МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Институт цифрового развития

ОТЧЁТ

по лабораторной работе №2.21

Дисциплина: «Основы программной инженерии»

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

Выполнила: студентка 2
курса группы Пиж-б-о-21-1
Джолдошова Мээрим
Бекболотовна

Ставрополь 2023

1. 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.

Код:

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


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

```
C:\OPI\lbrtt_2.21\PyCharm>python exmpl.py add --name="Any Tailor Joy" --post="Producer" --year=2006

C:\OPI\lbrtt_2.21\PyCharm>python exmpl.py add --name="Eva Green" --post="Actress" --year=2010

C:\OPI\lbrtt_2.21\PyCharm>python exmpl.py display
+------+--------------------------------+----------------------+----------+
|  No  |            Ф.И.О.              |      Должность       |   Год    |
+------+--------------------------------+----------------------+----------+
|    1 | Mia Goth                       | Actress              |     2008 |
+------+--------------------------------+----------------------+----------+
|    2 | Any Tailor Joy                 | Producer             |     2006 |
+------+--------------------------------+----------------------+----------+
|    3 | Eva Green                      | Actress              |     2010 |
+------+--------------------------------+----------------------+----------+

C:\OPI\lbrtt_2.21\PyCharm>
```

Рисунок 1 – Результат работы программы

```
C:\OPI\lbrtt_2.21\PyCharm>python exmpl.py select --period=17
+------+--------------------------------+----------------------+----------+
|  No  |             Ф.И.О.             |      Должность       |   Год    |
+------+--------------------------------+----------------------+----------+
|    1 | Any Tailor Joy                 | Producer             |     2006 |
+------+--------------------------------+----------------------+----------+

C:\OPI\lbrtt_2.21\PyCharm>
```

Рисунок 2 – Результат работы программы

2. 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.

Код:

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

import sys
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 рейсах.
    cursor.execute(
        """
    CREATE TABLE IF NOT EXISTS flights (
        flight_id INTEGER PRIMARY KEY AUTOINCREMENT,
        flight_destination TEXT NOT NULL,
        flight_tp TEXT NOT NULL
        )
        """
    )
    # Создать таблицу с информацией о номерах рейсов.
    cursor.execute(
        """
    CREATE TABLE IF NOT EXISTS numbers (
        flight_id INTEGER NOT NULL,
        number_flight_numer INTEGER NOT NULL,
        FOREIGN KEY(flight_id) REFERENCES flights(flight_id)
        )
        """
    )
    conn.close()
```

```python
def add_flight(
        database_path: Path,
        destination: str,
        number_flight: int,
        type_plane: str
) -> None:
    """
    Добавить рейс в базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    cursor.execute(
        """
        SELECT flight_id FROM flights WHERE flight_destination = ?
        """,
        (destination,)
    )
    row = cursor.fetchone()

    if row is None:
        cursor.execute(
            """
            INSERT INTO flights (flight_destination, flight_tp) VALUES (?, ?)
            """,
            (destination, type_plane)
        )
        flight_id = cursor.lastrowid

    else:
        flight_id = row[0]

    cursor.execute(
        """
        INSERT INTO numbers (flight_id, number_flight_numer)
        VALUES (?, ?)
        """,
        (flight_id, number_flight)
    )
    conn.commit()
    conn.close()


def display_flights(flights):
    """
    Вывод информации о рейсах.
    """
    if flights:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 18
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^18} |'.format(
                "№",
                "Destination",
                "NumberOfTheFlight",
                "TypeOfThePlane"
            )
        )
        print(line)

        for idx, flight in enumerate(flights, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>18} |'.format(
```

```
                    idx,
                    flight.get('destination', ''),
                    flight.get('number_flight', ''),
                    flight.get('type_plane', 0)
                )
            )
        print(line)
    else:
        print('list is empty')


def select_all(database_path: Path) -> t.List[t.Dict[str, t.Any]]:
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT flights.flight_destination, flights.flight_tp,
numbers.number_flight_numer
        FROM numbers
        INNER JOIN flights ON numbers.flight_id = flights.flight_id
        """
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "destination": row[0],
            "number_flight": row[1],
            "type_plane": row[2],
        }
        for row in rows
    ]


def select_flights(
        database_path: Path, tp: str
) -> t.List[t.Dict[str, t.Any]]:
    """
    Вывод на экран информации по типу рейса.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT flights.flight_destination, flights.flight_tp,
numbers.number_flight_numer
        FROM numbers
        INNER JOIN flights ON numbers.flight_id = flights.flight_id
        WHERE flights.flight_tp = ?
        """,
        (tp,)
    )
    rows = cursor.fetchall()
    conn.close()
    if len(rows) == 0:
        return []

    return [
        {
            "destination": row[0],
            "type_plane": row[1],
            "number_flight": 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",
        required=False,
        default=str(Path.cwd() / "data_ph.db"),
        help="The database file name"
    )

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

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

    # Создать субпарсер для добавления рейса.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new flight"
    )
    add.add_argument(
        "-d",
        "--destination",
        action="store",
        required=True,
        help="Destination of the flight"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        type=int,
        required=True,
        help="Number of the flight"
    )
    add.add_argument(
        "-t",
        "--type",
        action="store",
        required=True,
        help="Type of the plane"
    )

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

    # Создать субпарсер для выбора рейсов.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select flights by type"
    )
    select.add_argument(
        "-t",
        "--type",
        action="store",
        required=True,
        help="The required flight type"
    )
```

```
    # 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)

    if args.command == "add":
        add_flight(db_path, args.destination, args.number, args.type)
    elif args.command == "display":
        display_flights(select_all(db_path))
    elif args.command == "select":
        display_flights(select_flights(db_path, args.type))


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

```
C:\OPI\lbrtt_2.21\indiv>python inddvv.py add -d="China" -n=5 -t="passangers"

C:\OPI\lbrtt_2.21\indiv>python inddvv.py add -d="Kanada" -n=3 -t="passangers"

C:\OPI\lbrtt_2.21\indiv>python inddvv.py add -d="Bishkek" -n=2 -t="official"

C:\OPI\lbrtt_2.21\indiv>python inddvv.py add -d="Bangkok" -n=2 -t="passangers"

C:\OPI\lbrtt_2.21\indiv>python inddvv.py display
+------+-------------------------------+----------------------+--------------------+
|  №   |          Destination          |  NumberOfTheFlight   |   TypeOfThePlane   |
+------+-------------------------------+----------------------+--------------------+
|    1 | China                         | passangers           |                  5 |
|    2 | Kanada                        | passangers           |                  3 |
|    3 | Bishkek                       | official             |                  2 |
|    4 | Bangkok                       | passangers           |                  2 |
+------+-------------------------------+----------------------+--------------------+
```

Рисунок 3 – Результат работы команды add и display

```
C:\OPI\lbrtt_2.21\indiv>python inddvv.py select -t=passangers
+------+-------------------------------+----------------------+--------------------+
|  №   |          Destination          |  NumberOfTheFlight   |   TypeOfThePlane   |
+------+-------------------------------+----------------------+--------------------+
|    1 | China                         | 5                    |         passangers |
|    2 | Kanada                        | 3                    |         passangers |
|    3 | Bangkok                       | 2                    |         passangers |
+------+-------------------------------+----------------------+--------------------+

C:\OPI\lbrtt_2.21\indiv>python inddvv.py select -t=official
+------+-------------------------------+----------------------+--------------------+
|  №   |          Destination          |  NumberOfTheFlight   |   TypeOfThePlane   |
+------+-------------------------------+----------------------+--------------------+
|    1 | Bishkek                       | 2                    |           official |
+------+-------------------------------+----------------------+--------------------+

C:\OPI\lbrtt_2.21\indiv>
```

Рисунок 4 – Результат работы команды select

Вопросы для защиты работы:

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 базамиданных SQLite3?

В базе данных Python SQLite3 мы можем легко хранить дату или время,импортируя модуль datetime