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»

Виконав: студент III курсу

ФПМ групи КВ-84

Савицький Я.В.

Перевірив:

Київ – 2020

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з 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**!

Вимоги до оформлення звіту лабораторної роботи у електронному вигляді

Опис (файл README.md) лабораторної роботи у **репозиторії GitHub** включає: назву лабораторної роботи, структуру бази даних з лабораторної роботи №1.

Репозиторій має містить файл звіту у форматі PDF та програмний код файлів мовою Python (або іншою).

Звіт у форматі PDF має містити: титульний аркуш, завдання та відповіді на вимоги до звітування щодо пунктів 1-4 деталізованого завдання:

Вимоги до пункту №1 деталізованого завдання:

- ілюстрації обробки виняткових ситуацій (помилок) при уведенні/вилучення даних;
- ілюстрації валідації даних при уведенні користувачем.

Вимоги до пункту №2 деталізованого завдання:

- копії екрану (ілюстрації) з фрагментами згенерованих даних таблиць;
- копії SQL-запитів, що ілюструють генерацію при визначених вхідних параметрах.

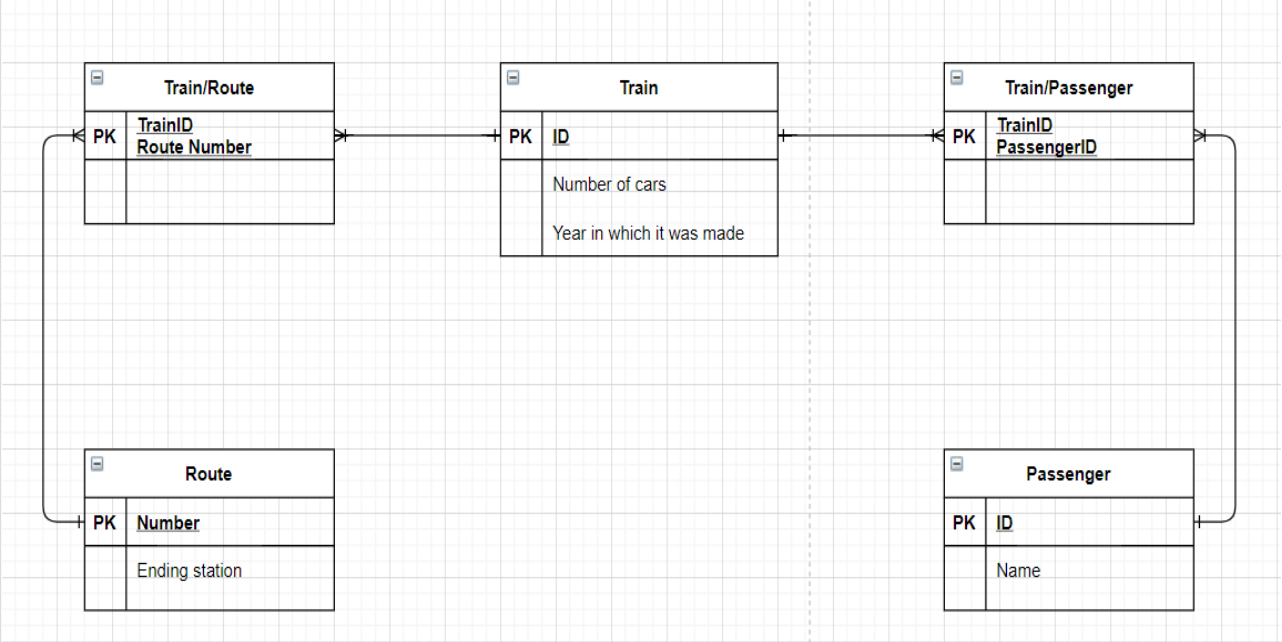
Вимоги до пункту №3 деталізованого завдання:

- ілюстрації уведення пошукового запиту та результатів виконання запитів;
- копії SQL-запитів, що ілюструють пошук з зазначеними початковими параметрами

Вимоги до пункту №4 деталізованого завдання:

- ілюстрації програмного коду з репозиторію Git.

Схема бази даних у графічному вигляді:



Train:

	id [PK] integer	number_of_cars integer	year_in_which_it_was_made integer
1	12	45	1895
2	45	41	1986
3	72	54	1993
4	81	52	1991
5	94	39	1983

Route:

	number [PK] integer	ending_station character varying (100)
1	2	Strasburg
2	3	Paris
3	4	Rome
4	5	Bordeaux
5	8	Barcelona
6	9	Madrid

Passenger:

	id [PK] integer	name character varying (25)
1	12155	Tom
2	12159	Natalia
3	12160	Artem
4	12161	Tanya

Train_Route:

	route_number [PK] integer	train_id [PK] integer
1	2	45
2	3	12
3	4	81
4	5	81
5	8	72
6	9	94

Train_Passenger:

	train_id [PK] integer	passenger_id [PK] integer
1	12	12160
2	45	12155
3	72	12161
4	81	12161
5	94	12159

Завдання 1.

Add row: (train_id = 10)

```
 id  num year
(10, 154, 1875)
(45, 41, 1986)
(72, 54, 1993)
(81, 52, 1991)
(94, 39, 1983)
(12, 45, 1895)
```

Add row: (train_id = 12 – error id)

```
Enter train id: 12
Enter number of cars: 15
Enter year in which it was made: 4848
You enter wrong data! Database has an element whit this primary key
```

Для інших таблиць ілюстрацій я не роблю, тому що там алгоритм той же:

```python
def add_elem(count, param_1, param_2, param_3):
    conn = psycopg2.connect("dbname=Train user=postgres password=")
    cur = conn.cursor()
    try:
        if count == 0:
            cur.execute("INSERT INTO train (id, number_of_cars, year_in_which_it_was_made) VALUES (%s, %s, %s)",
                        (param_1, param_2, param_3))
        elif count == 1:
            cur.execute("INSERT INTO route (number, ending_station) VALUES (%s, %s)", (param_1, param_2))
        elif count == 2:
            cur.execute("INSERT INTO passenger (id, name) VALUES (%s, %s)", (param_1, param_2))
        elif count == 3:
            cur.execute("INSERT INTO train_route (route_number, train_id) VALUES (%s, %s)", (param_1, param_2))
        elif count == 4:
            cur.execute("INSERT INTO train_passenger (train_id, passenger_id) VALUES (%s, %s)", (param_1, param_2))
    except:
        print("You enter wrong data! Database has an element whit this primary key")
    else:
        print("You add element to database!")
    finally:
        conn.commit()
        cur.close()
        conn.close()
```

Delete row: (train_id = 10)

```
 id  num year
(45, 41, 1986)
(72, 54, 1993)
(81, 52, 1991)
(94, 39, 1983)
(12, 45, 1895)
```

Для інших таблиць ілюстрацій я не роблю, тому що там алгоритм той же:

```python
def delete_elem(count, param_1):
    conn = psycopg2.connect("dbname=Train user=postgres password=")
    cur = conn.cursor()
    try:
        if count == 0:
            cur.execute("DELETE FROM train WHERE id = %s", (param_1,))
        elif count == 1:
            cur.execute("DELETE FROM route WHERE number = %s", (param_1,))
        elif count == 2:
            cur.execute("DELETE FROM passenger WHERE id = %s", (param_1,))
        elif count == 3:
            cur.execute("DELETE FROM train_route WHERE train_id = %s", (param_1,))
        elif count == 4:
            cur.execute("DELETE FROM train_passenger WHERE train_id = %s", (param_1,))
    except:
        print("You enter wrong data! Database does not have element whit this primary key")
    else:
        print("You delete element at database!")
    finally:
        conn.commit()
        cur.close()
        conn.close()
```

Update row: (prev_id = 12, new_id = 10)

```
 id  num year
(10, 154, 1548)
(45, 41, 1986)
(72, 54, 1993)
(81, 52, 1991)
(94, 39, 1983)
```

Update row: (prev_id = 45, new_id = 10)

```
Enter new train id: 10
Enter number of cars: 154
Enter year in which it was made: 4474
Enter previously train id: 45
You enter wrong data! Database has element whit this primary key
```

Для інших таблиць ілюстрацій я не роблю, тому що там алгоритм той же:

```
def update_elem(count, param_1, param_2, param_3, param_4):
    # param_4 - it's id_prev
    conn = psycopg2.connect("dbname=Train user=postgres password=")
    cur = conn.cursor()
    try:
        if count == 0:
            cur.execute("UPDATE train SET (id, number_of_cars, year_in_which_it_was_made) = (%s, %s, %s) WHERE id = %s",
                        (param_1, param_2, param_3, param_4,))
        elif count == 1:
            cur.execute("UPDATE route SET (number, ending_station) = (%s, %s) WHERE number = %s",
                        (param_1, param_2, param_4))
        elif count == 2:
            cur.execute("UPDATE passenger SET(id, name) = (%s, %s) WHERE id = %s", (param_1, param_2, param_4))
        elif count == 3:
            cur.execute("UPDATE train_route SET (route_number, train_id) = (%s, %s) WHERE train_id = %s",
                        (param_1, param_2, param_4))
        elif count == 4:
            cur.execute("UPDATE train_passenger SET (train_id, passenger_id) = (%s, %s) WHERE train_id = %s",
                        (param_1, param_2, param_4))
    except:
        print("You enter wrong data! Database has element whit this primary key")
    else:
        print("You update element at database!")
```

Завдання 2.
Генерація 3 рандомних рядків у таблиці train

```
 id  num year
(10, 154, 1548)
(45, 71, 1981)
(75, 25, 657)
(38, 32, 357)
(59, 63, 703)
(72, 54, 1993)
(81, 52, 1991)
(94, 39, 1983)
```

SQL request for train:
"INSERT INTO train (id, number_of_cars, year_in_which_it_was_made) select trunc(random()*100)::int, trunc(random()*100)::int, trunc(random()*1000)::int FROM generate_series(1,3)"
SQL request for route:
"INSERT INTO route (number, ending_station) select trunc(random()*100)::int, chr(trunc(65 + random()*25)::int) || chr(trunc(75 + random()*25)::int) FROM generate_series(1,3)"

SQL request for passenger:
"INSERT INTO passenger (id, name) select trunc(random()*100)::int, chr(trunc(75 + random()*25)::int) || chr(trunc(75 + random()*25)::int) FROM generate_series(1,3)"
SQL request for train_route:
"INSERT INTO train_route (route_number, train_id) select trunc(random()*100)::int, trunc(random()*100)::int FROM generate_series(1,3)"
SQL request for train_passenger:
"INSERT INTO train_passenger (train_id, passenger_id) select trunc(random()*100)::int, trunc(random()*100)::int FROM generate_series(1,3)"

Завдання 3.
Пошук інформації за train_id:

```
Enter train id for searching: 12
 id  num year route_num train_id
(12, 12, 1685, 3, 12)
```

SQL request for train:
"SELECT * FROM train as g1 inner join train_route as s on g1.id = s.train_id where g1.id = %s", (int(param_1),)
SQL request for route:
"SELECT * FROM route as g1 inner join train_route as s on g1.number = s.route_number where g1.number = %s", (int(param_1),)
SQL request for passenger:
"SELECT * FROM passenger as g1 inner join train_passenger as s on g1.id = s.passenger_id where g1.id = %s", (int(param_1),)