Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

«Лабораторная работа 2.21 Взаимодействие с базами данных SQLite3 с помощью языка программирования Python»

ОТЧЕТ
по лабораторной работе №24
дисциплины
«Основы программной инженерии»

Выполнил:
Луценко Дмитрий Андреевич
2 курс, группа ПИЖ-б-о-21-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и сопровождение программного обеспечения», очная форма обучения

_____
(подпись)

Проверил:
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Лабораторная работа 2.2 Взаимодействие с базами данных SQLite3 с помощью языка программирования Python

Ход работы:

```
#!/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.
    """
    if staff:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "No",
                "Ф.И.О.",
                "Должность",
                "Год"
            )
        )
        print(line)

        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()
```

```
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS posts (
        post_id INTEGER PRIMARY KEY AUTOINCREMENT,
        post_title TEXT NOT NULL
        )
        """
    )

    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()

    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]

    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
    ]


def main(command_line=None):
    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"
    )

    parser = argparse.ArgumentParser("workers")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )

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

    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"
    )

    _ = 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"
    )

    args = parser.parse_args(command_line)
```

```
    db_path = Path(args.db)
    create_db(db_path)

    if args.command == "add":
        add_worker(db_path, args.name, args.post, args.year)

    elif args.command == "display":
        display_workers(select_all(db_path))

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


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

Листинг 1 – Пример 1

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

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


def product_list(products: t.List[t.Dict[str, t.Any]]) -> None:
    """
    Вывод списка товаров
    """
    if products:
        line = '+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20
        )
        print(line)
        print(
            '| {:^25} | {:^15} | {:^14} |'.format(
                "Товар",
                "Магазин",
                "Стоимость"
            )
        )
        print(line)

        for product in products:
            print(
                '| {:^25} | {:^15} | {:^14} |'.format(
                    product.get('prod', ''),
                    product.get('shop', ''),
                    product.get('cost', 0)
                )
            )
            print(line)

    else:
        print("Список продуктов пуст.")


def create_db(database_path: Path) -> None:
```

```
    """
    Создать базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS shops(
            shop_id INTEGER PRIMARY KEY AUTOINCREMENT,
            shop_title TEXT NOT NULL
        )
        """
    )

    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS prods(
            prod_id INTEGER PRIMARY KEY AUTOINCREMENT,
            prod_name TEXT NOT NULL,
            shop_id INTEGER NOT NULL,
            prod_cost REAL NOT NULL,
            FOREIGN KEY(shop_id) REFERENCES shops(shop_id)
        )
        """
    )
    conn.close()


def add_product(
        database_path: Path,
        name: str,
        shop: str,
        cost: float
) -> None:
    """
    Ввод информации о товарах.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    cursor.execute(
        """
        SELECT shop_id FROM shops WHERE shop_title = ?
        """,
        (shop,)
    )
    row = cursor.fetchone()
    if row is None:
        cursor.execute(
            """
            INSERT INTO shops (shop_title) VALUES (?)
            """,
            (shop,)
        )
        shop_id = cursor.lastrowid
    else:
        shop_id = row[0]

    cursor.execute(
        """
        INSERT INTO prods (prod_name, shop_id, prod_cost)
        VALUES(?, ?, ?)
        """,
```

```
        (name, shop_id, cost)
    )
    conn.commit()
    conn.close()


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

    cursor.execute(
        """
        SELECT prods.prod_name, shops.shop_title, prods.prod_cost
        FROM prods
        INNER JOIN shops ON shops.shop_id = prods.shop_id
        """
    )
    rows = cursor.fetchall()

    conn.close()
    return [
        {
            "prod": row[0],
            "shop": row[1],
            "cost": row[2],
        }
        for row in rows
    ]


def select_by_shop(
        database_path: Path, shop: str
) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать товары из конкретного магазина.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    cursor.execute(
        """
        SELECT prods.prod_name, shops.shop_title, prods.prod_cost
        FROM prods
        INNER JOIN shops ON shops.shop_id = prods.shop_id
        WHERE shops.shop_title == ?
        """,
        (shop,)
    )
    rows = cursor.fetchall()

    conn.close()
    return [
        {
            "prod": row[0],
            "shop": row[1],
            "cost": row[2]
        }
        for row in rows
    ]
```

```python
def main(command_line=None):
    """
    Главная функция программы.
    """
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "--db",
        action="store",
        required=False,
        default=str(Path.home() / "products.db"),
        help="Имя файла базы данных"
    )

    parser = argparse.ArgumentParser("products")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.1"
    )
    subparsers = parser.add_subparsers(dest="command")

    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Добавить новый продукт"
    )
    add.add_argument(
        "-n",
        "--name",
        action="store",
        required=True,
        help="Название продукта"
    )
    add.add_argument(
        "-s",
        "--shop",
        action="store",
        help="Название магазина"
    )
    add.add_argument(
        "-c",
        "--cost",
        action="store",
        type=float,
        required=True,
        help="Стоимость товара"
    )

    list = subparsers.add_parser(
        "list",
        parents=[file_parser],
        help="Отобразить список товаров"
    )

    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Выбрать товары из магазина"
    )
    select.add_argument(
        "-s",
        "--shop",
        action="store",
        required=True,
```

```
        help="Название магазина"
    )

    args = parser.parse_args(command_line)

    db_path = Path(args.db)
    create_db(db_path)

    if args.command == "add":
        add_product(db_path, args.name, args.shop, args.cost)

    elif args.command == "list":
        product_list(select_all(db_path))

    elif args.command == "select":
        product_list(select_by_shop(db_path, args.shop))


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

Листинг 2 – Индивидуальное задание

Ответы на 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. Модуль sqlite3 содержит много классов, функций и констант.

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? Мы также можем 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? Закрыть соединение методом close().

5. Как осуществляется вставка данных в таблицу базы данных SQLite3? Для этого используется оператор INSERT INTO

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. Когда мы используем rowcount с оператором SELECT , будет возвращено значение -1, поскольку количество выбранных строк неизвестно до тех пор, пока все они не будут извлечены.

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 следующим образом:

CREATE TABLE IF NOT EXISTS table_name (column1, column2, ..., columnN)

DROP TABLE IF EXISTS table_name

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 .

```
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()