

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 9**

Выполнил:  
Дудкин Константин Александрович  
2 курс, группа ИВТ-б-о-22-1,  
09.03.01 «Информатика и  
вычислительная техника», очная  
форма обучения

---

(подпись)

Руководитель практики:  
Кандида технических наук, доцент  
кафедры инфокоммуникаций, доцент  
Воронкин Роман Александрович

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2024 г.

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

Цель: Исследовать взаимодействие с базами данных SQLite3 с помощью языка программирования Python

## Ход выполнения работы

Проработал пример в лабораторной работе:

[illegible]

Рисунок 1. Реализация сохранения данных в базе данных SQLite3

Выполнил индивидуальное задание — 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:

The code editor shows a Python script named `individual.py` that creates a SQLite database with a table `route`. The table has columns: `route_id` (INTEGER PRIMARY KEY AUTOINCREMENT), `name_id` (INTEGER NOT NULL), `first_station` (TEXT NOT NULL), and `second_station` (TEXT NOT NULL). It also includes a foreign key constraint: `FOREIGN KEY(name_id) REFERENCES start(name_id)`. The script then inserts four records into the `route` table.

The terminal window shows the execution of the script:

```
[code_raider@raiderpc Progs]$ python individual.py add -f "Минеральные воды" -s "Ставрополь"
[code_raider@raiderpc Progs]$ python individual.py add -f "Ессентуки" -s "Ставрополь"
[code_raider@raiderpc Progs]$ python individual.py add -f "Ессентуки" -s "Ставрополь"
[code_raider@raiderpc Progs]$ python individual.py display
```

№	Место отправки	Место прибытия
1	Минеральные воды	Ставрополь
2	Минеральные воды	Ессентуки
3	Ессентуки	Ставрополь
4	Ессентуки	Пятигорск

```
[code_raider@raiderpc Progs]$
```

Рисунок 2. Результат добавления значений в базу данных

```
[code_raider@raiderpc Progs]$ python individual.py display
```

№	Место отправки	Место прибытия
1	Минеральные воды	Ставрополь
2	Минеральные воды	Ессентуки
3	Ессентуки	Ставрополь
4	Ессентуки	Пятигорск

Рисунок 3. Результат работы display (показ таблицы базы данных)

```
[code_raider@raiderpc Progs]$ python individual.py select --second "Ессентуки"
```

№	Место отправки	Место прибытия
1	Минеральные воды	Ессентуки

```
[code_raider@raiderpc Progs]$ python individual.py select --second "Пятигорск"
```

№	Место отправки	Место прибытия
1	Ессентуки	Пятигорск

Рисунок 4. Результат работы select (просмотр определенных маршрутов)

## Ответы на вопросы

1. Модуль `sqlite3` предназначен для взаимодействия с базой данных SQLite3 из Python. Он позволяет создавать, управлять и взаимодействовать с базами данных SQLite3 из Python-приложений. Модуль `sqlite3` обеспечивает доступ к функциям SQLite3 через простой и удобный интерфейс, что делает его популярным инструментом для работы с базами данных в Python.

2. Для выполнения соединения с базой данных SQLite3 в Python с использованием модуля `sqlite3`, необходимо выполнить следующие шаги:

1. Импортировать модуль `sqlite3`: `import sqlite3`.

1) Установить соединение с базой данных: `connection = sqlite3.connect('database.db')`.

2) Создать курсор базы данных: `cursor = connection.cursor()`. Курсор представляет собой механизм, который позволяет выполнять операции с базой данных, такие как выполнение SQL-запросов, получение результатов и управление данными.

3. Для подключения к базе данных SQLite3, находящейся в оперативной памяти компьютера, можно использовать специальное ключевое слово `":memory:"`. Пример подключения к такой базе данных: `connection = sqlite3.connect(':memory:')`

4. Для корректного завершения работы с базой данных SQLite3 в Python, необходимо закрыть курсор и соединение с базой данных:

`cursor.close()`

`connection.close()`

5. Для вставки данных в таблицу базы данных SQLite3 в Python с использованием модуля `sqlite3`, можно выполнить SQL-запрос с использованием метода `execute()`:

`cursor.execute("INSERT INTO table_name (column1, column2) VALUES (value1, value2)")`

`connection.commit()`

6. Для обновления данных в таблице базы данных SQLite3 в Python с использованием модуля `sqlite3`, можно выполнить SQL-запрос с использованием метода `execute()`:

```
cursor.execute("UPDATE table_name SET column1 = new_value WHERE condition")
```

```
connection.commit()
```

7. Для выборки данных из базы данных SQLite3 в Python с использованием модуля `sqlite3`, можно выполнить SQL-запрос с использованием метода `execute()` и получить результаты с помощью метода `fetchall()`:

```
cursor.execute("SELECT * FROM table_name")
```

```
results = cursor.fetchall()
```

8. Метод `rowcount` в модуле `sqlite3` возвращает количество строк, затронутых последним выполненным SQL-запросом. Этот метод может быть использован для получения информации о количестве измененных или выбранных строк после выполнения операций в базе данных SQLite3.

9. Для получения списка всех таблиц базы данных SQLite3 в Python с использованием модуля `sqlite3`, можно выполнить SQL-запрос к системной таблице `sqlite_master`:

```
cursor.execute("SELECT name FROM sqlite_master WHERE type='table'")
```

```
tables = cursor.fetchall()
```

10. Для проверки существования таблицы при ее добавлении или удалении в базе данных SQLite3 в Python с использованием модуля `sqlite3`, можно выполнить запрос к системной таблице `sqlite_master` и проверить наличие соответствующей записи.

11. Для проверки существования таблицы при ее добавлении или удалении в базе данных SQLite3 в Python с использованием модуля `sqlite3`, можно выполнить запрос к системной таблице `sqlite_master` и проверить наличие соответствующей записи. Например, чтобы проверить

существование таблицы с именем "table\_name", можно выполнить следующий SQL-запрос:

```
cursor.execute("SELECT name FROM sqlite_master WHERE type='table'
AND name='table_name'")
```

Если результат запроса содержит записи, то таблица существует.

12. Для массовой вставки данных в базу данных SQLite3 в Python с использованием модуля sqlite3, можно воспользоваться методом executemany(). Например, если у нас есть список кортежей data с данными для вставки, то можно выполнить следующий код:

```
data = [(value1, value2), (value3, value4), (value5, value6)]
cursor.executemany("INSERT INTO table_name (column1, column2)
VALUES (?, ?)", data)
connection.commit()
```

13. При работе с датой и временем в базах данных SQLite3 в Python, можно использовать тип данных DATE для хранения даты и TIMESTAMP для хранения даты и времени. Также, можно воспользоваться модулем datetime для работы с датой и временем в Python и взаимодействия с базой данных SQLite3. Например, для вставки даты в таблицу можно использовать следующий код:

```
import datetime
current_date = datetime.date.today()
cursor.execute("INSERT INTO table_name (date_column) VALUES (?)"
(current_date,))
connection.commit()
```

Это позволит вставить текущую дату в столбец "date\_column" таблицы "table\_name".

Вывод: В ходе выполнения данной работы было изучено взаимодействие языка программирования Python с базами данных SQLite3