МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе №2.21

Взаимодействие с базами данных SQLite3 с помощью языка программирования Python.

по дисциплине «Технологии программирования и алгоритмизации»

Выполнил студент группы ИВТ-б-о-21-1

Лысенко И.А. « »_____20__г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____

(подпись)

Ставрополь 2023

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

Ход работы:

1. Создал репозиторий на GitHub: https://github.com/IsSveshuD/lab_2_21.git .

2. Проработал пример:

```python
#!/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"
            )
        )
```

```python
        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 индентификатор должности в базе данных.
    # Если такой 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()

```

```python
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
    ]
```

```python
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 data 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"
    )
```

```python
    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()
```

Рисунок 1 – Пример

3. Результат работы примера.

```
PS C:\Users\user\Documents\2 курс\4-й семестр\анализ данных\2.21> python "Primer 1.py" display --db 1.db
+------+--------------------------------+----------------------+----------+
|  No  |            Ф.И.О.              |      Должность       |   Год    |
+------+--------------------------------+----------------------+----------+
|    1 | Ivan                           | Student              |     2001 |
+------+--------------------------------+----------------------+----------+
|    2 | Ivan                           | Student              |     2001 |
+------+--------------------------------+----------------------+----------+
```

Рисунок 2 – Работа примера

4. Выполнил индивидуальное задание 1.

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


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


def display_route(routes: t.List[t.Dict[str, t.Any]]) -> None:
    """
    Отобразить списко маршрутов
    """
    if routes:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 10
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^10} |'.format(
                "№",
                "Начальный пункт",
                "Конечный пункт",
                "№ маршрута"
            )
        )
        print(line)
```

```
        for idx, worker in enumerate(routes, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>10} |'.format(
                    idx,
                    worker.get('start', ''),
                    worker.get('finish', ''),
                    worker.get('number', 0)
                )
            )
        print(line)
    else:
        print("Список маршрутов 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 numbers (
            number_id INTEGER PRIMARY KEY AUTOINCREMENT,
            number_title TEXT NOT NULL
        )
        """
    )
```

```
    # Создать таблицу с информацией о маршрутах.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS routes (
        routes_id INTEGER PRIMARY KEY AUTOINCREMENT,
        start TEXT NOT NULL,
        finish INTEGER NOT NULL,
        number_id INTEGER NOT NULL,
        FOREIGN KEY(number_id) REFERENCES numbers(number_id)
        )
        """
    )
    conn.close()


def add_route(
        database_path: Path,
        start: str,
        finish: str,
        number: int
) -> None:
    """
    Добавить маршрут в базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    # Получить индентификатор маршрута в базе данных.
    # Если такого нет, то добавить информацию о новом маршруте.
```

```
    cursor.execute(
        """
        SELECT number_id FROM numbers WHERE number_title = ?
        """,
        (number,)
    )
    row = cursor.fetchone()
    if row is None:
        cursor.execute(
            """
            INSERT INTO numbers (number_title) VALUES (?)
            """,
            (number,)
        )
        number_id = cursor.lastrowid

    else:
        number_id = row[0]

    # Добавить информацию о новом работнике.
    cursor.execute(
        """
        INSERT INTO routes (start, finish, number_id)
        VALUES (?, ?, ?)
        """,
        (start, finish, number_id)
    )

    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 routes.start, routes.finish, numbers.number_title
        FROM routes
        INNER JOIN numbers ON numbers.number_id = routes.number_id
        """
    )
    rows = cursor.fetchall()

    conn.close()
    return [
        {
            "start": row[0],
            "finish": row[1],
            "number": row[2],
        }
        for row in rows
    ]


def select_by_period(
    database_path: Path, period: int
) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать определенный маршрут.
    """
```

```
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    cursor.execute(
        """
        SELECT routes.start, routes.finish, numbers.number_title
        FROM routes
        INNER JOIN numbers ON numbers.number_id = routes.number_id
        WHERE numbers.number_title == ?
        """,
        (period,)
    )
    rows = cursor.fetchall()

    conn.close()
    return [
        {
            "start": row[0],
            "finish": row[1],
            "number": 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 data file name"
    )
    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("routes")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )
    subparsers = parser.add_subparsers(dest="command")
    # Создать субпарсер для добавления маршрута.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new route"
    )
    add.add_argument(
        "-s",
        "--start",
        action="store",
        required=True,
        help="The start of the route"
    )
```

```
    add.add_argument(
        "-f",
        "--finish",
        action="store",
        help="The finish of the route"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        type=int,
        required=True,
        help="The number of the route"
    )
    # Создать субпарсер для отображения всех маршрутов.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all routes"
    )
    # Создать субпарсер для выбора маршрута.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the route"
    )
    select.add_argument(
        "-N",
        "--period",
        action="store",
        type=int,
        required=True,
        help="The route"
    )
    # 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_route(db_path, args.start, args.finish, args.number)

    # Отобразить все маршруты.
    elif args.command == "display":
        display_route(select_all(db_path))

    # Выбрать требуемые маршруты.
    elif args.command == "select":
        display_route(select_by_period(db_path, args.period))
        pass


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

Рисунок 3 – Код задания 1.

5. Чтобы импортировать БД без использования опции --csv, необходимо сначала ввести команды .mode csv, и только потом импортировать БД.

```
PS C:\Users\user\Documents\2 курс\4-й семестр\анализ данных\2.21> python  "ind1.py" display --db 2.db
+------+--------------------------------+----------------------+------------+
|  №   |        Начальный пункт         |    Конечный пункт    | № маршрута |
+------+--------------------------------+----------------------+------------+
|    1 | Астрахань                      | Ростов               |          1 |
+------+--------------------------------+----------------------+------------+
```

Рисунок 4 – Работа задания один

6. Выполнил индивидуальное задание 2. Изучил пакет psycopg2. Добавил подключение к базе данных.

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


import argparse
import sqlite3
import typing as t
from pathlib import Path
import psycopg2


def connect():
    conn = psycopg2.connect(
        user="postgres",
        password="123321",
        host="127.0.0.1",
        port="5432",
        database="postgres"
    )

    return conn
```

Рисунок 5 – Задача 5. Количество городов по часовым поясам.

7. Результат работы индивидуального задания 2.

```
PS C:\Users\user\Documents\2 курс\4-й семестр\анализ данных\2.21> python  "ind2.py" display
+------+--------------------------------+----------------------+------------+
|  №   |        Начальный пункт         |    Конечный пункт    | № маршрута |
+------+--------------------------------+----------------------+------------+
|    1 | Ivan                           | Student              |       2001 |
|    2 | Москва                         | Питер                |         42 |
|    3 | Ставрополь                     | Пермь                |         32 |
|    4 | Астрахань                      | Ростов               |          1 |
|    5 | Астрахань                      | Ростов               |          1 |
|    6 | Астрахань                      | Ростов               |          1 |
+------+--------------------------------+----------------------+------------+
```

Рисунок 6 – Работа задания 2

Ответы на вопросы:

1. Каково назначение модуля sqlite3?

Модуль sqlite3 предназначен для взаимодействия с СУБД SQLite.

2. Как выполняется соединение с базой данных SQLite3? Что такое курсор базы данных?

Объект соединения создается с помощью функции connect(). 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?

С помощью функции: memory:

4. Как корректно завершить работу с базой данных SQLite3?

Сначала импортируется модуль sqlite3, а затем определяется функция с именем sql_connection . Внутри функции у нас есть блок try, где функция connect() возвращает объект соединения после установления соединения. В случае возникновения ошибок при установке соединения с базой данных выполняются операторы блока except , в котором в данном случае просто печатается содержимое объекта ошибки. После этого вне зависимости от того возникло или нет исключение по работе с базой данных, выполняются операторы блока finally, в котором соединение закрывается. Закрытие соединения необязательно, но это хорошая практика программирования, поэтому вы освобождаете память от любых неиспользуемых ресурсов.

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

Чтобы вставить данные в таблицу, используется оператор INSERTINTO.

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 .

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: были исследовано взаимодействие с базами данных SQLite3 с помощью языка программирования Python.