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
Бугайов Д.С.
Перевірив:

Київ – 2020

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:

```
select trunc(random()*1000)::int
from generate_series(1,100)
```

Data Output | Explain | Messages | Notific

	trunc integer
1	368
2	773
3	29
4	66
5	497
6	956

Приклад генерації 5 псевдовипадкових рядків:

```
select chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int)
from generate_series(1,5)
```

Data Output | Explain | Messages | Notifications

	?column? text
1	NE
2	MQ
3	RN
4	DW
5	DA

Приклад генерації псевдовипадкової мітки часу з діапазону доступний за посиланням.

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).

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. 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)

Відповідь на вимоги до пункту №1 деталізованого завдання:

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

```
Main menu:
1. Insert row
2. Delete row
3. Update row
4. Generate random rows
5. Search rows
6. Quit
>> 1

Choose the table you want to insert the row:

trolleybus
where_serviced
depot
service
route
what_services
phonebook

>> where_serviced

Input trolleybus_id of the where_serviced >> 2345

Input depot_name of the where_serviced >> Test

INSERT INTO failed: ERROR:  insert or update on table "where_serviced" violates foreign key
 constraint "where_serviced_depot_name_fkey"
DETAIL:  Key (depot_name)=(Test) is not present in table "depot".

Main menu:
1. Insert row
2. Delete row
3. Update row
4. Generate random rows
5. Search rows
6. Quit
>>
```

Ілюстрації валідації даних при уведенні користувачем:

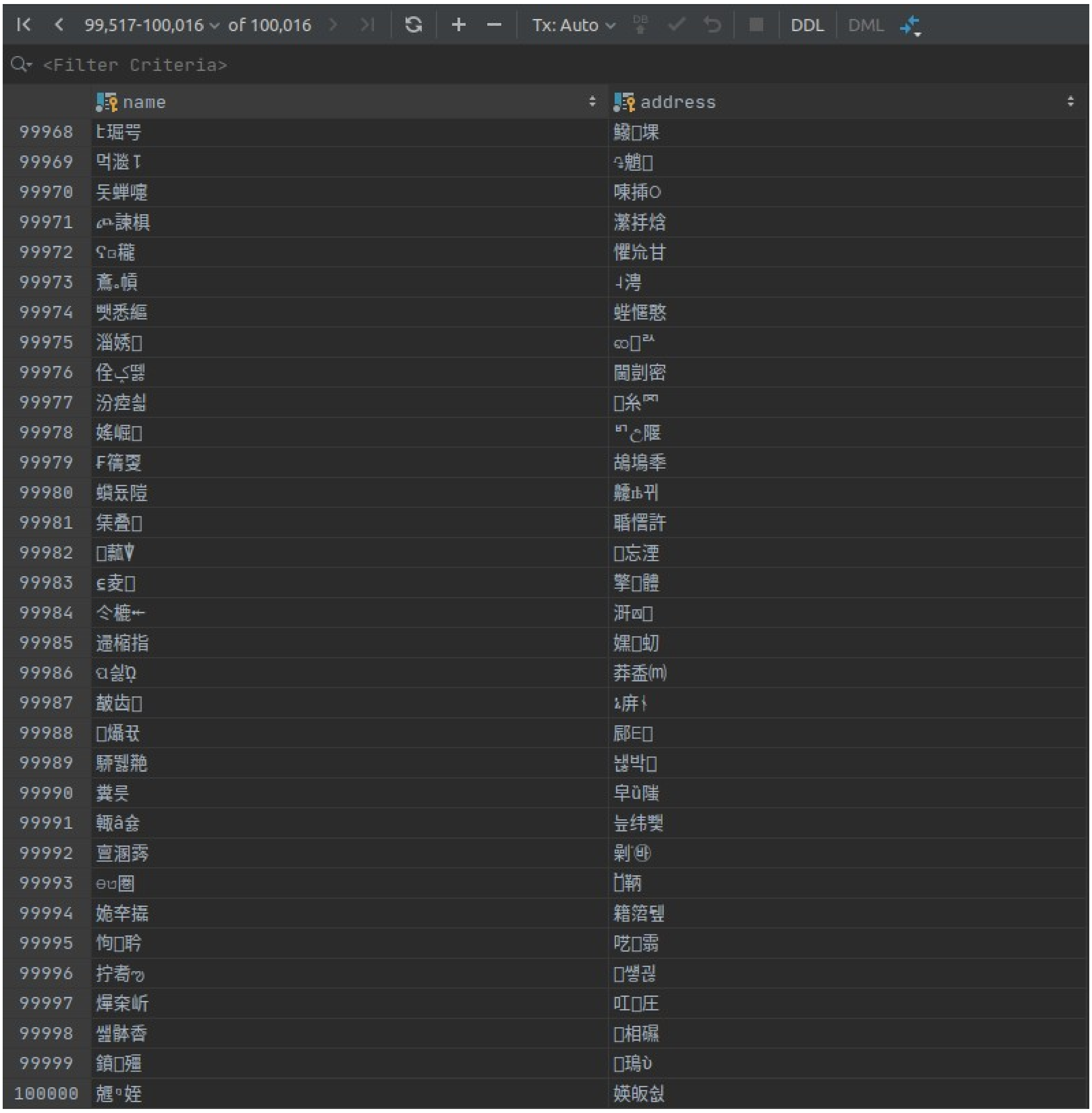
```
Main menu:
1. Insert row
2. Delete row
3. Update row
4. Generate random rows
5. Search rows
6. Quit
>> 5

Input quantity of attributes to search by >> 300

Input name of the attribute number 1 to search by >> id

Input left integer limit of id to search by >> TEST

Input right integer limit of id to search by >>
ERROR: Wrong input, must be integer above 0
```

Вимоги до пункту №2 деталізованого завдання:

Меню генерації:

```
Main menu:
1. Insert row
2. Delete row
3. Update row
4. Generate random rows
5. Search rows
6. Quit
>> 4

Choose the table you want to generate rows:

trolleybus
where_serviced
depot
service
route
what_services
phonebook

>> route

Input quantity of rows to generate >> 2

Generating 2 rows in route successful

Main menu:
1. Insert row
2. Delete row
3. Update row
4. Generate random rows
5. Search rows
6. Quit
>>
```

копії екрану з фрагментами згенерованих даних таблиць:



99,517-100,016 of 100,016 Tx: Auto DDL DML

<Filter Criteria>

	name	address
99968	ㅌ琚믕	鰷□堁
99969	먹滋ꀤ	ꩫ鯱□
99970	둣蝉噻	啭捕ㅇ
99971	ꩡ諌棋	潔抒焓
99972	ꩱ꩜穮	懼巟甘
99973	鳶ₒ幁	ꓧ湾
99974	뻿悉緼	蛘惬憨
99975	淄嫍□	ꩠ□ᆳ
99976	佺ڮ뗧	閫剅密
99977	汾痓쉷	□糸ᄜ
99978	媸崌□	ᄞ ꩭ腝
99979	₣箐叜	鵠塢季
99980	蟦烎隚	鱧ꩬ꿔
99981	枈叠□	暔愲許
99982	□藾₩	□忘湮
99983	ꞓ麦□	擎□軆
99984	令櫨⇠	泔ꩧ□
99985	遈槗指	媒□虭
99986	ɑ읤ꞔ	莽盉⒨
99987	皶齿□	ꩧ庰꡶
99988	□熻퀐	郿ᄐ□
99989	駢뙖艴	냄박□
99990	糞昊	早ꞏ陦
99991	輙â읖	늪纬嬰
99992	亶涠霒	剰ꩡ
99993	ɵꩭ圏	凵鞆
99994	媲卒揺	籍箔텔
99995	恂□聆	咜□霸
99996	拧耆ꩫ	□쌩괭
99997	熚栾岓	叿□压
99998	쐽舑香	□相磥
99999	鐿□殭	□瑦ꞏ
100000	趧ᄆ娙	媖皈쉾

*для генерації великої кількості унікальних ключів було взято широкий алфавіт

Копії SQL-запитів, що ілюструють генерацію при визначених вхідних параметрах:

```
Main menu:
1. Insert row
2. Delete row
3. Update row
4. Generate random rows
5. Search rows
6. Quit
>> 4

Choose the table you want to generate rows:

trolleybus
where_serviced
depot
service
route
what_services
phonebook

>> depot

Input quantity of rows to generate >> 4

INSERT INTO depot SELECT  chr(trunc(65 + random()
 *50000)::int) || chr(trunc(65 + random()*50000)
 ::int) || chr(trunc(65 + random()*50000)::int),
 chr(trunc(65 + random()*50000)::int) ||
 chr(trunc(65 + random()*50000)::int) ||
 chr(trunc(65 + random()*50000)::int) FROM
 generate_series(1,4)

Generating 4 rows in depot successful

Main menu:
1. Insert row
2. Delete row
3. Update row
4. Generate random rows
```

Вимоги до пункту №3 деталізованого завдання:

Ілюстрації уведення пошукового запиту та результатів виконання запитів:

```
Main menu:
1. Insert row
2. Delete row
3. Update row
4. Generate random rows
5. Search rows
6. Quit
>> 5

Input quantity of attributes to search by >> 1

Input name of the attribute number 1 to search by >> trolleybus_id

Input left integer limit of trolleybus_id to search by >> 1

Input right integer limit of trolleybus_id to search by >> 9999


Table: "where_serviced"

trolleybus_id       depot_name

1111        Trolleybus repair-operational depot #2
2345        Trolleybus repair-operational depot #1
6363        Kurenivs'ke Troleybusne Depo


Table: "service"

trolleybus_id       next

1111        2020-11-28
2345        2020-09-25
6363        2020-09-07


Time taken: 11[ms]
```

Копії SQL-запитів, що ілюструють пошук з зазначеними початковими параметрами

```
Main menu:
1. Insert row
2. Delete row
3. Update row
4. Generate random rows
5. Search rows
6. Quit
>> 5

Input quantity of attributes to search by >> 2

Input name of the attribute number 1 to search by
 >> number

Input left integer limit of number to search by >>
 171170153

Input right integer limit of number to search by >>
  171170155

Input name of the attribute number 2 to search by
 >> final_stops

Input string for final_stops to search by >> 玅蜻雲

DECLARE myportal CURSOR FOR SELECT * FROM route
 WHERE number >= '171170153' AND number <
 '171170155' AND final_stops LIKE '玅蜻雲'


Table: "route"

number      final_stops

171170154       玅蜻雲

Time taken: 18[ms]


Main menu:
```

Вимоги до пункту №4 деталізованого завдання:

Ілюстрації програмного коду з репозиторію Git: