

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**

*на тему: “Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL”*

Виконав:

студент ІII курсу

групи КВ-02

Недоколенко Д. Р.

Перевірив:

Павловський В. І.

Київ – 2022-2023

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

*Загальне завдання роботи* полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, 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).

**Логічна модель (схема) БД «Бібліотека»**

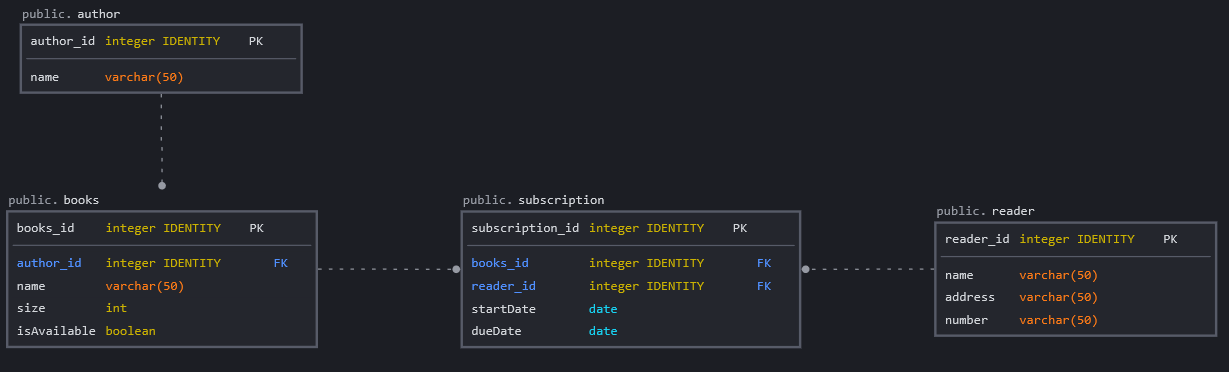
Логічну модель (схему бази даних) наведено на Рисунку 1.

Рисунок 1. – Схема бази даних (інструмент: sqldb.com)

Є незначні зміни в назвах: в isAvailable, startDate та dueDate була прибрана капіталізація.

**Середовище розробки та налаштування підключення до бази даних**

Для виконання лабораторної роботи використовувались мова програмування Python 3.9. та середовище розробки PyCharm Community Edition 2022.2.1.

**Шаблон проектування програмного додатку**

MVC – шаблон проектування, який використовується у програмі.

Model – представляє клас, що описує логіку використовуваних даних. Реалізований у файлі Model.py.

View – відповідає за ввід/вивід інформації. Реалізований у файлі View.py.

Controller – представляє клас, що обробляє ввід користувача, подає відповідну команду до Model.py та виводить результат за допомогою View.py. Реалізований у файлі Controller.py.

**Структура програми та її опис**

Структуру програми наведено на Рисунку 2.

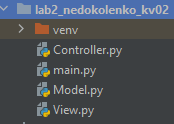


Рисунок 2 – Структура програми

1. main.py – точка входу в програму. Викликає функцію підключення до БД та запускає інтерфейс меню;
2. View.py – файл для відображення меню, виклику повідомлень про успішне виконання дій та виводу елементів на екран;
3. Controller.py – оброблює ввід користувача, подає відповідні команди до Model.py, очікує та виводить результат;
4. Model.py – реалізовує усю логіку програми: підключення до БД та виконання запитів до неї.

**Структура меню програми**

Початкову структуру меню програми наведено на Рисунку 3. Вона складається с 7 пунктів.

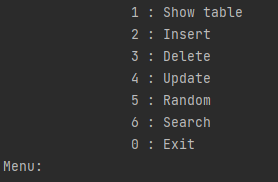


Рисунок 3 – Меню програми

Перший пункт Show table виконує перегляд обраної таблиці БД та виводить її дані. Користувач потрібен обрати номер таблиці.

Другий пункт Insert виконує внесення даних в обрану таблицю БД. Користувач потрібен обрати номер таблиці, колонки, які потрібно заповнити, а також їх значення.

Третій пункт Delete виконує вилучення даних з обраної таблиці БД. Користувач потрібен обрати номер таблиці, а також номер первинного ключа.

Четвертий пункт Update виконує редагування даних в обраній таблиці БД. Користувач потрібен обрати номер таблиці, колонку, які потрібно змінити та нове значення для неї.

П’ятий пункт Random виконує автоматичну генерацію «рандомізованих» значень в обрану таблицю. Користувач потрібен обрати номер таблиці та кількість значень, які треба додати.

Шостий пункт Search виконує один трьох з обраних користувачем запитів пошуку:

1. За діапазоном дат (початок та закінчення абонементу) та рядковим значенням (ім’я читача) в таблицях «reader» та «subscription»;
2. За логічним True або False (наявність книги), числовим (розмір книги) та рядковим (ім’я автора) значеннями в таблицях «author» та «books»;
3. За рядковим значенням (назвою книги), діапазоном дат (дата закінчення абонементу) та числовим значенням (розмір книги) в таблицях «books» та «subscription».

Останній пункт Exit виконує зупинення роботи програми та відключення від БД.

**Лістинг модуля «Model» згідно із шаблоном MVC**

**import** psycopg2

**from** psycopg2 **import** sql

**import** time

**class** Model\_db:

**def** \_\_init\_\_(self, username, password, host, port, dbname):

self.\_\_context = psycopg2.connect(user=username, password=password,

host=host, port=port, database=dbname)

self.\_\_cursor = self.\_\_context.cursor()

self.\_\_table\_names = None

**def** \_\_del\_\_(self):

self.\_\_cursor.close()

self.\_\_context.close()

**def** clear\_transaction(self):

self.\_\_context.rollback()

*# Отримання імен таблиць БД*

**def** get\_table\_names(self):

**if** self.\_\_table\_names **is** None:

self.\_\_cursor.execute(("""SELECT table\_name

FROM information\_schema.tables

WHERE table\_schema = 'public'"""))

self.\_\_table\_names = [table[0] **for** table **in** self.\_\_cursor]

**return** self.\_\_table\_names

*# Отримання імен та типів стовпчиків таблиці*

**def** get\_column\_types(self, table\_name):

self.\_\_cursor.execute("""SELECT column\_name, data\_type

FROM information\_schema.columns

WHERE table\_schema = 'public' AND table\_name = %s

ORDER BY table\_schema, table\_name""", (table\_name,))

**return** self.\_\_cursor.fetchall()

*# Отримання імен стовпчиків таблиці*

**def** get\_column\_names(self, table\_name):

self.\_\_cursor.execute("""

SELECT column\_name FROM information\_schema.columns

WHERE table\_schema = 'public' AND table\_name = %s

ORDER BY table\_schema, table\_name""", (table\_name,))

**return** [x[0] **for** x **in** self.\_\_cursor.fetchall()]

*# Отримання зовнішніх ключів таблиці*

**def** get\_foreign\_key\_info(self, table\_name):

self.\_\_cursor.execute("""

SELECT kcu.column\_name, ccu.table\_name AS

foreign\_table\_name,

ccu.column\_name AS foreign\_column\_name

FROM information\_schema.table\_constraints AS tc

JOIN information\_schema.key\_column\_usage AS kcu

ON tc.constraint\_name = kcu.constraint\_name

AND tc.table\_schema = kcu.table\_schema

JOIN information\_schema.constraint\_column\_usage AS ccu

ON ccu.constraint\_name = tc.constraint\_name

AND ccu.table\_schema = tc.table\_schema

WHERE tc.constraint\_type = 'FOREIGN KEY' AND

tc.table\_name=%s;""", (table\_name,))

**return** self.\_\_cursor.fetchall()

*# Отримання даних таблиці БД*

**def** get\_table\_data(self, table\_name):

*# Отримання Id таблиці*

id\_column = self.get\_column\_types(table\_name)[0][0]

cursor = self.\_\_cursor

**try**:

*# Отримання даних таблиці по ключу*

cursor.execute(

sql.SQL('SELECT \* FROM {} ORDER BY {} ASC'

).format(sql.Identifier(table\_name), sql.SQL(id\_column)))

**except** Exception **as** e:

**return** str(e)

**return** [col.name **for** col **in** cursor.description], cursor.fetchall()

*# Вставка даних в таблицю*

**def** insert\_data(self, table\_name, values):

*# Формування рядка значень VALUES запиту INSERT*

line = ''

columns = '('

**for** key **in** values:

**if** values[key]:

line += '%(' + key + ')s,'

columns += key + ','

columns = columns[:-1] + ')'

*# Виконання SQL-запиту на вставку даних в таблицю*

self.\_\_cursor.execute(

sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')')

.format(sql.Identifier(table\_name), sql.SQL(columns)),

values)

self.\_\_context.commit()

*# Видалення даних в таблиці*

**def** delete\_data(self, table\_name, value, cond):

self.\_\_cursor.execute(

sql.SQL('DELETE FROM {} WHERE {} = {}').format(sql.Identifier(table\_name),

sql.Identifier(value),

sql.SQL(cond)))

self.\_\_context.commit()

**def** delete\_with\_fk(self, table\_name1, table\_name2, value1, value2, value3, cond):

self.\_\_cursor.execute(

sql.SQL('DELETE FROM {} WHERE {} in (SELECT {} FROM {} WHERE {} = {})').

format(sql.Identifier(table\_name1),

sql.Identifier(value1),

sql.Identifier(value2),

sql.Identifier(table\_name2),

sql.Identifier(value3),

sql.SQL(cond)))

self.\_\_context.commit()

*# Оновлення даних в таблиці*

**def** change\_data(self, table\_name, values):

*# Формування рядка значень SET запиту UPDATE*

line = ''

condition = values.pop('condition')

**for** key **in** values:

**if** values[key]:

line += key + '=%(' + key + ')s,'

*# Виконання SQL-запиту на оновлення даних в таблиці*

self.\_\_cursor.execute(

sql.SQL('UPDATE {} SET ' + line[:-1] + ' WHERE {}')

.format(sql.Identifier(table\_name), sql.SQL(condition)),

values)

self.\_\_context.commit()

*# Генерація випадкових даних і заповнення таблиці*

**def** generate\_data(self, table\_name, count):

types = self.get\_column\_types(table\_name)

fk\_array = self.get\_foreign\_key\_info(table\_name)

select\_subquery = ""

insert\_query = "INSERT INTO " + table\_name + " ("

**for** i **in** range(1, len(types)):

t = types[i]

name = t[0]

types = t[1]

fk = [x **for** x **in** fk\_array **if** x[0] == name]

**if** fk:

select\_subquery += ('(SELECT {} FROM {} ORDER BY RANDOM(), ser LIMIT 1)'.format(fk[0][2], fk[0][1]))

**elif** types == 'integer':

select\_subquery += 'trunc(random()\*100)::INT'

**elif** types == 'character varying':

select\_subquery += 'chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + random()\*25)::INT)'

**elif** types == 'date':

select\_subquery += """ date(timestamp '2022-01-01' +

random() \*

(timestamp '2024-01-01' - timestamp '2022-01-01'))"""

**elif** types == 'boolean':

select\_subquery += 'random()::INT::BOOLEAN'

insert\_query += name

**if** i != len(types) - 1:

select\_subquery += ','

insert\_query += ','

**else**:

insert\_query += ') '

self.\_\_cursor.execute(

insert\_query + "SELECT " + select\_subquery +

" FROM generate\_series(1," + str(count) + ") as ser")

self.\_\_context.commit()

**def** join\_general(self, main\_query, condition=""):

new\_cond = condition

**if** condition:

new\_cond = "WHERE " + condition

t1 = time.time()

self.\_\_cursor.execute(main\_query.format(new\_cond))

t2 = time.time()

**print**((t2 - t1) \* 1000)

**return** self.\_\_cursor.fetchall()

*# reader subscription*

**def** join\_reader\_subscription(self, condition=""):

**return** self.join\_general("""SELECT \* FROM reader as r

JOIN subscription as s

ON r.reader\_id = s.reader\_id {}

ORDER BY r.reader\_id ASC""", condition)

*# author books*

**def** join\_author\_books(self, condition=""):

**return** self.join\_general("""SELECT \* FROM author as a

JOIN books as b

ON a.author\_id = b.author\_id {}

ORDER BY b.books\_id DESC""", condition)

*# subscription books*

**def** join\_subscription\_books(self, condition=""):

**return** self.join\_general("""SELECT \* FROM subscription as s

JOIN books as b

ON s.books\_id = b.books\_id {}

ORDER BY b.books\_id ASC""", condition)

Як вже було зазначено, клас «Model» виконує усю логіку програми: підключення до бази даних; отримання, внесення, видалення та редагування даних таблиць; генерування випадкових даних і заповнення ними таблиці; 3 різні запити пошуку за декількома атрибутами серед двох сутностей одночасно. Усі наведені функції буде викликати клас «Controller» при відповідному запиті користувача.

**Лістинги фрагментів програм внесення, редагування та вилучення даних у базі даних**

1. Внесення даних в таблицю:

Model.py:

**def** insert\_data(self, table\_name, values):

*# Формування рядка значень VALUES запиту INSERT*

line = ''

columns = '('

**for** key **in** values:

**if** values[key]:

line += '%(' + key + ')s,'

columns += key + ','

columns = columns[:-1] + ')'

*# Виконання SQL-запиту на вставку даних в таблицю*

self.\_\_cursor.execute(

sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')')

.format(sql.Identifier(table\_name), sql.SQL(columns)),

values)

self.\_\_context.commit()

Controller.py:

**def** insert(self, table\_name, values):

**try**:

**print**('===========')

self.model.insert\_data(table\_name, values)

self.view.display\_insert(table\_name, values)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося додати дані в таблицю", error)

Спочатку в циклі формується рядок значень для запиту INSERT. Потім за допомогою команди execute виконується SQL-запит на вставку даних в таблицю. Для роботи функції потрібно знати назву таблиці, а також рядок з колонками та їх значенням. Оскільки в моїх таблицях усі значення колонок NOT NULL, треба дати значення усім колонкам. Перехоплення такої помилки передбачено. Для простого обрання усіх колонок, в меню для значення values є можливість додати рядок «all», який заміниться на назви колонок обраної таблиці.

1. Видалення даних в таблицю:

Model.py:

**def** delete\_data(self, table\_name, value, cond):

self.\_\_cursor.execute(

sql.SQL('DELETE FROM {} WHERE {} = {}').format(sql.Identifier(table\_name),

sql.Identifier(value),

sql.SQL(cond)))

self.\_\_context.commit()

**def** delete\_with\_fk(self, table\_name1, table\_name2, value1, value2, value3, cond):

self.\_\_cursor.execute(

sql.SQL('DELETE FROM {} WHERE {} in (SELECT {} FROM {} WHERE {} = {})').

format(sql.Identifier(table\_name1),

sql.Identifier(value1),

sql.Identifier(value2),

sql.Identifier(table\_name2),

sql.Identifier(value3),

sql.SQL(cond)))

self.\_\_context.commit()

Controller.py:

**def** delete(self, table\_name, column, value):

**try**:

**print**('===========')

**if** table\_name == 'author':

self.delete\_author(table\_name, column, value)

**elif** table\_name == 'books':

self.delete\_books(table\_name, column, value)

*# self.delete\_author(table\_name, column, value)*

**elif** table\_name == 'reader':

self.delete\_reader(table\_name, column, value)

*# self.delete\_author(table\_name, column, value)*

**else**:

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося видалити дані з таблиці", error)

**def** delete\_author(self, table\_name, column, value):

**try**:

**print**('===========')

self.model.delete\_with\_fk('subscription', 'books', 'books\_id', 'books\_id', 'author\_id', value)

self.model.delete\_data('books', 'author\_id', value)

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося видалити дані з таблиці author", error)

**def** delete\_books(self, table\_name, column, value):

**try**:

**print**('===========')

self.model.delete\_data('subscription', 'books\_id', value)

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося видалити дані з таблиці books", error)

**def** delete\_reader(self, table\_name, column, value):

**try**:

**print**('===========')

self.model.delete\_data('subscription', 'reader\_id', value)

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося видалити дані з таблиці reader", error)

Для роботи функції потрібні назва таблиці, назва PK таблиці (отримується автоматично за назвою таблиці) та його значення. SQL-запит на видалення даних виконується за допомогою запиту DELETE. Також передбачена можливість видалення батьківських даних та відповідним їм дочірнім.

1. Редагування даних таблиці:

Model.py:

**def** change\_data(self, table\_name, values):

*# Формування рядка значень SET запиту UPDATE*

line = ''

condition = values.pop('condition')

**for** key **in** values:

**if** values[key]:

line += key + '=%(' + key + ')s,'

*# Виконання SQL-запиту на оновлення даних в таблиці*

self.\_\_cursor.execute(

sql.SQL('UPDATE {} SET ' + line[:-1] + ' WHERE {}')

.format(sql.Identifier(table\_name), sql.SQL(condition)),

values)

self.\_\_context.commit()

Controller.py:

**def** update(self, table\_name, values):

**try**:

**print**('===========')

self.model.change\_data(table\_name, values)

self.view.display\_update(table\_name, values)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося оновити дані в таблиці", error)

Як і при внесенні даних, спочатку в циклі формується рядок значень, далі виконується SQL-запит UPDATE, який оновлює дані таблиці. Для роботи функції потрібні назва таблиці та кортеж values, в якому назви колонок, значення яких треба змінити та відповідно нові значення для них.

**Лістинги фрагментів програми запитів пошуку**

Model.py:

**def** join\_general(self, main\_query, condition=""):

new\_cond = condition

**if** condition:

new\_cond = "WHERE " + condition

t1 = time.time()

self.\_\_cursor.execute(main\_query.format(new\_cond))

t2 = time.time()

**print**((t2 - t1) \* 1000)

**return** self.\_\_cursor.fetchall()

*# reader subscription*

**def** join\_reader\_subscription(self, condition=""):

**return** self.join\_general("""SELECT \* FROM reader as r

JOIN subscription as s

ON r.reader\_id = s.reader\_id {}

ORDER BY r.reader\_id ASC""", condition)

*# author books*

**def** join\_author\_books(self, condition=""):

**return** self.join\_general("""SELECT \* FROM author as a

JOIN books as b

ON a.author\_id = b.author\_id {}

ORDER BY b.books\_id DESC""", condition)

*# subscription books*

**def** join\_subscription\_books(self, condition=""):

**return** self.join\_general("""SELECT \* FROM subscription as s

JOIN books as b

ON s.books\_id = b.books\_id {}

ORDER BY b.books\_id ASC""", condition)

Controller.py:

Як вже було зазначено, реалізовано 3 різні запити для пошуку, а саме: в таблицях reader та subscription, author та books і subscription та books, а також загальна функція, яка приймає повну умову з трьох запитів, рахує їх швидкодію, та виводить результат пошуку.

Для першого запиту користувачу потрібно ввести ім’я читача, дати початку та закінчення абонементу, тоді відбудеться перевірка колонок, де startdate >= обраного початку, duedate <= обраного закінчення.

Другий запит виводить усі значення таблиці, в яких колонка isavailable дорівнює обраному користувачем логічному значенню True або False.

Третій запит виводить усі значення таблиці, в яких колонка name повністю або частково відповідає вказаному користувачем рядку.

**Лістинг фрагменту програми генерування випадкових даних в таблицях БД**

Model.py:

Controller.py:

**def** search(self, num):

**try**:

**print**('===========')

**if** num == '1':

tname = 'Reader | Subscription'

startd = input('Start date (xxxx-yy-zz) : ')

dued = input('Due date (xxxx-yy-zz) : ')

rname = input('Reader name : ')

cond = str("s.startdate >= " + "'" + str(startd) + "'" + " AND s.duedate <= " + "'" + str(dued) + "'" +

" AND r.name LIKE " + "'" + str(rname) + "%'")

*# print(cond)*

c = self.model.join\_reader\_subscription(cond)

self.view.show(c)

**if** **not** c:

**print**("Row not found")

**else**:

self.view.display\_search(tname, cond)

self.view.show(c)

**elif** num == '2':

tname = 'Author | Books'

b = input('(True or False : ')

size = input('Book size : ')

aname = input('Part of author name : ')

cond = str("b.isavailable = " + str(b) + " AND b.size >= " + str(size)

+ " AND a.name LIKE " + "'" + str(aname) + "%'")

*# print(cond)*

c = self.model.join\_author\_books(cond)

**if** **not** c:

**print**("Row not found")

**else**:

self.view.display\_search(tname, cond)

self.view.show(c)

**elif** num == '3':

tname = 'Books | Subscription'

ch = input('Part of book name : ')

duedate = input('Due date (xxxx-yy-zz) : ')

size = input('Book size : ')

cond = str("b.name LIKE " + "'" + str(ch) + "%'"

+ " AND s.duedate <= " + "'" + str(duedate) + "'"

+ " AND b.size >= " + str(size))

*# print(cond)*

c = self.model.join\_author\_books(cond)

**print**("+++")

self.view.show(c)

**print**("+++")

**if** **not** c:

**print**("Row not found")

**else**:

self.view.display\_search(tname, cond)

self.view.show(c)

**else**:

**print**("Incorrect input")

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося знайти дані в таблицях", error)

В циклі функція проходить обрану таблицю та перевіряє тип кожної колонки та в залежності від нього за допомогою запиту INSERT генерує та додає «рандомізоване» значення цього ж типу в таблицю. В моїх таблицях використовується такі типи: integer, date, character varying та boolean. Для роботи функції потрібні назва таблиці та яку кількість даних генерувати.

**Результати роботи програми**

1. На Рисунку 4 зображено роботу «Show table» (перегляд таблиці) для значень таблиці «reader», а на Рисунку 5 ця ж таблиця в застосунку pgAdmin 3. На Рисунку 6 зображений випадок, коли обрано неправильне значення для номеру таблиці.

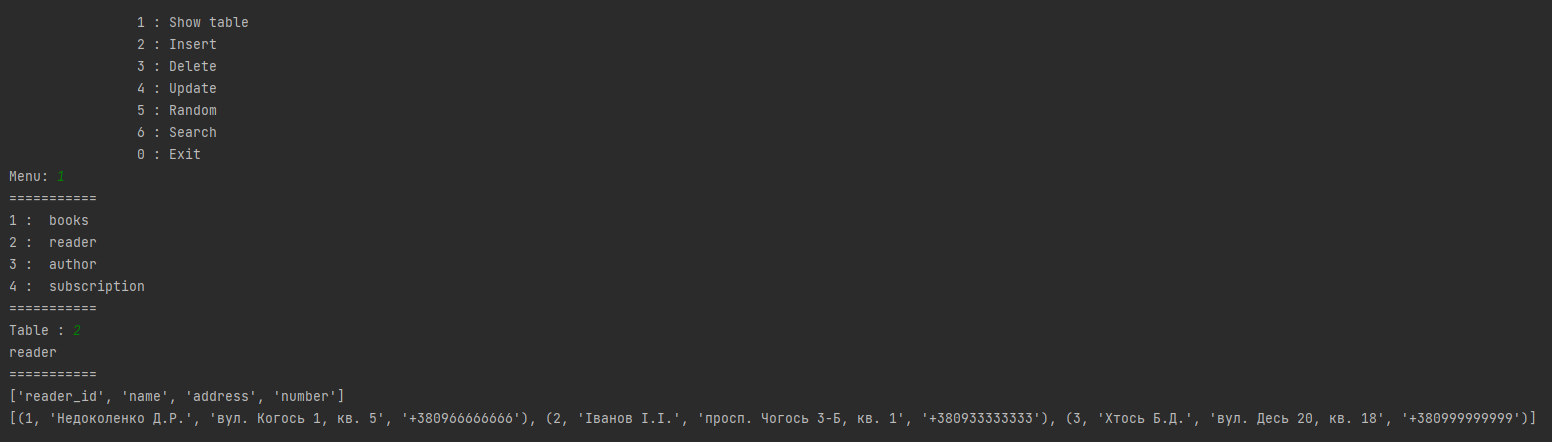


Рисунок 4 – Перегляд таблиці «reader» в програмі

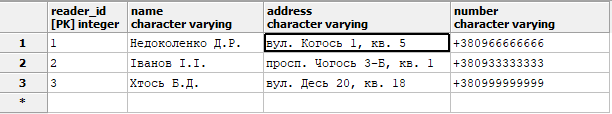


Рисунок 5 –Таблиця «reader» (pgAdmin 3)

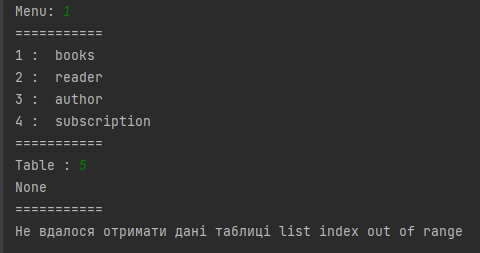


Рисунок 6 – Повідомлення про помилку, якщо обраного номера таблиці не існує

1. На Рисунку 7 зображено роботу «Insert» (внесення в таблицю) для таблиці «reader» з внесенням значень для усіх колонок та ця ж таблиця після операції в pgAdmin 3 (Рисунок 8). Якщо в таблицю вносити дані не для всіх колонок на екрану з’явиться повідомлення про помилку (Рисунок 9). При спробі додати в колонку неправильний тип, на екрані з’явиться повідомлення про помилку (Рисунок 10).

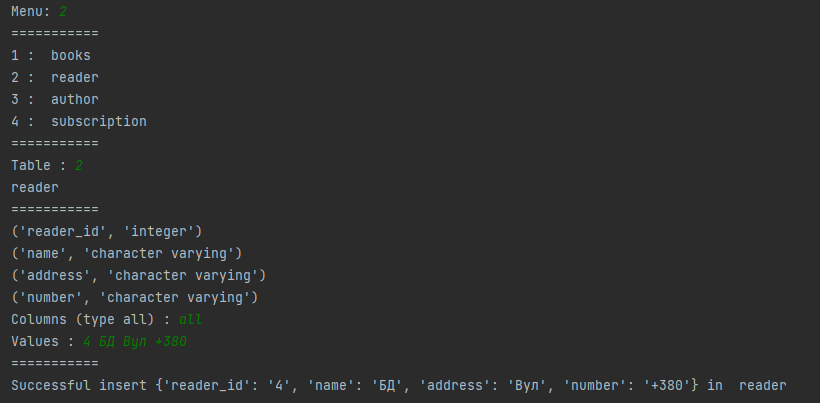


Рисунок 7 – Внесення даних в таблицю «reader» в програмі

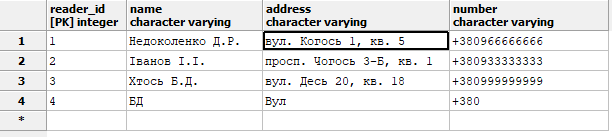


Рисунок 8 – Таблиця «reader» з новими даними (pgAdmin 3)

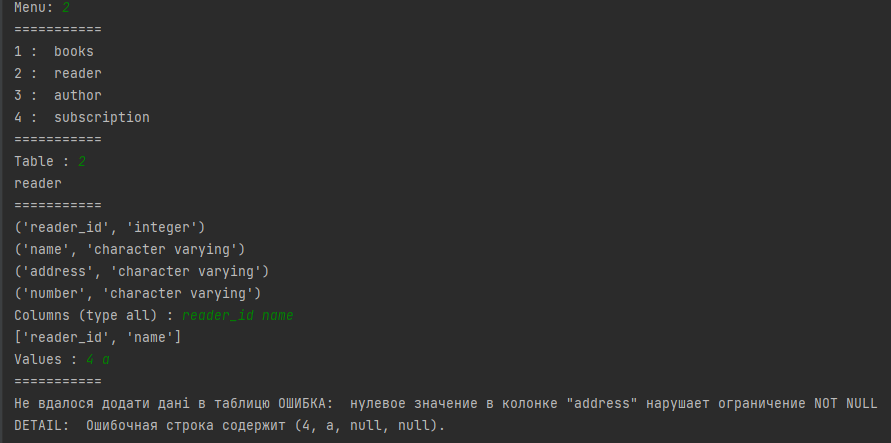


Рисунок 9 – Помилка при внесенні даних не для усіх колонок NOT NULL

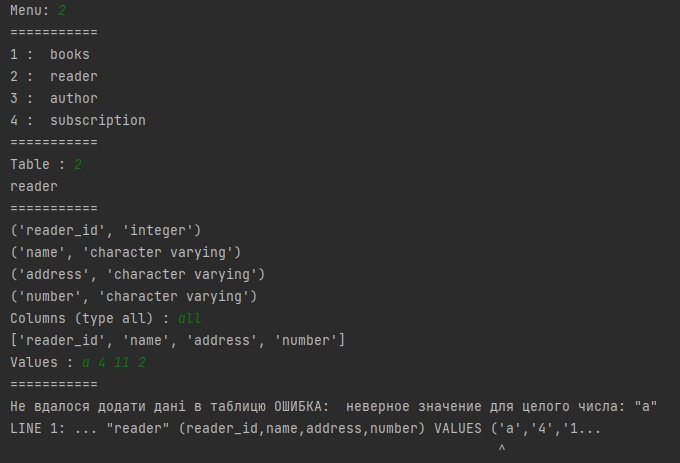


Рисунок 10 – Помилка при внесенні значень неправильного типу

1. На Рисунку 11 зображено роботу «Delete» (видалення даних) для значення з таблиці «subscription» (Рисунок 8). Ця ж таблиця до та після видалення в pgAdmin 3 зображена на Рисунку 12 та 13 відповідно.

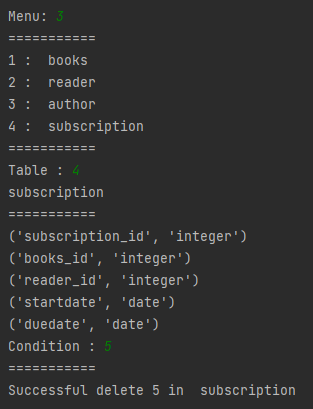


Рисунок 11 – Видалення даних з таблиці «subscription»

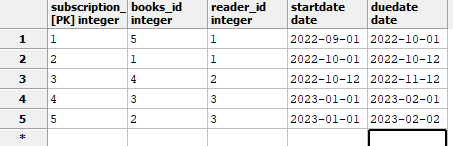


Рисунок 12 – Таблиця «subscription» (pgAdmin 3)

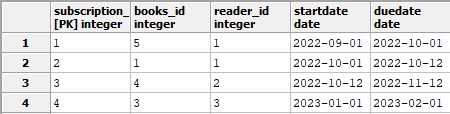


Рисунок 13 – Таблиця «subscription» після видалення (pgAdmin 3)

1. На Рисунку 14 зображено роботу «Update» (редагування даних в таблиці) для таблиці «author», в яку попередньо було створено автора newAuthor (Рисунок 15). Результат редагування в pgAdmin 3 зображено на Рисунку 16. Перевірку на зміну на ті самі дані не передбачено.

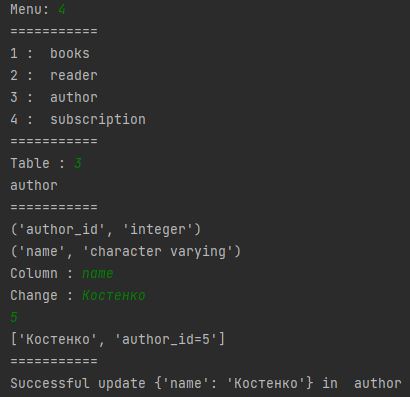


Рисунок 14 – Зміна імені автора з author\_id = 5 на Костенко в таблиці «author»

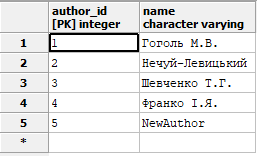


Рисунок 15 –Таблиця «author» (pgAdmin 3)

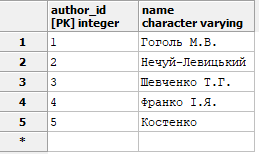


Рисунок 16 – Таблиця «author» після редагування (pgAdmin 3)

1. На Рисунку 17 зображено автоматичне пакетне генерування 100000 «рандомізованих» даних у таблицю «books». Частину цієї ж таблиці після генерування зображено на Рисунку 18.

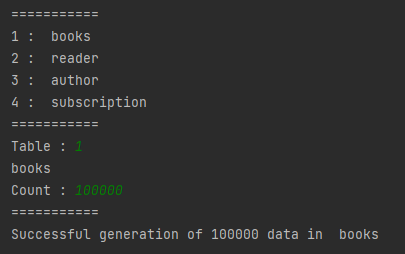


Рисунок 17 – Генерація 100000 випадкових даних в таблицю «books»

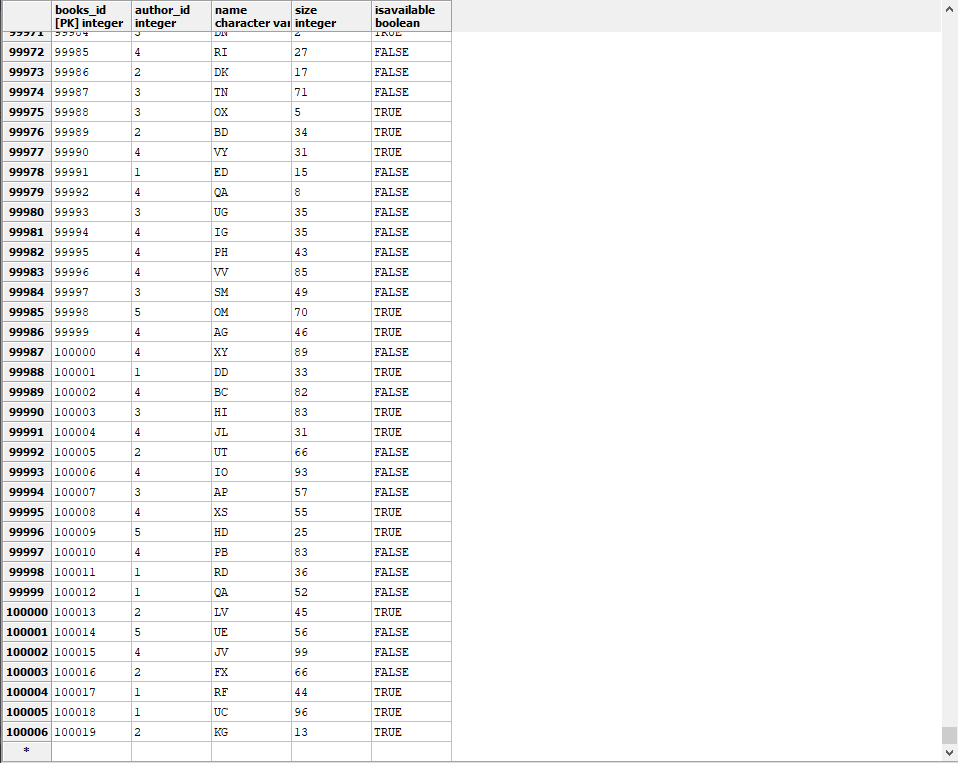


Рисунок 18 – Частина таблиці «books» після генерації випадкових даних (pgAdmin 3)

1. На Рисунку 19, 20 та 21 зображено пошук за одним із трьох запитів зазначених раніше відповідно.

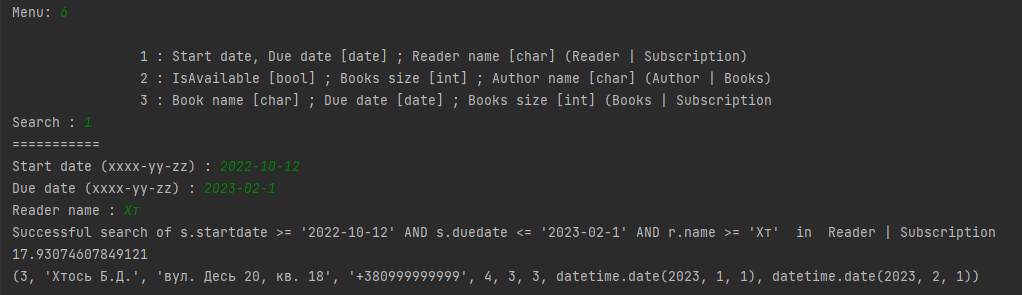


Рисунок 19 – Пошук значень за діапазоном дат початку та закінчення абонементу та за ім’ям читача в таблицях «reader» та «subscription»

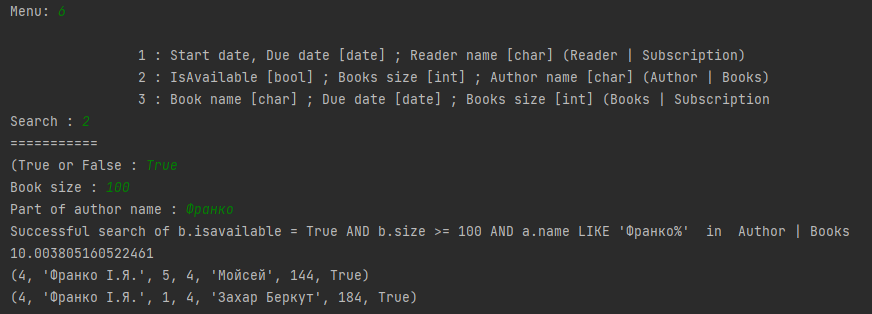


Рисунок 20 – Пошук значень за isavailable=False, діапазоном розміру книги та за ім’ям автора в таблицях «author» та «books»

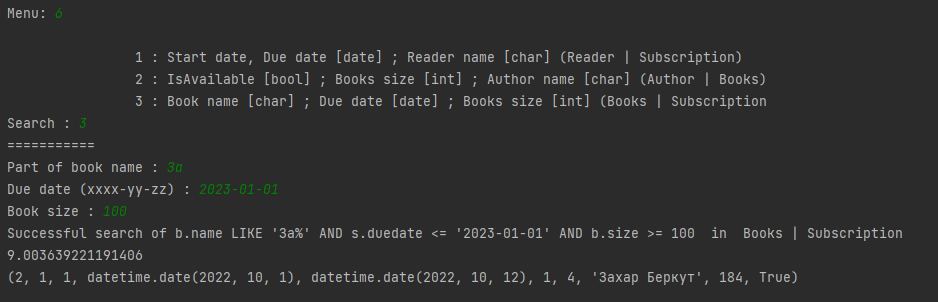


Рисунок 21 – Пошук значень за діапазоном назви книги, дати закінчення абонементу та розміру книги в таблицях «books» та «subscription»

**Лістинги та результати виконання операції вилучення запису батьківської таблиці та виведення вмісту дочірньої таблиці після цього вилучення**

Розглянемо батьківську таблицю «author» та дочірню «books». Щоб вилучити рядок з «author» треба спочатку видалити всі пов’язані з ним рядки з таблиці «books». Проте, треба зазначити, що «books» є батьківською для таблиці «subscription». Таким чином, нам спочатку треба видалити все з «subscription», потім з «books», а вже потім з «author». Для цього, окрім звичайної функції видалення, було створено функцію delete-with-fk.

Model.py:

**def** delete\_data(self, table\_name, value, cond):

self.\_\_cursor.execute(

sql.SQL('DELETE FROM {} WHERE {} = {}').format(sql.Identifier(table\_name),

sql.Identifier(value),

sql.SQL(cond)))

self.\_\_context.commit()

**def** delete\_with\_fk(self, table\_name1, table\_name2, value1, value2, value3, cond):

self.\_\_cursor.execute(

sql.SQL('DELETE FROM {} WHERE {} in (SELECT {} FROM {} WHERE {} = {})').

format(sql.Identifier(table\_name1),

sql.Identifier(value1),

sql.Identifier(value2),

sql.Identifier(table\_name2),

sql.Identifier(value3),

sql.SQL(cond)))

self.\_\_context.commit()

Controller.py:

**def** delete(self, table\_name, column, value):

**try**:

**print**('===========')

**if** table\_name == 'author':

self.delete\_author(table\_name, column, value)

**elif** table\_name == 'books':

self.delete\_books(table\_name, column, value)

*# self.delete\_author(table\_name, column, value)*

**elif** table\_name == 'reader':

self.delete\_reader(table\_name, column, value)

*# self.delete\_author(table\_name, column, value)*

**else**:

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося видалити дані з таблиці", error)

**def** delete\_author(self, table\_name, column, value):

**try**:

**print**('===========')

self.model.delete\_with\_fk('subscription', 'books', 'books\_id', 'books\_id', 'books\_id', value)

self.model.delete\_data('books', 'author\_id', value)

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося видалити дані з таблиці author", error)

**def** delete\_books(self, table\_name, column, value):

**try**:

**print**('===========')

self.model.delete\_data('subscription', 'books\_id', value)

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося видалити дані з таблиці books", error)

**def** delete\_reader(self, table\_name, column, value):

**try**:

**print**('===========')

self.model.delete\_data('subscription', 'reader\_id', value)

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося видалити дані з таблиці reader", error)

На Рисунках 22-24 зображені відповідно таблиці «author», «books» та «subscription». Спробуємо видалити автора з author\_id = 5 (Рисунок 25).

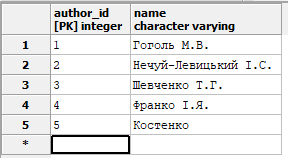


Рисунок 22 - Таблиця «author» (pgAdmin 3)

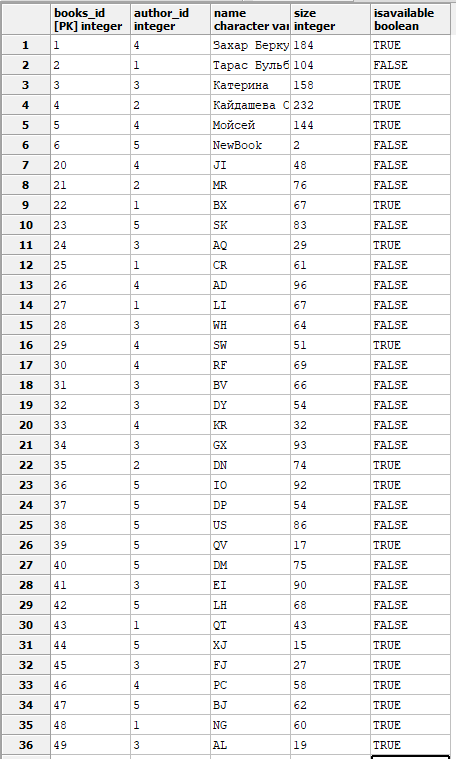


Рисунок 23 – Фрагмент таблиці «books» (pgAdmin 3)

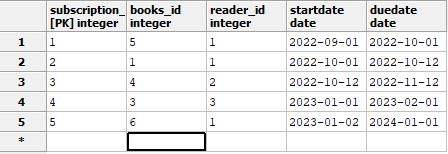


Рисунок 24 - Таблиця «subscription» (pgAdmin 3)

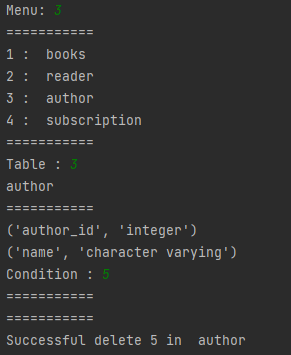


Рисунок 25 – Операція видалення рядку з author\_id = 5 з таблиці «author»

Результати вилучення зображені на Рисунках 26-28. Як можна побачити, все працює коректно. В «subscription» було два рядки з різними книгами (books\_id = 5 та 6), які мають одного автора, тому вони були видалені.

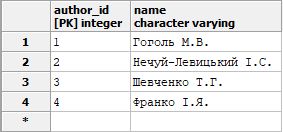


Рисунок 26 - Таблиця «author» після видалення (pgAdmin 3)

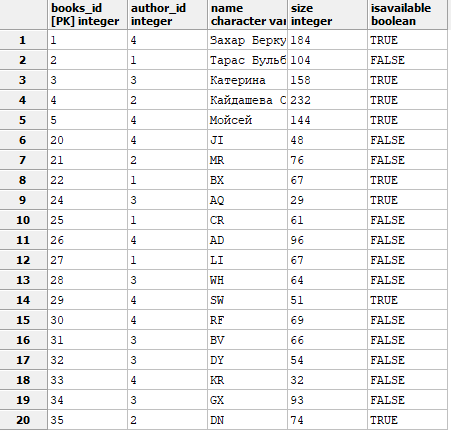


Рисунок 27 – Фрагмент таблиці «books» після видалення (pgAdmin 3)

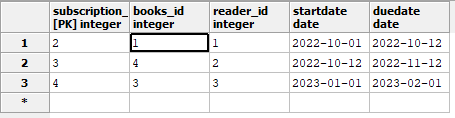


Рисунок 28 - Таблиця «subscription» після видалення (pgAdmin 3)

**Результати виконання операції вставки запису в дочірню таблицю та виведення повідомлення про її неможливість, якщо в батьківські таблиці нема відповідного запису**

Спробуємо додати в дочірню таблицю «books» (Рисунок 27) рядок зі значенням author\_id = 5, якого не існує в таблиці «author» (Рисунок 28). Результат операції зображений на Рисунку 29.

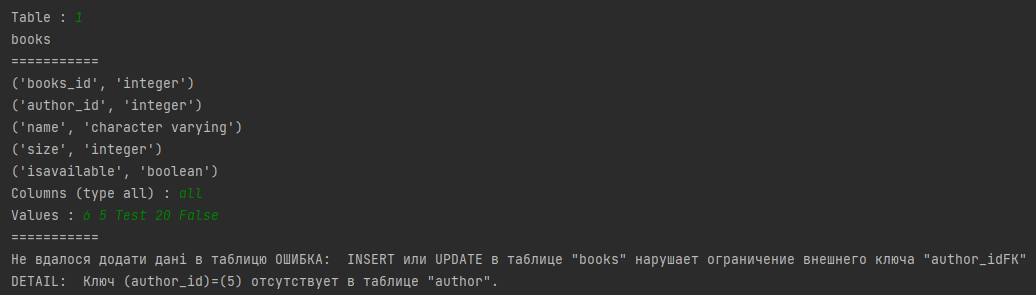


Рисунок 29 – Вставка запису в дочірню таблицю «books» з неіснуючим батьківським записом таблиці «author»

**Посилання для навігації по тексту програми**

[main.py](#main)

[Controller.py](#Controller)

[Model.py](#Model)

[View.py](#View)

**Текст програми**

Посилання на Git з текстом програми: <https://github.com/Zarrylon/db_kpi>

1. main.py:

**from** Model **import** Model\_db

**from** View **import** View\_db

**from** Controller **import** Controller\_db

c = Controller\_db(Model\_db("postgres", "qwerty", "localhost", "5434", "Library"), View\_db())

c.main\_menu()

**print**("Connection closed")

1. Controller.py:

**import** psycopg2

**class** Controller\_db:

**def** \_\_init\_\_(self, model, view):

self.model = model

self.view = view

**def** get\_table\_list(self):

**try**:

**print**('===========')

n = self.model.get\_table\_names()

self.view.show\_table(n)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося отримати назви таблиць", error)

**def** get\_columns(self, table\_name):

**try**:

**print**('===========')

cl = self.model.get\_column\_types(table\_name)

self.view.show(cl)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося отримати колонки таблиці", error)

**def** get\_names(self, table\_name):

**try**:

*# print('===========')*

cl = self.model.get\_column\_names(table\_name)

*# self.view.show(cl)*

**return** cl

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося отримати назви колонок таблиці", error)

**def** show\_table(self, table\_name):

**try**:

**print**('===========')

t = self.model.get\_table\_data(table\_name)

self.view.show(t)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося отримати дані таблиці", error)

**def** digit\_to\_table\_name(self):

**print**('===========')

number = input('Table : ')

**if** str(number).isdigit():

**if** number == '1':

**return** 'books'

**elif** number == '2':

**return** 'reader'

**elif** number == '3':

**return** 'author'

**elif** number == '4':

**return** 'subscription'

**else**:

**print**('Incorrect input')

self.digit\_to\_table\_name()

@staticmethod

**def** table\_name\_to\_pk(table\_name):

**if** table\_name == 'books':

**return** 'books\_id'

**elif** table\_name == 'reader':

**return** 'reader\_id'

**elif** table\_name == 'author':

**return** 'author\_id'

**elif** table\_name == 'subscription':

**return** 'subscription\_id'

**else**:

**print**('Incorrect input')

**return** ' '

**def** insert(self, table\_name, values):

**try**:

**print**('===========')

self.model.insert\_data(table\_name, values)

self.view.display\_insert(table\_name, values)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося додати дані в таблицю", error)

**def** delete(self, table\_name, column, value):

**try**:

**print**('===========')

**if** table\_name == 'author':

self.delete\_author(table\_name, column, value)

**elif** table\_name == 'books':

self.delete\_books(table\_name, column, value)

*# self.delete\_author(table\_name, column, value)*

**elif** table\_name == 'reader':

self.delete\_reader(table\_name, column, value)

*# self.delete\_author(table\_name, column, value)*

**else**:

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося видалити дані з таблиці", error)

**def** delete\_author(self, table\_name, column, value):

**try**:

**print**('===========')

self.model.delete\_with\_fk('subscription', 'books', 'books\_id', 'books\_id', 'author\_id', value)

self.model.delete\_data('books', 'author\_id', value)

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося видалити дані з таблиці author", error)

**def** delete\_books(self, table\_name, column, value):

**try**:

**print**('===========')

self.model.delete\_data('subscription', 'books\_id', value)

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося видалити дані з таблиці books", error)

**def** delete\_reader(self, table\_name, column, value):

**try**:

**print**('===========')

self.model.delete\_data('subscription', 'reader\_id', value)

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося видалити дані з таблиці reader", error)

**def** update(self, table\_name, values):

**try**:

**print**('===========')

self.model.change\_data(table\_name, values)

self.view.display\_update(table\_name, values)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося оновити дані в таблиці", error)

**def** random(self, table\_name, count):

**try**:

**print**('===========')

self.model.generate\_data(table\_name, count)

*# self.show\_table(table\_name)*

self.view.display\_random(table\_name, count)

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося згенерувати дані в таблиці", error)

**def** search(self, num):

**try**:

**print**('===========')

**if** num == '1':

tname = 'Reader | Subscription'

startd = input('Start date (xxxx-yy-zz) : ')

dued = input('Due date (xxxx-yy-zz) : ')

rname = input('Reader name : ')

cond = str("s.startdate >= " + "'" + str(startd) + "'" + " AND s.duedate <= " + "'" + str(dued) + "'" +

" AND r.name LIKE " + "'" + str(rname) + "%'")

*# print(cond)*

c = self.model.join\_reader\_subscription(cond)

**if** **not** c:

**print**("Row not found")

**else**:

self.view.display\_search(tname, cond)

self.view.show(c)

**elif** num == '2':

tname = 'Author | Books'

b = input('(True or False : ')

size = input('Book size : ')

aname = input('Part of author name : ')

cond = str("b.isavailable = " + str(b) + " AND b.size >= " + str(size)

+ " AND a.name LIKE " + "'" + str(aname) + "%'")

*# print(cond)*

c = self.model.join\_author\_books(cond)

**if** **not** c:

**print**("Row not found")

**else**:

self.view.display\_search(tname, cond)

self.view.show(c)

**elif** num == '3':

tname = 'Books | Subscription'

ch = input('Part of book name : ')

duedate = input('Due date (xxxx-yy-zz) : ')

size = input('Book size : ')

cond = str("b.name LIKE " + "'" + str(ch) + "%'"

+ " AND s.duedate <= " + "'" + str(duedate) + "'"

+ " AND b.size >= " + str(size))

*# print(cond)*

c = self.model.join\_author\_books(cond)

**if** **not** c:

**print**("Row not found")

**else**:

self.view.display\_search(tname, cond)

self.view.show(c)

**else**:

**print**("Incorrect input")

**except** (Exception, psycopg2.Error) **as** error:

**print**("Не вдалося знайти дані в таблицях", error)

**def** main\_menu(self):

is\_exit = False

**while** **not** is\_exit:

self.view.menu()

choice = input('Menu: ')

**if** choice == '0':

is\_exit = True

**elif** choice == '1':

self.get\_table\_list()

tn = self.digit\_to\_table\_name()

**print**(tn)

self.show\_table(tn)

self.model.clear\_transaction()

**elif** choice == '2':

self.get\_table\_list()

tn = self.digit\_to\_table\_name()

**print**(tn)

self.get\_columns(tn)

columns = input("Columns (type all) : ").split()

**if** columns == ['all']:

columns = self.get\_names(tn)

*# print(columns)*

val = input("Values : ").split(' ')

values = {key: value **for** (key, value) **in** zip(columns, val)}

self.insert(tn, values)

self.model.clear\_transaction()

**elif** choice == '3':

self.get\_table\_list()

tn = self.digit\_to\_table\_name()

**print**(tn)

self.get\_columns(tn)

t\_id = self.table\_name\_to\_pk(tn)

condition = input("Condition : ")

self.delete(tn, t\_id, condition)

self.model.clear\_transaction()

**elif** choice == '4':

self.get\_table\_list()

tn = self.digit\_to\_table\_name()

**print**(tn)

self.get\_columns(tn)

columns = (input("Column : ") + ' condition').split(' ')

val1 = input("Change : ").split(' ')

**print**('pk\_id : ')

val2 = (self.table\_name\_to\_pk(tn) + '=' + input('')).split(' ')

val = (val1 + val2)

**print**(val)

values = {key: value **for** (key, value) **in** zip(columns, val)}

self.update(tn, values)

self.model.clear\_transaction()

**elif** choice == '5':

self.get\_table\_list()

tn = self.digit\_to\_table\_name()

**print**(tn)

count = input('Count : ')

self.random(tn, count)

self.model.clear\_transaction()

**elif** choice == '6':

self.view.choose\_option()

c = input('Search : ')

self.search(c)

self.model.clear\_transaction()

**else**:

**print**('Incorrect input')

1. Model.py:

**import** psycopg2

**from** psycopg2 **import** sql

**import** time

**class** Model\_db:

**def** \_\_init\_\_(self, username, password, host, port, dbname):

self.\_\_context = psycopg2.connect(user=username, password=password,

host=host, port=port, database=dbname)

self.\_\_cursor = self.\_\_context.cursor()

self.\_\_table\_names = None

**def** \_\_del\_\_(self):

self.\_\_cursor.close()

self.\_\_context.close()

**def** clear\_transaction(self):

self.\_\_context.rollback()

*# Отримання імен таблиць БД*

**def** get\_table\_names(self):

**if** self.\_\_table\_names **is** None:

self.\_\_cursor.execute(("""SELECT table\_name

FROM information\_schema.tables

WHERE table\_schema = 'public'"""))

self.\_\_table\_names = [table[0] **for** table **in** self.\_\_cursor]

**return** self.\_\_table\_names

*# Отримання імен та типів стовпчиків таблиці*

**def** get\_column\_types(self, table\_name):

self.\_\_cursor.execute("""SELECT column\_name, data\_type

FROM information\_schema.columns

WHERE table\_schema = 'public' AND table\_name = %s

ORDER BY table\_schema, table\_name""", (table\_name,))

**return** self.\_\_cursor.fetchall()

*# Отримання імен стовпчиків таблиці*

**def** get\_column\_names(self, table\_name):

self.\_\_cursor.execute("""

SELECT column\_name FROM information\_schema.columns

WHERE table\_schema = 'public' AND table\_name = %s

ORDER BY table\_schema, table\_name""", (table\_name,))

**return** [x[0] **for** x **in** self.\_\_cursor.fetchall()]

*# Отримання зовнішніх ключів таблиці*

**def** get\_foreign\_key\_info(self, table\_name):

self.\_\_cursor.execute("""

SELECT kcu.column\_name, ccu.table\_name AS

foreign\_table\_name,

ccu.column\_name AS foreign\_column\_name

FROM information\_schema.table\_constraints AS tc

JOIN information\_schema.key\_column\_usage AS kcu

ON tc.constraint\_name = kcu.constraint\_name

AND tc.table\_schema = kcu.table\_schema

JOIN information\_schema.constraint\_column\_usage AS ccu

ON ccu.constraint\_name = tc.constraint\_name

AND ccu.table\_schema = tc.table\_schema

WHERE tc.constraint\_type = 'FOREIGN KEY' AND

tc.table\_name=%s;""", (table\_name,))

**return** self.\_\_cursor.fetchall()

*# Отримання даних таблиці БД*

**def** get\_table\_data(self, table\_name):

*# Отримання Id таблиці*

id\_column = self.get\_column\_types(table\_name)[0][0]

cursor = self.\_\_cursor

**try**:

*# Отримання даних таблиці по ключу*

cursor.execute(

sql.SQL('SELECT \* FROM {} ORDER BY {} ASC'

).format(sql.Identifier(table\_name), sql.SQL(id\_column)))

**except** Exception **as** e:

**return** str(e)

**return** [col.name **for** col **in** cursor.description], cursor.fetchall()

*# Вставка даних в таблицю*

**def** insert\_data(self, table\_name, values):

*# Формування рядка значень VALUES запиту INSERT*

line = ''

columns = '('

**for** key **in** values:

**if** values[key]:

line += '%(' + key + ')s,'

columns += key + ','

columns = columns[:-1] + ')'

*# Виконання SQL-запиту на вставку даних в таблицю*

self.\_\_cursor.execute(

sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')')

.format(sql.Identifier(table\_name), sql.SQL(columns)),

values)

self.\_\_context.commit()

*# Видалення даних в таблиці*

**def** delete\_data(self, table\_name, value, cond):

self.\_\_cursor.execute(

sql.SQL('DELETE FROM {} WHERE {} = {}').format(sql.Identifier(table\_name),

sql.Identifier(value),

sql.SQL(cond)))

self.\_\_context.commit()

**def** delete\_with\_fk(self, table\_name1, table\_name2, value1, value2, value3, cond):

self.\_\_cursor.execute(

sql.SQL('DELETE FROM {} WHERE {} in (SELECT {} FROM {} WHERE {} = {})').

format(sql.Identifier(table\_name1),

sql.Identifier(value1),

sql.Identifier(value2),

sql.Identifier(table\_name2),

sql.Identifier(value3),

sql.SQL(cond)))

self.\_\_context.commit()

*# Оновлення даних в таблиці*

**def** change\_data(self, table\_name, values):

*# Формування рядка значень SET запиту UPDATE*

line = ''

condition = values.pop('condition')

**for** key **in** values:

**if** values[key]:

line += key + '=%(' + key + ')s,'

*# Виконання SQL-запиту на оновлення даних в таблиці*

self.\_\_cursor.execute(

sql.SQL('UPDATE {} SET ' + line[:-1] + ' WHERE {}')

.format(sql.Identifier(table\_name), sql.SQL(condition)),

values)

self.\_\_context.commit()

*# Генерація випадкових даних і заповнення таблиці*

**def** generate\_data(self, table\_name, count):

types = self.get\_column\_types(table\_name)

fk\_array = self.get\_foreign\_key\_info(table\_name)

select\_subquery = ""

insert\_query = "INSERT INTO " + table\_name + " ("

**for** i **in** range(1, len(types)):

t = types[i]

name = t[0]

types = t[1]

fk = [x **for** x **in** fk\_array **if** x[0] == name]

**if** fk:

select\_subquery += ('(SELECT {} FROM {} ORDER BY RANDOM(), ser LIMIT 1)'.format(fk[0][2], fk[0][1]))

**elif** types == 'integer':

select\_subquery += 'trunc(random()\*100)::INT'

**elif** types == 'character varying':

select\_subquery += 'chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + random()\*25)::INT)'

**elif** types == 'date':

select\_subquery += """ date(timestamp '2022-01-01' +

random() \*

(timestamp '2024-01-01' - timestamp '2022-01-01'))"""

**elif** types == 'boolean':

select\_subquery += 'random()::INT::BOOLEAN'

insert\_query += name

**if** i != len(types) - 1:

select\_subquery += ','

insert\_query += ','

**else**:

insert\_query += ') '

self.\_\_cursor.execute(

insert\_query + "SELECT " + select\_subquery +

" FROM generate\_series(1," + str(count) + ") as ser")

self.\_\_context.commit()

**def** join\_general(self, main\_query, condition=""):

new\_cond = condition

**if** condition:

new\_cond = "WHERE " + condition

t1 = time.time()

self.\_\_cursor.execute(main\_query.format(new\_cond))

t2 = time.time()

**print**((t2 - t1) \* 1000)

**return** self.\_\_cursor.fetchall()

*# reader subscription*

**def** join\_reader\_subscription(self, condition=""):

**return** self.join\_general("""SELECT \* FROM reader as r

JOIN subscription as s

ON r.reader\_id = s.reader\_id {}

ORDER BY r.reader\_id ASC""", condition)

*# author books*

**def** join\_author\_books(self, condition=""):

**return** self.join\_general("""SELECT \* FROM author as a

JOIN books as b

ON a.author\_id = b.author\_id {}

ORDER BY b.books\_id DESC""", condition)

*# subscription books*

**def** join\_subscription\_books(self, condition=""):

**return** self.join\_general("""SELECT \* FROM subscription as s

JOIN books as b

ON s.books\_id = b.books\_id {}

ORDER BY b.books\_id ASC""", condition)

1. View.py:

**class** View\_db:

@staticmethod

**def** show\_table(db):

i = 1

**for** element **in** db:

**print**(i, ': ', element)

i += 1

@staticmethod

**def** show(db):

**for** element **in** db:

**print**(element)

@staticmethod

**def** menu():

**print**('''

1 : Show table

2 : Insert

3 : Delete

4 : Update

5 : Random

6 : Search

0 : Exit''')

@staticmethod

**def** display\_insert(tname, insert):

**print**('Successful insert', insert, 'in ', tname)

@staticmethod

**def** display\_delete(tname, delete):

**print**('Successful delete', delete, 'in ', tname)

@staticmethod

**def** display\_update(tname, update):

**print**('Successful update', update, 'in ', tname)

@staticmethod

**def** display\_random(tname, random\_count):

**print**('Successful generation of', random\_count, 'data in ', tname)

@staticmethod

**def** choose\_option():

**print**('''

1 : Start date, Due date [date] ; Reader name [char] (Reader | Subscription)

2 : IsAvailable [bool] ; Books size [int] ; Author name [char] (Author | Books)

3 : Book name [char] ; Due date [date] ; Books size [int] (Books | Subscription''')

@staticmethod

**def** display\_search(tname, search\_res):

**print**('Successful search of', search\_res, ' in ', tname)