

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное  
образовательное учреждение высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций**

**Отчет по лабораторной работе № 2.21**

**«Взаимодействие с базами данных SQLite3 с помощью языка  
программирования Python»  
по дисциплине «Основы программной инженерии»**

Выполнил студент группы

ПИЖ-б-о-21-1

Зиберов Александр

«    » мая 2023 г.

Подпись студента \_\_\_\_\_

Работа защищена

«    » \_\_\_\_\_ 20\_\_ г.

Проверил Воронкин Р.А. \_\_\_\_\_  
(подпись)

Ставрополь, 2023

## Цель работы:

Исследовать базовые возможности системы управления базами данных SQLite3 и ее интеграцию в программы на языке Python.

## Выполнение работы:

Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT, рисунок 1.

Ссылка: <https://github.com/afk552/lab2.21>

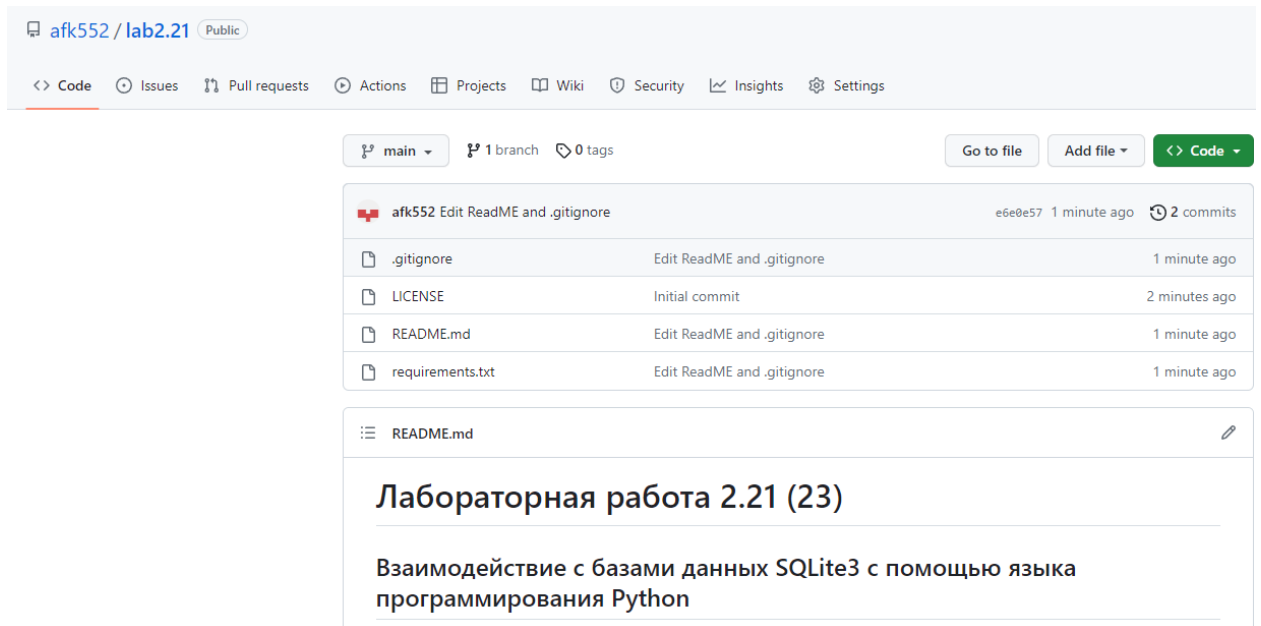


Рисунок 1 – Удаленный репозиторий на GitHub

Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm, рисунок 2.

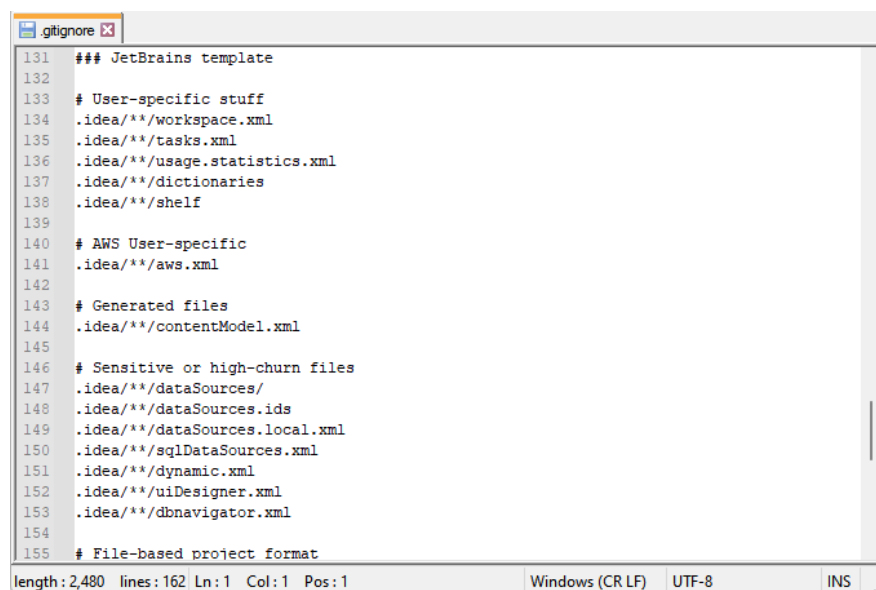


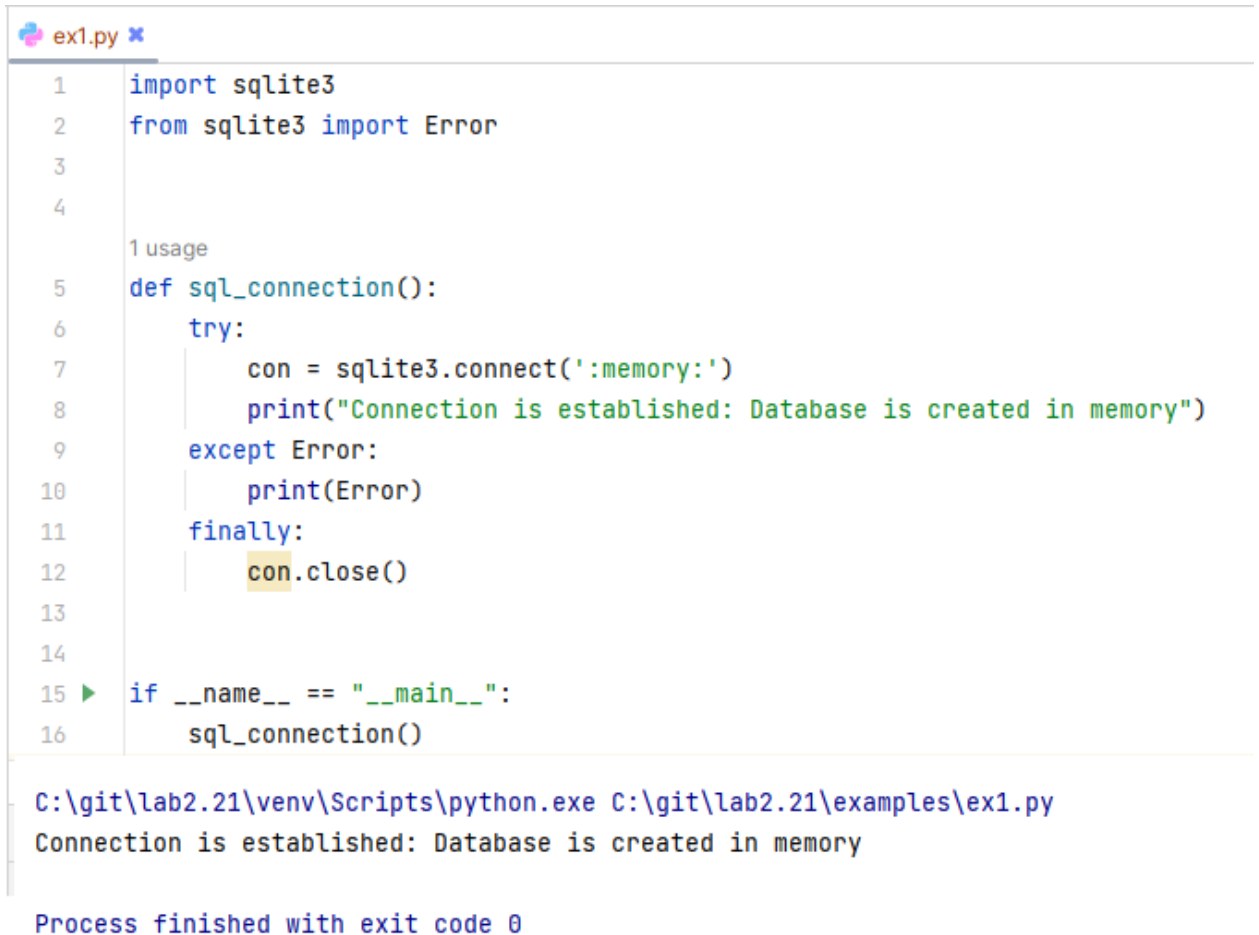
Рисунок 2 – Окно блокнота

Организируйте свой репозиторий в соответствии с моделью ветвления git-flow, рисунок 3.

```
C:\git\lab2.21>git checkout -b develop  
Switched to a new branch 'develop'
```

Рисунок 3 – Окно командной строки

## Примеры



The screenshot shows a code editor window titled 'ex1.py'. The code is a Python script that uses the sqlite3 module to create an in-memory database. The script includes imports, a function definition, and a main block. Below the code, the execution output is shown, indicating that the connection was established successfully and the process finished with exit code 0.

```
1 import sqlite3  
2 from sqlite3 import Error  
3  
4  
5 1 usage  
6 def sql_connection():  
7     try:  
8         con = sqlite3.connect(':memory:')  
9         print("Connection is established: Database is created in memory")  
10    except Error:  
11        print(Error)  
12    finally:  
13        con.close()  
14  
15 ► if __name__ == "__main__":  
16    sql_connection()
```

C:\git\lab2.21\venv\Scripts\python.exe C:\git\lab2.21\examples\ex1.py  
Connection is established: Database is created in memory  
  
Process finished with exit code 0

Рисунок 4 – Пример 1

```
ex2.py x
1 import sqlite3
2 from sqlite3 import Error
3
4
5 1 usage
6 def sql_connection():
7     try:
8         con = sqlite3.connect('mydatabase.db')
9         return con
10
11     except Error:
12         print(Error)
13
14     return None
15
16 1 usage
17 def sql_table(con):
18     cursor_obj = con.cursor()
19     cursor_obj.execute(
20         """
21         CREATE TABLE employees (
22             id integer PRIMARY KEY,
23             name text,
24             salary real,
25             department text,
26             position text,
27             hireDate text)
28         """
29     )
30     con.commit()
31
32 if __name__ == "__main__":
33     con = sql_connection()
34     sql_table(con)
```

Рисунок 5 – Пример 2

ex3.py

```
1 import sqlite3
2
3 con = sqlite3.connect('mydatabase.db')
4
5 1 usage
6 def sql_insert(con, entities):
7     cursor_obj = con.cursor()
8     cursor_obj.execute(
9         """
10         INSERT INTO employees(id, name, salary, department, position, hireDate)
11         VALUES(?, ?, ?, ?, ?, ?)
12         """,
13         entities
14     )
15     con.commit()
16 entities = (2, 'Andrew', 800, 'IT', 'Tech', '2018-02-06')
17 sql_insert(con, entities)
```

Рисунок 6 – Пример 3

ex4.py

```
1 import sqlite3
2
3 con = sqlite3.connect('mydatabase.db')
4
5 1 usage
6 def sql_update(con):
7     cursor_obj = con.cursor()
8     cursor_obj.execute(
9         "UPDATE employees SET name = 'Rogers' where id = 2"
10     )
11     con.commit()
12
13
14 sql_update(con)
15
```

Рисунок 7 – Пример 4

```
ex5.py x
1 import sqlite3
2
3 con = sqlite3.connect('mydatabase.db')
4
5
6 1 usage
7 def sql_fetch(con):
8     cursor_obj = con.cursor()
9     cursor_obj.execute("SELECT * FROM employees")
10    rows = cursor_obj.fetchall()
11    for row in rows:
12        print(row)
13
14 sql_fetch(con)
15
```

Рисунок 8 – Пример 5

```
ex6.py x
1 import sqlite3
2
3 con = sqlite3.connect('mydatabase.db')
4
5
6 1 usage
7 def sql_fetch(con):
8     cursor_obj = con.cursor()
9     cursor_obj.execute(
10         "SELECT id, name FROM employees WHERE salary > 800.0"
11     )
12     rows = cursor_obj.fetchall()
13     for row in rows:
14         print(row)
15
16 sql_fetch(con)
17
```

Рисунок 9 – Пример 6

```

ex7.py x
1 import sqlite3
2
3 con = sqlite3.connect('mydatabase.db')
4
5
6 1 usage
7 def sql_fetch(con):
8     cursor_obj = con.cursor()
9     cursor_obj.execute(
10         "SELECT name from sqlite_master where type='table'"
11     )
12     print(cursor_obj.fetchall())
13
14 sql_fetch(con)

```

Рисунок 10 – Пример 7

```

ex8.py x
1 import sqlite3
2
3 con = sqlite3.connect('mydatabase.db')
4
5
6 1 usage
7 def sql_fetch(con):
8     cursor_obj = con.cursor()
9     cursor_obj.execute(
10         "CREATE TABLE IF NOT EXISTS projects(id INTEGER, name TEXT)"
11     )
12     con.commit()
13
14 sql_fetch(con)

```

Рисунок 11 – Пример 8

ex9.py x

```
1 import sqlite3
2
3 con = sqlite3.connect('mydatabase.db')
4 cursor_obj = con.cursor()
5 cursor_obj.execute(
6     "CREATE TABLE IF NOT EXISTS projects(id INTEGER, name TEXT)"
7 )
8 data = [
9     (1, "Ridesharing"),
10    (2, "Water Purifying"),
11    (3, "Forensics"),
12    (4, "Botany")
13 ]
14 cursor_obj.executemany("INSERT INTO projects VALUES (?, ?)", data)
15 con.commit()
```

Рисунок 12 – Пример 9

ex10.py x

```
1 import sqlite3
2 import datetime
3
4 con = sqlite3.connect('mydatabase.db')
5 cursor_obj = con.cursor()
6 cursor_obj.execute(
7     """
8     CREATE TABLE IF NOT EXISTS assignments(
9         id INTEGER, name TEXT, date DATE
10     )
11     """
12 )
13 data = [
14     (1, "Ridesharing", datetime.date(2017, 1, 2)),
15     (2, "Water Purifying", datetime.date(2018, 3, 4))
16 ]
17 cursor_obj.executemany("INSERT INTO assignments VALUES(?, ?, ?)", data)
18 con.commit()
19
```

Рисунок 13 – Пример 10



```
(venv) PS C:\git\lab2.21\examples> py workers.py --help
usage: workers [-h] [--version] {add,display,select} ...
positional arguments:
  {add,display,select}
    add                Add a new worker
    display            Display all workers
    select            Select the workers
options:
  -h, --help            show this help message and exit
  --version            show program's version number and exit
(venv) PS C:\git\lab2.21\examples> py workers.py add -h
usage: workers add [-h] [--db DB] -n NAME [-p POST] -y YEAR

options:
  -y YEAR, --year YEAR  The year of hiring
(venv) PS C:\git\lab2.21\examples> py workers.py add -n Name -p 436242 -y 2077
(venv) PS C:\git\lab2.21\examples> py workers.py display
```

No	Ф.И.О.	Должность	Год
1	Name	436242	2077

Рисунок 14 – Пример (workers), выполнение

### 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.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import argparse
5  import sqlite3
6  import typing as t
7  from pathlib import Path
8
9
10 1 usage
11 def create_db(database_path: Path) -> None:
12     """
13     Создать базу данных.
14     """
15     conn = sqlite3.connect(database_path)
16     cursor = conn.cursor()
17     # Создать таблицу с людьми и днями рождения
18     cursor.execute(
19         """
20         CREATE TABLE IF NOT EXISTS people (
21             person_id INTEGER PRIMARY KEY AUTOINCREMENT,
22             person_name TEXT NOT NULL,
23             person_birth TEXT NOT NULL,
24             FOREIGN KEY(person_id) REFERENCES pnumbers(person_id)
25         )
26         """
27     )
28     # Создать таблицу с номерами телефонов людей
29     cursor.execute(
30         """
31         CREATE TABLE IF NOT EXISTS pnumbers (
32             person_id INTEGER PRIMARY KEY AUTOINCREMENT,
33             pnumber INTEGER NOT NULL
34         )
35         """
36     )
37     conn.close()
38
39
40 1 usage
41 def add_people(
42     database_path: Path, name: str, pnumber: int, birth: str
43 ) -> None:
44     """
45     Добавить человека в БД
46     """
47     conn = sqlite3.connect(database_path)
48     cursor = conn.cursor()
49
50     cursor.execute(
51         """
52         SELECT person_id FROM people WHERE person_name = ?
53         """
54     )
55     row = cursor.fetchone()
56
57     if row is None:
58         cursor.execute(
59             """
60             INSERT INTO people (person_name, person_birth) VALUES (?, ?)
61             """
62         )
63         (name, birth),
64     )
65     person_id = cursor.lastrowid
66
67     else:
68         person_id = row[0]
69         # Добавить информацию о человеке
70     cursor.execute(
71         """
72         INSERT INTO pnumbers (person_id, pnumber)
73         VALUES (?, ?)
74         """
75     )
76     (person_id, pnumber),
77 )
78     conn.commit()
79     conn.close()

```

Рисунок 15 – Код индивидуального задания (1)

```

80 1 usage
81 def select_all(database_path: Path) -> t.List[t.Dict[str, t.Any]]:
82     """
83     Вывести всех людей из БД
84     """
85     conn = sqlite3.connect(database_path)
86     cursor = conn.cursor()
87     cursor.execute(
88         """
89         SELECT people.person_name, people.person_birth, pnumbers.pnumber
90         FROM pnumbers
91         INNER JOIN people ON people.person_id = pnumbers.person_id
92         """
93     )
94     rows = cursor.fetchall()
95     conn.close()
96     return [
97         {
98             "name": row[0],
99             "pnumber": row[1],
100             "birth": row[2],
101         }
102     ]
103
104
105 1 usage
106 def find_people(database_path: Path, birth: str) -> t.List[t.Dict[str, t.Any]]:
107     """
108     Вывод на экран информации о человеке по дате рождения
109     """
110     conn = sqlite3.connect(database_path)
111     cursor = conn.cursor()
112     cursor.execute(
113         """
114         SELECT people.person_name, people.person_birth, pnumbers.pnumber
115         FROM pnumbers
116         INNER JOIN people ON people.person_id = pnumbers.person_id
117         WHERE people.person_birth LIKE ? || '%'
118         """
119     )
120     (birth),
121 )
122     rows = cursor.fetchall()
123     conn.close()
124     if len(rows) == 0:
125         return []
126
127     return [
128         {
129             "name": row[0],
130             "birth": row[1],
131             "pnumber": row[2],
132         }
133     ]
134
135 2 usages
136 def display_people(people_list: t.List[t.Dict[str, t.Any]]) -> None:
137     """
138     Вывести людей из списка
139     """
140     if people_list:
141         line = "+-{}-+-{}-+-{}-+-{}-+-".format(
142             "-" * 4, "-" * 30, "-" * 14, "-" * 19
143         )
144         print(line)
145         print(
146             "| {:^4} | {:^30} | {:^14} | {:^19} |".format(
147                 "\n/n", "Фамилия Имя", "Дата рождения", "Номер телефона"
148             )
149         )
150         print(line)
151
152         for nmbr, person in enumerate(people_list, 1):
153             print(
154                 "| {:>4} | {:<30} | {:<14} | {:>19} |".format(
155                     nmbr,
156                     person.get("name", ""),
157                     person.get("pnumber", ""),
158                     person.get("birth", ""),
159                 )
160             )
161             print(line)
162
163     else:
164         print("Список пуст.")
165

```

Рисунок 16 – Код индивидуального задания (2)

```

100 def main(command_line=None):
101     """
102     Главная функция программы.
103     """
104     # Создать родительский парсер для определения имени файла
105     file_parser = argparse.ArgumentParser(add_help=False)
106     file_parser.add_argument(
107         "--db",
108         action="store",
109         required=False,
110         default=str(Path.cwd() / "people_data.db"),
111         help="Имя файла БД",
112     )
113
114     # Создать основной парсер командной строки
115     parser = argparse.ArgumentParser("people")
116     parser.add_argument(
117         "--version", action="version", version=f"%(prog)s alpha beta 0.0.1"
118     )
119
120     subparsers = parser.add_subparsers(dest="command")
121
122     # Создать субпарсер для добавления работника
123     add = subparsers.add_parser(
124         "add", parents=[file_parser], help="Добавить нового человека"
125     )
126     add.add_argument(
127         "-n",
128         "--name",
129         action="store",
130         required=True,
131         help="Имя и фамилия человека",
132     )
133     add.add_argument(
134         "-p", "--pnumber", type=int, action="store", help="Номер телефона"
135     )
136     add.add_argument(
137         "-b", "--birth", action="store", required=True, help="Дата рождения"
138     )
139
140     # Создать субпарсер для отображения всех людей
141     _ = subparsers.add_parser(
142         "display", parents=[file_parser], help="Отобразить всех людей"
143     )
144
145     # Создать субпарсер для поиска людей по фамилии
146     select = subparsers.add_parser(
147         "select", parents=[file_parser], help="Выбор человека"
148     )
149     select.add_argument(
150         "-b", "--birth", action="store", required=True, help="Дата рождения"
151     )
152
153     if __name__ == "__main__":
154         main()
155
156     # Выполнить разбор аргументов командной строки
157     args = parser.parse_args(command_line)
158
159     # Получить путь к файлу БД
160     db_path = Path(args.db)
161     create_db(db_path)
162
163     match args.command:
164         # Добавить человека
165         case "add":
166             add_people(db_path, args.name, args.pnumber, args.birth)
167         # Отобразить всех людей
168         case "display":
169             display_people(select_all(db_path))
170         # Выбрать людей по дате рождения
171         case "select":
172             display_people(find_people(db_path, args.birth))
173
174     if __name__ == "__main__":
175         main()

```

Рисунок 17 – Код индивидуального задания (3)

```

(venv) C:\git\lab2.21\indiv_task>py indiv.py --help
usage: people [-h] [--version] {add,display,select} ...

```

positional arguments:

```

{add,display,select}
    add            Добавить нового человека
    display        Отобразить всех людей
    select         Выбор человека

```

options:

```

-h, --help            show this help message and exit
--version            show program's version number and exit

```

```

(venv) C:\git\lab2.21\indiv_task>py indiv.py add -n Name -b 02.02.2002 -p 89743724724

```

```

(venv) C:\git\lab2.21\indiv_task>py indiv.py display

```

№п/п	Фамилия Имя	Дата рождения	Номер телефона
1	Name	02.02.2002	89743724724

```

(venv) C:\git\lab2.21\indiv_task>py indiv.py select -b 02.02.2002

```

№п/п	Фамилия Имя	Дата рождения	Номер телефона
1	Name	89743724724	02.02.2002

```

(venv) C:\git\lab2.21\indiv_task>

```

Name	Type
Tables (3)	
people	
pnumbers	
sqlite_sequence	
Indices (0)	
Views (0)	
Triggers (0)	

```

1 SELECT * FROM people

```

person_id	person_name	person_birth
1	Name	02.02.2002

```

1 SELECT * FROM pnumbers

```

person_id	pnumber
1	89743724724

Рисунок 18 – Выполнение индивидуального задания

**Вывод:** Были изучены базовые возможности системы управления базами данных SQLite3 и ее интеграция в программы на языке Python.

### **Контрольные вопросы:**

#### **1. Каково назначение модуля sqlite3?**

Непосредственно модуль sqlite3 – это API к СУБД SQLite. Своего рода адаптер, который переводит команды, написанные на Питоне, в команды, которые понимает SQLite. Как и наоборот, доставляет ответы от SQLite в python-программу.

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

Чтобы использовать SQLite3 в Python, прежде всего, вам нужно будет импортировать модуль sqlite3, а затем создать объект соединения, который соединит нас с базой данных и позволит нам выполнять операторы SQL. Объект соединения создается с помощью функции connect(). Вызов функции connect() приводит к созданию объекта-экземпляра от класса Connection. Этот объект обеспечивает связь с файлом базы данных, представляет конкретную БД в программе.

Для взаимодействия с базой данных SQLite3 в Python необходимо создать объект cursor. Вы можете создать его с помощью метода cursor() . Курсор SQLite3 – это метод объекта соединения. Для выполнения инструкций SQLite3 сначала устанавливается соединение, а затем создается объект курсора с использованием объекта соединения.

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

Создать базу данных в оперативной памяти с помощью функции :memory: with the connect. Такая база данных называется базой данных в памяти.

#### **4. Как корректно завершить работу с базой данных SQLite3?**

`con.close()`

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

Чтобы вставить данные в таблицу, используется оператор `INSERT INTO`. Мы также можем передавать значения/аргументы оператору `INSERT` в методе `execute()`.

#### **6. Как осуществляется обновление данных таблицы базы данных SQLite3?**

Чтобы обновить данные в таблице, просто создайте соединение, затем создайте объект курсора с помощью соединения и, наконец, используйте оператор `UPDATE` в методе `execute()`.

#### **7. Как осуществляется выборка данных из базы данных SQLite3?**

Оператор `SELECT` используется для выбора данных из определенной таблицы. Если вы хотите выбрать все столбцы данных из таблицы, вы можете использовать звездочку (\*). Синтаксис для этого будет следующим. В SQLite3 оператор `SELECT` выполняется в методе `execute()` объекта `cursor`.

#### **8. Каково назначение метода rowcount?**

SQLite3 `rowcount` используется для возврата количества строк, которые были затронуты или выбраны последним выполненным SQL-запросом.

#### **9. Как получить список всех таблиц базы данных SQLite3?**

Чтобы перечислить все таблицы в базе данных SQLite3, вы должны запросить данные из таблицы `sqlite_master`, а затем использовать `fetchall()` для получения результатов из инструкции `SELECT`.

sqlite\_master – это главная таблица в SQLite3, которая хранит все таблицы.

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

Чтобы проверить, не существует ли таблица уже, мы используем IF NOT EXISTS с оператором CREATE TABLE:

```
CREATE TABLE IF NOT EXISTS table_name (column1, column2, ..., columnN)
```

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

Метод executemany можно использовать для вставки нескольких строк одновременно.

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

В базе данных Python SQLite3 мы можем легко хранить дату или время, импортируя модуль datetime