6 в: Диспетчер автоперевозок. В бд хранятся сведения об автомобилях, водителях, грузах, адресах, рейсах.Предполагается,

что за один рейс можно перевезти только один груз в один пункт.Все рейсы выполняются в пределах одной рабочей смены.  
Таблицы: Пункты (Код пункта, название), Грузы (Код груза, название, масса),  
Автомобили (Код автомобиля, марка, госномер, грузоподъемность);  
Водители (Код водителя, ФИО, телефон);  
Рейсы (Код рейса, код водителя, код автомобиля, код пункта, код груза, время выезда, время в пути).  
Определить: - список водителей в рейсе на данный момент времени; - ср загруженность (по массе) автомобилей.  
import sqlite3  
import sys  
from db\_transport import DatabaseManager  
from design import \*  
from PyQt6.QtWidgets import QApplication, QMainWindow, QMessageBox  
import openpyxl  
from openpyxl.styles import Font  
  
**class Transport(QMainWindow):** # наш класс явл наследником класса QMainWindow  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_() # вызываем метод для родительского класса  
 self.ui = Ui\_main\_window() # создаем экземпляр интерфейса  
 self.ui.setupUi(self) # настраиваем интерфейс  
 self.db = DatabaseManager()  
 self.init\_db()  
 self.load\_cb()  
 self.ui.save\_route\_btn.clicked.connect(self.save\_route)  
 self.ui.query1\_btn.clicked.connect(self.query1)  
 self.ui.query2\_btn.clicked.connect(self.query2)

def init\_db(self):  
 ***"""подключаемся к бд """***self.conn = sqlite3.connect('road\_transport.db')  
 self.cursor = self.conn.cursor()

def load\_cb(self):  
 ***"""заполняем комбо бокс"""***# водители  
 self.cursor.execute('select idDriver, name from drivers')  
 drivers = self.cursor.fetchall()  
 self.ui.drivers\_cb.clear()  
 for driver in drivers:  
 self.ui.drivers\_cb.addItem(driver[1], driver[0])

# автомобили  
 self.cursor.execute("select idCar, brand ||' '|| num from cars")  
 cars = self.cursor.fetchall()  
 self.ui.car\_cb.clear()  
 for car in cars:  
 self.ui.car\_cb.addItem(car[1], car[0]

def save\_route(self):  
 ***""" сохраняем рейс в бд """***id\_driver = self.ui.drivers\_cb.currentData()  
 id\_car = self.ui.car\_cb.currentData()  
 id\_point = self.ui.point\_cb.currentData()  
 id\_cargo = self.ui.cargo\_cb.currentData()  
 time\_out = self.ui.departure\_lineEdit.text()  
 time\_in = self.ui.time\_in\_lineEdit.text()  
  
 if not all([id\_driver,id\_point, id\_car, id\_cargo, time\_out, time\_in]):  
 QMessageBox.warning(self, "Ошибка", "Заполните все поля!")  
 return  
  
 try:  
 query = """ insert into routes (driver\_id, car\_id, point\_id, cargo\_id, time\_out, time\_in)   
 values (?, ?, ?, ?, ?, ?) """  
 self.cursor.execute(query, (id\_driver, id\_car, id\_point, id\_cargo, time\_out, time\_in))  
 self.conn.commit()  
 QMessageBox.information(self, "Успех", "рейс добавлен")  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Ошибка при добавлении рейса: {e}")  
  
 **def closeDb(self, event):**  
 self.conn.close()  
 event.accept()  
  
 **def query1(self):**  
 try:  
 drivers = self.db.get\_drivers\_in\_road()

if drivers:

message = “водители в рейсе: \n”

for driver in drivers:

message += f"Водитель: {driver[0]}, Время выезда: {driver[1]}, Время в пути: {driver[2]} ч\n"

QMessageBox.information(self, "результаты запроса", message)

else:

QMessageBox.critical(self, "Информация", "Нет данных")  
 except Exception as e:  
 QMessageBox.warning(self, "Ошибка", f"Ошибка при выполнении запроса {str(e)}")  
  
 **def query2(self):**  
 try:  
 avg\_load = self.db.get\_avg\_car\_load()  
 result = f"средняя загруженность авто: {avg\_load:.2f}%"  
 self.export\_to\_excel()  
 print(result)  
  
 QMessageBox.information(self, “результаты запроса”, result)  
 except Exception as e:  
 print(f"Ошибка: {e}")  
 QMessageBox.information(self, "Ошибка", "Ошибка при выполнении запроса")  
  
 **def export\_to\_excel(self):**  
 *""" экспорт в иксель """* try:  
 avg\_load = self.db.get\_avg\_car\_load()  
 workbook = openpyxl.Workbook()  
 sheet = workbook.active  
 sheet.title = 'Диспетчер автоперевозок'  
 sheet.cell(row=1, column=1, value="Средняя загруженность автомобилей (%)")  
 sheet.cell(row=1, column=1).font = Font(bold=True)  
 sheet.cell(row=2, column=1, value=round(avg\_load, 2) if avg\_load else "Нет данных")  
 file\_name = "road\_transport.xlsx"  
 workbook.save(file\_name)  
 print(f"Данные успешно экспортированы в файл {file\_name}.")  
 QMessageBox.information(self, "Экспорт в Excel", f"Средняя загруженность экспортирована в файл {file\_name}.")  
 except Exception as e:  
 print(f"Ошибка при экспорте в Excel: {e}")  
 QMessageBox.critical(self, "Ошибка", "Не удалось экспортировать данные."  
**if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 window = Transport()  
 window.show()  
 sys.exit(app.exec())**

import sqlite3  
from datetime import datetime, timedelta  
  
  
**class DatabaseManager():**  
 def \_\_init\_\_(self):  
 self.conn = sqlite3.connect('road\_transport.db')  
 self.cursor = self.conn.cursor()  
  
 def create\_tables(self):  
 self.cursor.execute(''' CREATE TABLE IF NOT EXISTS points (  
 idPoint integer primary key autoincrement,  
 name text not null ) ''')  
  
 self.cursor.execute(''' create table if not exists cargo (  
 idCargo integer primary key autoincrement,  
 name text not null,  
 weight real not null ) ''')  
   
 self.cursor.execute(''' create table if not exists routes (  
 idRoute integer primary key autoincrement,  
 driver\_id integer,  
 car\_id integer,  
 point\_id integer,  
 cargo\_id integer,  
 time\_out text,  
 time\_in text,  
 foreign key (driver\_id) references drivers(idDriver),  
 foreign key (car\_id) references cars(idCar),  
 foreign key (point\_id) references points (idPoint),  
 foreign key (cargo\_id) references cargo(idCargo))  
 ''')  
 self.cursor.executescript('''  
 INSERT OR IGNORE INTO points (idPoint, name) VALUES (1, 'Москва'), (2, 'Санкт-Петербург');  
 INSERT OR IGNORE INTO routes (idRoute, driver\_id, car\_id, point\_id, cargo\_id, time\_out, time\_in) VALUES   
 (1, 1, 1, 1, 1, '2024-12-17 09:00:00', 5),   
 (2, 2, 2, 2, 2, '2024-12-17 10:00:00' , 2),  
 (1, 2, 2, 1, 2, '2024-12-18 23:50:00', 10);  
 ''')

self.conn.commit()  
 print('таблицы успешно созданы')  
  
 **def close\_db(self):**  
 self.cursor.close()  
 self.conn.close()  
  
 # запросы sql  
 #получаем водителей в рейсе  
 **def get\_drivers\_in\_road(self):**  
 current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S") #strfrime для преобразования времени в строку  
 query = """ select drivers.name, routes.time\_out, routes.time\_in from routes  
 join drivers on routes.driver\_id = drivers.idDriver  
 where datetime(routes.time\_out) <= ? and datetime(routes.time\_out, '+' || routes.time\_in || ' hours') > ? """  
 self.cursor.execute(query, (current\_time, current\_time))  
 return self.cursor.fetchall()  
  
 # средняя загруженность автомобилей  
 **def get\_avg\_car\_load(self):** query = """ select avg(cargo.weight / cars.capacity ) \* 100 as avg\_load  
 from routes   
 join cargo on routes.cargo\_id = cargo.idCargo  
 join cars on routes.car\_id = cars.idCar """  
 self.cursor.execute(query)  
 result = self.cursor.fetchone()  
 return result[0] if result else 0  
  
def main():  
 db = DatabaseManager()  
 db.create\_tables()  
  
  
 print('check sql queries')  
 print("==================")  
  
 print('\n1. drivers list in travel\_time:')  
 drivers\_in\_transit = db.get\_drivers\_in\_road()  
 if drivers\_in\_transit:  
 for driver in drivers\_in\_transit:  
 print(f"Водитель: {driver[0]}, Время выезда: {driver[1]}, Время в пути (часы): {driver[2]}")  
 else:  
 print('нет водителей в пути')  
  
  
 # Вывод средней загруженности автомобилей  
 print("\n2. Средняя загруженность автомобилей (в процентах):")  
 avg\_load = db.get\_avg\_car\_load()  
 print(f"{avg\_load:.2f}%")  
  
 db.close\_db()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

8.Ежедневник.В бд хранятся сведения о намеченных событиях (делах, встречах и звонках) и их выполнении.  
Таблицы: Виды событий (Код вида события, название вида), События (Код события, код вида события, название события,

время начала, продолжительность, отметка о выполнении).  
Требуется: - определить относительную долю загрузки рабочего дня по приоритетам;  
- построить список невыполненных дел в порядке срока давности.

import sqlite3  
  
**class DataBase():**  
 def \_\_init\_\_(self):  
 self.conn = sqlite3.connect('planner.db')  
 self.cursor = self.conn.cursor()  
  
 def create\_tables(self):  
 self.cursor.executescript("""   
 insert or ignore into type\_event (idType, name) values   
 (1, 'Высокий'),   
 (2, 'Средний'),  
 (3, 'Низкий');  
  
 insert or ignore into event (idEvent, id\_type, name, start\_time, duration, mark\_done) values   
 (1, 1, 'Подготовка презентации к совещанию', '10:00 2024-12-18 ', 60, 1 ),  
 (2, 3, 'Уборка дома', '20:00 2024-12-20', 120, 0 ),  
 (3, 2, 'Подача заявления на визу до 14 декабря', '12:00 2024-12-01', 120, 0),  
 (4, 2, 'Проверка почты', '10:00 2024-12-04', 60, 1),  
 (5, 1, 'Посещение врача', '14:00 2024-12-20', 120, 1); """)  
  
 self.conn.commit()  
 print('таблицы созданы')  
  
 def close\_db(self): …  
 # определить относительную долю загрузки рабочего дня по приоритетам  
 **def query1(self):**  
 query = """ select type\_event.name,   
 sum(event.duration) as total\_duration,   
 (sum(event.duration) \* 100.0 / (select sum(duration) from event)) as percentage\_share  
 from event   
 join type\_event on event.id\_type = type\_event.idType  
 group by type\_event.name"""  
 self.cursor.execute(query)  
 result = self.cursor.fetchall()  
 return result if result else 0  
  
 # построить список невыполненных дел в порядке срока давности  
 **def query2(self):** query = """ select event.name, event.start\_time from event  
 where event.mark\_done = 0  
 order by event.start\_time"""  
 self.cursor.execute(query)  
 result = self.cursor.fetchall()  
 return result if result else 0

import sqlite3  
from planner\_db import DataBase  
from design import \*  
import sys  
from PyQt6.QtWidgets import QApplication, QMainWindow, QMessageBox  
from openpyxl import Workbook  
  
**class Planner(QMainWindow):** def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.ui = Ui\_main()  
 self.ui.setupUi(self)  
 self.db = DataBase()  
 self.init\_db()  
 self.load\_type\_cb()  
 self.ui.add\_event\_btn.clicked.connect(self.add\_event)  
 self.ui.query1\_btn.clicked.connect(self.query1)  
 self.ui.query2\_btn.clicked.connect(self.query2)  
  
 def init\_db(self): . . .  
   
 **def load\_type\_cb(self):**  
 *""" заполняем комбо бокс типа события"""* self.cursor.execute(""" select \* from type\_event """)  
 types = self.cursor.fetchall()  
 self.ui.type\_event\_cb.clear()  
 for type in types:  
 self.ui.type\_event\_cb.addItem(type[1], type[0])  
 **def add\_event(self):**  
 *"""добавить событие в бд"""* name = self.ui.textEdit.toPlainText()  
 id\_type = self.ui.type\_event\_cb.currentData()  
 start\_time = self.ui.start\_time\_lineEdit.text()  
 duration = self.ui.lineEdit.text()  
 mark\_done = 1 if self.ui.done\_cb.isChecked() else 0  
 print(  
 f"DEBUG: Name={name}, ID Type={id\_type}, Start Time={start\_time}, Duration={duration}, Mark Done={mark\_done}")  
  
 if not all([name, id\_type, start\_time, duration]):  
 QMessageBox.warning(self, "Ошибка", "Заполните все поля!")  
 return  
  
 try:  
 query = """ insert into event (id\_type, name, start\_time, duration, mark\_done)   
 values (?, ?, ?, ?, ?) """  
 self.cursor.execute(query, (id\_type, name, start\_time, duration, mark\_done))  
 self.conn.commit()  
 QMessageBox.information(self, "Успех", "Событие добавлено")  
  
 except Exception as e:  
 print(f"DEBUG: SQLite Error: {e}")  
 QMessageBox.critical(self, "Ошибка", f"Ошибка при добавлении: {e}")  
  
 **def query1(self):** try:  
 data = self.db.query1()  
  
 if not data:  
 self.ui.query1\_result.setPlainText('относительная доля загрузки не определена')  
  
 else:  
 result = "Относительная доля загрузки рабочего дня:\n"  
 for row in data:  
 result += f"{row[0]}: {row[1]}%\n"  
 self.ui.query1\_result.setPlainText(result)  
 except Exception as e:  
 QMessageBox.critical(self, "ошибка", f"ошибка при выполнении запроса {e}")  
  
 **def query2(self):**  
 try:  
 data = self.db.query2()  
  
 if not data:  
 self.ui.query2\_result.setPlainText("Все дела выполнены")  
 else:  
 result = "список невыполненных дел \n"  
 for row in data:  
 result += f" {row[0]} - {row[1]}\n"  
 self.ui.query2\_result.setPlainText(result)  
  
 """ экспорт в иксель """  
 wb = Workbook()  
 ws = wb.active  
 ws.title = 'Невыполненные дела'  
 ws.append(["Название", "Дата и время начала"])  
  
 for row in data:  
 ws.append(row)  
  
 excel\_file = 'planner.xlsx'  
 wb.save(excel\_file)  
 QMessageBox.information(self, "Успех", f"Данные экспортированы в файл {excel\_file}")  
  
 except Exception as e:  
 QMessageBox.critical(self, "ошибка", f"ошибка при выполнении запроса и экспорте {e}")  
  
 **def close\_db**(self, event):. . .

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

#9. Кадровое агентство.В бд хранятся сведения о вакансиях, включая следующие требования: образование, профессию, уровень владения компьютером.  
#Таблицы: Должности (Код должности, название),Профессии (Код профессии, название), Образование (Код образования, название), Уровень владения компьютером (Код уровня, название),Вакансия (Код вакансии, дата опубликования, название вакансии, оклад, актуальность, код профессии, код образования, код уровня).  
Требуется:- выдать список актуальных (незакрытых вакансий): - построить сравнительную диаграмму среднего оклада в зависимости от уровня владения компьютером.

import sys  
from PyQt6.QtWidgets import QApplication, QWidget, QMainWindow, QMessageBox  
from hr\_db import \*  
from untitled import \*  
from second\_win import \*

from datetime import datetime  
  
  
class Hr\_window(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.ui = Ui\_Form()  
 self.ui.setupUi(self)  
  
 self.db = DataBase()  
 self.init\_db()  
 self.educ\_cbox()  
 self.comp\_level()  
 self.prof\_combobox()  
  
 self.ui.save\_btn.clicked.connect(self.save\_hr)  
 self.ui.query1\_btn.clicked.connect(self.query1)  
 self.ui.query2\_btn.clicked.connect(self.query2)  
  
 **def init\_db(self): . . .**  
  
  
 **def educ\_cbox(self):**  
 ***"""комбо бокс образование"""***self.cursor.execute("""select idEducation, name from education""")  
 educations = self.cursor.fetchall()  
 self.ui.ed\_comboBox.clear()  
 for educ in educations:  
 self.ui.ed\_comboBox.addItem(educ[1], educ[0])  
  
 **def save\_hr(self):**  
 name = self.ui.name\_lineEdit.text()  
 salary = self.ui.salary\_lineEdit.text()  
 prof = self.ui.prof\_comboBox.currentData()  
 educ = self.ui.ed\_comboBox.currentData()  
 level = self.ui.level\_comboBox.currentData()  
 date\_public = datetime.now().strftime("%Y-%m-%d")  
  
 if not all([name, salary, prof, educ, level]):  
 QMessageBox.warning(self, "Предупреждение", "Заполните все поля")  
 return  
  
 **try:  
 salary = float(salary)  
 if salary <= 0:  
 raise ValueError  
 except ValueError:  
 QMessageBox.critical(self, "Внимание", "Введите положительное число")**  
  
 try:  
 query = """insert into vacancy (date\_public, name, salary, actuality, id\_prof, id\_educ, id\_level)   
 values (?, ?, ?, ?, ?, ?, ?) """  
  
 self.cursor.execute(query, (date\_public, name, salary, 'актуально', prof, educ, level))  
 self.conn.commit()  
 QMessageBox.information(self, "успех", "вакансия добавлена")  
 except Exception as e:  
 QMessageBox.critical(self, 'Ошибка', f"не удалось добавить {str(e)}")  
  
 **def query1(self):** try:  
 result = self.db.query\_1()  
 if result:  
 text = "\n".join(f"{id\_vac} {name} {salary} {date\_public} {actuality}"  
 for id\_vac, name, salary, date\_public, actuality in result)  
 else:  
 text = f"нет данных о продажах"  
 **self.win2 = Second\_window(self, text)  
 self.win2.show()  
 self.hide()**  
  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Не удалось выполнить запрос: {str(e)}")  
 **def query2(self):** try:  
 results = self.db.query\_2()  
 if results:  
 message = "Средний оклад \n"  
 for result in results:  
 message += f"Уровень владения комп {result[0]} Средний оклад {result[1]} \n"  
 QMessageBox.information(self, "результаты запроса", message)  
 else:  
 QMessageBox.critical(self, "Информация", "Нет данных")  
 except Exception as e:  
 QMessageBox.warning(self, "Ошибка", f"Ошибка при выполнении запроса {str(e)}")  
  
 **def closeEvent(self, a0):** . . .  
   
if \_\_name\_\_ == "\_\_main\_\_": . . .

import sqlite3  
  
**class DataBase():**  
 def \_\_init\_\_(self):  
 self.conn = sqlite3.connect('hr.db')  
 self.cursor = self.conn.cursor()  
  
 def create\_tables(self):  
 self.cursor.executescript(""" insert or ignore into post(idPost, name) values   
 (1, 'Менеджер'), (2, 'Разработчик'), (3, 'Дизайнер');  
   
 insert or ignore into profession(idProfession, name) values  
 (1, 'IT-специалист'), (2, 'Графический дизайнер'), (3, 'Маркетинговый специалист'), (4, 'Дата-аналитик');  
   
 insert or ignore into education (idEducation, name) values (1, 'Высшее'), (2, 'СПО');  
   
 insert or ignore into comp\_level (idLevel, name) values (1, 'junior'), (2, 'middle'), (3, 'senior');   
   
 insert or ignore into vacancy (idVacancy, date\_public, name, salary, actuality, id\_prof, id\_educ, id\_level)  
 values (1, '2024-12-10', 'software developer' , 70000, 'актуально', 1, 1, 3),   
 (2, '2024-12-10', 'менеджер по маркетингу' , 60000, 'актуально', 3, 1, 2),  
 (3, '2024-12-15', 'Графический дизайнер', 60000, "неактуальнo", 2, 2, 2),  
 (4, '2024-12-22', 'Аналитик данных', 80000, "актуальнo", 4, 1, 3);  
 """)  
  
 self.conn.commit()  
 print('созданы')  
  
 def close\_db(self):  
 self.cursor.close()  
 self.conn.close()  
  
 def query\_1(self):  
 *"""получить список актуальных вакансий"""* query = """ select idVacancy, date\_public, name, salary, actuality   
 from vacancy   
 where actuality = 'актуально' """  
 self.cursor.execute(query)  
 result = self.cursor.fetchall()  
 return result if result else 0  
  
 def query\_2 (self):  
 *""" среднего оклада в зависимости от уровня владения компьютером"""* query = """ select comp\_level.name, avg(vacancy.salary) as avg\_salary   
 from vacancy   
 join comp\_level on vacancy.id\_level = comp\_level.idLevel  
 group by comp\_level.name  
 order by avg\_salary desc"""  
 self.cursor.execute(query)  
 result = self.cursor.fetchall()  
 return result if result else 0  
  
  
db = DataBase()  
db.create\_tables()  
db.close\_db()

from PyQt6.QtWidgets import QWidget, QApplication, QMessageBox  
from win2 import \*  
from main import \*  
  
**class Second\_window(QtWidgets.QWidget):**  
 def \_\_init\_\_(self, main\_window, text):  
 super().\_\_init\_\_()  
 self.ui = Ui\_win2()  
 self.ui.setupUi(self)  
 self.main\_win = main\_window  
  
 self.ui.q1\_plainTextEdit.setPlainText(text)  
 self.ui.exit\_pushButton.clicked.connect(self.back)  
  
 def back(self):  
 self.main\_win.show()  
 self.close()

# 11. Кинотеатр. Таблицы: Зал (Код зала, вместимость); Фильм (Код фильма, Название, Жанр);

Продажа (Код продажи, Код фильма, Код зала, Время, Номер места).  
Определить: - Процент наполнения зала по сеансам; - Самый популярный жанр

import sys  
  
import openpyxl  
  
from cinema\_db import \*  
from design import \*  
import sqlite3  
from PyQt6.QtWidgets import QApplication, QMainWindow, QMessageBox  
from datetime import datetime, timedelta  
from second\_window import \*  
  
  
**class Cinema(QMainWindow):**  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.ui = Ui\_Form()  
 self.ui.setupUi(self)  
  
 self.db = DataBase()  
 self.init\_db()  
 self.load\_hall\_cb()  
 self.load\_movie\_cb()  
 self.choose\_seat()  
 self.export\_to\_excel()  
  
 self.ui.hall\_cb.currentIndexChanged.connect(self.update\_session\_time)  
 self.ui.movie\_cb.currentIndexChanged.connect(self.update\_session\_time)  
  
 self.ui.query1\_btn.clicked.connect(self.query1)  
 self.ui.query2\_btn.clicked.connect(self.open\_win2)  
  
 def init\_db(self): . . .  
   
 **def load\_hall\_cb(self):**  
 *""" комбо бокс: зал"""* self.ui.hall\_cb.blockSignals(True) # Отключаем сигналы  
 self.cursor.execute("""select \* from hall """)  
 halls = self.cursor.fetchall()  
 self.ui.hall\_cb.clear()  
 for hall in halls:  
 self.ui.hall\_cb.addItem(str(hall[1]), hall[0])  
 self.ui.hall\_cb.blockSignals(False) # Включаем сигналы

**def get\_session\_time(self, start\_time, end\_time, duration, occupied\_times):**  
 *""" сгенерируем список времени сеансов на основе  
 начального времени, конечного, длительности, исключая уже занятые места"""* # start\_time, end\_time - временные границы расписания  
 # duration - продолжительность 1 сеанса (в минутах)  
 # occupied\_times - список уже занятых временных слотов  
  
 times = []  
 current\_time = start\_time # инициализация текущего времени с начальным временем  
  
 #проверяем, чтобы текущий сеанс заканчивался до end\_time  
 #timedelta - добавляет длительность сеанса к текущему времени  
 while current\_time + timedelta(minutes=duration) <= end\_time:  
 session\_str = current\_time.strftime("%H:%M")  
 if session\_str not in occupied\_times: # исключаем занятые слоты  
 times.append(session\_str)  
 current\_time += timedelta(minutes=duration + 30) # интервал между сеансами  
 return times

**def get\_occupied\_times(self, id\_hall, id\_movie):**  
 *"""получаем список уже занятых слотов"""* #distinct - выбирает уникальные временные слоты  
 try:  
 query = """ select distinct time   
 from sale   
 where id\_hall = ? and id\_movie = ?"""  
  
 self.cursor.execute(query, (id\_hall, id\_movie))  
 occupied\_times = [row[0] for row in self.cursor.fetchall()] #извлекает результаты  
 # и добавляет в список  
 return occupied\_times  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Не удалось получить занятые слоты: {str(e)}")  
 return []  
  
 **def update\_session\_time(self):** *"""обновляем комбобокс с доступным временем сеансов"""* try:  
 id\_hall = self.ui.hall\_cb.currentData()  
 id\_movie = self.ui.movie\_cb.currentData()  
  
 if not id\_hall or not id\_movie:  
 return  
  
 # параметры расписания  
 start\_time = datetime.strptime("10:00", "%H:%M")  
 end\_time = datetime.strptime("22:00", "%H:%M")  
 film\_duration = 120  
  
 occupied\_times = self.get\_occupied\_times(id\_hall, id\_movie)  
 session\_times = self.get\_session\_time(start\_time, end\_time, film\_duration, occupied\_times)  
  
 self.ui.time\_cb.clear()  
 if session\_times:  
 self.ui.time\_cb.addItems(session\_times)  
 else:  
 self.ui.time\_cb.addItem("нет доступного времени")  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Не удалось обновить время: {str(e)}")  
 print(e)  
  
 **def choose\_seat(self):**  
 # подключаем обработчики к кнопкам  
 #Обработчики: Используются лямбда - функции, чтобы передать  
 # место в качестве аргумента в book\_seat  
 self.ui.btn\_1a.clicked.connect(lambda: self.book\_seat("1A"))  
 self.ui.btn\_2a.clicked.connect(lambda: self.book\_seat("2A"))

. . .

**def book\_seat(self, seat):**  
 hall = self.ui.hall\_cb.currentData()  
 movie = self.ui.movie\_cb.currentData()  
 time = self.ui.time\_cb.currentText()  
  
 # Проверка заполнения полей  
 if not (hall and movie and time):  
 QtWidgets.QMessageBox.warning(None, "Ошибка", "Выберите зал, фильм и время!")  
 return  
  
 try:  
 self.cursor.execute("insert into sale (id\_movie, id\_hall, time, num\_seat) values (?, ?, ?, ?)", (hall, movie, time, seat))  
 self.conn.commit()  
 QMessageBox.information(self, "успех", f"Вы забронировали место {seat}" )  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка базы данных", str(e))  
  
 **def query1(self):** *""" запрос 1"""* results = self.db.query1()  
 if results:  
 message = "Процент наполнения зала: \n"  
 for result in results:  
 message += f"Зал {result[0]}, Сеанс {result[1]}: {result[4]}\n"  
 QMessageBox.information(self, "Процент наполнения", message)  
 else:  
 QMessageBox.critical(self, "Информация", "Нет данных")  
  
 **def open\_win2(self):**  
 *""" открытие 2 окна"""* result = self.db.query2()  
 if result:  
 genre, count = result[0]  
 text = f"Жанр: {genre}\nКоличество продаж: {count}"  
 else:  
 text = f"Нет данных о продажах."  
  
 self.win2 = Second\_window(self, text)  
 self.win2.show()  
 self.hide()  
 **def export\_to\_excel(self):**  
 try:  
 data = self.db.query1()  
 workbook = openpyxl.Workbook()  
 sheet = workbook.active  
 sheet.title = 'Процент наполненности залов'  
 sheet.append(["Id Зала", "Сеанс", "Продано билетов", "Вместимость зала", "Процент наполненности" ])  
  
 for row in data:  
 sheet.append(row)  
  
 excel\_file = 'cinema.xlsx'  
 workbook.save(excel\_file)  
 QMessageBox.information(self, "успех", f"данные сохранены в иксель {excel\_file}")  
  
 except Exception as e:  
 QMessageBox.critical(self, "ошибка", "ошибка при экспорте" )  
  
 **def closeEvent(self, a0): . . .**  
  
if \_\_name\_\_ == "\_\_main\_\_": .. .  
  
**class Second\_window(QtWidgets.QWidget):** def \_\_init\_\_(self, main\_window, text):  
 super().\_\_init\_\_()  
 self.ui = Ui\_win2()  
 self.ui.setupUi(self)  
 self.main\_window = main\_window  
 self.ui.query2\_plain\_text.setPlainText(text)  
 self.ui.back\_btn.clicked.connect(self.back\_btn)  
  
  
 def back\_btn(self):  
 self.main\_window.show()  
 self.close()

**class DataBase():**  
 def \_\_init\_\_(self):  
 self.conn = sqlite3.connect('cinema.db')  
 self.cursor = self.conn.cursor()  
  
 def create\_table(self):  
  
 self.cursor.executescript("""   
 insert or ignore into hall (idHall, capacity) values  
 (1, 200),   
 (2, 150),   
 (3, 100);  
   
 insert or ignore into movie (idMovie, name, genre) values  
 (1, 'Субстанция', 'хоррор'),  
 (2, 'Рай под ногами матерей', 'драма'),  
 (3, 'Как взломать экзамен', 'триллер'),  
 (4, 'Продавец на полставки', 'комедия');  
   
 insert or ignore into sale (idSale, id\_movie, id\_hall, time, num\_seat) values  
 (1, 2, 1, '2024-12-20 17:30', '7A'),  
 (2, 2, 1, '2024-12-20 17:30', '8A'),  
 (3, 2, 1, '2024-12-20 17:30', '9A'),  
 (4, 4, 3, '2024-12-21 22:00', '5C'),  
 (4, 4, 3, '2024-12-21 22:00', '9D'),  
 (5, 3, 2, '2024-12-22 18:30', '1B');  
 """)  
  
 self.conn.commit()  
 print('таблицы созданы')  
  
 **def query1(self):**  
 *"""запрос 1: Процент наполнения зала по сеансам """* query = """ select sale.id\_hall, sale.time, count(sale.num\_seat) as sold\_seat,  
 hall.capacity, round((count(sale.num\_seat) \* 100.0 / hall.capacity), 2) as percent\_fill  
 from sale  
 join hall on sale.id\_hall = hall.idHall  
 group by sale.id\_hall, sale.time"""  
  
 self.cursor.execute(query)  
 result = self.cursor.fetchall()  
 return result if result else 0  
  
 **def query2(self):**  
 *"""запрос 2: Самый популярный жанр """* query = """ select movie.genre, count(sale.idSale) as sale\_count  
 from sale  
 join movie on sale.id\_movie = movie.idMovie  
 group by movie.genre  
 order by sale\_count desc  
 limit 1 """  
  
 self.cursor.execute(query)  
 result = self.cursor.fetchall()  
 return result if result else 0  
  
 **def close\_db(self): . . .**

1. Автосалон. Менеджеры автосалона осуществляют продажу клиентам автомобилей различных марок.

Таблицы: Покупатели (Код покупателя, ФИО), Менеджеры (Код менеджера, ФИО), Автомобили (код, марка), Продажи (Код менеджера, код автомобиля, государственный номер, код покупателя, дата, цена).

Определить:

- среднюю сумму сделки

- долю продаж автомобилей разных марок.

**class DataBase:**  
 def \_\_init\_\_(self):  
 self.conn = sqlite3.connect(`autosalon.db`)  
 self.cursor = self.conn.cursor()  
  
 def create\_table(self):

. . .  
 self.conn.commit()  
  
 def insert\_buyers(self, name, surname, lastname):  
 self.cursor.execute(  
 "INSERT INTO buyers (name, surname, lastname) VALUES (?, ?, ?)",  
 (name, surname, lastname)  
 )  
 self.conn.commit()  
  
 def insert\_managers(self, name, surname, lastname):  
 self.cursor.execute(  
 "INSERT INTO managers (name, surname, lastname) VALUES (?, ?, ?)",  
 (name, surname, lastname)  
 )  
 self.conn.commit()  
  
 def insert\_cars(self, stamp):  
 self.cursor.execute(  
 "INSERT INTO cars (stamp) VALUES (?)",  
 (stamp,)  
 )  
 self.conn.commit()  
  
 def insert\_sales(self, idManager, idAuto, gos\_num, idBuyer, data, price):  
 self.cursor.execute(  
 "INSERT INTO sales (idManager, idAuto, gos\_num, idBuyer, data, price) VALUES (?, ?, ?, ?, ?, ?)",  
 (idManager, idAuto, gos\_num, idBuyer, data, price)  
 )  
 self.conn.commit()  
  
 **def get\_average\_sale(self):**  
 *"""Вычисляет среднюю сумму сделки."""* self.cursor.execute("SELECT AVG(price) FROM sales")  
 result = self.cursor.fetchone()  
 return result[0] if result and result[0] is not None else 0  
  
 **def get\_sales\_share\_by\_brand(self):**  
 *"""Вычисляет долю продаж автомобилей разных марок."""* query = """  
 SELECT cars.stamp, COUNT(sales.idSales) \* 1.0 / (SELECT COUNT(\*) FROM sales) AS share  
 FROM sales  
 JOIN cars ON sales.idAuto = cars.idCars  
 GROUP BY cars.stamp  
 """  
 self.cursor.execute(query)  
 return self.cursor.fetchall()  
  
 def close(self):  
 self.conn.commit()  
 self.cursor.close()  
 self.conn.close()

import sys  
from PyQt6.QtWidgets import QApplication, QMainWindow, QMessageBox, QWidget  
from PyQt6.QtGui import QStandardItemModel, QStandardItem  
from form1 import Ui\_MainWindow # Основная форма  
from form2 import Form # Вторая форма  
from salon import DataBase # Класс для работы с базой данных  
  
  
**class MyWindow(QMainWindow):** def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.ui = Ui\_MainWindow()  
 self.ui.setupUi(self)  
  
 self.db = DataBase()  
 self.db.create\_table()  
  
 # Подключение кнопок основной формы  
 self.ui.pushButton.clicked.connect(self.save\_db)  
 self.ui.pushButton\_2.clicked.connect(self.open\_form2)  
 self.ui.pushButton\_3.clicked.connect(self.show\_sales\_share)  
  
 # Создание формы 2  
 self.form2 = QMainWindow()  
 self.ui\_form2 = Form()  
 self.ui\_form2.setupUi(self.form2)  
  
 # Подключение кнопки "назад" на форме 2  
 self.ui\_form2.pushButton.clicked.connect(self.return\_to\_main)  
  
 **def save\_db(self):** try:  
 buyers\_name = self.ui.name\_b.text()  
 buyers\_surname = self.ui.surname\_b.text()  
 buyers\_lastname = self.ui.lastname\_b.text()  
 manager\_name = self.ui.name\_m.text()  
 manager\_surname = self.ui.surname\_m.text()  
 manager\_lastname = self.ui.lastaname\_m.text()  
 car\_stamp = self.ui.stamp.text()  
 sale\_number = self.ui.gos\_number.text()  
 sale\_price = self.ui.price.text()  
 **sale\_date = self.ui.dateEdit.date().toString("yyyy-MM-dd")**  
 if not all([buyers\_name, buyers\_surname, buyers\_lastname,  
 manager\_name, manager\_surname, manager\_lastname,  
 car\_stamp, sale\_number, sale\_price, sale\_date]):  
 QMessageBox.warning(self, "Ошибка", "Все поля должны быть заполнены")  
 return  
  
 self.db.insert\_buyers(buyers\_name, buyers\_surname, buyers\_lastname)  
 self.db.insert\_managers(manager\_name, manager\_surname, manager\_lastname)  
 self.db.insert\_cars(car\_stamp)  
  
 idManager = self.db.cursor.execute(  
 "SELECT idManagers FROM managers WHERE name = ? AND surname = ? AND lastname = ?",  
 (manager\_name, manager\_surname, manager\_lastname)  
 ).fetchone()[0]  
  
 idBuyer = self.db.cursor.execute(  
 "SELECT idBuyers FROM buyers WHERE name = ? AND surname = ? AND lastname = ?",  
 (buyers\_name, buyers\_surname, buyers\_lastname)  
 ).fetchone()[0]  
  
 idAuto = self.db.cursor.execute(  
 "SELECT idCars FROM cars WHERE stamp = ?",  
 (car\_stamp,)  
 ).fetchone()[0]  
  
 self.db.insert\_sales(idManager, idAuto, sale\_number, idBuyer, sale\_date, int(sale\_price))  
  
 QMessageBox.information(self, "Успех", "Данные успешно сохранены!")  
  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Произошла ошибка: {e}")  
  
 **def open\_form2(self):**  
 try:  
 # Получение средней суммы сделки  
 average\_sale = self.db.get\_average\_sale()  
  
 # Отображение данных в ListView формы 2  
 model = QStandardItemModel(self.ui\_form2.listView)  
 item = QStandardItem(f"Средняя сумма сделки: {average\_sale}")  
 model.appendRow(item)  
 self.ui\_form2.listView.setModel(model)  
  
 # Показ формы 2  
 self.form2.show()  
 self.hide()  
  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Произошла ошибка: {e}")  
  
 **def show\_sales\_share(self):**  
 try:  
 sales\_share = self.db.get\_sales\_share\_by\_brand()  
 result = "Доля продаж автомобилей по маркам:\n"  
 for stamp, share in sales\_share:  
 result += f"{stamp}: {share:.2%}\n"  
 QMessageBox.information(self, "Доля продаж автомобилей", result)  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Произошла ошибка: {e}")  
  
 **def return\_to\_main(self):** *"""Возвращение к основной форме."""* self.form2.hide()  
 self.show()  
  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
. . .

18V. *Туристский клуб. Туристский клуб организует одно – и многодневные пешеходные туры в  
группах с руководителем по разным маршрутам и разной категории сложности.  
Таблицы: Маршруты (Код маршрута, название, продолжительность,  
категория сложности),Руководители (Код руководителя, ФИО, телефон),  
Группы (Код группы, название группы, код маршрута, код руководителя,  
количество туристов, дата отправления).  
Требуется:- определить перечень групп на маршруте; - построить сравнительную диаграмму количества туристов по уровню  
сложности маршрутов.*

**class Turist(QMainWindow):** def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 self.ui = Ui\_Form()  
 self.ui.setupUi(self)  
  
 self.insert\_cb\_routes()  
 self.insert\_cb\_directors()  
  
 self.ui.Button\_save.clicked.connect(self.save\_group)  
 self.ui.Button\_query1.clicked.connect(self.open\_window2)  
 self.ui.Button\_query2.clicked.connect(self.query2)  
  
 **def insert\_cb\_routes(self):**  
 cursor.execute("SELECT id\_route, name\_route FROM route")  
 routes = cursor.fetchall()  
 for route in routes:  
 self.ui.comboBox\_route.addItem(route[1], route[0])  
 **def save\_group(self):**  
 name\_group = self.ui.lineEdit\_name.text()  
 id\_routes = self.ui.comboBox\_route.currentData()  
 id\_directors = self.ui.comboBox\_director.currentData()  
 count\_turists = self.ui.lineEdit\_count.text()  
 selected\_date = self.ui.dateEdit.date().toString("yyyy-MM-dd")  
  
 if name\_group and selected\_date and count\_turists.isdigit(): #если все полня заполнены и count\_turists - число, то...  
 try:  
 cursor.execute(  
 "INSERT INTO group\_tour (name\_group, id\_route, id\_director, number\_of\_tourists, departure\_date) values (%s, %s, %s, %s, %s)",  
 (name\_group, id\_routes, id\_directors, int(count\_turists), selected\_date)) #из кода сверху  
 connect.commit()  
 QMessageBox.information(self, 'Успех', 'Все поля заполнены')  
 except MySQLdb.Error as e:  
 QMessageBox.warning(self, 'ошибка', f'не получилось добавить событие: {e}')  
 else:  
 QMessageBox.warning(self, 'ошибка', 'заполните все поля корректно.')  
  
 **def query1(self):**  
 try:  
 query = ("SELECT group\_tour.name\_group, route.name\_route FROM group\_tour "  
 "JOIN route ON group\_tour.id\_route = route.id\_route")  
 cursor.execute(query)  
 groups = cursor.fetchall()  
  
 if not groups:  
 return "На данный момент нет групп на маршруте"  
 else:  
 result = "Группа на маршруте:\n"  
 for group in groups:  
 result += f"Группа: {group[0]}, Маршрут: {group[1]}\n"  
 return result  
 except MySQLdb.Error as e:  
 QMessageBox.warning(self, 'Ошибка', f'Ошибка при выполнении запроса: {e}')  
 return None  
  
 **def query2(self):**  
 try:  
 query = ("SELECT SUM(group\_tour.number\_of\_tourists) as total\_tourist, route.difficult from "  
 "group\_tour join route on route.id\_route = group\_tour.id\_route group by route.difficult")  
 cursor.execute(query)  
 count\_tourists = cursor.fetchall()  
  
 if not count\_tourists:  
 QMessageBox.information(self, 'Сравнение туристов', 'На данный момент нет туристов в группах')  
 else:  
 result = "Туристы: \n"  
 for count\_tourist in count\_tourists:  
 result += f"Группа: {count\_tourist[0]}, Маршрут: {count\_tourist[1]} \n"  
 QMessageBox.information(self, 'Сравнение туристов', result)  
 except MySQLdb.Error as e:  
 QMessageBox.warning(self, 'ошибка', f'Ошибка при выполнении запроса {e}')

**def open\_window2(self):**  
 # Метод для открытия второго окна с результатами первого запроса  
 results = self.query1() # Получаем результаты первого запроса  
 window2 = QDialog() # Создаем новое диалоговое окно  
 win2 = Ui\_Form2() # Инициализируем интерфейс второго окна  
 win2.setupUi(window2) # Устанавливаем интерфейс в диалоговом окне  
  
 if results:  
 win2.plainTextEdit.setPlainText(results) # Выводим результаты в текстовое поле  
 # Связываем кнопку "Назад" с методом закрытия окна  
 win2.Button\_back.clicked.connect(window2.close)  
 window2.exec() # Запускаем диалоговое окно  
  
if \_\_name\_\_ == "\_\_main\_\_":

1. Автовокзал. Автобусы производят рейсы до станций по расписанию. Каждый автобус вмещает не более определенного количества пассажиров

Таблицы: станции (код станции, название станции), автобусы (код автобуса, марка автобуса, государственный номер, вместимость), рейсы (код рейса, код станции, код автобуса, время отправления).

Определить: - сколько выполняется рейсов до каждой станции? - каково общее количество пассажиров?

from PyQt6.QtWidgets import QApplication, QMainWindow, QMessageBox  
from form1 import Ui\_MainWindow  
from auto import DataBase  
  
import sqlite3  
  
**class MyWindow(QMainWindow):**  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.ui = Ui\_MainWindow()  
 self.ui.setupUi(self)  
  
 # Инициализация базы данных  
 self.db = DataBase()  
 self.db.create\_table()

# Подключение кнопок  
 self.ui.pushButton.clicked.connect(self.save\_to\_db)  
  
 **def save\_to\_db(self):**  
 try:  
 # Получение данных из полей ввода  
 bus\_brand = self.ui.lineEdit\_2.text()  
 bus\_num = self.ui.lineEdit\_3.text()  
 bus\_capacity = self.ui.lineEdit\_4.text()  
 station\_name = self.ui.lineEdit.text()  
 departure\_time = self.ui.timeEdit.time().toString("HH:mm")  
 # Проверка на заполненность  
 if not (bus\_brand and bus\_num and bus\_capacity and station\_name and departure\_time):  
 QMessageBox.warning(self, "Ошибка", "Все поля должны быть заполнены!")  
 return  
  
 # Сохранение данных в таблицы  
 self.db.insert\_bus(bus\_brand, bus\_num, int(bus\_capacity))  
 self.db.insert\_station(station\_name)  
  
 # Получение внешних ключей  
 self.db.cursor.execute("SELECT idBus FROM buses WHERE brand = ? AND num = ?", (bus\_brand, bus\_num))  
 id\_bus = self.db.cursor.fetchone()[0]  
  
 self.db.cursor.execute("SELECT idStation FROM station WHERE name = ?", (station\_name,))  
 id\_station = self.db.cursor.fetchone()[0]  
  
 # Сохранение маршрута  
 self.db.insert\_route(id\_station, id\_bus, departure\_time)  
  
 # Вывод результатов SQL-запросов  
 total\_routes = self.db.get\_total\_routes()  
 passengers = self.db.total\_passengers()  
  
 route\_msg = "\n".join([f"Станция: {row[0]}, Рейсов: {row[1]}" for row in total\_routes])  
 QMessageBox.information(self, "Информация о рейсах",  
 f"{route\_msg}\nОбщее количество пассажиров: {passengers}")  
  
 # Успех  
 QMessageBox.information(self, "Успех", "Данные успешно сохранены!")  
 #self.clear\_fields()  
  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Произошла ошибка: {e}")  
  
  
# Основная программа  
if \_\_name\_\_ == "\_\_main\_\_": . . .

import sqlite3  
import os  
  
**class DataBase:**  
 def \_\_init\_\_(self):  
 db\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'auto.db')  
 self.conn = sqlite3.connect(db\_path)  
 self.cursor = self.conn.cursor()  
  
 def create\_table(self):  
 # Создание таблиц  
 self.conn.commit()  
  
 def insert\_bus(self, brand, num, capacity):  
 self.cursor.execute("INSERT INTO buses (brand, num, capacity) VALUES (?, ?, ?)", (brand, num, capacity))  
 self.conn.commit()  
  
 def insert\_station(self, name):  
 self.cursor.execute("INSERT INTO station (name) VALUES (?)", (name,))  
 self.conn.commit()  
  
 def insert\_route(self, id\_station, id\_bus, departure\_time):  
 self.cursor.execute("INSERT INTO routes (id\_station, id\_bus, departure\_time) VALUES (?, ?, ?)",  
 (id\_station, id\_bus, departure\_time))  
 self.conn.commit()  
  
  
 def get\_total\_routes(self):  
 cursor = self.conn.cursor()  
 cursor.execute("""SELECT station.name, COUNT(routes.idRoute) AS trip\_count   
 FROM station   
 LEFT JOIN routes ON station.idStation = routes.id\_station   
 GROUP BY station.name;""")  
 return cursor.fetchall()  
  
 def total\_passengers(self):  
 cursor = self.conn.cursor()  
 cursor.execute('''SELECT SUM(buses.capacity) AS total\_passengers   
 FROM routes   
 JOIN buses ON routes.id\_bus = buses.idBus;''')  
 return cursor.fetchone()[0]  
  
 def close(self):  
 self.conn.commit()  
 self.cursor.close()  
 self.conn.close()  
  
  
  
# Использование класса  
db = DataBase()  
db.create\_table()  
db.close()