



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота № 2

з дисципліни “Реляційні БД”

тема “Створення додатку бази даних, орієнтованого на взаємодію з СУБД
PostgreSQL”

Виконав

студент III курсу

групи КП-81

Подлеснюк Богдан
Анатолійович

Зарахована

“ ____ ” “ ____ ” 20__ р.

викладачем

Радченко К.О.

Київ 2020

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

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі No1, 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 введення/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 підлеглий 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! 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 введення 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>)

Вимоги до інтерфейсу користувача

Використовувати консольний інтерфейс користувача.

Вимоги до інструментарію

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

Мова програмування – Python 3.6-3.7

Середовище розробки програмного забезпечення – PyCharm Community Edition 2020.

Завдання 1

ілюстрації обробки виняткових ситуацій (помилки) при введенні/вилучення даних:

вилучення/оновлення:

Name of table: product

1 - Get

2 - Delete

3 - Update

4 - Insert

5 - Back

```
>> 1
GET product
Enter condition (SQL) or leave empty:
id | name | description | price | total_quantity

-----
1 | TEXTURED CHECK OVERSHIRT |YELLOW - 3057/374 Relaxed fit collared overshirt made of textured fabric. | 2899 | 5786 |
2 | CORDUROY DUNGAREES |BEIGE - 8574/335 Dungarees with adjustable straps and a chest flap pocket. | 2299 | 3143 |
3 |DOUBLE-BREADED COAT WITH BELTDETAILS
|ORANGE - 5928/654 Oversized coat featuring a notched lapel collar, long sleeves, welt pockets at the hip and an inside pocket. | 5999 | 2245 |
4 | SLIM FIT TROUSERS |BLACK - 0706/362 Trousers made of very stretchy fabric. | 1499 | 5677 |
5 |CONTRAST MULTI-PIECE SNEAKERS |MULTICOLOURED - 2241/620 Lace-up sneakers. | 1899 | 7888 |
6 | t1 | desc | 1000 | 1456 |
-----
```

Name of table: product

1 - Get

2 - Delete

3 - Update

4 - Insert

5 - Back

```
>> 2
DELETE product
Enter condition (SQL):
id | name | description | price | total_quantity

id = 6

```

```
>> 2
DELETE product
Enter condition (SQL):
id | name | description | price | total_quantity

id = 6
0

Name of table: product

update or delete on table "product" violates foreign key constraint "product_id_fkey" on table "order_item"
DETAIL:  Key (id)=(6) is still referenced from table "order_item".

1 - Get
2 - Delete
3 - Update
4 - Insert
5 - Back

>> |
```

ілюстрації валідації даних при введенні користувачем:
успішне створення продукту:

```
Name of table: product
```

- 1 - Get
- 2 - Delete
- 3 - Update
- 4 - Insert
- 5 - Back

```

>> 4
INSERT product
Enter columns divided with commas, then do the same for values in format: ['value1', 'value2', ...]
id | name | description | price | total_quantity

name, description, price, total_quantity
'name', 'desc', '4321', '432'

```

Name of table: product

Insert is successful!

- 1 - Get
- 2 - Delete
- 3 - Update
- 4 - Insert
- 5 - Back

неправильний синтаксис створення:

```

>> 4
INSERT product
Enter columns divided with commas, then do the same for values in format: ['value1', 'value2', ...]
id | name | description | price | total_quantity

ada

```

Name of table: product

syntax error at or near ")"

LINE 1: INSERT INTO product (ada) VALUES ();
^

- 1 - Get
- 2 - Delete
- 3 - Update
- 4 - Insert
- 5 - Back

Завдання 2

копії екрану (ілюстрації) з фрагментами згенерованих даних в таблиці product:

```
Select the table to work with | command:
```

- 1 - buyer
- 2 - order_item
- 3 - order
- 4 - product
- 5 - Fill table "product" by random data (10 items)
- 6 - Find buyers by criteria
- 7 - Find orders by criteria
- 8 - Find products by criteria
- 9 - Exit

```
>> 5
```

```
00
```

```
Select the table to work with | command:
```

```
Generated successfully
```

- 1 - buyer
- 2 - order_item
- 3 - order
- 4 - product
- 5 - Fill table "product" by random data (10 items)
- 6 - Find buyers by criteria
- 7 - Find orders by criteria
- 8 - Find products by criteria
- 9 - Exit

```
>>
```

```
>> |
|GET product
Enter condition (SQL) or leave empty:
id | name | description | price | total_quantity
```

id	name	description	price	total_quantity
1	TEXTURED CHECK OVERSHIRT	YELLOW - 3857/374 Relaxed fit collared overshirt made of textured fabric.	2899	5786
2	CORDUROY DUNGAREES	BEIGE - 8574/335 Dungarees with adjustable straps and a chest flap pocket.	2299	3143
3	DOUBLE-BREADED COAT WITH BELTDETAILS	ORANGE - 5928/654 Oversized coat featuring a notched lapel collar, long sleeves, welt pockets at the hip and an inside pocket.	5999	2245
4	SLIM FIT TROUSERS	BLACK - 8706/362 Trousers made of very stretchy fabric.	1499	5677
5	CONTRAST MULTI-PIECE SNEAKERS	MULTICOLOURED - 2241/620 Lace-up sneakers.	1899	7888
6	t1	desc	1000	1456
19	name	desc	4321	432
20	fc341f9782	74887a265f46f96	2782	4299
21	e926bbe58a	a8a4769b2eac7c5	4506	4735
22	e9f40c9d11	ba3c8e0e0082f8	1779	1744
23	2ffeb50040	b88ee9ca00f33fe	3645	6744
24	d57486aa2a	06ca3506e18d458	2449	437
25	7e7311e881	ee7d1024092c6dd	952	5568
26	6b54db2f44	4bad2c2d95e85cb	4462	1537
27	d265d9cfce	ef3742fa2cae329	4242	175
28	887f5cc311	4577a450691a417	973	2062
29	596da2216d	793d06ea0a3dde2	3413	5500
30	8593671239	01ce333d59963af	4688	2008

Завдання 3

ілюстрації введення пошукового запиту та результатів виконання запитів:

1) Пошук клієнтів , що мають ім'я, що починається на 'NAME' та мешкають в місті, назва якого починається на 'CITY', а також мають замовлення більше, ніж на N штук товару.

Select the table to work with | command:

- 1 - buyer
- 2 - order_item
- 3 - order
- 4 - product
- 5 - Fill table "product" by random data (10 items)
- 6 - Find buyers by criteria
- 7 - Find orders by criteria
- 8 - Find products by criteria
- 9 - Exit

>> 6

Find buyers from city that begins on:

Enter city:

0

with name that begins on:

Enter name:

A

Find buyers that have order with amount of product more than:

Enter product amount:

0

- 3 - order
- 4 - product
- 5 - Fill table "product" by random data (10 items)
- 6 - Find buyers by criteria
- 7 - Find orders by criteria
- 8 - Find products by criteria
- 9 - Exit

>> 4

Find buyers from city that begins on:

Enter city:

0

with name that begins on:

Enter name:

0

Find buyers that have order with amount of product more than:

Enter product amount:

0

buyer_id	name	city	order_date	order_id	product_id	amount
1	Alex Johnson	Oakland	2020-09-26	1	1	1
1	Alex Johnson	Oakland	2020-09-26	1	2	1
1	Alex Johnson	Oakland	2020-09-26	1	4	1

time elapsed: 0.0050 seconds

2) Всі замовлення від покупців з ім'ям, що починається на 'NAME' та з міста 'CITY' з поштовим кодом 'POSTAL_CODE'

```
Select the table to work with | command:

1 - buyer
2 - order_item
3 - order
4 - product
5 - Fill table "product" by random data (10 items)
6 - Find buyers by criteria
7 - Find orders by criteria
8 - Find products by criteria
9 - Exit

>> 7
Find orders from buyers from city that begins on:
Enter city:
0
with postal code:
Enter postal code:
94501
with this buyers name starting on:
Enter name:
A

-----
name | email | city | postal_code | order_id | order_date |
-----
Alex Johnson | bebra@gmail.com | Oakland | 94501 | 1 | 2020-09-26 |
-----

time elapsed: 0.0049 seconds
```

3) Всі продукти, на які є замовлення, з вартістю більшою ніж PRICE, загальною кількістю товару на складах TOTAL_QUANTITY та числом бажаної кількості товару для купівлі, щод більше за AMOUNT

Find orders of product that have price more than:

Enter price:

2000

with total quantity of that product more than:

Enter total quantity:

3000

And amount of product in order is more than:

Enter product amount:

0

product_id	name	price	amount	total_quantity
1	TEXTURED CHECK OVERSHIRT	2899	1	5786
2	CORDUROY DUNGAREES	2299	1	3143

time elapsed: 0.0068 seconds

Завдання 4

main.py

```
from controller import Controller

Controller().show_init_menu()
```

controller.py

```
from consolemenu import SelectionMenu

from model import Model
from view import View
import time;

TABLES_NAMES = ['buyer', 'order_item', 'order', 'product']
TABLES = {
    'buyer': ['id', 'email', 'name', 'hashed_password', 'city',
    'postal_code'],
    'order_item': ['id', 'order_id', 'amount', 'product_id'],
    'order': ['id', 'date', 'buyer_id'],
    'product': ['id', 'name', 'description', 'price', 'total_quantity']
}

def getInput(msg, tableName = ''):
    print(msg)
    if tableName:
        print(' | '.join(TABLES[tableName]), end='\n\n')
    return input()

def getInsertInput(msg, tableName):
    print(msg)
    print(' | '.join(TABLES[tableName]), end='\n\n')
    return input(), input()

def pressEnter():
```

```

input()

class Controller:
    def __init__(self):
        self.model = Model()
        self.view = View()

    def show_init_menu(self, msg=''):
        selectionMenu = SelectionMenu(
            TABLES_NAMES + ['Fill table "product" by random data (10 items)',
'Find buyers by criteria',
'Find orders by criteria', 'Find products by criteria'],
title='Select the table to work with | command:', subtitle=msg)
        selectionMenu.show()

        index = selectionMenu.selected_option
        if index < len(TABLES_NAMES):
            tableName = TABLES_NAMES[index]
            self.show_entity_menu(tableName)
        elif index == 4:
            self.fillByRandom()
        elif index == 5:
            self.searchByersByCriteria()
        elif index == 6:
            self.searchOrdersByCriteria()
        elif index == 7:
            self.searchProductsByCriteria()
        else:
            print('Exiting...')

    def show_entity_menu(self, tableName, msg=''):
        options = ['Get', 'Delete', 'Update', 'Insert']
        functions = [self.get, self.delete, self.update, self.insert]

        selectionMenu = SelectionMenu(options, f'Name of table:
{tableName}',
exit_option_text='Back', subtitle=msg)
        selectionMenu.show()
        try:
            function = functions[selectionMenu.selected_option]
            function(tableName)
        except IndexError:
            self.show_init_menu()

    def get(self, tableName):
        try:
            condition = getInput(
f'GET {tableName}\nEnter condition (SQL) or leave empty:',
tableName)
            data = self.model.get(tableName, condition)
            self.view.print(data)
            pressEnter()
            self.show_entity_menu(tableName)
        except Exception as err:
            self.show_entity_menu(tableName, str(err))

    def insert(self, tableName):
        try:
            columns, values = getInsertInput(
f"INSERT {tableName}\nEnter columns divided with commas, then

```

```

do the same for values in format: ['value1', 'value2', ...]", tableName)
    self.model.insert(tableName, columns, values)
    self.show_entity_menu(tableName, 'Insert is successful!')
except Exception as err:
    self.show_entity_menu(tableName, str(err))

def delete(self, tableName):
    try:
        condition = getInput(
            f'DELETE {tableName}\n Enter condition (SQL):', tableName)
        self.model.delete(tableName, condition)
        self.show_entity_menu(tableName, 'Delete is successful')
    except Exception as err:
        self.show_entity_menu(tableName, str(err))

def update(self, tableName):
    try:
        condition = getInput(
            f'UPDATE {tableName}\nEnter condition (SQL):', tableName)
        statement = getInput(
            "Enter SQL statement in format [<key>='<value>']", tableName)

        self.model.update(tableName, condition, statement)
        self.show_entity_menu(tableName, 'Update is successful')
    except Exception as err:
        self.show_entity_menu(tableName, str(err))

def searchByersByCriteria(self):
    try:
        city = getInput(
            'Find buyers from city that begins on: \nEnter city:')
        name = getInput(
            'with name that begins on: \nEnter name:')
        amount = getInput(
            'Find buyers that have order with amount of product more
than: \nEnter product amount:')

        time_start = time.perf_counter()
        data = self.model.searchByersByCriteria(city, name, amount)
        time_end = time.perf_counter()
        self.view.print(data)
        print(f'\ntime elapsed: {time_end-time_start:0.4f} seconds')
        pressEnter()
        self.show_init_menu()
    except Exception as err:
        self.show_init_menu(str(err))

def searchOrdersByCriteria(self):
    try:
        city = getInput(
            'Find orders from buyers from city that begins on: \nEnter
city:')

        postal_code = getInput(
            'with postal code: \nEnter postal code:')
        name = getInput(
            'with this buyers name starting on: \nEnter name:')

        time_start = time.perf_counter()
        data = self.model.searchOrdersByCriteria(city, name,
postal_code)

```

```

        time_end = time.perf_counter()
        self.view.print(data)
        print(f'\ntime elapsed: {time_end - time_start:0.4f}
seconds')

        pressEnter()
        self.show_init_menu()
    except Exception as err:
        self.show_init_menu(str(err))

    def searchProductsByCriteria(self):
        try:
            price = getInput(
                'Find orders of product that have price more than: \nEnter
price:')
            total_quantity = getInput(
                'with total quantity of that product more than: \nEnter
total quantity:')
            amount = getInput(
                'And amount of product in order is more than: \nEnter
product amount:')

            time_start = time.perf_counter()
            data = self.model.searchProductsByCriteria(price, amount,
total_quantity)
            time_end = time.perf_counter()
            self.view.print(data)
            print(f'\ntime elapsed: {time_end - time_start:0.4f}
seconds')

            pressEnter()
            self.show_init_menu()
        except Exception as err:
            self.show_init_menu(str(err))

    def fillByRandom(self):
        try:
            self.model.fillProductByRandomData()
            self.show_init_menu('Generated successfully')

        except Exception as err:
            self.show_init_menu(str(err))

```

model.py

```

import psycopg2

class Model:
    def __init__(self):
        try:
            self.connection = psycopg2.connect(host="localhost",
port="5432",

database='clothingStoreDatabase', user='postgres',
password='qazpl,12345')
            self.cursor = self.connection.cursor()
        except (Exception, psycopg2.Error) as error:
            print("Помилка при з'єднанні з PostgreSQL", error)

```

```

def get_col_names(self):
    return [d[0] for d in self.cursor.description]

def create_db(self):
    f = open("create_db.txt", "r")

    self.cursor.execute(f.read())
    self.connection.commit()

def get(self, tname, condition):
    try:
        query = f'SELECT * FROM {tname}'

        if condition:
            query += ' WHERE ' + condition

        self.cursor.execute(query)
        return self.get_col_names(), self.cursor.fetchall()
    finally:
        self.connection.commit()

def insert(self, tname, columns, values):
    try:
        query = f'INSERT INTO {tname} ({columns}) VALUES ({values});'

        self.cursor.execute(query)
    finally:
        self.connection.commit()

def delete(self, tname, condition):
    try:
        query = f'DELETE FROM {tname} WHERE {condition};'

        self.cursor.execute(query)
    finally:
        self.connection.commit()

def update(self, tname, condition, statement):
    try:
        query = f'UPDATE {tname} SET {statement} WHERE {condition}'

        self.cursor.execute(query)
    finally:
        self.connection.commit()

def searchByersByCriteria(self, city, name, amount):
    query = f'''
        SELECT buyer_id, name, city, date AS order_date,
order_id, product_id, amount
        FROM order_item AS o_item
        JOIN "order" AS o
        ON o.id = o_item.order_id
        JOIN buyer AS b
        ON b.id = o.buyer_id
        WHERE amount > {amount} AND b.city LIKE '{city}%' AND
b.name LIKE '{name}%'
    '''

    try:
        self.cursor.execute(query)
        return self.get_col_names(), self.cursor.fetchall()

```

```

        finally:
            self.connection.commit()

    def searchOrdersByCriteria(self, city, name, postal_code):
        query = f'''
            SELECT name, email, city, postal_code, o.id as order_id,
date as order_date
            FROM buyer AS b
            JOIN "order" AS o
            ON b.id = o.buyer_id
            WHERE b.city LIKE '{city}%' AND b.name LIKE '{name}%' AND
b.postal_code = '{postal_code}';
            '''

        try:
            self.cursor.execute(query)
            return self.get_col_names(), self.cursor.fetchall()
        finally:
            self.connection.commit()

    def searchProductsByCriteria(self, price, amount, total_quantity):
        query = f'''
            SELECT DISTINCT product_id, name, price, amount,
total_quantity
            FROM order_item AS o_item
            JOIN product AS p
            ON o_item.product_id = p.id
            WHERE price > {price} AND amount > {amount} AND
total_quantity > {total_quantity};
            '''

        try:
            self.cursor.execute(query)
            return self.get_col_names(), self.cursor.fetchall()
        finally:
            self.connection.commit()

    def fillProductByRandomData(self):
        sql = """
        CREATE OR REPLACE FUNCTION randomProducts()
        RETURNS void AS $$
        DECLARE
            step integer := 0;
        BEGIN
            LOOP EXIT WHEN step > 10;
                INSERT INTO product (name, description, price,
total_quantity)
                VALUES (
                    substring(md5(random()::text), 1, 10),
                    substring(md5(random()::text), 1, 15),
                    (random() * (5000 - 1) + 1)::integer,
                    (random() * (7000 - 1) + 1)::integer
                );
                step := step + 1;
            END LOOP ;
        END;
        $$ LANGUAGE PLPGSQL;
        SELECT randomProducts();
        """

        try:
            self.cursor.execute(sql)
        finally:

```



```
self.connection.commit()
```

view.py

```
from consolemenu import *
from consolemenu.items import *

class View:
    def print(self, data):
        columns, rows = data
        lineLen = 30 * len(columns)

        self.printSeparator(lineLen)
        self.printRow(columns)
        self.printSeparator(lineLen)

        for row in rows:
            self.printRow(row)
        self.printSeparator(lineLen)

    def printRow(self, row):
        for col in row:
            print(str(col).rjust(26, ' ') + ' |', end='')
        print('')

    def printSeparator(self, length):
        print('-' * length)
```