

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

Гольовський А. А.

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.

###### ***Посилання на код в GitHub:***

***https://github.com/Adniry/DB/tree/master/lab3***

***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**

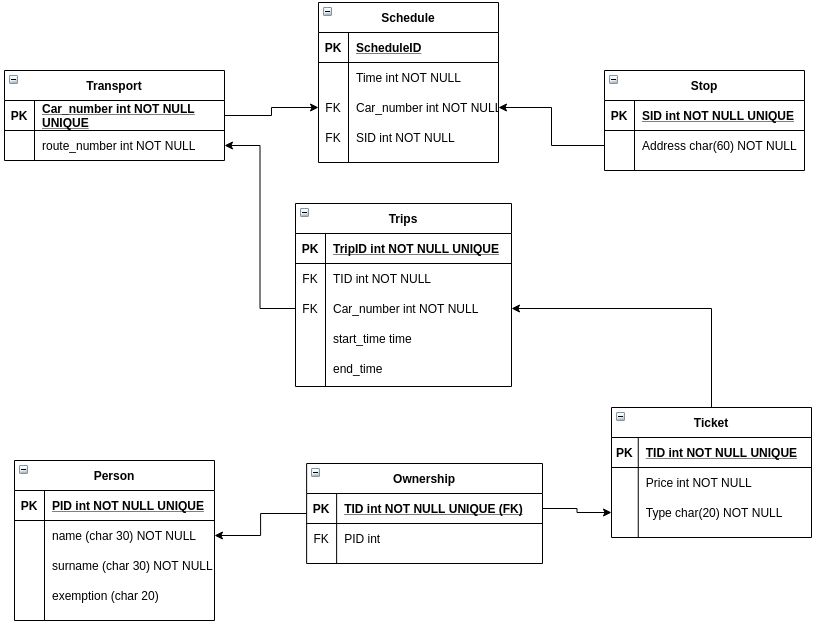
Опис бази даних:

База даних репрезентує предметну область «Міський громадський транспорт». Основні сутності:

* Людина (пасажир)
* Транспорт
* Зупинка
* Квиток

В результаті нормалізації було додано ще 3 таблиці, текстове представлення кінцевого варіанту бази даних наведено в таблиці:

|  |  |  |
| --- | --- | --- |
| **Відношення** | **Атрибут** | **Тип** |
| Відношення “Person” містить інформацію про людину-користувача транспортом. | PID – унікальний номер особи в БД  Name – ім’я особи. Не допускає NULL.  Surname – прізвище особи. Не допускає NULL.  Exemption – тип пільги особи. Не допускає NULL | Числовий, SERIAL PK  Текстовий(30)  Текстовий(30)  Текстовий(20) |
| Відношення ”Ticket” містить інформацію про квитки на транспорт | TicketID – унікальний номер квитка в БД.  Operation\_time – час дії квитка. Не допускає NULL.  Price – ціна квитка. Не допускає NULL. | Числовий, SERIAL PK  Часова мітка  Грошовий |
| Відношення ”Transport” містить інформацію про одиниці транспорту в парку | Car\_number – унікальний номер одиниці транспорту  Route\_number – номер маршруту. Не допускає NULL | Числовий, SERIAL PK  Числовий |
| Відношення ”Stop” містить інформацію про зупинки в місті | SID - унікальний номер зупинки в БД  Address – адреса зупинки. Не допускає NULL | Числовий, SERIAL PK  Текстовий(50) |
| Відношення “Schedule” містить інформацію про час зупинки кожної одиниці транспорту на кожній зупинці її маршруту. | ScheduleID – унікальний номер запису розкладу.  Time – час зупинки. Не допускає NULL  SID – ідентифікатор зупинки  Car\_number – ідентифікатор транспортної одиниці | Числовий SERIAL PK  Часова мітка  Числовий FK  Числовий FK |
| Відношення “Trip” містить інформацію про кожну поїздку | TripID – унікальний номер поїздки  Car\_number – ідентифікатор одиниці транспорту, на якій було здійснено поїздку  TicketID – ідентифікатор квитка, за допомогою якого було здійснено поїздку  Start time – час посадки  End time – час висадки | Числовий SERIAL PK  Числовий FK  Числовий FK  Часова мітка  Часова мітка |
| Відношення “Ownership” містить інформацію про те, кому який квиток належить | TID — ідентифікатор квитка  PID — ідентифікатор особи-власника квитка | Числовий PK FK  Числовий FK |

Рисунок 1 — Логічна модель предметної області “Міський транспорт”.

**Засоби програмування і спосіб доступу до БД**

Мова програмування: python;

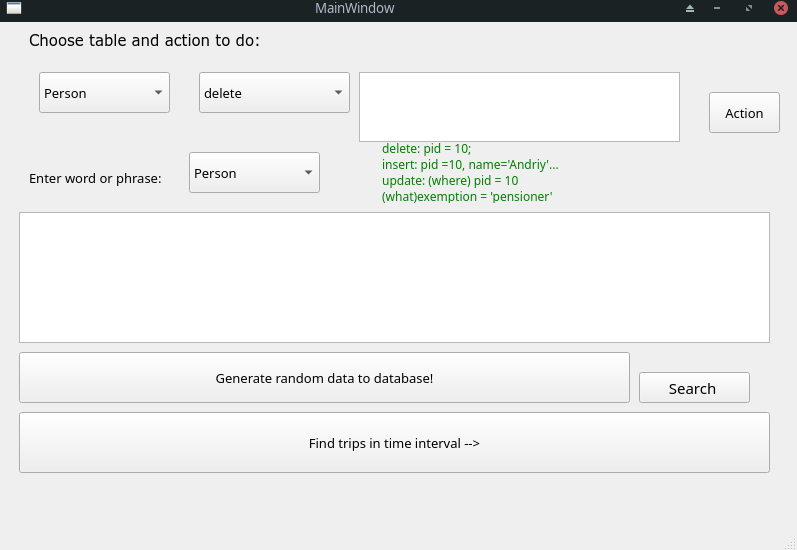
Середовище програмування (IDE): PyCharm;

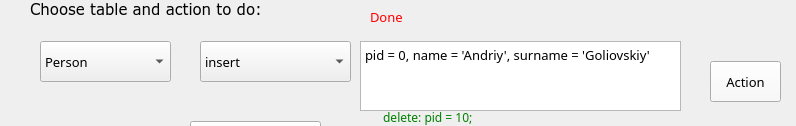
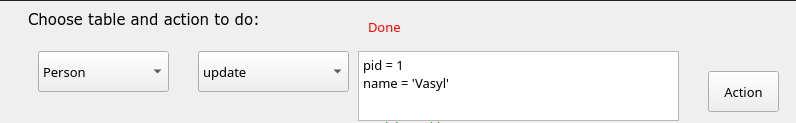
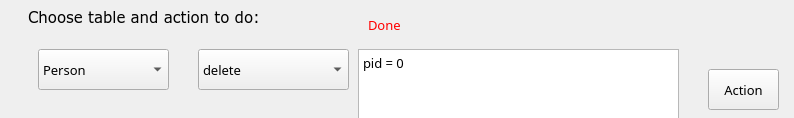
Засіб для розробки графічного інтерфейсу: QtDesigner, pyuic5;

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 Person(DeclarativeBase):  
 \_\_tablename\_\_ = 'person'  
  
 pid = Column(Integer, primary\_key=True)  
 name = Column(String)  
 surname = Column(String)  
 exemption = Column(String)  
  
class Transport(DeclarativeBase):  
 \_\_tablename\_\_ = 'transport'  
  
 car\_number = Column(Integer, primary\_key=True)  
 route\_number = Column(Integer)  
  
class Stop:  
 \_\_tablename\_\_ = 'stop'  
  
 sid = Column(Integer, primary\_key=True)  
 address = Column(String)  
  
class Ticket:  
 \_\_tablename\_\_ = 'ticket'  
  
 tid = Column(Integer, primary\_key=True)  
 price = Column(Numeric)  
 operation\_time = Column(DateTime)  
  
class Schedule:  
 \_\_tablename\_\_ = 'schedule'  
  
 schedule\_id = Column(Integer, primary\_key=True)  
 car\_number = Column("car\_number", Integer, ForeignKey("transport.car\_number", ondelete="CASCADE"))  
 sid = Column("sid", Integer, ForeignKey("stop.sid", ondelete="CASCADE"))  
 time = Column(DateTime)  
  
class Ownership:  
 \_\_tablename\_\_ = 'ownership'  
  
 tid = Column("tid", Integer, ForeignKey("ticket.tid", ondelete="CASCADE"), primary\_key=True)  
 pid = Column("pid", Integer, ForeignKey('person.pid', ondelete="CASCADE"))  
  
class Trip:  
 \_\_tablename\_\_ = 'tripid'  
  
 tripid = Column("tripid", Integer, primary\_key=True)  
 car\_number = Column("car\_number", Integer, ForeignKey("transport.car\_number", ondelete="CASCADE"))  
 tid = Column("tid", Integer, ForeignKey("ticket.tid", ondelete="CASCADE"))  
 start\_time = Column("start\_time", DateTime)  
 end\_time = Column('end\_time', DateTime)

Код запитів на видалення, вставку, оновлення 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()  
  
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()

**Завдання 2**

Btree

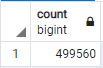
Для дослідження індексу була створена таблиця test\_btree, яка має дві колонки: числову і текстову. Вони проіндексовані як BTree. У таблицю було занесено 1000000 записів.

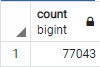
Виконаємо запити для пошуку:





Результати:





Для неіндексованої таблиці:





Для проіндексованої таблиці:



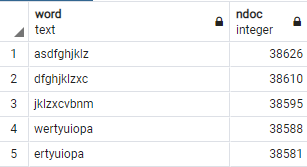


GIN

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:

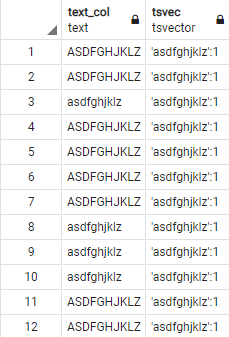




Проведемо пошук по першому слову:



Результат:



Для неіндексованої таблиці:



Для проіндексованої таблиці:



**Завдання 3**

Тригер:



Принцип роботи:

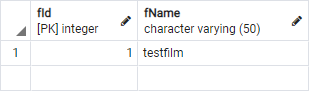
Тригер спрацьовує при вставці чи оновленні у таблиці film. Якщо значення колонки filmName, яке вставляється чи оновлюється, дорівнює testfilm, то цей новий запис заноситься у таблицю filmLog. В іншому випадку, до кожного значення fName у таблиці filmLog додається “not test”.

Результат роботи:

Вставляємо запис:

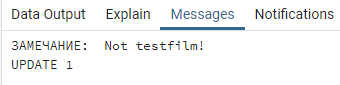


filmLog:

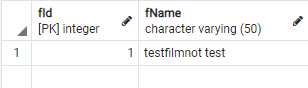


Оновимо запис:





filmLog:



**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  
import sqlalchemy.ext.declarative  
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 = sqlalchemy.ext.declarative.declarative\_base()  
metadata = DeclarativeBase.metadata  
DATABASE\_URI = 'postgresql://postgres:14725@localhost:5433/db2'  
engine = create\_engine(DATABASE\_URI)  
class Person(DeclarativeBase):  
 \_\_tablename\_\_ = 'person'  
  
 pid = Column(Integer, primary\_key=True)  
 name = Column(String)  
 surname = Column(String)  
 exemption = Column(String)  
  
class Transport(DeclarativeBase):  
 \_\_tablename\_\_ = 'transport'  
  
 car\_number = Column(Integer, primary\_key=True)  
 route\_number = Column(Integer)  
  
class Stop:  
 \_\_tablename\_\_ = 'stop'  
  
 sid = Column(Integer, primary\_key=True)  
 address = Column(String)  
  
class Ticket:  
 \_\_tablename\_\_ = 'ticket'  
  
 tid = Column(Integer, primary\_key=True)  
 price = Column(Numeric)  
 operation\_time = Column(DateTime)  
  
class Schedule:  
 \_\_tablename\_\_ = 'schedule'  
  
 schedule\_id = Column(Integer, primary\_key=True)  
 car\_number = Column("car\_number", Integer, ForeignKey("transport.car\_number", ondelete="CASCADE"))  
 sid = Column("sid", Integer, ForeignKey("stop.sid", ondelete="CASCADE"))  
 time = Column(DateTime)  
  
class Ownership:  
 \_\_tablename\_\_ = 'ownership'  
  
 tid = Column("tid", Integer, ForeignKey("ticket.tid", ondelete="CASCADE"), primary\_key=True)  
 pid = Column("pid", Integer, ForeignKey('person.pid', ondelete="CASCADE"))  
  
class Trip:  
 \_\_tablename\_\_ = 'tripid'  
  
 tripid = Column("tripid", Integer, primary\_key=True)  
 car\_number = Column("car\_number", Integer, ForeignKey("transport.car\_number", ondelete="CASCADE"))  
 tid = Column("tid", Integer, ForeignKey("ticket.tid", ondelete="CASCADE"))  
 start\_time = Column("start\_time", DateTime)  
 end\_time = Column('end\_time', DateTime)  
  
DeclarativeBase.metadata.create\_all(engine)  
  
  
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:  
 DATABASE\_URI = 'postgresql://adniry:14725@localhost:5432/db2'  
 self.engine = create\_engine(DATABASE\_URI)  
 #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['person']).delete()  
 self.session.query(self.base.classes['ticket']).delete()  
 self.session.query(self.base.classes['stop']).delete()  
 self.session.query(self.base.classes['schedule']).delete()  
 self.session.query(self.base.classes['ownership']).delete()  
 self.session.query(self.base.classes['trip']).delete()  
 self.session.query(self.base.classes['transport']).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))

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 view.py:

# -\*- coding: utf-8 -\*-  
  
# Form implementation generated from reading ui file '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(798, 529)  
 self.centralwidget = QtWidgets.QWidget(Database)  
 self.centralwidget.setObjectName("centralwidget")  
 self.pushButton = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton.setGeometry(QtCore.QRect(710, 70, 71, 41))  
 self.pushButton.setObjectName("pushButton")  
 self.table = QtWidgets.QComboBox(self.centralwidget)  
 self.table.setGeometry(QtCore.QRect(40, 50, 131, 41))  
 self.table.setObjectName("table")  
 self.table.addItem("")  
 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, 50, 321, 70))  
 self.textEdit.setObjectName("textEdit")  
 self.genData = QtWidgets.QPushButton(self.centralwidget)  
 self.genData.setGeometry(QtCore.QRect(20, 330, 611, 51))  
 self.genData.setObjectName("genData")  
 self.textSearch = QtWidgets.QTextEdit(self.centralwidget)  
 self.textSearch.setGeometry(QtCore.QRect(20, 190, 751, 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(30, 140, 151, 31))  
 self.labelSearch.setObjectName("labelSearch")  
 self.pushButton\_2 = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton\_2.setGeometry(QtCore.QRect(20, 390, 751, 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(640, 350, 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(380, 120, 241, 61))  
 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(370, 10, 401, 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(190, 130, 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.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, 798, 25))  
 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", "Person"))  
 self.table.setItemText(1, \_translate("Database", "Transport"))  
 self.table.setItemText(2, \_translate("Database", "Stop"))  
 self.table.setItemText(3, \_translate("Database", "Ticket"))  
 self.table.setItemText(4, \_translate("Database", "Schedule"))  
 self.table.setItemText(5, \_translate("Database", "Trip"))  
 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:\'Noto Sans\'; 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 trips in time interval -->"))  
 self.pushButton\_3.setText(\_translate("Database", "Search "))  
 self.info.setText(\_translate("Database", " delete: pid = 10;\n"  
" insert: pid =10, name=\'Andriy\'... \n"  
" update: (where) pid = 10 \n"  
" (what)exemption = \'pensioner\' "))  
 self.full\_text\_box.setItemText(0, \_translate("Database", "Person"))  
 self.full\_text\_box.setItemText(1, \_translate("Database", "Transport"))  
 self.full\_text\_box.setItemText(2, \_translate("Database", "Stop"))  
 self.full\_text\_box.setItemText(3, \_translate("Database", "Ticket"))  
 self.full\_text\_box.setItemText(4, \_translate("Database", "Schedule"))  
 self.full\_text\_box.setItemText(5, \_translate("Database", "Trip"))

Файл trip\_d.py:

# -\*- coding: utf-8 -\*-  
  
# Form implementation generated from reading ui file 'trip\_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:"))