

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних**

**систем**

**Розрахунково-графічна робота**

з дисципліни **Бази даних і засоби управління**

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

Виконала:

студентка ІII курсу

групи КВ-12

Павленко Л.П.

Перевірив: Павловський В. І.

Київ – 2023

*Мета роботи* є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

*Загальне завдання*:

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).

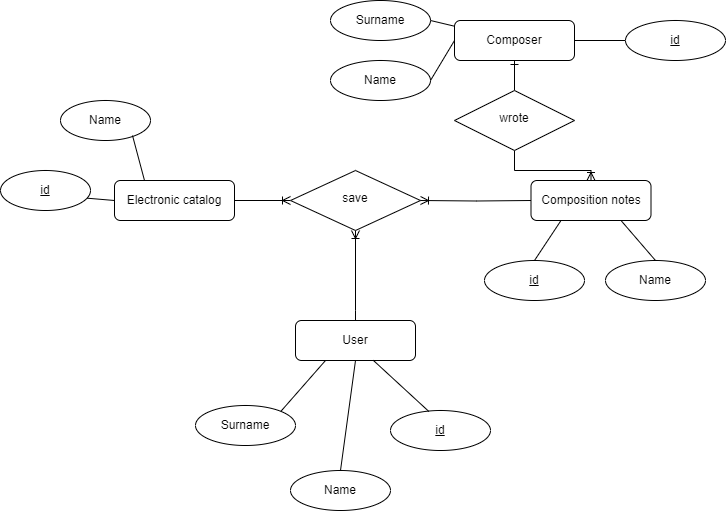


Рисунок 1 – ER-діаграма, побудована за нотацією “Пташиної лапки (Crow’s foot)”

**Сутності**

Згідно цієї області для того щоб побудувати бази даних виділено наступні сутності:

Сутність «Електронний каталог» з атрибутами : ID, назва;

Сутність «Ноти» з атрибутами : ID, назва композиції, номер композиції;

Сутність «Композитор» з атрибутами : ID, Ім'я, Прізвище.

Сутність «Користувач» з атрибутами : ID, Ім'я, Прізвище.

**Опис зв'язків**

Користувач завантажує ноти в каталог, отже ці сутності (користувач, ноти та каталог) утворюють зв'язок. Зв'язок передбачає, що електронний каталог може містити багато нот композицій, як і ноти композиції можуть зберігатися в декількох каталогах, отже зв'язок M:N. Користувач може додавати декілька нот композицій в декілька каталогів, отже зв’язок в цьому випадку M:N. Декілька користувачів можуть додати композицію в каталог, отже зв’язок також M:N

Композитор може написати декілька творів, натомість ноти композиції зазвичай мають одного композитора. Отже між сутностями Композитор та Ноти зв’язок 1:N.

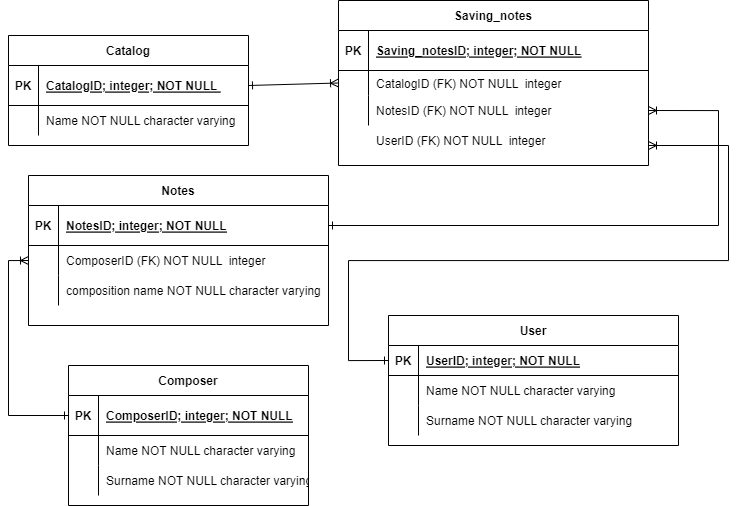


Рисунок 2 – Схема бази даних

**Схема меню користувача**

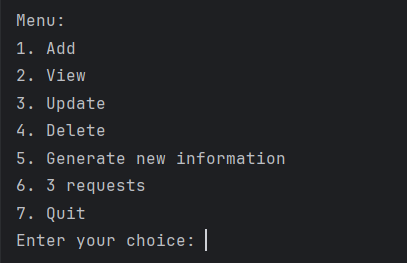


Рисунок 1 – Схема меню користувача

1. Перший пункт меню «Add». Вибираючи цей пункт, користувач може вибрати в меню, яке буде відображатись після вибору пункту бажану таблицю, в яку він буде додавати дані, після чого користувачу необхідно буде ввести дані, залежно від таблиці. Після введення користувачем ID, відбувається перевірка на наявність такого ж ідентифікатора та на тип даних введених користувачем, якщо ID наявне або тип даних введений невірно, то буде відображатись помилка, якщо всі дані введено правильно то буде відображатись повідомлення про успішне додавання даних.

2. Другий пункт «View». При виборі цього пункту, також буде відображатись додаткове меню для вибору таблиці, після вибору бажаного пункту, користувач може переглянути всю таблицю.

3. Третій пункт «Update». Після вибору цього пункту також буде відображатись меню з вибором таблиці, в якій користувач бажає змінити дані. Після вибору в цьому меню користувач може ввести дані, які бажає змінити та ідентифікатор того рядка, який бажає змінити.

4. Четвертий пункт «Delete». Коли користувач обирає цей пункт, відображається меню таблиць, вибираючи один з пунктів, користувач може ввести ID того рядка який бажає видалити, якщо користувач ввів ID, якого немає в таблиці, то виведеться текстове повідомлення про помилку.

5. П’ятий пункт «Generate new information». Якщо вибрати цей пункт, то в таблиці «User» можна сформувати рядки з випадковими значеннями, з кількість рядків, з якого значення по яке вводить користувач з клавіатури.

6. Шостий пункт «3 requests». Після вибору цього пункту, користувачу необхідно буде ввести значення констант для фільтрації. В даному пункті розміщено три запити. Після кожного введення констант для запиту, буде відображатись час виконання запиту, та таблиця за запитом.

**Мова програмування та бібліотеки, які використовувались**

При написанні даної програми використовувалась мова Python та використовувалось середовище розробки PyCharm 2023.2.3. Також використовувалась бібліотека psycopg2 для доступу до бази даних.

**Пункт №1. Реалізація функції перегляду у, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу**

**Скріншоти результатів виконання операції вилучення запису батьківськоїтаблиці та виведення вмісту дочірньої таблиці після вилучення**

На рисунках 2, 3 та 4 зображений вміст таблиць «Composer», «Notes», «Saving\_notes» до видалення рядка в таблиці «Composer».

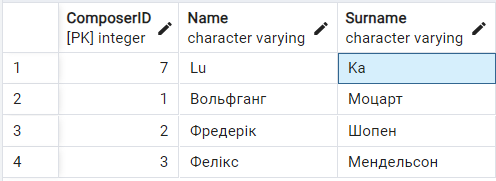


Рисунок 2 – Таблиця «Composer», до вилучення рядка з ідентифікатором ComposerID = 7

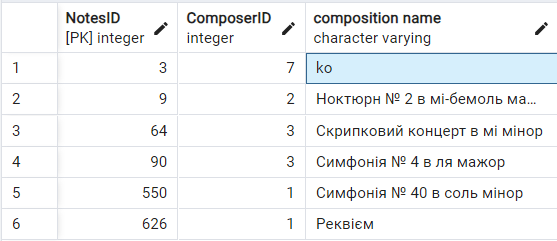


Рисунок 3 – Таблиця «Notes», до вилучення рядка з ідентифікатором ComposerID = 7

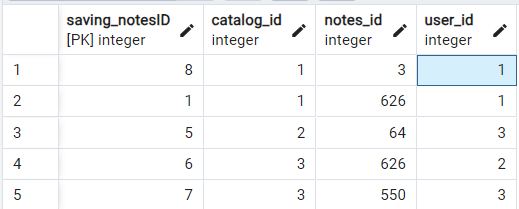


Рисунок 4 – таблиця «Saving\_notes», до вилучення рядка з ідентифікатором Notes ID = 3

На рисунку 5 зображено введення користувачем вибору операції, в даному випадку «Delete», вибору таблиці в якій користувач збирається видалити рядок , в даному випадку це батьківська таблиця «Composer» та номер ComposerID, який дорівнює 7.

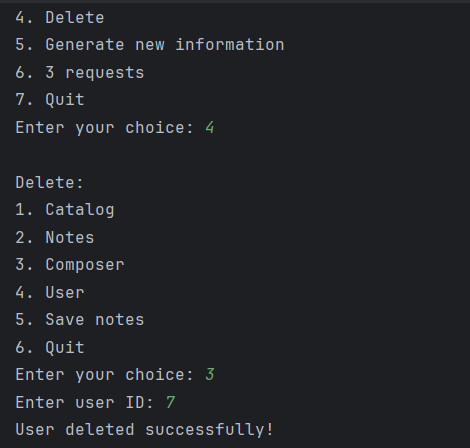


Рисунок 5 – Вибір користувача для видалення рядка з таблиці «Composer»

На рисунках 6, 7, 8 зображений вміст батьківської таблиці «Composer» та двох дочірніх «Notes», «Saving\_notes» після видалення рядка в Composer.

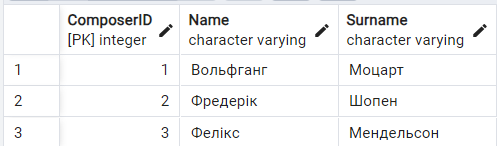


Рисунок 6 – таблиця «Composer» після видалення рядка з ідентифікатором ComposerID = 7

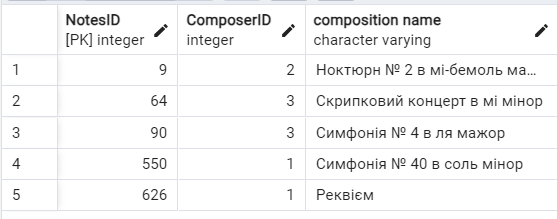


Рисунок 7 - таблиця «Notes» після видалення рядка з ідентифікатором ComposerID = 7

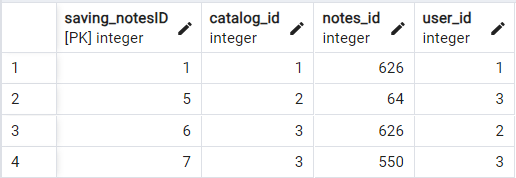


Рисунок 8 – таблиця «Saving\_notes» після видалення рядка з ідентифікатором NotesID = 3

**Результати виконання операції вставки запису в дочірню таблицю:**

На рисунку 9, зображується введення користувачем даних, які він бажає внести в saving notes ID, catalog ID, notes ID, user ID. Користувач ввів дані які є в таблиці «Notes» але відсутні в таблицях «Catalog» та «User», це можна перевірити на рисунках 10, 11 та 12.

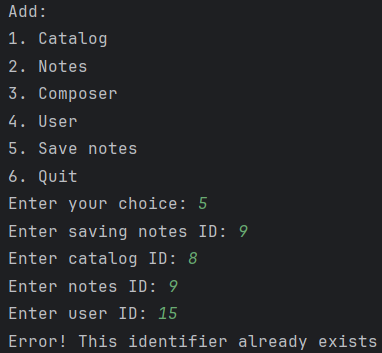


Рисунок 9 – Введення даних в дочірню таблицю «Save notes», де ми вводимо значення

На рисунках 10, 11 та 12 відображається вміст таблиць, який підтверджує чи правильні дані ввів користувач.

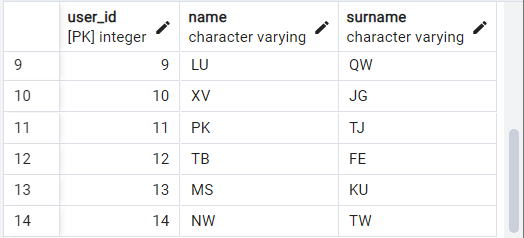


Рисунок 10 – Дані таблиці «User»

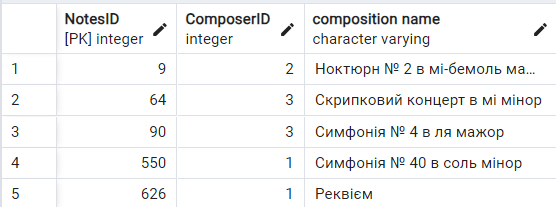


Рисунок 11 – Дані таблиці «Notes»

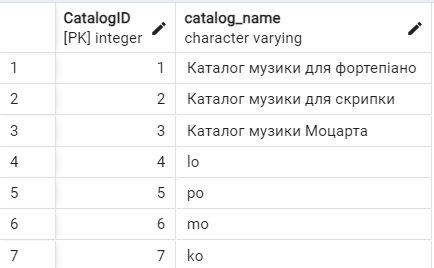


Рисунок 12 – Дані таблиці «Catalog»

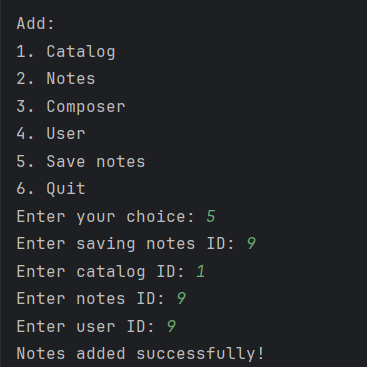
На рисунку 13 відображається введення користувачем даних які є в усіх трьох таблицях «User», «Notes» та «Catalog», ми це можемо перевірити на рисунках 10, 11 та 12, що і необхідно для внесення даних в дочірню таблицю, і як ми бачимо на рисунку 14 дані в таблицю «Save notes» додались успішно.

Рисунок 13 – Введення користувачем даних які існують в таблицях «User», «Notes», «Catalog»

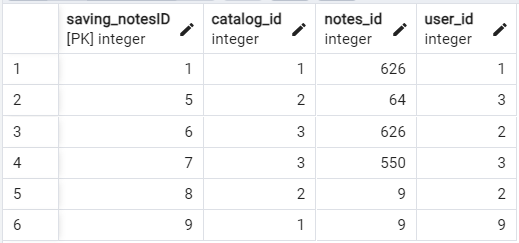
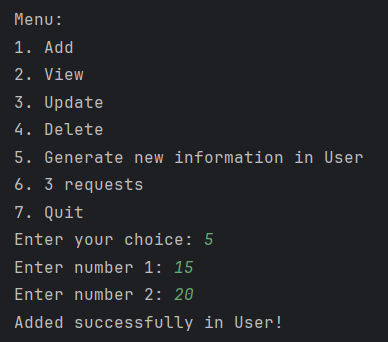


Рисунок 14 – Таблиця «Save notes», після додавання нових даних

**Пункт 2. Автоматичне генерування «рандомізованих» даних у базі**

****На рисунку 15 користувач вводить значення з якого та по яке будуть формуватись дані.

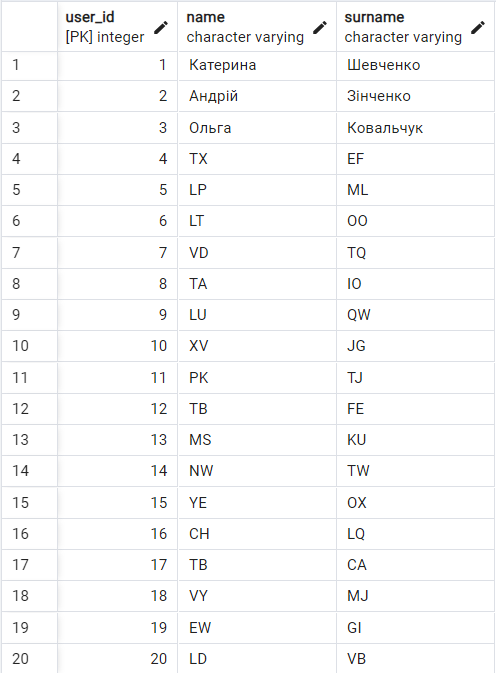
Рисунок 15 – Вибір користувачем пункту для генерації додаткових рядків

Рисунок 16 – Таблиця «User», в якій автоматично сформувались дані

**SQL-запити, які ілюструють генерацію при визначених вхідних параметрах**

c.execute('INSERT INTO "User" ("user\_id", "name", "surname") SELECT generate\_series as user\_id, chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int) as name,chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int) as surname FROM generate\_series(%s, %s)', (num1, num2))

**Пункт №3. Реалізація 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**

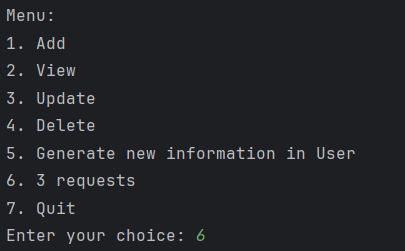
****

Рисунок 17 – Введення користувачем вибору пункту меню

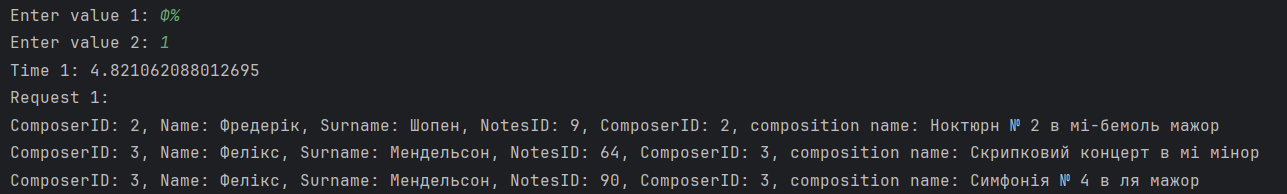


Рисунок 18 – Введення початкових параметрів для пошуку за першим запитом та вивеведення таблиці за запитом

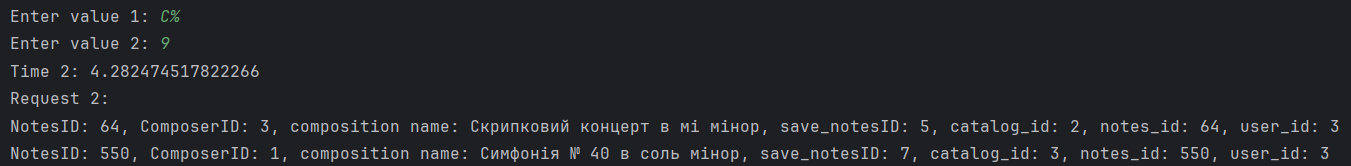


Рисунок 19 - Введення початкових параметрів для пошуку за другим запитом та вивеведення таблиці за запитом

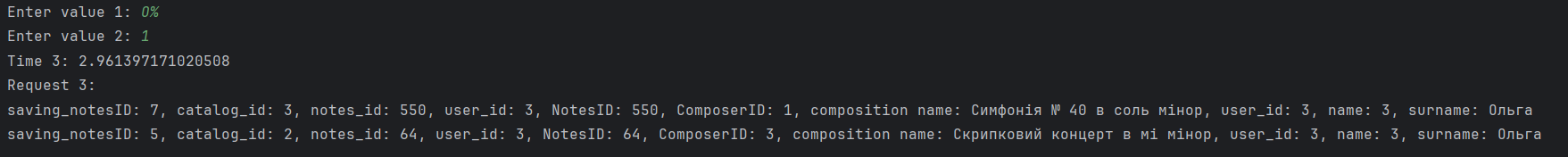


Рисунок 20 - Введення початкових параметрів для пошуку за третім запитом та вивеведення таблиці за запитом

**SQL-запити, що ілюструють генерацію при визначених вхідних параметрах**

c.execute('SELECT \* FROM "Composer" INNER JOIN "Notes" ON "Composer"."ComposerID" = "Notes"."ComposerID" WHERE "Composer"."Name" LIKE %s AND "NotesID" > %s group by "Composer"."ComposerID", "Notes"."NotesID";', (val1, val2))

c.execute('SELECT \* FROM "Notes" INNER JOIN "Saving\_notes" ON "Notes"."NotesID" = "Saving\_notes"."notes\_id" WHERE "Notes"."composition name" LIKE %s or catalog\_id > %s group by "Saving\_notes"."saving\_notesID", "Notes"."NotesID"', (val1, val2))

c.execute('SELECT \* FROM "Saving\_notes" INNER JOIN "Notes" ON "Saving\_notes"."notes\_id" = "Notes"."NotesID" INNER JOIN "User" ON "Saving\_notes"."user\_id" = "User"."user\_id" WHERE "User"."name" LIKE %s AND "catalog\_id" > %s group by "Saving\_notes"."saving\_notesID", "Notes"."NotesID", "User"."user\_id"', (val1, val2))

**Шаблон MVC (модель-подання-контролер)**

**Model**

В файлі Model ми реалізовуємо операції CRUD, тобто за допомогою бібліотеки psycopg2 записуємо SQL запити та реалізовуємо операції CRUD.

**View**

В файлі View ми записуємо введення та виведення даних.

**Controller**

За допомогою контролера ми приймаємо дані введені користувачем за допомогою методів записаних в файлі View, та використовуємо їх з методами створеними в файлі Model. Після чого виводимо результати за допомогою методів з файлу View.

**Код програми**

**main.py**

from controller import Controller

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

controller = Controller()

controller.run()

**view.py**

class View:

*#show request\_1()*

def show\_request\_1(self, requests):

if requests:

print("Request 1:")

for request in requests:

print(f"ComposerID: {request[0]}, Name: {request[1]}, Surname: {request[2]}, NotesID: {request[3]}, ComposerID: {request[4]}, composition name: {request[5]}")

else:

print("No request found.")

def show\_request\_2(self, requests):

if requests:

print("Request 2:")

for request in requests:

print(f"NotesID: {request[0]}, ComposerID: {request[1]}, composition name: {request[2]}, save\_notesID: {request[3]}, catalog\_id: {request[4]}, notes\_id: {request[5]}, user\_id: {request[6]}")

else:

print("No request found.")

def show\_request\_3(self, requests):

if requests:

print("Request 3:")

for request in requests:

print(f"Saving\_notesID: {request[0]}, CatalogID: {request[1]}, NotesID: {request[2]}, UserID: {request[3]}, NotesID: {request[4]}, ComposerID: {request[5]}, composition name: {request[6]}, user\_id: {request[7]}, name: {request[8]}, surname: {request[9]}")

else:

print("No request found.")

def input\_req1(self):

val1 = input("Enter the first letter of the composer's name: ")

val3 = val1 + "%"

val2 = input("Enter number: ")

return val3, val2

def input\_req2(self):

val1 = input("Enter the first letter of the composition: ")

val3 = val1 + "%"

val2 = input("Enter number: ")

return val3, val2

def input\_req3(self):

val1 = input("Enter the first letter of the username: ")

val3 = val1 + "%"

val2 = input("Enter number: ")

return val3, val2

*# catalog*

def show\_catalog(self, catalogs):

if catalogs:

print("catalogs:")

for catalog in catalogs:

print(f"CatalogID: {catalog[0]}, catalog name: {catalog[1]}")

else:

print("No catalog found.")

def get\_catalog\_input(self):

catal\_name = input("Enter catalog name: ")

return catal\_name

def get\_catalogID(self):

catalogID = input("Enter catalog ID: ")

return catalogID

*# user*

def show\_user(self, users):

if users:

print("Users:")

for User in users:

print(f"ID: {User[0]}, Name: {User[1]}, Surname: {User[2]}")

else:

print("No users found.")

def get\_user\_input(self):

Name = input("Enter user Name: ")

Surname = input("Enter user Surname: ")

return Name, Surname

def get\_userID(self):

userID = input("Enter user ID: ")

return userID

*# composer*

def show\_composer(self, composers):

if composers:

print("Users:")

for composer in composers:

print(f"ID: {composer[0]}, Name: {composer[1]}, Surname: {composer[2]}")

else:

print("No composer found.")

def get\_composer\_input(self):

Name = input("Enter composer Name: ")

Surname = input("Enter composer Surname: ")

return Name, Surname

def get\_composerID(self):

composerID = input("Enter composer ID: ")

return composerID

*# notes*

def show\_notes(self, notes\_1):

if notes\_1:

print("Notes:")

for notes in notes\_1:

print(f"NotesID: {notes[0]}, ComposerID: {notes[1]}, composition name: {notes[2]}")

else:

print("No notes found.")

def get\_notes\_input(self):

composer\_id = input("Enter composer ID: ")

comp\_name = input("Enter composition name: ")

return composer\_id, comp\_name

def get\_notesID(self):

notesID = input("Enter notes ID: ")

return notesID

*#save notes*

def show\_save\_notes(self, save\_notes):

if save\_notes:

print("Notes:")

for notes in save\_notes:

print(f"Saving\_notesID: {notes[0]}, ComposerID: {notes[1]}, NotesID: {notes[2]}, UserID: {notes[3]}")

else:

print("No save notes found.")

def get\_save\_notes\_input(self):

catalog\_id = input("Enter catalog ID: ")

notes\_id = input("Enter notes ID: ")

user\_id = input("Enter user ID: ")

return catalog\_id, notes\_id, user\_id

def get\_save\_notesID(self):

save\_notesID = input("Enter saving notes ID: ")

return save\_notesID

def input\_num(self):

num1 = int(input("Enter number 1: "))

num2 = int(input("Enter number 2: "))

return num1, num2

*# output*

def show\_message(self, message):

print(message)

**model.py**

import psycopg2

import time

class Model:

def \_\_init\_\_(self):

self.conn = psycopg2.connect(

dbname='Deanery',

user='postgres',

password='qwerty',

host='localhost',

port=5432

)

def gener\_add\_user(self, num1, num2):

c = self.conn.cursor()

num = 0

for i in range(num1, num2+1):

c.execute('SELECT \* FROM "User" WHERE "UserID" = %s', (i,))

check = c.fetchall()

if check:

print("UserID %s already exists", i)

num = 1

if num == 0:

c.execute('INSERT INTO "User" ("UserID", "Name", "Surname") SELECT generate\_series as UserID, chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int) as Name,chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int) as Surname FROM generate\_series(%s, %s)', (num1, num2))

self.conn.commit()

else:

print("Error! Identifiers already exist")

*#3 requests*

def request\_1(self, val1, val2):

c = self.conn.cursor()

start\_t = time.time()

c.execute('SELECT \* FROM "Composer" INNER JOIN "Notes" ON "Composer"."ComposerID" = "Notes"."ComposerID" WHERE "Composer"."Name" LIKE %s AND "NotesID" > %s group by "Composer"."ComposerID", "Notes"."NotesID";', (val1, val2))

end\_t = time.time()

time\_t = (end\_t - start\_t) \* 1000

print("Time 1:", time\_t)

return c.fetchall()

def request\_2(self, val1, val2):

c = self.conn.cursor()

start\_t = time.time()

c.execute('SELECT \* FROM "Notes" INNER JOIN "Saving\_notes" ON "Notes"."NotesID" = "Saving\_notes"."NotesID" WHERE "Notes"."composition name" LIKE %s OR "Saving\_notes"."CatalogID" > %s group by "Saving\_notes"."Saving\_notesID", "Notes"."NotesID"', (val1, val2))

end\_t = time.time()

time\_t = (end\_t - start\_t) \* 1000

print("Time 2:", time\_t)

return c.fetchall()

def request\_3(self, val1, val2):

c = self.conn.cursor()

start\_t = time.time()

c.execute('SELECT \* FROM "Saving\_notes" INNER JOIN "Notes" ON "Saving\_notes"."NotesID" = "Notes"."NotesID" INNER JOIN "User" ON "Saving\_notes"."UserID" = "User"."UserID" WHERE "User"."Name" LIKE %s AND "Saving\_notes"."CatalogID" > %s group by "Saving\_notes"."Saving\_notesID", "Notes"."NotesID", "User"."UserID"', (val1, val2))

end\_t = time.time()

time\_t = (end\_t - start\_t) \* 1000

print("Time 3:", time\_t)

return c.fetchall()

*# for catalog*

def add\_catalog(self, catalog\_id, name):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Catalog" WHERE "CatalogID" = %s', (catalog\_id,))

check = c.fetchall()

if check:

print("Error! This identifier already exists")

else:

c.execute('INSERT INTO "Catalog" ("CatalogID", "catalog\_name") VALUES (%s, %s)', (catalog\_id, name))

self.conn.commit()

print("Catalog added successfully!")

def update\_catalog(self, catalog\_id, catal\_name):

c = self.conn.cursor()

c.execute('UPDATE "Catalog" SET "catalog\_name"=%s WHERE "CatalogID"=%s', (catal\_name, catalog\_id))

self.conn.commit()

print("Catalog updated successfully!")

def delete\_catalog(self, catalog\_id):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Catalog" WHERE "CatalogID" = %s', (catalog\_id,))

check1 = c.fetchall()

c.execute('SELECT \* FROM "Saving\_notes" WHERE "CatalogID" = %s', (catalog\_id,))

check2 = c.fetchall()

if check1 and check2:

c.execute('DELETE FROM "Saving\_notes" WHERE "CatalogID"=%s', (catalog\_id,))

self.conn.commit()

c.execute('DELETE FROM "Catalog" WHERE "CatalogID"=%s', (catalog\_id,))

self.conn.commit()

print("Catalog deleted successfully!")

elif check1:

c.execute('DELETE FROM "Catalog" WHERE "CatalogID"=%s', (catalog\_id,))

self.conn.commit()

print("Catalog deleted successfully!")

else:

print("Error! This CatalogID not exist")

def get\_all\_catalog(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Catalog"')

return c.fetchall()

*# for user*

def add\_user(self, user\_id, Name, Surname):

c = self.conn.cursor()

c.execute('SELECT \* FROM "User" WHERE "UserID" = %s', (user\_id,))

check = c.fetchall()

if check:

print("Error! This identifier already exists")

else:

c.execute('INSERT INTO "User" ("UserID","Name", "Surname") VALUES (%s, %s, %s)', (user\_id, Name, Surname))

self.conn.commit()

print("User added successfully!")

def get\_all\_user(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM "User"')

return c.fetchall()

def update\_user(self, user\_id, Name, Surname):

c = self.conn.cursor()

c.execute('UPDATE "User" SET "Name"=%s, "Surname"=%s WHERE "UserID"=%s', (Name, Surname, user\_id))

self.conn.commit()

def delete\_user(self, user\_id):

c = self.conn.cursor()

c.execute('SELECT \* FROM "User" WHERE "UserID" = %s', (user\_id,))

check1 = c.fetchall()

c.execute('SELECT \* FROM "Saving\_notes" WHERE "UserID" = %s', (user\_id,))

check2 = c.fetchall()

if check1 and check2:

c.execute('DELETE FROM "Saving\_notes" WHERE "UserID"=%s', (user\_id,))

self.conn.commit()

c.execute('DELETE FROM "User" WHERE "UserID"=%s', (user\_id,))

self.conn.commit()

print("User deleted successfully!")

elif check1:

c.execute('DELETE FROM "User" WHERE "UserID"=%s', (user\_id,))

self.conn.commit()

print("User deleted successfully!")

else:

print("Error! This UserID not exist")

*# for composer*

def add\_composer(self, composer\_id, Name, Surname):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Composer" WHERE "ComposerID" = %s', (composer\_id,))

check = c.fetchall()

if check:

print("Error! This identifier already exists")

else:

c.execute('INSERT INTO "Composer" ("ComposerID", "Name", "Surname") VALUES (%s, %s, %s)',(composer\_id, Name, Surname))

self.conn.commit()

print("Composer added successfully!")

def get\_all\_composer(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Composer"')

return c.fetchall()

def update\_composer(self, composer\_id, Name, Surname):

c = self.conn.cursor()

c.execute('UPDATE "Composer" SET "Name"=%s, "Surname"=%s WHERE "ComposerID"=%s', (Name, Surname, composer\_id))

self.conn.commit()

print("Composer updated successfully!")

def delete\_composer(self, composer\_id):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Composer" WHERE "ComposerID" = %s', (composer\_id,))

check1 = c.fetchall()

c.execute('SELECT \* FROM "Notes" WHERE "ComposerID" = %s', (composer\_id,))

check2 = c.fetchall()

if check1 and check2:

c.execute('SELECT "NotesID" FROM "Notes" WHERE "ComposerID" = %s', (composer\_id,))

n = c.fetchall()

c.execute('DELETE FROM "Saving\_notes" WHERE "NotesID"= %s', (n[0],))

self.conn.commit()

c.execute('DELETE FROM "Notes" WHERE "ComposerID"=%s', (composer\_id,))

self.conn.commit()

c.execute('DELETE FROM "Composer" WHERE "ComposerID"=%s', (composer\_id,))

self.conn.commit()

print("Composer deleted successfully!")

elif check1:

c.execute('DELETE FROM "Composer" WHERE "ComposerID"=%s', (composer\_id,))

self.conn.commit()

print("Composer deleted successfully!")

else:

print("Error! This ComposerID not exist")

*# for notes*

def add\_notes(self, notes\_id, composer\_id, comp\_name):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Notes" WHERE "NotesID" = %s', (notes\_id,))

check2 = c.fetchall()

if check2:

print("Error! This identifier NotesID already exists")

else:

c.execute('SELECT \* FROM "Composer" WHERE "ComposerID" = %s', (composer\_id,))

check1 = c.fetchall()

if check1:

c.execute('INSERT INTO "Notes" ("NotesID","ComposerID", "composition name") VALUES (%s, %s, %s)',(notes\_id, composer\_id, comp\_name))

self.conn.commit()

print("Notes added successfully!")

else:

print("Error! This identifier ComposerID already exists")

def get\_all\_notes(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Notes"')

return c.fetchall()

def update\_notes(self, notes\_id, composer\_id, comp\_name):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Composer" WHERE "ComposerID" = %s', (composer\_id,))

check = c.fetchall()

if check:

c.execute('UPDATE "Notes" SET "ComposerID"=%s, "composition name"=%s WHERE "NotesID"=%s',(composer\_id, comp\_name, notes\_id))

self.conn.commit()

print("Notes updated successfully!")

else:

print("Error! The ID that was changed is not in the parent table")

def delete\_notes(self, notes\_id):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Notes" WHERE "NotesID" = %s', (notes\_id,))

check1 = c.fetchall()

c.execute('SELECT \* FROM "Notes" WHERE "NotesID" = %s', (notes\_id,))

check2 = c.fetchall()

if check1 and check2:

c.execute('DELETE FROM "Saving\_notes" WHERE "NotesID"=%s', (notes\_id,))

self.conn.commit()

c.execute('DELETE FROM "Notes" WHERE "NotesID"=%s', (notes\_id,))

self.conn.commit()

print("Notes deleted successfully!")

elif check1:

c.execute('DELETE FROM "Notes" WHERE "NotesID"=%s', (notes\_id,))

self.conn.commit()

print("Notes deleted successfully!")

else:

print("Error! This NotesID not exist")

*#save notes*

def add\_save\_notes(self, save\_notes\_id, catalog\_id, notes\_id, user\_id):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Saving\_notes" WHERE "Saving\_notesID" = %s', (save\_notes\_id,))

check = c.fetchall()

if check:

print("Error! This identifier already exists")

else:

c.execute('SELECT \* FROM "Catalog" WHERE "CatalogID" = %s', (catalog\_id,))

check1 = c.fetchall()

c.execute('SELECT \* FROM "Notes" WHERE "NotesID" = %s', (notes\_id,))

check2 = c.fetchall()

c.execute('SELECT \* FROM "User" WHERE "UserID" = %s', (user\_id,))

check3 = c.fetchall()

if check1 and check2 and check3:

c.execute('INSERT INTO "Saving\_notes" ("Saving\_notesID","CatalogID","NotesID", "UserID") VALUES (%s, %s, %s, %s)',(save\_notes\_id, catalog\_id, notes\_id, user\_id))

self.conn.commit()

print("Notes added successfully!")

else:

print("Error! This identifier already exists")

def get\_all\_save\_notes(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Saving\_notes"')

return c.fetchall()

def update\_save\_notes(self, catalog\_id, notes\_id, user\_id, save\_notes\_id):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Catalog" WHERE "CatalogID" = %s', (catalog\_id,))

check1 = c.fetchall()

c.execute('SELECT \* FROM "Notes" WHERE "NotesID" = %s', (notes\_id,))

check2 = c.fetchall()

c.execute('SELECT \* FROM "User" WHERE "UserID" = %s', (user\_id,))

check3 = c.fetchall()

if check1 and check2 and check3:

c.execute('UPDATE "Saving\_notes" SET "CatalogID"=%s, "NotesID"=%s, "UserID"=%s WHERE "Saving\_notesID"=%s',(catalog\_id, notes\_id, user\_id, save\_notes\_id))

self.conn.commit()

print("Saving notes updated successfully!")

else:

print("Error! The ID that was changed is not in the parent table")

def delete\_save\_notes(self, save\_notes\_id):

c = self.conn.cursor()

c.execute('SELECT \* FROM "Saving\_notes" WHERE "Saving\_notesID" = %s', (save\_notes\_id,))

check = c.fetchall()

if check:

c.execute('DELETE FROM "Saving\_notes" WHERE "Saving\_notesID"=%s', (save\_notes\_id,))

self.conn.commit()

print("Saving notes deleted successfully!")

else:

print("Error! Saving notes ID not exist")

**controller.py**

from model import Model

from view import View

class Controller:

def \_\_init\_\_(self):

self.model = Model()

self.view = View()

def run(self):

while True:

category = self.show\_menu()

if category == '1':

self.run\_add()

elif category == '2':

self.run\_view()

elif category == '3':

self.run\_update()

elif category == '4':

self.run\_delete()

elif category == '5':

self.gener\_user()

elif category == '6':

self.requests()

elif category == '7':

break

def show\_menu(self):

self.view.show\_message("\nMenu:")

self.view.show\_message("1. Add")

self.view.show\_message("2. View")

self.view.show\_message("3. Update")

self.view.show\_message("4. Delete")

self.view.show\_message("5. Generate new information in User")

self.view.show\_message("6. 3 requests")

self.view.show\_message("7. Quit")

return input("Enter your choice: ")

def menu(self):

self.view.show\_message("1. Catalog")

self.view.show\_message("2. Notes")

self.view.show\_message("3. Composer")

self.view.show\_message("4. User")

self.view.show\_message("5. Save notes")

self.view.show\_message("6. Quit")

return input("Enter your choice: ")

*# run menu*

def run\_add(self):

self.view.show\_message("\nAdd:")

while True:

category1 = self.menu()

if category1 == '1':

self.add\_catalog()

elif category1 == '2':

self.add\_notes()

elif category1 == '3':

self.add\_composer()

elif category1 == '4':

self.add\_user()

elif category1 == '5':

self.add\_save\_notes()

elif category1 == '6':

break

def run\_view(self):

self.view.show\_message("\nView:")

while True:

category2 = self.menu()

if category2 == '1':

self.view\_catalog()

elif category2 == '2':

self.view\_notes()

elif category2 == '3':

self.view\_composer()

elif category2 == '4':

self.view\_user()

elif category2 == '5':

self.view\_save\_notes()

elif category2 == '6':

break

def run\_update(self):

self.view.show\_message("\nUpdate:")

while True:

category3 = self.menu()

if category3 == '1':

self.update\_catalog()

elif category3 == '2':

self.update\_notes()

elif category3 == '3':

self.update\_composer()

elif category3 == '4':

self.update\_user()

elif category3 == '5':

self.update\_save\_notes()

elif category3 == '6':

break

def run\_delete(self):

self.view.show\_message("\nDelete:")

while True:

category4 = self.menu()

if category4 == '1':

self.delete\_catalog()

elif category4 == '2':

self.delete\_notes()

elif category4 == '3':

self.delete\_composer()

elif category4 == '4':

self.delete\_user()

elif category4 == '5':

self.delete\_save\_notes()

elif category4 == '6':

break

def gener\_user(self):

num1, num2 = self.view.input\_num()

self.model.gener\_add\_user(num1, num2)

print("Added successfully in User!")

def requests(self):

val1, val2 = self.view.input\_req1()

if val2.isdigit():

request\_1 = self.model.request\_1(val1, val2)

self.view.show\_request\_1(request\_1)

else:

print("Error! you entered an incorrect type.")

val3, val4 = self.view.input\_req2()

if val4.isdigit():

request\_2 = self.model.request\_2(val3, val4)

self.view.show\_request\_2(request\_2)

else:

print("Error! you entered an incorrect type.")

val5, val6 = self.view.input\_req3()

if val6.isdigit():

request\_3 = self.model.request\_3(val5, val6)

self.view.show\_request\_3(request\_3)

else:

print("Error! you entered an incorrect type.")

*# func controller*

*#save notes*

def add\_save\_notes(self):

save\_notes\_id = self.view.get\_save\_notesID()

catalog\_id, notes\_id, user\_id = self.view.get\_save\_notes\_input()

if save\_notes\_id.isdigit() and catalog\_id.isdigit() and notes\_id.isdigit() and user\_id.isdigit():

self.model.add\_save\_notes(save\_notes\_id, catalog\_id, notes\_id, user\_id)

else:

print("Error! you entered an incorrect id.")

def view\_save\_notes(self):

save\_notes = self.model.get\_all\_save\_notes()

self.view.show\_save\_notes(save\_notes)

def update\_save\_notes(self):

save\_notes\_id = self.view.get\_save\_notesID()

catalog\_id, notes\_id, user\_id = self.view.get\_save\_notes\_input()

if save\_notes\_id.isdigit() and catalog\_id.isdigit() and notes\_id.isdigit() and user\_id.isdigit():

self.model.update\_save\_notes(catalog\_id, notes\_id, user\_id, save\_notes\_id)

else:

print("Error! you entered an incorrect id.")

def delete\_save\_notes(self):

save\_notes\_id = self.view.get\_save\_notesID()

if save\_notes\_id.isdigit():

self.model.delete\_save\_notes(save\_notes\_id)

else:

print("Error! you entered an incorrect id.")

*# catalog*

def add\_catalog(self):

catalog\_name = self.view.get\_catalog\_input()

catalog\_id = self.view.get\_catalogID()

if catalog\_id.isdigit():

self.model.add\_catalog(catalog\_id, catalog\_name)

else:

print("Error! you entered an incorrect type.")

def view\_catalog(self):

catalog = self.model.get\_all\_catalog()

self.view.show\_catalog(catalog)

def update\_catalog(self):

catalog\_id = self.view.get\_catalogID()

catalog\_name = self.view.get\_catalog\_input()

if catalog\_id.isdigit():

self.model.update\_catalog(catalog\_id, catalog\_name)

else:

print("Error! you entered an incorrect type.")

def delete\_catalog(self):

catalog\_id = self.view.get\_catalogID()

if catalog\_id.isdigit():

self.model.delete\_catalog(catalog\_id)

else:

print("Error! you entered an incorrect type.")

*# notes*

def add\_notes(self):

composer\_id, comp\_name = self.view.get\_notes\_input()

notes\_id = self.view.get\_notesID()

if notes\_id.isdigit() and composer\_id.isdigit():

self.model.add\_notes(notes\_id, composer\_id, comp\_name)

else:

print("Error! you entered an incorrect type.")

def view\_notes(self):

notes = self.model.get\_all\_notes()

self.view.show\_notes(notes)

def update\_notes(self):

notes\_id = self.view.get\_notesID()

composer\_id, comp\_name = self.view.get\_notes\_input()

if notes\_id.isdigit() and composer\_id.isdigit():

self.model.update\_notes(notes\_id, composer\_id, comp\_name)

else:

print("Error! you entered an incorrect type.")

def delete\_notes(self):

notes\_id = self.view.get\_notesID()

if notes\_id.isdigit():

self.model.delete\_notes(notes\_id)

else:

print("Error! you entered an incorrect type.")

*# user*

def add\_user(self):

name, surname = self.view.get\_user\_input()

user\_id = self.view.get\_userID()

if user\_id.isdigit():

self.model.add\_user(user\_id, name, surname)

else:

print("Error! you entered an incorrect type.")

def view\_user(self):

users = self.model.get\_all\_user()

self.view.show\_user(users)

def update\_user(self):

user\_id = self.view.get\_userID()

name, surname = self.view.get\_user\_input()

if user\_id.isdigit() and name.isalpha() and surname.isalpha():

self.model.update\_user(user\_id, name, surname)

self.view.show\_message("User updated successfully!")

else:

print("Error! you entered an incorrect type.")

def delete\_user(self):

user\_id = self.view.get\_userID()

if user\_id.isdigit():

self.model.delete\_user(user\_id)

else:

print("Error! you entered an incorrect type.")

*# composer*

def add\_composer(self):

name, surname = self.view.get\_composer\_input()

composer\_id = self.view.get\_composerID()

if composer\_id.isdigit() and name.isalpha() and surname.isalpha():

self.model.add\_composer(composer\_id, name, surname)

else:

print("Error! you entered an incorrect type.")

def view\_composer(self):

composers = self.model.get\_all\_composer()

self.view.show\_composer(composers)

def update\_composer(self):

composer\_id = self.view.get\_composerID()

name, surname = self.view.get\_composer\_input()

if composer\_id.isdigit() and name.isalpha() and surname.isalpha():

self.model.update\_composer(composer\_id, name, surname)

else:

print("Error! you entered an incorrect type.")

def delete\_composer(self):

composer\_id = self.view.get\_composerID()

if composer\_id.isdigit():

self.model.delete\_composer(composer\_id)

else:

print("Error! you entered an incorrect type.")

Telegram: <https://t.me/lumi_789>