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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ

По лабораторной работе №7

Дисциплины «Анализ данных»

Выполнил:

Пустяков Андрей Сергеевич

2 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и вычислительная техника (профиль) «Программное обеспечение средств вычислительной
техники и автоматизированных систем», очная форма обучения

_____

(подпись)

Руководитель практики:

Воронкин Р. А. кандидат технических наук, доцент, доцент кафедры инфокоммуникаций

_____

(подпись)

Ставрополь, 2024 г.

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

Цель: изучить работу с базами данных SQLite3 с помощью языка программирования Python версии 3.x.

Ход работы:

Создание общедоступного репозитория на «GitHub», клонирование репозитория, редактирование файла «.gitignore», организация репозитория согласно модели ветвления «git flow» (рис. 1).

```
C:\Users\Andrey\Desktop\Анализ_данных\7_лаба\Analysis_data_laboratory_work_7>git flow init

Which branch should be used for bringing forth production releases?
   - main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
```

Рисунок 1 – Создание репозитория

Проработка примеров лабораторной работы:

Пример 1.

Необходимо для примера 1 лабораторной работы 2.17 реализовать возможность хранения данных в базе данных SQLite3.

Код программы данной задачи:

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

import argparse
import sqlite3
import typing as t
from pathlib import Path


def display_workers(staff: t.List[t.Dict[str, t.Any]]) -> None:
    """
    Отобразить список работников.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
```

```
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)

        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Ф.И.О.",
                "Должность",
                "Год"
            )
        )
        print(line)

        # Вывести данные о всех сотрудниках.
        for idx, worker in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('name', ''),
                    worker.get('post', ''),
                    worker.get('year', 0)
                )
            )

        print(line)

    else:
        print("Список работников пуст.")


def create_db(database_path: Path) -> None:
    """
    Создать базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    # Создать таблицу с информацией о должностях.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS posts (
        post_id INTEGER PRIMARY KEY AUTOINCREMENT,
        post_title TEXT NOT NULL
        )
        """
    )

    # Создать таблицу с информацией о работниках.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS workers (
        worker_id INTEGER PRIMARY KEY AUTOINCREMENT,
        worker_name TEXT NOT NULL,
        post_id INTEGER NOT NULL,
        worker_year INTEGER NOT NULL,
        FOREIGN KEY(post_id) REFERENCES posts(post_id)
        )
        """
    )
```

```
    conn.close()


def add_worker(database_path: Path, name: str, post: str, year: int) -> None:
    """
    Добавить работника в базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    # Получить идентификатор должности в базе данных.
    # Если такой записи нет, то добавить информацию о новой должности.
    cursor.execute(
        """
        SELECT post_id FROM posts WHERE post_title = ?
        """,
        (post,)
    )
    row = cursor.fetchone()
    if row is None:
        cursor.execute(
            """
            INSERT INTO posts (post_title) VALUES (?)
            """,
            (post,)
        )
        post_id = cursor.lastrowid

    else:
        post_id = row[0]

    # Добавить информацию о новом работнике.
    cursor.execute(
        """
        INSERT INTO workers (worker_name, post_id, worker_year)
        VALUES (?, ?, ?)
        """,
        (name, post_id, year)
    )

    conn.commit()
    conn.close()


def select_all(database_path: Path) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать всех работников.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT workers.worker_name, posts.post_title, workers.worker_year
        FROM workers
        INNER JOIN posts ON posts.post_id = workers.post_id
        """
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "name": row[0],
            "post": row[1],
            "year": row[2],
```

```
        }
        for row in rows
    ]


def select_by_period(database_path: Path, period: int) -> t.List[t.Dict[str,
t.Any]]:
    """
    Выбрать всех работников с периодом работы больше заданного.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT workers.worker_name, posts.post_title, workers.worker_year
        FROM workers
        INNER JOIN posts ON posts.post_id = workers.post_id
        WHERE (strftime('%Y', date('now')) - workers.worker_year) >= ?
        """,
        (period,)
        )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "name": row[0],
            "post": row[1],
            "year": row[2],
        }
        for row in rows
    ]


def main(command_line=None):
    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "--db",
        action="store",
        required=False,
        default=str(Path.home() / "workers.db"),
        help="The database file name"
    )

    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("workers")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )

    subparsers = parser.add_subparsers(dest="command")

    # Создать субпарсер для добавления работника.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new worker"
    )
    add.add_argument(
        "-n",
        "--name",
        action="store",
```

```
        required=True,
        help="The worker's name"
    )
    add.add_argument(
        "-p",
        "--post",
        action="store",
        help="The worker's post"
    )
    add.add_argument(
        "-y",
        "--year",
        action="store",
        type=int,
        required=True,
        help="The year of hiring"
    )

    # Создать субпарсер для отображения всех работников.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all workers"
    )

    # Создать субпарсер для выбора работников.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the workers"
    )
    select.add_argument(
        "-P",
        "--period",
        action="store",
        type=int,
        required=True,
        help="The required period"
    )

    # Выполнить разбор аргументов командной строки.
    args = parser.parse_args(command_line)

    # Получить путь к файлу базы данных.
    db_path = Path(args.db)
    create_db(db_path)

    # Добавить работника.
    if args.command == "add":
        add_worker(db_path, args.name, args.post, args.year)

    # Отобразить всех работников.
    elif args.command == "display":
        display_workers(select_all(db_path))

    # Выбрать требуемых рааботников.
    elif args.command == "select":
        display_workers(select_by_period(db_path, args.period))
        pass


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

Результаты работы данной программы, ввод и вывод записей в базе данных и файл базы данных (рис. 2).

```
C:\Users\Andrey>cd C:\Users\Andrey\Desktop\Анализ_данных\7_лаба\Analysis_data_laboratory_work_7\examples

C:\Users\Andrey\Desktop\Анализ_данных\7_лаба\Analysis_data_laboratory_work_7\examples>python example_1.py add --db example.db -n Andrey -p Director -y 2000

C:\Users\Andrey\Desktop\Анализ_данных\7_лаба\Analysis_data_laboratory_work_7\examples>python example_1.py display --db example.db
+------+----------------------------------+----------------------+----------+
|  №   |            Ф.И.О.                |      Должность       |   Год    |
+------+----------------------------------+----------------------+----------+
|    1 | Andrey                           | Director             |     2000 |
+------+----------------------------------+----------------------+----------+
```

Рисунок 2 – Результаты работы программы примера 1

Выполнение индивидуальных заданий:

Задание 1.

Для своего варианта лабораторной работы 2.17 необходимо реализовать хранение данных в базе данных SQLite3. Информация в базе данных должна храниться не менее чем в двух таблицах.

Необходимо использовать словарь, содержащий следующие ключи: название пункта назначения; номер поезда; время отправления. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по времени отправления поезда; вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение (Вариант 26 (7), работа 2.8). Код программы индивидуального задания:

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

import argparse
import sqlite3
import typing as t
from pathlib import Path


def display_trains(trains: t.List[t.Dict[str, t.Any]]) -> None:
    """
    Отобразить список поездов.
    """
    # Проверить, что список поездов не пуст.
    if trains:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
```

```
            '-' * 30,
            '-' * 13,
            '-' * 18,
            '-' * 30
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^13} | {:^18} | {:^30} |'.format(
                "№",
                "Пункт отправления",
                "Номер поезда",
                "Время отправления",
                "Пункт назначения"
            )
        )
        print(line)

        # Вывести данные о всех поездах.
        for idx, train in enumerate(trains, 1):
            print(
                '| {:>4} | {:<30} | {:<13} | {:>18} | {:<30} |'.format(
                    idx,
                    train.get('departure_point', ''),
                    train.get('number_train', ''),
                    train.get('time_departure', ''),
                    train.get('destination', '')
                )
            )
            print(line)

    else:
        print("Список поездов пуст.")


def create_db(database_path: Path) -> None:
    """
    Создать базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    # Создать таблицу с информацией о станциях.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS stations (
        station_id INTEGER PRIMARY KEY AUTOINCREMENT,
        station_name TEXT NOT NULL
        )
        """
    )

    # Создать таблицу с информацией о поездах.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS trains (
        train_id INTEGER PRIMARY KEY AUTOINCREMENT,
        departure_id INTEGER NOT NULL,
        train_number TEXT NOT NULL,
        time_departure TEXT NOT NULL,
        destination_id INTEGER NOT NULL,
        FOREIGN KEY(departure_id) REFERENCES stations(station_id),
        FOREIGN KEY(destination_id) REFERENCES stations(station_id)
        )
        """
```

```
    )

    conn.close()


def add_train(database_path: Path, departure_point: str, number_train: str,
time_departure: str, destination: str) -> None:
    """
    Добавить поезд в базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    # Получить идентификатор станции отправления в базе данных.
    # Если такой записи нет, то добавить информацию о новой станции.
    cursor.execute(
        """
        SELECT station_id FROM stations WHERE station_name = ?
        """,
        (departure_point,)
    )
    row = cursor.fetchone()
    if row is None:
        cursor.execute(
            """
            INSERT INTO stations (station_name) VALUES (?)
            """,
            (departure_point,)
        )
        departure_id = cursor.lastrowid
    else:
        departure_id = row[0]

    # Получить идентификатор станции назначения в базе данных.
    cursor.execute(
        """
        SELECT station_id FROM stations WHERE station_name = ?
        """,
        (destination,)
    )
    row = cursor.fetchone()
    if row is None:
        cursor.execute(
            """
            INSERT INTO stations (station_name) VALUES (?)
            """,
            (destination,)
        )
        destination_id = cursor.lastrowid
    else:
        destination_id = row[0]

    # Добавить информацию о новом поезде.
    cursor.execute(
        """
        INSERT INTO trains (departure_id, train_number, time_departure,
destination_id)
        VALUES (?, ?, ?, ?)
        """,
        (departure_id, number_train, time_departure, destination_id)
    )

    conn.commit()
    conn.close()
```

```python
def select_all(database_path: Path) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать все поезда.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT
            trains.train_number,
            t1.station_name AS departure_point,
            trains.time_departure,
            t2.station_name AS destination
        FROM trains
        JOIN stations t1 ON t1.station_id = trains.departure_id
        JOIN stations t2 ON t2.station_id = trains.destination_id
        """
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "number_train": row[0],
            "departure_point": row[1],
            "time_departure": row[2],
            "destination": row[3],
        }
        for row in rows
    ]


def select_by_destination(database_path: Path, destination: str) ->
t.List[t.Dict[str, t.Any]]:
    """
    Выбрать все поезда, направляющиеся в указанный пункт назначения.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT
            trains.train_number,
            t1.station_name AS departure_point,
            trains.time_departure,
            t2.station_name AS destination
        FROM trains
        JOIN stations t1 ON t1.station_id = trains.departure_id
        JOIN stations t2 ON t2.station_id = trains.destination_id
        WHERE LOWER(t2.station_name) = LOWER(?)
        """,
        (destination,)
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "number_train": row[0],
            "departure_point": row[1],
            "time_departure": row[2],
            "destination": row[3],
        }
        for row in rows
```

```
    ]


def main(command_line=None):
    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "--db",
        action="store",
        required=False,
        default=str(Path.home() / "trains.db"),
        help="The database file name"
    )

    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("trains")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )

    subparsers = parser.add_subparsers(dest="command")

    # Создать субпарсер для добавления поезда.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new train"
    )
    add.add_argument(
        "-dep",
        "--departure_point",
        action="store",
        required=True,
        help="The train's departure point"
    )
    add.add_argument(
        "-n",
        "--number_train",
        action="store",
        required=True,
        help="The train's number"
    )
    add.add_argument(
        "-t",
        "--time_departure",
        action="store",
        required=True,
        help="The time departure of train"
    )
    add.add_argument(
        "-des",
        "--destination",
        action="store",
        required=True,
        help="The destination of train"
    )

    # Создать субпарсер для отображения всех поездов.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all trains"
```

```
    )

    # Создать субпарсер для выбора поездов по пунктам назначения.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the trains"
    )
    select.add_argument(
        "-P",
        "--point_user",
        action="store",
        required=True,
        help="The required point"
    )

    # Выполнить разбор аргументов командной строки.
    args = parser.parse_args(command_line)

    # Получить путь к файлу базы данных.
    db_path = Path(args.db)
    create_db(db_path)

    # Добавить поезд.
    if args.command == "add":
        add_train(
            db_path,
            args.departure_point,
            args.number_train,
            args.time_departure,
            args.destination
        )

    # Отобразить все поезда.
    elif args.command == "display":
        display_trains(select_all(db_path))

    # Выбрать требуемые поезда.
    elif args.command == "select":
        selected = select_by_destination(db_path, args.point_user)
        display_trains(selected)


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

Результаты работы данной программы, ввод и вывод записи в таблицы и содержимое таблиц (рис. 3).

```
C:\Users\Andrey>cd C:\Users\Andrey\Desktop\Анализ_данных\7_лаба\Analysis_data_laboratory_work_7\individual

C:\Users\Andrey\Desktop\Анализ_данных\7_лаба\Analysis_data_laboratory_work_7\individual>python individual_1.py add --db train
s.db --departure_point "Moscow" --number_train "123" --time_departure "15:30" --destination "Saint Petersburg"
python: can't open file 'C:\\Users\\Andrey\\Desktop\\Анализ_данных\\7_лаба\\Analysis_data_laboratory_work_7\\individual\\indi
vidual_1.py': [Errno 2] No such file or directory

C:\Users\Andrey\Desktop\Анализ_данных\7_лаба\Analysis_data_laboratory_work_7\individual>python induvidual_1.py add --db train
s.db --departure_point "Moscow" --number_train "123" --time_departure "15:30" --destination "Saint Petersburg"

C:\Users\Andrey\Desktop\Анализ_данных\7_лаба\Analysis_data_laboratory_work_7\individual>python induvidual_1.py display --db t
rains.db
+------+--------------------------------+---------------+--------------------+--------------------------------+
|  №   |       Пункт отправления       | Номер поезда  | Время отправления  |       Пункт назначения        |
+------+--------------------------------+---------------+--------------------+--------------------------------+
|    1 | Moscow                         | 123           |              15:30 | Saint Petersburg               |
+------+--------------------------------+---------------+--------------------+--------------------------------+
```

Рисунок 3 – Результаты работы программы индивидуального задания 1

Выполнение задания повышенной сложности:

Задание 1.

Необходимо доработать предыдущее задание, самостоятельно изучить работу с пакетом python-psycopg2 для работы с базами данных PostgreSQL. Для своего варианта лабораторной работы 2.17 необходимо реализовать возможность хранения данных в базе данных СУБД PostgreSQL. Информация в базе данных должна храниться не менее чем в двух таблицах.

Код программы задания повышенной сложности:

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

import argparse
import psycopg2


def connect_db():
    """
    Установить соединение с базой данных PostgreSQL.
    """
    conn = psycopg2.connect(
        dbname="trains_db",
        user="train_user",
        password="train_password",
        host="localhost",
        port="5432"
    )
    return conn


def create_tables():
    """
    Создать таблицы в базе данных.
    """
    conn = connect_db()
    cursor = conn.cursor()
```

```
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS stations (
        station_id SERIAL PRIMARY KEY,
        station_name TEXT NOT NULL
        )
        """
    )

    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS trains (
        train_id SERIAL PRIMARY KEY,
        departure_point INTEGER NOT NULL,
        number_train TEXT NOT NULL,
        time_departure TEXT NOT NULL,
        destination INTEGER NOT NULL,
        FOREIGN KEY (departure_point) REFERENCES stations (station_id),
        FOREIGN KEY (destination) REFERENCES stations (station_id)
        )
        """
    )

    conn.commit()
    conn.close()


def add_train(name_dep, number_train, time_departure, name_dest):
    """
    Добавить данные о поезде.
    """
    conn = connect_db()
    cursor = conn.cursor()

    # Получить или добавить станцию отправления.
    cursor.execute(
        "SELECT station_id FROM stations WHERE station_name = %s",
        (name_dep,)
    )
    dep_row = cursor.fetchone()
    if dep_row is None:
        cursor.execute(
            "INSERT INTO stations (station_name) VALUES (%s) RETURNING
station_id",
            (name_dep,)
        )
        dep_id = cursor.fetchone()[0]
    else:
        dep_id = dep_row[0]

    # Получить или добавить станцию назначения.
    cursor.execute(
        "SELECT station_id FROM stations WHERE station_name = %s",
        (name_dest,)
    )
    dest_row = cursor.fetchone()
    if dest_row is None:
        cursor.execute(
            "INSERT INTO stations (station_name) VALUES (%s) RETURNING
station_id",
            (name_dest,)
        )
        dest_id = cursor.fetchone()[0]
```

```
    else:
        dest_id = dest_row[0]

    # Добавить поезд.
    cursor.execute(
        """
        INSERT INTO trains (departure_point, number_train, time_departure,
destination)
        VALUES (%s, %s, %s, %s)
        """,
        (dep_id,
         number_train,
         time_departure,
         dest_id)
    )

    conn.commit()
    conn.close()


def display_trains():
    """
    Отобразить список поездов со станциями.
    """
    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute(
        """
        SELECT t.train_id, s1.station_name AS departure, t.number_train,
t.time_departure,
        s2.station_name AS destination
        FROM trains t
        JOIN stations s1 ON t.departure_point = s1.station_id
        JOIN stations s2 ON t.destination = s2.station_id
        """
    )

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

    if rows:
        line = '+-{}-+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 13,
            '-' * 18,
            '-' * 30)
        print(line)

        print('| {:^4} | {:^30} | {:^13} | {:^18} | {:^30} |'.format(
            "№",
            "Пункт отправления",
            "Номер поезда",
            "Время отправления",
            "Пункт назначения")
        )
        print(line)

        for row in rows:
            print('| {:>4} | {:<30} | {:<13} | {:>18} | {:<30} |'.format(
                row[0],
                row[1],
                row[2],
```

```
                row[3],
                row[4]))
            print(line)

    else:
        print("Список поездов пуст.")


def select_trains(point_user):
    """
    Выбрать поезда по пункту назначения.
    """
    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute(
        """
        SELECT t.train_id, s1.station_name AS departure, t.number_train,
        t.time_departure, s2.station_name AS destination
        FROM trains t
        JOIN stations s1 ON t.departure_point = s1.station_id
        JOIN stations s2 ON t.destination = s2.station_id
        WHERE LOWER(s2.station_name) = %s
        """,
        (point_user.lower(),)
    )

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

    if rows:
        line = '+-{}-+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 13,
            '-' * 18,
            '-' * 30)
        print(line)

        print('| {:^4} | {:^30} | {:^13} | {:^18} | {:^30} |'.format(
            "№",
            "Пункт отправления",
            "Номер поезда",
            "Время отправления",
            "Пункт назначения")
        )
        print(line)

        for row in rows:
            print('| {:>4} | {:<30} | {:<13} | {:>18} | {:<30} |'.format(
                row[0],
                row[1],
                row[2],
                row[3],
                row[4]))
            print(line)

    else:
        print("Список поездов пуст.")


def main(command_line=None):
    parser = argparse.ArgumentParser("trains")
    parser.add_argument(
```

```
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )

    subparsers = parser.add_subparsers(dest="command")

    add = subparsers.add_parser(
        "add",
        help="Add a new train"
    )
    add.add_argument(
        "-dep",
        "--departure_point",
        action="store",
        required=True,
        help="The train's departure point"
    )
    add.add_argument(
        "-n",
        "--number_train",
        action="store",
        required=True,
        help="The train's number"
    )
    add.add_argument(
        "-t",
        "--time_departure",
        action="store",
        required=True,
        help="The time departure of train"
    )
    add.add_argument(
        "-des",
        "--destination",
        action="store",
        required=True,
        help="The destination of train"
    )

    _ = subparsers.add_parser(
        "display",
        help="Display all trains"
    )

    select = subparsers.add_parser(
        "select",
        help="Select the trains"
    )
    select.add_argument(
        "-P",
        "--point_user",
        action="store",
        required=True,
        help="The required point"
    )

    args = parser.parse_args(command_line)

    create_tables()

    if args.command == "add":
        add_train(
            args.departure_point,
```

```
            args.number_train,
            args.time_departure,
            args.destination
        )
    elif args.command == "display":
        display_trains()
    elif args.command == "select":
        select_trains(args.point_user)


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

Результаты работы программы задания повышенной сложности (рис. 4, 5).

```
C:\Users\Andrey>cd C:\Users\Andrey\Desktop\Анализ_данных\7_лаба\Analysis_data_laboratory_work_7\individual

C:\Users\Andrey\Desktop\Анализ_данных\7_лаба\Analysis_data_laboratory_work_7\individual>python individual_1.py add --db train
s.db --departure_point "Moscow" --number_train "123" --time_departure "15:30" --destination "Saint Petersburg"
python: can't open file 'C:\\Users\\Andrey\\Desktop\\Анализ_данных\\7_лаба\\Analysis_data_laboratory_work_7\\individual\\indi
vidual_1.py': [Errno 2] No such file or directory

C:\Users\Andrey\Desktop\Анализ_данных\7_лаба\Analysis_data_laboratory_work_7\individual>python induvidual_1.py add --db train
s.db --departure_point "Moscow" --number_train "123" --time_departure "15:30" --destination "Saint Petersburg"

C:\Users\Andrey\Desktop\Анализ_данных\7_лаба\Analysis_data_laboratory_work_7\individual>python induvidual_1.py display --db t
rains.db
+------+-------------------------------+--------------+--------------------+-------------------------------+
|  №   |       Пункт отправления       | Номер поезда | Время отправления  |       Пункт назначения        |
+------+-------------------------------+--------------+--------------------+-------------------------------+
|    1 | Moscow                        | 123          |              15:30 | Saint Petersburg              |
+------+-------------------------------+--------------+--------------------+-------------------------------+
```

Рисунок 4 – Результаты работы программы

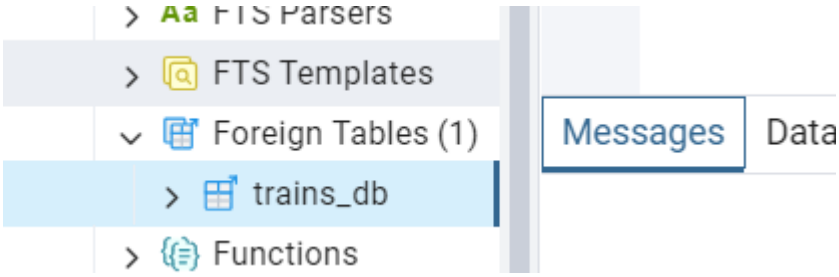

Рисунок 5 – Таблица в базе данных

Ответы на контрольные вопросы:

1. Каково назначение модуля sqlite3?

Модуль sqlite3 предоставляет интерфейс для взаимодействия с базами данных SQLite из программ, написанных на языке Python. Он позволяет создавать, управлять и выполнять запросы к базам данных SQLite.

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

Соединение с базой данных SQLite3 выполняется с использованием функции sqlite3.connect(). Эта функция возвращает объект соединения. Курсор базы данных (cursor) используется для выполнения SQL-запросов и получения результатов.

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

Чтобы подключиться к базе данных SQLite3, находящейся в оперативной памяти, нужно использовать специальное имя файла базы данных ":memory:" при вызове connect.

4. Как корректно завершить работу с базой данных SQLite3?

Чтобы корректно завершить работу с базой данных SQLite3, вызовите метод close объекта соединения.

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

Вставка данных в таблицу выполняется с использованием SQL-запроса INSERT INTO.

6. Как осуществляется обновление данных таблицы базы данных SQLite3?

Обновление данных в таблице выполняется с использованием SQLзапроса UPDATE.

7. Как осуществляется выборка данных из базы данных SQLite3?

Выборка данных выполняется с использованием SQL-запроса SELECT.

8. Каково назначение метода rowcount?

Метод rowcount возвращает количество строк, затронутых последним выполненным SQL-запросом.

9. Как получить список всех таблиц базы данных SQLite3?

Используйте запрос к таблице sqlite_master.

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

Вы можете использовать условие IF NOT EXISTS при создании таблицы и запрос к sqlite_master при проверке существования.

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

Для массовой вставки данных обычно используется метод executemany.

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

SQLite3 поддерживает тип данных DATE и TIMESTAMP для хранения даты и времени. При вставке и выборке данных с использованием этих типов следует использовать соответствующий формат.

Вывод: в ходе выполнения лабораторной работы были исследованы базовые возможности системы управления базами данных SQLite3 в языке Python.