

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»**

Виконав: студент ІII курсу

ФПМ групи КВ-72

Бербега В.О.

Перевірив(ла):

Київ – 2019

**Ознайомлення з базовими операціями СУБД PostgreSQL**

*Мета роботи:* здобуття практичних навичок проектування та побудови реляційних баз даних та створення прикладних програм з базами даних

*Завдання* роботи полягає у наступному:

1. Виконати нормалізацію бази даних, яка була створена у лабораторній роботі №1, до третьої нормальної форми (3НФ);
2. Реалізувати функціональні вимоги, наведені нижче.

*Функціональні вимоги:*

1. Реалізувати внесення, редагування та вилучення даних у базі засобами консольного інтерфейсу;
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі;
3. Забезпечити реалізацію пошуку за двома-трьома атрибутами з двох сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як перелічення, для логічного типу – значення True/False, для дат – у рамках діапазону дат;
4. Забезпечити реалізацію повнотекстового пошуку за будь-яким текстовим атрибутом бази даних засобами PostgreSQL з виділенням знайденого фрагменту.

*Вимоги до інтерфейсу користувача:*

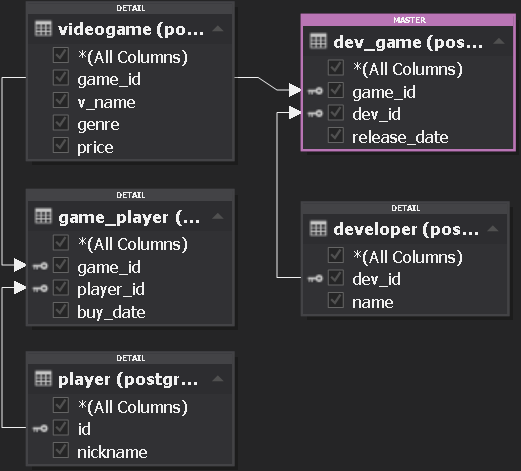
1. Використовувати консольний інтерфейс користувача.

**Варіант 1**

Пошук за атрибутами має відбуватися по переліченню та діапазону чисел.

Повнотекстовий пошук з обов’язковим входженням цілого слова та по цілій фразі.

**Нормалізована модель даних**



**Опис програми**

Програма створена за патерном MVC (Model-View-Controller). Складається відповідно з класів Model , View та Controller.

У класі Model реалізовані функції , що здійснюють SQL запити до Бази Даних, а також функція, що виконує з’єднання з БД.

У класі View реалізовані функції, що використовуються для відображення в консоль пунктів меню, виводу даних з таблиць, тобто функції, що відображують певну інформацію в консоль.

У класі Controller реалізовані функції для відповідних меню та допоміжні функції.

**Лістинг програми**

**Controller**

import os

import re

tables = {'1':'developer',

'2':'player',

'3':'videogame',

'4':'dev\_game',

'5':'game\_player',

}

columns = {'developer':['dev\_id', 'name'],

'player':['id', 'nickname'],

'videogame':['game\_id', 'v\_name', 'genre', 'price'],

'dev\_game':['game\_id', 'dev\_id', 'release\_date'],

'game\_player':['game\_id', 'player\_id', 'buy\_date']

}

columnins = {'developer':['name'],

'player':['nickname'],

'videogame':['v\_name', 'genre', 'price'],

'dev\_game':['game\_id','dev\_id', 'release\_date'],

'game\_player':['game\_id','player\_id', 'buy\_date']

}

fts\_columns = {'developer':['name'],

'player':['nickname'],

'videogame':['v\_name', 'genre'],

}

import model as model

import view as view

def game\_price\_range():

print("Input Range")

print("price more than: ")

lw= input()

print("price less than: ")

hg = input()

where = ("WHERE ((price >= "+lw+")AND(price <= "+hg+")) ORDER BY game\_id")

return(model.select1("videogame", where))

def game\_of\_developers(v\_game):

for temp in v\_game:

print("\n",temp.strip())

try:

yield model.select1("videogame, dev\_game, developer",

"WHERE dev\_game.dev\_id = developer.dev\_id \

AND dev\_game.game\_id = videogame.game\_id \

AND (developer.name ILIKE '{}' \

OR developer.name ILIKE '{}')".format(temp.strip(),temp.strip()), "videogame.game\_id, v\_name, genre, price ")

except (Exception, psycopg2.Error) as error:

print(error)

return

def insert\_into\_table(table\_num):

table = tables[table\_num]

table\_columns = columnins[table]

values = []

try:

for i in range(len(table\_columns)):

answer = input(table\_columns[i] + ' = ')

values.append(answer)

except Exception as error:

print(error)

return

print("INSERT INTO " + table + " VALUES(" + ', '.join(values) + ')')

res = model.insert(table, table\_columns, values)

return res

def delete\_from\_table(table\_num):

table = tables[table\_num]

table\_columns = columns[table]

while True:

print("Choose column to delete by:")

view.table\_columns\_names(table\_num)

chosen\_column\_num = input()

if re.match(r'^[1-{}]{}$'.format(len(table\_columns), "{1}"),

chosen\_column\_num):

chosen\_column = table\_columns[int(chosen\_column\_num)-1]

print("Input value: ")

print("DELETE FROM {} WHERE {} = ...".format(table, chosen\_column))

value = "'" + input() + "'"

print("DELETE FROM {} WHERE {} = {}".format(table,

chosen\_column, value))

where = " WHERE {} = {}".format(chosen\_column, value)

res = model.delete(table, where)

return res

elif chosen\_column\_num == '0':

return

else:

print("No such option. Check your input")

def update\_table(table\_num):

table = tables[table\_num]

table\_columns = columns[table]

while True:

print("Choose column to update:")

view.table\_columns\_names(table\_num)

chosen\_column\_num = input()

if re.match(r'^[1-{}]{}$'.format(len(table\_columns), "{1}"),

chosen\_column\_num):

set\_column = table\_columns[int(chosen\_column\_num)-1]

print("Input value: ")

print("UPDATE {} SET {} = ...".format(table, set\_column))

set\_value = "'" + input() + "'"

print("Choose column to update by:")

view.table\_columns\_names(table\_num)

chosen\_column\_num = input()

if re.match(r'^[1-{}]{}$'.format(len(table\_columns), "{1}"),

chosen\_column\_num):

where\_column = table\_columns[int(chosen\_column\_num)-1]

print("Input value: ")

print("UPDATE {} SET {} = {} WHERE {} = ...".format(table,

set\_column, set\_value, where\_column))

where\_value = "'" + input() + "'"

print("UPDATE {} SET {} = {} WHERE {} = {}".format(table,

set\_column, set\_value, where\_column, where\_value))

set = " SET {} = {}".format(set\_column, set\_value)

where = " WHERE {} = {}".format(where\_column, where\_value)

res = model.update(table, set, where)

return res

elif chosen\_column\_num == '0':

return

else:

print("No such option. Check your input")

elif chosen\_column\_num == '0':

return

else:

print("No such option. Check your input")

def fts\_table(text, mode, table\_num):

table = tables[table\_num]

to\_tsvector = fts\_columns[table]

where = ' || '.join("to\_tsvector(coalesce({}, ''))".format(w)

for w in to\_tsvector)

where += " @@ plainto\_tsquery('{}')".format(text)

return(model.full\_text\_search(table, where, mode))

def main\_menu():

while True:

view.show\_main\_menu()

option = input()

if re.match(r'^[1-5]{1}$', option):

while True:

view.tables\_names()

chosen\_table = input()

if re.match(r'^[1-6]{1}$', chosen\_table):

table = view.tables[chosen\_table]

if option == '1':

table = tables[chosen\_table]

table\_columns = columns[table]

notes = model.select(table, table\_columns[0])

view.print\_table(chosen\_table, notes)

elif option == '2':

res = insert\_into\_table(chosen\_table)

if not res:

print("Data wasn't inserted")

else:

print("Successfully inserted")

elif option == '3':

res = update\_table(chosen\_table)

if not res:

print("Data wasn't updated")

else:

print("Operation successfull")

elif option == '4':

res = delete\_from\_table(chosen\_table)

if not res:

print("Data wasn't deleted")

else:

print("Operation successfull")

elif option == '5':

text = input("Input text to search: ")

view.fts\_mode()

mode = input()

if re.match(r'^[1,2]{1}$', mode):

notes = fts\_table(text, mode, chosen\_table)

view.print\_table(chosen\_table, notes)

elif back\_to\_menu == '1':

break

else:

print("No such option. Check your input")

elif chosen\_table == '0':

break

else:

print("No such option. Check your input")

view.back\_to\_menu()

back\_to\_menu = input()

if back\_to\_menu == '0':

continue

elif back\_to\_menu == '1':

break

else:

print("No such option. Check your input")

elif option == '6':

view.print\_table('3', game\_price\_range())

elif option == '7':

dev = input("Input authors names separated with comma: ")

dev = dev.split(', ')

for dev\_list in game\_of\_developers(dev):

view.print\_table('3', dev\_list)

elif option == '8':

num\_of\_rand = input("Number of random players\n")

res = model.random\_author(num\_of\_rand)

if not res:

print("Data wasn't updated")

else:

print("Successfully updated")

elif option == '9':

os.system("cls")

elif option == '0':

exit()

else:

print("No such option. Check your input")

if \_\_name\_\_ == "\_\_main\_\_":

main\_menu() **View**

import os

from controller import tables, columns

def show\_main\_menu():

print('''\n+=+=+=+=+=+=+=+Main menu:=+=+=+=+=+=+=+

1. Select

2. Insert

3. Update

4. Delete

5. Full text search

6. Games in price range

7. Game of developer

8. Random player

9. Clear screen

0. Exit\t\n+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+''')

def fts\_mode():

print('''\n+=+=+=+=+=+=+Search mode:=+=+=+=+=+=+=+

1. Required word entry

2. Full phrase

0. Back to menu\n+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+''')

def back\_to\_menu():

print('''\n+=+=+=+=+=Back to main menu:+=+=+=+=+=+

1. Yes

0. No\n+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+''')

def table\_columns\_names(table):

table\_columns = columns[tables[table]]

print('+='\*20)

for i in range(len(table\_columns)):

print(str(i+1) + ". " + table\_columns[i])

print("0. Back to menu")

print('+='\*20)

def tables\_names():

print('+='\*20)

print("Choose table:")

for k, v in tables.items():

print("{}. {}".format(k, v))

print("0. Back to menu")

print('+='\*20)

def print\_table(table, notes):

if not notes:

print('+='\*20)

print('No data')

print('+='\*20)

return

print()

table\_columns = columns[tables[table]]

print('+='\*20)

for note in notes:

for i in range(len(note)):

print(table\_columns[i] + ' -', note[i])

print('+='\*20)

**Model**

import psycopg2

import random, string

import datetime

def connect\_db():

try:

res = psycopg2.connect(host="localhost", port="5432",

database="postgres", user="postgres",

password='password')

except (Exception, psycopg2.Error) as error:

res = False

print(error)

return res

def select(table, id = "", fields = "\*", where = ""):

conn = connect\_db()

if not conn: return False

cursor = conn.cursor()

try:

cursor.execute("SELECT " + fields + " FROM " + table + ' ' + where + "ORDER BY " + id +" ASC")

res = cursor.fetchall()

except (Exception, psycopg2.Error) as error:

res = False

print(error)

conn.commit()

cursor.close()

conn.close()

return res

def select1(table, where = "", fields = "\*"):

conn = connect\_db()

if not conn: return False

cursor = conn.cursor()

try:

cursor.execute("SELECT " + fields + " FROM " + table + ' ' + where)

res = cursor.fetchall()

except (Exception, psycopg2.Error) as error:

res = False

print(error)

conn.commit()

cursor.close()

conn.close()

return res

def insert(table, fields = "", values = ""):

conn = connect\_db()

if not conn: return False

cursor = conn.cursor()

if type(values) is list:

values = ["'{0}'".format(x) for x in values]

values = ', '.join(values)

if type(fields) is list:

fields = ', '.join(fields)

try:

cursor.execute("INSERT INTO " + table + '(' + fields + ')' +

" VALUES " + '(' + values + ')')

res = True

except (Exception, psycopg2.Error) as error:

res = False

print(error)

conn.commit()

cursor.close()

conn.close()

return res

def delete(table, where = ""):

conn = connect\_db()

if not conn: return False

cursor = conn.cursor()

try:

cursor.execute("DELETE FROM " + table + ' ' + where)

res = True

except (Exception, psycopg2.Error) as error:

res = False

print(error)

conn.commit()

cursor.close()

conn.close()

return res

def update(table, set = "", where = ""):

conn = connect\_db()

if not conn: return False

cursor = conn.cursor()

try:

cursor.execute("UPDATE " + table + ' ' + set + ' ' + where)

res = True

except (Exception, psycopg2.Error) as error:

res = False

print(error)

conn.commit()

cursor.close()

conn.close()

return res

def random\_author(num):

conn = connect\_db()

if not conn: return False

cursor = conn.cursor()

try:

cursor.execute("insert into player (nickname) select \* FROM rand\_player({})".format(num))

res = True

except (Exception, psycopg2.Error) as error:

res = False

print(error)

conn.commit()

cursor.close()

conn.close()

return res

def full\_text\_search(table, where, mode):

conn = connect\_db()

if not conn: return False

cursor = conn.cursor()

try:

if mode == '1':

cursor.execute("select \* from {} where {}".format(table, where))

res = cursor.fetchall()

elif mode == '2':

cursor.execute("select \* from {} where {}".format(table, where))

res = cursor.fetchall()

except (Exception, psycopg2.Error) as error:

res = None

print(error)

conn.commit()

cursor.close()

conn.close()

return res

**Скріншот результатів виконання операції вилучення**

Спроба вилучення кортежу на який є посилання

