

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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2.21
дисциплины «Анализ данных»
Вариант 13

Выполнил:
Иващенко Олег Андреевич
2 курс, группа ИВТ-б-о-22-1,
09.03.02 «Информационные и
вычислительные машины»,
направленность (профиль)
«Программное обеспечение
средств вычислительной техники
и автоматизированных систем»

(подпись)

Руководитель практики:
Воронкин Роман Александрович,
доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

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

Цель: Изучить взаимодействие с базами данных SQLite3 при помощи ЯП Python.

Порядок выполнения работы

Пример 1. Для примера 1 лабораторной работы 2.17 реализовать возможность хранения данных в базе данных SQLite3.

Листинг 1 – Код примера

```
#!/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(
                "№",
                "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_id,) = cursor.fetchone()
    if post_id 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 работников.
    elif args.command == "select":
        display_workers(select_by_period(db_path, args.period))
    pass

if __name__ == "__main__":
    main()

```

```

PS C:\Users\UnnamedUser\Documents\СКФУ\Python\Analysis_2.21\exec> python example.py add --db="Работники" -n="Петров Петр" -p="Директор" -y=2000
PS C:\Users\UnnamedUser\Documents\СКФУ\Python\Analysis_2.21\exec> python example.py display --db="Работники"

```

%	Ф.И.О.	Должность	Год
1	Петров Петр	Директор	2000

Рисунок 1.1 – Добавление и вывод новой записи в базу данных

```
PS C:\Users\UnnamedUser\Documents\КФУ\Python\Analysis_2.21\exec> python example.py display --db="Работники"
+-----+-----+-----+-----+
| % | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Петров Петр | Директор | 2000 |
+-----+-----+-----+-----+
| 2 | Иванов Иван | Зам. директора | 2002 |
+-----+-----+-----+-----+
| 3 | Сидоров Сидор | Менеджер | 2005 |
+-----+-----+-----+-----+
PS C:\Users\UnnamedUser\Documents\КФУ\Python\Analysis_2.21\exec> python example.py select --db="Работники" -P=20
+-----+-----+-----+-----+
| % | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Петров Петр | Директор | 2000 |
+-----+-----+-----+-----+
| 2 | Иванов Иван | Зам. директора | 2002 |
+-----+-----+-----+-----+
```

Рисунок 1.2 – Выборка записей из базы данных

Индивидуальное 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.

Листинг 2 – Код программы

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

import json
from datetime import datetime
import argparse
import os.path
from pathlib import Path
import sqlite3
import typing as t

def print_help():
    """
    Функция вывода доступных пользователю команд
    """

    print("list - вывод всех добавленных записей")
    print("add - добавление новых записей")
    print("find - найти запись по фамилии")
    print("exit - завершение работы программы")

def add_worker(database_path: Path, name: str, phone: str, year: int) -> None:
    """
    Функция добавления новой записи, возвращает запись
    """

    connection = sqlite3.connect(database_path)
    cursor = connection.cursor()

    cursor.execute(
        """
        INSERT INTO workers (worker_name, phone_number, worker_year)
        VALUES (?, ?, ?)
        """,
        (name, phone, year)
    )
```

```

)

connection.commit()
connection.close()

def print_list(staff: t.List[t.Dict[str, t.Any]]) -> None:
    """
    Функция выводит на экран список всех существующих записей
    """
    if staff:
        line = '+-{}--{}--{}--{}--+'.format(
            '_' * 4,
            '_' * 30,
            '_' * 20,
            '_' * 8
        )
        print(line)

        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Ф.И.О.",
                "Номер телефона",
                "Год"
            )
        )
        print(line)

        for idx, worker in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('name', ''),
                    worker.get('phone_number', ''),
                    worker.get('year', 0)
                )
            )
            print(line)

    else:
        print("Список работников пуст.")

def create_db(database_path: Path) -> None:
    """
    Создание базы данных
    """
    connection = sqlite3.connect(database_path)
    cursor = connection.cursor()

    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS workers (
            worker_id INTEGER PRIMARY KEY AUTOINCREMENT,
            worker_name TEXT NOT NULL,
            phone_number TEXT NOT NULL,
            worker_year INTEGER NOT NULL
        )
        """
    )

    connection.close()

```



```

def select_all(database_path: Path) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбор всех записей из базы данных
    """
    connection = sqlite3.connect(database_path)
    cursor = connection.cursor()

    cursor.execute(
        """
        SELECT workers.worker_name, workers.phone_number, workers.worker_year
        FROM workers
        """
    )
    rows = cursor.fetchall()

    connection.close()
    return [
        {
            "name": row[0],
            "phone_number": 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]]:
    """
    Выборка по периоду
    """
    connection = sqlite3.connect(database_path)
    cursor = connection.cursor()

    cursor.execute(
        """
        SELECT workers.worker_name, workers.worker_phone_number, workers.worker_year
        FROM workers
        WHERE (strftime('%Y', date('now')) - workers.worker_year) >= ?
        """
        ,
        (period,)
    )
    rows = cursor.fetchall()

    connection.close()
    return [
        {
            "name": row[0],
            "phone_number": 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="Название файла базы даанных"
    )

    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="Имя работника"
    )
    add.add_argument(
        "-p",
        "--phone",
        action="store",
        help="Номер телефона работника"
    )
    add.add_argument(
        "-d",
        "--date",
        action="store",
        required=True,
        help="Дата нанятия"
    )

    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Вывести на экран всех работников"
    )

    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Выборка работников"
    )
    select.add_argument(
        "-p",
        "--period",
        action="store",
        type=int,
        required=True,
        help="Требуемый период"
    )

    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.phone, args.date)

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

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

if __name__ == "__main__":
    """
    Основная программа
    """
    main()

```

```

PS C:\Users\UnnamedUser\Documents\CKOY\Python\Analysis_2.21\exec> python individual.py add -n="Петров Петр" -p="78889993264" -y="2000" --db="Workers"
PS C:\Users\UnnamedUser\Documents\CKOY\Python\Analysis_2.21\exec> python individual.py display --db="Workers"

```

%	Ф.И.О.	Номер телефона	Год
1	Петров Петр	78889993264	2000

Рисунок 2.1 – Создание базы данных, добавление первой записи и вывод всех записей на экран

```

PS C:\Users\UnnamedUser\Documents\CKOY\Python\Analysis_2.21\exec> python individual.py display --db="Workers"

```

%	Ф.И.О.	Номер телефона	Год
1	Петров Петр	78889993264	2000
2	Иванов Иван	78886665533	2005
3	Сидоров Сидор	79997463521	2020

```

PS C:\Users\UnnamedUser\Documents\CKOY\Python\Analysis_2.21\exec> python individual.py select --db="Workers" -p=20

```

%	Ф.И.О.	Номер телефона	Год
1	Петров Петр	78889993264	2000

```

PS C:\Users\UnnamedUser\Documents\CKOY\Python\Analysis_2.21\exec> python individual.py select --db="Workers" -p=15

```

%	Ф.И.О.	Номер телефона	Год
1	Петров Петр	78889993264	2000
2	Иванов Иван	78886665533	2005

Рисунок 2.2 – Выборка записей из базы данных

Листинг 3 – Код программы dif.py

```

from datetime import datetime
import argparse
import pycopg2
from pycopg2 import sql

def print_help():
    """

```

```

Функция вывода доступных пользователю команд
"""

print("list - вывод всех добавленных записей")
print("add - добавление новых записей")
print("find - найти запись по фамилии")
print("exit - завершение работы программы")

def add_worker(cursor, surname, name, phone, date):
    """
    Функция добавления новой записи
    """
    cursor.execute(
        sql.SQL("INSERT INTO workers (surname, name, phone, date) "
                "VALUES (%s, %s, %s, %s)",
                (surname, name, phone, date)
        )
    )
    print("Запись успешно добавлена")

def print_list(cursor):
    """
    Функция выводит на экран список всех существующих записей
    """
    cursor.execute("SELECT surname, name, phone, date FROM workers")
    for row in cursor.fetchall():
        print(f"{row[0]} {row[1]} | {row[2]} | {row[3]}")

def find_member(cursor, period):
    """
    Функция для вывода на экран всех записей, чьи фамилии совпадают
    с введённой и чей год поступления на работу не ранее указанного
    """
    cursor.execute(
        sql.SQL("SELECT surname, name, phone, date FROM workers "
                "WHERE extract(year from date) <= %s",
                (datetime.now().year - period,)
        )
    )
    return cursor.fetchall()

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

    parser.add_argument(
        "--host",
        action="store",
        default="localhost",
        help="PostgreSQL server host"
    )

    parser.add_argument(
        "--port",
        action="store",
        default="5432",
        help="PostgreSQL server port"
    )

    parser.add_argument(

```

```

        "--database",
        action="store",
        default="workersDB",
        required=False,
        help="PostgreSQL database name"
    )
    parser.add_argument(
        "--user",
        action="store",
        required=False,
        help="PostgreSQL user"
    )
    parser.add_argument(
        "--password",
        action="store",
        required=False,
        help="PostgreSQL password"
    )

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

    add = subparsers.add_parser("add", help="Add a new worker")
    add.add_argument(
        "-s",
        "--surname",
        action="store",
        required=True,
        help="The worker's surname"
    )
    add.add_argument(
        "-n",
        "--name",
        action="store",
        required=True,
        help="The worker's name"
    )
    add.add_argument(
        "-p",
        "--phone",
        action="store",
        help="The worker's phone"
    )
    add.add_argument(
        "-d",
        "--date",
        action="store",
        required=True,
        help="The date of hiring"
    )

    _ = subparsers.add_parser("display", help="Display all workers")

    select = subparsers.add_parser("select", help="Select the workers")
    select.add_argument(
        "-p",
        "--period",
        action="store",
        type=int,
        required=True,
        help="The required period"
    )

args = parser.parse_args()

```

```

conn = psycopg2.connect(
    host=args.host,
    port=args.port,
    database=args.database,
    user="postgres",
    password="admin"
)

cursor = conn.cursor()

cursor.execute("CREATE TABLE IF NOT EXISTS workers "
              "(surname VARCHAR, name VARCHAR, phone VARCHAR, date DATE)")

if args.command == "add":
    add_worker(cursor, args.surname, args.name, args.phone, args.date)
    conn.commit()

elif args.command == "display":
    print_list(cursor)

elif args.command == "select":
    selected = find_member(cursor, args.period)
    for row in selected:
        print(f"{row[0]} {row[1]} | {row[2]} | {row[2]}")

cursor.close()
conn.close()

if __name__ == "__main__":
    main()

```

```

Anaconda Powershell Prompt
(base) PS C:\Users\UnnamedUser> conda activate DataAnalysis
(DataAnalysis) PS C:\Users\UnnamedUser> conda install Psycopg2
Retrieving notices: ..working.. done
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
current version: 23.7.4
latest version: 24.4.0

Please update conda by running

$ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use

conda install conda=24.4.0

## Package Plan ##

environment location: C:\Users\UnnamedUser\anaconda3\envs\DataAnalysis
added / updated specs:
- psycopg2

The following packages will be downloaded:

package | build | size
-----|-----|-----
libpq-12.17 | h986ac69_0 | 3.2 MB
openssl-3.0.13 | h2bbff1b_1 | 7.5 MB
psycopg2-2.9.9 | py312h2bbff1b_0 | 161 KB
Total: 10.8 MB

The following NEW packages will be INSTALLED:

```

Рисунок 3.1 - Установка пакета PostgreSQL

```

PS C:\Users\UnnamedUser\Documents\КФУ\Python\Analysis_2.21\exec> python dif.py add -s "Иванов" -n "Иван" -р "7999666513
2" -d "26.06.2010"
Запись успешно добавлена

```

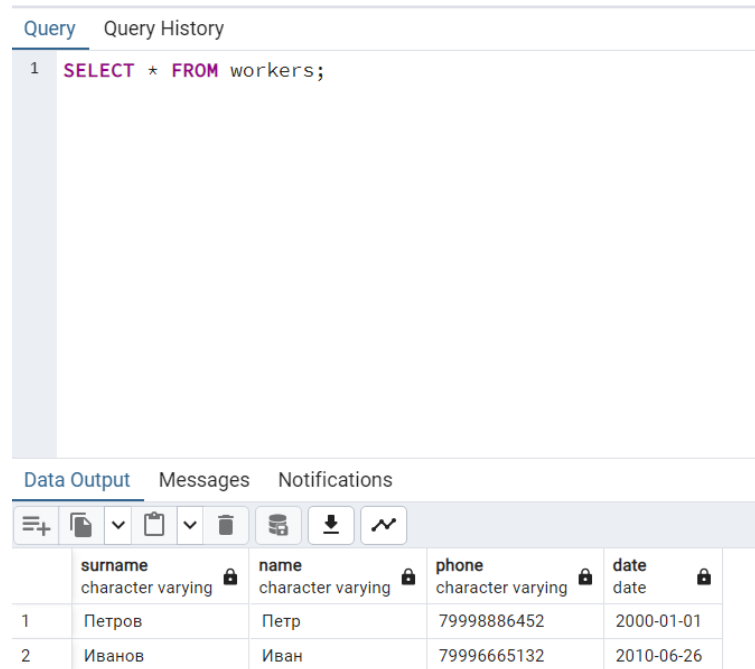
Рисунок 3.2 – Добавление новой записи в таблицу

```

PS C:\Users\UnnamedUser\Documents\СКОУ\Python\Analysis_2.21\exec> python dif.py display
Петров Петр | 79998886452 | 2000-01-01
Иванов Иван | 79996665132 | 2010-06-26
PS C:\Users\UnnamedUser\Documents\СКОУ\Python\Analysis_2.21\exec> python dif.py select -p=20
Петров Петр | 79998886452 | 79998886452

```

Рисунок 3.3 – Вывод списка сотрудников и выборка



The screenshot shows a database query interface. At the top, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query: `1 SELECT * FROM workers;`. Below the query, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table of results. The table has four columns: 'surname', 'name', 'phone', and 'date'. The first two columns are labeled 'character varying' and the last two are labeled 'date'. The table contains two rows of data.

	surname character varying	name character varying	phone character varying	date date
1	Петров	Петр	79998886452	2000-01-01
2	Иванов	Иван	79996665132	2010-06-26

Рисунок 3.4 – Данные в таблице PostgreSQL

Контрольные вопросы

1. Каково назначение модуля sqlite3?

Модуль SQLite3 предоставляет интерфейс для взаимодействия с базой данных SQLite3 из программ на Python.

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

Соединение с базой данных SQLite3 выполняется с помощью функции `connect()` из модуля `sqlite3`. Курсор базы данных – объект, который используется для выполнения SQL-запросов и работы с результатами этих запросов.

3. Как подключиться к базе данных SQLite3, находящейся в оперативной памяти компьютера?

Для подключения к базе данных SQLite3, находящейся в оперативной памяти устройства, нужно использовать специальное имя файла ":memory:" при вызове функции connect().

4. Как корректно завершить работу с базой данных SQLite3?

Для корректного завершения работы с базой данных SQLite3 необходимо закрыть соединение с помощью метода close().

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

Вставка данных в таблицу базы данных SQLite3 осуществляется с использованием оператора SQL INSERT.

6. Как осуществляется обновление данных таблицы базы данных SQLite3?

Обновление данных таблицы базы данных SQLite3 выполняется с помощью оператора SQL UPDATE.

7. Как осуществляется выборка данных из базы данных SQLite3?

Выборка данных из базы данных SQLite3 осуществляется с использованием оператора SQL SELECT.

8. Каково назначение метода rowcount?

Метод rowcount возвращает количество строк, затронутых последним выполненным запросом.

9. Как получить список всех таблиц базы данных SQLite3?

Для получения списка всех таблиц базы данных SQLite3 можно выполнить запрос к системной таблице `sqlite_master`.

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

Для выполнения проверки существования таблицы как при её добавлении, так и при её удалении, можно использовать операторы `CREATE TABLE IF NOT EXISTS` и `DROP TABLE IF EXISTS` соответственно.

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

Метод `executemany()` позволяет вставить сразу множество данных в базу данных SQLite3.

12. Как осуществляется работа с датой и временем при работе с базами данных SQLite3

Работа с датой и временем в базе данных SQLite3 осуществляется с использованием специальных типов данных, таких как `DATE`, `TIME` и `DATETIME`, а также функций и операторов для работы с ними в SQL-запросах.

Выводы: В процессе выполнения лабораторной работы были изучены методы взаимодействия баз данных SQLite3 с помощью языка программирования Python. Были проработаны примеры лабораторной работы, выполнено индивидуальное задание и задание повышенной сложности.