

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

Основы кроссплатформенного программирования

Отчет по лабораторной работе №2.21

Взаимодействие с базами данных SQLite3 с помощью языка
программирования Python.

Выполнил студент группы

ИВТ-б-о-21-1

Мальцев Н.А. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил доцент

Кафедры инфокоммуникаций, старший
преподаватель

Воронкин Р.А.

(подпись)

Ставрополь 2023

Цель работы: изучение возможностей взаимодействия с базами данных SQLite3 с помощью языка программирования Python версии 3.10.

Порядок выполнения работы:

Пример 1. Создание базы данных.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sqlite3
from sqlite3 import Error
def sql_connection():
    try:
        con = sqlite3.connect(':memory:')
        print("Connection is established: Database is created in memory")
    except Error:
        print(Error)
    finally:
        con.close()
if __name__ == "__main__":
    sql_connection()
```

Пример 2. Создание таблиц.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sqlite3
from sqlite3 import Error
def sql_connection():
    try:
        con = sqlite3.connect(':memory:')
        print("Connection is established: Database is created in memory")
    except Error:
        print(Error)
    finally:
        con.close()
if __name__ == "__main__":
    sql_connection()
```

Результат работы программы – созданная в базе данных таблица:

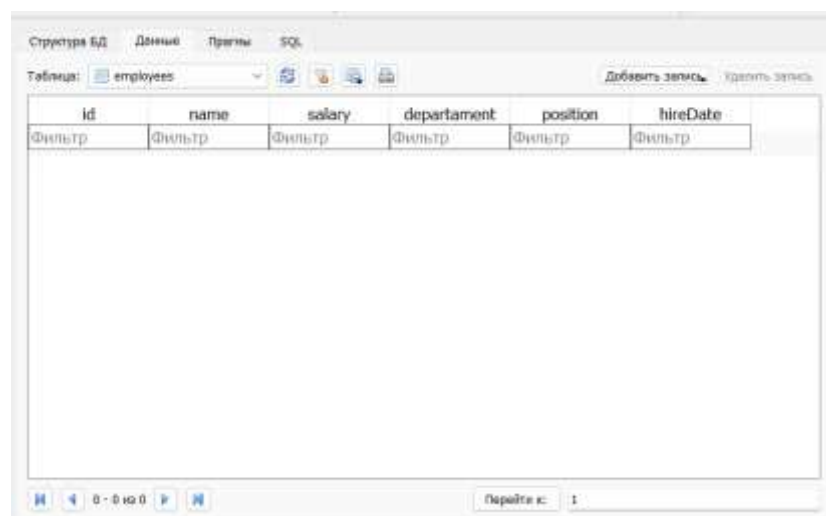


Рисунок 1. Созданная страница в DB Browser for SQLite

Пример 3. Вставка данных в таблицу.

Код программы:

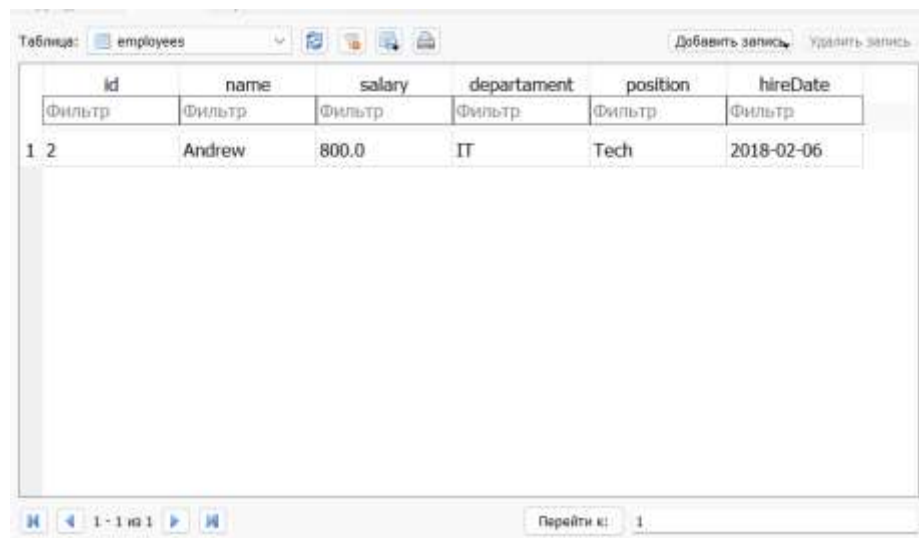
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sqlite3

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

def sql_insert(con, entities):
    cursor_obj = con.cursor()
    cursor_obj.execute(
        """
        INSERT INTO employees(id, name, salary, departament, position,
hireDate)
        VALUES(?, ?, ?, ?, ?, ?)
        """, entities
    )
    con.commit()

entities = (2, 'Andrew', 800, 'IT', 'Tech', '2018-02-06')
sql_insert(con, entities)
```

Результат работы программы:



The screenshot shows the DB Browser for SQLite interface. At the top, the table 'employees' is selected. Below it, a table view displays the data. The table has six columns: id, name, salary, departament, position, and hireDate. There is one row of data with the following values: id=2, name=Andrew, salary=800.0, departament=IT, position=Tech, and hireDate=2018-02-06. The interface includes buttons for 'Добавить запись' (Add record) and 'Удалить запись' (Delete record) at the top right, and navigation controls at the bottom.

id	name	salary	departament	position	hireDate
2	Andrew	800.0	IT	Tech	2018-02-06

Рисунок 2. Созданная запись на странице в DB Browser for SQLite

Пример 4. Обновление данных в таблицах.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sqlite3

def sql_update(con):
    cursor_obj = con.cursor()
    cursor_obj.execute(
        "UPDATE employees SET name = 'Rogers' where id = 2"
```

```

)
con.commit()

if __name__ == "__main__":
    con = sqlite3.connect('mydatabase.db')
    sql_update(con)

```

Результат выполнения:

Таблица: employees Добавить запись Удалить запись

	id	name	salary	departament	position	hireDate
	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр
1	2	Rogers	800.0	IT	Tech	2018-02-06

Рисунок 3. Изменённая запись на странице в DB Browser for SQLite

Пример 5. Выборка данных из таблицы.

Выборка всех данных из таблицы.

Код программы:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sqlite3

def sql_fetch(con):
    cursor_obj = con.cursor()
    cursor_obj.execute("SELECT * FROM employees")
    [print(row) for row in cursor_obj.fetchall()]

if __name__ == "__main__":
    con = sqlite3.connect('mydatabase.db')
    sql_fetch(con)
    con.close()

```

Результат выполнения программы:

```

"C:\Users\Николай Мальцев\AppData\Local\Programs\Python\Python38-64\python.exe"
(2, 'Rogers', 800.0, 'IT', 'Tech', '2018-02-06')

```

Рисунок 4. Результат работы программы в консоли

Выборка определённых данных из таблицы.

Код программы:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sqlite3

```

```
def sql_fetch(con):
    cursor_obj = con.cursor()
    cursor_obj.execute("SELECT id, name FROM employees")
    [print(row) for row in cursor_obj.fetchall()]

if __name__ == "__main__":
    con = sqlite3.connect('mydatabase.db')
    sql_fetch(con)
    con.close()
```

Результат выполнения программы:

```
"C:\Users\Николай Мальце
(2, 'Rogers')
```

Рисунок 5. Результат работы программы в консоли

Пример 6. Получение списка таблиц.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sqlite3

def sql_fetch(con):
    cursor_obj = con.cursor()
    cursor_obj.execute(
        "SELECT name from sqlite_master where type='table'"
    )
    print(cursor_obj.fetchall())

if __name__ == "__main__":
    con = sqlite3.connect('mydatabase.db')
    sql_fetch(con)
    con.close()
```

Результат работы программы:

```
"C:\Users\Николай Мальце
[('employees',)]
```

Рисунок 6. Список таблиц в базе данных

Пример 7. Проверка существования таблицы.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sqlite3
def sql_fetch(con):
    cursor_obj = con.cursor()
    cursor_obj.execute(
        "CREATE TABLE IF NOT EXISTS projects(id INTEGER, name TEXT)"
    )
    con.commit()

if __name__ == "__main__":
    con = sqlite3.connect('mydatabase.db')
    sql_fetch(con)
    con.close()
```

Результат выполнения программы:

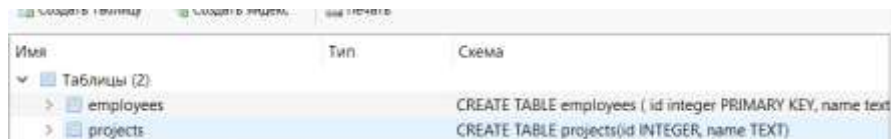


Рисунок 7. Созданная в БД таблица projects

Пример 8. Execute many (массовая вставка).

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect('mydatabase.db')
    cursor_obj = con.cursor()
    cursor_obj.execute(
        "CREATE TABLE IF NOT EXISTS projects(id INTEGER, name TEXT)"
    )
    data = [
        (1, "Ridesharing"),
        (2, "Water Purifying"),
        (3, "Forensics"),
        (4, "Botany")
    ]
    cursor_obj.executemany("INSERT INTO projects VALUES (?, ?)", data)
    con.commit()
    con.close()
```

Результат работы программы:

Таблица: projects

	id	name
Фильтр	Фильтр	
1	1	Ridesharing
2	2	Water Purifying
3	3	Forensics
4	4	Botany

Рисунок 8. Результат добавления данных на страницу

Пример 9. SQLite3 datetime.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sqlite3
import datetime

if __name__ == "__main__":
    con = sqlite3.connect('mydatabase.db')
    cursor_obj = con.cursor()
    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()
    con.close()
```

Результат работы программы:

Таблица:	assignments				
	id	name	date		
	Фильтр	Фильтр	Фильтр		
1	1	Ridesharing	2017-01-02		
2	2	Water Purifying	2018-03-04		

Рисунок 9. Данные формата datetime в таблице assignments

Пример 10. Реализация возможности хранения в базе данных SQLite3 для примера из лабораторной работы 2.17.

Код программы:

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

```

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

Команда для добавления нового работника в БД, прописанная в терминале:

```
PS C:\Users\Николай Мальцев\PycharmProjects\Lab_2.21> cd "Примеры"
PS C:\Users\Николай Мальцев\PycharmProjects\Lab_2.21\Примеры> python example_16.py add --name="Сидоров Сидор" --post="Планы инженер" --year=2012
PS C:\Users\Николай Мальцев\PycharmProjects\Lab_2.21\Примеры>
```

Рисунок 10. Ввод команды в терминал

Результат работы программы:

Таблица: workers

	worker_id	worker_name	post_id	worker_year
	Фильтр	Фильтр	Фильтр	Фильтр
1	1	Сидоров Сидор	1	2012

Рисунок 11. Новая БД и таблица с одной записью

Индивидуальное задание.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sqlite3
from pathlib import Path
import argparse
import typing as t

def create_db(database_path: Path) -> None:
    """
    Создать базу данных
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    # Создать таблицу с информацией о дате рождения
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS birth (
            date_id INTEGER PRIMARY KEY AUTOINCREMENT,
            birth_date INTEGER NOT NULL
        )
    """
    )
```

```

# Создать таблицу с информацией о пользователях
cursor.execute(
    """
    CREATE TABLE IF NOT EXISTS users (
        user_id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_name TEXT NOT NULL,
        user_number TEXT NOT NULL,
        date_id INTEGER NOT NULL,
        FOREIGN KEY(date_id) REFERENCES birth(date_id)
    )
    """
)
conn.close()
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,
            '-' * 20
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^20} |'.format(
                "№",
                "Имя",
                "Номер телефона",
                "Дата рождения"
            )
        )
        print(line)
        # Вывести данные о всех сотрудниках
        for idx, user in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>20} |'.format(
                    idx,
                    user.get('name', ''),
                    user.get('number', ''),
                    user.get('year', 0)
                )
            )
            print(line)
    else:
        print("Список работников пуст.")
def add_worker(
    database_path: Path,
    name: str,
    number: str,
    date: 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 date_id FROM birth WHERE birth_date = ?
            """,
            (date,)
        )
        row = cursor.fetchone()
        if row is None:
            cursor.execute(
                """
                INSERT INTO birth (birth_date) VALUES (?)
                """,
                (date,)
            )
            date_id = cursor.lastrowid
        else:
            date_id = row[0]

        # Добавить информацию о новом работнике
        cursor.execute(
            """
            INSERT INTO users (user_name, date_id, user_number)
            VALUES (?, ?, ?)
            """,
            (name, date_id, number)
        )
        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 users.user_name, birth.birth_date, users.user_number
        FROM users
        INNER JOIN birth ON birth.date_id = users.date_id
        """
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "name": row[0],
            "number": row[1],
            "year": row[2],
        }
        for row in rows
    ]
def select_by_period(
    database_path: Path, pnumber: int
) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать всех пользователей с заданным номером телефона.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT users.user_name, birth.birth_date, users.user_number
        FROM users
        """
    )

```

```

        INNER JOIN birth ON birth.date_id = users.date_id
        WHERE users.user_number == ?
        """
        (pnumber,)
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "name": row[0],
            "number": 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() / "users.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 пользователей.
    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",
        "--phone",
        action="store",
        help="The worker's post"
    )
    add.add_argument(
        "-b",
        "--birth",
        action="store",
        type=int,
        required=True,
        help="Birthdate"
    )
    # Создать субпарсер для отображения всех пользователей.
    _ = subparsers.add_parser(
        "display",

```

```

        parents=[file_parser],
        help="Display all workers"
    )

    # Создать субпарсер для выбора пользователей.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the workers"
    )
    select.add_argument(
        "-N",
        "--number",
        action="store",
        type=int,
        required=True,
        help="The required phone number"
    )

    # 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 пользователей.
    if args.command == "add":
        add_worker(db_path, args.name, args.phone, args.birth)

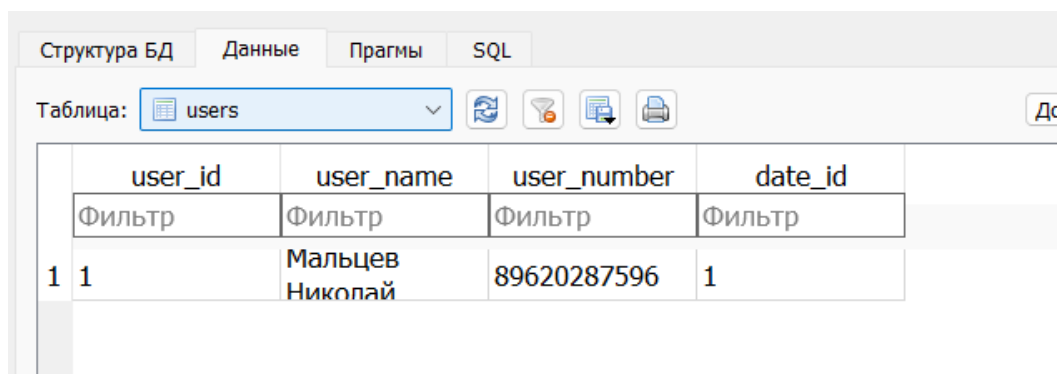
    # Отобразить всех рпользователей.
    elif args.command == "display":
        display_workers(select_all(db_path))

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

if __name__ == "__main__":
    main()

```

Результат работы программы:



The screenshot shows a database management interface with tabs for 'Структура БД', 'Данные', 'Прагмы', and 'SQL'. The 'Данные' tab is active, showing a table named 'users'. The table has four columns: 'user_id', 'user_name', 'user_number', and 'date_id'. Below the column headers, there are filter boxes. The first row of data shows 'user_id' as 1, 'user_name' as 'Мальцев Николай', 'user_number' as '89620287596', and 'date_id' as 1.

	user_id	user_name	user_number	date_id
1	1	Мальцев Николай	89620287596	1

Рисунок 12. Созданная база данных и запись в одной из её таблиц

Ответы на вопросы:

1. Каково назначение модуля SQLite3?

Данный модуль является API к СУБД SQLite и представляет из себя своего рода адаптер, который переводит команды, написанные на Python в запросы SQLite.

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

Соединение с БД происходит при помощи метода connect класса sqlite3.

Курсор — это механизм, который позволяет перемещаться по записям в базе данных. Курсоры облегчают последующую обработку в сочетании с обходом, такую как извлечение, добавление и удаление записей базы данных. Курсор базы данных, характерный для обхода, делает курсоры похожими на концепцию итератора в языке программирования.

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

Для подключения к БД в оперативной памяти необходимо методу connect передать в качестве параметра строку не с названием базы данной, а строку «:memory:».

4. Как корректно завершить работу с базой данных?

После обращения к БД необходимо закрывать соединение, вызвав метод close.

5. Как осуществляется вставка данных в таблицу базы данных?

С помощью оператора INSERT INTO <название БД> VALUES (<данные, перечисленные через запятую>).

6. Как осуществляется обновление данных таблицы базы данных?

С помощью оператора UPDATE <название БД> SET <поле> = <значение> where <поле> = <значение>

7. Как осуществляется выборка данных из базы данных?

С помощью оператора SELECT <поле> FROM <название БД>

8. Каково назначение метода rowcount?

Используется для возврата количества строк, которые были затронуты последним запросом.

9. Как получить список всех таблиц базы данных?

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 её добавлении, так и при её 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?

Метод `executemany` можно использовать для вставки нескольких строк одновременно.

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

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

Вывод: в ходе работы были изучены возможности взаимодействия с базами данных SQLite3 с помощью языка программирования Python версии 3.10.