

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:** «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Github: https://github.com/illya-U/bd\_lab\_2022/tree/develop/lab\_2

Виконав: студент ІІI курсу ФПМ групи КВ-04

Устименко І.В.

Перевірив:

Київ – 2022

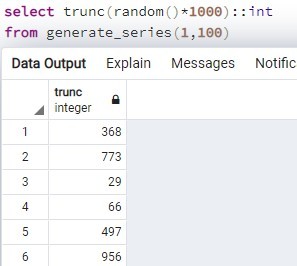
*Загальне завдання* роботи полягає в такому:

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 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**!

Приклад генерації 100 псевдовипадкових чисел:



Приклад генерації 5 псевдовипадкових рядків:



Приклад генерації псевдовипадкової мітки часу з діапазону [доступний](https://stackoverflow.com/questions/22964272/postgresql-get-a-random-datetime-timestamp-between-two-datetime-timestamp/22965061) [за посиланням](https://stackoverflow.com/questions/22964272/postgresql-get-a-random-datetime-timestamp-between-two-datetime-timestamp/22965061).

Кількість даних для генерування має вводити користувач з клавіатури. Для тесту взяти 100 000 записів для однієї-двох таблиць.

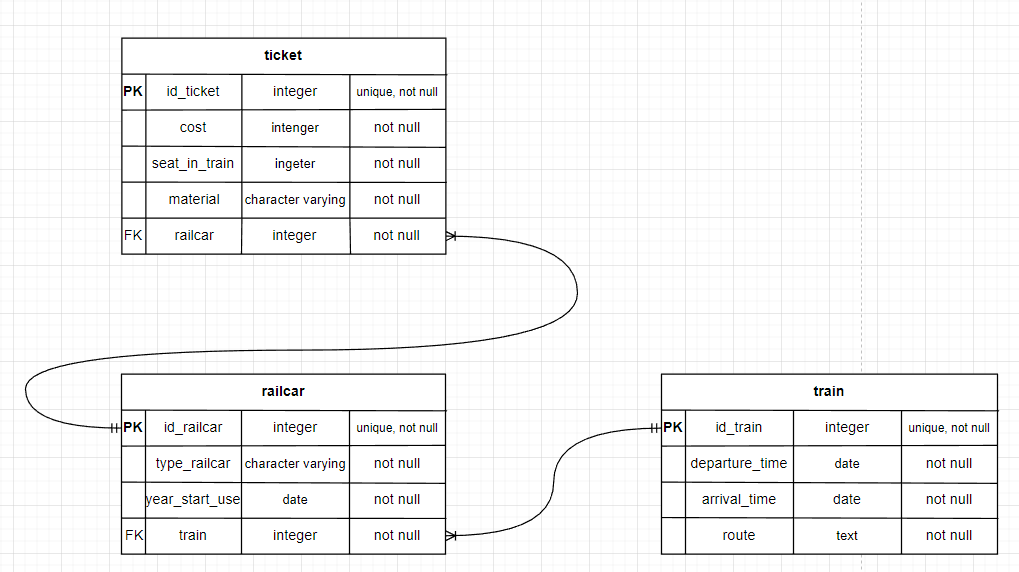
Особливу увагу слід звернути на відповідність даних вимогам зовнішніх ключів з метою уникнення помилок порушення обмежень цілісності foreign key).

1. 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.

1. 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).

Рекомендована бібліотека взаємодії з PostgreSQL Psycopg2: <http://initd.org/psycopg/docs/usage.html)>

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сутність | Атрибут | Опис Атрибуту | Тип | Обмеженя |
| ticket | id | unique identifier | integer | not null  unique |
| cost | ціна в ₴ | integer | not null |
| seat\_in\_the\_train | Місце перебування пасажира протягом  більшої частини шляху | integer | not null |
| material | Матеріал з якого виготовлено квиток | character varying | not null |
| train | посилання на характеристику | integer | not null |
| raicar | id | unique identifier | integer | not null  unique |
| type\_railcar | тип вагону | character  varying | not null |
| year\_start\_use | рік початку експлуатації вагону | date | not null |
| train | посилання на характеристику | integer | not null |
| train | id | unique identifier | integer | not null  unique |
| departure\_time | час відправлення  потягу | date | not null |
| arrival\_time | час прибуття  потягу | date | not null |
| route | Шлях по якому прямує потяг | text | not null |

Опис функціоналу меню:

Update – оновлення данних в певній колонці,

Add – додання нової колонки,

Delete – видалення нової колонки

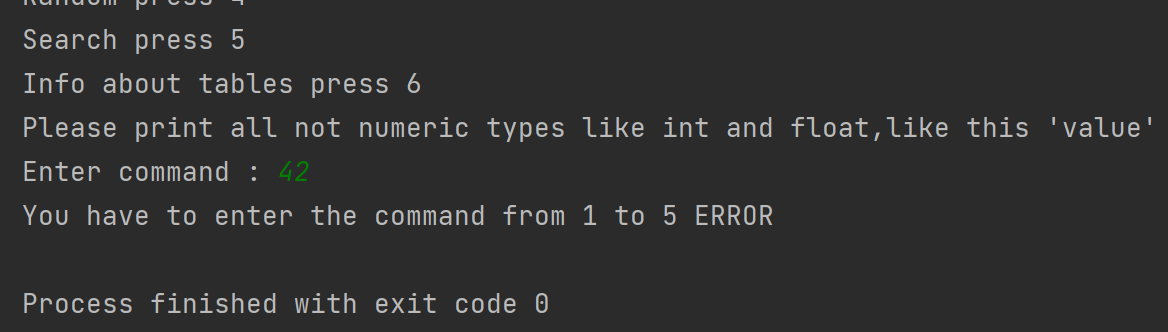
Random - рандомна генерація нових значень для колонок на n раз

Search – пошук по базовим данним якоїсь колонки

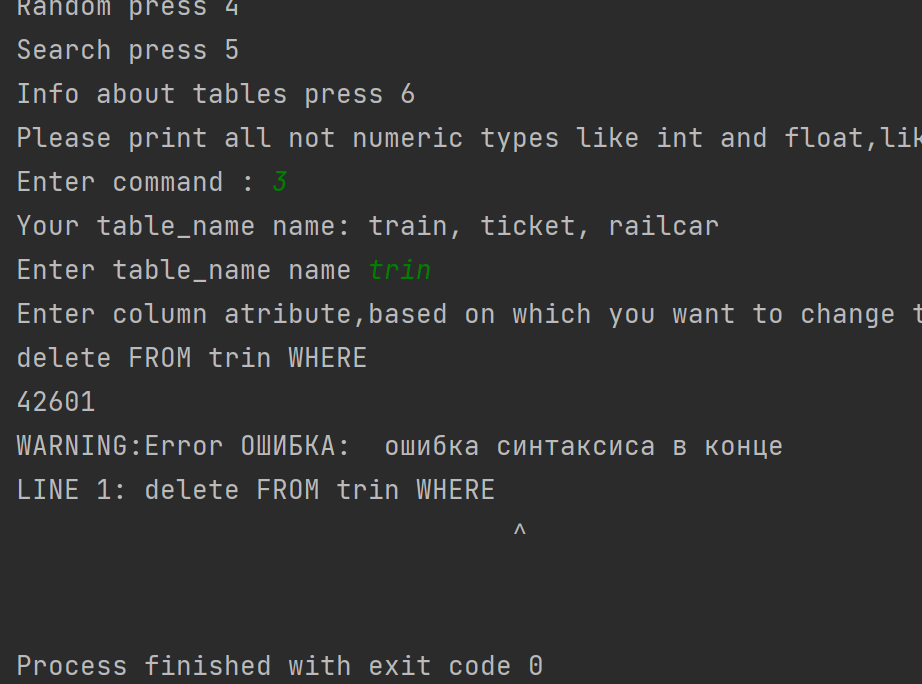
Info about tables – повна інформація по бд

# Відповідь на вимоги до пункту №1 деталізованого завдання:

# Ілюстрації обробки виняткових ситуацій (помилок) при уведенні/вилучення даних:

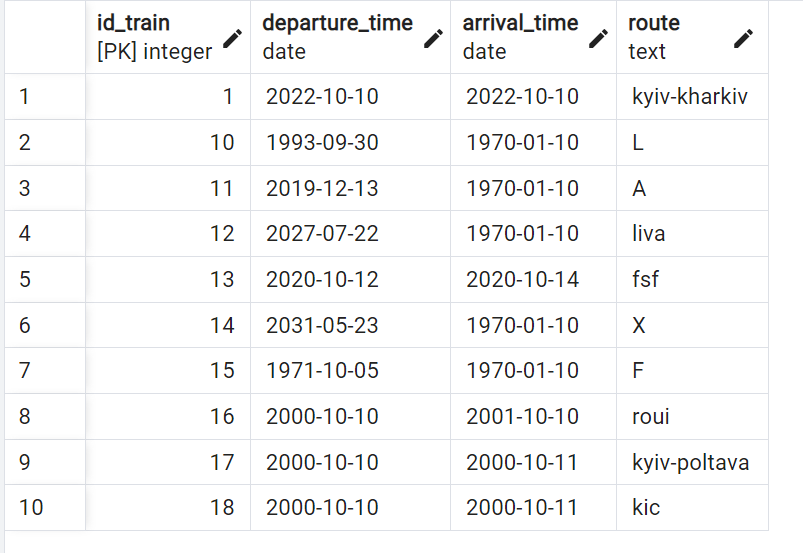


Ілюстрації валідації даних при уведенні користувачем:

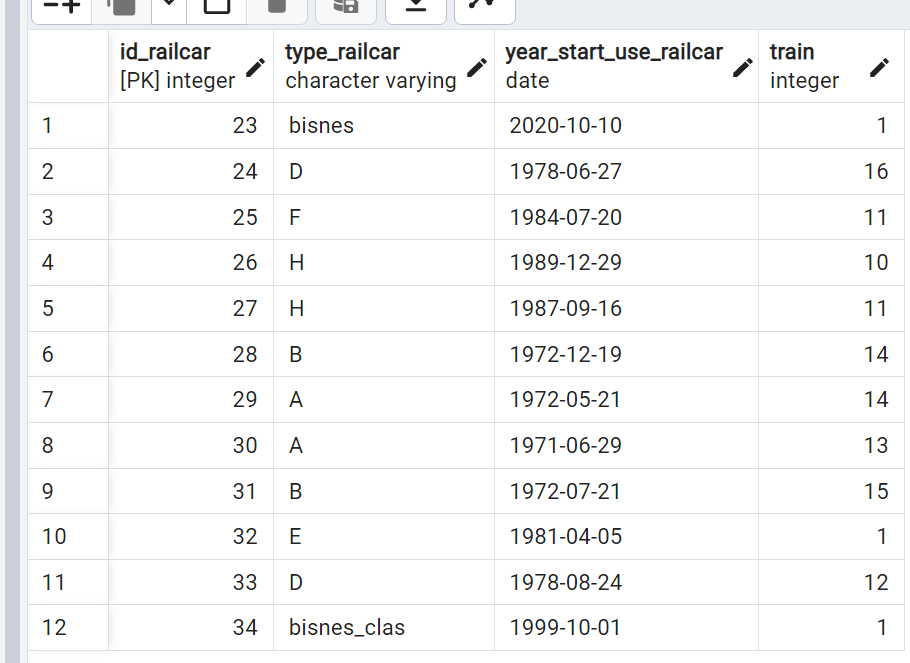


Видалення данних

Даннi потяга до видалення:



Даннi вагона до видалення:



Update press 1

Add press 2

Delete press 3

Random press 4

Search press 5

Info about tables press 6

Please print all not numeric types like int and float,like this 'value'

Enter command : 3

Your table\_name name: train, ticket, railcar

Enter table\_name name train

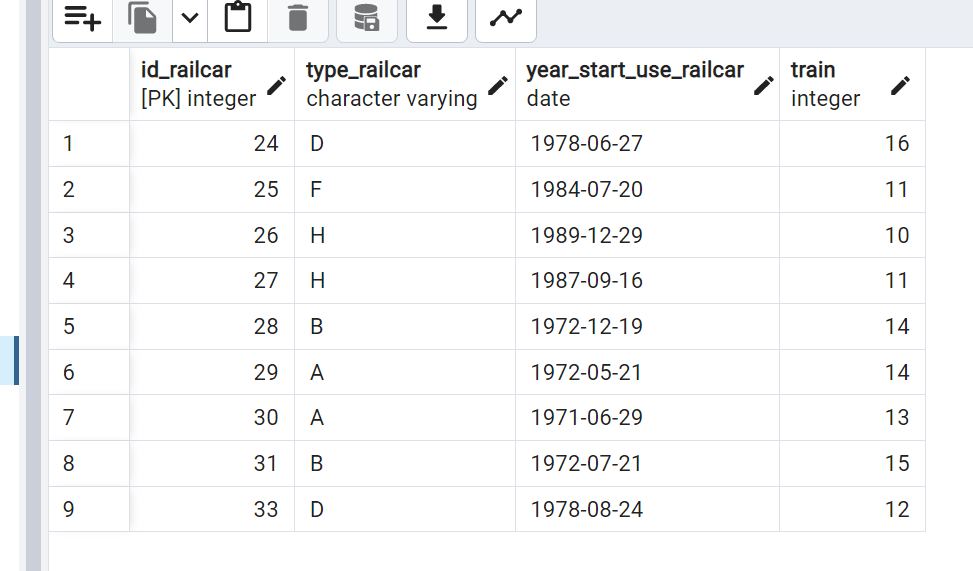
Enter column atribute,based on which you want to change this fileld like 'Column\_name = 'value'' and press Enter,when you want to stop enter columns press '-' and press Enter:id\_train = 1

Enter column atribute,based on which you want to change this fileld like 'Column\_name = 'value'' and press Enter,when you want to stop enter columns press '-' and press Enter:-

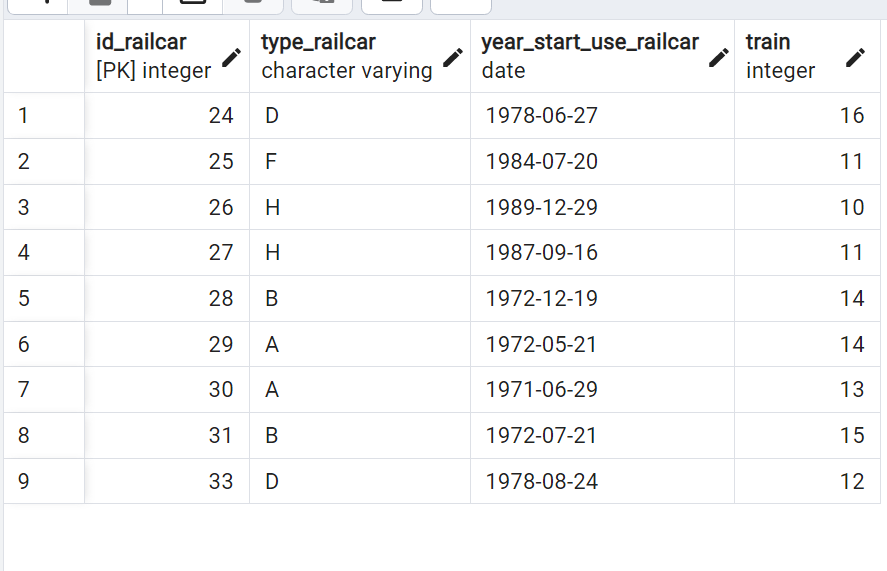
delete FROM train WHERE id\_train = 1

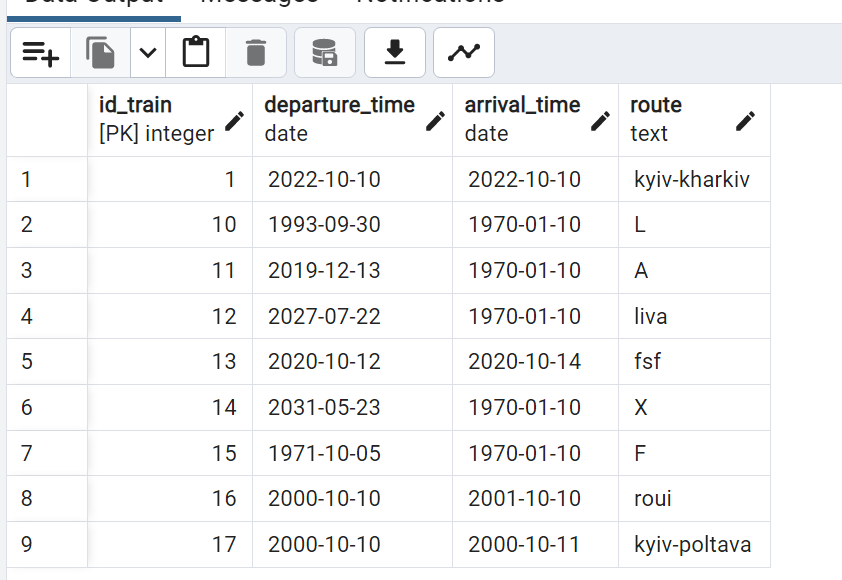
Process finished with exit code 0

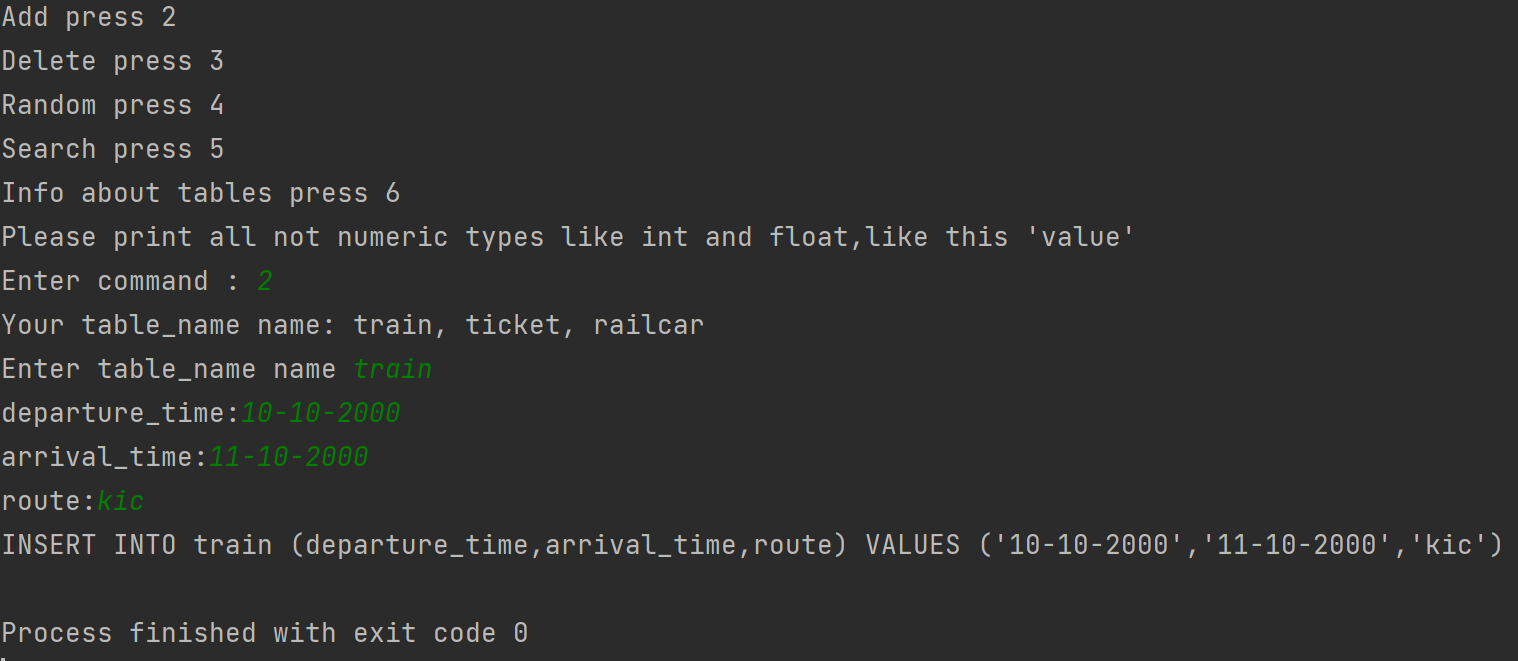
Потяг після видалення:

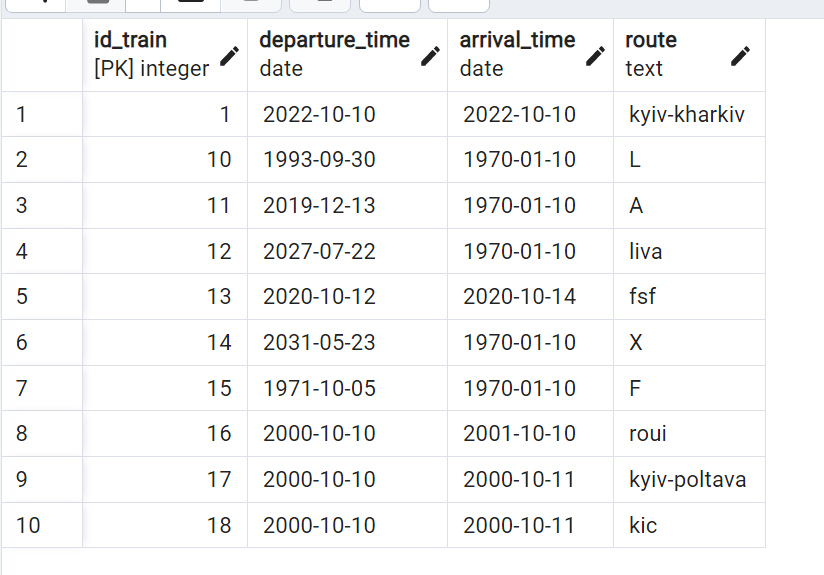


Вагон після видалення:



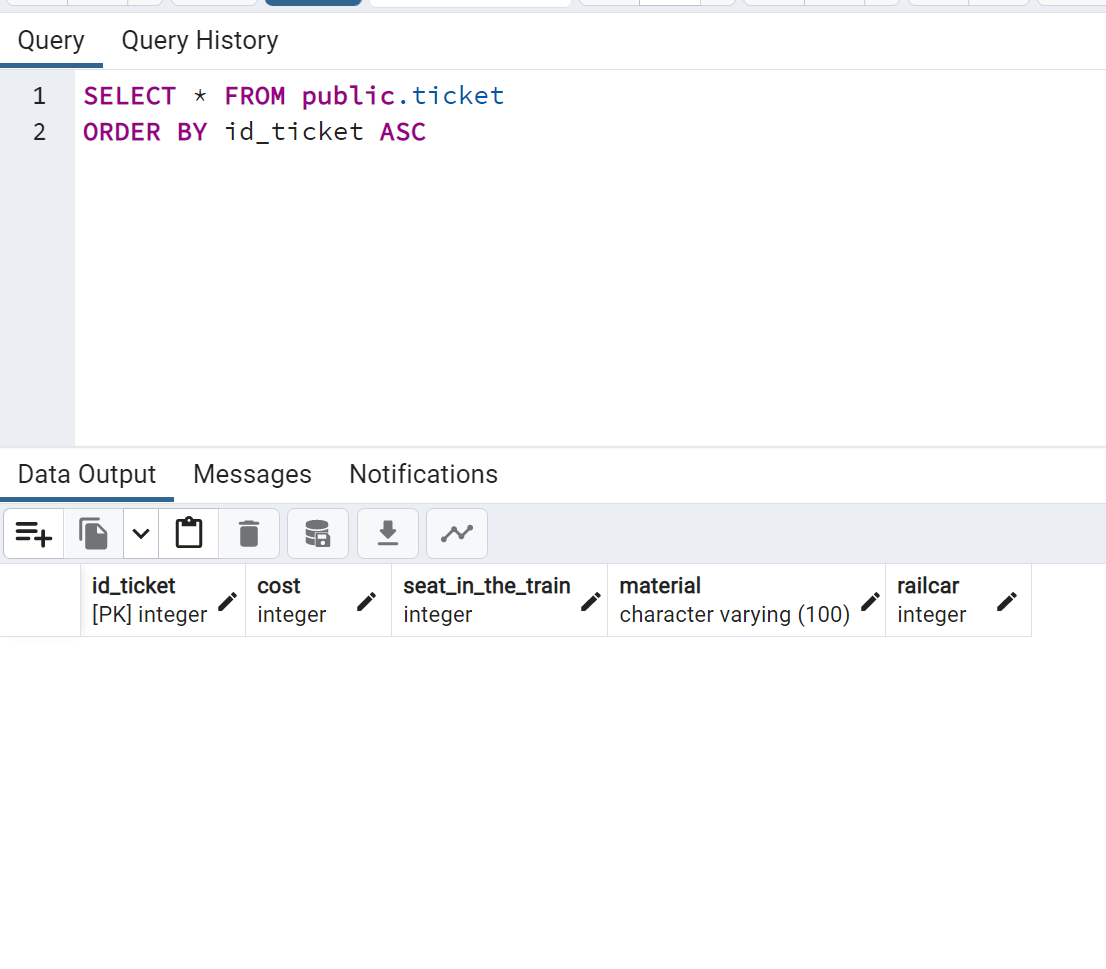
Додавання данних: 

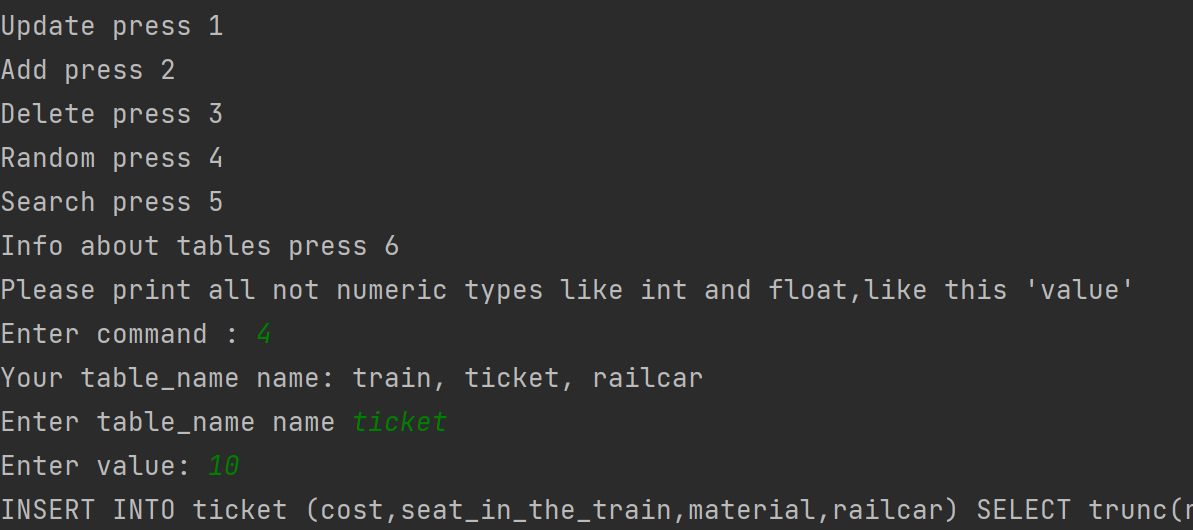
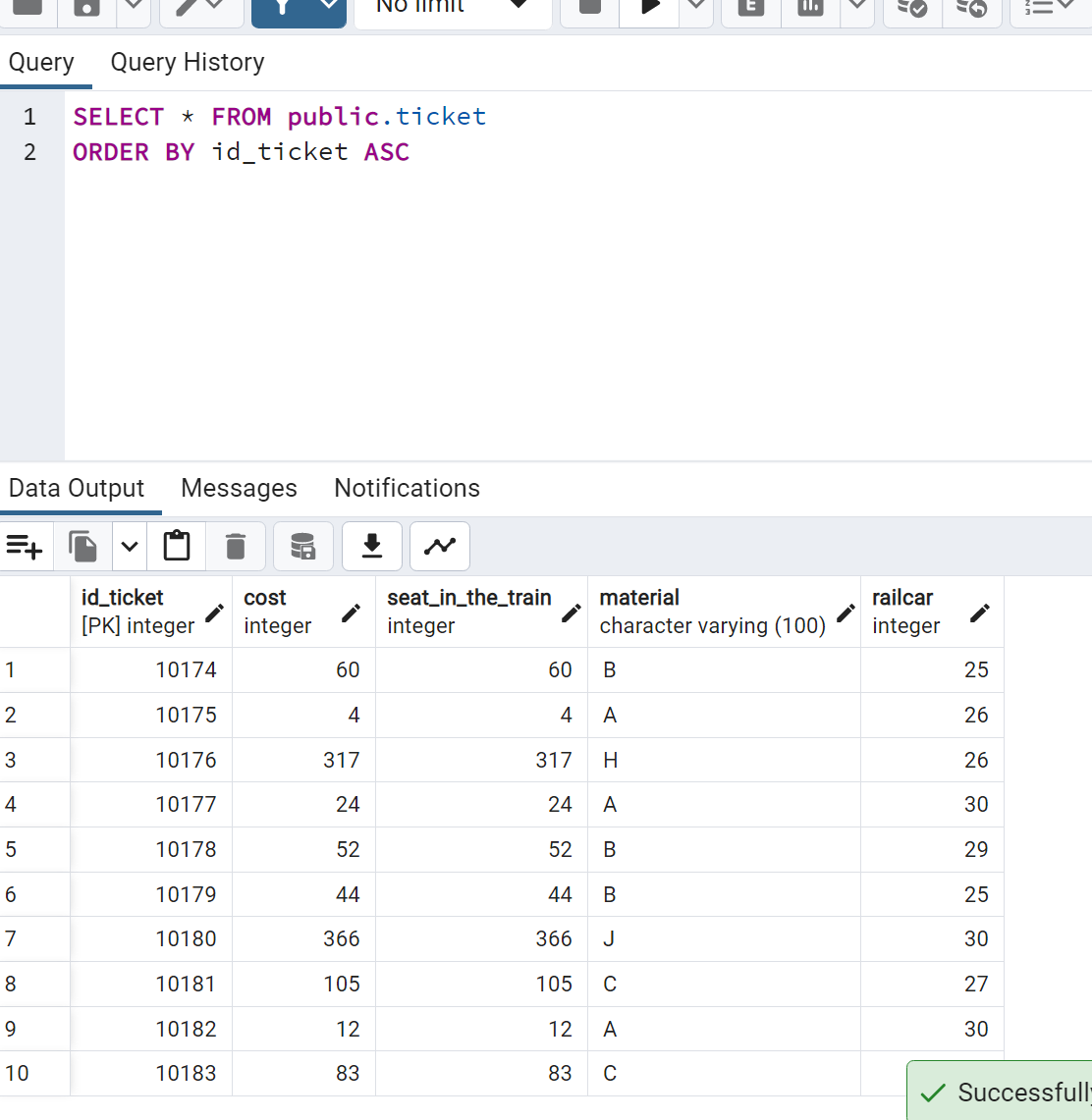




# Вимоги до пункту №2 деталізованого завдання:

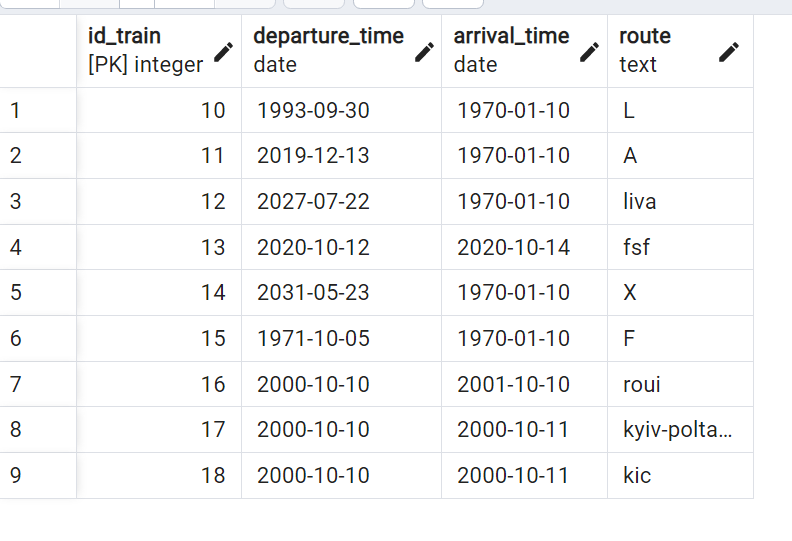
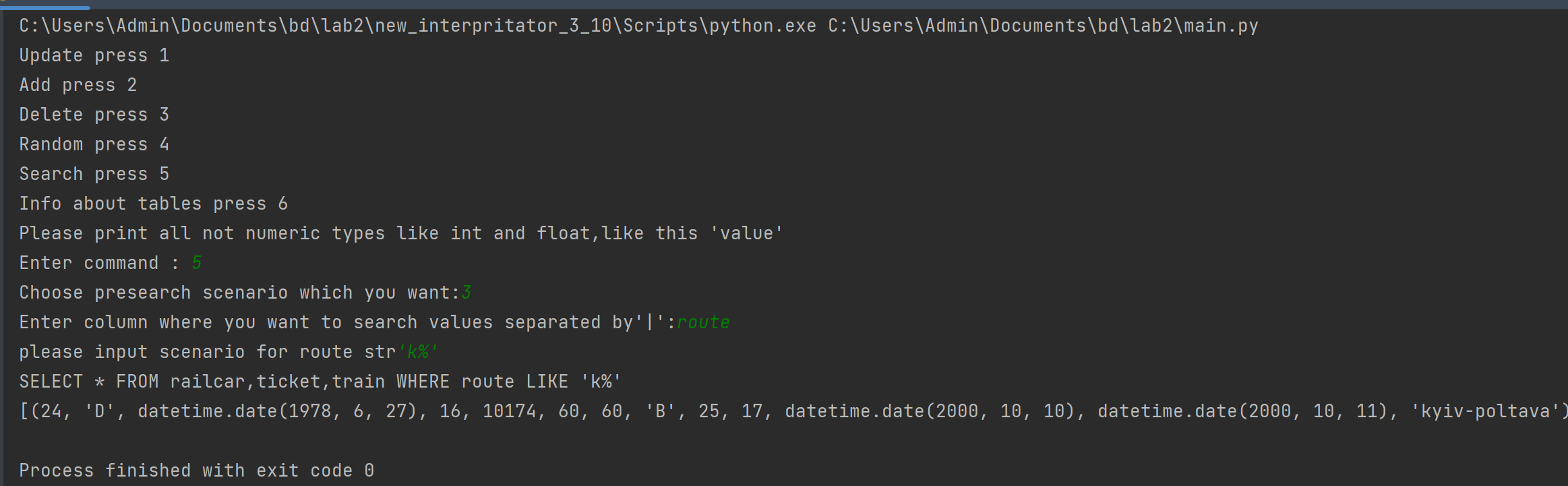
Меню генерації:



# Вимоги до пункту №3 деталізованого завдання:

Ілюстрації уведення пошукового запиту та результатів виконання запитів:

# Вимоги до пункту №4 деталізованого завдання:

Ілюстрації програмного коду з репозиторію Git:

main.py:

import psycopg2  
from psycopg2 import errors  
from model import \*  
from view import \*  
import sys  
  
  
def request():  
 input\_command = comand\_identification()  
  
 if (input\_command == '1'):  
 table = table\_name()  
 str\_of\_columns = input\_colums\_str\_upd()  
 str\_of\_updating\_column= take\_inf\_about\_param(str\_of\_columns)  
 str\_of\_based\_column = take\_inf\_based()  
 updt(table,str\_of\_updating\_column,str\_of\_based\_column)  
 elif (input\_command == '2'):  
 table = table\_name()  
 inf = take\_inf\_for\_adding(table)  
 add\_inf(table,inf)  
 elif (input\_command == '3'):  
 table = table\_name()  
 str\_of\_based\_column = take\_inf\_based()  
 delete(table,str\_of\_based\_column)  
 elif (input\_command == '4'):  
 table = table\_name()  
 data = Data()  
 random(table, data)  
 elif (input\_command == '5'):  
 scenario = choose\_scenario\_search()  
 str\_of\_columns = input\_colums\_str\_search()  
 dict\_of\_searching\_var = take\_searching\_rows(str\_of\_columns)  
 list\_of\_searching = Search(scenario,dict\_of\_searching\_var)  
 print\_searching\_values(list\_of\_searching)  
 elif (input\_command == '6'):  
 print\_info(info())  
 elif (input\_command is not str(range(6))):  
 comndErr()  
 sys.exit()  
  
  
menu()  
  
request()

view.py(функції виводу данних і їх вводу)

import sys  
import pprint  
import additional\_inf  
  
def comndErr():  
 print("You have to enter the command from 1 to 5 ERROR")  
  
  
def comand\_identification():  
 return input('Enter command : ')  
  
def table\_name():  
 print('Your table\_name name: train, ticket, railcar ')  
 return input('Enter table\_name name ')  
  
def print\_info(my\_dict):  
 pprint.pprint(my\_dict)  
  
def input\_colums\_str\_upd():  
 return input("Enter column which you want to update separated by'|':")  
  
def input\_colums\_str\_search():  
 return input("Enter column where you want to search values separated by'|':")  
  
def input\_inf\_for\_column(column\_name):  
 return input(column\_name + ':')  
  
def finding\_fields\_which\_you\_want\_change():  
 return input("Enter column atribute,based on which you want to change this fileld like 'Column\_name = 'value'' and press Enter,when you want to stop enter columns press '-' and press Enter:")  
  
def Data():  
 return input('Enter value: ')  
  
def row():  
 return int(input('Enter value: '))  
  
def tablevalid():  
 print('The table\_name name is wrong ERROR')  
 sys.exit()  
  
def choose\_scenario\_search():  
 scenario = input("Choose presearch scenario which you want:")  
 return (int(scenario) - 1)  
  
def menu():  
 print("Update press 1")  
 print("Add press 2")  
 print("Delete press 3")  
 print("Random press 4")  
 print("Search press 5")  
 print("Info about tables press 6")  
 print("Please print all not numeric types like int and float,like this 'value'")  
  
  
def take\_inf\_based():  
 list\_of\_based\_column = []  
 while (1):  
 temp\_str = finding\_fields\_which\_you\_want\_change()  
 if(temp\_str == "-"):  
 break  
 list\_of\_based\_column.append(temp\_str)  
 str\_of\_based\_column = " AND ".join(list\_of\_based\_column)  
 return str\_of\_based\_column  
  
def take\_inf\_for\_adding(table\_name):  
 mass = []  
 for param in additional\_inf.parametrs[table\_name][1:]:  
 mass.append(input\_inf\_for\_column(param[0]))  
 return mass  
  
def take\_inf\_about\_param(str\_of\_columns):  
 columns = str\_of\_columns.split("|")  
 str\_of\_updating\_column = ""  
 for column in columns:  
 str\_of\_updating\_column += column + " = " + input\_inf\_for\_column(column) + " , "  
 str\_of\_updating\_column = str\_of\_updating\_column.rstrip(", ")  
 return str\_of\_updating\_column  
  
def take\_searching\_rows(str\_of\_columns):  
 columns = str\_of\_columns.split("|")  
 dict\_of\_tacking = {}  
 for column in columns:  
 type\_ = search\_type\_of\_column(column)  
 match type\_:  
 case "int":  
 dict\_of\_tacking[column] = input(f"please input range int numbers for {column} separated by '|'").split("|")  
 case "str":  
 dict\_of\_tacking[column] = input(f"please input scenario for {column} str")  
 case "time":  
 dict\_of\_tacking[column] = input(f"please input range between two dates for {column} separated by '|'").split("|")  
 return dict\_of\_tacking  
  
  
def search\_type\_of\_column(column\_name):  
 for tables in additional\_inf.parametrs:  
 for parametr in additional\_inf.parametrs[tables]:  
 if(column\_name == parametr[0]):  
 return parametr[1]  
 raise("this column is not exist")  
  
def print\_searching\_values(list\_of\_searching):  
 print(list\_of\_searching)

model.py(логіка програми)

import psycopg2  
import additional\_inf  
  
  
def setup\_conf(func):  
 def wrapper(\*args, \*\*kwargs):  
 con = psycopg2.connect(  
 database="railway ticket sales service",  
 user="postgres",  
 password="1111",  
 host="localhost",  
 port="5432"  
 )  
 con.set\_session(autocommit=True)  
 curso\_r = con.cursor()  
 smth = func(con, curso\_r, \*args, \*\*kwargs)  
 curso\_r.close()  
 con.close()  
 return smth  
 return wrapper  
  
  
@setup\_conf  
def info(con, cursor\_r):  
 dict\_of\_all\_tables = {}  
 try:  
 for table in additional\_inf.parametrs:  
 dict\_of\_all\_tables[table] = []  
 flag\_of\_take\_memory = 1  
 for param in additional\_inf.parametrs[table]:  
 param = param[0]  
 cursor\_r.execute(additional\_inf.set\_command["inf"].replace("table", table).replace("param", param))  
 list\_of\_all\_parametrs = cursor\_r.fetchall()  
 if (flag\_of\_take\_memory):  
 for memory\_take in range(len(list\_of\_all\_parametrs)):  
 dict\_of\_all\_tables[table].append({})  
 flag\_of\_take\_memory = 0  
 for i in range(len(list\_of\_all\_parametrs)):  
 dict\_of\_all\_tables[table][i][param] = list\_of\_all\_parametrs[i][0]  
 flag\_of\_take\_memory = 1  
 except psycopg2.Error as err:  
 print(err.pgcode)  
 print(f'WARNING:Error {err}')  
 else:  
 return dict\_of\_all\_tables  
  
@setup\_conf  
def random(con, cursor\_r, table\_name, n):  
 param = ",".join([\*map(lambda x:additional\_inf.parametrs[table\_name][x][0], range(1, len(additional\_inf.parametrs[table\_name])))])  
 try:  
 for i in range(int(n)):  
 take\_sql\_str\_to\_two\_stream(cursor\_r, additional\_inf.set\_command["random"][table\_name].replace("parametrs", param))  
 except psycopg2.Error as err:  
 print(err.pgcode)  
 print(f'WARNING:Error {err}')  
  
@setup\_conf  
def delete(con, cursor\_r, table\_name, str\_of\_based\_column):  
 try:  
 take\_sql\_str\_to\_two\_stream(cursor\_r, additional\_inf.set\_command["delete"].replace("str\_of\_based\_column", str\_of\_based\_column).replace("table\_name", table\_name))  
 except psycopg2.Error as err:  
 print(err.pgcode)  
 print(f'WARNING:Error {err}')  
  
@setup\_conf  
def add\_inf(con, cursor\_r, table\_name, mass):  
 param = ",".join([\*map(lambda x: additional\_inf.parametrs[table\_name][x][0], range(1, len(additional\_inf.parametrs[table\_name])))])  
 mass = [\*map(lambda x: "'{0}'".format(x), mass)]  
 values = ",".join(mass)  
 try:  
 take\_sql\_str\_to\_two\_stream(cursor\_r, additional\_inf.set\_command["add\_inf"].replace("table\_name", table\_name).replace("parametrs", param).replace("values", values))  
 except psycopg2.Error as err:  
 print(err.pgcode)  
 print(f'WARNING:Error {err}')  
  
  
@setup\_conf  
def updt(con, cursor\_r, table\_name, str\_of\_updating\_column, str\_of\_based\_column):  
 try:  
 take\_sql\_str\_to\_two\_stream(cursor\_r, additional\_inf.set\_command["upd\_inf"].replace("table\_name", table\_name).replace("str\_of\_updating\_column", str\_of\_updating\_column).replace("str\_of\_based\_column", str\_of\_based\_column))  
 except psycopg2.Error as err:  
 print(err.pgcode)  
 print(f'WARNING:Error {err}')  
  
  
@setup\_conf  
def Search(con, cursor\_r, scenario, dict\_of\_searching\_var):  
 list\_of\_commands = []  
 for key, value in dict\_of\_searching\_var.items():  
 if(isinstance(value, list)):  
 list\_of\_commands.append(key + " > " + value[0] + " AND " + key + " < " + value[1])  
 else:  
 list\_of\_commands.append(key + " LIKE " + value)  
 commands = ' AND '.join(list\_of\_commands)  
 try:  
 take\_sql\_str\_to\_two\_stream(cursor\_r, additional\_inf.set\_command["presearch"][scenario] + commands)  
 searching\_values = cursor\_r.fetchall()  
 except psycopg2.Error as err:  
 print(err.pgcode)  
 print(f'WARNING:Error {err}')  
 else:  
 return searching\_values  
  
def take\_sql\_str\_to\_two\_stream(cursor\_r, sql\_stream):  
 print(sql\_stream)  
 cursor\_r.execute(sql\_stream)

set\_command(словник для інформації(використовуються як json))

set\_command = \  
 {  
 "random":  
 {  
 "train":"INSERT INTO train (parametrs) SELECT timestamp '1970-01-10 20:00:00' + random()\* (timestamp '2033-01-20 20:00:00' - timestamp '1970-01-10 10:00:00'),timestamp '1970-01-10 20:00:00'+ random() \* (timestamp '1970-01-20 20:00:00' - timestamp '1970-01-20 20:00:00'),chr(trunc(65+random()\*25)::int)",  
 "railcar":f"INSERT INTO railcar (parametrs) SELECT chr(trunc(65+random()\*25)::int), timestamp '1970-01-10 20:00:00'+ random() \* (timestamp '2033-01-20 20:00:00' - timestamp '1970-01-10 10:00:00'),id\_train From train order by random() limit 1",  
 "ticket":f"INSERT INTO ticket (parametrs) SELECT trunc(random()\*1000)::int,trunc(random()\*1000)::int,chr(trunc(65+random()\*25)::int),id\_railcar FROM railcar order by random() limit 1"  
 },  
 "delete":"delete FROM table\_name WHERE str\_of\_based\_column",  
 "add\_inf":"INSERT INTO table\_name (parametrs) VALUES (values)",  
 "upd\_inf":f"UPDATE table\_name SET str\_of\_updating\_column WHERE str\_of\_based\_column;",  
 "inf":"SELECT param FROM table",  
 "presearch":  
 [  
 "SELECT \* FROM train, ticket WHERE ",  
 "SELECT id\_railcar,id\_ticket,material,cost FROM railcar, ticket WHERE ",  
 "SELECT \* FROM railcar,ticket,train WHERE "  
 ]  
 }  
  
parametrs = \  
 {  
 "train":[("id\_train","int"),("departure\_time","time"),("arrival\_time","time"),("route","str")],  
 "railcar":[("id\_railcar","int"),("type\_railcar","str"),("year\_start\_use\_railcar","time"),("train","int")],  
 "ticket":[("id\_ticket","int"),("cost","int"),("seat\_in\_the\_train","int"),("material","str"),("railcar","int")]  
 }

requirements.txt(використанні ліби)

attrs==22.1.0  
colorama==0.4.6  
exceptiongroup==1.0.1  
iniconfig==1.1.1  
packaging==21.3  
pluggy==1.0.0  
psycopg2==2.9.5  
pyparsing==3.0.9  
tomli==2.0.1