

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 №2

Выполнила:  
Беседина Инга Олеговна  
2 курс, группа ИВТ-б-о-22-1,  
09.03.01 «Информатика и  
вычислительная техника», очная  
форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Р. А., канд. технических  
наук, доцент, 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:** Изучить взаимодействие с базами данных SQLite3 с помощью языка программирования Python

### Ход работы

Пример 1. Создание базы данных.

```
import sqlite3
from sqlite3 import Error

def sql_connection():
    try:
        con = sqlite3.connect(':memory:')
        print("Connection is established: Database is created in memory")
    except Error:
        print(Error)

    finally:
        con.close()

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

```
(venv) PS C:\Rep\DA_7\Project> & C:/Users/besed/AppData/Local/Programs/Python/Python311/python.exe c:/Rep/DA_7/Project/ex1.py
Connection is established: Database is created in memory
```

Рисунок 1. Результат работы программы

Пример 2. Создание таблиц.

```
import sqlite3
from sqlite3 import Error

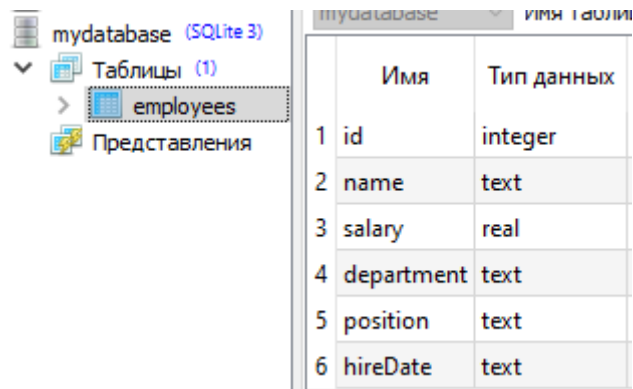
def sql_connection():
    try:
        con = sqlite3.Connection('mydatabase.db')
        return con
    except Error:
        print(Error)

    return None
```

```
def sql_table(con):
    cursor_obj = con.cursor()
    cursor_obj.execute(
        """
CREATE TABLE employees (
    id integer PRIMARY KEY,
    name text,
    salary real,
    department text,
    position text,
    hireDate text)
        """
    )

    con.commit()

if __name__ == "__main__":
    con = sql_connection()
    sql_table(con)
```



The screenshot shows a database management interface. On the left, a tree view displays 'mydatabase (SQLite 3)' with a sub-entry 'Таблицы (1)' (Tables (1)) containing the 'employees' table. On the right, a detailed view of the 'employees' table structure is shown, listing columns and their data types.

	Имя	Тип данных
1	id	integer
2	name	text
3	salary	real
4	department	text
5	position	text
6	hireDate	text

Рисунок 2. Созданная таблица

### Пример 3. Вставка данных в таблицу.

```
import sqlite3

con = sqlite3.connect('mydatabase.db')

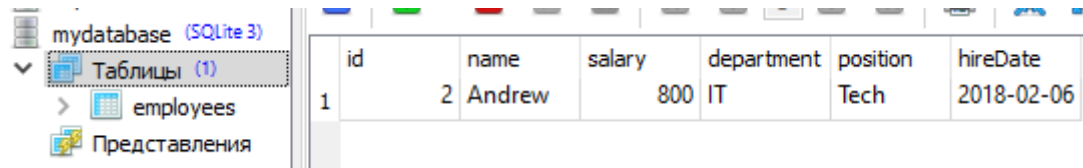
def sql_insert(con, entities):
    cursor_obj = con.cursor()
    cursor_obj.execute(
        """
INSERT INTO employees(id, name, salary, department, position, hireDate)
VALUES(?, ?, ?, ?, ?, ?)
        """
    )
```

```

"""
    entities
)
con.commit()

entities = (2, 'Andrew', 800, 'IT', 'Tech', '2018-02-06')
sql_insert(con, entities)

```



The screenshot shows the SQLite3 database interface. On the left, a tree view shows 'mydatabase (SQLite 3)' with a folder 'Таблицы (1)' containing the 'employees' table. The main area displays the 'employees' table with the following data:

id	name	salary	department	position	hireDate	
1	2	Andrew	800	IT	Tech	2018-02-06

Рисунок 3. Вставленные данные

Пример 4. Обновление данных в таблицах.

```

import sqlite3

con = sqlite3.connect('mydatabase.db')

def sql_update(con):
    cursor_obj = con.cursor()
    cursor_obj.execute(
        "UPDATE employees SET name = 'Rogers' where id = 2"
    )
    con.commit()

sql_update(con)

```

mydatabase (SQLite 3)

Таблицы (1)

employees

Представления

	id	name	salary	department	position	hireDate
1	2	Rogers	800	IT	Tech	2018-02-06

Рисунок 4. Обновленные данные

Пример 5. Выборка данных из таблицы

```

import sqlite3

con = sqlite3.connect('mydatabase.db')

```

```
def sql_fetch(con):
    cursor_obj = con.cursor()
    cursor_obj.execute("SELECT * FROM employees")
    [print(row) for row in cursor_obj.fetchall()]

sql_fetch(con)
```

```
(venv) PS C:\Rep\DA_7\Project> & C:/Users/besed/AppData/Local/Programs/Python/Python311/python.exe c:/Rep/DA_7/Project/ex5.py
(2, 'Rogers', 800.0, 'IT', 'Tech', '2018-02-06')
```

Рисунок 5. Выборка данных из таблицы

### Пример 6. Получение списка таблиц.

```
import sqlite3

con = sqlite3.connect('mydatabase.db')

def sql_fetch(con):
    cursor_obj = con.cursor()
    cursor_obj.execute(
        "SELECT name from sqlite_master where type='table'"
    )

    print(cursor_obj.fetchall())

sql_fetch(con)
```

```
(venv) PS C:\Rep\DA_7\Project> & C:/Users/besed/AppData/Local/Programs/Python/Python311/python.exe c:/Rep/DA_7/Project/ex6.py
[('employees',)]
```

Рисунок 6. Список таблиц

### Пример 7. Проверка существования таблицы.

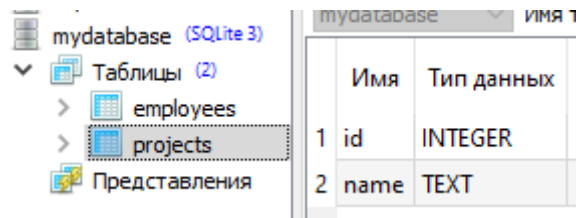
```
import sqlite3

con = sqlite3.connect('mydatabase.db')

def sql_fetch(con):
    cursor_obj = con.cursor()
    cursor_obj.execute(
        "CREATE TABLE IF NOT EXISTS projects(id INTEGER, name TEXT)"
    )
```

```
con.commit()

sql_fetch(con)
```



Имя	Тип данных
1 id	INTEGER
2 name	TEXT

Рисунок 7. Созданная таблица

Пример 8. SQLite3 Execute many (массовая вставка).

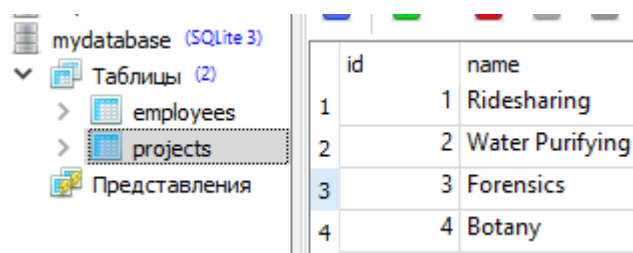
```
import sqlite3

con = sqlite3.connect('mydatabase.db')

cursor_obj = con.cursor()
cursor_obj.execute(
    "CREATE TABLE IF NOT EXISTS projects(id INTEGER, name TEXT)"
)

data = [
    (1, "Ridesharing"),
    (2, "Water Purifying"),
    (3, "Forensics"),
    (4, "Botany")
]
cursor_obj.executemany("INSERT INTO projects VALUES (?, ?)", data)

con.commit()
```



id	name
1	1 Ridesharing
2	2 Water Purifying
3	3 Forensics
4	4 Botany

Рисунок 8. Данные таблицы

### Пример 9. SQLite3 datetime .

```
import sqlite3
import datetime


con = sqlite3.connect('mydatabase.db')

cursor_obj = con.cursor()
cursor_obj.execute(
    """
    CREATE TABLE IF NOT EXISTS assignments(
        id INTEGER, name TEXT, date DATE
    )
    """
)

data = [
    (1, "Ridesharing", datetime.date(2017, 1, 2)),
    (2, "Water Purifying", datetime.date(2018, 3, 4))
]

cursor_obj.executemany("INSERT INTO assignments VALUES(?, ?, ?)", data)

con.commit()
```



The screenshot shows a SQLite3 database window titled 'mydatabase (SQLite 3)'. On the left, a tree view shows 'Таблицы (3)' (Tables (3)) with 'assignments', 'employees', and 'projects'. The 'assignments' table is selected, and its data is displayed in a table on the right. The table has columns 'id', 'name', and 'date'. It contains 4 rows of data.

	id	name	date
1	1	Ridesharing	2017-01-02
2	2	Water Purifying	2018-03-04
3	1	Ridesharing	2017-01-02
4	2	Water Purifying	2018-03-04

Рисунок 9. Добавленная таблица

Пример 10: Для примера 1 лабораторной работы 2.17 реализуйте возможность хранения данных в базе данных SQLite3.

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

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

```

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:
    4
    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 работников.
    elif args.command == "select":
        display_workers(select_by_period(db_path, args.period))
    pass

if __name__ == "__main__":
    main()

```

```

(venv) PS C:\Rep\DA_7\Project> python ex10.py add --db="C:\Rep\DA_7\workers.db" --name="Сидоров Сидор" --post="Главный инженер
(venv) PS C:\Rep\DA_7\Project> python ex10.py add --db="C:\Rep\DA_7\workers.db" --name="Журавлёв А.В" --post="Менеджер" --year=2005
(venv) PS C:\Rep\DA_7\Project> python ex10.py add --db="C:\Rep\DA_7\workers.db" --name="Михайлова С.А" --post="Дизайнер" --year=2014

```

Рисунок 10. Добавление данных

```
(venv) PS C:\Rep\DA_7\Project> python ex10.py display --db="C:\Rep\DA_7\workers.db"
```

№	Ф.И.О.	Должность	Год
1	Сидоров Сидор	Главный инженер	2012
2	Журавлёв А.В	Менеджер	2005
3	Михайлова С.А	Дизайнер	2014
4	Третьякова К.Б	Директор	2002

Рисунок 11. Вывод данных в консоль

	worker_id	worker_name	post_id	worker_year
1	1	Сидоров Сидор	1	2012
2	2	Журавлёв А.В	2	2005
3	3	Михайлова С.А	3	2014
4	4	Третьякова К.Б	4	2002

Рисунок 12. Данные в базе данных

```
(venv) PS C:\Rep\DA_7\Project> python ex10.py select --db="C:\Rep\DA_7\workers.db" --period=15
```

№	Ф.И.О.	Должность	Год
1	Журавлёв А.В	Менеджер	2005
2	Третьякова К.Б	Директор	2002

Рисунок 13. Вывод работников, чей стаж превышает заданное значение

## Индивидуальное задание

### 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 display_students(staff: t.List[t.Dict[str, t.Any]]) -> None:
    """
    Отобразить список студентов.
    """
    # Проверить, что список студентов не пуст.
    if staff:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 14
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^14} |'.format(
                "№",
                "Ф.И.О.",
                "Группа",
                "Оценки"
            )
        )
        print(line)

        # Вывести данные о всех студентах.
        for idx, student in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>14} |'.format(
                    idx,
                    student.get('name', ''),
                    student.get('group', ''),
                    student.get('grades', '')
                )
            )
            print(line)

    else:
        print("Список студентов пуст.")

def create_db(database_path: Path) -> None:
    """
    Создать базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

```

```

# Создать таблицу с информацией о группах.
cursor.execute(
    """
    CREATE TABLE IF NOT EXISTS groups (
        group_id INTEGER PRIMARY KEY AUTOINCREMENT,
        group_title TEXT NOT NULL
    )
    """
)

# Создать таблицу с информацией о студентах.
cursor.execute(
    """
    CREATE TABLE IF NOT EXISTS students (
        student_id INTEGER PRIMARY KEY AUTOINCREMENT,
        student_name TEXT NOT NULL,
        group_id INTEGER NOT NULL,
        student_grades TEXT NOT NULL,
        FOREIGN KEY(group_id) REFERENCES groups(group_id)
    )
    """
)

conn.close()

def add_student(
    database_path: Path,
    name: str,
    group: str,
    grades: str
) -> None:
    """
    Добавить студента в базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    # Получить идентификатор имени в базе данных.
    # Если такой записи нет, то добавить информацию о новом имени.
    cursor.execute(
        """
        SELECT group_id FROM groups WHERE group_title = ?
        """,
        (group,)
    )
    row = cursor.fetchone()
    if row is None:
        4

```



```

        cursor.execute(
            """
            INSERT INTO groups (group_title) VALUES (?)
            """,
            (group,)
        )
        group_id = cursor.lastrowid
    else:
        group_id = row[0]

    # Добавить информацию о новом студенте.
    cursor.execute(
        """
        INSERT INTO students (student_name, group_id, student_grades)
        VALUES (?, ?, ?)
        """,
        (name, group_id, grades)
    )

    conn.commit()
    conn.close()

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

    cursor.execute(
        """
        SELECT students.student_name, groups.group_title,
students.student_grades
        FROM students
        INNER JOIN groups ON groups.group_id = students.group_id
        """
    )
    rows = cursor.fetchall()

    data_with_avg = []
    for row in rows:
        grades = list(map(int, row[2].split(',')))
        average = sum(grades) / len(grades)
        data_with_avg.append((row[0], row[1], row[2], average))

    # Сортировка данных по среднему баллу

```

```

sorted_data = sorted(data_with_avg, key=lambda x: x[3])

conn.close()
return [
    {
        "name": row[0],
        "group": row[1],
        "grades": row[2],
    }
    for row in sorted_data
]

def select_students(
    database_path: Path
) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать всех студентов, имеющих оценки 4 и 5.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    # вывод на дисплей фамилий и групп для всех студентов, имеющих оценки 4 и
5
    # Извлечение данных из столбца базы данных
    cursor.execute("""
        SELECT students.student_name, groups.group_title,
students.student_grades
        FROM students
        INNER JOIN groups ON groups.group_id = students.group_id
    """)
    rows = cursor.fetchall()

    selected_data = []
    for row in rows:
        grades = list(map(int, row[2].split(',')))
        if 2 not in grades and 3 not in grades:
            average = sum(grades) / len(grades)
            selected_data.append((row[0], row[1], row[2], average))

    selected_data = sorted(selected_data, key=lambda x: x[3])

    conn.close()

    return [
        {
            "name": row[0],

```

```

        "group": row[1],
        "grades": row[2],
    }
    for row in selected_data
]

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() / "students.db"),
        help="The database file name"
    )

    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("students")
    parser.add_argument(
        "--version",
        action="version",
        version="% (prog)s 0.1.0"
    )

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

    # Создать субпарсер для добавления студента.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new student"
    )
    add.add_argument(
        "-n",
        "--name",
        action="store",
        required=True,
        help="The student's name"
    )
    add.add_argument(
        "-g",
        "--group",
        action="store",
        help="The student's group"
    )

```

```

add.add_argument(
    "--grades",
    action="store",
    required=True,
    help="Grades received"
)

# Создать субпарсер для отображения всех студентов.
_ = subparsers.add_parser(
    "display",
    parents=[file_parser],
    help="Display all students"
)

# Создать субпарсер для выбора студентов.
select = subparsers.add_parser(
    "select",
    parents=[file_parser],
    help="Select the students"
)

# 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 студента.
if args.command == "add":
    add_student(db_path, args.name, args.group, args.grades)

# Отобразить всех студентов.
elif args.command == "display":
    display_students(select_all(db_path))

# Выбрать требуемых студентов.
elif args.command == "select":
    display_students(select_students(db_path))
    pass

if __name__ == "__main__":
    main()

```

```
(venv) PS C:\Rep\DA_7\Project> python indv.py add --db="C:\Rep\DA_7\Students.db" --name="Третьякова" --group="фил-б-о-20-1" --grades="4,4,4,5,4"
(venv) PS C:\Rep\DA_7\Project> python indv.py add --db="C:\Rep\DA_7\Students.db" --name="Журавлёв" --group="стр-б-о-22-1" --grades="5,5,5,4,5"
(venv) PS C:\Rep\DA_7\Project> python indv.py add --db="C:\Rep\DA_7\Students.db" --name="Сидоров" --group="нгд-б-о-22-1" --grades="3,4,3,3,3"
(venv) PS C:\Rep\DA_7\Project> python indv.py add --db="C:\Rep\DA_7\Students.db" --name="Михайлова" --group="хим-б-о-21-1" --grades="3,3,4,3,4"
```

Рисунок 14. Добавление данных в базу данных

Табличный вид    Форма

1

	student_id	student_name	group_id	student_grades
1	1	Третьякова	1	4,4,4,5,4
2	2	Журавлёв	2	5,5,5,4,5
3	3	Сидоров	3	3,4,3,3,3
4	4	Михайлова	4	3,3,4,3,4

Рисунок 15. Первая таблица базы данных

Табличный вид    Форма

1

	group_id	group_title
1	1	фил-б-о-20-1
2	2	стр-б-о-22-1
3	3	нгд-б-о-22-1
4	4	хим-б-о-21-1

Рисунок 16. Вторая таблица базы данных

```
(venv) D:\Rep\DA_7\Project>python indv1.py display --db="D:\Rep\DA_7\Students.db"
+-----+-----+-----+-----+
| № | Ф.И.О. | Группа | Оценки |
+-----+-----+-----+-----+
| 1 | Сидоров | нгд-б-о-22-1 | 3,4,3,3,3 |
| 2 | Михайлова | хим-б-о-21-1 | 3,3,4,3,4 |
| 3 | Третьякова | фил-б-о-20-1 | 4,4,4,5,4 |
| 4 | Журавлёв | стр-б-о-22-1 | 5,5,5,4,5 |
+-----+-----+-----+-----+
```

Рисунок 17. Вывод информации обо всех студентах в консоль

```
(venv) D:\Rep\DA_7\Project>python indv1.py select --db="D:\Rep\DA_7\Students.db"
+-----+-----+-----+-----+
| № | Ф.И.О. | Группа | Оценки |
+-----+-----+-----+-----+
| 1 | Третьякова | фил-б-о-20-1 | 4,4,4,5,4 |
| 2 | Журавлёв | стр-б-о-22-1 | 5,5,5,4,5 |
+-----+-----+-----+-----+
```

Рисунок 18. Выбор студентов, имеющих оценки 4, 5

## 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 -*-

"""
Вариант 2
Использовать словарь, содержащий следующие ключи: фамилия и инициалы; номер
группы; успеваемость (список из пяти элементов). Написать программу,
выполняющую
следующие действия: ввод с клавиатуры данных в список, состоящий из словарей
заданной
структуры; записи должны быть упорядочены по возрастанию среднего балла;
вывод на
дисплей фамилий и номеров групп для всех студентов, имеющих оценки 4 и 5;
если таких
студентов нет, вывести соответствующее сообщение. Необходимо реализовать
возможность хранения данных в базе данных СУБД PostgreSQL.
Информация в базе данных должна храниться не менее чем в двух таблицах.
"""

import argparse
import psycopg2
from psycopg2 import sql
import typing as t

def display_students(staff: t.List[t.Dict[str, t.Any]]) -> None:
    """
    Отобразить список студентов.
    """
    # Проверить, что список студентов не пуст.
    if staff:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 14
        )
    )
```

```

print(line)
print(
    '| {:^4} | {:^30} | {:^20} | {:^14} |'.format(
        "№",
        "Ф.И.О.",
        "Группа",
        "Оценки"
    )
)
print(line)

# Вывести данные о всех студентах.
for idx, student in enumerate(staff, 1):
    print(
        '| {:>4} | {:<30} | {:<20} | {:>14} |'.format(
            idx,
            student.get('name', ''),
            student.get('group', ''),
            student.get('grades', '')
        )
    )
print(line)

else:
    print("Список студентов пуст.")

def create_db(db_name) -> None:
    """
    Создать базу данных.
    """
    conn = psycopg2.connect(dbname='postgres', user='postgres',
                            password='1111', host='localhost')

    conn.autocommit = True

    cur = conn.cursor()

    cur.execute("SELECT 1 FROM pg_catalog.pg_database WHERE datname = %s",
                (db_name,))
    exists = cur.fetchone()

    if not exists:
        cur.execute("CREATE DATABASE {}".format(db_name))

    cur.close()
    conn.close()

```

```

conn = psycopg2.connect(dbname=db_name, user='postgres',
                        password='1111', host='localhost')

cur = conn.cursor()

# Создать таблицу с информацией о группах.
cur.execute(
    """
        CREATE TABLE IF NOT EXISTS groups (
            group_id SERIAL PRIMARY KEY,
            group_title VARCHAR(20) NOT NULL
        )
    """
)

# Создать таблицу с информацией о студентах.
cur.execute(
    """
        CREATE TABLE IF NOT EXISTS students (
            student_id SERIAL PRIMARY KEY,
            student_name VARCHAR(50) NOT NULL,
            group_id INTEGER NOT NULL,
            student_grades VARCHAR(20) NOT NULL,
            FOREIGN KEY(group_id) REFERENCES groups(group_id)
        )
    """
)

conn.commit()

cur.close()
conn.close()

def add_student(
    db_name: str,
    name: str,
    group: str,
    grades: str
) -> None:
    """
    Добавить студента в базу данных.
    """
    conn = psycopg2.connect(dbname=db_name, user='postgres',
                            password='1111', host='localhost')

    conn.autocommit = True

    cursor = conn.cursor()

```



```

# Получить идентификатор группы в базе данных.
# Если такой записи нет, то добавить информацию о новой группе.
cursor.execute(
    """
    SELECT group_id FROM groups WHERE group_title = %s
    """, (group,)
)
row = cursor.fetchone()
if row is None:
    cursor.execute(
        """
        INSERT INTO groups (group_title) VALUES (%s)
        RETURNING group_id
        """, (group,)
    )
    group_id = cursor.fetchone()[0]
else:
    group_id = row[0]

# Добавить информацию о новом студенте.
cursor.execute(
    """
    INSERT INTO students (student_name, group_id, student_grades)
    VALUES (%s, %s, %s)
    """, (name, group_id, grades)
)
conn.commit()

cursor.close()
conn.close()

def select_all(db_name) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать всех студентов.
    """
    conn = psycopg2.connect(dbname=db_name, user='postgres',
                           password='1111', host='localhost')
    cursor = conn.cursor()

    cursor.execute(
        """
        SELECT students.student_name, groups.group_title,
students.student_grades
        FROM students

```

```

        INNER JOIN groups ON groups.group_id = students.group_id
        """
    )
    rows = cursor.fetchall()

    data_with_avg = []
    for row in rows:
        grades = list(map(int, row[2].split(',')))
        average = sum(grades) / len(grades)
        data_with_avg.append((row[0], row[1], row[2], average))

    # Сортировка данных по среднему баллу
    sorted_data = sorted(data_with_avg, key=lambda x: x[3])

    cursor.close()
    conn.close()

    return [
        {
            "name": row[0],
            "group": row[1],
            "grades": row[2],
        }
        for row in sorted_data
    ]

def select_students(db_name) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать всех студентов, имеющих оценки 4 и 5.
    """
    conn = psycopg2.connect(dbname=db_name, user='postgres',
                            password='1111', host='localhost')
    cursor = conn.cursor()

    # вывод на дисплей фамилий и групп для всех студентов, имеющих оценки 4 и
5
    # Извлечение данных из столбца базы данных
    cursor.execute("""
        SELECT students.student_name, groups.group_title,
students.student_grades
        FROM students
        INNER JOIN groups ON groups.group_id = students.group_id
        """)
    rows = cursor.fetchall()

    selected_data = []

```

```

for row in rows:
    grades = list(map(int, row[2].split(',')))
    if 2 not in grades and 3 not in grades:
        average = sum(grades) / len(grades)
        selected_data.append((row[0], row[1], row[2], average))

selected_data = sorted(selected_data, key=lambda x: x[3])

cursor.close()
conn.close()

return [
    {
        "name": row[0],
        "group": row[1],
        "grades": row[2],
    }
    for row in selected_data
]

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=True,
        help="The database file name"
    )

    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("students")
    parser.add_argument(
        "--version",
        action="version",
        version="% (prog)s 0.1.0"
    )

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

    # Создать субпарсер для добавления студента.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new student"
    )

```

```
add.add_argument(
    "-n",
    "--name",
    action="store",
    required=True,
    help="The student's name"
)
add.add_argument(
    "-g",
    "--group",
    action="store",
    help="The student's group"
)
add.add_argument(
    "--grades",
    action="store",
    required=True,
    help="Grades received"
)

# Создать субпарсер для отображения всех студентов.
_ = subparsers.add_parser(
    "display",
    parents=[file_parser],
    help="Display all students"
)

# Создать субпарсер для выбора студентов.
_ = subparsers.add_parser(
    "select",
    parents=[file_parser],
    help="Select the students"
)

# Выполнить разбор аргументов командной строки.
args = parser.parse_args(command_line)

# создать базу данных.
create_db(args.db)

# Добавить студента.
if args.command == "add":
    add_student(args.db, args.name, args.group, args.grades)

# Отобразить всех студентов.
elif args.command == "display":
    display_students(select_all(args.db))
```

```

# Выбрать требуемых студентов.
elif args.command == "select":
    display_students(select_students(args.db))
    pass

if __name__ == "__main__":
    main()

```

```

(venv) D:\test_project>python main.py add --db="students" --name="Константинов М.М" --group="Фил-6-о-21-2" --grades="4,3,5,4,4"
(venv) D:\test_project>python main.py add --db="students" --name="Короткова К.Д" --group="Фил-6-о-21-2" --grades="4,4,5,5,4"
(venv) D:\test_project>python main.py add --db="students" --name="Павлов К.М" --group="СТР-6-о-21-2" --grades="3,3,4,4,3"
(venv) D:\test_project>python main.py add --db="students" --name="Краснова Е.В" --group="ХИМ-6-о-22-1" --grades="5,4,5,5,5"
(venv) D:\test_project>python main.py add --db="students" --name="Беляев М.М" --group="НГД-6-о-22-2" --grades="4,2,4,4,5"

```

Рисунок 19. Добавление данных в базу данных

	student_id [PK] integer	student_name character varying (50)	group_id integer	student_grades character varying (20)
1	1	Константинов М.М	1	4,3,5,4,4
2	2	Короткова К.Д	1	4,4,5,5,4
3	3	Павлов К.М	2	3,3,4,4,3
4	4	Краснова Е.В	3	5,4,5,5,5
5	5	Беляев М.М	4	4,2,4,4,5

Рисунок 20. Первая таблица базы данных

	group_id [PK] integer	group_title character varying (20)
1	1	Фил-6-о-21-2
2	2	СТР-6-о-21-2
3	3	ХИМ-6-о-22-1
4	4	НГД-6-о-22-2

Рисунок 21. Вторая таблица базы данных

```

(venv) D:\test_project>python main.py display --db="students"

```

№	Ф.И.О.	Группа	Оценки
1	Павлов К.М	СТР-6-о-21-2	3,3,4,4,3
2	Беляев М.М	НГД-6-о-22-2	4,2,4,4,5
3	Константинов М.М	Фил-6-о-21-2	4,3,5,4,4
4	Короткова К.Д	Фил-6-о-21-2	4,4,5,5,4
5	Краснова Е.В	ХИМ-6-о-22-1	5,4,5,5,5

Рисунок 22. Вывод информации обо всех студентах в консоль

```
(venv) D:\test_project>python main.py select --db="students"
```

№	Ф.И.О.	Группа	Оценки
1	Короткова К.Д	ФИЛ-6-о-21-2	4,4,5,5,4
2	Краснова Е.В	ХИМ-6-о-22-1	5,4,5,5,5

Рисунок 23. Выбор студентов, имеющих оценки 4, 5

### Контрольные вопросы:

1. Модуль `sqlite3` в Python предоставляет простой способ взаимодействия с базами данных SQLite из Python. Он позволяет создавать, подключаться к базам данных, выполнять запросы и манипулировать данными.

2. Соединение с базой данных SQLite3 выполняется с помощью функции `connect` модуля `sqlite3`. Курсор базы данных представляет собой объект, который используется для выполнения SQL-запросов и манипуляции данными в базе.

3. Для подключения к базе данных SQLite3, находящейся в оперативной памяти компьютера, необходимо использовать специальное имя файла `":memory:"` в качестве имени файла базы данных при вызове функции `connect`. Например: `conn = sqlite3.connect(':memory:')`.

4. Для корректного завершения работы с базой данных SQLite3 необходимо закрыть соединение с помощью метода `close`, вызванного на объекте соединения.

5. Для вставки данных в таблицу базы данных SQLite3 используется метод `execute` объекта курсора, передавая SQL-запрос `INSERT`.

6. Обновление данных таблицы базы данных SQLite3 осуществляется с помощью SQL-запроса `UPDATE`, который выполняется через метод `execute` объекта курсора.

7. Выборка данных из базы данных SQLite3 осуществляется с помощью SQL-запроса `SELECT`, который также выполняется через метод `execute` объекта курсора.

8. Метод `rowcount` используется для получения количества строк, затронутых последним выполненным запросом (например, количество строк, затронутых INSERT, UPDATE или DELETE).

9. Чтобы получить список всех таблиц базы данных SQLite3, можно выполнить запрос к системной таблице `sqlite_master`.

10. Для проверки существования таблицы при её добавлении или удалении можно использовать запрос к системной таблице `sqlite_master` или выполнить попытку выполнить операцию и обработать исключение, если таблица уже существует или отсутствует.

11. Массовая вставка данных в базу данных SQLite3 может быть выполнена с использованием метода `executemany` объекта курсора, который позволяет вставить несколько строк за один раз.

12. При работе с датой и временем в базе данных SQLite3 можно использовать специальные типы данных, такие как DATE, TIME, DATETIME или TIMESTAMP, а также функции для работы с датами и временем, такие как `date()`, `time()` и т.д.

**Вывод:** В ходе выполнения лабораторной работы было изучено взаимодействие с базами данных SQLite3 с помощью языка программирования Python.