Изображение выглядит как эмблема, символ, герб, нашивка

Автоматически созданное описание

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА №1**

**по дисциплине**

**«Системный анализ данных в системах поддержки принятия**

**решений»**

**Онтология**

Студент группы:ИКБО-04-22 \_\_Кликушин В.И.\_ *(Ф. И.О. студента)*

Преподаватель \_\_Железняк Л.М.\_\_

*(Ф.И.О. преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc178504398)

[1 ОНТОЛОГИЯ 5](#_Toc178504399)

[1.1 Понятие онтологии 5](#_Toc178504400)

[1.2 Постановка задачи 6](#_Toc178504401)

[1.3 Описание предметной области 6](#_Toc178504402)

[1.4 Иерархия классов 6](#_Toc178504403)

[1.5 Реализация в Protégé 7](#_Toc178504404)

[1.6 Результаты работы программы 11](#_Toc178504405)

[ЗАКЛЮЧЕНИЕ 15](#_Toc178504406)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 16](#_Toc178504407)

[ПРИЛОЖЕНИЯ 17](#_Toc178504408)

ВВЕДЕНИЕ

В современном мире, характеризующемся бурным развитием информационных технологий, возникает острая необходимость в систематизации и структуризации знаний. Традиционные методы хранения и обработки информации оказываются неэффективными перед лицом колоссальных объемов данных, их разнообразных форматов и чрезвычайной зашумленности. В этих условиях важную роль играет онтология – область знаний, занимающаяся формальным описанием концепций и отношений между ними в какой-либо предметной области.

Возникновение онтологий и их стремительное развитие связано с проявлением в нашей реальности следующих новых факторов:

* колоссальный рост объемов информации, предъявляемых для обработки (анализа, использования) специалистам самых различных областей деятельности;
* чрезвычайная зашумленность этих потоков (повторы, противоречивость, разноуровневость);
* острая необходимость в использовании одних и тех же знаний разными специалистами в разных целях;
* всеобщая интернетизация нашей жизни и острая необходимость в структуризации информации для её представления пользователям и более эффективного поиска;
* необходимость сокращения времени на поиск нужной информации и повышения качества информационных услуг в Интернете.

1 онтология

1.1 Понятие онтологии

Появление онтологий стало ответом ряда наук, связанных с информационными технологиями и системами искусственного интеллекта на перечисленные проблемы. Именно они обеспечили возможность их перехода на новый качественный уровень обработки и поиска информации. Наиболее распространенными стали следующие определения.

Онтология – это точная спецификация концептуализации.

Онтология – это формальная точная спецификация совместно используемой концептуализации.

Онтологии – это базы знаний специального типа, которые могут читаться и пониматься, отчуждаться от разработчика и/или физически разделяться их пользователями.

Под концептуализацией понимается абстрактная модель явлений (процессов) в мире, составленная посредством определения существенных для описания данных явлений понятий, т.е. концептов. Точность подразумевает, что типы используемых понятий и ограничения на область применения данных понятий явно определены. Формальность означает, что онтология должна быть ориентирована на компьютерное представление, что исключает использование естественных языков в полной мере, в связи с их неоднозначностью и сложностью. Совместное использование отражает понятие того, что онтология описывает всеобщие знания, т.е. не персональные знания одного человека, а знания, принятые в группе, сообществе.

1.2 Постановка задачи

Выбрать предметную область по личному интересу и изучить её. Составить модель онтологии по предметной области. Реализовать модель созданной онтологии в программе Protégé. Написать программный продукт на выбранном языке программирования, реализующий построенную модель онтологии.

1.3 Описание предметной области

В качестве предметной области выбрана «Киноиндустрия». Актуальность выбранной предметной области заключается в том, что просмотр фильмов и сериалов – неотъемлемая часть времяпровождения многих людей на планете. Онтология киноиндустрии — это мощный инструмент, который может значительно повысить эффективность работы в этой сфере, улучшить качество информации и стимулировать развитие инновационных сервисов. Основными понятиями для рассматриваемой предметной области являются фильмы, сериалы, актёры и режиссёры. Модель онтологии позволяет найти взаимоотношения между объектами, например, узнать в каких фильмах снимался определённый актёр. Яркий пример сервиса, использующего схожую онтологию – «Кинопоиск». В этом сервисе онтология используется для организации контента и предоставления персонализированных рекомендаций.

1.4 Иерархия классов

В рассматриваемой онтологии предполагается иерархия классов, в которой, родительским классом является непосредственно абстрактный класс «Киноиндустрия». Киноиндустрия включает в себя фильмы, актёров и режиссёров. В свою очередь фильмы включают в себя многосерийные фильмы и полнометражные фильмы. Иерархия классов показана на рисунке 1.4.1.

Изображение выглядит как диаграмма, текст, линия, План

Автоматически созданное описание

Рисунок 1.4.1 – Иерархия классов онтологии

1.5 Реализация в Protégé

Иерархия онтологии была перенесена в Protégé (Рисунок 1.5.1).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 1.5.1 – Иерархия классов в Protégé

Класс «Киноиндустрия» является абстрактным. Созданы слоты для класса «Фильм» (Рисунок 1.5.2).

Изображение выглядит как текст, программное обеспечение, снимок экрана, линия

Автоматически созданное описание

Рисунок 1.5.2 – Слоты для класса «Фильм»

Слот актёры содержит ссылку на объект класса «Актёр», слот название имеет строковый тип. Созданы слоты для класса «Многосерийный фильм» (Рисунок 1.5.3).

Изображение выглядит как текст, линия, число, снимок экрана

Автоматически созданное описание

Рисунок 1.5.3 - Слоты для класса «Многосерийный фильм»

Класс «Многосерийный фильм» содержит дополнительные слоты «Количество сезонов» и «Количество эпизодов». Созданы слоты для класса «Полнометражный фильм» (Рисунок 1.5.4).

Изображение выглядит как текст, линия, число, Параллельный

Автоматически созданное описание

Рисунок 1.5.4 - Слоты для класса «Полнометражный фильм»

Объекты класса «Полнометражный фильм» и «Многосерийный фильм» также являются экземплярами класса «Фильм» и содержат слоты, определенные в классе «Фильм». Созданы слоты для класса «Актёр» (Рисунок 1.5.5).

Изображение выглядит как текст, линия, число, снимок экрана

Автоматически созданное описание

Рисунок 1.5.5 – Слоты для класса «Актёр»

Класс «Актёр» содержит единственный слот «ФИО», имеющий строковый тип. Созданы слоты для класса «Режиссёр» (Рисунок 1.5.6).

Изображение выглядит как текст, линия, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 1.5.6 – Слоты для класса «Режиссёр»

Класс «Режиссёр» содержит слот «ФИО», имеющий строковый тип и слот «Руководит», содержащий ссылки на экземпляры класса «Фильм».

Создано несколько объектов классов (Рисунок 1.5.7).

Изображение выглядит как текст, программное обеспечение, снимок экрана, число

Автоматически созданное описание

Рисунок 1.5.7 – Созданные объекты классов

Для каждого объекта заполнены соответствующие слоты (Рисунок 1.5.8).

Изображение выглядит как текст, программное обеспечение, линия, Шрифт

Автоматически созданное описание

Рисунок 1.5.8 – Пример заполнения слотов для созданного экземпляра класса

Произведено несколько запросов (Рисунок 1.5.9 – 1.5.11).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 1.5.9 – Запрос от класса «Режиссёр»

Запрос позволяет узнать режиссёров, которые руководят выбранным фильмом.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 1.5.10 – Запрос от класса «Полнометражный фильм»

Запрос показывает сущности актёров, которые снимаются в выбранном фильме.

Изображение выглядит как текст, снимок экрана, линия, программное обеспечение

Автоматически созданное описание

Рисунок 1.5.11 – Запрос от класса «Актёр»

Запрос позволяет узнать, в каких фильмах снимался определенный актёр, а затем узнать режиссеров этих фильмов.

1.6 Результаты работы программы

Разработан программный продукт, описывающий созданную модель онтологии. Для большей наглядности добавлены дополнительные объекты в каждый класс (Рисунок 1.6.1).

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 1.6.1 – Созданные объекты классов

Результаты выполнения запросов от каждого классы представлены на рисунках 1.6.2–1.6.6.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.6.2 – Запрос от класса «Фильм»

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.6.3 – Запрос от класса «Многосерийный фильм»

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 1.6.4 – Запрос от класса «Полнометражный фильм»

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 1.6.5 – Запрос от класса «Актёр»

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 1.6.6 – Запрос от класса «Режиссёр»

Заключение

В ходе выполнения данной практической работы изучены теоретические основы системного анализа и использования онтологий в широком ряде задач, получены навыки построения онтологий и работы с ними, включая создание классов для описания выбранной предметной области, создание слотов в классах и создание экземпляров. С помощью инструменты работы с онтологиями Protégé выполнены запросы на получение объектов по различным запросам.   
В качестве закрепления полученных знаний написана программа на языке Python, способная работать с онтологией выбранной предметной области. В её функционал входит возможность писать запросы на получение экземпляров и связанных объектов.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Protege URL: <https://protegewiki.stanford.edu/wiki/ProtegeDocs> (Дата обращения: 16.09.2024).
2. Сорокин А. Б. Теория графов в машинном обучении [Электронный ресурс]: учебное пособие / А. Б. Сорокин, Р. Э. Семенов. — М.: РТУ МИРЭА, 2023.
3. Сорокин А. Б. Введение в роевой интеллект: теория, расчет и приложения [Электронный ресурс]: учебно-метод. пособие / А. Б. Сорокин. — М.: РТУ МИРЭА, 2019.
4. Тузовский А. Ф., Чириков С. В., Ямпольский В. З. Системы управления знаниями (методы и технологии). – Томск: Издательство НТЛ, 2020 – 260 c.

ПРИЛОЖЕНИЯ

Приложение А – Код реализации онтологии на языке Python.

Приложение А

Код реализации онтологии на языке Python.

*Листинг А.1. Реализация онтологии.*

QUERY = {'Фильм': ('Актёры', 'Название'),

'Многосерийный фильм': ('Актёры', 'Название', 'Количество сезонов', 'Количество эпизодов'),

'Полнометражный фильм': ('Актёры', 'Название', 'Длительность'),

'Актёр': ('ФИО', ),

'Режиссёр': ('ФИО', 'Руководит')}

FUNCTION = {'Актёры': ('contains', 'does not contain'),

'Название': ('contains', 'does not contain', 'is',

'is not', 'begins with', 'ends with'),

'Количество сезонов': ('is', 'is greater then', 'is less then'),

'Количество эпизодов': ('is', 'is greater then', 'is less then'),

'Длительность': ('is', 'is greater then', 'is less then'),

'ФИО': ('contains', 'does not contain', 'is',

'is not', 'begins with', 'ends with'),

'Руководит': ('contains', 'does not contain')}

class MovieIndustry:

def \_\_init\_\_(self, \*args, \*\*kwargs):

raise TypeError(f"Can't instantiate abstract class {

\_\_class\_\_.\_\_name\_\_}")

class Film(MovieIndustry):

def \_\_init\_\_(self, name, actors):

self.\_name = name

self.\_actors = list(actors)

def \_\_str\_\_(self):

return self.\_name

def \_\_repr\_\_(self):

return f"{\_\_class\_\_.\_\_name\_\_}('{self.\_name}', {self.\_actors})"

def \_\_eq\_\_(self, other):

if type(other) is \_\_class\_\_:

return other.\_name == self.\_name

elif type(other) is str:

return other == self.\_name

return NotImplemented

def \_\_contains\_\_(self, obj):

return obj in self.\_name

def has\_actor(self, actor):

return actor in self.\_actors

class SerialFilm(Film):

def \_\_init\_\_(self, name, actors, num\_seasons, num\_episodes):

super().\_\_init\_\_(name, actors)

self.\_num\_seasons = num\_seasons

self.\_num\_episodes = num\_episodes

Продолжение Листинга А.1.

def \_\_repr\_\_(self):

return f"{\_\_class\_\_.\_\_name\_\_}('{self.\_name}', {self.\_actors}, {self.\_num\_seasons}, {self.\_num\_episodes})"

class FeatureFilm(Film):

def \_\_init\_\_(self, name, actors, length):

super().\_\_init\_\_(name, actors)

self.\_length = length

def \_\_repr\_\_(self):

return f"{\_\_class\_\_.\_\_name\_\_}('{self.\_name}', {self.\_actors}, {self.\_length})"

class Actor(MovieIndustry):

def \_\_init\_\_(self, name):

self.\_name = name

def \_\_str\_\_(self):

return self.\_name

def \_\_repr\_\_(self):

return f"{\_\_class\_\_.\_\_name\_\_}('{self.\_name}')"

def \_\_eq\_\_(self, other):

if type(other) is \_\_class\_\_:

return other.\_name == self.\_name

elif type(other) is str:

return other == self.\_name

return NotImplemented

def \_\_contains\_\_(self, obj):

return obj in self.\_name

def startswith(self, value):

return self.\_name.startswith(value)

def endswith(self, value):

return self.\_name.endswith(value)

class FilmDirector(MovieIndustry):

def \_\_init\_\_(self, name, films):

self.\_name = name

self.\_films = list(films)

def \_\_str\_\_(self):

return self.\_name

def \_\_repr\_\_(self):

return f"{\_\_class\_\_.\_\_name\_\_}('{self.\_name}', {self.\_films})"

def \_\_eq\_\_(self, other):

if type(other) is \_\_class\_\_:

return other.\_name == self.\_name

elif type(other) is str:

return other == self.\_name

return NotImplemented

def \_\_contains\_\_(self, obj):

Продолжение Листинга А.1.

return obj in self.\_name

def startswith(self, value):

return self.\_name.startswith(value)

def endswith(self, value):

return self.\_name.endswith(value)

def has\_movie(self, value):

return value in self.\_films

class Queries:

def \_\_init\_\_(self, cls, slot, function, value):

self.\_cls = cls

self.\_slot = slot

self.\_value = value

match function:

case 'contains':

self.\_func = \_\_class\_\_.contains

case 'does not contain':

self.\_func = \_\_class\_\_.does\_not\_contains

case 'is':

self.\_func = \_\_class\_\_.equal

case 'is not':

self.\_func = \_\_class\_\_.not\_equal

case 'begins with':

self.\_func = \_\_class\_\_.startswith

case 'ends with':

self.\_func = \_\_class\_\_.endswith

case 'is greater then':

self.\_func = function

case 'is less then':

self.\_func = function

self.\_result = None

def find(self):

match self.\_cls:

case 'Фильм':

if self.\_slot == 'Название':

found\_films = list(filter(lambda object: self.\_func(self, object), feature\_films + serial\_films))

elif self.\_func == \_\_class\_\_.contains:

found\_films = list(filter(lambda object: object.has\_actor(self.\_value), feature\_films + serial\_films))

else:

found\_films = list(filter(lambda object: not object.has\_actor(self.\_value), feature\_films + serial\_films))

for film in found\_films:

print('•', film)

for film\_director in film\_directors:

if film\_director.has\_movie(film):

print(' •', film\_director)

case 'Многосерийный фильм':

if self.\_slot == 'Актёры' and self.\_func == \_\_class\_\_.contains:

found\_films = list(filter(lambda object: object.has\_actor(self.\_value), serial\_films))

elif self.\_slot == 'Актёры' and self.\_func == \_\_class\_\_.does\_not\_contains:

found\_films = list(filter(lambda object: not object.has\_actor(self.\_value), serial\_films))

Продолжение Листинга А.1.

elif self.\_slot == 'Количество сезонов' and self.\_func == \_\_class\_\_.equal:

found\_films = list(filter(lambda object: object.\_num\_seasons == int(self.\_value), serial\_films))

elif self.\_slot == 'Количество сезонов' and self.\_func == 'is greater then':

found\_films = list(filter(lambda object: object.\_num\_seasons > int(self.\_value), serial\_films))

elif self.\_slot == 'Количество сезонов' and self.\_func == 'is less then':

found\_films = list(filter(lambda object: object.\_num\_seasons < int(self.\_value), serial\_films))

elif self.\_slot == 'Количество эпизодов' and self.\_func == \_\_class\_\_.equal:

found\_films = list(filter(lambda object: object.\_num\_episodes == int(self.\_value), serial\_films))

elif self.\_slot == 'Количество эпизодов' and self.\_func == 'is greater then':

found\_films = list(filter(lambda object: object.\_num\_episodes > int(self.\_value), serial\_films))

elif self.\_slot == 'Количество эпизодов' and self.\_func == 'is less then':

found\_films = list(filter(lambda object: object.\_num\_episodes < int(self.\_value), serial\_films))

else:

found\_films = list(filter(lambda object: self.\_func(self, object), serial\_films))

for film in found\_films:

print('•', film)

for film\_director in film\_directors:

if film\_director.has\_movie(film):

print(' •', film\_director)

case 'Полнометражный фильм':

if self.\_slot == 'Актёры' and self.\_func == \_\_class\_\_.contains:

found\_films = list(filter(lambda object: object.has\_actor(self.\_value), feature\_films))

elif self.\_slot == 'Актёры' and self.\_func == \_\_class\_\_.does\_not\_contains:

found\_films = list(filter(lambda object: not object.has\_actor(self.\_value), feature\_films))

elif self.\_slot == 'Длительность' and self.\_func == \_\_class\_\_.equal:

found\_films = list(filter(lambda object: object.\_length == int(self.\_value), feature\_films))

elif self.\_slot == 'Длительность' and self.\_func == 'is greater then':

found\_films = list(filter(lambda object: object.\_length > int(self.\_value), feature\_films))

elif self.\_slot == 'Длительность' and self.\_func == 'is less then':

found\_films = list(filter(lambda object: object.\_length < int(self.\_value), feature\_films))

else:

found\_films = list(filter(lambda object: self.\_func(self, object), feature\_films))

for film in found\_films:

print('•', film)

for film\_director in film\_directors:

if film\_director.has\_movie(film):

print(' •', film\_director)

case 'Актёр':

Продолжение Листинга А.1.

found\_actors = list(

filter(lambda object: self.\_func(self, object), actors))

for actor in found\_actors:

print('•', actor)

for film in serial\_films + feature\_films:

if film.has\_actor(actor):

print(' •', film)

for film\_director in film\_directors:

if film\_director.has\_movie(film):

print('\t•', film\_director)

case 'Режиссёр':

if self.\_slot == 'ФИО':

self.\_result = list(

filter(lambda object: self.\_func(self, object), film\_directors))

elif self.\_func == \_\_class\_\_.contains:

self.\_result = list(

filter(lambda object: object.has\_movie(self.\_value), film\_directors))

else:

self.\_result = list(

filter(lambda object: not object.has\_movie(self.\_value), film\_directors))

for item in self.\_result:

print('•', str(item))

def contains(self, object):

return self.\_value in object

def does\_not\_contains(self, object):

return self.\_value not in object

def equal(self, object):

return self.\_value == object

def not\_equal(self, object):

return self.\_value != object

def startswith(self, object):

return object.startswith(self.\_value)

def endswith(self, object):

return object.endswith(self.\_value)

def make\_query():

'''Функция для написания запроса'''

print("\033[4mВыберите класс:\033[0m")

for num, cls in enumerate(QUERY, 1):

print(num, '-', '\033[93m' + cls + '\033[0m')

cls\_num = input()

if cls\_num not in map(str, range(1, len(QUERY) + 1)):

raise TypeError('Некорректный номер класса')

cls = list(QUERY.keys())[int(cls\_num) - 1]

print("\033[4mВыберите слот:\033[0m")

for num, slot in enumerate(QUERY[cls], 1):

print(num, '-', '\033[94m' + slot + '\033[0m')

slot\_num = input()

if slot\_num not in map(str, range(1, len(QUERY[cls]) + 1)):

raise TypeError('Некорректный номер слота')

Продолжение Листинга А.1.

slot = QUERY[cls][int(slot\_num) - 1]

print("\033[4mВыберите функцию запроса:\033[0m")

for num, func in enumerate(FUNCTION[slot], 1):

print(num, '-', '\033[93m' + func + '\033[0m')

func\_num = input()

if func\_num not in map(str, range(1, len(FUNCTION[slot]) + 1)):

raise TypeError('Некорректный номер функции')

func = FUNCTION[slot][int(func\_num) - 1]

value = input("\033[4mВведите значение запроса:\033[0m ")

query\_obj = Queries(cls, slot, func, value)

query\_obj.find()

actors = [Actor('Bryan Cranston'), Actor('Anna Gunn'),

Actor('Aaron Paul'), Actor('Dean Norris'),

Actor('Betsy Brandt'), Actor('RJ Mitte'),

Actor('Bob Odenkirk'), Actor('Giancarlo Esposito'),

Actor('Jonathan Banks'), Actor('Steven Michael Quezada'),

Actor('Cillian Murphy'), Actor('Emily Blunt'),

Actor('Matt Damon'), Actor('Robert Downey Jr.'),

Actor('Florence Pugh'), Actor('Josh Hartnett'),

Actor('David Krumholtz'), Actor('Benny Safdie'),

Actor('Alden Ehrenreich'), Actor('Kenneth Branagh'),

Actor('Leonardo DiCaprio'), Actor('Jonah Hill'),

Actor('Margot Robbie'), Actor('Kyle Chandler'),

Actor('Rob Reiner'), Actor('P.J. Byrne'),

Actor('Jon Bernthal'), Actor('Cristin Milioti'),

Actor('Jean Dujardin'), Actor('Matthew McConaughey'),

Actor('Ryan Gosling'), Actor('America Ferrera'),

Actor('Ariana Greenblatt'), Actor('Kate McKinnon'),

Actor('Issa Rae'), Actor('Will Ferrell'),

Actor('Michael Cera'), Actor('Simu Liu'),

Actor('Alexandra Shipp'), Actor('Joseph Gordon-Levitt'),

Actor('Elliot Page'), Actor('Tom Hardy'),

Actor('Ken Watanabe'), Actor('Dileep Rao'),

Actor('Tom Berenger'), Actor('Marion Cotillard'),

Actor('Pete Postlethwaite'), Actor('Paul Anderson'),

Actor('Sophie Rundle'), Actor('Helen McCrory'),

Actor('Ned Dennehy'), Actor('Finn Cole'),

Actor("Natasha O'Keeffe"), Actor('Ian Peck'),

Actor('Harry Kirton'), Actor('Packy Lee')]

serial\_films = [SerialFilm('Breaking Bad', actors[:10], 5, 62),

SerialFilm('Peaky Blinders', [actors[10]] + actors[47:56], 6, 36)]

feature\_films = [FeatureFilm('Oppenheimer', actors[10:20], 180),

FeatureFilm('The Wolf of Wall Street', actors[20:30], 172),

FeatureFilm('Barbie', [actors[22]] + actors[30:39], 104),

FeatureFilm('Inception', [actors[20]] + [actors[10]] + actors[39:47], 148)]

film\_directors = [FilmDirector('Michelle MacLaren', [serial\_films[0]]),

FilmDirector('Adam Bernstein', [serial\_films[0]]),

FilmDirector('Vince Gilligan', [serial\_films[0]]),

FilmDirector('Christopher Nolan', [

feature\_films[0], feature\_films[3]]),

FilmDirector('Martin Scorsese', [feature\_films[1]]),

Окончание Листинга А.1.

FilmDirector('Greta Gerwig', [feature\_films[2]]),

FilmDirector('Anthony Byrne', [serial\_films[1]]),

FilmDirector('Colm McCarthy', [serial\_films[1]]),

FilmDirector('Tim Mielants', [serial\_films[1]]),

FilmDirector('David Caffrey', [serial\_films[1]]),

FilmDirector('Otto Bathurst', [serial\_films[1]]),

FilmDirector('Tom Harper', [serial\_films[1]])]

make\_query()