

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Лабораторна робота №3**

з дисципліни

**«Бази даних і засоби управління»**

Тема: «***Засоби оптимізації роботи СУБД PostgreSQL***»

Виконав: студент 3 курсу

ФПМ групи КВ-82

Бодашевський Д.Р.

Перевірив: Павловський В.І.

Київ – 2020

**Лабораторна робота № 3.**

**Засоби оптимізації роботи СУБД PostgreSQL**

***Мета роботи:*** здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

***Завдання:***

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.

**Варіант 4**

У другому завданні проаналізувати індекси *GIN, BRIN*

Умова для тригера – *after delete, insert*

***Зміст звіту:***

1. *Логічна модель бази даних;*
2. *Засоби програмування додатку, спосіб доступу до БД;*
3. *Ілюстрація роботи додатку та відповіді на питання.*
4. *Код програми*

**Логічна модель бази даних**

Опис бази даних:

База даних репрезентує предметну область «Інтернет-аукціон картин». Основні сутності:

* Автор Картини
* Клієнт
* Профіль клієнта
* Номер мобільного
* Картина

|  |  |  |
| --- | --- | --- |
| **ВІДНОШЕННЯ** | **АТРИБУТ** | **ТИП(Розмір)** |
| Відношення *“Author”*  Вміщує інформацію про автора | A\_Id – унікальний ID автора в БД  Full\_name – Ім’я та призвіще автора  Paintings\_count – Кількість картин автора | Числовий  Текстовий  Числовий |
| Відношення *“Client”*  Вміщує інформацію про клієнта | C\_Id - унікальний ID клієнта в БД  Name – ім’я клієнта  Surname – призвіще клієнта  Age – вік клієнта | Числовий  Текстовий  Текстовий  Числовий |
| Відношення *“Profile”*  Вміщує інформацію про профіль клієнта інтернет-аукціону | Pr\_Id – унікальний ID профіля в БД  e-mail – e-mail клієнта  Paintings\_count – кількість куплених картин  Registration\_date – дата реєстрації профілю | Числовий  Текстовий  Числовий  Дата |
| Відношення *“Painting”*  Вміщує в собі інформацію про картину | P\_Id – унікальний ID картини в БД  Author – автор картини  Genre – жанр картини  Price – ціна картини  Date– дата створення картини  Owner\_id – ID власника картини | Числовий  Текстовий  Текстовий  Грошовий  Дата  Числовий |
| Відношення “*Phone number*”  Вміщує в собі інформацію про мобільний номер клієнта | Number – мобільний номер  Сl\_Id – ID клієнта, якому належить мобільний номер | Текст  Числовий |

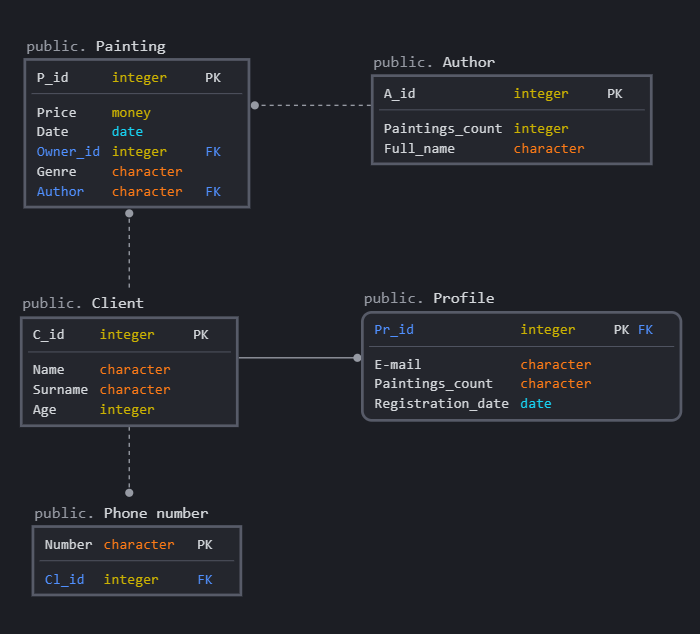
****

Рисунок 1 — Логічна модель предметної області “Інтернет-аукціон картин”.

**Засоби програмування і спосіб доступу до БД**

Мова програмування: python;

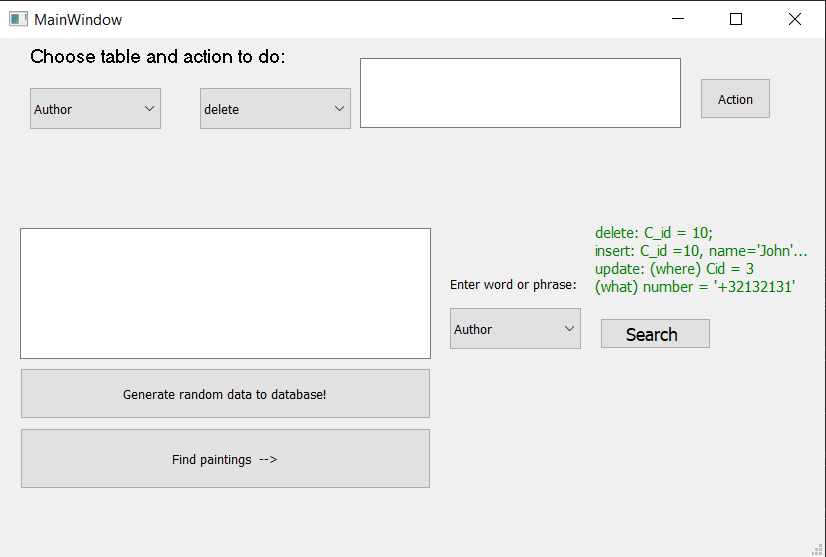
Середовище програмування (IDE): PyCharm;

Засіб для розробки графічного інтерфейсу: QtDesigner, pyuic5;

Спосіб доступу до БД: бібліотека SQLAlchemy.

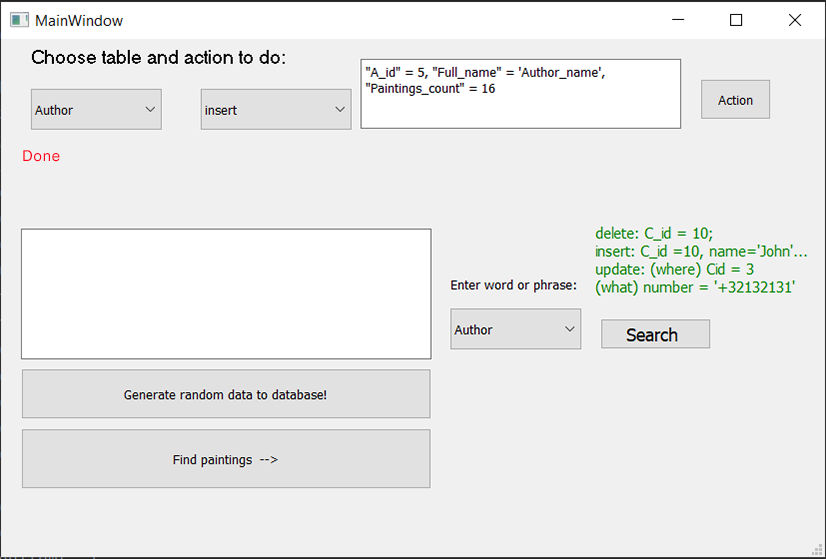
**Ілюстрація роботи додатку та відповіді на питання**

Головне меню додатку:

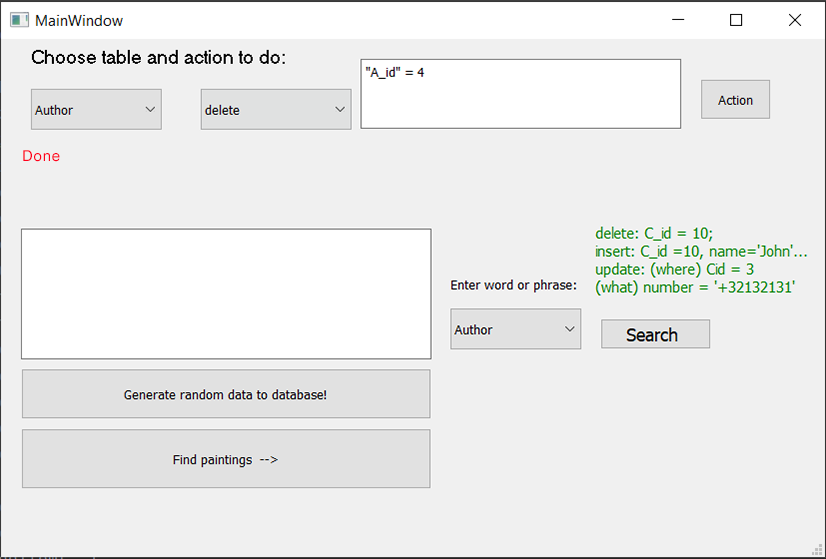


Формати запитів:

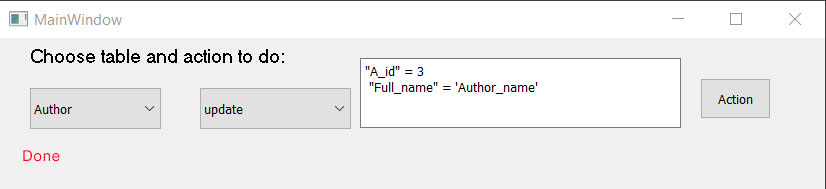
* На введення:



* На видалення:



* На оновлення:



**Завдання 1**

Код опису класів засобами SQLAlchemy:

class Author(DeclarativeBase):  
 \_\_tablename\_\_ = 'Author'  
  
 Full\_name = Column('Full\_name', String(30))  
 A\_id = Column('A\_id', Integer,primary\_key=True)  
 Paintings\_count = Column('Paintings\_count', Integer)  
  
  
 class Painting(DeclarativeBase):  
 \_\_tablename\_\_ = 'Painting'  
  
 P\_id = Column('P\_id', Integer, primary\_key=True)  
 Price = Column('Price', Integer)  
 Date = Column('Date', DateTime)  
 Owner\_id = Column("Owner\_id", Integer,ForeignKey("Client.C\_id"))  
 Genre = Column("Genre", String(15))  
 Author = Column("Author", String(30),ForeignKey("Author.Full\_name"))  
  
  
 class Client(DeclarativeBase):  
 \_\_tablename\_\_ = "Client"  
  
 Name = Column("Name", String(30))  
 C\_id = Column("C\_id", Integer, primary\_key=True)  
 Surnamename = Column("Surname", String(30))  
 Age = Column("Age", Integer)  
  
  
 class Phonenumber(DeclarativeBase):  
 \_\_tablename\_\_ = "Phone number"  
  
 Cl\_id = Column("Cl\_id", Integer,ForeignKey("Client.C\_id") )  
 Number = Column("Number", String(15) ,primary\_key=True)  
  
  
 class Profile(DeclarativeBase):  
 \_\_tablename\_\_ = "Profile"  
  
 Pr\_id = Column("Pr\_id", Integer, ForeignKey("Client.C\_id"),primary\_key=True)  
 Email = Column("E-mail", String(30))  
 Paintings\_count = Column("Paintings\_count", String(30))  
 Registration\_date = Column("Registration\_date", DateTime)

Код запиту на видалення даних:

def delete\_request(self, table, where):  
temp = Table(table, self.metadata, autoload=True, autoload\_with=self.engine)  
 query = delete(temp).where(text(str(where)))  
 results = self.session.execute(query)  
 results = self.session.execute(select([temp])).fetchall()  
 output('output.txt', results)  
 self.session.commit()

Код запиту на вставку даних:  
def insert\_request(self, table, condition):  
 temp = Table(table, self.metadata, autoload=True, autoload\_with=self.engine)  
 res = eval('dict(' + condition + ')')  
 query = insert(temp)  
 ResultProxy = self.session.execute(query, res)  
 results = self.session.execute(select([temp])).fetchall()  
 output('output.txt', results)  
 self.session.commit()

Код запиту на оновлення даних:  
def update\_request(self, table, condition):  
 temp = Table(table, self.metadata, autoload=True, autoload\_with=self.engine)  
 where, what = condition.split(',')  
 res = eval('dict(' + what + ')')  
 query = update(temp).values(res).where(text(where))  
 results = self.session.execute(query)  
 results = self.session.execute(select([temp])).fetchall()  
 output('output.txt', results)  
 self.session.commit()

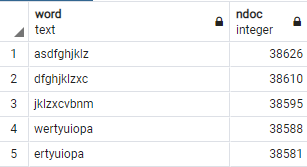
**Завдання 2**

GIN

Для дослідження індексу була створена таблиця test\_gin, яка має дві колонки типу text та tsvector. Колонка з типом tsvector проіндексована як GIN. У таблицю було занесено 1000000 записів.

Знайдемо, які слова зустрічаються найчастіше:

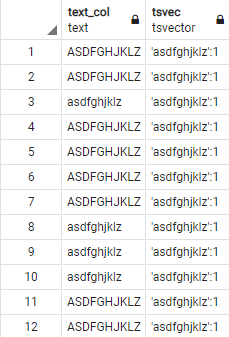




Проведемо пошук по першому слову:



Результат:



Для неіндексованої таблиці:



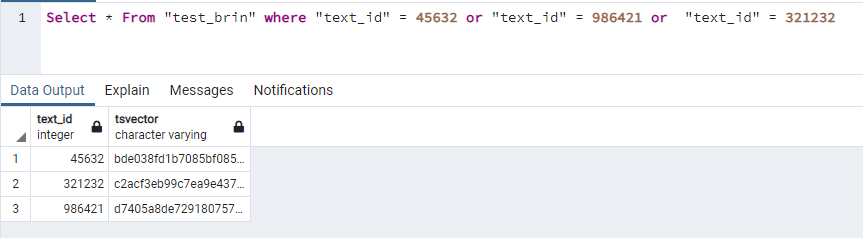
Для проіндексованої таблиці:



BRIN

Для дослідження індексу була створена таблиця test\_brin, яка має дві колонки типу integer та character varying. Колонка з типом integer проіндексована як BRIN. У таблицю було занесено 1000000 записів.

Виконаємо тестовий запит на пошук записів за значення поля text\_id.



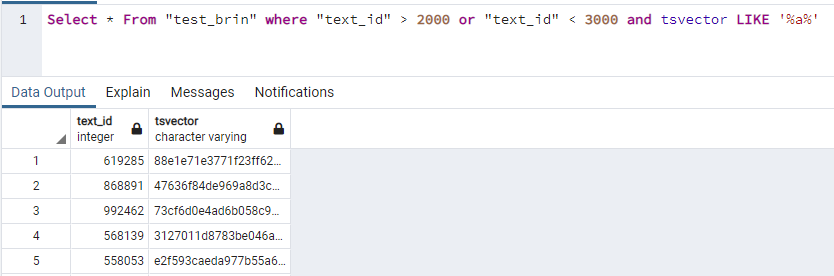
Для неіндексованої таблиці:



Для проіндексованої таблиці:



Запит, з використанням предикату над індексом:



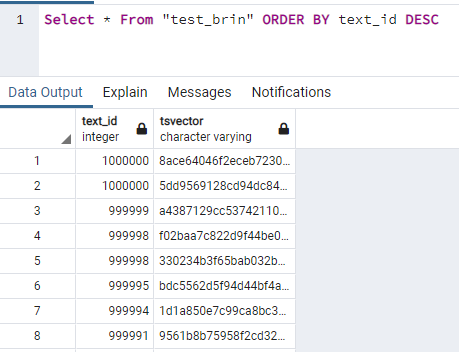
Для неіндексованої таблиці:



Для проіндексованої таблиці:



Запит, з використанням сортування, яке потребує багато порівнянь об’єктів:



Для неіндексованої таблиці:

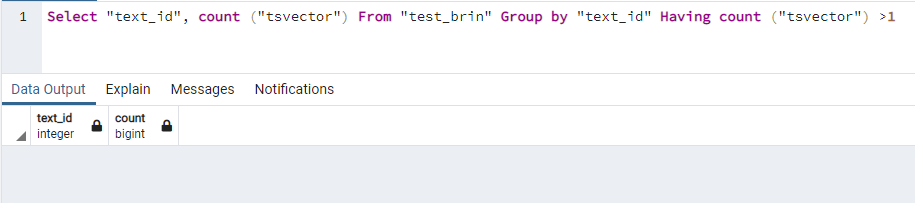


Для проіндексованої таблиці:



.

Запит пошуку користувачів з однаковим полем text\_id:



Для неіндексованої таблиці:



Для проіндексованої таблиці:

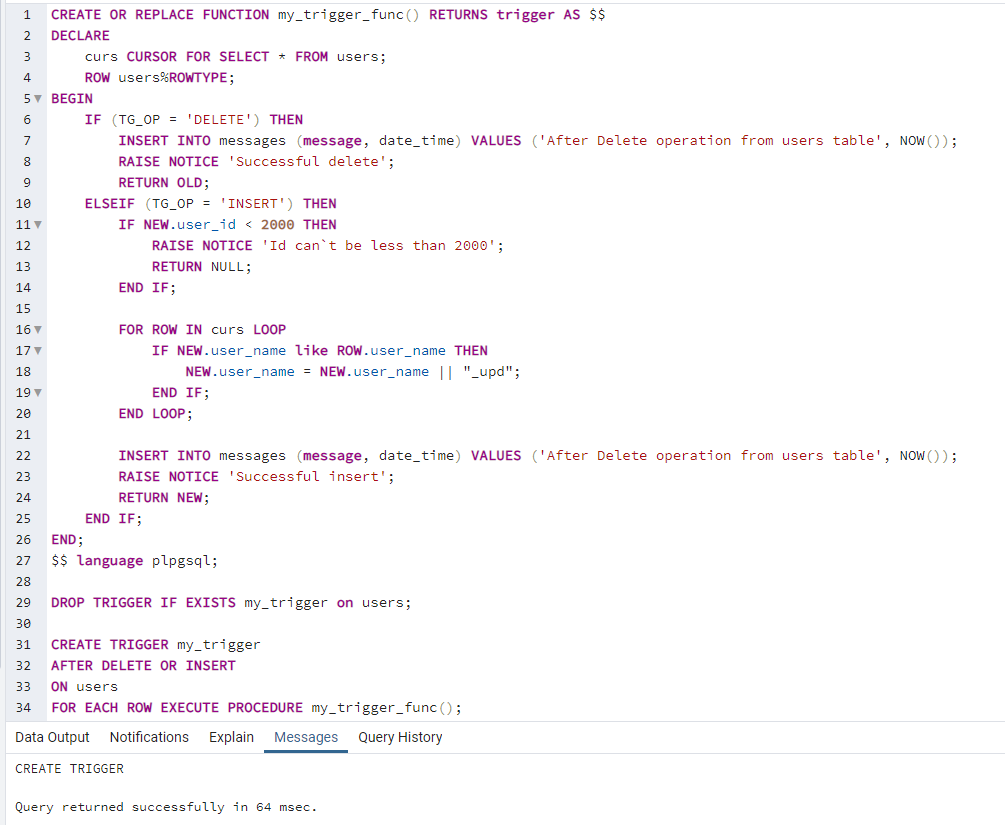


**Завдання 3**

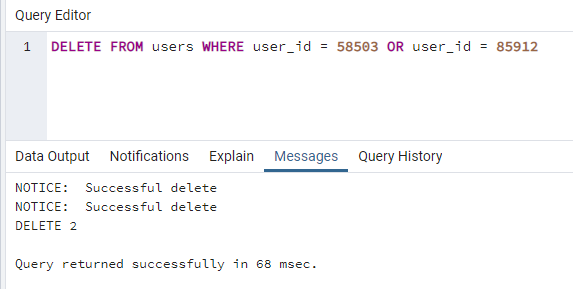
Для тестування тригерів створимо таблицю, для зберігання повідомлень від тригеру.



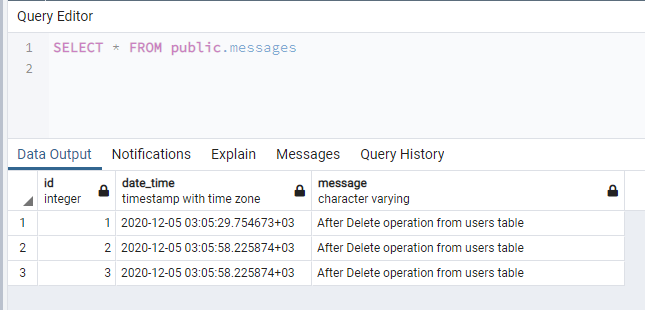
Команда створення тригеру:



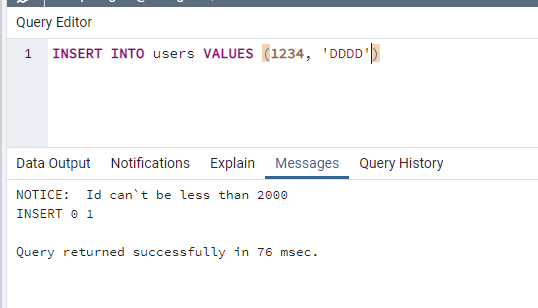
Виконаємо запит видалення з таблиці:



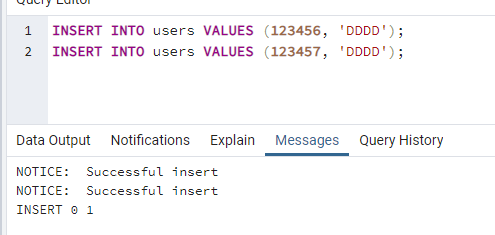
Бачимо повідомлення від тригеру. Зміст таблиці Messages також був змінений:



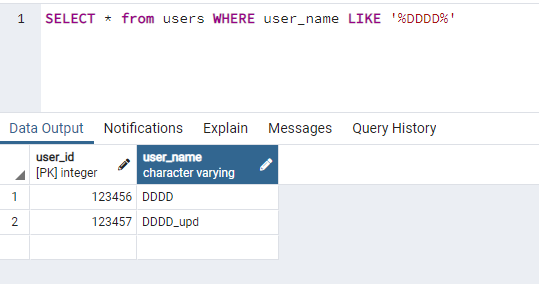
Виконаємо запит додавання даних:



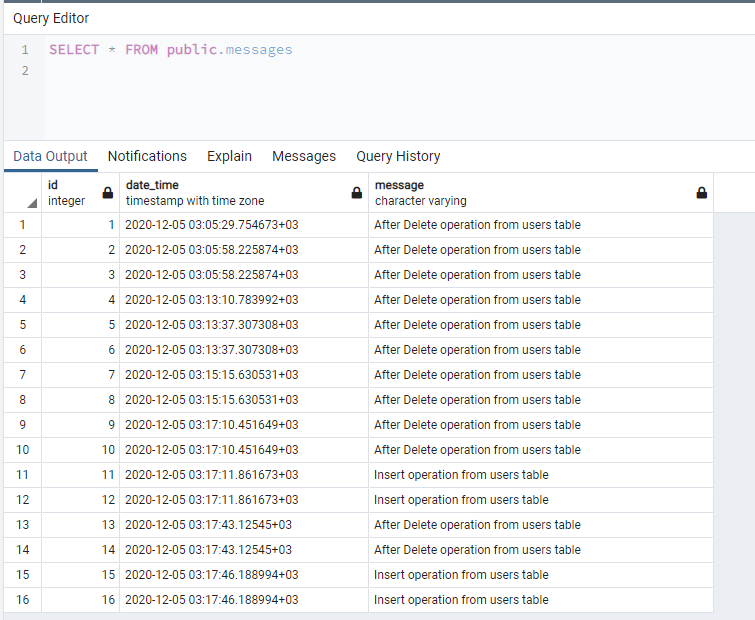
Тригер повернув «Помилку» через те, що ми не виконали його умову додавання даних. Виконаємо запит, з збільшенням індексів:



Бачимо повідомлення від тригеру. Зміст таблиці users був змінений наступним чином:



Зміст таблиці Logs також був змінений:



**Код програми**

Файл model.py:

from sqlalchemy import Table, Column, create\_engine, insert, delete, text, select, update  
from sqlalchemy.orm import relationship  
from sqlalchemy import Integer, ForeignKey, String, Unicode, Numeric, Boolean, DateTime, TIMESTAMP, MetaData  
from sqlalchemy.ext.declarative import declarative\_base  
from sqlalchemy.orm import backref, relation  
from sqlalchemy.exc import ArgumentError  
from sqlalchemy.ext.automap import automap\_base  
from sqlalchemy.orm import sessionmaker  
import json  
import random  
import string  
import datetime  
  
DeclarativeBase = declarative\_base()  
metadata = DeclarativeBase.metadata  
  
  
class Author(DeclarativeBase):  
 \_\_tablename\_\_ = 'Author'  
  
 Full\_name = Column('Full\_name', String(30))  
 A\_id = Column('A\_id', Integer,primary\_key=True)  
 Paintings\_count = Column('Paintings\_count', Integer)  
  
  
 class Painting(DeclarativeBase):  
 \_\_tablename\_\_ = 'Painting'  
  
 P\_id = Column('P\_id', Integer, primary\_key=True)  
 Price = Column('Price', Integer)  
 Date = Column('Date', TIMESTAMP)  
 Owner\_id = Column("Owner\_id", Integer,ForeignKey("Client.C\_id"))  
 Genre = Column("Genre", String(15))  
 Author = Column("Author", String(30),ForeignKey("Author.Full\_name"))  
  
  
 class Client(DeclarativeBase):  
 \_\_tablename\_\_ = "Client"  
  
 Name = Column("Name", String(30))  
 C\_id = Column("C\_id", Integer, primary\_key=True)  
 Surnamename = Column("Surname", String(30))  
 Age = Column("Age", Integer)  
  
  
 class Phonenumber(DeclarativeBase):  
 \_\_tablename\_\_ = "Phone number"  
  
 Cl\_id = Column("Cl\_id", Integer,ForeignKey("Client.C\_id") )  
 Number = Column("Number", String(15) ,primary\_key=True)  
  
  
 class Profile(DeclarativeBase):  
 \_\_tablename\_\_ = "Profile"  
  
 Pr\_id = Column("Pr\_id", Integer, ForeignKey("Client.C\_id"),primary\_key=True)  
 Email = Column("E-mail", String(30))  
 Paintings\_count = Column("Paintings\_count", String(30))  
 Registration\_date = Column("Registration\_date", TIMESTAMP)  
  
  
  
 DeclarativeBase.metadata.create\_all()  
  
  
def randomString(stringLength=8, flag=True):  
 *"""Generate a random string with the combination of lowercase and uppercase letters """* if flag:  
 letters = string.ascii\_letters + ' '  
 else:  
 letters = string.digits  
 return ''.join(random.choice(letters) for i in range(stringLength))  
  
def output(filename, data):  
 with open(filename, 'w+') as f:  
 f.writelines("%s\n" % place for place in data)  
  
  
class Database:  
  
 def \_\_init\_\_(self):  
 try:  
 self.engine = create\_engine('postgresql://postgres:6969@localhost:5433/postgres')  
 self.metadata = MetaData() #DeclarativeBase.metadata  
 self.metadata.reflect(self.engine)  
 self.base = automap\_base(metadata=self.metadata)  
 self.base.prepare()  
 session\_class = sessionmaker(bind=self.engine)  
  
 self.session = session\_class()  
  
  
 except ArgumentError:  
 print('Argument error')  
  
 def delete\_all(self):  
 *"""  
 It deletes all items and all lists  
 """* self.session.query(self.base.classes['client']).delete()  
 self.session.query(self.base.classes['cargo']).delete()  
 self.session.query(self.base.classes['department']).delete()  
 self.session.query(self.base.classes['packing']).delete()  
 self.session.query(self.base.classes['worker']).delete()  
 self.session.query(self.base.classes['ref\_worker\_cargo']).delete()  
 self.session.query(self.base.classes['ref\_client\_worker']).delete()  
 self.session.commit()  
  
 def save\_all(self, objects):  
 *"""  
 It commits objects created by outer scope* ***:param*** *objects: a list of objects to save  
 """* self.session.add\_all(objects)  
 self.session.commit()  
  
 def delete\_request(self, table, where):  
 *'''  
 deletes the row with condition where* ***:param*** *table: name of the table* ***:param*** *where: condition to delete* ***:return****:  
 '''* temp = Table(table, self.metadata, autoload=True, autoload\_with=self.engine)  
 query = delete(temp).where(text(str(where)))  
 results = self.session.execute(query)  
 results = self.session.execute(select([temp])).fetchall()  
 output('output.txt', results)  
 self.session.commit()  
  
 def insert\_request(self, table, condition):  
 temp = Table(table, self.metadata, autoload=True, autoload\_with=self.engine)  
 res = eval('dict(' + condition + ')')  
 query = insert(temp)  
 ResultProxy = self.session.execute(query, res)  
 results = self.session.execute(select([temp])).fetchall()  
 output('output.txt', results)  
 self.session.commit()  
  
 def update\_request(self, table, condition):  
 temp = Table(table, self.metadata, autoload=True, autoload\_with=self.engine)  
 where, what = condition.split(',')  
 res = eval('dict(' + what + ')')  
 query = update(temp).values(res).where(text(where))  
 results = self.session.execute(query)  
 results = self.session.execute(select([temp])).fetchall()  
 output('output.txt', results)  
 self.session.commit()  
  
 def requestFormat(self, comboTable, comboAction, textAction, Controller):  
 Controller.gen\_label.setText('')  
  
 if comboAction == 'delete':  
 try:  
 self.delete\_request(comboTable, textAction)  
 Controller.error.setText('Done')  
  
 except Exception as error:  
 session\_class = sessionmaker(bind=self.engine)  
 self.session = session\_class()  
 Controller.error.setText(str(error))  
 elif comboAction == 'insert':  
 try:  
 self.insert\_request(comboTable, textAction)  
 Controller.error.setText('Done')  
  
 except Exception as error:  
 session\_class = sessionmaker(bind=self.engine)  
 self.session = session\_class()  
 Controller.error.setText(str(error))  
 elif comboAction == 'update':  
 try:  
 self.update\_request(comboTable, textAction)  
 Controller.error.setText('Done')  
  
 except Exception as error:  
 session\_class = sessionmaker(bind=self.engine)  
 self.session = session\_class()  
 Controller.error.setText(str(error))  
  
 def generate\_values(self):  
 with open('data.json', 'r+') as f:  
 data = json.load(f)  
 start\_last\_number = data['last\_number'] + 1  
 start\_estimated\_value = data['estimated\_value'] + 1  
 time = data['date']  
 amount = 20  
 dep = Table('department', self.metadata, autoload=True, autoload\_with=self.engine)  
 worker = Table('worker', self.metadata, autoload=True, autoload\_with=self.engine)  
 client = Table('client', self.metadata, autoload=True, autoload\_with=self.engine)  
 cargo = Table('cargo', self.metadata, autoload=True, autoload\_with=self.engine)  
 packing = Table('packing', self.metadata, autoload=True, autoload\_with=self.engine)  
 cargo\_worker = Table('ref\_worker\_cargo', self.metadata, autoload=True, autoload\_with=self.engine)  
 client\_worker = Table('ref\_client\_worker', self.metadata, autoload=True, autoload\_with=self.engine)  
  
 values\_dep =list()  
 values\_worker = list()  
 values\_client = list()  
 values\_cargo = list()  
 values\_packing = list()  
 values\_ref\_client\_worker = list()  
 values\_ref\_worker\_cargo = list()  
 curr = datetime.datetime.strptime(time, '%Y-%m-%d %H:%M:%S')  
  
 for i in range(start\_last\_number, start\_last\_number+amount):  
 # print(i)  
 temp = randomString(10, False)  
  
 if i%2 == 0:  
 values\_dep.append({'number\_d': i, 'address': randomString(10, True), 'd\_type': 'postal', 'street\_number': random.randint(1, 500)})  
 values\_worker.append({'id': i, 'full\_name': randomString(10, True), 'position': 'casher', 'working\_hours': randomString(5, False), 'salary': random.randint(7000, 15000), 'dep\_number': i})  
 values\_cargo.append({'barcode': i, 'cargo\_type': randomString(20, True), 'estimated\_value': random.randint(200, 1000), 'client\_id': temp, 'worker\_id': i, 'delivered': False})  
 values\_client.append({'full\_name': randomString(20, True), 'client\_type' : 'recipient', 'client\_number': temp}) #*TODO randomString with numbers* else:  
 values\_dep.append({'number\_d': i, 'address': randomString(10, True), 'd\_type': 'cargo','street\_number': random.randint(1, 500)})  
 values\_worker.append({'id': i, 'full\_name': randomString(10, True), 'position': 'heavier','working\_hours': randomString(5, False), 'salary': random.randint(7000, 15000),'dep\_number': i})  
 values\_cargo.append({'barcode': i, 'cargo\_type': randomString(20, True), 'estimated\_value': random.randint(200, 1000), 'client\_id': temp, 'worker\_id': i, 'delivered': True})  
 values\_client.append({'full\_name': randomString(20, True), 'client\_type' : 'sender', 'client\_number': temp}) #*TODO randomString with numbers* values\_packing.append({'packing\_code': i, 'packing\_type': randomString(10, True), 'amount': random.randint(1, 10), 'price': random.randint(1, 36), 'weight': random.randint(1, 30), 'cargo\_barcode': i})  
 values\_ref\_client\_worker.append({'worker\_id': i, 'client\_number': temp, 'time': curr})  
 values\_ref\_worker\_cargo.append({'worker\_id': i, 'cargo\_barcode': i, 'time': curr})  
  
 curr += datetime.timedelta(minutes=random.randrange(1, 20000))  
  
 data = {'last\_number': start\_last\_number + amount, 'estimated\_value': start\_estimated\_value + amount, 'date': str(curr)}  
 with open('data.json', 'w+') as f:  
 json.dump(data, f)  
  
 self.session.execute(insert(dep, values\_dep))  
 self.session.execute(insert(worker, values\_worker))  
 self.session.execute(insert(client, values\_client))  
 self.session.execute(insert(cargo, values\_cargo))  
 self.session.execute(insert(packing, values\_packing))  
 self.session.execute(insert(cargo\_worker, values\_ref\_worker\_cargo))  
 self.session.execute(insert(client\_worker, values\_ref\_client\_worker))  
  
 self.session.commit()  
  
 def gen\_values(self, Controller):  
 print(Controller)  
 self.generate\_values()

Файл controller.py

from view import \*  
from model import \*  
from trip\_d import \*  
import sys  
  
  
  
class TripController( Ui\_Dialog):  
 def \_\_init\_\_(self, MainWindow,db):  
 super().\_\_init\_\_(MainWindow)  
 self.db = db  
 self.columns = ' '  
 self.pushButton.clicked.connect(self.get\_values\_call)  
   
 def get\_values\_call(self):  
 self.db.getValues(self)  
  
class Controller(Ui\_Database):  
   
 def gen\_values\_call(self):  
 self.db.gen\_values(self)  
   
  
 def \_\_init\_\_(self, MainWindow):  
 self.ui = Ui\_Database(MainWindow)  
 self.db = Database()  
 super().\_\_init\_\_(MainWindow)  
 self.gen\_label.setText('')  
  
 self.pushButton.clicked.connect(self.saveInfo)  
 self.genData.clicked.connect(self.gen\_values\_call)  
 self.pushButton\_2.clicked.connect(self.showDialog)  
 #self.search.clicked.connect(self.full\_text\_search\_call)  
 self.pushButton\_3.clicked.connect(self.full\_str\_call)  
 self.comboTable = None  
 self.comboAction = None  
 self.textAction = None  
 self.columns = ' '  
 self.full\_text = ''  
 self.full\_search\_table = ''  
 self.Flag = True  
 self.window = QtWidgets.QDialog(MainWindow)  
 self.ui = TripController(self.window, self.db)  
  
 def saveInfo(self):  
 self.comboAction = self.action.currentText()  
 self.comboTable = self.table.currentText()  
 self.textAction = self.textEdit.toPlainText()  
 self.db.requestFormat(self.comboTable, self.comboAction, self.textAction, self)  
 if not self.Flag:  
 self.error\_dialog = QtWidgets.QErrorMessage()  
 self.error\_dialog.showMessage('Unable to perceive the request')  
 def showDialog(self):  
 self.window.show()  
  
  
 def full\_str\_call(self):  
 self.db.full\_string(self)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QtWidgets.QApplication(sys.argv)  
 MainWindow = QtWidgets.QMainWindow()  
 MainWindow.show()  
 cntr = Controller(MainWindow)  
  
 sys.exit(app.exec\_())

Файл app\_ui.py:

# -\*- coding: utf-8 -\*-  
  
# Form implementation generated from reading ui file 'app\_ui.ui'  
#  
# Created by: PyQt5 UI code generator 5.15.1  
#  
# WARNING: Any manual changes made to this file will be lost when pyuic5 is  
# run again. Do not edit this file unless you know what you are doing.  
  
  
from PyQt5 import QtCore, QtGui, QtWidgets  
  
  
class Ui\_Database(object):  
 def \_\_init\_\_(self, Database):  
 Database.setObjectName("Database")  
 Database.resize(825, 519)  
 self.centralwidget = QtWidgets.QWidget(Database)  
 self.centralwidget.setObjectName("centralwidget")  
 self.pushButton = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton.setGeometry(QtCore.QRect(700, 40, 71, 41))  
 self.pushButton.setObjectName("pushButton")  
 self.table = QtWidgets.QComboBox(self.centralwidget)  
 self.table.setGeometry(QtCore.QRect(30, 50, 131, 41))  
 self.table.setObjectName("table")  
 self.table.addItem("")  
 self.table.addItem("")  
 self.table.addItem("")  
 self.table.addItem("")  
 self.table.addItem("")  
 self.action = QtWidgets.QComboBox(self.centralwidget)  
 self.action.setGeometry(QtCore.QRect(200, 50, 151, 41))  
 self.action.setObjectName("action")  
 self.action.addItem("")  
 self.action.addItem("")  
 self.action.addItem("")  
 self.actLabel = QtWidgets.QLabel(self.centralwidget)  
 self.actLabel.setGeometry(QtCore.QRect(30, 10, 291, 16))  
 font = QtGui.QFont()  
 font.setFamily("Ubuntu")  
 font.setPointSize(11)  
 font.setBold(False)  
 font.setItalic(False)  
 font.setWeight(50)  
 self.actLabel.setFont(font)  
 self.actLabel.setObjectName("actLabel")  
 self.textEdit = QtWidgets.QTextEdit(self.centralwidget)  
 self.textEdit.setGeometry(QtCore.QRect(360, 20, 321, 70))  
 self.textEdit.setObjectName("textEdit")  
 self.genData = QtWidgets.QPushButton(self.centralwidget)  
 self.genData.setGeometry(QtCore.QRect(20, 330, 411, 51))  
 self.genData.setObjectName("genData")  
 self.textSearch = QtWidgets.QTextEdit(self.centralwidget)  
 self.textSearch.setGeometry(QtCore.QRect(20, 190, 411, 131))  
 font = QtGui.QFont()  
 font.setPointSize(9)  
 self.textSearch.setFont(font)  
 self.textSearch.setAutoFormatting(QtWidgets.QTextEdit.AutoAll)  
 self.textSearch.setOverwriteMode(True)  
 self.textSearch.setObjectName("textSearch")  
 self.labelSearch = QtWidgets.QLabel(self.centralwidget)  
 self.labelSearch.setGeometry(QtCore.QRect(450, 230, 151, 31))  
 self.labelSearch.setObjectName("labelSearch")  
 self.pushButton\_2 = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton\_2.setGeometry(QtCore.QRect(20, 390, 411, 61))  
 self.pushButton\_2.setObjectName("pushButton\_2")  
 self.label = QtWidgets.QLabel(self.centralwidget)  
 self.label.setGeometry(QtCore.QRect(30, 300, 63, 20))  
 self.label.setText("")  
 self.label.setObjectName("label")  
 self.pushButton\_3 = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton\_3.setGeometry(QtCore.QRect(600, 280, 111, 31))  
 font = QtGui.QFont()  
 font.setPointSize(11)  
 self.pushButton\_3.setFont(font)  
 self.pushButton\_3.setObjectName("pushButton\_3")  
 self.info = QtWidgets.QLabel(self.centralwidget)  
 self.info.setGeometry(QtCore.QRect(590, 180, 241, 81))  
 font = QtGui.QFont()  
 font.setPointSize(9)  
 self.info.setFont(font)  
 self.info.setStyleSheet("color:green")  
 self.info.setObjectName("info")  
 self.error = QtWidgets.QLabel(self.centralwidget)  
 self.error.setGeometry(QtCore.QRect(20, 110, 751, 31))  
 self.error.setStyleSheet("color:red;")  
 self.error.setText("")  
 self.error.setObjectName("error")  
 self.full\_text\_box = QtWidgets.QComboBox(self.centralwidget)  
 self.full\_text\_box.setGeometry(QtCore.QRect(450, 270, 131, 41))  
 self.full\_text\_box.setObjectName("full\_text\_box")  
 self.full\_text\_box.addItem("")  
 self.full\_text\_box.addItem("")  
 self.full\_text\_box.addItem("")  
 self.full\_text\_box.addItem("")  
 self.full\_text\_box.addItem("")  
 self.gen\_label = QtWidgets.QLabel(self.centralwidget)  
 self.gen\_label.setGeometry(QtCore.QRect(660, 320, 62, 19))  
 self.gen\_label.setText("")  
 self.gen\_label.setObjectName("gen\_label")  
 Database.setCentralWidget(self.centralwidget)  
 self.menubar = QtWidgets.QMenuBar(Database)  
 self.menubar.setGeometry(QtCore.QRect(0, 0, 825, 26))  
 self.menubar.setObjectName("menubar")  
 Database.setMenuBar(self.menubar)  
 self.statusbar = QtWidgets.QStatusBar(Database)  
 self.statusbar.setObjectName("statusbar")  
 Database.setStatusBar(self.statusbar)  
  
 self.retranslateUi(Database)  
 QtCore.QMetaObject.connectSlotsByName(Database)  
  
 def retranslateUi(self, Database):  
 \_translate = QtCore.QCoreApplication.translate  
 Database.setWindowTitle(\_translate("Database", "MainWindow"))  
 self.pushButton.setText(\_translate("Database", "Action"))  
 self.table.setItemText(0, \_translate("Database", "Author"))  
 self.table.setItemText(1, \_translate("Database", "Client"))  
 self.table.setItemText(2, \_translate("Database", "Painting"))  
 self.table.setItemText(3, \_translate("Database", "Phone number"))  
 self.table.setItemText(4, \_translate("Database", "Profile"))  
 self.action.setItemText(0, \_translate("Database", "delete"))  
 self.action.setItemText(1, \_translate("Database", "update"))  
 self.action.setItemText(2, \_translate("Database", "insert"))  
 self.actLabel.setText(\_translate("Database", "Choose table and action to do:"))  
 self.genData.setText(\_translate("Database", "Generate random data to database!"))  
 self.textSearch.setHtml(\_translate("Database", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"  
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"  
"p, li { white-space: pre-wrap; }\n"  
"</style></head><body style=\" font-family:\'MS Shell Dlg 2\'; font-size:9pt; font-weight:400; font-style:normal;\">\n"  
"<p style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px; font-family:\'Cantarell\';\"><br /></p></body></html>"))  
 self.labelSearch.setText(\_translate("Database", "Enter word or phrase:"))  
 self.pushButton\_2.setText(\_translate("Database", "Find paintings -->"))  
 self.pushButton\_3.setText(\_translate("Database", "Search "))  
 self.info.setText(\_translate("Database", " delete: C\_id = 10;\n"  
" insert: C\_id =10, name=\'John\'... \n"  
" update: (where) Cid = 3 \n"  
" (what) number = \'+32132131\' "))  
 self.full\_text\_box.setItemText(0, \_translate("Database", "Author"))  
 self.full\_text\_box.setItemText(1, \_translate("Database", "Client"))  
 self.full\_text\_box.setItemText(2, \_translate("Database", "Painting"))  
 self.full\_text\_box.setItemText(3, \_translate("Database", "Phone number"))  
 self.full\_text\_box.setItemText(4, \_translate("Database", "Profile"))

Файл painting\_dial.py:

# -\*- coding: utf-8 -\*-  
  
# Form implementation generated from reading ui file 'painting\_dial.ui'  
#  
# Created by: PyQt5 UI code generator 5.15.1  
#  
# WARNING: Any manual changes made to this file will be lost when pyuic5 is  
# run again. Do not edit this file unless you know what you are doing.  
  
  
from PyQt5 import QtCore, QtGui, QtWidgets  
  
  
class Ui\_Dialog(object):  
 def \_\_init\_\_(self, Dialog):  
 Dialog.setObjectName("Dialog")  
 Dialog.resize(1158, 586)  
 self.pushButton = QtWidgets.QPushButton(Dialog)  
 self.pushButton.setGeometry(QtCore.QRect(10, 90, 1121, 31))  
 self.pushButton.setObjectName("pushButton")  
 self.label = QtWidgets.QLabel(Dialog)  
 self.label.setGeometry(QtCore.QRect(10, 0, 391, 51))  
 self.label.setObjectName("label")  
 self.label\_2 = QtWidgets.QLabel(Dialog)  
 self.label\_2.setGeometry(QtCore.QRect(20, 40, 391, 51))  
 self.label\_2.setObjectName("label\_2")  
 self.label\_3 = QtWidgets.QLabel(Dialog)  
 self.label\_3.setGeometry(QtCore.QRect(360, 40, 391, 51))  
 self.label\_3.setObjectName("label\_3")  
 self.textEdit = QtWidgets.QTextEdit(Dialog)  
 self.textEdit.setGeometry(QtCore.QRect(60, 50, 271, 31))  
 self.textEdit.setObjectName("textEdit")  
 self.textEdit\_2 = QtWidgets.QTextEdit(Dialog)  
 self.textEdit\_2.setGeometry(QtCore.QRect(400, 50, 281, 31))  
 self.textEdit\_2.setObjectName("textEdit\_2")  
 self.plainTextEdit = QtWidgets.QPlainTextEdit(Dialog)  
 self.plainTextEdit.setGeometry(QtCore.QRect(10, 130, 1121, 451))  
 font = QtGui.QFont()  
 font.setPointSize(8)  
 self.plainTextEdit.setFont(font)  
 self.plainTextEdit.setObjectName("plainTextEdit")  
  
 self.retranslateUi(Dialog)  
 QtCore.QMetaObject.connectSlotsByName(Dialog)  
  
 def retranslateUi(self, Dialog):  
 \_translate = QtCore.QCoreApplication.translate  
 Dialog.setWindowTitle(\_translate("Dialog", "Dialog"))  
 self.pushButton.setText(\_translate("Dialog", "Get Values"))  
 self.label.setText(\_translate("Dialog", "Please, enter time interval:"))  
 self.label\_2.setText(\_translate("Dialog", "From:"))  
 self.label\_3.setText(\_translate("Dialog", "To:"))