Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО–КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №12
дисциплины
«Объектно-ориентированное программирование»
Вариант 13

Выполнил:
Рябинин Егор Алексеевич
3 курс, группа ИВТ-б-о-23-2,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Проверил:
Доцент департамента цифровых, робототехнических систем и электроники института перспективной инженерии
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г

Тема: Взаимодействие с базами данных SQLite3 с помощью языка программирования Python.

Цель: приобретение навыков взаимодействия с базами данных SQLite3 с помощью языка программирования Python версии 3.13.3.

Порядок выполнения работы:

Ссылка на репозиторий:

https://github.com/bohemiaaaaa/Lab12_Object-oriented-programming

Задание №1. Во всех заданиях требуется разработать консольное приложение на Python, реализующее хранение, обработку и отображение структурированных данных. Каждая программа должна использовать объектно-ориентированный подход, а описания основных сущностей необходимо выполнить с помощью декоратора @dataclass. Все данные должны сохраняться в базе SQLite, при этом структура базы должна включать не менее двух таблиц, связанных внешним ключом или логической зависимостью. Приложение должно автоматически создавать базу данных и её таблицы при первом запуске, если файл базы отсутствует.

Работу с базой данных следует инкапсулировать в отдельном классе-репозитории, содержащем методы для добавления, выборки и обработки данных. Логику формирования SQL-запросов, а также создание и настройку соединения со SQLite рекомендуется разместить внутри этого класса. Приложение должно обеспечивать корректное преобразование строк таблиц в объекты соответствующих датаклассов и обратно.

Взаимодействие пользователя с программой необходимо организовать через интерфейс командной строки с использованием модуля argparse или click. Каждая программа должна поддерживать подкоманды для добавления новых объектов, отображения всех записей, выполнения выборок по определённым критериям и указания пути к файлу базы. Должна быть предусмотрена команда для вывода справки и, при необходимости, информации о версии программы.

Вывод данных в консоль следует оформлять в аккуратном табличном виде или в ином структурированном формате, обеспечивающем удобство чтения. Все операции должны выполняться устойчиво к ошибкам отсутствующих данных или пустых выборок. Структура кода должна быть модульной и расширяемой.

13. Управление авиарейсами.

Разработать систему авиарейсов: таблицы flights и airports. Реализовать добавление рейса, аэропорта, выборку рейсов по аэропорту назначения.

Листинг программы 1:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import argparse
import sqlite3
from dataclasses import dataclass
from pathlib import Path


@dataclass
class Airport:
    code: str
    name: str
    city: str


@dataclass
class Flight:
    number: str
    departure_airport: str
    arrival_airport: str
    departure_time: str
    arrival_time: str


class FlightRepository:
    def __init__(self, db_path: Path):
        self.db_path = db_path
        self._create_tables()

    def _connect(self):
        return sqlite3.connect(self.db_path)

    def _create_tables(self):
        conn = self._connect()
        cursor = conn.cursor()

        cursor.execute(
            """
            CREATE TABLE IF NOT EXISTS airports (
                code TEXT PRIMARY KEY,
                name TEXT NOT NULL,
                city TEXT NOT NULL
            )
```

```
            """
        )

        cursor.execute(
            """
            CREATE TABLE IF NOT EXISTS flights (
                number TEXT PRIMARY KEY,
                departure_airport TEXT NOT NULL,
                arrival_airport TEXT NOT NULL,
                departure_time TEXT NOT NULL,
                arrival_time TEXT NOT NULL,
                FOREIGN KEY(departure_airport) REFERENCES airports(code),
                FOREIGN KEY(arrival_airport) REFERENCES airports(code)
            )
            """
        )

        conn.commit()
        conn.close()

    def add_airport(self, code: str, name: str, city: str) -> None:
        conn = self._connect()
        cursor = conn.cursor()

        cursor.execute(
            """
            INSERT INTO airports (code, name, city)
            VALUES (?, ?, ?)
            """,
            (code, name, city),
        )

        conn.commit()
        conn.close()

    def add_flight(
        self,
        number: str,
        departure_airport: str,
        arrival_airport: str,
        departure_time: str,
        arrival_time: str,
    ) -> None:
        conn = self._connect()
        cursor = conn.cursor()

        cursor.execute(
            """
            INSERT INTO flights (
                number, departure_airport, arrival_airport,
                departure_time, arrival_time
            )
            VALUES (?, ?, ?, ?, ?)
            """,
            (number, departure_airport, arrival_airport, departure_time,
arrival_time),
        )

        conn.commit()
        conn.close()

    def get_all_flights(self) -> list[Flight]:
        conn = self._connect()
        cursor = conn.cursor()
```

```
        cursor.execute(
            """
            SELECT
                f.number,
                f.departure_airport,
                f.arrival_airport,
                f.departure_time,
                f.arrival_time
            FROM flights f
            """
        )

        rows = cursor.fetchall()
        conn.close()

        return [
            Flight(
                number=row[0],
                departure_airport=row[1],
                arrival_airport=row[2],
                departure_time=row[3],
                arrival_time=row[4],
            )
            for row in rows
        ]

    def get_flights_by_destination(self, airport_code: str) -> list[Flight]:
        conn = self._connect()
        cursor = conn.cursor()

        cursor.execute(
            """
            SELECT
                f.number,
                f.departure_airport,
                f.arrival_airport,
                f.departure_time,
                f.arrival_time
            FROM flights f
            WHERE f.arrival_airport = ?
            """,
            (airport_code,),
        )

        rows = cursor.fetchall()
        conn.close()

        return [
            Flight(
                number=row[0],
                departure_airport=row[1],
                arrival_airport=row[2],
                departure_time=row[3],
                arrival_time=row[4],
            )
            for row in rows
        ]

    def get_all_airports(self) -> list[Airport]:
        conn = self._connect()
        cursor = conn.cursor()

        cursor.execute(
```

```
            """
            SELECT code, name, city
            FROM airports
            """
        )

        rows = cursor.fetchall()
        conn.close()

        return [Airport(code=row[0], name=row[1], city=row[2]) for row in
rows]


def display_flights(flights: list[Flight]) -> None:
    if not flights:
        print("Список рейсов пуст.")
        return

    line = "+{}+{}+{}+{}+{}+".format("-" * 10, "-" * 20, "-" * 20, "-" * 16,
"-" * 16)

    print(line)
    print(
        "|{:^10}|{:^20}|{:^20}|{:^16}|{:^16}|".format(
            "Номер",
            "Аэропорт вылета",
            "Аэропорт прибытия",
            "Время вылета",
            "Время прибытия",
        )
    )
    print(line)

    for flight in flights:
        print(
            "|{:<10}|{:<20}|{:<20}|{:<16}|{:<16}|".format(
                flight.number,
                flight.departure_airport,
                flight.arrival_airport,
                flight.departure_time,
                flight.arrival_time,
            )
        )

    print(line)


def display_airports(airports: list[Airport]) -> None:
    if not airports:
        print("Список аэропортов пуст.")
        return

    line = "+{}+{}+{}+".format("-" * 6, "-" * 30, "-" * 20)

    print(line)
    print("|{:^6}|{:^30}|{:^20}|".format("Код", "Название", "Город"))
    print(line)

    for airport in airports:
        print("|{:<6}|{:<30}|{:<20}|".format(airport.code, airport.name,
airport.city))

    print(line)
```

```python
def main():
    parser = argparse.ArgumentParser(
        description="Управление авиарейсами",
        formatter_class=argparse.RawDescriptionHelpFormatter,
    )

    parser.add_argument(
        "--db",
        default="airports.db",
        help="Путь к файлу базы данных (по умолчанию: airports.db)",
    )

    subparsers = parser.add_subparsers(dest="command", help="Команды")

    # Команда добавления аэропорта
    airport_parser = subparsers.add_parser(
        "add-airport", help="Добавить новый аэропорт"
    )
    airport_parser.add_argument("--code", required=True, help="Код
аэропорта")
    airport_parser.add_argument("--name", required=True, help="Название
аэропорта")
    airport_parser.add_argument("--city", required=True, help="Город")

    # Команда добавления рейса
    flight_parser = subparsers.add_parser("add-flight", help="Добавить новый
рейс")
    flight_parser.add_argument("--number", required=True, help="Номер рейса")
    flight_parser.add_argument(
        "--departure", required=True, help="Код аэропорта вылета"
    )
    flight_parser.add_argument(
        "--arrival", required=True, help="Код аэропорта прибытия"
    )
    flight_parser.add_argument(
        "--departure-time",
        required=True,
        help="Время вылета (Формат: ГГГГ-ММ-ДД ЧЧ:ММ)",
    )
    flight_parser.add_argument(
        "--arrival-time",
        required=True,
        help="Время прибытия (Формат: ГГГГ-ММ-ДД ЧЧ:ММ)",
    )

    # Команда отображения всех рейсов.
    subparsers.add_parser("show-flights", help="Показать все рейсы")

    # Команда отображения всех аэропортов
    subparsers.add_parser("show-airports", help="Показать все аэропорты")

    # Команда выборки рейсов по аэропорту назначения
    select_parser = subparsers.add_parser(
        "select-by-destination", help="Выборка рейсов по аэропорту
назначения"
    )
    select_parser.add_argument(
        "--airport", required=True, help="Код аэропорта назначения"
    )

    args = parser.parse_args()

    repo = FlightRepository(Path(args.db))
```

```
    if args.command == "add-airport":
        repo.add_airport(args.code, args.name, args.city)
        print(f"Аэропорт {args.code} добавлен.")

    elif args.command == "add-flight":
        repo.add_flight(
            args.number,
            args.departure,
            args.arrival,
            args.departure_time,
            args.arrival_time,
        )
        print(f"Рейс {args.number} добавлен.")

    elif args.command == "show-flights":
        flights = repo.get_all_flights()
        display_flights(flights)

    elif args.command == "show-airports":
        airports = repo.get_all_airports()
        display_airports(airports)

    elif args.command == "select-by-destination":
        flights = repo.get_flights_by_destination(args.airport)
        if flights:
            print(f"Рейсы с прибытием в аэропорт {args.airport}:")
            display_flights(flights)
        else:
            print(f"Рейсов с прибытием в аэропорт {args.airport} не
найдено.")

    else:
        parser.print_help()


if __name__ == "__main__":
    main()
```

Результаты работы программы:

```
PS C:\Users\4isto\OOP_lab12> python tasks/task1.py add-airport --code LED --name "Пулково" --city "Санкт-Петербург"
Аэропорт LED добавлен.
PS C:\Users\4isto\OOP_lab12> python tasks/task1.py add-airport --code DME --name "Домодедово" --city "Москва"
Аэропорт DME добавлен.
```

Рисунок 1 – Создание и добавление аэропортов

```
PS C:\Users\4isto\OOP_lab12> python tasks/task1.py add-flight --number SU300 --departure SVO --arrival DME --departure-time "2024-05-20 16:00" --arrival-time "2024-05-20 16:45"
Рейс SU300 добавлен.
PS C:\Users\4isto\OOP_lab12> python tasks/task1.py add-flight --number SU100 --departure SVO --arrival LED --departure-time "2024-05-20 10:00" --arrival-time "2024-05-20 11:30"
Рейс SU100 добавлен.
```

Рисунок 2 – Создание и добавление рейсов

```
PS C:\Users\4isto\OOP_lab12> python tasks/task1.py select-by-destination --airport DME
Рейсы с прибытием в аэропорт DME:
+---------+--------------------+--------------------+----------------+----------------+
| Номер   | Аэропорт вылета    | Аэропорт прибытия  | Время вылета   | Время прибытия |
+---------+--------------------+--------------------+----------------+----------------+
|SU300    |SVO                 |DME                 |2024-05-20 16:00|2024-05-20 16:45|
+---------+--------------------+--------------------+----------------+----------------+
```

Рисунок 3 – Выборка рейсов

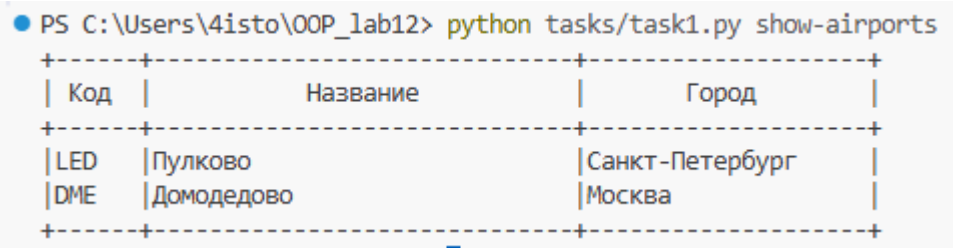

Рисунок 4 – Просмотр всех аэропортов

Задание №2. Изучите, как работать с базами данных SQLite с помощью пакета sqlalchemy. После этого возьмите ваш вариант задания №1 и выполните его заново, заменив всю работу с базой данных на механизм, предоставляемый SQLAlchemy. Ваша программа должна использовать модели, описанные через SQLAlchemy, связи между таблицами и сессию для выполнения операций с данными. Это задание относится к заданиям повышенной сложности и предполагает самостоятельное изучение нового инструмента.

Листинг программы 2:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import argparse
from datetime import datetime
from pathlib import Path

from sqlalchemy import Column, DateTime, ForeignKey, String, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import Session, relationship, sessionmaker

Base = declarative_base()


class Airport(Base):
    __tablename__ = "airports"

    code = Column(String, primary_key=True)
    name = Column(String, nullable=False)
    city = Column(String, nullable=False)

    departing_flights = relationship(
        "Flight",
        foreign_keys="Flight.departure_airport_code",
        back_populates="departure_airport",
    )
    arriving_flights = relationship(
        "Flight",
        foreign_keys="Flight.arrival_airport_code",
        back_populates="arrival_airport",
    )
```

```
class Flight(Base):
    __tablename__ = "flights"

    number = Column(String, primary_key=True)
    departure_airport_code = Column(String, ForeignKey("airports.code"),
nullable=False)
    arrival_airport_code = Column(String, ForeignKey("airports.code"),
nullable=False)
    departure_time = Column(DateTime, nullable=False)
    arrival_time = Column(DateTime, nullable=False)

    departure_airport = relationship(
        "Airport",
        foreign_keys=[departure_airport_code],
        back_populates="departing_flights",
    )
    arrival_airport = relationship(
        "Airport",
        foreign_keys=[arrival_airport_code],
        back_populates="arriving_flights",
    )


class FlightRepository:
    def __init__(self, db_path: Path):
        self.db_path = db_path
        self.engine = create_engine(f"sqlite:///{db_path}")
        Base.metadata.create_all(self.engine)
        self.Session = sessionmaker(bind=self.engine)

    def _get_session(self) -> Session:
        return self.Session()

    def add_airport(self, code: str, name: str, city: str) -> None:
        with self._get_session() as session:
            airport = Airport(code=code, name=name, city=city)
            session.add(airport)
            session.commit()

    def add_flight(
        self,
        number: str,
        departure_airport_code: str,
        arrival_airport_code: str,
        departure_time_str: str,
        arrival_time_str: str,
    ) -> None:
        with self._get_session() as session:
            departure_time = datetime.strptime(departure_time_str, "%Y-%m-%d
%H:%M")
            arrival_time = datetime.strptime(arrival_time_str, "%Y-%m-%d
%H:%M")

            flight = Flight(
                number=number,
                departure_airport_code=departure_airport_code,
                arrival_airport_code=arrival_airport_code,
                departure_time=departure_time,
                arrival_time=arrival_time,
            )
            session.add(flight)
            session.commit()

    def get_all_flights(self):
```

```
        with self._get_session() as session:
            flights = session.query(Flight).all()
            return flights

    def get_flights_by_destination(self, airport_code: str):
        with self._get_session() as session:
            flights = (
                session.query(Flight)
                .filter(Flight.arrival_airport_code == airport_code)
                .all()
            )
            return flights

    def get_all_airports(self):
        with self._get_session() as session:
            airports = session.query(Airport).all()
            return airports


def display_flights(flights: list[Flight]) -> None:
    if not flights:
        print("Список рейсов пуст.")
        return

    line = "+{}+{}+{}+{}+{}+".format("-" * 10, "-" * 20, "-" * 20, "-" * 16,
"-" * 16)

    print(line)
    print(
        "|{:^10}|{:^20}|{:^20}|{:^16}|{:^16}|".format(
            "Номер",
            "Аэропорт вылета",
            "Аэропорт прибытия",
            "Время вылета",
            "Время прибытия",
        )
    )
    print(line)

    for flight in flights:
        departure_time = flight.departure_time.strftime("%Y-%m-%d %H:%M")
        arrival_time = flight.arrival_time.strftime("%Y-%m-%d %H:%M")

        print(
            "|{:<10}|{:<20}|{:<20}|{:<16}|{:<16}|".format(
                flight.number,
                flight.departure_airport_code,
                flight.arrival_airport_code,
                departure_time,
                arrival_time,
            )
        )

    print(line)


def display_airports(airports: list[Airport]) -> None:
    if not airports:
        print("Список аэропортов пуст.")
        return

    line = "+{}+{}+{}+".format("-" * 6, "-" * 30, "-" * 20)

    print(line)
```

```
    print("|{:^6}|{:^30}|{:^20}|".format("Код", "Название", "Город"))
    print(line)

    for airport in airports:
        print("|{:<6}|{:<30}|{:<20}|".format(airport.code, airport.name,
airport.city))

    print(line)


def main():
    parser = argparse.ArgumentParser(
        description="Управление авиарейсами (SQLAlchemy)",
        formatter_class=argparse.RawDescriptionHelpFormatter,
    )

    parser.add_argument(
        "--db",
        default="airports_sa.db",
        help="Путь к файлу базы данных (по умолчанию: airports_sa.db)",
    )

    subparsers = parser.add_subparsers(dest="command", help="Команды")

    airport_parser = subparsers.add_parser(
        "add-airport", help="Добавить новый аэропорт"
    )
    airport_parser.add_argument("--code", required=True, help="Код
аэропорта")
    airport_parser.add_argument("--name", required=True, help="Название
аэропорта")
    airport_parser.add_argument("--city", required=True, help="Город")

    flight_parser = subparsers.add_parser("add-flight", help="Добавить новый
рейс")
    flight_parser.add_argument("--number", required=True, help="Номер рейса")
    flight_parser.add_argument(
        "--departure", required=True, help="Код аэропорта вылета"
    )
    flight_parser.add_argument(
        "--arrival", required=True, help="Код аэропорта прибытия"
    )
    flight_parser.add_argument(
        "--departure-time",
        required=True,
        help="Время вылета (Формат: ГГГГ-ММ-ДД ЧЧ:ММ)",
    )
    flight_parser.add_argument(
        "--arrival-time",
        required=True,
        help="Время прибытия (Формат: ГГГГ-ММ-ДД ЧЧ:ММ)",
    )

    subparsers.add_parser("show-flights", help="Показать все рейсы")

    subparsers.add_parser("show-airports", help="Показать все аэропорты")

    select_parser = subparsers.add_parser(
        "select-by-destination", help="Выборка рейсов по аэропорту
назначения"
    )
    select_parser.add_argument(
        "--airport", required=True, help="Код аэропорта назначения"
    )
```

```
    args = parser.parse_args()

    repo = FlightRepository(Path(args.db))

    if args.command == "add-airport":
        repo.add_airport(args.code, args.name, args.city)
        print(f"Аэропорт {args.code} добавлен.")

    elif args.command == "add-flight":
        repo.add_flight(
            args.number,
            args.departure,
            args.arrival,
            args.departure_time,
            args.arrival_time,
        )
        print(f"Рейс {args.number} добавлен.")

    elif args.command == "show-flights":
        flights = repo.get_all_flights()
        display_flights(flights)

    elif args.command == "show-airports":
        airports = repo.get_all_airports()
        display_airports(airports)

    elif args.command == "select-by-destination":
        flights = repo.get_flights_by_destination(args.airport)
        if flights:
            print(f"Рейсы с прибытием в аэропорт {args.airport}:")
            display_flights(flights)
        else:
            print(f"Рейсов с прибытием в аэропорт {args.airport} не
найдено.")

    else:
        parser.print_help()


if __name__ == "__main__":
    main()
```

Результаты работы программы:

```
PS C:\Users\4isto\OOP_lab12> python tasks/task2.py add-airport --code LED --name "Пулково" --city "Санкт-Петербург"
C:\Users\4isto\OOP_lab12\tasks\task2.py:13: MovedIn20Warning: The ``declarative_base()`` function is now available as s
n SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  Base = declarative_base()
Аэропорт LED добавлен.
PS C:\Users\4isto\OOP_lab12> python tasks/task2.py add-airport --code DME --name "Домодедово" --city "Москва"
C:\Users\4isto\OOP_lab12\tasks\task2.py:13: MovedIn20Warning: The ``declarative_base()`` function is now available as s
n SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  Base = declarative_base()
Аэропорт DME добавлен.
PS C:\Users\4isto\OOP_lab12> python tasks/task2.py add-airport --code JFK --name "John F. Kennedy" --city "New York"
C:\Users\4isto\OOP_lab12\tasks\task2.py:13: MovedIn20Warning: The ``declarative_base()`` function is now available as s
n SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  Base = declarative_base()
Аэропорт JFK добавлен.
PS C:\Users\4isto\OOP_lab12> python tasks/task2.py show-airports
C:\Users\4isto\OOP_lab12\tasks\task2.py:13: MovedIn20Warning: The ``declarative_base()`` function is now available as s
n SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  Base = declarative_base()
+------+------------------------------+--------------------+
| Код  |           Название           |       Город        |
+------+------------------------------+--------------------+
|LED   |Пулково                       |Санкт-Петербург     |
|DME   |Домодедово                    |Москва              |
|JFK   |John F. Kennedy               |New York            |
+------+------------------------------+--------------------+
```

Рисунок 1 – Добавление аэропортов и их просмотр

```
PS C:\Users\4isto\OOP_lab12> python tasks/task2.py add-flight --number SU200 --departure LED --arrival DME --departure-time "2024-05-20 14:00" --arrival-time "2024-05-20 15:30"
C:\Users\4isto\OOP_lab12\tasks\task2.py:13: MovedIn20Warning: The ``declarative_base()`` function is now available as sqlalchemy.orm.declarative_base(). (deprecated since: 2.0)
n SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  Base = declarative_base()
Рейс SU200 добавлен.
PS C:\Users\4isto\OOP_lab12> python tasks/task2.py add-flight --number SU300 --departure SVO --arrival DME --departure-time "2024-05-20 16:00" --arrival-time "2024-05-20 16:45"
C:\Users\4isto\OOP_lab12\tasks\task2.py:13: MovedIn20Warning: The ``declarative_base()`` function is now available as sqlalchemy.orm.declarative_base(). (deprecated since: 2.0)
n SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  Base = declarative_base()
Рейс SU300 добавлен.
PS C:\Users\4isto\OOP_lab12> python tasks/task2.py show-flights
C:\Users\4isto\OOP_lab12\tasks\task2.py:13: MovedIn20Warning: The ``declarative_base()`` function is now available as sqlalchemy.orm.declarative_base(). (deprecated since: 2.0)
n SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  Base = declarative_base()
+---------+----------------+-----------------+---------------+---------------+
| Номер   | Аэропорт вылета | Аэропорт прибытия | Время вылета | Время прибытия |
+---------+----------------+-----------------+---------------+---------------+
|SU200    |LED             |DME              |2024-05-20 14:00|2024-05-20 15:30|
|SU300    |SVO             |DME              |2024-05-20 16:00|2024-05-20 16:45|
+---------+----------------+-----------------+---------------+---------------+
```

Рисунок 2 – Добавление рейсов и их просмотр

```
PS C:\Users\4isto\OOP_lab12> python tasks/task2.py select-by-destination --airport DME
C:\Users\4isto\OOP_lab12\tasks\task2.py:13: MovedIn20Warning: The ``declarative_base()`` f
n SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  Base = declarative_base()
Рейсы с прибытием в аэропорт DME:
+----------+--------------------+--------------------+----------------+----------------+
|  Номер   |  Аэропорт вылета   | Аэропорт прибытия  |  Время вылета  | Время прибытия |
+----------+--------------------+--------------------+----------------+----------------+
|SU200     |LED                 |DME                 |2024-05-20 14:00|2024-05-20 15:30|
|SU300     |SVO                 |DME                 |2024-05-20 16:00|2024-05-20 16:45|
+----------+--------------------+--------------------+----------------+----------------+
```

Рисунок 3 – Выборка по значению

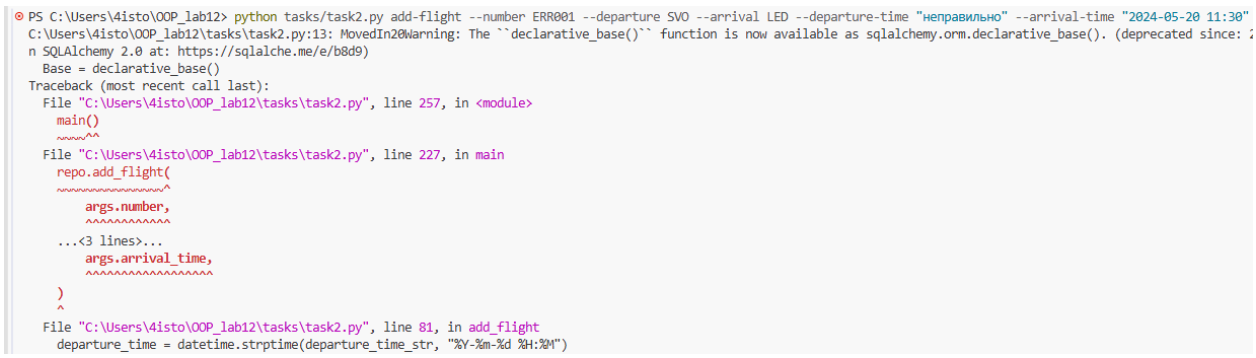

Рисунок 4 – Обработка ошибки

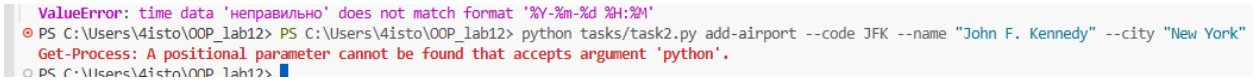

Рисунок 5 – Ошибка при добавлении уже существующей записи

Тесты для написанных программ:

Листинг программы 3:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import os
import tempfile
from pathlib import Path

import pytest
from task1 import Airport, Flight, FlightRepository


class TestFlightRepositoryV1:
    @pytest.fixture
    def temp_db_path(self):
        fd, db_path = tempfile.mkstemp(suffix=".db")
        os.close(fd)
        yield Path(db_path)
        try:
            if os.path.exists(db_path):
                os.unlink(db_path)
        except (OSError, PermissionError):
            pass

    @pytest.fixture
    def repo(self, temp_db_path):
        return FlightRepository(temp_db_path)

    def test_create_tables(self, repo):
        repo.add_airport("TEST", "Тестовый аэропорт", "Тестовый город")
        airports = repo.get_all_airports()
        assert len(airports) == 1
        assert airports[0].code == "TEST"

    def test_add_airport(self, repo):
        repo.add_airport("SVO", "Шереметьево", "Москва")
        airports = repo.get_all_airports()
        assert len(airports) == 1
        airport = airports[0]
        assert airport.code == "SVO"
```

```
        assert airport.name == "Шереметьево"
        assert airport.city == "Москва"

    def test_add_multiple_airports(self, repo):
        airports_data = [
            ("SVO", "Шереметьево", "Москва"),
            ("LED", "Пулково", "Санкт-Петербург"),
            ("DME", "Домодедово", "Москва"),
        ]
        for code, name, city in airports_data:
            repo.add_airport(code, name, city)
        airports = repo.get_all_airports()
        assert len(airports) == 3
        codes = {a.code for a in airports}
        assert codes == {"SVO", "LED", "DME"}

    def test_add_flight(self, repo):
        repo.add_airport("SVO", "Шереметьево", "Москва")
        repo.add_airport("LED", "Пулково", "Санкт-Петербург")
        repo.add_flight("SU100", "SVO", "LED", "2024-05-20 10:00", "2024-05-
20 11:30")
        flights = repo.get_all_flights()
        assert len(flights) == 1
        flight = flights[0]
        assert flight.number == "SU100"
        assert flight.departure_airport == "SVO"
        assert flight.arrival_airport == "LED"
        assert flight.departure_time == "2024-05-20 10:00"
        assert flight.arrival_time == "2024-05-20 11:30"

    def test_get_flights_by_destination(self, repo):
        repo.add_airport("SVO", "Шереметьево", "Москва")
        repo.add_airport("LED", "Пулково", "Санкт-Петербург")
        repo.add_airport("DME", "Домодедово", "Москва")
        flights_data = [
            ("SU100", "SVO", "LED", "2024-05-20 10:00", "2024-05-20 11:30"),
            ("SU200", "LED", "DME", "2024-05-20 14:00", "2024-05-20 15:30"),
            ("SU300", "SVO", "DME", "2024-05-20 16:00", "2024-05-20 16:45"),
        ]
        for number, departure, arrival, dep_time, arr_time in flights_data:
            repo.add_flight(number, departure, arrival, dep_time, arr_time)
        moscow_flights = repo.get_flights_by_destination("DME")
        assert len(moscow_flights) == 2
        flight_numbers = {f.number for f in moscow_flights}
        assert flight_numbers == {"SU200", "SU300"}
        spb_flights = repo.get_flights_by_destination("LED")
        assert len(spb_flights) == 1
        assert spb_flights[0].number == "SU100"
        empty_flights = repo.get_flights_by_destination("XXX")
        assert len(empty_flights) == 0

    def test_empty_repository(self, repo):
        airports = repo.get_all_airports()
        flights = repo.get_all_flights()
        assert len(airports) == 0
        assert len(flights) == 0

    def test_duplicate_airport(self, repo):
        repo.add_airport("SVO", "Шереметьево", "Москва")
        try:
            repo.add_airport("SVO", "Другое название", "Другой город")
            pytest.fail("Expected exception for duplicate airport")
        except Exception:
            pass
```

```
    def test_airport_dataclass(self):
        airport = Airport(code="TEST", name="Тест", city="Город")
        assert airport.code == "TEST"
        assert airport.name == "Тест"
        assert airport.city == "Город"
        airport2 = Airport(code="TEST", name="Тест", city="Город")
        assert airport == airport2

    def test_flight_dataclass(self):
        flight = Flight(
            number="SU100",
            departure_airport="SVO",
            arrival_airport="LED",
            departure_time="2024-05-20 10:00",
            arrival_time="2024-05-20 11:30",
        )
        assert flight.number == "SU100"
        assert flight.departure_airport == "SVO"
        assert flight.arrival_airport == "LED"
        assert flight.departure_time == "2024-05-20 10:00"
        assert flight.arrival_time == "2024-05-20 11:30"
        flight2 = Flight(
            number="SU100",
            departure_airport="SVO",
            arrival_airport="LED",
            departure_time="2024-05-20 10:00",
            arrival_time="2024-05-20 11:30",
        )
        assert flight == flight2


class TestDisplayFunctions:
    def test_display_flights_empty(self, capsys):
        from task1 import display_flights

        display_flights([])
        captured = capsys.readouterr()
        assert "Список рейсов пуст." in captured.out

    def test_display_flights_with_data(self, capsys):
        from task1 import Flight, display_flights

        flights = [
            Flight(
                number="SU100",
                departure_airport="SVO",
                arrival_airport="LED",
                departure_time="2024-05-20 10:00",
                arrival_time="2024-05-20 11:30",
            )
        ]
        display_flights(flights)
        captured = capsys.readouterr()
        assert "SU100" in captured.out
        assert "SVO" in captured.out
        assert "LED" in captured.out
        assert "2024-05-20 10:00" in captured.out
        assert "2024-05-20 11:30" in captured.out
        assert "Номер" in captured.out
        assert "Аэропорт вылета" in captured.out

    def test_display_airports_empty(self, capsys):
        from task1 import display_airports
```

```
        display_airports([])
        captured = capsys.readouterr()
        assert "Список аэропортов пуст." in captured.out

    def test_display_airports_with_data(self, capsys):
        from task1 import Airport, display_airports

        airports = [Airport(code="SVO", name="Шереметьево", city="Москва")]
        display_airports(airports)
        captured = capsys.readouterr()
        assert "SVO" in captured.out
        assert "Шереметьево" in captured.out
        assert "Москва" in captured.out
        assert "Код" in captured.out
        assert "Название" in captured.out
        assert "Город" in captured.out
```

Листинг программы 4:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import os
import tempfile
from datetime import datetime
from pathlib import Path

import pytest

from tasks.task2 import (
    Airport,
    Flight,
    FlightRepository,
    display_airports,
    display_flights,
)


class TestFlightRepositorySQLAlchemy:
    @pytest.fixture
    def temp_db_path(self):
        fd, db_path = tempfile.mkstemp(suffix=".db")
        os.close(fd)
        yield Path(db_path)
        try:
            if os.path.exists(db_path):
                os.unlink(db_path)
        except (OSError, PermissionError):
            pass

    @pytest.fixture
    def repo(self, temp_db_path):
        return FlightRepository(temp_db_path)

    def test_create_tables(self, repo):
        repo.add_airport("TEST", "Тестовый аэропорт", "Тестовый город")
        airports = repo.get_all_airports()
        assert len(airports) == 1
        assert airports[0].code == "TEST"

    def test_add_airport(self, repo):
```

```
        repo.add_airport("SVO", "Шереметьево", "Москва")
        airports = repo.get_all_airports()
        assert len(airports) == 1
        airport = airports[0]
        assert isinstance(airport, Airport)
        assert airport.code == "SVO"
        assert airport.name == "Шереметьево"
        assert airport.city == "Москва"

    def test_add_multiple_airports(self, repo):
        airports_data = [
            ("SVO", "Шереметьево", "Москва"),
            ("LED", "Пулково", "Санкт-Петербург"),
            ("DME", "Домодедово", "Москва"),
        ]
        for code, name, city in airports_data:
            repo.add_airport(code, name, city)
        airports = repo.get_all_airports()
        assert len(airports) == 3
        codes = {a.code for a in airports}
        assert codes == {"SVO", "LED", "DME"}

    def test_add_flight(self, repo):
        repo.add_airport("SVO", "Шереметьево", "Москва")
        repo.add_airport("LED", "Пулково", "Санкт-Петербург")
        repo.add_flight("SU100", "SVO", "LED", "2024-05-20 10:00", "2024-05-
20 11:30")
        flights = repo.get_all_flights()
        assert len(flights) == 1
        flight = flights[0]
        assert isinstance(flight, Flight)
        assert flight.number == "SU100"
        assert flight.departure_airport_code == "SVO"
        assert flight.arrival_airport_code == "LED"
        assert isinstance(flight.departure_time, datetime)
        assert flight.departure_time.year == 2024
        assert flight.departure_time.month == 5
        assert flight.departure_time.day == 20
        assert flight.departure_time.hour == 10
        assert flight.departure_time.minute == 0
        assert isinstance(flight.arrival_time, datetime)
        assert flight.arrival_time.year == 2024
        assert flight.arrival_time.month == 5
        assert flight.arrival_time.day == 20
        assert flight.arrival_time.hour == 11
        assert flight.arrival_time.minute == 30

    def test_relationships_in_session(self, repo):
        with repo._get_session() as session:
            airport1 = Airport(code="SVO", name="Шереметьево", city="Москва")
            airport2 = Airport(code="LED", name="Пулково", city="Санкт-
Петербург")
            session.add_all([airport1, airport2])
            session.commit()

            dep_time = datetime(2024, 5, 20, 10, 0)
            arr_time = datetime(2024, 5, 20, 11, 30)
            flight = Flight(
                number="SU100",
                departure_airport_code="SVO",
                arrival_airport_code="LED",
                departure_time=dep_time,
                arrival_time=arr_time,
            )
```

```
            session.add(flight)
            session.commit()

            assert flight.departure_airport is not None
            assert flight.arrival_airport is not None
            assert flight.departure_airport.code == "SVO"
            assert flight.arrival_airport.code == "LED"

    def test_get_flights_by_destination(self, repo):
        repo.add_airport("SVO", "Шереметьево", "Москва")
        repo.add_airport("LED", "Пулково", "Санкт-Петербург")
        repo.add_airport("DME", "Домодедово", "Москва")
        flights_data = [
            ("SU100", "SVO", "LED", "2024-05-20 10:00", "2024-05-20 11:30"),
            ("SU200", "LED", "DME", "2024-05-20 14:00", "2024-05-20 15:30"),
            ("SU300", "SVO", "DME", "2024-05-20 16:00", "2024-05-20 16:45"),
        ]
        for number, departure, arrival, dep_time, arr_time in flights_data:
            repo.add_flight(number, departure, arrival, dep_time, arr_time)
        moscow_flights = repo.get_flights_by_destination("DME")
        assert len(moscow_flights) == 2
        flight_numbers = {f.number for f in moscow_flights}
        assert flight_numbers == {"SU200", "SU300"}
        spb_flights = repo.get_flights_by_destination("LED")
        assert len(spb_flights) == 1
        assert spb_flights[0].number == "SU100"
        empty_flights = repo.get_flights_by_destination("XXX")
        assert len(empty_flights) == 0

    def test_empty_repository(self, repo):
        airports = repo.get_all_airports()
        flights = repo.get_all_flights()
        assert len(airports) == 0
        assert len(flights) == 0

    def test_duplicate_airport(self, repo):
        repo.add_airport("SVO", "Шереметьево", "Москва")
        try:
            repo.add_airport("SVO", "Другое название", "Другой город")
            pytest.fail("Expected exception for duplicate airport")
        except Exception:
            pass

    def test_invalid_time_format(self, repo):
        repo.add_airport("SVO", "Шереметьево", "Москва")
        repo.add_airport("LED", "Пулково", "Санкт-Петербург")
        try:
            repo.add_flight(
                "SU100", "SVO", "LED", "неправильный-формат", "2024-05-20
11:30"
            )
            pytest.fail("Expected ValueError for invalid time format")
        except ValueError:
            pass

    def test_airport_model(self):
        airport = Airport(code="TEST", name="Тест", city="Город")
        assert airport.code == "TEST"
        assert airport.name == "Тест"
        assert airport.city == "Город"

    def test_flight_model(self):
        dep_time = datetime(2024, 5, 20, 10, 0)
        arr_time = datetime(2024, 5, 20, 11, 30)
```

```
        flight = Flight(
            number="SU100",
            departure_airport_code="SVO",
            arrival_airport_code="LED",
            departure_time=dep_time,
            arrival_time=arr_time,
        )
        assert flight.number == "SU100"
        assert flight.departure_airport_code == "SVO"
        assert flight.arrival_airport_code == "LED"
        assert flight.departure_time == dep_time
        assert flight.arrival_time == arr_time


class TestDisplayFunctionsSQLAlchemy:
    def test_display_flights_empty(self, capsys):
        display_flights([])
        captured = capsys.readouterr()
        assert "Список рейсов пуст." in captured.out

    def test_display_flights_with_data(self, capsys):
        from datetime import datetime

        dep_time = datetime(2024, 5, 20, 10, 0)
        arr_time = datetime(2024, 5, 20, 11, 30)
        flights = [
            Flight(
                number="SU100",
                departure_airport_code="SVO",
                arrival_airport_code="LED",
                departure_time=dep_time,
                arrival_time=arr_time,
            )
        ]
        display_flights(flights)
        captured = capsys.readouterr()
        assert "SU100" in captured.out
        assert "SVO" in captured.out
        assert "LED" in captured.out
        assert "2024-05-20 10:00" in captured.out
        assert "2024-05-20 11:30" in captured.out

    def test_display_airports_empty(self, capsys):
        display_airports([])
        captured = capsys.readouterr()
        assert "Список аэропортов пуст." in captured.out

    def test_display_airports_with_data(self, capsys):
        airports = [Airport(code="SVO", name="Шереметьево", city="Москва")]
        display_airports(airports)
        captured = capsys.readouterr()
        assert "SVO" in captured.out
        assert "Шереметьево" in captured.out
        assert "Москва" in captured.out
```

```
tests/test_task1.py::TestFlightRepositoryV1::test_create_tables PASSED
tests/test_task1.py::TestFlightRepositoryV1::test_add_airport PASSED
tests/test_task1.py::TestFlightRepositoryV1::test_add_multiple_airports PASSED
tests/test_task1.py::TestFlightRepositoryV1::test_add_flight PASSED
tests/test_task1.py::TestFlightRepositoryV1::test_get_flights_by_destination PASSED
tests/test_task1.py::TestFlightRepositoryV1::test_empty_repository PASSED
tests/test_task1.py::TestFlightRepositoryV1::test_duplicate_airport PASSED
tests/test_task1.py::TestFlightRepositoryV1::test_airport_dataclass PASSED
tests/test_task1.py::TestFlightRepositoryV1::test_flight_dataclass PASSED
tests/test_task1.py::TestDisplayFunctions::test_display_flights_empty PASSED
tests/test_task1.py::TestDisplayFunctions::test_display_flights_with_data PASSED
tests/test_task1.py::TestDisplayFunctions::test_display_airports_empty PASSED
tests/test_task1.py::TestDisplayFunctions::test_display_airports_with_data PASSED
tests/test_task2.py::TestFlightRepositorySQLAlchemy::test_create_tables PASSED
tests/test_task2.py::TestFlightRepositorySQLAlchemy::test_add_airport PASSED
tests/test_task2.py::TestFlightRepositorySQLAlchemy::test_add_multiple_airports PASSED
tests/test_task2.py::TestFlightRepositorySQLAlchemy::test_add_flight PASSED
tests/test_task2.py::TestFlightRepositorySQLAlchemy::test_relationships_in_session PASSED
tests/test_task2.py::TestFlightRepositorySQLAlchemy::test_get_flights_by_destination PASSED
tests/test_task2.py::TestFlightRepositorySQLAlchemy::test_empty_repository PASSED
tests/test_task2.py::TestFlightRepositorySQLAlchemy::test_duplicate_airport PASSED
tests/test_task2.py::TestFlightRepositorySQLAlchemy::test_invalid_time_format PASSED
tests/test_task2.py::TestFlightRepositorySQLAlchemy::test_airport_model PASSED
tests/test_task2.py::TestFlightRepositorySQLAlchemy::test_flight_model PASSED
tests/test_task2.py::TestDisplayFunctionsSQLAlchemy::test_display_flights_empty PASSED
tests/test_task2.py::TestDisplayFunctionsSQLAlchemy::test_display_flights_with_data PASSED
tests/test_task2.py::TestDisplayFunctionsSQLAlchemy::test_display_airports_empty PASSED
tests/test_task2.py::TestDisplayFunctionsSQLAlchemy::test_display_airports_with_data PASSED
```

Рисунок 6 – Результат работы тестов

Контрольные вопросы:

1. Каково назначение модуля sqlite3?

Назначение модуля sqlite3 в Python – предоставить интерфейс для работы с базами данных SQLite. Он позволяет создавать, подключаться, запрашивать и управлять базами данных SQLite напрямую из кода на Python без необходимости использования внешнего сервера баз данных.

2. Как выполняется соединение с базой данных SQLite3? Что такое курсор базы данных?

Соединение с базой данных SQLite3 выполняется с помощью функции sqlite3.connect(), которая принимает путь к файлу базы данных. Если файл существует, происходит подключение к нему; если нет – создается новый файл базы данных. Курсор базы данных – это объект, который позволяет выполнять SQL-запросы и получать результаты. Он создается из соединения методом .cursor() и используется для выполнения таких операций, как execute(), fetchall() и других.

3. Как подключиться к базе данных SQLite3, находящейся в оперативной памяти компьютера?

Чтобы подключиться к базе данных SQLite3 в оперативной памяти (in-memory database), нужно передать специальную строку “:memory:” в функцию sqlite3.connect().

4. Как корректно завершить работу с базой данных SQLite3?

Корректное завершение работы с базой данных SQLite3 включает закрытие курсора с помощью метода .close() и закрытие соединения с помощью метода .close(). Также рекомендуется использовать контекстный менеджер (оператор with) или явно вызывать .commit() для сохранения изменений перед закрытием.

5. Как осуществляется вставка данных в таблицу базы данных SQLite3?

Вставка данных осуществляется с помощью SQL-запроса INSERT INTO. Создается курсор, выполняется метод cursor.execute() с соответствующим SQL-запросом и данными, после чего вызывается connection.commit() для фиксации изменений.

6. Как осуществляется обновление данных таблицы базы данных SQLite3?

Обновление данных осуществляется с помощью SQL-запроса UPDATE. Аналогично вставке, используется cursor.execute() с запросом UPDATE, указывающим таблицу, новые значения и условие для строк, после чего выполняется connection.commit().

7. Как осуществляется выборка данных из базы данных SQLite3?

Выборка данных осуществляется с помощью SQL-запроса SELECT. Выполняется cursor.execute() с запросом SELECT, затем для получения результатов используются методы курсора, такие как fetchone(), fetchall() или fetchmany().

8. Каково назначение метода rowcount?

Метод rowcount объекта курсора возвращает количество строк, затронутых последней операцией INSERT, UPDATE или DELETE. Он показывает, сколько строк было изменено в результате выполнения запроса.

9. Как получить список всех таблиц базы данных SQLite3?

Чтобы получить список всех таблиц в базе данных SQLite3, можно выполнить SQL-запрос к системной таблице sqlite_master: SELECT name FROM sqlite_master WHERE type='table';.

10. Как выполнить проверку существования таблицы как при ее добавлении, так и при ее удалении?

При добавлении таблицы можно использовать конструкцию CREATE TABLE IF NOT EXISTS имя_таблицы, которая создаст таблицу только если она не существует. При удалении – DROP TABLE IF EXISTS имя_таблицы, которая удалит таблицу только если она существует.

11. Как выполнить массовую вставку данных в базу данных SQLite3?

Массовая вставка данных выполняется с помощью метода cursor.executemany(). Он принимает SQL-запрос INSERT и последовательность (например, список кортежей) с данными для вставки, что позволяет вставить множество строк за один вызов.

12. Как осуществляется работа с датой и временем при работе с базами данных SQLite3?

SQLite3 поддерживает хранение даты и времени в виде текста (формат ISO 8601), вещественных чисел (юлианские дни) или целых чисел (Unix time). Для работы с датой и временем в Python можно использовать модуль datetime. SQLite3 может автоматически конвертировать объекты datetime Python в строковый формат и обратно при использовании адаптеров и конвертеров, зарегистрированных через sqlite3.register_adapter() и sqlite3.register_converter(). Также для работы с датами и временем можно использовать встроенные функции SQLite, такие как date(), time(), datetime() и strftime().

Вывод: в ходе лабораторной работы были приобретены навыки взаимодействия с базами данных SQLite3 с помощью языка программирования Python версии 3.13.3.