

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра програмного забезпечення та комп’ютерних систем

**Лабораторна робота №2**

з дисципліни

**«Бази даних і засоби управління»**

Виконав: студент ІII курсу

ФПМ групи КП-73

Литвиненко Антон

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.

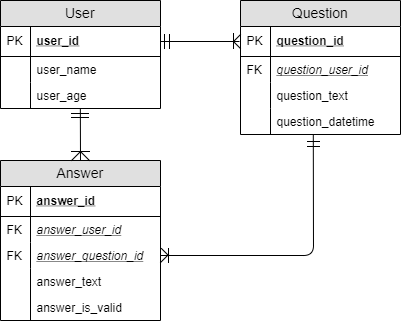
**Варіант 13**

Пошук за атрибутами має відбуватися по діапазону дат та логічному типу.

Повнотекстовий пошук зі «слово не входить» та за цілою фразою.

**Нормалізована модель даних**

(була у 3НФ ще у лаб1)



Таблиця User знаходиться у 1 нормальній формі , бо всі поля мають тільки атомарні значення. Таблиця User знаходиться у 2НФ, бо знаходиться у 1НФ та кожний її неключовий атрибут функціонально повністю залежить від первинного ключа user\_id. Таблиця User знаходиться у 3НФ , бо знаходиться у 2НФ та кожний її атрибут нетранзитивно залежить від первинного ключа.

Таблиця Question знаходиться у 1 нормальній формі , бо всі поля мають тільки атомарні значення. Таблиця Question знаходиться у 2НФ, бо знаходиться у 1НФ та кожний її неключовий атрибут функціонально повністю залежить від первинного ключа question\_id. Таблиця Question знаходиться у 3НФ , бо знаходиться у 2НФ та кожний її атрибут нетранзитивно залежить від первинного ключа.

Таблиця Answer знаходиться у 1 нормальній формі , бо всі поля мають тільки атомарні значення. Таблиця Answer знаходиться у 2 НФ, бо знаходиться у 1НФ та кожний її неключовий атрибут функціонально повністю залежить від первинного ключа answer\_id. Таблиця Answer знаходиться у 3НФ , бо знаходиться у 2НФ та кожний її атрибут нетранзитивно залежить від первинного ключа.

**Опис програми**

Програма створена за патерном MVC (Model-View-Controller). Складається відповідно з модулів model , view та controller.

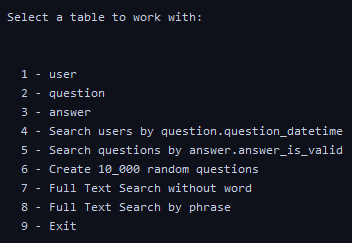
У модулі model реалізовані функції , що здійснюють SQL запити до Бази Даних.

У модулі view реалізовані функції виводу даних з таблиць та зчитування даних з клавіатури.

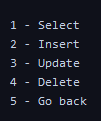
У модулі controller реалізовані функції для відповідних меню та допоміжні функції.

**Опис структури меню програми**

Меню програми можна розглядати як її концептуальну модель



Пункти 1-3:



Пункт 4-5: пошук по діапазону дат та по логічному типу відповідно.

Пункт 6: створення 10 000 випадкових Запитань.

Пункт 7: повнотексовий пошук без входження слова

Пункт 8: повнотекстовий пошук за фразою

**Лістинг програми**

*from* controller *import* display\_main\_menu  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 display\_main\_menu()

**Controller**

*from* consolemenu *import* SelectionMenu  
  
*import* model  
*import* view  
  
  
*def* display\_main\_menu(err='', table=*None*):  
 tables = *list*(model.TABLES.keys())  
  
 menu = SelectionMenu(tables + ['Search users by question.question\_datetime',  
 'Search questions by answer.answer\_is\_valid',  
 'Create 10\_000 random questions',  
 'Full Text Search without word',  
 'Full Text Search by phrase'], subtitle=err,  
 title="Select a table to work with:")  
 menu.show()  
  
 index = menu.selected\_option  
 *if* index < *len*(tables):  
 table = tables[index]  
 display\_secondary\_menu(table)  
 *elif* index == *len*(tables):  
 search\_users\_by\_question\_date()  
 *elif* index == *len*(tables) + 1:  
 search\_questions\_by\_answer\_is\_valid()  
 *elif* index == *len*(tables) + 2:  
 create\_random\_questions()  
 *elif* index == *len*(tables) + 3:  
 full\_text\_search\_without\_word()  
 *elif* index == *len*(tables) + 4:  
 full\_text\_search\_by\_phrase()  
 *else*:  
 *print*('Bye! Have a nice day!')  
  
  
*def* display\_secondary\_menu(table, subtitle=''):  
 opts = ['Select', 'Insert', 'Update', 'Delete']  
 steps = [select, insert, update, delete, display\_main\_menu]  
  
 menu = SelectionMenu(  
 opts, subtitle=subtitle,  
 title=f'Selected table "{table}"', exit\_option\_text='Go back',)  
 menu.show()  
 index = menu.selected\_option  
 steps[index](table=table)  
  
  
*def* select(table):  
 query = view.multiple\_input(table, 'Enter requested fields:')  
 data = model.get(table, query)  
 view.print\_entities(table, data)  
 view.press\_enter()  
 display\_secondary\_menu(table)  
  
  
*def* insert(table):  
 data = view.multiple\_input(table, 'Enter new fields values:')  
 model.insert(table, data)  
 display\_secondary\_menu(table, 'Insertion was made successfully')  
  
  
*def* update(table):  
 condition = view.single\_input(  
 table, 'Enter requirement of row to be changed:')  
 query = view.multiple\_input(table, 'Enter new fields values:')  
  
 model.update(table, condition, query)  
 display\_secondary\_menu(table, 'Update was made successfully')  
  
  
*def* delete(table):  
 query = view.multiple\_input(  
 table, 'Enter requirement of row to be deleted:')  
  
 model.delete(table, query)  
 display\_secondary\_menu(table, 'Deletion was made successfully')  
  
  
*def* search\_users\_by\_question\_date():  
 dates = view.specified\_input(msg='Enter datetime range divided in format <dd/mm/yyyy hh:mm:ss>-<dd/mm/yyyy hh:mm:ss>:').split('-')  
 data = model.search\_users\_by\_question\_date(dates)  
 view.print\_entities(f'User who wrote questions in date range={dates}', data)  
 view.press\_enter()  
 display\_main\_menu()  
  
  
*def* search\_questions\_by\_answer\_is\_valid():  
 query = view.specified\_input(  
 'is\_admin', 'Enter answer.is\_valid value:'  
 ).lower() *in* ['true', 't', 'yes', 'y', '+']  
  
 data = model.search\_questions\_by\_answer\_is\_valid(query)  
 view.print\_entities(  
 f'Questions which where answered with answer.is\_valid={query}:', data)  
 view.press\_enter()  
 display\_main\_menu()  
  
  
*def* full\_text\_search\_without\_word():  
 query = view.specified\_input(  
 'word', 'Enter your word to be absent in the document:')  
 data = model.full\_text\_search\_without\_word(query)  
 view.print\_entities(  
 f'Documents without word=`{query}`', data)  
 view.press\_enter()  
 display\_main\_menu()  
  
  
*def* full\_text\_search\_by\_phrase():  
 query = view.specified\_input(  
 'phrase', 'Enter your phrase to be found in document:')  
 data = model.full\_text\_search\_by\_phrase(query)  
 view.print\_entities(  
 f'Documents with phrase=`{query}`', data)  
 view.press\_enter()  
 display\_main\_menu()  
  
  
*def* create\_random\_questions():  
 model.create\_random\_questions()  
 display\_main\_menu('10\_000 random questions were successfully added')

**View**

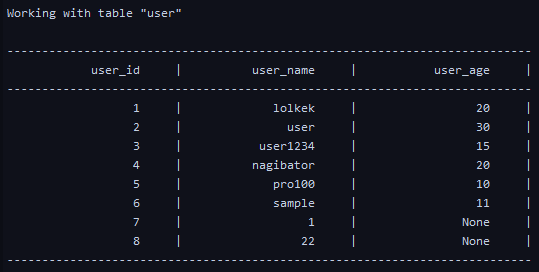
*import* model  
  
COLUMN\_WIDTH = 25  
  
  
*def* print\_entities(table, data):  
 entities, cols = data  
  
 separator\_line = '-' \* COLUMN\_WIDTH \* *len*(cols)  
  
 *print*(f'Working with table "{table}"', end='\n\n')  
 *print*(separator\_line)  
 *print*(''.join([f'{col} |'.rjust(COLUMN\_WIDTH, ' ') *for* col *in* cols]))  
 *print*(separator\_line)  
  
 *for* entity *in* entities:  
 *print*(''.join([f'{col} |'.rjust(COLUMN\_WIDTH, ' ') *for* col *in* entity]))  
 *print*(separator\_line)  
  
  
*def* specified\_input(colname=*None*, msg=*None*):  
 *if* msg:  
 *print*(msg)  
 *if* colname:  
 *print*(f'{colname}=', end='')  
 *return input*()  
  
  
*def* single\_input(tname, msg):  
 *print*(msg)  
 *print*('(use format <attribute>=<value>)')  
 *print*(f'({"/".join(model.TABLES[tname])})', end='\n\n')  
  
 *while True*:  
 data = *input*()  
 *if not* data *or* data.count('=') != 1:  
 *print*('Invalid input, try one more time')  
 *continue* data = data.split('=')  
 col, val = data[0].strip(), data[1].strip()  
 *if* col.lower() *in* [tcol.lower() *for* tcol *in* model.TABLES[tname]]:  
 *return* col, val  
 *else*:  
 *print*(f'Invalid column name "{col}" for table "{tname}"')  
  
  
*def* multiple\_input(tname, msg):  
 *print*(msg)  
 *print*('(use format <attribute>=<value>)')  
 *print*(f'({"/".join(model.TABLES[tname])})', end='\n\n')  
  
 res = {}  
 *while True*:  
 data = *input*()  
 *if not* data:  
 *break  
 if* data.count('=') != 1:  
 *print*('Invalid input')  
 *continue* data = data.split('=')  
 col, val = data[0].strip(), data[1].strip()  
 *if* col.lower() *in* [tcol.lower() *for* tcol *in* model.TABLES[tname]]:  
 res[col] = val  
 *else*:  
 *print*(f'Invalid column name "{col}" for table "{tname}"')  
  
 *return* res  
  
  
*def* multiline\_input(msg):  
 *print*(msg, end='\n\n')  
  
 lines = []  
 *while True*:  
 line = *input*()  
 *if not* line:  
 *break* lines.append(line)  
  
 *return* '\n'.join(lines)  
  
  
*def* press\_enter():  
 *input*()

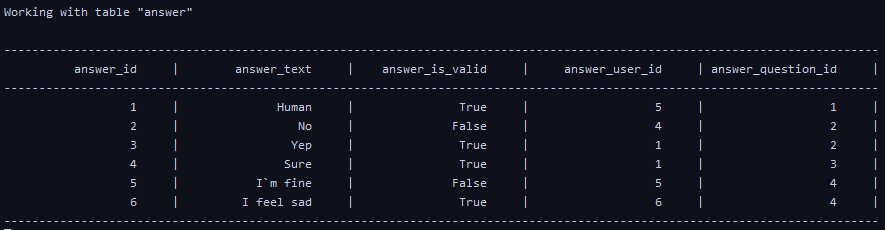
**Model**

*import* psycopg2  
  
conn = psycopg2.connect("dbname='labs' user='anton'"  
 "host='localhost' password='password'")  
cursor = conn.cursor()  
  
TABLES = {  
 'user': ('user\_id', 'user\_name', 'user\_age'),  
 'question': ('question\_id', 'question\_text', 'question\_datetime', 'question\_user\_id'),  
 'answer': ('answer\_id', 'answer\_text', 'answer\_is\_valid', 'answer\_user\_id', 'answer\_question\_id')  
}  
  
  
*def* insert(table\_name, opts):  
 *try*:  
 cols = opts.keys()  
 vals = [f"'{val}'" *for* val *in* opts.values()]  
 command = f'insert into "{table\_name}" ({", ".join(cols)}) ' + \  
 f'values ({", ".join(vals)})'  
 cursor.execute(command)  
 *finally*:  
 conn.commit()  
  
  
*def* get(table\_name, opts=*None*):  
 command = f'select \* from "{table\_name}"'  
  
 *if* opts:  
 conditions = [f"{col}='{opts[col]}'" *for* col *in* opts]  
 command = f'{command} where {" and ".join(conditions)}'  
  
 cursor.execute(command)  
 *return* cursor.fetchall(), TABLES[table\_name]  
  
  
*def* update(table\_name, condition, opts):  
 *try*:  
 column, value = condition  
 updates = ', '.join([f"{col} = '{opts[col]}'" *for* col *in* opts])  
 command = f'update "{table\_name}" set {updates} where {column}=\'{value}\''  
  
 cursor.execute(command)  
 conn.commit()  
 *finally*:  
 conn.commit()  
  
  
*def* delete(table\_name, opts):  
 *try*:  
 conditions = [f"{col}='{opts[col]}'" *for* col *in* opts]  
 command = f'delete from "{table\_name}" where {" and ".join(conditions)}'  
 cursor.execute(command)  
 *finally*:  
 conn.commit()  
  
  
*def* search\_questions\_by\_answer\_is\_valid(is\_valid):  
 command = f'''  
 select question\_id, question\_text, question\_datetime, question\_user\_id from question  
 join answer a on question.question\_id = a.answer\_question\_id  
 where answer\_is\_valid={is\_valid}  
 group by question\_id;'''  
  
 cursor.execute(command)  
 *return* cursor.fetchall(), TABLES['question']  
  
  
*def* search\_users\_by\_question\_date(dates):  
 start, end = dates  
 command = f"""  
 select "user".user\_id, user\_name, user\_age from "user"  
 join question q on "user".user\_id = q.question\_user\_id  
 where question\_datetime between '{start}' and '{end}'  
 group by user\_id;"""  
  
 cursor.execute(command)  
 *return* cursor.fetchall(), TABLES['user']  
  
  
*def* full\_text\_search\_without\_word(word):  
 command = f"""  
 select question\_text, answer\_text, answer\_is\_valid from question  
 join answer a on question.question\_id = a.answer\_question\_id  
 where to\_tsvector(question\_text) || to\_tsvector(answer\_text) @@ to\_tsquery('!{word}');"""  
 cursor.execute(command)  
 *return* cursor.fetchall(), ('question\_text', 'answer\_text')  
  
  
*def* full\_text\_search\_by\_phrase(phrase):  
 command = f"""  
 select ts\_headline(question\_text, phraseto\_tsquery('{phrase}'), 'StartSel=\033[91m, StopSel=\033[0m'),  
 ts\_headline(answer\_text, phraseto\_tsquery('{phrase}'), 'StartSel=\033[91m, StopSel=\033[0m')  
 from question join answer a on question.question\_id = a.answer\_question\_id  
 where to\_tsvector(question\_text) || to\_tsvector(answer\_text) @@ phraseto\_tsquery('{phrase}');"""  
 cursor.execute(command)  
 *return* cursor.fetchall(), ('question\_text', 'answer\_text')  
  
  
*def* create\_random\_questions():  
 *try*:  
 *with open*('sql/random.sql', 'r') *as* file:  
 sql = file.read()  
 cursor.execute(sql)  
 *finally*:  
 conn.commit()

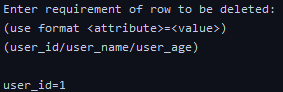
**Скріншот результатів виконання операції вилучення**

Спроба вилучення Користувача на який є посилання





Видаляємо користувача на якого є посилання:





Вдале видалення і демонстрація результату вилучення

