МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет о лабораторной работе №2.21 по дисциплине основы программной инженерии

Выполнил:
Шальнев Владимир Сергеевич,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:
Доцент кафедры
прикладной математики и
компьютерной безопасности,
Воронкин Р.А.

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2022 г.

Выполнение:

Пример:

```
#!/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,
        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 рааботников.
    elif args.command == "select":
        display_workers(select_by_period(db_path, args.period))
        pass


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

Работа примера:

```
(venv) F:\pythonProject\lab_2_9\example>python ex1.py add --db="F:\pythonProject\lab_2_9\example\workers.db"  --name="Сидоров Сидор" --post="Главный инженер" --year=2012

(venv) F:\pythonProject\lab_2_9\example>python ex1.py display --db="F:\pythonProject\lab_2_9\example\workers.db"
+------+--------------------------------+----------------------+----------+
|  No  |           Ф.И.О.               |     Должность        |   Год    |
+------+--------------------------------+----------------------+----------+
|    1 | Сидоров Сидор                  | Главный инженер      |     2012 |
```

Индивидуальное задание:

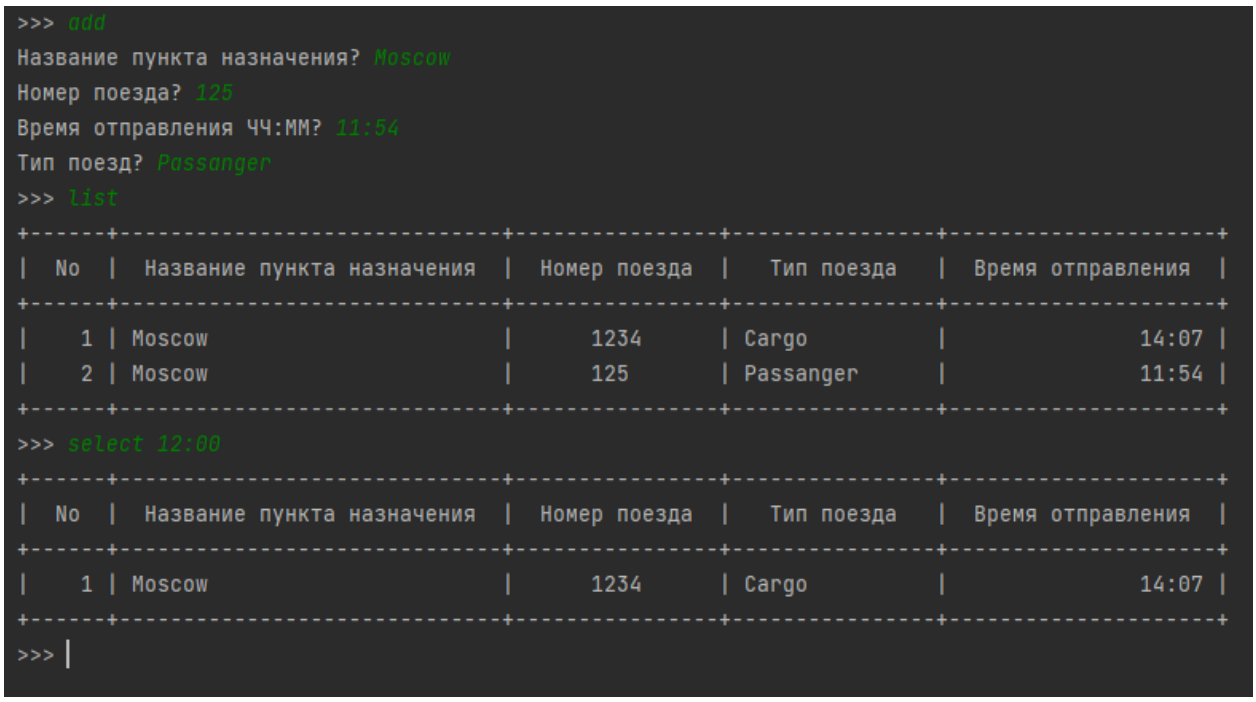
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import sys
from datetime import datetime
import sqlite3

DB_PATH = "./data.db"


def create_db():
    con = sqlite3.connect(DB_PATH)
    cur = con.cursor()

    cur.execute(
        '''
        CREATE TABLE IF NOT EXISTS types(
            type_id INTEGER PRIMARY KEY AUTOINCREMENT,
            train_type TEXT NOT NULL
        )
        '''
    )

    cur.execute(
        '''
        CREATE TABLE IF NOT EXISTS departures(
            departure_id INTEGER PRIMARY KEY AUTOINCREMENT,
            train_number INTEGER NOT NULL,
            destination TEXT NOT NULL,
            type_id INTEGER NOT NULL,
            time DATE,
            FOREIGN KEY(type_id) REFERENCES types(type_id)
        )
        '''
    )
    con.commit()
    con.close()


def select_all():
    pass
```

```
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT
            departures.train_number,
            types.train_type,
            departures.destination,
            departures.time
        FROM departures
        INNER JOIN types ON types.type_id = departures.type_id
        """
    )
    rows = cursor.fetchall()
    conn.commit()
    conn.close()

    return [
        {
            'destination': row[2],
            'number': row[0],
            'train_type': row[1],
            'time': datetime.strptime(row[3], '%H:%M'),
        }
        for row in rows
    ]


def add():
    destination = input("Название пункта назначения? ")
    number = int(input("Номер поезда? "))
    time = input("Время отправления ЧЧ:ММ? ")
    train_type = input("Тип поезд? ")
    time = datetime.strptime(time, '%H:%M')

    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()

    cursor.execute(
        """
        SELECT type_id FROM types WHERE train_type = ?
        """,
        (train_type,)
    )
    row = cursor.fetchone()

    if row is None:
        cursor.execute(
            """
            INSERT INTO types (train_type) VALUES (?)
            """,
            (train_type,)
        )
        type_id = cursor.lastrowid
    else:
        type_id = row[0]

    cursor.execute(
        """
        INSERT INTO departures (train_number, destination, type_id, time)
        VALUES (?, ?, ?, ?)
        """,
        (number, destination, type_id, time.strftime("%H:%M"))
    )
```

```
    conn.commit()
    conn.close()


def select(command):
    count = 0
    parts = command.split(' ', maxsplit=1)
    time = datetime.strptime(parts[1], '%H:%M')

    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()

    cursor.execute(
        """
        SELECT
            departures.train_number,
            types.train_type,
            departures.destination,
            departures.time
        FROM departures
        INNER JOIN types ON types.type_id = departures.type_id
        WHERE departures.time > ?
        """,
        (time.strftime("%H:%M"),)
    )
    rows = cursor.fetchall()
    conn.close()

    trains = [
        {
            'destination': row[2],
            'number': row[0],
            'train_type': row[1],
            'time': datetime.strptime(row[3], '%H:%M'),
        }
        for row in rows
    ]

    if len(trains) == 0:
        print("Отправлений позже этого времени нет.")
    else:
        print_list(trains)


def main():
    create_db()
    while True:
        command = get_command()
        if command == 'exit':
            break

        elif command == 'add':
            add()

        elif command == 'list':
            print_list(select_all())

        elif command.startswith('select'):
            select(command)

        elif command == 'help':
            print_help()
```

```
        else:
            print(f"Неизвестная команда {command}", file=sys.stderr)


def get_command():
    return input(">>> ").lower()


def print_list(trains):
    line = '+-{}-+-{}-+-{}-+-{}-+-{}-+'.format(
        '-' * 4,
        '-' * 28,
        '-' * 14,
        '-' * 14,
        '-' * 19
    )
    print(line)
    print(
        '| {:^4} | {:^28} | {:^14} | {:^14} | {:^19} |'.format(
            "No",
            "Название пункта назначения",
            "Номер поезда",
            "Тип поезда",
            "Время отправления"
        )
    )
    print(line)
    for idx, train in enumerate(trains, 1):
        print(
            '| {:>4} | {:<28} | {:^14} | {:<14} | {:>19} |'.format(
                idx,
                train.get('destination', ''),
                train.get('number', ''),
                train.get('train_type', ''),
                train.get('time', 0).strftime("%H:%M")
            )
        )
    print(line)


def print_help():
    print("Список команд:\n")
    print("add - добавить отправление;")
    print("list - вывести список отправлений;")
    print("select <ЧЧ:ММ> - вывод на экран информации о "
          "поездах, отправляющихся после этого времени;")
    print("help - отобразить справку;")
    print("exit - завершить работу с программой.")


if __name__ == '__main__':
    main()
```

Пример работы индивидуального задания:



Ответы на 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() :

import sqlite3

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

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?

con.close()

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

entities = (2, ‘Andrew’, 800, ‘IT’, ‘Tech’, ‘2018-02-06’)

cursor_obj.execute("INSERT INTO employees VALUES(1, ‘John’, 700, ‘HR’, ‘Manager’, ‘2017-01-04’)")

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 ().

```
cursor_obj.execute(
    "UPDATE employees SET name = 'Rogers' where id = 2"
)
```

7. Как осуществляется выборка данных из базы данных SQLite3?

```
cursor_obj.execute("SELECT * FROM employees")
```

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 .

```
cursor_obj.execute(
    "SELECT name from sqlite_master where type='table'"
)
```

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

При создании таблицы мы должны убедиться, что она еще не существует. Аналогично, при удалении/удалении таблицы она должна существовать. Чтобы проверить, не существует ли таблица уже, мы используем IF NOT EXISTS с оператором CREATE TABLE следующим образом:

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

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

Метод executemany можно использовать для вставки нескольких строк одновременно.

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

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 .

```
data = [
```

```
            (1, "Ridesharing", datetime.date(2017, 1, 2)),
            (2, "Water Purifying", datetime.date(2018, 3, 4))
        ]
        cursor_obj.executemany("INSERT INTO assignments VALUES(?, ?, ?)",
data)
```