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
№220дисциплины «Основы программной инженерии»

Выполнил:
Джараян Арег Александрович
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и сопровождение программного обеспечения», очная форма обучения

_____
(подпись)

Проверил Воронкин Роман Александрович
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Лабораторная работа 4.1 Элементы объектно-ориентированного программирования в языке Python.

Цель работы: приобретение навыков по работе с классами объектами при написании программ с помощью языка программирования Python версии 3.x.

Ход работы.

1. Создание нового репозитория с лицензией MIT.

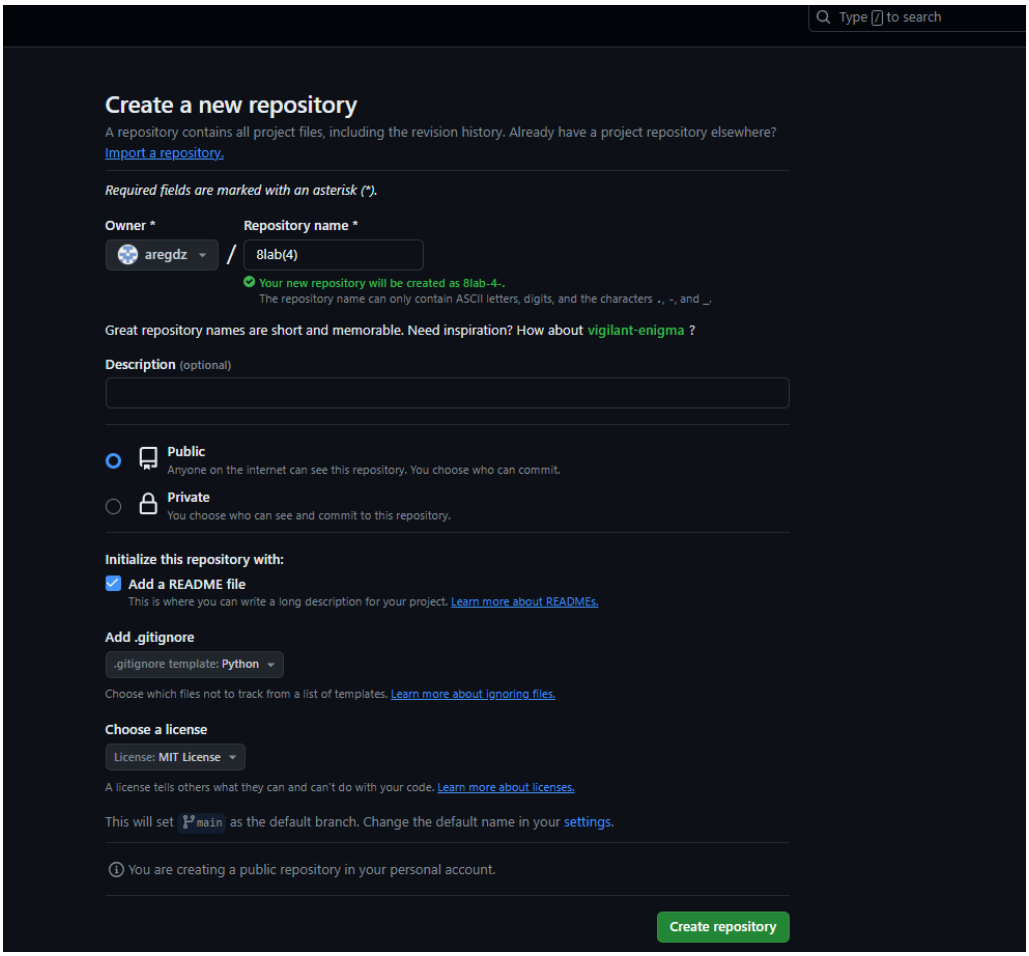

Рисунок 1 – создание репозитория

2. Клонировал репозиторий на рабочий ПК.

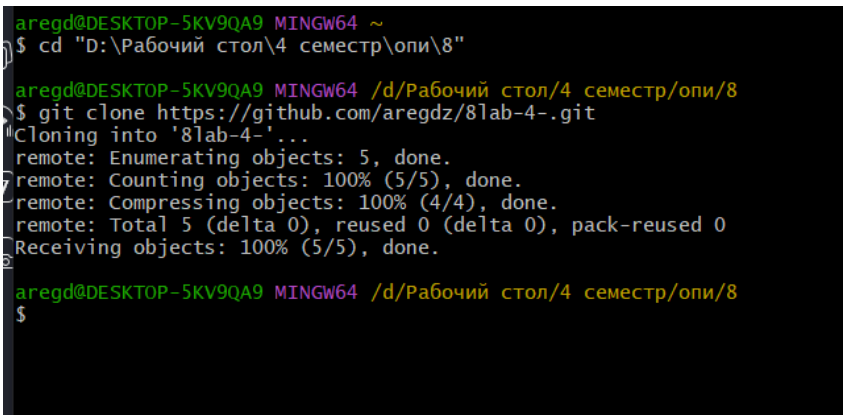

Рисунок 2 – клонирование репозитория

3. Дополнил файл .gitignore необходимыми инструкциями.

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
```

Рисунок 4 – Файл .gitignore

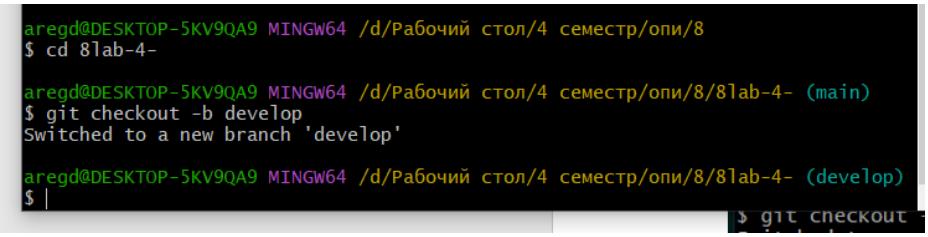

Рисунок 4 – организация ветки

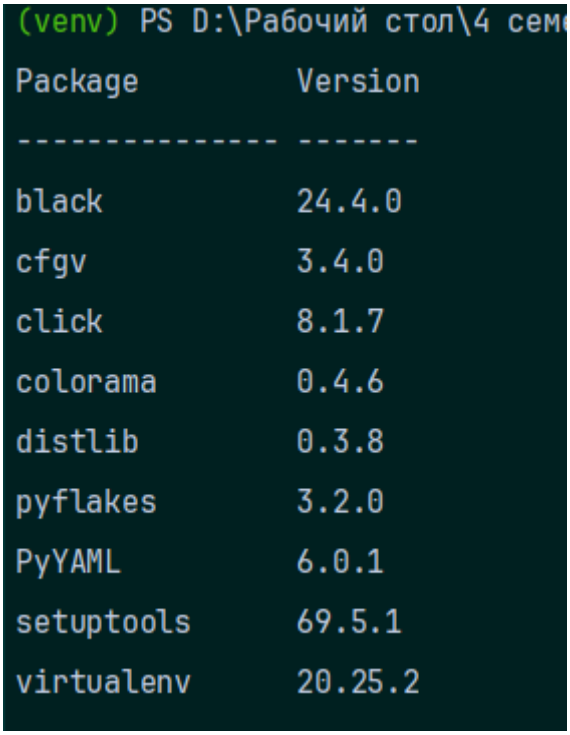

Рисунок 5 – создание виртуального окружения

4.Пример 1.

```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(
                "№",
                "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))

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

Рисунок 6 – пример 1

5. 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.

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

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


def display_flights(flights: t.List[t.Dict[str, t.Any]]) -> None:
    if flights:
        line = "+-{}-+-{}-+".format("-" * 30, "-" * 8)
        print(line)
        print("| {:^30} | {:^8} |".format("Departure Date", "Destination"))
        print(line)
        for flight in flights:
            print("| {:<30} | {:>8} |".format(flight.get("departure_date", ""),
flight.get("destination", "")))
        print(line)
    else:
        print("List of flights is empty.")


def create_db(database_path: Path) -> None:
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS flights (
            flight_id INTEGER PRIMARY KEY AUTOINCREMENT,
            destination TEXT NOT NULL,
            departure_date TEXT NOT NULL,
            aircraft_type_id INTEGER,
            FOREIGN KEY (aircraft_type_id) REFERENCES aircraft_types (id)
        )
        """
    )
    cursor.execute(
        """
```

```
        CREATE TABLE IF NOT EXISTS aircraft_types (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            type TEXT NOT NULL
        )
        """
    )
    conn.close()


def add_flight(database_path: Path, destination: str, departure_date: str,
aircraft_type_id: int) -> None:
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        INSERT INTO flights (destination, departure_date, aircraft_type_id)
        VALUES (?, ?, ?)
        """,
        (destination, departure_date, aircraft_type_id)
    )
    conn.commit()
    conn.close()


def select_flights(database_path: Path, date: str) -> t.List[t.Dict[str, t.Any]]:
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT flights.departure_date, flights.destination, aircraft_types.type
        FROM flights
        JOIN aircraft_types ON flights.aircraft_type_id = aircraft_types.id
        WHERE departure_date = ?
        """,
        (date,)
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "departure_date": row[0],
            "destination": row[1],
            "aircraft_type": row[2]
        }
        for row in rows
    ]


def main(command_line=None):
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename", action="store", help="The database file name"
    )

    parser = argparse.ArgumentParser("flights")
    parser.add_argument(
        "--version", action="version", version="%(prog)s 0.1.0"
    )
    subparsers = parser.add_subparsers(dest="command")

    create_db(Path("flights.db"))

    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(
        "-dd",
        "--departure_date",
        action="store",
        required=True,
        help="Departure date of the flight",
    )
    add.add_argument(
        "-at",
        "--aircraft_type",
        action="store",
        required=True,
        help="Aircraft type of the flight",
    )

    _ = subparsers.add_parser(
        "display", parents=[file_parser], help="Display all flights"
    )

    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select flights by departure date",
    )
    select.add_argument(
        "-D",
        "--date",
        action="store",
        required=True,
        help="Departure date to select flights",
    )

    args = parser.parse_args(command_line)

    if args.command == "add":
        conn = sqlite3.connect(args.filename)
        cursor = conn.cursor()
        cursor.execute(
            """
            INSERT INTO aircraft_types (type)
            VALUES (?)
            """,
            (args.aircraft_type,)
        )
        aircraft_type_id = cursor.lastrowid
        conn.commit()
        conn.close()

        add_flight(
            Path(args.filename), args.destination, args.departure_date,
aircraft_type_id
        )

    elif args.command == "display":
        all_flights = select_flights(Path(args.filename), "")
        display_flights(all_flights)

    elif args.command == "select":
        selected_flights = select_flights(Path(args.filename), args.date)
        display_flights(selected_flights)


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

Рисунок 8 – Выполнение задания 1

```
[notice] To update, run: D:\Рабочий стол\4 семестр\опи\9\9lab\venv\Scripts\python.exe -m pip install --upgrade pip
(venv) PS D:\Рабочий стол\4 семестр\опи\8\8lab-4-> python 1zadanie.py add -d "Москва" -dd "2023-02-11" -at "Boeing 74437" flights.db
(venv) PS D:\Рабочий стол\4 семестр\опи\8\8lab-4->
```

Рисунок 9 – пример выполнение задания 1

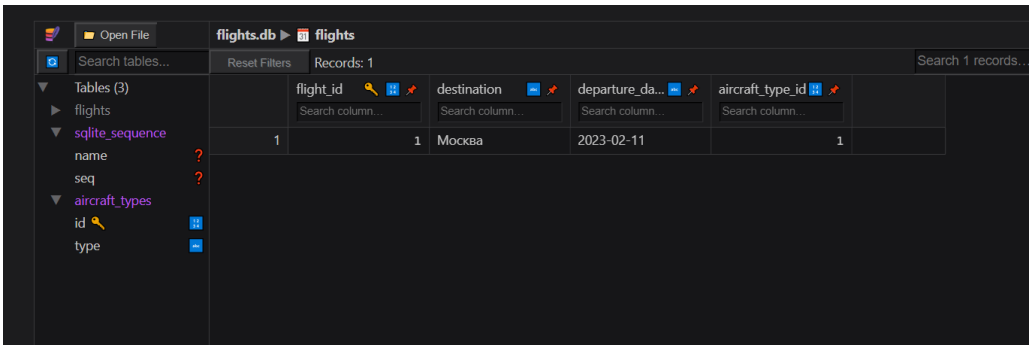

Рисунок 10 – база данных

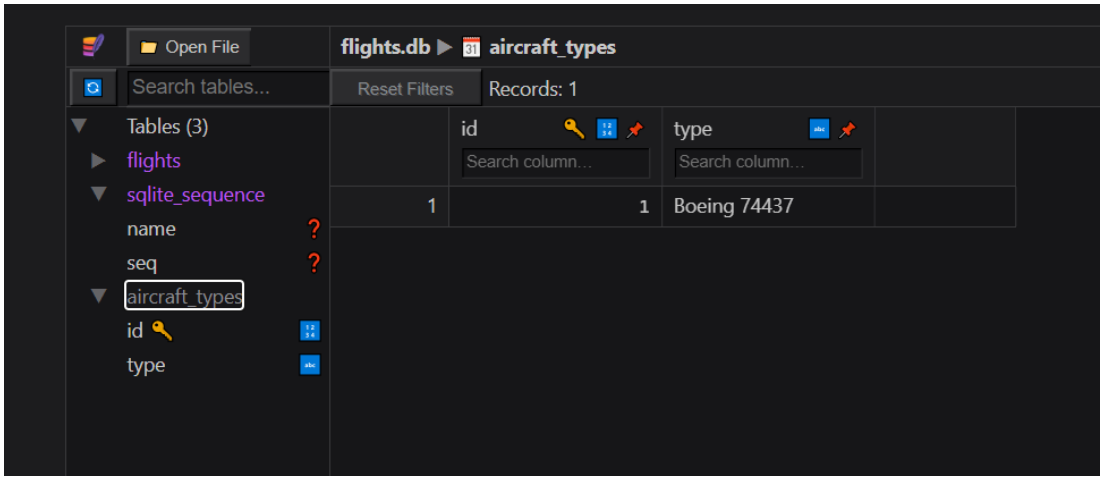

Рисунок 11 – база данных

Контрольные вопросы:

1.Назначение модуля sqlite3:

Модуль sqlite3 в Python предоставляет интерфейс для взаимодействия с базой данных SQLite. Он позволяет создавать, управлять и взаимодействовать с базами данных SQLite из кода Python.

2.Соединение с базой данных SQLite3 и курсор:

Для соединения с базой данных SQLite3 используется функция connect() модуля sqlite3. Эта функция принимает путь к файлу базы данных. После

установления соединения создается объект курсора базы данных с помощью метода cursor(). Курсор используется для выполнения SQL-запросов и получения результатов.

3.Подключение к базе данных SQLite3 в оперативной памяти:

Для подключения к базе данных SQLite3, расположенной в оперативной памяти, вместо имени файла базы данных нужно использовать строку :memory: при вызове функции connect().

4.Завершение работы с базой данных SQLite3:

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

5.Вставка данных в таблицу базы данных SQLite3:

Для вставки данных в таблицу используется SQL-запрос INSERT INTO. Метод execute() объекта курсора используется для выполнения запроса.

6.Обновление данных таблицы базы данных SQLite3:

Для обновления данных в таблице используется SQL-запрос UPDATE. Метод execute() объекта курсора также используется для выполнения запроса.

7.Выборка данных из базы данных SQLite3:

Для выборки данных из таблицы используется SQL-запрос SELECT. Метод execute() объекта курсора также используется для выполнения запроса, а методы fetchone(), fetchall() или fetchmany() - для получения результатов выборки.

8.Назначение метода rowcount:

Метод rowcount возвращает количество строк, затронутых последним выполненным запросом. Он может быть использован для определения числа строк, вставленных, обновленных или удаленных.

9.Получение списка всех таблиц базы данных SQLite3:

Для получения списка всех таблиц в базе данных SQLite3 можно выполнить SQL-запрос к системной таблице sqlite_master.

10.Проверка существования таблицы:

Для проверки существования таблицы можно выполнить SQL-запрос к системной таблице sqlite_master и проверить наличие нужной таблицы в результатах запроса.

11.Массовая вставка данных в базу данных SQLite3:

Массовая вставка данных в базу данных SQLite3 может быть выполнена с помощью метода executemany() объекта курсора. Этот метод позволяет выполнить множество операций вставки данных за один вызов.

12.Работа с датой и временем:

Для работы с датой и временем в базе данных SQLite3 обычно используются типы данных DATE, TIME, DATETIME или TIMESTAMP. Вы можете использовать функции SQLite для работы с датой и временем в SQL-запросах. Например, функции DATE(), TIME(), DATETIME(), NOW(), DATE() и другие.