Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

09.03.01 "Информатика и вычислительная техника"

профиль "Программное обеспечение средств вычислительной техники и автоматизированных систем"

**ОТЧЕТ**

по учебной практике

на кафедре Прикладной Математики и Кибернетики

Выполнил:

студент гр. ИП-311 Скурятин Дмитрий Евгеньевич

«09» апреля 2025г.

Руководитель практики

доцент каф. ПМиК Приставка П.А.

«09» апреля 2025г. Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Новосибирск 2025 г.

Оглавление

[1. Условие задачи 3](#_Toc195051529)

[2. Описание используемых алгоритмов 4](#_Toc195051530)

[3. Листинг программы 5](#_Toc195051531)

[4. Результаты тестирования 8](#_Toc195051532)

[5. Список используемых источников 9](#_Toc195051533)

# 1. Условие задачи

Разработать программу реализующую ввод, хранение и обработку данных о 250 лучших фильмах на основе данных сайта https:/www.kinopoisk.ru/lists/movies/top250/

Общие требования к программе:

1. Язык разработки: Python версии не ниже 3.x

2. Операционная система: определяются студентом

3. Набор свойств фильмов:

* Name – наименование
* alternativeName
* Year – год выпуска
* shortDescription – короткое описание
* movieLength- длительность
* top250 – позиция в рейтинге

4. Ввод данных

Непосредственно с главной страницы сайта https:/www.kinopoisk.ru/lists/movies/top250/ в момент запуска программы. Загрузка и парсинг веб-страницы производится с помощью библиотек Requests и Beautifulsoup или их аналогов. Примечание: допускается считывание строчек в количестве менее 250 (Например, 10 строчек с данными о фильмах).

5. Хранение:

Типы и структуры для хранения данных: определяются студентом

6. Обработка:

Реализовать функцию поиска информации о свойствах фильма по его названию.

# 2. Описание используемых алгоритмов

Программа взаимодействует с API Кинопоиска, загружает данные о фильмах, сохраняет их в базу данных SQLite и предоставляет интерфейс для поиска и вывода информации.

1. **fetch\_movies\_data() -** Получение данных о фильмах через API.

Отправляет GET-запрос к API\_URL с заголовком X-API-KEY для аутентификации.

Параметры limit=10 и selectFields указывают, что нужно получить 10 фильмов с определёнными полями (название, год, описание и т.д.).

Возвращает список фильмов docs ответа API. В случае ошибки выводит сообщение и возвращает None.

2. **create\_connection(db\_file)** - Подключение к SQLite-базе данных.

Создаёт соединение с файлом базы данных “movies.db”, если файла нет, он будет создан.

3. **create\_table(conn)** - Создание таблицы movies в базе данных.

Таблица содержит поля: id (автоинкремент), name, alternativeName, year, shortDescription, movieLength, top250.

Использует SQL-запрос с использованием IF NOT EXISTS, чтобы избежать повторного создания.

4. **insert\_data(conn, data)** - Запись данных о фильмах в таблицу.

Для каждого фильма из data выполняет SQL-запрос INSERT, подставляя значения из API-ответа.

Поля, отсутствующие в API-ответе, заменяются на значения по умолчанию (например, "Неизвестно" или 0).

Фиксирует изменения в базе (conn.commit()).

5. **clear\_table(conn)** - Очистка таблицы movies.

Удаляет все записи с помощью DELETE FROM movies.

Используется для обновления данных при каждом запуске программы.

6. **output\_films(conn)** - Вывод всех фильмов из таблицы.

Выполняет SQL-запрос SELECT \* FROM movies и возвращает результат.

7. **search\_movie(conn, name) -** Поиск фильмов по названию.

Использует SQL-запрос с LIKE для нечёткого поиска.

Возвращает все совпадения.

8. **main()** - Основная логика программы.

**1 -** Загружает данные через fetch\_movies\_data().

**2** - Создаёт или подключается к базе данных “movies.db”.

**3** - Создаёт таблицу, очищает её и вставляет новые данные.

**4** -Предлагает пользователю выбрать действие:

**Клавиша "1"**: Вывести все фильмы из базы.

**Клавиша "2"**: Поиск по названию. Поиск продолжается до ввода exit.

# 3. Листинг программы

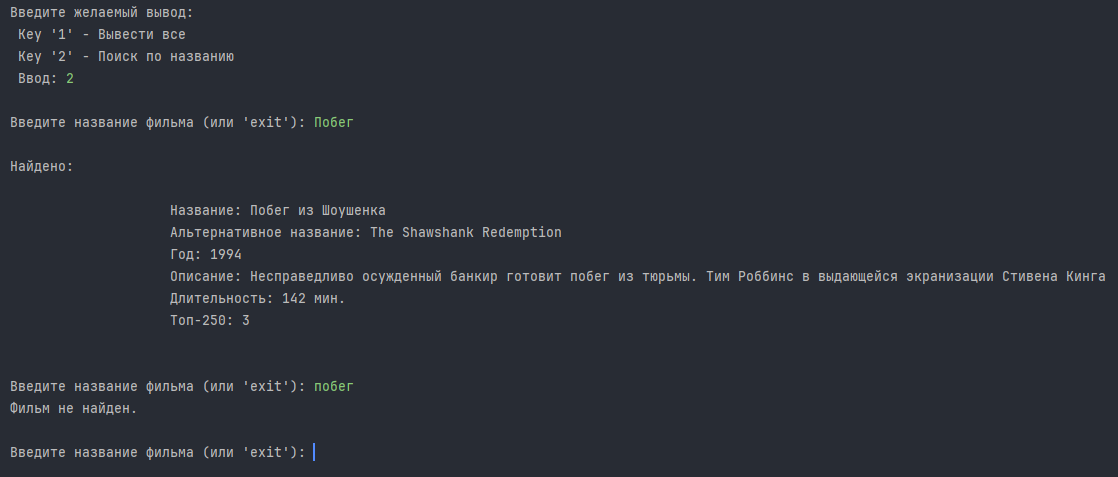
*import* requests  
*import* sqlite3  
*from* sqlite3 *import* Error  
  
API\_KEY = "SQ5YV86-JE24ZGB-NAWT0E9-MCDKQQ3"  
API\_URL = "https://api.kinopoisk.dev/v1.4/movie?page=1&limit=10&lists=top250"  
  
  
*def* fetch\_movies\_data():  
 headers = {"X-API-KEY": API\_KEY}  
 params = {  
 "limit": 10,  
 "selectFields": ["name", "alternativeName", "year", "shortDescription", "movieLength", "top250"]  
 }  
  
 *try*:  
 response = requests.get(API\_URL, headers=headers, params=params)  
 data = response.json()  
 *return* data.get("docs", [])  
 *except* Exception *as* e:  
 print(f"Ошибка при получении данных: {e}")  
 *return None  
  
  
def* create\_connection(db\_file):  
 *try*:  
 *return* sqlite3.connect(db\_file)  
 *except* Error *as* e:  
 print(f"Ошибка подключения к БД: {e}")  
 *return None  
  
  
def* create\_table(conn):  
 sql = """  
 CREATE TABLE IF NOT EXISTS movies (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name TEXT NOT NULL,  
 alternativeName TEXT,  
 year INTEGER,  
 shortDescription TEXT,  
 movieLength INTEGER,  
 top250 INTEGER  
 )  
 """  
 conn.execute(sql)  
  
  
*def* insert\_data(conn, data):  
 sql = """  
 INSERT INTO movies (name, alternativeName, year, shortDescription, movieLength, top250)  
 VALUES (?, ?, ?, ?, ?, ?)  
 """  
 cursor = conn.cursor()  
 *for* movie *in* data:  
 *try*:  
 cursor.execute(sql, (  
 movie.get("name", "Неизвестно"),  
 movie.get("alternativeName", "Нет"),  
 movie.get("year", 0),  
 movie.get("shortDescription", "Нет описания"),  
 movie.get("movieLength", 0),  
 movie.get("top250")  
 ))  
 *except* Error *as* e:  
 print(f"Ошибка вставки: {e}")  
 conn.commit()  
  
  
*def* clear\_table(conn):  
 conn.execute("DELETE FROM movies")  
 conn.commit()  
  
  
*def* output\_films(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT \* FROM movies")  
 *return* cursor.fetchall()  
  
  
*def* search\_movie(conn, name):  
 cursor = conn.cursor()  
 cursor.execute("SELECT \* FROM movies WHERE name LIKE ?", (f"%{name}%",))  
 *return* cursor.fetchall()  
  
  
*def* main():  
 movies\_data = fetch\_movies\_data()  
 *if not* movies\_data:  
 print("Не удалось загрузить данные.")  
 *return* conn = create\_connection("movies.db")  
 *if not* conn:  
 *return* create\_table(conn)  
 clear\_table(conn)  
 insert\_data(conn, movies\_data)  
  
 targ = input("Введите желаемый вывод: \n Key '1' - Вывести все\n Key '2' - Поиск по названию\n Ввод: ")  
 *if* targ == '2':  
 *while True*:  
 search\_term = input("\nВведите название фильма (или 'exit'): ").strip()  
 *if* search\_term.lower() == "exit":  
 *break* results = search\_movie(conn, search\_term)  
 *if* results:  
 print("\nНайдено:")  
 *for* row *in* results:  
 print(f"""  
 Название: {row[1]}  
 Альтернативное название: {row[2]}  
 Год: {row[3]}  
 Описание: {row[4]}  
 Длительность: {row[5]} мин.  
 Топ-250: {row[6] *if* row[6] *else* 'Нет данных'}  
 """)  
 *else*:  
 print("Фильм не найден.")  
  
 conn.close()  
 *elif* targ == '1':  
 result = output\_films(conn)  
 *if* result:  
 *for* row *in* result:  
 print(f"""  
 Название: {row[1]}  
 Альтернативное название: {row[2]}  
 Год: {row[3]}  
 Описание: {row[4]}  
 Длительность: {row[5]} мин.  
 Топ-250: {row[6] *if* row[6] *else* 'Нет данных'}  
 """)  
 *else*:  
 print("Фильмs не найдены.")  
  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 main()

# 4. Результаты тестирования

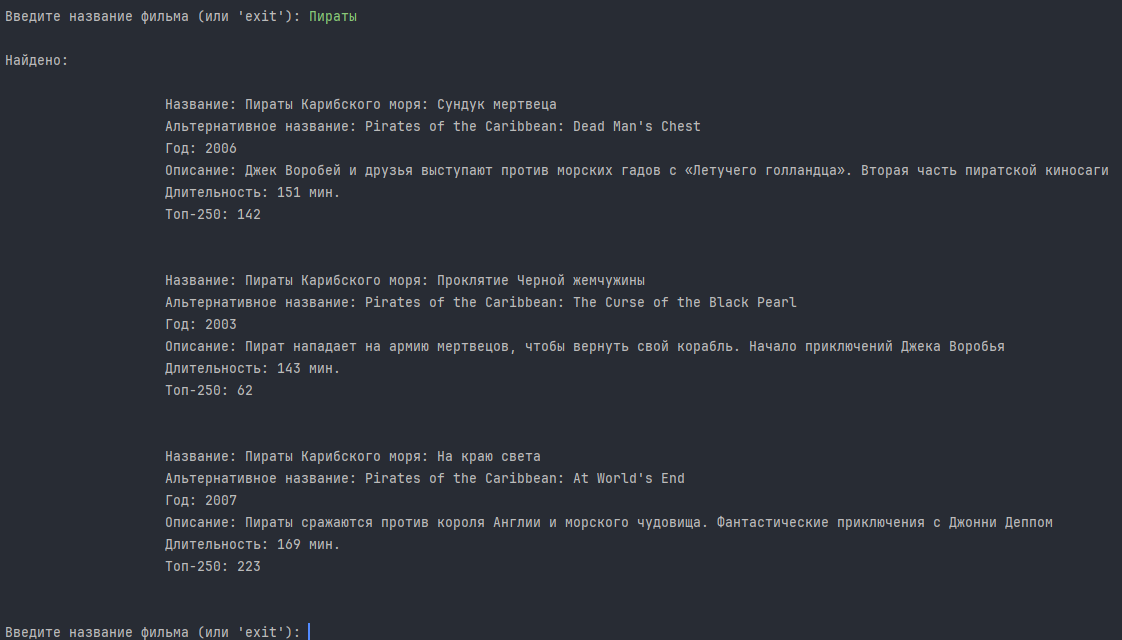
При запуске программы происходит создание и заполнение базы данных:

«Типо скриншот базы данных»

Программа чувствительна к регистру ввода:



Если фильмов несколько с похожим названием, то выводятся все:



# 5. Список используемых источников