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»
Вариант 29

Выполнил:
Саенко Андрей Максимович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А., канд. технических наук, доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

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

Цель: получить навыки использования sqlite3 в программах на языке Python

Порядок выполнения работы:

Проработаны примеры лабораторной работы:

Пример 1. Для примера 1 лабораторной работы 2.17 реализуйте возможность хранения данных в базе данных SQLite3.

Код программы:

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

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

# Использование SQLite3 в задаче из работы 2.17 для хранения данных

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 раработников.
elif args.command == "select":
    display_workers(select_by_period(db_path, args.period))
```

```
        pass

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

Результат работы программы:

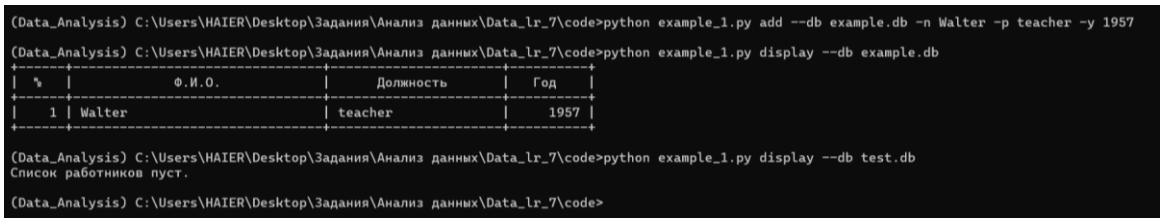

Рисунок 1 – Создание базы данных и добавление нового работника

Выполнены задания:

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.

Вариант 29

Код программы:

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

# 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.

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

def display_routes(staff: t.List[t.Dict[str, t.Any]]) -> None:
    """
    Отобразить список маршрутов
    """
    if staff:
        line = "+-{}-+-{}-+-{}-+-{}-+".format(
            "-" * 4, "-" * 30, "-" * 20, "-" * 10
        )
        print(line)
        print(
            "| {:^4} | {:^30} | {:^20} | {:^10} |".format(
                "№", "Начальный пункт", "Конечный пункт", "Номер"
            )
        )
```

```
        print(line)

        for idx, route in enumerate(staff, 1):
            print(
                "| {:>4} | {:<30} | {:<20} | {:>8} |".format(
                    idx,
                    route.get("start", ""),
                    route.get("end", ""),
                    route.get("number", 0),
                )
            )
            print(line)
    else:
        print("Список  маршрутов пуст.")

def create_db(database_path: Path) -> None:
    """
    Создать базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    # Таблица с конечными станциями.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS end_stations (
            station_id INTEGER PRIMARY KEY AUTOINCREMENT,
            station_title TEXT NOT NULL
        )
        """
    )
    # Таблица с начальными станциями и номерами маршрутов.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS routes (
            route_id INTEGER PRIMARY KEY AUTOINCREMENT,
            start_name TEXT NOT NULL,
            station_id INTEGER NOT NULL,
            route_number INTEGER NOT NULL,
            FOREIGN KEY(station_id) REFERENCES end_stations(station_id)
        )
        """
    )
    conn.close()

def add_route(
    database_path: Path, start_name: str, end_stations: str, number: int
) -> None:
    """
    Добавить маршрут.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
```

```
    cursor.execute(
        """
        SELECT station_id FROM end_stations WHERE station_title = ?
        """,
        (end_stations,),
    )
    row = cursor.fetchone()
    if row is None:
        cursor.execute(
            """
            INSERT INTO end_stations (station_title) VALUES (?)
            """,
            (end_stations,),
        )
        station_id = cursor.lastrowid
    else:
        station_id = row[0]

    cursor.execute(
        """
        INSERT INTO routes (start_name, station_id, route_number)
        VALUES (?, ?, ?)
        """,
        (start_name, station_id, number),
    )
    conn.commit()
    conn.close()

def select_all(database_path: Path) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать всех маршруты.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT routes.start_name, end_stations.station_title,
        routes.route_number
        FROM routes
        INNER JOIN end_stations ON
        end_stations.station_id = routes.station_id
        """
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "start": row[0],
            "end": row[1],
            "number": row[2],
        }
```

```
        for row in rows
    ]

def select_routes(database_path: Path, station)-> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать все маршруты, начальный или конечный пункт которых
    равен заданному.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    cursor.execute(
        """
        SELECT routes.start_name, end_stations.station_title,
        routes.route_number
        FROM routes
        INNER JOIN end_stations ON
        end_stations.station_id = routes.station_id
        WHERE routes.start_name = ? OR end_stations.station_title = ?
        """,
        (station,station,),
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "start": row[0],
            "end": row[1],
            "number": row[2],
        }
        for row in rows
    ]

def main(command_line=None):
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "--db",
        action="store",
        required=False,
        default=str(Path.home() / "routes.db"),
        help="The data file name",
    )

    parser = argparse.ArgumentParser("routes")
    parser.add_argument(
        "--version", action="version", version="%(prog)s 0.1.0"
    )

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

    add = subparsers.add_parser(
        "add", parents=[file_parser], help="Add a new route"
```

```
    )
    add.add_argument(
        "-s",
        "--start",
        action="store",
        required=True,
        help="The start station name",
    )
    add.add_argument(
        "-e",
        "--end",
        action="store",
        required=True,
        help="The end station name",
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        type=int,
        required=True,
        help="Number of route",
    )

    _ = subparsers.add_parser(
        "display", parents=[file_parser], help="Display all routes"
    )

    select = subparsers.add_parser(
        "select", parents=[file_parser], help="Select the routes"
    )
    select.add_argument(
        "--sr",
        action="store",
        required=True,
        help="Select the route",
    )
    args = parser.parse_args(command_line)

    db_path = Path(args.db)
    create_db(db_path)
    if args.command == "add":
        add_route(db_path, args.start, args.end, args.number)

    elif args.command == "display":
        display_routes(select_all(db_path))

    elif args.command == "select":
        display_routes(select_routes(db_path, args.sr))
        pass
if __name__ == "__main__":
    main()
```

Результат работы программы:

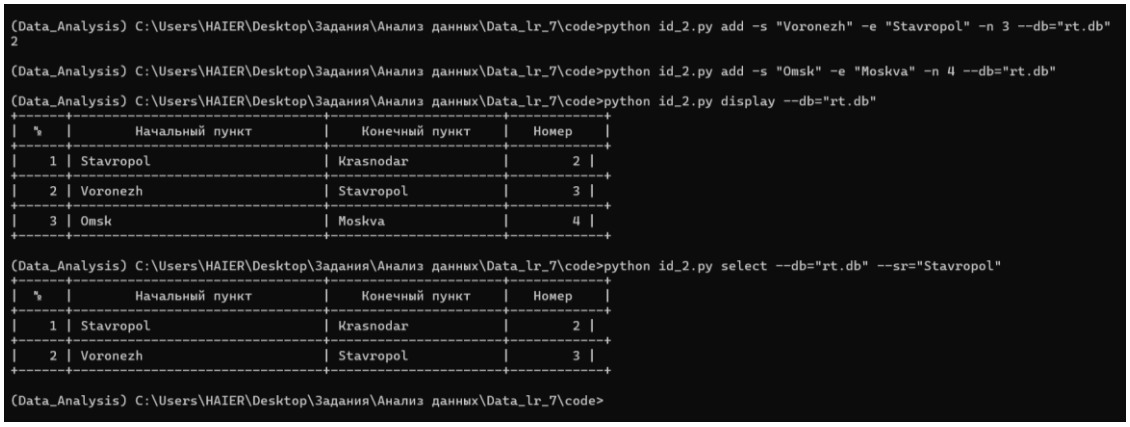

Рисунок 2 – Создание базы данных, добавление маршрутов и вывод маршрутов, одной из точек которого является Ставрополь

Выводы: В ходе выполнения работы получены навыки написания программ на языке Python, использующих SQLite