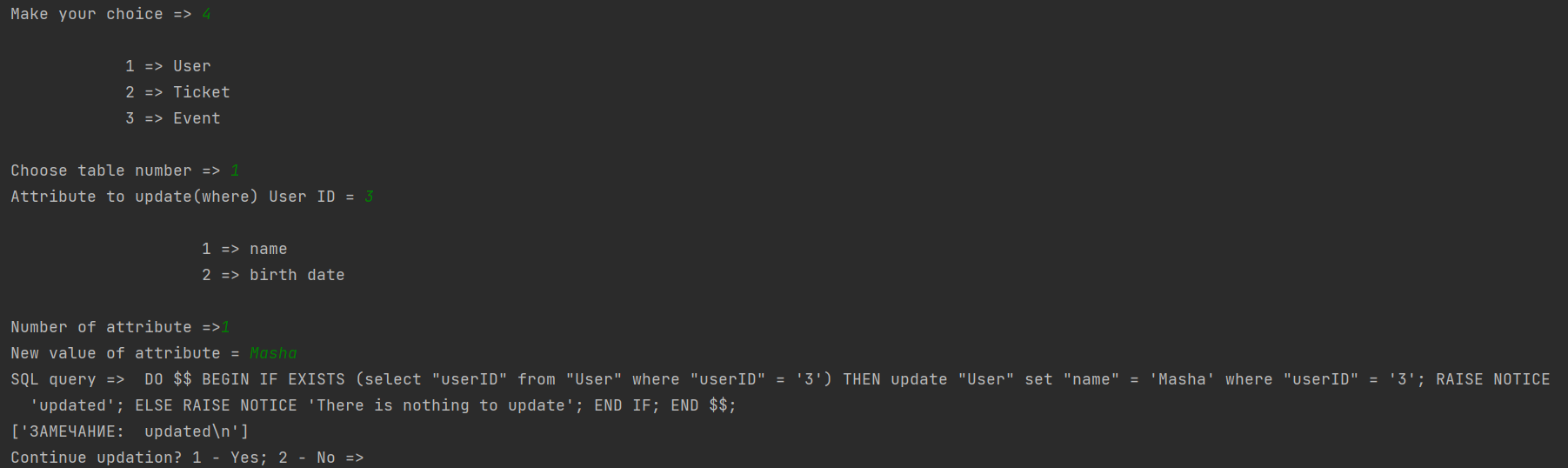
**Лістинг операції insert:**

@staticmethod  
def insert\_for\_table1(usname, usbirth\_date, usid):  
 connection = controller.makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 notice = "'This User ID already exists'"  
 insert = 'DO $$ BEGIN if not exists (select "userID" from "User" where "userID" = {}) then INSERT ' \  
 'INTO "User"("userID", "name", "birth date") VALUES ({},{},{}); ' \  
 'raise notice {}; else raise notice {}; ' \  
 'end if; end $$;'.format(usid, usid, usname, usbirth\_date, "'added'", notice)  
 restart = False  
 print('SQl query => ', insert)  
 cursor.execute(insert)  
 connection.commit()  
 print(connection.notices)  
 cursor.close()  
 controller.closeConnect(connection)  
  
@staticmethod  
def insert\_for\_table2(poid, poname, potopic, pouser):  
 connection = controller.makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 notice = "'This Ticket ID already exists or this User ID does not exist'"  
 insert = 'DO $$ BEGIN IF EXISTS (select "userID" from "User" where "userID" = {}) and not exists ' \  
 '(select "ticketID" from "Ticket" where "ticketID" = {}) THEN ' \  
 'INSERT INTO "Ticket"("ticketID", "price", "type", "userID" ) values ({}, {}, {}, {}); RAISE NOTICE {};' \  
 ' ELSE RAISE NOTICE {}; END IF; ' \  
 'END $$;'.format(pouser, poid, poid, poname, potopic, pouser, "'added'", notice)  
 restart = False  
 print('SQl query => ', insert)  
 cursor.execute(insert)  
 connection.commit()  
 print(connection.notices)  
 cursor.close()  
 controller.closeConnect(connection)  
  
@staticmethod  
def insert\_for\_table3(coid, codate, cotext, copost):  
 connection = controller.makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 notice = "'This Event ID already exists or this Ticket ID does not exist'"  
 insert = 'DO $$ BEGIN IF EXISTS (select "ticketID" from "Ticket" where "ticketID" = {}) and not exists ' \  
 '(select "eventID" from "Event" where "eventID" = {}) THEN ' \  
 'INSERT INTO "Event"("eventID", "date", "purpose, ticket ID") values ({}, {}, {}, {}); RAISE NOTICE {};' \  
 ' ELSE RAISE NOTICE {}; END IF; END $$;'.format(copost, coid, coid, codate, cotext, copost,  
 "'added'", notice)  
 restart = False  
 print('SQl query => ', insert)  
 cursor.execute(insert)  
 connection.commit()  
 print(connection.notices)  
 cursor.close()  
 controller.closeConnect(connection)

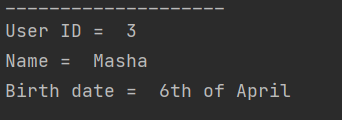
**Редагування даних**

**Таблиця до:**

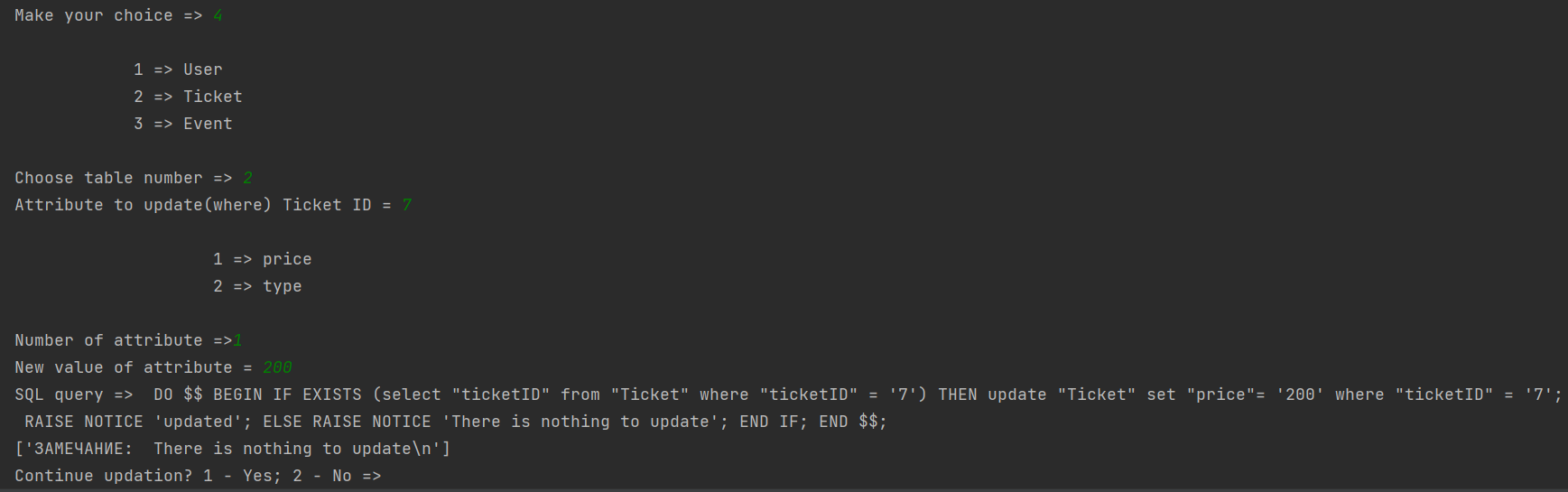




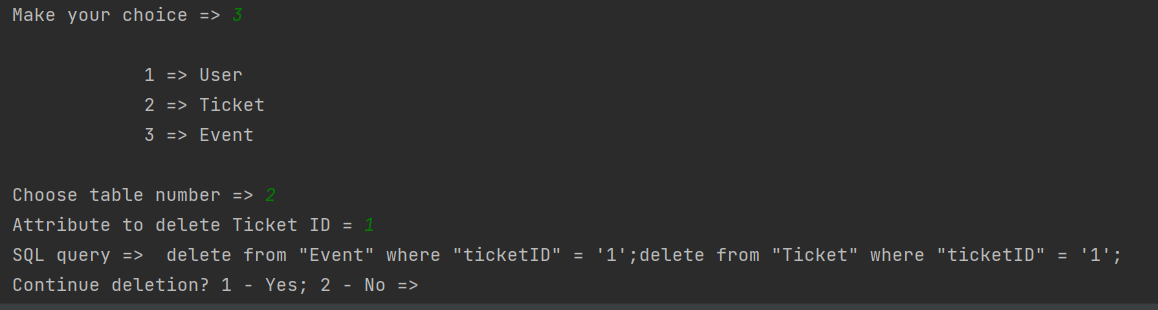
**Таблиця після:**

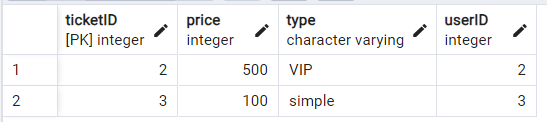


**Якщо заданого користувачем айді немає, виводиться помилка.**



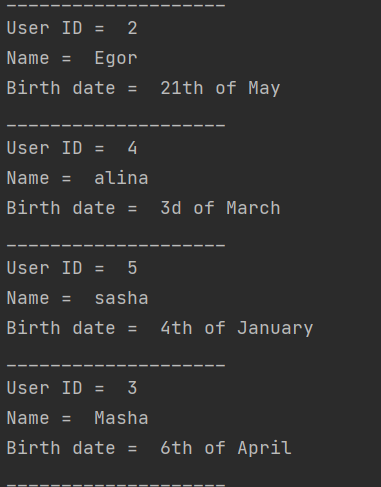
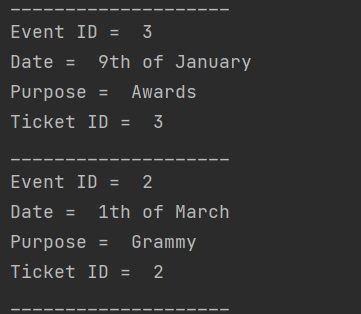
**Вилучення даних**





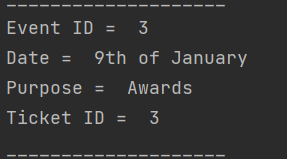
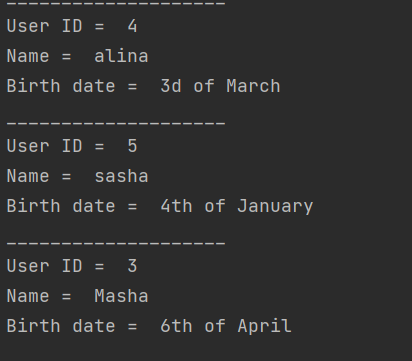
**Вилучення рядка з батьківської таблиці**

**До:**

****

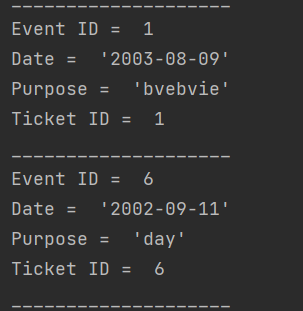
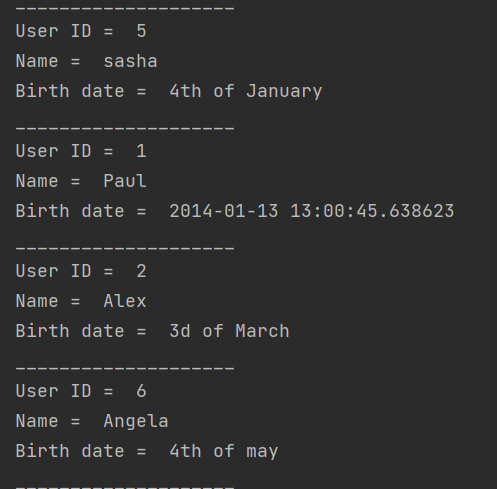
Видаляємо User ID = 2

**Після:**

****

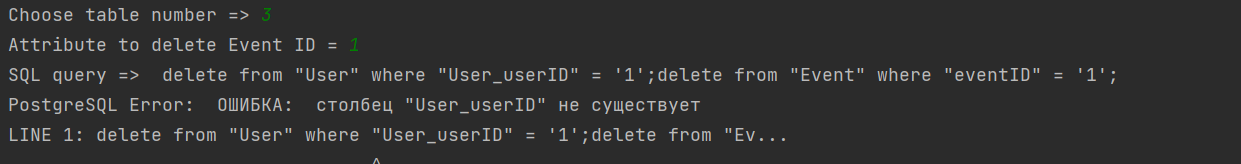
Тобто при видаленні рядка в батьківській таблиці, автоматично видаляється рядок і в дочірній.

**До:**

****

Хочемо видлалити з таблиці Event, event ID=1.

**Після**

****

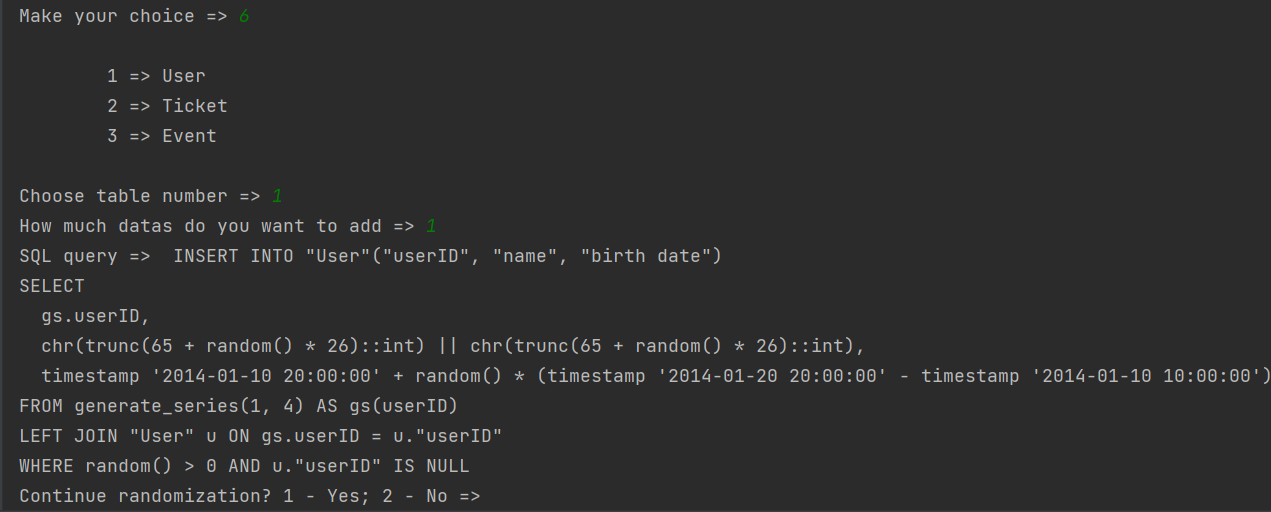
Відповідно ми отримаємо помилку, оскільки дані в таблиці Event залежать від даних таблиці User, тому таке видалення неможливе.

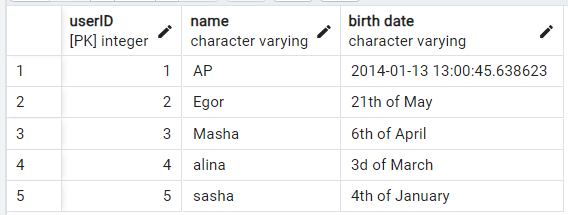
**Лістинг операції delete:**

@staticmethod  
def delete\_for\_table1(usid):  
 connection = controller.makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 delete ='delete from "Event" where "ticketID" in (select "ticketID" from "Ticket" where "userID" = {});' \  
 'delete from "Ticket" where "userID" = {};' \  
 'delete from "User" where "userID" = {};'.format(usid, usid, usid)  
  
 restart = False  
 print("SQL query => ", delete)  
 cursor.execute(delete)  
 connection.commit()  
 cursor.close()  
 controller.closeConnect(connection)  
  
@staticmethod  
def delete\_for\_table2(poid):  
 connection = controller.makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 delete = 'delete from "Event" where "ticketID" = {};' \  
 'delete from "Ticket" where "ticketID" = {};'.format(poid, poid)  
 restart = False  
 print("SQL query => ", delete)  
 cursor.execute(delete)  
 connection.commit()  
 cursor.close()  
 controller.closeConnect(connection)  
  
@staticmethod  
def delete\_for\_table3(coid):  
 connection = controller.makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 delete = 'delete from "User" where "User\_userID" = {};' \  
 'delete from "Event" where "eventID" = {};'.format(coid, coid)  
 restart = False  
 print("SQL query => ", delete)  
 cursor.execute(delete)  
 connection.commit()  
 cursor.close()  
 controller.closeConnect(connection)

**Завдання 2**

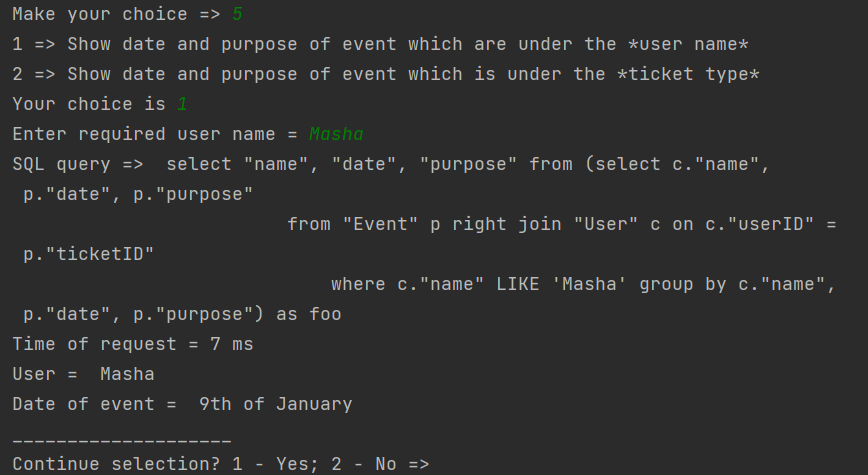
Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.



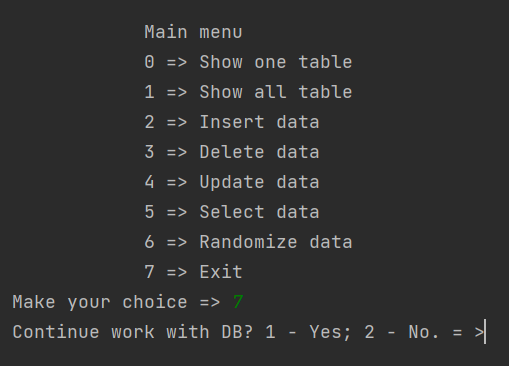


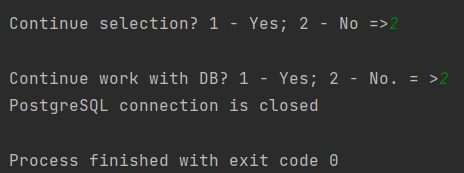
**Завдання 3**

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.



**Вихід із програми**

****



**Програмний модуль model.py**

**Цей програмний модуль відповідає за всю ЛОГІКУ проекта.**

import psycopg2  
import time  
  
  
def makeConnect():  
 return psycopg2.connect(  
 dbname='postgres',  
 user='postgres',  
 password='123',  
 host='localhost',  
 port=5432  
 )  
  
  
def closeConnect(connection):  
 connection.commit()  
 connection.close()  
  
tables = {  
 1: 'User',  
 2: 'Ticket',  
 3: 'Event',  
}  
  
class Model:  
 @staticmethod  
 def validTable(table):  
 incorrect = True  
 while incorrect:  
 if str(table).isdigit():  
 table = int(table)  
 if table >= 1 and table <= 3:  
 incorrect = False  
 else:  
 print('Incorrect input, try again.')  
 else:  
 print('Incorrect input, try again.')  
 return table  
  
 @staticmethod  
 def showOneTable(table):  
 connection = makeConnect()  
 cursor = connection.cursor()  
 table\_name = '''"''' + tables[table] + '''"'''  
 print(tables[table])  
 show = 'select \* from public.{}'.format(table\_name)  
 print("SQL query => ", show)  
 print('')  
 cursor.execute(show)  
 records = cursor.fetchall()  
 cursor.close()  
 closeConnect(connection)  
 return records  
  
 @staticmethod  
 def insert\_for\_table1(usname, usbirth\_date, usid):  
 connection = makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 notice = "'This User ID already exists'"  
 insert = 'DO $$ BEGIN if not exists (select "userID" from "User" where "userID" = {}) then INSERT ' \  
 'INTO "User"("userID", "name", "birth date") VALUES ({},{},{}); ' \  
 'raise notice {}; else raise notice {}; ' \  
 'end if; end $$;'.format(usid, usid, usname, usbirth\_date, "'added'", notice)  
 restart = False  
 print('SQl query => ', insert)  
 cursor.execute(insert)  
 connection.commit()  
 print(connection.notices)  
 cursor.close()  
 closeConnect(connection)  
  
 @staticmethod  
 def insert\_for\_table2(poid, poname, potopic, pouser):  
 connection = makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 notice = "'This Ticket ID already exists or this User ID does not exist'"  
 insert = 'DO $$ BEGIN IF EXISTS (select "userID" from "User" where "userID" = {}) and not exists ' \  
 '(select "ticketID" from "Ticket" where "ticketID" = {}) THEN ' \  
 'INSERT INTO "Ticket"("ticketID", "price", "type", "userID" ) values ({}, {}, {}, {}); RAISE NOTICE {};' \  
 ' ELSE RAISE NOTICE {}; END IF; ' \  
 'END $$;'.format(pouser, poid, poid, poname, potopic, pouser, "'added'", notice)  
 restart = False  
 print('SQl query => ', insert)  
 cursor.execute(insert)  
 connection.commit()  
 print(connection.notices)  
 cursor.close()  
 closeConnect(connection)  
  
 @staticmethod  
 def insert\_for\_table3(coid, codate, cotext, copost):  
 connection = makeConnect()  
 cursor = connection.cursor()  
  
 insert = '''  
 DO $$   
 BEGIN   
 IF EXISTS (SELECT "ticketID" FROM "Ticket" WHERE "ticketID" = {})   
 AND NOT EXISTS (SELECT "eventID" FROM "Event" WHERE "eventID" = {})   
 THEN   
 INSERT INTO "Event"("eventID", "date", "purpose", "ticketID")   
 VALUES ({}, '{}', '{}', {});   
 RAISE NOTICE 'added';   
 ELSE   
 RAISE NOTICE 'This Event ID already exists or this Ticket ID does not exist';   
 END IF;   
 END $$;  
 '''.format(coid, coid, coid, codate, cotext, copost)  
 cursor.execute(insert)  
 connection.commit()  
 print(connection.notices)  
 cursor.close()  
 closeConnect(connection)  
  
 @staticmethod  
 def delete\_for\_table1(usid):  
 connection = makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 delete ='delete from "Event" where "ticketID" in (select "ticketID" from "Ticket" where "userID" = {});' \  
 'delete from "Ticket" where "userID" = {};' \  
 'delete from "User" where "userID" = {};'.format(usid, usid, usid)  
  
 restart = False  
 print("SQL query => ", delete)  
 cursor.execute(delete)  
 connection.commit()  
 cursor.close()  
 closeConnect(connection)  
  
 @staticmethod  
 def delete\_for\_table2(poid):  
 connection = makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 delete = 'delete from "Event" where "ticketID" = {};' \  
 'delete from "Ticket" where "ticketID" = {};'.format(poid, poid)  
 restart = False  
 print("SQL query => ", delete)  
 cursor.execute(delete)  
 connection.commit()  
 cursor.close()  
 closeConnect(connection)  
  
 @staticmethod  
 def delete\_for\_table3(coid):  
 connection = makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 delete = 'delete from "User" where "User\_userID" = {};' \  
 'delete from "Event" where "eventID" = {};'.format(coid, coid)  
 restart = False  
 print("SQL query => ", delete)  
 cursor.execute(delete)  
 connection.commit()  
 cursor.close()  
 closeConnect(connection)  
  
 @staticmethod  
 def update\_for\_table1(usid, set):  
 connection = makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 notice = "'There is nothing to update'"  
 update = 'DO $$ BEGIN IF EXISTS (select "userID" from "User" where "userID" = {}) THEN ' \  
 'update "User" set {} where "userID" = {}; ' \  
 'RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF; ' \  
 'END $$;'.format(usid, set, usid, "'updated'", notice)  
 restart = False  
 pass  
 print("SQL query => ", update)  
 cursor.execute(update)  
 connection.commit()  
 print(connection.notices)  
 cursor.close()  
 closeConnect(connection)  
 pass  
  
 @staticmethod  
 def update\_for\_table2(poid, set):  
 connection = makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 notice = "'There is nothing to update'"  
 update = 'DO $$ BEGIN IF EXISTS (select "ticketID" from "Ticket" where "ticketID" = {}) THEN ' \  
 'update "Ticket" set {} where "ticketID" = {}; ' \  
 'RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF; ' \  
 'END $$;'.format(poid, set, poid, "'updated'", notice)  
 restart = False  
 pass  
 print("SQL query => ", update)  
 cursor.execute(update)  
 connection.commit()  
 print(connection.notices)  
 cursor.close()  
 closeConnect(connection)  
 pass  
  
 @staticmethod  
 def update\_for\_table3(coid, set):  
 connection = makeConnect()  
 cursor = connection.cursor()  
 restart = True  
 while restart:  
 notice = "'There is nothing to update'"  
 update = 'DO $$ BEGIN IF EXISTS (select "eventID" from "Event" where "eventID" = {}) THEN ' \  
 'update "Event" set {} where "eventID" = {}; ' \  
 'RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF; ' \  
 'END $$;'.format(coid, set, coid, "'updated'", notice)  
 restart = False  
 pass  
 print("SQL query => ", update)  
 cursor.execute(update)  
 connection.commit()  
 print(connection.notices)  
 cursor.close()  
 closeConnect(connection)  
 pass  
  
 @staticmethod  
 def select1(user):  
 connection = makeConnect()  
 cursor = connection.cursor()  
 select = """select "name", "date", "purpose" from (select c."name", p."date", p."purpose"  
 from "Event" p right join "User" c on c."userID" = p."ticketID"  
 where c."name" LIKE '{}' group by c."name", p."date", p."purpose") as foo""".format(  
 user)  
 print("SQL query => ", select)  
 beg = int(time.time() \* 1000)  
 cursor.execute(select)  
 end = int(time.time() \* 1000) - beg  
 records = cursor.fetchall()  
 print('Time of request = {} ms'.format(end))  
 cursor.close()  
 closeConnect(connection)  
 return records  
  
 @staticmethod  
 def select2(ticket):  
 connection = makeConnect()  
 cursor = connection.cursor()  
 select = """SELECT c."type", p."date", p."purpose"  
 FROM "Ticket" c  
 LEFT JOIN "Event" p ON c."ticketID" = p."ticketID"  
 WHERE c."type" LIKE '{}'""".format(ticket)  
 print("SQL query => ", select)  
 beg = int(time.time() \* 1000)  
 cursor.execute(select)  
 end = int(time.time() \* 1000) - beg  
 records = cursor.fetchall()  
 print('Time of request = {} ms'.format(end))  
 cursor.close()  
 closeConnect(connection)  
 return records  
  
  
  
 @staticmethod  
 def random(table, num):  
 connection = makeConnect()  
 cursor = connection.cursor()  
 incorrect = True  
 while incorrect:  
 if table == 1:  
 insert = """INSERT INTO "User"("userID", "name", "birth date")   
SELECT  
 gs.userID,  
 chr(trunc(65 + random() \* 26)::int) || chr(trunc(65 + random() \* 26)::int),  
 timestamp '2014-01-10 20:00:00' + random() \* (timestamp '2014-01-20 20:00:00' - timestamp '2014-01-10 10:00:00')  
FROM generate\_series(1, 1) AS gs(userID)  
LEFT JOIN "User" u ON gs.userID = u."userID"  
WHERE random() > 0 AND u."userID" IS NULL""".format(num)  
  
 incorrect = False  
 elif table == 2:  
 insert = """INSERT INTO "Ticket" ("ticketID", "price", "type", "userID")  
 SELECT  
 gs.ticketID,  
 trunc(random() \* 100)::int,  
 chr(trunc(65 + random() \* 26)::int) || chr(trunc(65 + random() \* 26)::int),  
 generate\_series(5, 7)  
 FROM generate\_series(8, 10) AS gs(ticketID)  
 LEFT JOIN "Ticket" t ON gs.ticketID = t."ticketID"  
 WHERE t."ticketID" IS NULL  
 AND random() > 0;  
 """.format(num)  
 incorrect = False  
 elif table == 3:  
 event\_id = 2 # Замініть це значення на конкретне eventId  
 insert = """INSERT INTO "Event" ("eventID", "date", "purpose", "ticketID")  
 SELECT  
 {0}, -- Вставте eventID тут  
 timestamp '2023-01-01' + random() \* (timestamp '2023-12-31' - timestamp '2023-01-01'),  
 chr(trunc(65 + random() \* 26)::int) || chr(trunc(65 + random() \* 26)::int),  
 gs.ticketID  
 FROM generate\_series(1, 5) AS gs(ticketID)  
 LEFT JOIN "Event" e ON gs.ticketID = e."ticketID"  
 WHERE e."eventId" IS NULL  
 AND random() > 0;""".format(event\_id)  
 incorrect = False  
 else:  
 print('Incorrect input, try again.')  
 print("SQL query => ", insert)  
 cursor.execute(insert)  
 connection.commit()  
 cursor.close()  
 closeConnect(connection)

**Маємо словник tables для виведення та обирання таблиць за певним числом. Також клас Model, який включає всі потрібні функції:**

**validTable – функція, яка переводить рядок у число, перевіряє наявність такої таблиці та повертає її;**

**showAllTables – функція, яка виводить всі таблиці;**

**showOneTable – функція, яка виводить тільки задану таблицю;**

**insert\_for table1 – функція, яка виконує операцію внесення даних для 1 таблиці;**

**insert\_for table2 – функція, яка виконує операцію внесення даних для 2 таблиці;**

**insert\_for table3 – функція, яка виконує операцію внесення даних для 3 таблиці;**

**delete\_for\_table1 – функція, яка виконує операцію вилучення даних для 1 таблиці;**

**delete\_for\_table2 – функція, яка виконує операцію вилучення даних для 2 таблиці;**

**delete\_for\_table3 – функція, яка виконує операцію вилучення даних для 3 таблиці;**

**update\_for\_table1 – функція, яка виконує операцію редагування даних для 1 таблиці;**

**update\_for\_table2 – функція, яка виконує операцію редагування даних для 2 таблиці;**

**update\_for\_table3 – функція, яка виконує операцію редагування даних для 3 таблиці;**

**select1 – функція, яка виконує перший пошуковий запит;**

**select2 – функція, яка виконує другий пошуковий запит;**

**select3 – функція, яка виконує третій пошуковий запит;**

**random – функція, яка рандомно генерує дані таблиці.**