

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ  
ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное  
учреждение высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций**

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

**Отчет по лабораторной работе № 2.21**

**по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-21-1

Пуценко И.А. \_\_\_\_\_ « » 2023г.

Подпись студента \_\_\_\_\_

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

Проверил Воронкин Р.А. \_\_\_\_\_

(подпись)

Ставрополь 2023

**Цель работы:** построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.

**Ход работы:**

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия IT и язык программирования Python.
3. Выполните клонирование созданного репозитория.
4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.
6. Создайте проект PyCharm в папке репозитория.
7. Проработать примеры лабораторной работы. Создайте для них отдельные модули языка. Зафиксируйте изменения в репозитории.

Пример 1. 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(
                "No",
                "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:
        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,
        ##### !!!!!!!!!!!!!!!!!!!!! Path.home() /
        "workers.db",
        default=str(Path.cwd() /
        "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()

```

8. Приведите в отчете скриншоты работы программ решения индивидуальных заданий.

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_ways(waypoints: t.List[t.Dict[str, t.Any]]) ->
None:
    """
        Отобразить список маршрутов.
    """
    if
waypoints:
        # Заголовок таблицы.
        line =
'+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 30,
            '-' * 15
        )
print(line)
print(
    '| {:^4} | {:^30} | {:^30} | {:^15} |'.format(
        "№",
        "Название начального маршрута",
        "Название конечного маршрута",
        "Номер маршрута"
    )
)
print(line)
# Вывести данные о всех маршрутах.
for idx, way in enumerate(waypoints, 1):
    print(
        '| {:>4} | {:<30} | {:<30} | {:>15} |'.format(
idx,
            way.get('start', ''),
way.get('finish', ''),
            way.get('num',
0)
        )
    )
print(line)
else:
    print("Список пуст")
def create_db(database_path: Path) ->
None:
    """
        Создать базу данных.
    """
    conn = sqlite3.connect(database_path)
cursor = conn.cursor()
    # Создать таблицу с информацией о маршрутах.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS routs (
routes_id INTEGER PRIMARY KEY AUTOINCREMENT,

```

```

        routes_num INTEGER NOT NULL
    )
    """
)
# Создать таблицу с информацией о начальных и конечных точках маршрутов
cursor.execute(
    """
    CREATE TABLE IF NOT EXISTS waypoints (
        waypoints_id INTEGER PRIMARY KEY
        AUTOINCREMENT,
        start_way TEXT NOT NULL,
        routes_id INTEGER NOT NULL,
        finish_way TEXT
        NOT NULL,
        FOREIGN KEY(routes_id) REFERENCES routs(routes_id)
    )
    """
)
conn.close()

def add_way(
    database_path: Path,
    start: str,
    finish:
    str,
    num: int
) -> None:
    """
    Добавить данные о маршруте.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    # Получить идентификатор маршрута в базе данных.
    # Если такой записи нет, то добавить информацию о новом маршруте.
    cursor.execute(
        """
        SELECT routes_id FROM routs WHERE routes_num = ?
        """,
        (num,)
    )
    row = cursor.fetchone()
    if row is None:
        cursor.execute(
            """
            INSERT INTO routs (routes_num) VALUES (?)
            """,
            (num,)
        )
        routes_id = cursor.lastrowid
    else:
        routes_id = row[0]
    # Добавить информацию о новых начальных и конечных точек маршрутов.
    cursor.execute(
        """
        INSERT INTO waypoints (routes_id, start_way, finish_way)
        VALUES (?, ?, ?)
        """,
        (routes_id, start, finish)
    )
    conn.commit()
    conn.close()

```

```
def select_all(database_path: Path) -> t.List[t.Dict[str,  
t.Any]]:
```



"""

*Выбрать все точки маршрутов*

"""



```

    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT waypoints.start_way, waypoints.finish_way, routs.routes_num
        FROM waypoints
        INNER JOIN routs ON routs.routes_id = waypoints.routes_id
        """
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "start": row[0],
            "finish": row[1],
            "num": row[2],
        }
        for row in rows
    ]

def find_ways(
    database_path: Path, num: int) \
    -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать маршрут с данным номером.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT waypoints.start_way, waypoints.finish_way, routs.routes_num
        FROM waypoints
        INNER JOIN routs ON routs.routes_id = waypoints.routes_id
        WHERE routs.routes_num = ?
        """,
        (num,)
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "start": row[0],
            "finish": row[1],
            "num": 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() / "waypoints.db"),
        help="The database file name"
    )

```

```
# Создать основной парсер командной строки.  
parser = argparse.ArgumentParser("waypoints")  
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 way"
    )
    add.add_argument(
        "-s", "--start",
        action="store",
        required=True,
        help="Start Route"
    )
    add.add_argument(
        "-f", "--finish",
        action="store",
        help="Final Route"
    )
    add.add_argument(
        "-n", "--
num",
        action="store",
        required=True,
        help="Route number"
    )

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

    # Создать субпарсер для поиска студентов.
    find = subparsers.add_parser(
        "find",
        parents=[file_parser],
        help="find the ways"
    )

    # 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 маршрут.
    if args.command == "add":
        add_way(db_path, args.start, args.finish, args.num)
    # Отобразить все маршруты.

```

```
elif args.command == "display":
    display_ways(select_all(db_path))
    # Выбрать требуемые маршруты.
elif args.command == "find":
    display_ways(find_ways(db_path))
pass
if __name__ ==
'__main__':
    main()
```


### 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`