

**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 «Технологии программирования и алгоритмизации»**

Выполнил студент группы ИВТ-б-о-21-1

Лысенко И.А. « » \_\_\_\_\_ 20\_\_ г.

Подпись студента \_\_\_\_\_

Работа защищена « » \_\_\_\_\_ 20\_\_ г.

Проверил Воронкин Р.А. \_\_\_\_\_

(подпись)

Ставрополь 2023

**Цель работы:** исследовать взаимодействие с базами данных SQLite3 с помощью языка программирования Python.

**Ход работы:**

1. Создал репозиторий на GitHub:  
[https://github.com/IsSveshuD/lab\\_2\\_21.git](https://github.com/IsSveshuD/lab_2_21.git).

2. Проработал пример:

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  import argparse
6  import sqlite3
7  import typing as t
8  from pathlib import Path
9
10
11 def display_workers(staff: t.List[t.Dict[str, t.Any]]) -> None:
12     """
13     Отобразить список работников.
14     """
15     # Проверить, что список работников не пуст.
16     if staff:
17         # Заголовок таблицы.
18         line = '+-{}-+-{}-+-{}-+-{}-+'.format(
19             '-' * 4,
20             '-' * 30,
21             '-' * 20,
22             '-' * 8
23         )
24         print(line)
25         print(
26             '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
27                 "No",
28                 "Ф.И.О.",
29                 "Должность",
30                 "Год"
31             )
32         )
33
34
35     for idx, worker in enumerate(staff, 1):
36         print(
37             '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
38                 idx,
39                 worker.get('name', ''),
40                 worker.get('post', ''),
41                 worker.get('year', 0)
42             )
43         )
44     else:
45         print("Список работников пуст.")
46
47
48 def create_db(database_path: Path) -> None:
49     """
50     Создать базу данных.
51     """
52     conn = sqlite3.connect(database_path)
53     cursor = conn.cursor()
54
55     # Создать таблицу с информацией о должностях.
56     cursor.execute(
57         """
58         CREATE TABLE IF NOT EXISTS posts (
59             post_id INTEGER PRIMARY KEY AUTOINCREMENT,
60             post_title TEXT NOT NULL
61         )
62         """
63     )
```

```

64 # Создать таблицу с информацией о работниках.
65 cursor.execute(
66     """
67     CREATE TABLE IF NOT EXISTS workers (
68     worker_id INTEGER PRIMARY KEY AUTOINCREMENT,
69     worker_name TEXT NOT NULL,
70     post_id INTEGER NOT NULL,
71     worker_year INTEGER NOT NULL,
72     FOREIGN KEY(post_id) REFERENCES posts(post_id)
73     )
74     """
75 )
76 conn.close()
77
78
79 def add_worker(
80     database_path: Path,
81     name: str,
82     post: str,
83     year: int
84 ) -> None:
85     """
86     Добавить работника в базу данных.
87     """
88     conn = sqlite3.connect(database_path)
89     cursor = conn.cursor()
90
91     # Получить идентификатор должности в базе данных.
92     # Если такой нет, то добавить информацию о новой должности.
93     cursor.execute(
94         """
95         SELECT post_id FROM posts WHERE post_title = ?
96         """,
97         (post,)
98     )
99     row = cursor.fetchone()
100     if row is None:
101         cursor.execute(
102             """
103             INSERT INTO posts (post_title) VALUES (?)
104             """,
105             (post,)
106         )
107         post_id = cursor.lastrowid
108     else:
109         post_id = row[0]
110
111     # Добавить информацию о новом работнике.
112     cursor.execute(
113         """
114         INSERT INTO workers (worker_name, post_id, worker_year)
115         VALUES (?, ?, ?)
116         """,
117         (name, post_id, year)
118     )
119
120
121     conn.commit()
122     conn.close()
123

```

```

125 def select_all(database_path: Path) -> t.List[t.Dict[str, t.Any]]:
126     """
127     Выбрать всех работников.
128     """
129     conn = sqlite3.connect(database_path)
130     cursor = conn.cursor()
131     cursor.execute(
132         """
133         SELECT workers.worker_name, posts.post_title, workers.worker_year
134         FROM workers
135         INNER JOIN posts ON posts.post_id = workers.post_id
136         """
137     )
138     rows = cursor.fetchall()
139
140     conn.close()
141     return [
142         {
143             "name": row[0],
144             "post": row[1],
145             "year": row[2],
146         }
147         for row in rows
148     ]
149
150
151 def select_by_period(
152     database_path: Path, period: int
153 ) -> t.List[t.Dict[str, t.Any]]:
154     """

```

```

155     Выбрать всех работников с периодом работы больше заданного.
156     """
157     conn = sqlite3.connect(database_path)
158     cursor = conn.cursor()
159
160     cursor.execute(
161         """
162         SELECT workers.worker_name, posts.post_title, workers.worker_year
163         FROM workers
164         INNER JOIN posts ON posts.post_id = workers.post_id
165         WHERE (strftime('%Y', date('now')) - workers.worker_year) >= ?
166         """,
167         (period,)
168     )
169     rows = cursor.fetchall()
170
171     conn.close()
172     return [
173         {
174             "name": row[0],
175             "post": row[1],
176             "year": row[2],
177         }
178         for row in rows
179     ]

```

```

182 def main(command_line=None):
183     # Создать родительский парсер для определения имени файла.
184     file_parser = argparse.ArgumentParser(add_help=False)
185     file_parser.add_argument(
186         "--db",
187         action="store",
188         required=False,
189         default=str(Path.home() / "workers.db"),
190         help="The data file name"
191     )
192     # Создать основной парсер командной строки.
193     parser = argparse.ArgumentParser("workers")
194     parser.add_argument(
195         "--version",
196         action="version",
197         version="%prog)s 0.1.0"
198     )
199     subparsers = parser.add_subparsers(dest="command")
200     # Создать субпарсер для добавления работника.
201     add = subparsers.add_parser(
202         "add",
203         parents=[file_parser],
204         help="Add a new worker"
205     )
206     add.add_argument(
207         "-n",
208         "--name",
209         action="store",
210         required=True,
211         help="The worker's name"
212     )

```

```

213     add.add_argument(
214         "-p",
215         "--post",
216         action="store",
217         help="The worker's post"
218     )
219     add.add_argument(
220         "-y",
221         "--year",
222         action="store",
223         type=int,
224         required=True,
225         help="The year of hiring"
226     )
227     # Создать субпарсер для отображения всех работников.
228     _ = subparsers.add_parser(
229         "display",
230         parents=[file_parser],
231         help="Display all workers"
232     )
233     # Создать субпарсер для выбора работников.
234     select = subparsers.add_parser(
235         "select",
236         parents=[file_parser],
237         help="Select the workers"
238     )

```

```

239     select_argument(
240         "-p",
241         "--period",
242         action="store",
243         type=int,
244         required=True,
245         help="The required period"
246     )
247     # Выполнить разбор аргументов командной строки.
248     args = parser.parse_args(command_line)
249     # Получить путь к файлу базы данных.
250     db_path = Path(args.db)
251     create_db(db_path)
252
253     # Добавить работника.
254     if args.command == "add":
255         add_worker(db_path, args.name, args.post, args.year)
256
257     # Отобразить всех работников.
258     elif args.command == "display":
259         display_workers(select_all(db_path))
260
261     # Выбрать требуемых работников.
262     elif args.command == "select":
263         display_workers(select_by_period(db_path, args.period))
264     pass
265
266
267 if __name__ == "__main__":
268     main()
269

```

Рисунок 1 – Пример

### 3. Результат работы примера.

```
PS C:\Users\user\Documents\2 курс\4-й семестр\анализ данных\2.21> python "Primer 1.py" display --db 1.db
```

No	Ф.И.О.	Должность	Год
1	Ivan	Student	2001
2	Ivan	Student	2001

Рисунок 2 – Работа примера

**4.** Выполнил индивидуальное задание 1.

```

1 #!/usr/bin/env python3
2 ## -*- coding: utf-8 -*-
3
4
5 import argparse
6 import sqlite3
7 import typing as t
8 from pathlib import Path
9
10
11 def display_route(routes: t.List[t.Dict[str, t.Any]]) -> None:
12     """
13     Отобразить список маршрутов
14     """
15     if routes:
16         line = '{}{}{}{}{}{}'.format(
17             '-' * 4,
18             '-' * 30,
19             '-' * 20,
20             '-' * 10
21         )
22         print(line)
23         print(
24             '|{:^4}|{: ^30}|{: ^20}|{: ^10}|'.format(
25                 "№",
26                 "Начальный пункт",
27                 "Конечный пункт",
28                 "№ маршрута"
29             )
30         )
31         print(line)

```

```

33         for idx, worker in enumerate(routes, 1):
34             print(
35                 '| {:>4} | {:<30} | {:<20} | {:>10} |'.format(
36                     idx,
37                     worker.get('start', ''),
38                     worker.get('finish', ''),
39                     worker.get('number', 0)
40                 )
41             )
42         print(line)
43     else:
44         print("Список маршрутов пуст.")
45
46
47 def create_db(database_path: Path) -> None:
48     """
49     Создать базу данных.
50     """
51     conn = sqlite3.connect(database_path)
52     cursor = conn.cursor()
53
54     # Создать таблицу с информацией о маршрутах.
55     cursor.execute(
56         """
57         CREATE TABLE IF NOT EXISTS numbers (
58             number_id INTEGER PRIMARY KEY AUTOINCREMENT,
59             number_title TEXT NOT NULL
60         )
61         """
62     )

```

```

63     # Создать таблицу с информацией о маршрутах.
64     cursor.execute(
65         """
66         CREATE TABLE IF NOT EXISTS routes (
67             routes_id INTEGER PRIMARY KEY AUTOINCREMENT,
68             start TEXT NOT NULL,
69             finish INTEGER NOT NULL,
70             number_id INTEGER NOT NULL,
71             FOREIGN KEY(number_id) REFERENCES numbers(number_id)
72         )
73         """
74     )
75     conn.close()
76
77
78 def add_route(
79     database_path: Path,
80     start: str,
81     finish: str,
82     number: int
83 ) -> None:
84     """
85     Добавить маршрут в базу данных.
86     """
87     conn = sqlite3.connect(database_path)
88     cursor = conn.cursor()
89
90     # Получить идентификатор маршрута в базе данных.
91     # Если такого нет, то добавить информацию о новом маршруте.

```

```

92     cursor.execute(
93         """
94         SELECT number_id FROM numbers WHERE number_title = ?
95         """
96         (number,)
97     )
98     row = cursor.fetchone()
99     if row is None:
100         cursor.execute(
101             """
102             INSERT INTO numbers (number_title) VALUES (?)
103             """
104             (number,)
105         )
106         number_id = cursor.lastrowid
107
108     else:
109         number_id = row[0]
110
111     # Добавить информацию о новом работнике.
112     cursor.execute(
113         """
114         INSERT INTO routes (start, finish, number_id)
115         VALUES (?, ?, ?)
116         """
117         (start, finish, number_id)
118     )
119
120     conn.commit()
121     conn.close()
122

```

```

124 def select_all(database_path: Path) -> t.List[t.Dict[str, t.Any]]:
125     """
126     Выбрать всех работников.
127     """
128     conn = sqlite3.connect(database_path)
129     cursor = conn.cursor()
130     cursor.execute(
131         """
132         SELECT routes.start, routes.finish, numbers.number_title
133         FROM routes
134         INNER JOIN numbers ON numbers.number_id = routes.number_id
135         """
136     )
137     rows = cursor.fetchall()
138
139     conn.close()
140     return [
141         {
142             "start": row[0],
143             "finish": row[1],
144             "number": row[2],
145         }
146         for row in rows
147     ]
148
149
150 def select_by_period(
151     database_path: Path, period: int
152 ) -> t.List[t.Dict[str, t.Any]]:
153     """
154     Выбрать определенный маршрут.
155     """

```



```

156     conn = sqlite3.connect(database_path)
157     cursor = conn.cursor()
158
159     cursor.execute(
160         """
161         SELECT routes.start, routes.finish, numbers.number_title
162         FROM routes
163         INNER JOIN numbers ON numbers.number_id = routes.number_id
164         WHERE numbers.number_title == ?
165         """,
166         (period,)
167     )
168     rows = cursor.fetchall()
169
170     conn.close()
171     return [
172         {
173             "start": row[0],
174             "finish": row[1],
175             "number": row[2],
176         }
177         for row in rows
178     ]

```

```

181 def main(command_line=None):
182     # Создать родительский парсер для определения имени файла.
183     file_parser = argparse.ArgumentParser(add_help=False)
184     file_parser.add_argument(
185         "--db",
186         action="store",
187         required=False,
188         default=str(Path.home() / "workers.db"),
189         help="The data file name"
190     )
191     # Создать основной парсер командной строки.
192     parser = argparse.ArgumentParser("routes")
193     parser.add_argument(
194         "--version",
195         action="version",
196         version="% (prog)s 0.1.0"
197     )
198     subparsers = parser.add_subparsers(dest="command")
199     # Создать субпарсер для добавления маршрута.
200     add = subparsers.add_parser(
201         "add",
202         parents=[file_parser],
203         help="Add a new route"
204     )
205     add.add_argument(
206         "-s",
207         "--start",
208         action="store",
209         required=True,
210         help="The start of the route"
211     )

```

```

212     add.add_argument(
213         "-f",
214         "--finish",
215         action="store",
216         help="The finish of the route"
217     )
218     add.add_argument(
219         "-n",
220         "--number",
221         action="store",
222         type=int,
223         required=True,
224         help="The number of the route"
225     )
226     # Создать субпарсер для отображения всех маршрутов.
227     _ = subparsers.add_parser(
228         "display",
229         parents=[file_parser],
230         help="Display all routes"
231     )
232     # Создать субпарсер для выбора маршрута.
233     select = subparsers.add_parser(
234         "select",
235         parents=[file_parser],
236         help="Select the route"
237     )
238     select.add_argument(
239         "-N",
240         "--period",
241         action="store",
242         type=int,
243         required=True,
244         help="The route"
245     )
246     # Выполнить разбор аргументов командной строки.
247     args = parser.parse_args(command_line)
248     # Получить путь к файлу базы данных.
249     db_path = Path(args.db)
250     create_db(db_path)
251
252     # Добавить маршрут.
253     if args.command == "add":
254         add_route(db_path, args.start, args.finish, args.number)
255
256     # Отобразить все маршруты.
257     elif args.command == "display":
258         display_route(select_all(db_path))
259
260     # Выбрать требуемые маршруты.
261     elif args.command == "select":
262         display_route(select_by_period(db_path, args.period))
263     else:
264         pass
265
266     if __name__ == '__main__':
267         main()

```

Рисунок 3 – Код задания 1.

5. Чтобы импортировать БД без использования опции `--csv`, необходимо сначала ввести команды `.mode csv`, и только потом импортировать БД.

```
PS C:\Users\user\Documents\2 курс\4-й семестр\анализ данных\2.21> python "ind1.py" display --db 2.db
```

№	Начальный пункт	Конечный пункт	№ маршрута
1	Астрахань	Ростов	1

Рисунок 4 – Работа задания один

6. Выполнил индивидуальное задание 2. Изучил пакет `psycopg2`. Добавил подключение к базе данных.

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  import argparse
6  import sqlite3
7  import typing as t
8  from pathlib import Path
9  import psycopg2
10
11
12  def connect():
13      conn = psycopg2.connect(
14          user="postgres",
15          password="123321",
16          host="127.0.0.1",
17          port="5432",
18          database="postgres"
19      )
20
21  return conn
```

Рисунок 5 – Задача 5. Количество городов по часовым поясам.

7. Результат работы индивидуального задания 2.

```
PS C:\Users\user\Documents\2 курс\4-й семестр\анализ данных\2.21> python "ind2.py" display
```

№	Начальный пункт	Конечный пункт	№ маршрута
1	Ivan	Student	2001
2	Москва	Питер	42
3	Ставрополь	Пермь	32
4	Астрахань	Ростов	1
5	Астрахань	Ростов	1
6	Астрахань	Ростов	1

Рисунок 6 – Работа задания 2

## Ответы на вопросы:

### 1. Каково назначение модуля sqlite3?

Модуль sqlite3 предназначен для взаимодействия с СУБД SQLite.

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

Объект соединения создается с помощью функции `connect()`. 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?

С помощью функции: `memory`:

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

Сначала импортируется модуль `sqlite3`, а затем определяется функция с именем `sql_connection`. Внутри функции у нас есть блок `try`, где функция `connect()` возвращает объект соединения после установления соединения. В случае возникновения ошибок при установке соединения с базой данных выполняются операторы блока `except`, в котором в данном случае просто печатается содержимое объекта ошибки. После этого вне зависимости от того возникло или нет исключение по работе с базой данных, выполняются операторы блока `finally`, в котором соединение закрывается. Заккрытие соединения необязательно, но это хорошая практика программирования, поэтому вы освобождаете память от любых неиспользуемых ресурсов.

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

Чтобы вставить данные в таблицу, используется оператор `INSERT INTO`.

### 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 (\*).

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 .

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

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.

**Вывод:** были исследовано взаимодействие с базами данных SQLite3 с помощью языка программирования Python.