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: «**Ознайомлення з базовими операціями СУБД PostgreSQL**»

Виконав: студент III курсу

ФПМ групи КВ-82

Іваненко О. А.

Перевірив: Павловський В. І.

**Київ – 2020**

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

*Порядок виконання роботи*:

1. Зазвичай перехід від концептуальної моделі предметного середовища до реляційної моделі даних дає третю нормальну форму (3НФ) її відношень. Але іноді, в першу чергу при малому досвіді проектувальника, це не так. Тому треба перевірити отримані в лабораторній роботі №1 відношення на відповідність до 3НФ і, якщо потрібно, провести додаткову нормалізацію даних;
2. Встановити систему програмування Python 3.7.х;
3. Встановити середовище розробки програмного забезпечення – PyCharm Community Edition 2019;
4. Налаштувати систему програмування Python для роботи з БД через власну консоль або систему програмування IDLE. Виконати команду (модуль) pip (Python Package Installer) в командному рядку Windows;
5. Налаштувати середовище розробки PyCharm для роботи з БД;
6. Реалізувати внесення, редагування та вилучення даних у БД, розробивши програму з консольним інтерфейсом, використавши IDE PyCharm, що містить меню для вибору таблиці бази даних і, далі, операції над нею: додати, вилучити, редагувати, переглянути, виконати пошук:
7. імпортувати пакет psycopg2;
8. з’єднатися з БД;
9. отримати курсор;
10. створити запит;
11. отримати та обробити результати запиту;
12. Передбачити автоматичне пакетне генерування "рандомізованих" даних у БД, а саме внесення 10000 рядків у одну із таблиць. Дані мають бути згенерованими не програмою, а відповідним SQL-запитом;
13. Реалізувати статичні запити (див. методвказівки Python + PostgreSQL) на пошук за 2-3 атрибутами з 2–х чи більше сутностей одночасно для:
14. числових атрибутів – у рамках діапазону,
15. рядкових – як перелічення,
16. логічного типу – значення True/False, для дат – у рамках діапазону дат.

Якщо даних якогось з перелічених типів в БД нема, підібрати розширення складу атрибутів відповідних таблиць;

1. Реалізувати динамічні (параметричні) запити (див. методвказівки Python + PostgreSQL) на пошук даних відповідно до умов пункту 8;
2. Дослідити всі режими обмеження ON DELETE при описі зв’язку між батьківською та дочірньої таблицями за результатами вилучення запису батьківської таблиці та вмісту дочірньої таблиці після цього вилучення;
3. Передбачити в програмі перехоплення помилки – повідомлення про неможливість видалення запису батьківської таблиці за наявності залежних даних в дочірній таблиці, з виведенням повідомлення про неможливість вилучення;
4. Передбачити в програмі перехоплення помилки – повідомлення про неможливість вставки запису в дочірню таблицю за відсутності в батьківській таблиці відповідного запису, з виведенням повідомлення про неможливість вставки

*Зміст звіту*:

1. Варіант завдання;
2. Нормалізована модель даних;
3. Лістинги програм з директивами внесення, редагування та вилучення даних у базі даних та результати виконання цих директив;
4. Лістинги програм зі статичними та динамічними запитами пошуку;
5. Лістинг модуля «модель» згідно із шаблоном MVC;
6. Скріншоти результатів виконання операції вилучення запису батьківської таблиці та виведення вмісту дочірньої таблиці після цього вилучення, а якщо воно неможливе, то результат перехоплення помилки з виведенням повідомлення про неможливість такого видалення за наявності залежних даних.

**Нормалізована модель даннх БД «Соціальна мережа»**

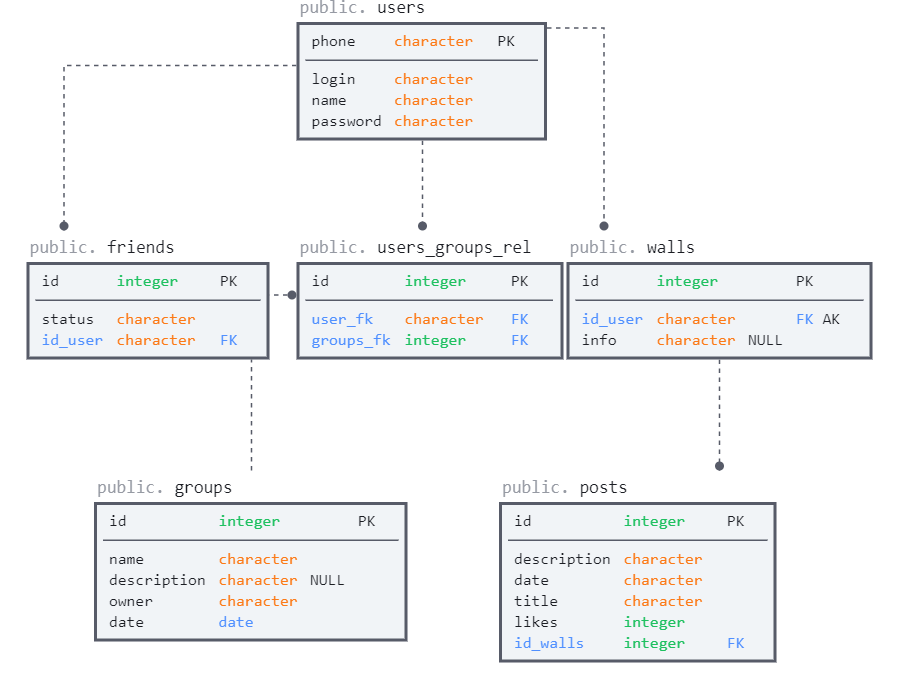


Рисунок 1 - Логічна модель (Структура) БД “ Соціальна мережа ” (засобами SqlDMB)

Усі таблиці (відношення) знаходяться в 3 НФ тому, що у кожній із них:

1. Всі атрибути є атомарними та відсутні повторення рядків (1НФ);
2. Первинний ключ складається лише з одного атрибуту (2НФ);
3. Кожний не первинний атрибут не є транзитивно залежним від первинного ключа (3НФ).

**Опис програми**

Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL та реалізовує функціональні вимоги, що наведені у завданні. Доступ до БД отримується за допомогою python бібліотеки psycopg2. Програма складається з 4 основних модулів, та допоміжного файла config,ini, який забезпечує підключення до БД через localhost.

Опис модулів:

1. View.py – містить універсальні методи для виводу даних на екран.
2. Model.py – містить клас Database, з усіма методами для обробки БД, та управління програмою.
3. Config.py – містить клас Config з методами для підключення (відключення) бази даних до (від) програми.
4. Controller.py – містить клас Controller з методами для контролю введених/виведених данних через програмне меню.

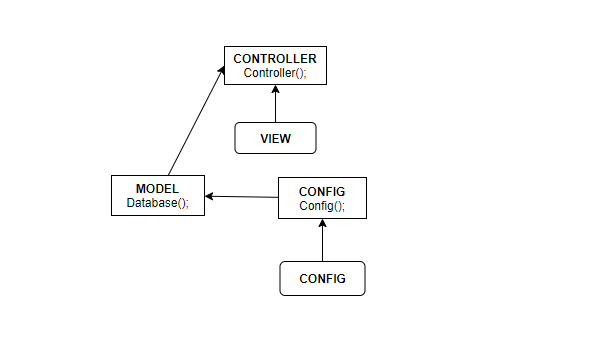


Рисунок 2 – Концептуальна модель програми

**Посилання для навігації по тексту програми**

[**1. Лістінг файлу config.ini**](#configini)

**[2. Лістінг модуля Config.py](#configpy)**

[**3. Лістінг модуля View.py**](#viewpy)

[**4. Лістінг модуля Controller.py**](#controllerpy)

[**4.1. Запуск програми**](#запуск)

[**5. Лістінг модуля Model.py**](#modelpy)

[**5.1. Прості операції с БД**](#простыеоперации)

[**5.1.1 Додавання до БД**](#додавання)

[**5.1.2 Редагування в БД**](#редагуваня)

[**5.1.3 Видалення з БД**](#видалення)

[**5.2. Рандомізація даних в БД**](#рандомизация)

[**5.3. Реалізація пошуку за декількома атрибутами в БД**](#пошук)

[**6. Дослідження ON\_DELETE**](#ondelete)

**Лістінг файлу config.ini**

*[postgresql]  
host*=localhost  
*port*=5432  
*database*=social\_network  
*user*=postgres  
*password*=6969

**Лістінг модуля Config.py**

*from* configparser *import* ConfigParser  
  
*class* Config:  
 *def \_\_init\_\_*(*self*, config\_file):  
 parser = ConfigParser()  
 parser.read(config\_file)  
 db = {}  
 section = 'postgresql'  
 *if* parser.has\_section(section):  
 params = parser.items(section)  
 *for* param *in* params:  
 *if* param[0] == 'host':  
 *self*.host = param[1]  
 *elif* param[0] == 'port':  
 *self*.port = param[1]  
 *elif* param[0] == 'database':  
 *self*.database = param[1]  
 *elif* param[0] == 'user':  
 *self*.user = param[1]  
 *elif* param[0] == 'password':  
 *self*.password = param[1]  
 *else*:  
 *raise Exception*('Wrong key {0} in section {1}'.format(param[0], section))  
 *else*:  
 *raise Exception*('Section {0} not found in the {1} file'.format(section, config\_file))  
  
 *def* to\_dict(*self*):  
 *return* {  
 'host': *self*.host,  
 'port': *self*.port,  
 'database': *self*.database,  
 'user': *self*.user,  
 'password': *self*.password  
 }  
  
 *def \_\_str\_\_*(*self*):  
 *return* "[Config: {}, {}, {}, {}, {}]".format(*self*.host, *self*.port, *self*.database, *self*.user, *self*.password)

**Лістінг модуля View.py**

*def* printer(res):  
 *if type*(res)==*tuple*:  
 max\_width = []  
 limit\_len = 40  
 *for* index, elem *in enumerate*(res[1]):  
 max\_width.append(*max*([*len*(*str*(x[index])) *for* x *in* res[0]]+[*len*(elem)]))  
 *if* max\_width[index]>limit\_len:  
 max\_width[index] = limit\_len  
 *print*(elem.rjust(max\_width[index])+ " | ", end='')  
 *print*('')  
 *for* item *in* res[0]:  
 *print*(" | ".join(*map*(*lambda* x: *str*(x[1])[:*min*(max\_width[x[0]],*len*(*str*(x[1])))].rjust(max\_width[x[0]]) , *enumerate*(item)))+" | ")  
 *elif* res == *True*:  
 *print*("Without feedback")  
 *elif* res == *False*:  
 *print*("Error")

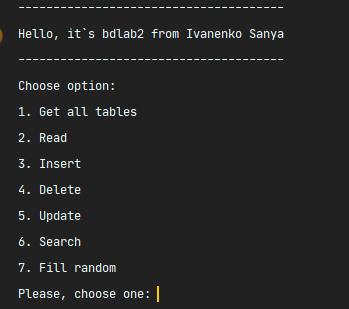
**Лістінг модуля Controller.py**

*from* model *import* \*  
*from* view *import* \*  
*import* sys  
*import* sys *# sys нужен для передачи argv в QApplication  
  
# Import the necessary packages*all\_tables\_list = 1  
tables\_dict = {"users": ["login", "name", "password", "phone"],  
 "groups": ["id", "name", "description", "owner", "date"],  
 "friends": ["status", "id", "id\_user"],  
 "walls": ["id", "id\_user", "info"],  
 "posts": ["description", "date", "title", "likes", "id", "id\_walls"]}  
tables = ["users", "groups", "friends", "walls", "posts"]  
columns = [["login", "name", "password", "phone"],  
 ["id", "name", "description", "owner", "date"],  
 ["status", "id", "id\_user"],  
 ["id", "id\_user", "info"],  
 ["description", "date", "title", "likes", "id", "id\_walls"]]  
read\_number = 2  
insert\_number = 3  
delete\_number = 4  
update\_number = 5  
search\_number = 6  
random\_fill = 7  
  
  
*class* Controller():  
 *def \_\_init\_\_*(*self*):  
 *self*.db = Database()  
  
c = Controller()  
  
*# printer(c.db.read("friends",["id"]))#read test  
# printer(c.db.insert("friends",["status","id\_user"], ["best","+380679201293"]))#insert test ничего не вернёт, поэтому нужно смотреть таблицу  
# printer(c.db.delete("friends","id\_user = '+380679201293'"))#delete test норм удалило  
def* print\_footer(len):  
 *print*("-" \* len)  
  
  
*def* menu():  
 print\_footer(38)  
 *print*("Hello, it`s bdlab2 from Ivanenko Sanya")  
 print\_footer(38)  
 *print*("Choose option:")  
 *print*("1. Get all tables")  
 *print* ("2. Read")  
 *print*("3. Insert")  
 *print*("4. Delete")  
 *print*("5. Update")  
 *print*("6. Search")  
 *print*("7. Fill random")  
 chosen = *input*("Please, choose one: ")  
 *try*:  
 chosen = *int*(chosen)  
 *except Exception as* error:  
 *print*(error)  
 *return None  
 return* chosen  
  
  
*while* (*True*):  
 menu\_number = menu()  
 *if* menu\_number == all\_tables\_list:  
 *for* index, elem *in enumerate*(tables):  
 *print*("Table:" + elem + "\n With columns: " + ",".join(columns[index]))  
 *elif* menu\_number == random\_fill:  
 c.db.fill\_random()  
 *elif* menu\_number == read\_number:  
 chosen\_table = *input*("Please, enter table from this list: " + "(" + ",".join(tables) + ") >> ")  
 *if* chosen\_table *in* tables:  
 chosen\_column = *input*("Please, enter columns from this list (splitted by comma): " + "(" + ",".join(  
 tables\_dict[chosen\_table]) + ") >> ")  
 chosen\_column = chosen\_column.split(',')  
 printer(c.db.read(chosen\_table, chosen\_column))  
 *elif* menu\_number == insert\_number:  
 chosen\_table = *input*(("Please, enter table from this list: " + "(" + ",".join(tables) + ") >> "))  
 *if* chosen\_table *in* tables:  
 columns\_list = *input*("Please, enter columns from this list (splitted by comma): " + "(" + ",".join(  
 tables\_dict[chosen\_table]) + ") >> ")  
 columns\_list = columns\_list.split(",")  
 values = *input*("Please, enter values for each column(splitted by comma: >>")  
 values = values.split(",")  
 printer(c.db.insert(chosen\_table, columns\_list, values))  
 *elif* menu\_number == delete\_number:  
 chosen\_table = *input*(("Please, enter table from this list: " + "(" + ",".join(tables) + ") >> "))  
 *if* chosen\_table *in* tables:  
 condition = *input*("Please, enter condition for items to be deleted):" + " >> ")  
 printer(c.db.delete(chosen\_table, condition))  
  
 *elif* menu\_number == update\_number:  
 chosen\_table = *input*(("Please, enter table from this list: " + "(" + ",".join(tables) + ") >> "))  
 *if* chosen\_table *in* tables:  
 columns\_list = *input*("Please, enter columns from this list (splitted by comma): " + "(" + ",".join(  
 tables\_dict[chosen\_table]) + ") >> ")  
 columns\_list = columns\_list.split(",")  
 values = *input*("Please, enter values for each column(splitted by comma): >>")  
 values = values.split(",")  
 condition = *input*("Please, enter condition for items to be deleted):" + " >> ")  
 printer(c.db.update(chosen\_table, columns\_list, values, condition))  
 *else*:  
 *print*(*str*(chosen\_table) + "is not in tables list")  
 *elif* menu\_number == search\_number:  
 *print*("1. Пошук постів та по номеру телефону автора, опису поста та даті публікації\n"  
 "2. Пошук людей по групах, на які вони підписані, по телефону та по даті оформлення підписки \n"  
 "3. Пошук кількості вподобань на пості за тематикою, айді автора та ліміту кількості лайків \n")  
 chosen\_num = *int*(*input*("Enter num: >> "))  
 *if* chosen\_num == 1:  
 attributes = ["+380%","< '2018-31-12'","another %"]  
 *# attributes[0] = input("Phone like:")  
 # attributes[1] = input("Date cond:")  
 # attributes[2] = input("Descriprion cond:")* printer(c.db.search(1,attributes))  
 *if* chosen\_num == 2:  
 attributes = ["about japan","between '2000-01-01' and '2020-12-12'","like '+380%'"]  
 printer(c.db.search(2,attributes))  
 *if* chosen\_num == 3:  
 attributes = *None* printer(c.db.search(3,attributes))

**Лістінг модуля Model.py**

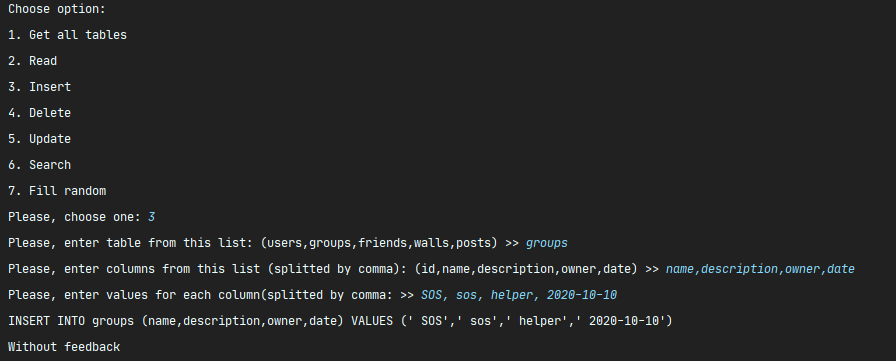
*import* psycopg2 *as* PS  
*from* configparser *import* ConfigParser  
  
  
*class* Database:  
 *def* request(*self*, req, fetch\_results=*False*):  
 *pass  
  
 def* config(*self*, filename='config.ini', section='postgresql'):  
 parser = ConfigParser()  
 parser.read(filename)  
 db = {}  
 *if* parser.has\_section(section):  
 params = parser.items(section)  
 *for* param *in* params:  
 db[param[0]] = param[1]  
 *else*:  
 *raise Exception*('Section {0} not found in the {1} file'.format(section, filename))  
 *return* db  
  
 *def* get\_request(*self*, req, get\_results=*False*):  
 *try*:  
 cursor = *self*.conn.cursor()  
 cursor.execute(req)  
 *self*.conn.commit()  
 *if* cursor.description *is not None*:  
 *self*.colnames = [desc[0] *for* desc *in* cursor.description]  
 *if* get\_results:  
 *return* cursor.fetchall(), *self*.colnames  
 *else*:  
 *self*.conn.commit()  
 *return True  
 except*(*Exception*, ps.DatabaseError, ps.ProgrammingError) *as* error:  
 *self*.conn.rollback()  
 *self*.gen\_error = error  
 *self*.erFlag = *True  
 print*(error)  
 *return False  
  
 def \_\_init\_\_*(*self*):  
 *self*.conn = *None  
 self*.error = ''  
 *self*.gen\_error = ''  
 *self*.erFlag = *False  
 self*.Gen = *True  
 self*.colnames = *list*()  
 *try*:  
 params = *self*.config('config.ini')  
 *self*.conn = ps.connect(\*\*params)  
 *print*(*self*.conn)  
  
 *except*(*Exception*, ps.DatabaseError) *as* error:  
 *print*(error)  
  
 *def* test(*self*):  
 req = "SELECT *\** FROM book;"  
 *return self*.get\_request(req, *True*)  
  
 *def* read(*self*, table, columns):  
 req = "SELECT %s FROM %s" % (",".join(columns) *if* (columns *is not None and len*(columns) != 0) *else* "\*", table)  
 *return self*.get\_request(req, *True*)  
  
 *def* delete(*self*, table, cond):  
 req = "DELETE FROM %s WHERE %s" % (table, cond)  
 *return self*.get\_request(req)  
  
 *def* search(*self*, mode, attributes):  
 req = ""  
  
 *if* mode == 1:  
 select = "SELECT p.title,p.description,u.name FROM posts p "  
 join = "JOIN walls w ON w.id=p.id\_walls JOIN Users u ON w.id\_user=u.phone "  
 cond = "WHERE w.id\_user LIKE '{}' AND p.date {} AND p.description LIKE '{}'".format(attributes[0],  
 attributes[1],  
 attributes[2])  
 req = select + join + cond  
 *elif* mode == 2:  
 select = "SELECT u.name,g.name FROM users u JOIN users\_groups\_rel rel ON u.phone=rel.user\_fk "  
 join = "JOIN groups g ON g.id=rel.groups\_fk where g.name='anime group' and g.date = '2020-01-02' and u.phone like '+380%' GROUP BY g.name, u.name "  
 *# cond = "WHERE g.name='{0}' and date {1} and u.phone {2}".format(attributes[0],  
 # attributes[1],  
 # attributes[2])  
 #cond = "where g.name='about japan' and date > '2000-01-01' and '2020-12-12' and u.phone like '+380%'"* req = select + join  
 *print*(req)  
 *elif* mode == 3:  
 req = "select p.likes from Posts p join Walls w on w.id=p.id\_walls where title='birthday!' and id\_user='+380679201293' and likes>0"  
 *return self*.get\_request(req, *True*)  
  
 *def* insert(*self*, table, columns\_data, values\_data, count=1):  
 columns = ",".join(columns\_data)  
 raw\_values = ",".join([x.lstrip("!") *if* x.startswith("!") *else* "'{}'".format(x) *for* x *in* values\_data])  
 values = ",".join(["({0})".format(raw\_values) *for* x *in range*(0, count)])  
 req = "INSERT INTO %s (%s) VALUES %s" % (table, columns, values)  
 *print*(req)  
 *return self*.get\_request(req)  
  
 *def* update(*self*, table, column\_data, value\_data, cond):  
 update\_query = ",".join(  
 ["{} = {}".format(element[0],  
 element[1].lstrip("!") *if* element[1].startswith("!")  
 *else* "'{}'".format(element[1]))  
 *for* element *in zip*(column\_data, value\_data)])  
 req = "UPDATE {0} SET {1} WHERE {2}".format(table, update\_query, cond)  
 *return self*.get\_request(req)  
  
 *def* by\_key\_value(*self*, key, value):  
 *return* "{0}='{1}'".format(key, value)  
  
 *def* len\_eq(*self*, key, len):  
 *return* "length({0}) = {1}".format(key, len)  
  
 *def* gen\_int\_req(*self*, min, range):  
 *return* "! trunc({0}+random()\*{1})::int".format(min, range)  
  
 *def* gen\_date(*self*):  
 *return* "! timestamp '2004-01-10' + random() \* (timestamp '2004-01-20' - timestamp '2004-01-10')"  
  
 *def* gen\_date\_string(*self*):  
 *return* "! {0} || '-' || {1} || '-' || {2}".format(*self*.gen\_string\_req(4)[1:],  
 *self*.gen\_string\_req(2)[1:],  
 *self*.gen\_string\_req(2)[1:])  
  
 *def* gen\_string\_req(*self*, len, min\_char=65, char\_range=25):  
 req = ["chr(trunc({0}+random()\*{1})::int)".format(min\_char, char\_range)]  
 *return* "!" + "||".join(req \* len)  
  
 *def* fill\_random(*self*):  
 *return self*.insert(  
 "groups",  
 ["name", "description", "owner", "date"],  
 [*self*.gen\_string\_req(10),  
 *self*.gen\_string\_req(5),  
 *self*.gen\_string\_req(10),  
 *self*.gen\_date()],  
 *int*(*input*("Input count of random users to insert: ")))

**Запуск програми**

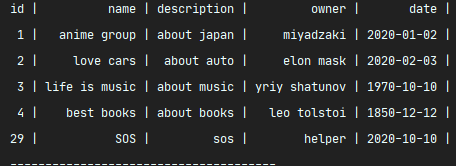
****

**Прості операції з БД**

1.Додавання даних до БД(*def* insert(*self*, table, columns\_data, values\_data, count=1):)



Результат додавання



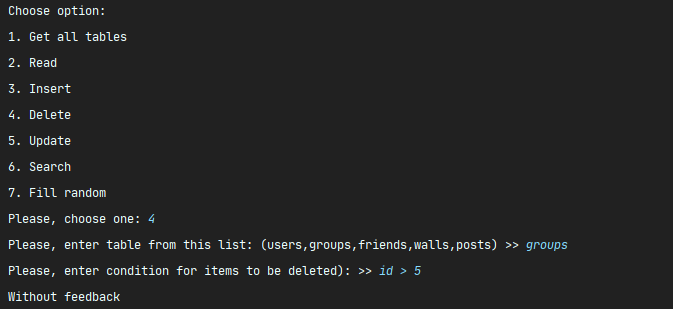
2. Редагування данних (*def* update(*self*, table, column\_data, value\_data, cond))



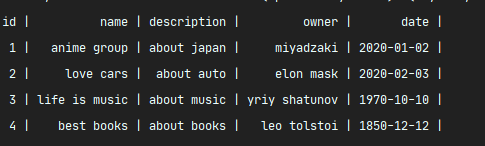
Результат редагування



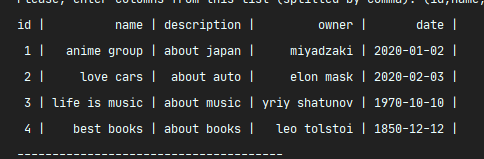
3. Видалення даних (*def* delete(*self*, table, cond):)

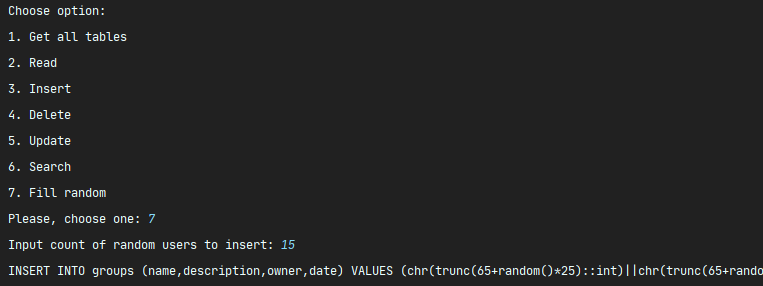


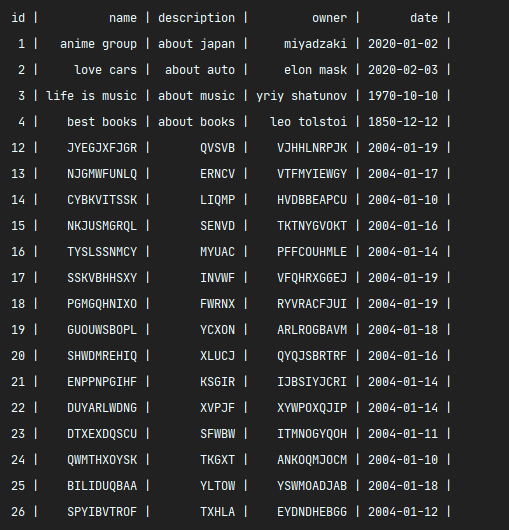
Результат видалення

****

**Додавання рандомізованих даних до таблиці** groups

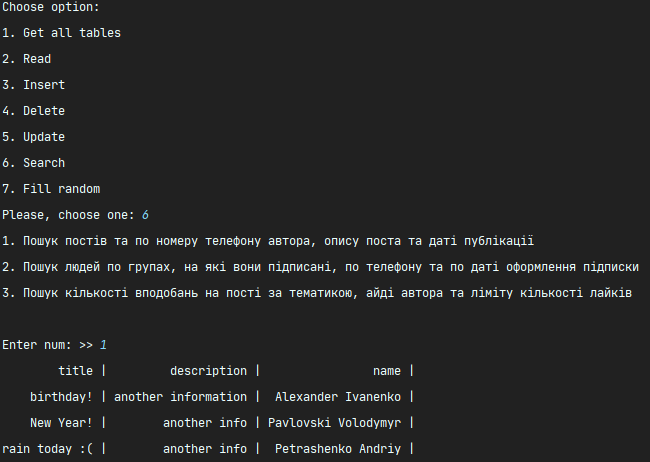


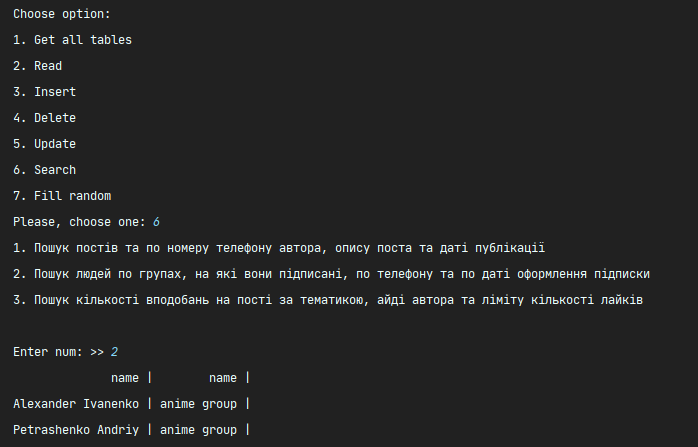


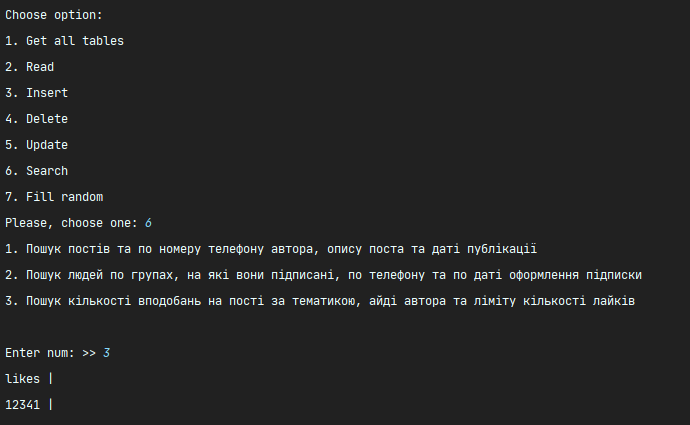


**Пошук в БД за декількома атрибутами**

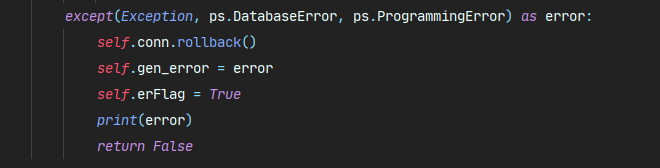
Пошук за двома-трьома атрибутами з декількох сутностей одночасно за варіантом (search(*self*, mode, attributes):):





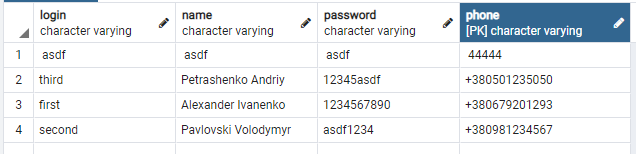


**Обробка помилок при введенні/вилученні та валідації даних**

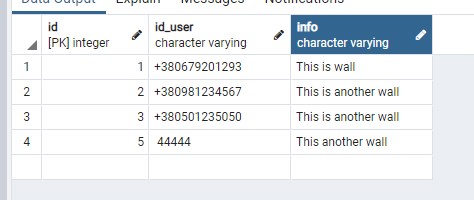
****

**Дослідження ON DELETE**

Таблиця батьківська (users)

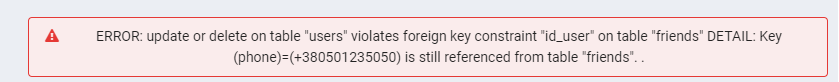


Таблиця дочірня (walls)



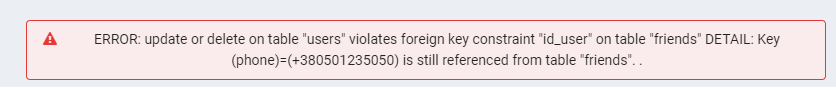
***Дослідження NO ACTION***

При видаленні запису з таблиці users, phone якого присутній в записі walls, отримуємо повідомлення про помилку.



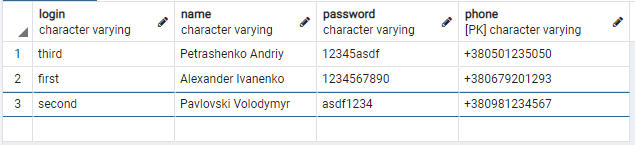
***Дослідження RESTRICT***

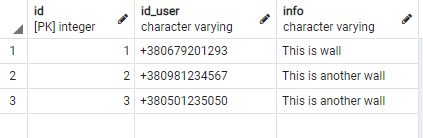
При видаленні запису з таблиці отримуємо таке ж повідомлення про помилку, як і в режимі NO ACTION.



***Дослідження CASCADE***

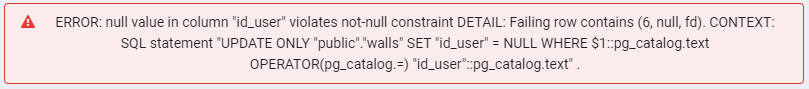
В режимі CASCADE при видаленні запису з батьківської видаляється відповідний запис у дочірній таблиці також:





***Дослідження SET NULL***

В режимі SET NULL, якщо поля не мають обмеження NOT NULL, запис видалиться, і в дочірній таблиці встановиться NULL. У нашому випадку є обмеження NOT NULL, тому є помилка.



***Дослідження SET DEFAULT***

В налаштуваннях id\_user поле DEFAULT не вставновлено. Оскільки SET DEFAULT намагається його встановити як значення phone. Якщо встановити DEFAULT в будь-яке значення, то при видаленні двох + записів, відповідні записи таблиці users та дочірньої таблиці будуть мати однакове значення (DEFAULT), що порушує реляційність нашої БД.

