

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

Институт цифрового развития  
Кафедра инфокоммуникаций

ОТЧЁТ

по лабораторной работе №2.21

**Дисциплина: «Анализ данных»**

**Тема: «Взаимодействие с базами данных SQLite3 с помощью языка  
программирования Python»**

Выполнил:

Елифанов Алексей Александрович  
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 Роман Александрович

---

(подпись)

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

Ставрополь, 2024 г.

Цель: изучить взаимодействие с базами данных SQLite3 с помощью языка программирования Python.

Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.
2. Проработал пример лабораторной работы: Для примера 1 лабораторной работы 2.17 реализуйте возможность хранения данных в базе данных SQLite3.

```

> p prim1 -h
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
> p prim1 display -h
usage: workers display [-h] [--db DB]

options:
  -h, --help            show this help message and exit
  --db DB              The database file name
> p prim1 display --db prim1.db
+-----+-----+-----+-----+
| No |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
|  1 | alexei                   |      student        |      2022     |
+-----+-----+-----+-----+
|  2 | victorkozh              |      student        |      2020     |
+-----+-----+-----+-----+
|  3 | victor                   |      children       |      2017     |
+-----+-----+-----+-----+
|  4 | al                       |      student        |      2010     |
+-----+-----+-----+-----+
> p prim1 add -h
usage: workers add [-h] [--db DB] -n NAME [-p POST] -y YEAR

options:
  -h, --help            show this help message and exit
  --db DB              The database file name
  -n NAME, --name NAME  The worker's name
  -p POST, --post POST  The worker's post
  -y YEAR, --year YEAR  The year of hiring
> p prim1 add --db prim1.db -n vadim -p children -y 2018
> p prim1 select -h
usage: workers select [-h] [--db DB] -P PERIOD

options:
  -h, --help            show this help message and exit
  --db DB              The database file name
  -P PERIOD, --period PERIOD
                        The required period
> p prim1 select --db prim1.db -P 3
+-----+-----+-----+-----+
| No |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
|  1 | victorkozh              |      student        |      2020     |
+-----+-----+-----+-----+
|  2 | victor                   |      children       |      2017     |
+-----+-----+-----+-----+
|  3 | al                       |      student        |      2010     |
+-----+-----+-----+-----+
|  4 | vadim                   |      children       |      2018     |
+-----+-----+-----+-----+

```

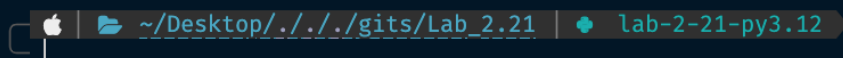


Рисунок 1. Работа программы prim1.py

3. 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.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import argparse
import sqlite3
import typing as t
from pathlib import Path

def create_db(database_path: Path) -> None:
    """
    Создать базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    # Создать таблицу с информацией о путях.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS start (
            start_id INTEGER PRIMARY KEY AUTOINCREMENT,
            start_point TEXT UNIQUE NOT NULL
        );
        """
    )
    # Создать таблицу с информацией о работниках.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS routes (
            start_id INTEGER NOT NULL,
            route_number INTEGER PRIMARY KEY,
            end_point TEXT NOT NULL,
            FOREIGN KEY(start_id) REFERENCES start(start_id)
        )
        """
    )
    conn.close()

def select_all(database_path: Path) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать все маршруты.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
```

```

cursor.execute(
    """
    SELECT start.start_point, routes.end_point, routes.route_number
    FROM start
    INNER JOIN routes ON routes.start_id = start.start_id
    """
)
rows = cursor.fetchall()
conn.close()
return [
    {
        "начальный пункт": row[0],
        "конечный пункт": row[1],
        "номер маршрута": row[2],
    }
    for row in rows
]
def add_route(database_path: Path, start: str, end: str, count: int) -> None:
    """
    Добавить данные о маршруте.
    """
    start = start.lower()
    end = end.lower()
    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 start_id FROM start WHERE start_point = ?
        """,
        (start,),
    )
    row = cursor.fetchone()
    if not row:
        cursor.execute(
            """
            INSERT INTO start (start_point) VALUES (?)
            """,
            (start,),
        )
        start_id = cursor.lastrowid
    else:
        start_id = row[0]
    # Добавить информацию о новом работнике.

```

```

cursor.execute(
    """
    INSERT INTO routes (start_id, route_number, end_point)
    VALUES (?, ?, ?)
    """,
    (start_id, count, end),
)
conn.commit()
conn.close()
def display_routes(routes: t.List[t.Dict[str, t.Any]]) -> None:
    """
    Отобразить список маршрутов.
    """
    if routes:
        line = "+-{}--{}--{}-+".format("-" * 30, "-" * 20, "-" * 8)
        print(line)
        print("| {:^30} | {:^20} | {:^8} |".format("Начало", "Конец", "Номер"))
        print(line)
        for route in routes:
            print(
                "| {:<30} | {:<20} | {:>8} |".format(
                    route.get("начальный пункт", ""),
                    route.get("конечный пункт", ""),
                    route.get("номер маршрута", ""),
                )
            )
            print(line)
        else:
            print("Список маршрутов пуст.")
def select_routes(
    database_path: Path, name_point: str
) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать маршруты с заданным пунктом отправления или прибытия.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT s.start_point, r.end_point, r.route_number
        FROM start s
        JOIN routes r ON s.start_id = r.start_id
        WHERE s.start_point = ? OR r.end_point = ?
        """,
        (name_point, name_point),
    )

```

```

)
rows = cursor.fetchall()
conn.close()
return [
    {
        "начальный пункт": row[0],
        "конечный пункт": row[1],
        "номер маршрута": row[2],
    }
    for row in rows
]
def main(command_line=None):
    """
    Главная функция программы.
    """
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "--db",
        action="store",
        default=str(Path.home() / "routes.db"),
        help="The database 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 route start"
    )
    add.add_argument(
        "-e", "--end", action="store", required=True, help="The route endpoint"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        type=int,
        required=True,
        help="The number of route",
    )
    _ = subparsers.add_parser(

```

```

    "list", parents=[file_parser], help="Display all routes"
)
select = subparsers.add_parser(
    "select", parents=[file_parser], help="Select the routes"
)
select.add_argument(
    "-p",
    "--point",
    action="store",
    required=True,
    help="Routes starting or ending at this point",
)
args = parser.parse_args(command_line)
# Получить путь к файлу базы данных.
db_path = Path(args.db)
create_db(db_path)
match args.command:
    case "add":
        add_route(db_path, args.start, args.end, args.number)
    case "list":
        display_routes(select_all(db_path))
    case "select":
        selected = select_routes(db_path, args.point)
        display_routes(selected)
if __name__ == "__main__":
    main()

```



```

> p ind --help
usage: routes [-h] [--version] {add,list,select} ...

positional arguments:
  {add,list,select}
    add                Add a new route
    list               Display all routes
    select             Select the routes

options:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
> p ind list --db ind.db
+-----+-----+-----+
|          Начало          |          Конец          |   Номер   |
+-----+-----+-----+
| stav                    | brasil                  |    1    |
| nevin                   | brasil                  |    2    |
| stav                    | nevin                   |    4    |
+-----+-----+-----+
> p ind add --db ind.db -s brasil -e moscow -n 3
> p ind select --db ind.db -p brasil
+-----+-----+-----+
|          Начало          |          Конец          |   Номер   |
+-----+-----+-----+
| stav                    | brasil                  |    1    |
| nevin                   | brasil                  |    2    |
| brasil                  | moscow                  |    3    |
+-----+-----+-----+
> p ind select --db ind.db -p stav
+-----+-----+-----+
|          Начало          |          Конец          |   Номер   |
+-----+-----+-----+
| stav                    | brasil                  |    1    |
| stav                    | nevin                   |    4    |
+-----+-----+-----+
> p ind list
Список маршрутов пуст.
> p ind add -s moscow -e stav -n 11
> p ind list
+-----+-----+-----+
|          Начало          |          Конец          |   Номер   |
+-----+-----+-----+
| moscow                  | stav                    |    11    |
+-----+-----+-----+

```

Рисунок 2. Результат работы программы ind.py

4. Выполнил задание повышенной сложности: самостоятельно изучите 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.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import argparse
import typing as t
import psycopg2 as pgsq
def create_db(database_path: str) -> None:
    """
    Создать базу данных.
    """
    try:
        conn = pgsq.connect(
            dbname=database_path, user="aleksejepifanov", host="localhost"
        )
    except pgsq.OperationalError:
        conn = pgsq.connect(
            dbname="postgres", user="aleksejepifanov", host="localhost"
        )
        cur = conn.cursor()
        conn.autocommit = True
        cur.execute(
            f"CREATE DATABASE {database_path}",
        )
        conn.autocommit = False
        conn.close()
        conn = pgsq.connect(
            dbname=database_path, user="aleksejepifanov", host="localhost"
        )
        cursor = conn.cursor()
        # Создать таблицу с информацией о путях.
        cursor.execute(
            """
            CREATE TABLE IF NOT EXISTS start (
                start_id SERIAL PRIMARY KEY,
                start_point TEXT UNIQUE NOT NULL
            );
            """
        )
        # Создать таблицу с информацией о работниках.
        cursor.execute(
            """
```

```

CREATE TABLE IF NOT EXISTS routes (
    start_id INTEGER NOT NULL,
    route_number INTEGER PRIMARY KEY,
    end_point TEXT NOT NULL,
    FOREIGN KEY(start_id) REFERENCES start(start_id)
)
"""

)
conn.commit()
conn.close()
def select_all(database_path: str) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать все маршруты.
    """
    conn = conn = pgsql.connect(
        dbname=database_path, user="aleksejepifanov", host="localhost"
    )
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT start.start_point, routes.end_point, routes.route_number
        FROM start
        INNER JOIN routes ON routes.start_id = start.start_id
        """
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "начальный пункт": row[0],
            "конечный пункт": row[1],
            "номер маршрута": row[2],
        }
        for row in rows
    ]
def add_route(database_path: str, start: str, end: str, count: int) -> None:
    """
    Добавить данные о маршруте.
    """
    start = start.lower()
    end = end.lower()
    conn = pgsql.connect(
        dbname=database_path, user="aleksejepifanov", host="localhost"
    )
    cursor = conn.cursor()

```

```

# Получить идентификатор должности в базе данных.
# Если такой записи нет, то добавить информацию о новой должности.
cursor.execute(
    """
    SELECT start_id FROM start WHERE start_point = %s
    """,
    (start,),
)
row = cursor.fetchone()
if not row:
    cursor.execute(
        """
        INSERT INTO start (start_point) VALUES (%s) RETURNING start_id
        """,
        (start,),
    )
    start_id = cursor.fetchone()[0]
else:
    start_id = row[0]
# Добавить информацию о новом работнике.
cursor.execute(
    """
    INSERT INTO routes (start_id, route_number, end_point)
    VALUES (%s, %s, %s)
    """,
    (start_id, count, end),
)
conn.commit()
conn.close()
def display_routes(routes: t.List[t.Dict[str, t.Any]]) -> None:
    """
    Отобразить список маршрутов.
    """
    if routes:
        line = "+-{-}-{-}-{-}-+ ".format("-" * 30, "-" * 20, "-" * 8)
        print(line)
        print("| {:^30} | {:^20} | {:^8} | ".format("Начало", "Конец", "Номер"))
        print(line)
        for route in routes:
            print(
                "| {:<30} | {:<20} | {:>8} | ".format(
                    route.get("начальный пункт", ""),
                    route.get("конечный пункт", ""),
                    route.get("номер маршрута", ""),
                )
            )

```

```

        )
        print(line)
    else:
        print("Список маршрутов пуст.")
def select_routes(
    database_path: str, name_point: str
) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать маршруты с заданным пунктом отправления или прибытия.
    """
    conn = pgsql.connect(
        dbname=database_path, user="aleksejepifanov", host="localhost"
    )
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT s.start_point, r.end_point, r.route_number
        FROM start s
        JOIN routes r ON s.start_id = r.start_id
        WHERE s.start_point = %s OR r.end_point = %s
        """,
        (name_point, name_point),
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "начальный пункт": row[0],
            "конечный пункт": row[1],
            "номер маршрута": row[2],
        }
        for row in rows
    ]
def main(command_line=None):
    """
    Главная функция программы.
    """
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "--db",
        action="store",
        default="routes",
        help="The database 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 route start"
)
add.add_argument(
    "-e", "--end", action="store", required=True, help="The route endpoint"
)
add.add_argument(
    "-n",
    "--number",
    action="store",
    type=int,
    required=True,
    help="The number of route",
)
_ = subparsers.add_parser(
    "list", parents=[file_parser], help="Display all routes"
)
select = subparsers.add_parser(
    "select", parents=[file_parser], help="Select the routes"
)
select.add_argument(
    "-p",
    "--point",
    action="store",
    required=True,
    help="Routes starting or ending at this point",
)
args = parser.parse_args(command_line)
# Получить путь к файлу базы данных.
db_path = args.db
create_db(db_path)
match args.command:
    case "add":
        add_route(db_path, args.start, args.end, args.number)
    case "list":
        display_routes(select_all(db_path))
    case "select":
        selected = select_routes(db_path, args.point)

```

```

        display_routes(selected)
if __name__ == "__main__":
    main()

```

```

> p hard add --db hard -s stav -e nevin -n 14
> p hard add --db hard -s nevin -e moscow -n 11
> p hard list
Список маршрутов пуст.
> p hard --db hard list
usage: routes [-h] [--version] {add,list,select} ...
routes: error: argument command: invalid choice: 'hard' (choose from 'add', 'list', 'select')
> p hard list --db hard
+-----+-----+-----+
|          Начало          |          Конец          |          Номер          |
+-----+-----+-----+
| stav                    | nevin                    |          14            |
| nevin                   | moscow                   |          11            |
+-----+-----+-----+
> p hard add -s nevin -e rostow -n 5
> p hard list
+-----+-----+-----+
|          Начало          |          Конец          |          Номер          |
+-----+-----+-----+
| nevin                   | rostow                   |          5             |
+-----+-----+-----+
> p hard add --db hard -s moscow -e stav -n 1
> p hard select --db hard -p moscow
+-----+-----+-----+
|          Начало          |          Конец          |          Номер          |
+-----+-----+-----+
| nevin                   | moscow                   |          11            |
| moscow                  | stav                     |          1             |
+-----+-----+-----+
> p hard select --db hard -p stav
+-----+-----+-----+
|          Начало          |          Конец          |          Номер          |
+-----+-----+-----+
| stav                    | nevin                    |          14            |
| moscow                  | stav                     |          1             |
+-----+-----+-----+

```

Рисунок 3. Результат работы программы hard.py

Контрольные вопросы:

1. Каково назначение модуля sqlite3?

Инструменты для работы с конкретной СУБД не являются базовыми командами и объектами самого языка. Обычно они подключаются через импорт модуля или библиотеки. 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?

Объект соединения создается с помощью функции `connect()` с параметром – путем к базе данных.

Для взаимодействия с базой данных SQLite3 в Python необходимо создать объект `cursor`. Создать его можно 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.

Теперь можно использовать объект `cursor` для вызова метода `execute()` для выполнения любых SQL-запросов.

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

```
con = sqlite3.connect(':memory:')
```

4. Как корректно завершить работу с базой данных SQLite3?

```
conn.close()
```

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

```
# Сам запрос
```

```
sql_query = "INSERT INTO my_table (column1, column2, column3)
VALUES (?, ?, ?)"
```

```
# Данные для вставки
```

```
data_to_insert = ('value1', 'value2', 'value3')
```

```
# Выполнение SQL-запроса с данными
```

```
cursor.execute(sql_query, data_to_insert)
```

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 .

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

```
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
cursor.execute("SELECT name FROM sqlite_master WHERE
type='table';")
# Получение списка таблиц
tables = cursor.fetchall()
```

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: в результате выполнения работы были исследованы взаимодействия с базами данных SQLite3 с помощью языка программирования python.