

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 КП-72

Кобрин Данило Романович

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 9**

Пошук за атрибутами має відбуватися по логічному типу та текстовому

Повнотекстовий пошук з обов’язковим входженням цілого слова та по не входженню.

**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.

**Опис структури меню програми**

Меню програми можна розглядати як її концептуальну модель

|  |
| --- |
| import psycopg2  from controller import Controller  def array\_to\_tuple(schema\_array):  str\_schema = '('  for x in range(0, len(schema\_array) - 1):  str\_schema = str\_schema + schema\_array[x] + ', '  str\_schema = str\_schema + schema\_array[len(schema\_array) - 1]  str\_schema = str\_schema + ')'  return str\_schema  class DBOperations(Controller):  schemas = {'teams': ['name', 'city', 'cr\_date'],  'game': ['score\_a', 'score\_b', 'match\_date'],  'player': ['age', 'first\_name', 'surname', 'phone\_number', 'is\_active'],  'stadium': ['capacity', 'address', 'phone\_number']  }  text\_cols = ['name', 'city', 'first\_name', 'surname', 'address']  def \_\_init\_\_(self):  try:  self.connection = psycopg2.connect(user="postgres",  password="111",  host="127.0.0.1",  port="5432",  database="Sports")  self.cursor = self.connection.cursor()  except (Exception, psycopg2.Error) as error:  print ("Error while connecting to PostgreSQL", error)  def disconnect(self):  if self.connection:  self.cursor.close()  self.connection.close()  print("PostgreSQL connection is closed") |
| def get\_schemas(self):  return self.schemas  def read(self, table, conditions, limit):  query = "select \* from {0}" \  " WHERE {1} " \  " LIMIT {2}" \  .format(table, conditions, limit)  self.cursor.execute(query)  rec = self.cursor.fetchall()  return rec  def insert(self, table, data):  if table in self.schemas:  query = "insert into {0}" \  " values {1}" \  .format(table + array\_to\_tuple(self.schemas[table]), data)  res = self.cursor.execute(query)  self.connection.commit()  return True  else:  raise Exception('SQL error during insert operation')  def update(self, table, columns, condition):  if table in self.schemas:  query = "update {0}" \  " set {1}" \  " where {2}" \  .format(table, columns, condition)  self.cursor.execute(query)  self.connection.commit()  return True  else:  raise Exception('SQL error during update operation') |
| '''  pass table name and condition as in 'id>1' or 'name=John'  '''  def delete(self, table, data\_condition):  if table in self.schemas:  query = "delete from {0}" \  " where {1}" \  .format(table, data\_condition)  res = self.cursor.execute(query)  self.connection.commit()  return True  else:  raise Exception('SQL error during delete operation')  def fulltext\_search(self, table, column, text\_piece):  if column in self.text\_cols and table in self.schemas:  query = "Select \* \  from {0} where \  to\_tsvector('english', {1}) @@ to\_tsquery('english', '{2}')".format(table, column, text\_piece)  self.cursor.execute(query)  rec = self.cursor.fetchall()  return rec  else:  raise Exception('Cannot do full-text search on non-text column')  def no\_fulltext\_search(self, table, column, text\_piece):  if column in self.text\_cols and table in self.schemas:  query = "Select \* \  from {0} where \  to\_tsvector('english', {1}) @@ to\_tsquery('english', '!{2}')".format(table, column, text\_piece)  self.cursor.execute(query)  rec = self.cursor.fetchall()  return rec  else:  raise Exception('Cannot do full-text search on non-text column') |
| from controller import Controller  from conView import concrete\_view  from db\_crud import DBOperations  import randomTableFiller  dbo = DBOperations()  controller = Controller()  controller.set\_model(dbo)  view = concrete\_view()  view.set\_data(controller.get\_schemas())  def get\_table\_class(table\_chosen):  if table\_chosen == 'teams':  return randomTableFiller.RandomTeam  if table\_chosen == 'game':  return randomTableFiller.RandomGame  if table\_chosen == 'player':  return randomTableFiller.RandomPlayer  if table\_chosen == 'stadium':  return randomTableFiller.RandomStadium  def tuple\_from\_table\_dictionary(table\_name, table\_dictionary):  schema = controller.get\_schemas()[table\_name]  data\_tuple = "("  for x in range(0, len(schema) - 1):  data\_tuple += table\_dictionary[schema[x]] + ", "  data\_tuple += table\_dictionary[schema[len(schema) - 1]]  data\_tuple += ")"  return data\_tuple  def key\_value\_string\_from\_dictionary(dictionary):  kv\_string = ""  for elem in dictionary:  kv\_string += elem + '=' + dictionary[elem]  if elem != dictionary.keys()[-1]:  kv\_string += ', '  return kv\_string |
| def run():  while True:  table\_chosen = view.launch\_table\_prompt()  if table\_chosen == 'Exit':  break  operation\_chosen = view.launch\_operation\_prompt()  try:  if operation\_chosen != 'Back':  if operation\_chosen == 'Preview':  data = view.launch\_select\_prompt(table\_chosen)  res = controller.read(table\_chosen, data['cond'], data['limit'])  view.display\_select\_data(res, table\_chosen)  if operation\_chosen == 'Add':  data\_dict = view.launch\_insert\_prompt(table\_chosen)  data = tuple\_from\_table\_dictionary(table\_chosen, data\_dict)  controller.insert(table\_chosen, data)  view.display\_insert\_data(data, table\_chosen)  if operation\_chosen == 'Add Random':  data\_dict = get\_table\_class(table\_chosen).get\_random()  data = tuple\_from\_table\_dictionary(table\_chosen, data\_dict)  controller.insert(table\_chosen, data)  view.display\_insert\_data(data, table\_chosen)  if operation\_chosen == 'Delete':  data = view.launch\_delete\_prompt(table\_chosen)  res = controller.delete(table\_chosen, data)  view.display\_delete\_data(data, table\_chosen)  if operation\_chosen == 'Update':  data = view.launch\_update\_prompt(table\_chosen)  columns = key\_value\_string\_from\_dictionary(data['columns'])  controller.update(table\_chosen, columns, data['condition']) |
| view.display\_update\_data(data['columns'], data['condition'], table\_chosen)  if operation\_chosen == 'Text search':  data = view.launch\_text\_search\_prompt(table\_chosen)  if data['in'] == True:  res = controller.fulltext\_search(table\_chosen, data['col'], data['textp'])  else:  res = controller.no\_fulltext\_search(table\_chosen, data['col'], data['textp'])  view.display\_select\_data(res, table\_chosen)  if operation\_chosen == 'Joint search':  data = view.launch\_joint\_search\_prompt()  res = controller.joint\_search(data['player\_is\_active'], data['team\_name'])  view.display\_select\_data(res, table\_chosen)  except Exception as error:  print('Error in model: ' + error.message)  controller.force\_commit()  controller.disconnect() |

Приклади роботи програми



