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

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.

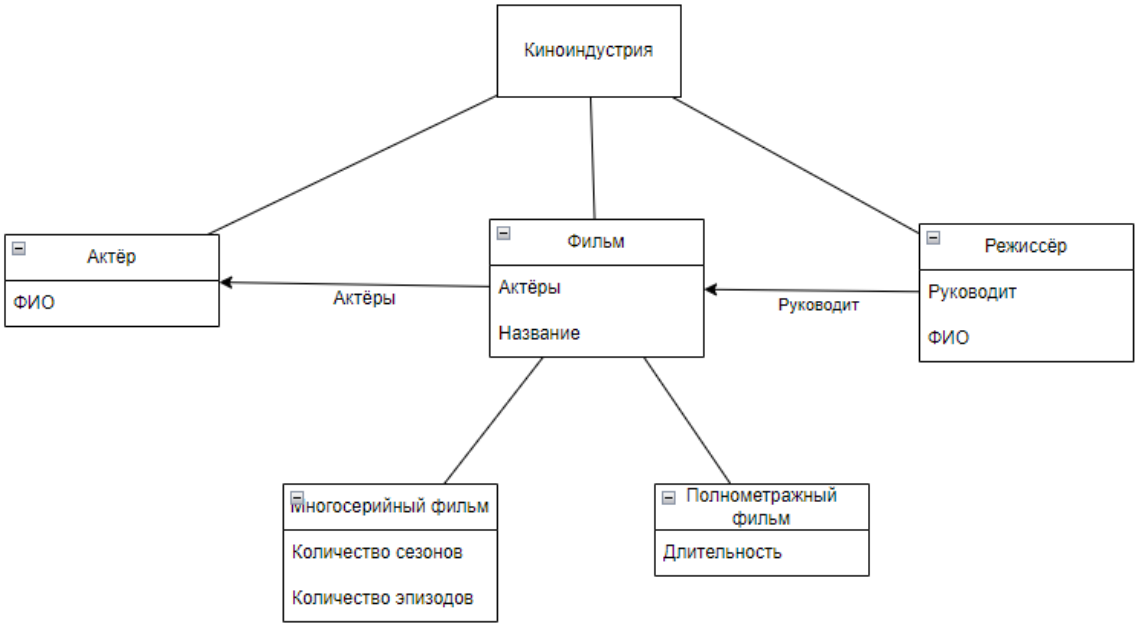

Рисунок 1.4.1 – Иерархия классов онтологии

1.5 Реализация в Protégé

Иерархия онтологии была перенесена в Protégé (Рисунок 1.5.1).

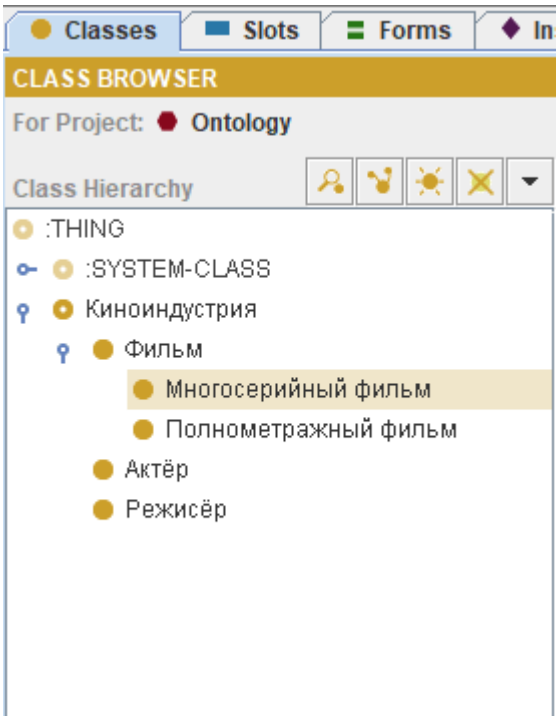

Рисунок 1.5.1 – Иерархия классов в Protégé

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

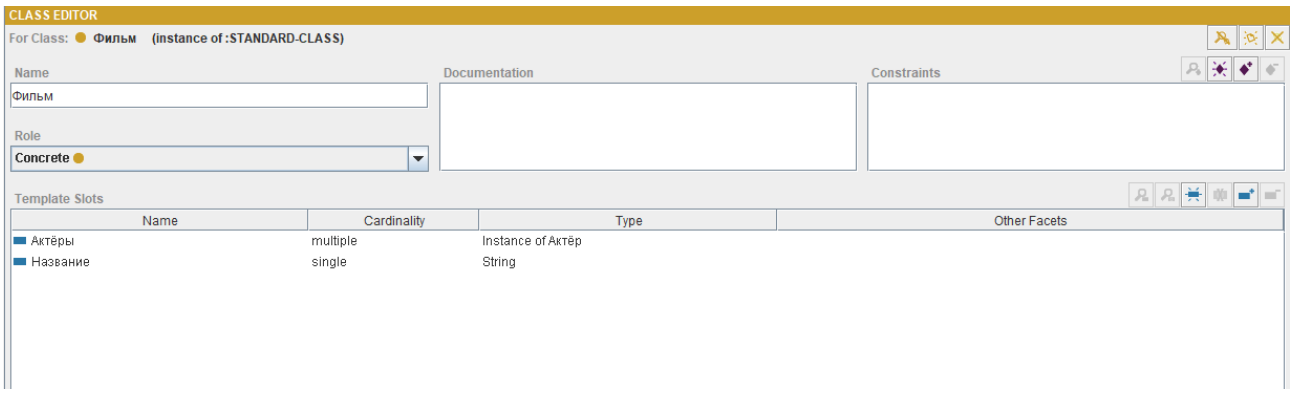

Рисунок 1.5.2 – Слоты для класса «Фильм»

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

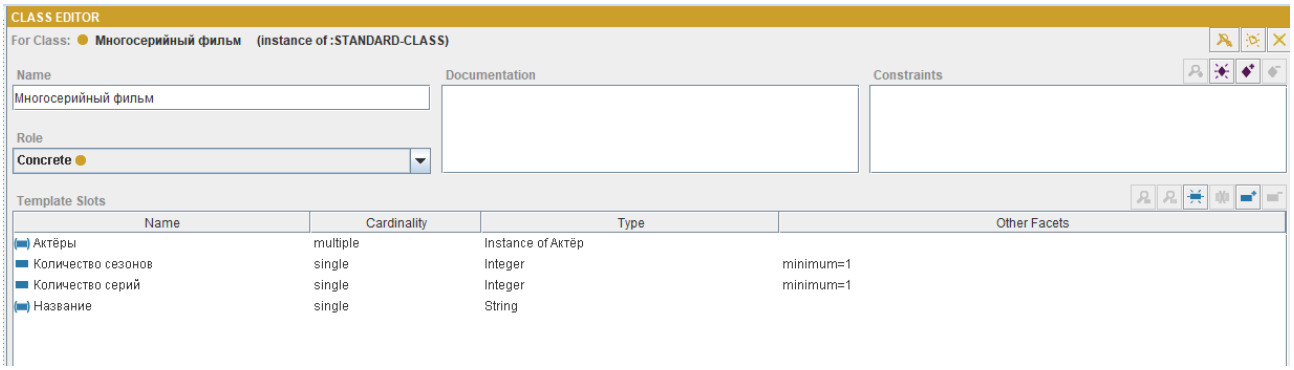

Рисунок 1.5.3 - Слоты для класса «Многосерийный фильм»

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

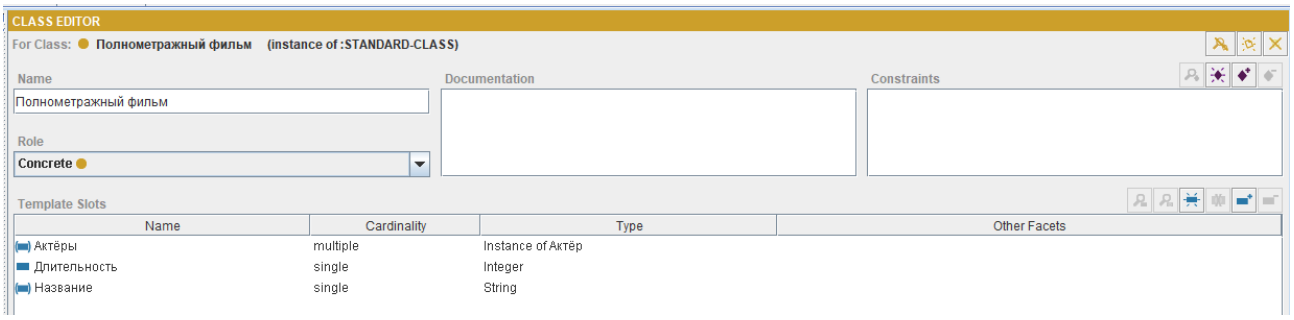

Рисунок 1.5.4 - Слоты для класса «Полнометражный фильм»

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).

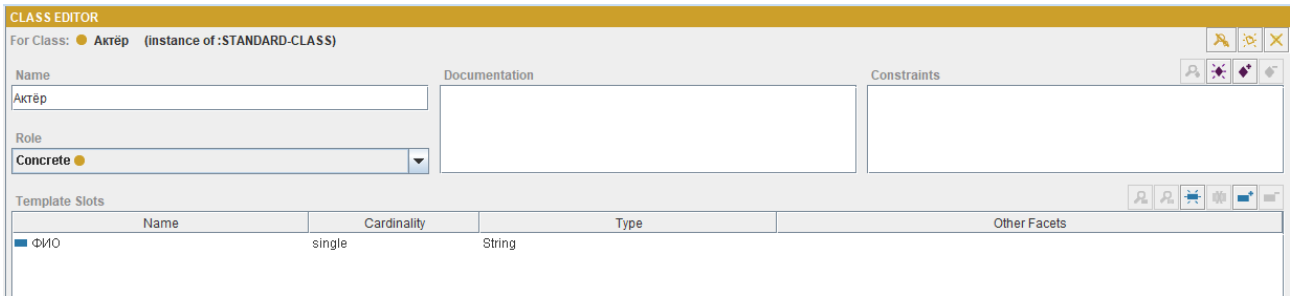

Рисунок 1.5.5 – Слоты для класса «Актёр»

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

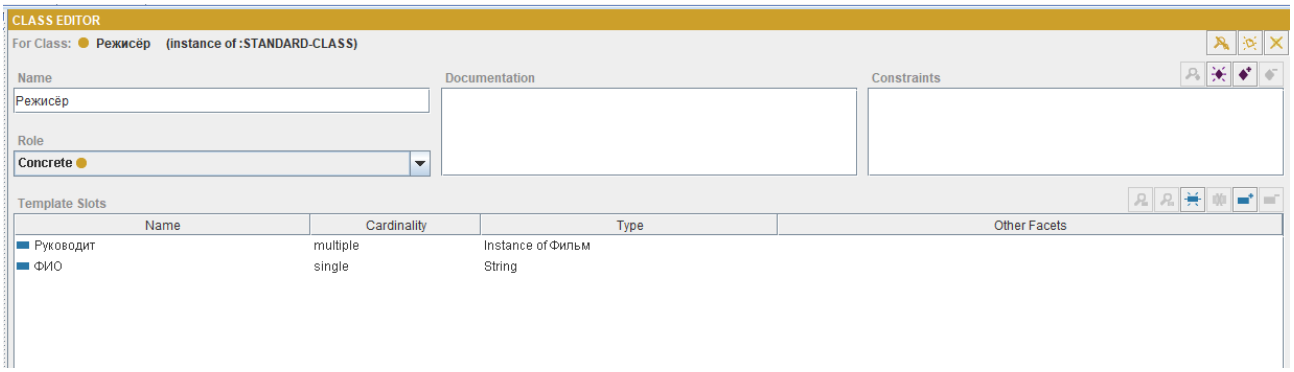

Рисунок 1.5.6 – Слоты для класса «Режиссёр»

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).

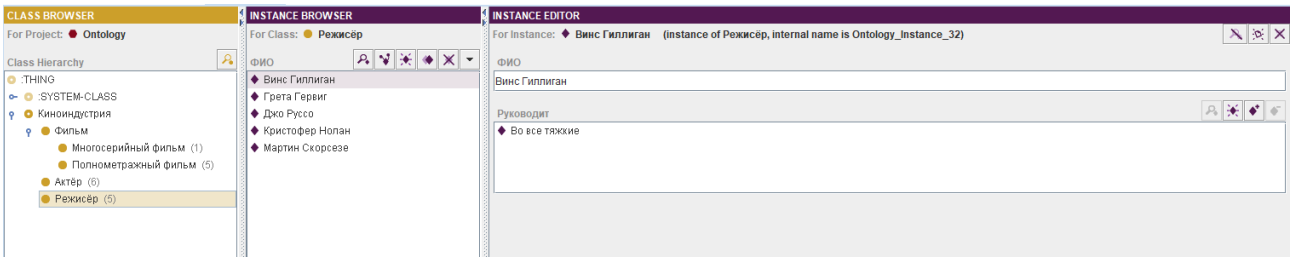

Рисунок 1.5.7 – Созданные объекты классов

Для каждого объекта заполнены соответствующие слоты (Рисунок 1.5.8).

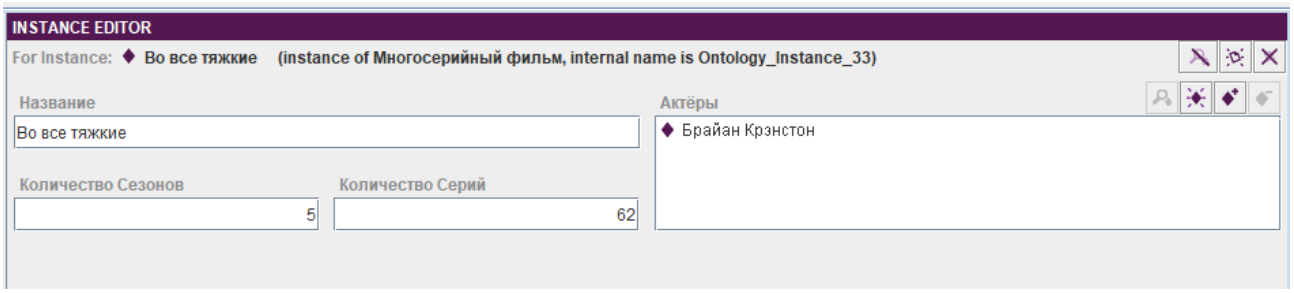

Рисунок 1.5.8 – Пример заполнения слотов для созданного экземпляра класса

Произведено несколько запросов (Рисунок 1.5.9 – 1.5.11).

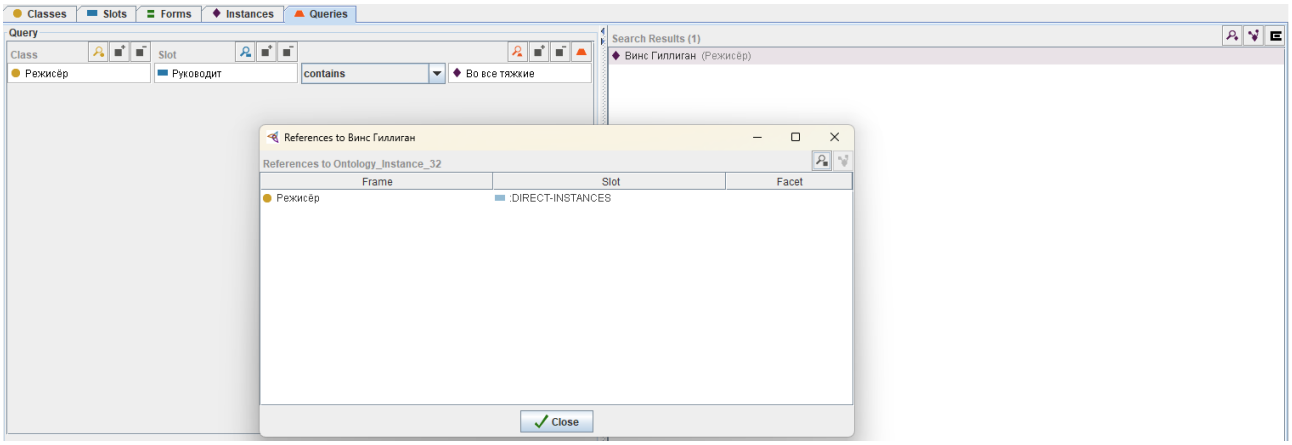

Рисунок 1.5.9 – Запрос от класса «Режиссёр»

Запрос позволяет узнать режиссёров, которые руководят выбранным фильмом.

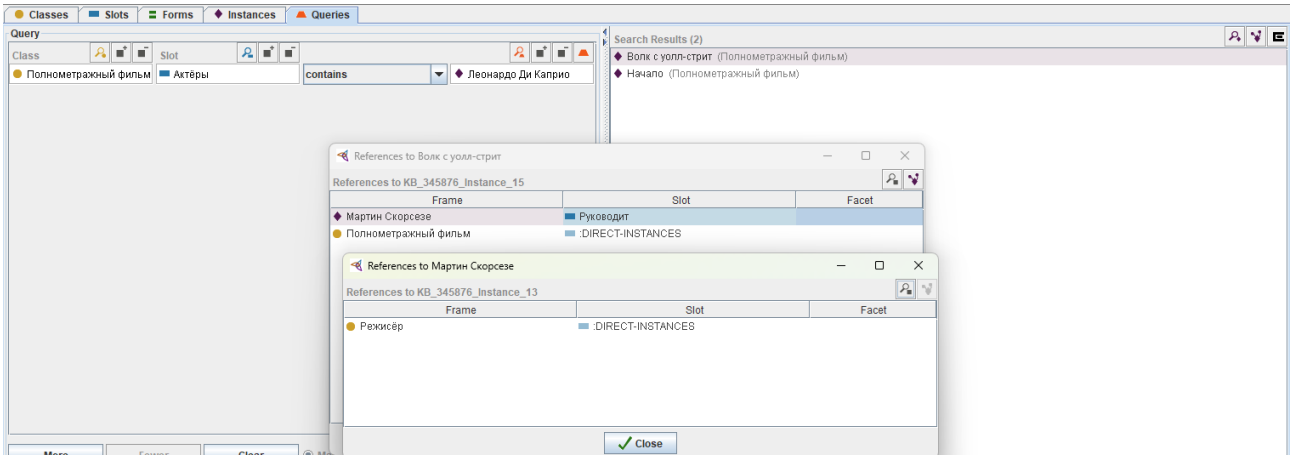

Рисунок 1.5.10 – Запрос от класса «Полнометражный фильм»

Запрос показывает сущности актёров, которые снимаются в выбранном фильме.

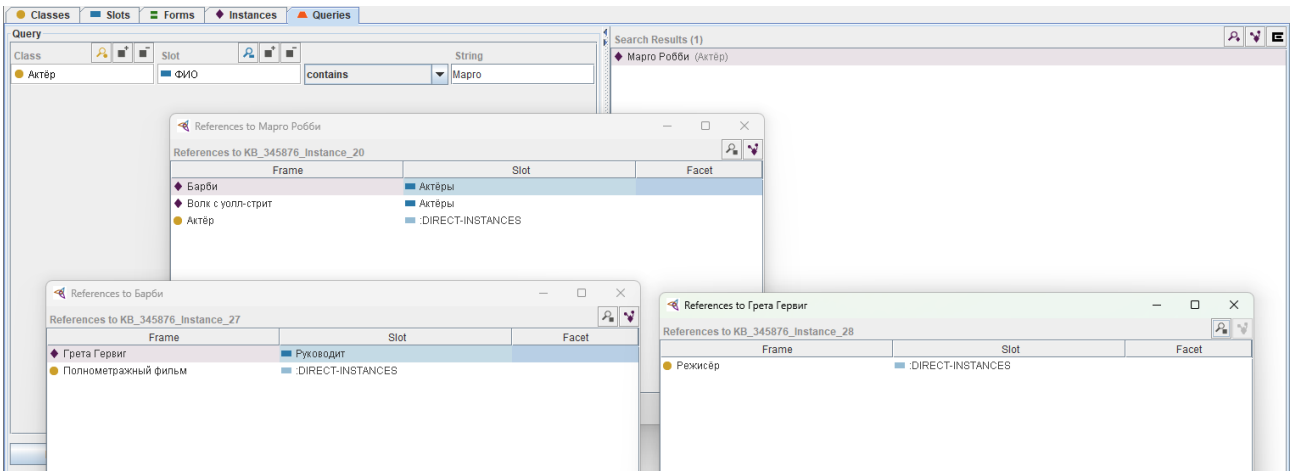

Рисунок 1.5.11 – Запрос от класса «Актёр»

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).

```
            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', [···
                  FilmDirector('Martin Scorsese', [feature_films[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]])]
```

Рисунок 1.6.1 – Созданные объекты классов

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

```
Выберите класс:
1 - Фильм
2 - Многосерийный фильм
3 - Полнометражный фильм
4 - Актёр
5 - Режиссёр
1
Выберите слот:
1 - Актёры
2 - Название
1
Выберите функцию запроса:
1 - contains
2 - does not contain
1
Введите значение запроса: Margot Robbie
• The Wolf of Wall Street
    • Martin Scorsese
• Barbie
    • Greta Gerwig
```

Рисунок 1.6.2 – Запрос от класса «Фильм»

```
Выберите класс:
1 - Фильм
2 - Многосерийный фильм
3 - Полнометражный фильм
4 - Актёр
5 - Режиссёр
2
Выберите слот:
1 - Актёры
2 - Название
3 - Количество сезонов
4 - Количество эпизодов
3
Выберите функцию запроса:
1 - is
2 - is greater then
3 - is less then
2
Введите значение запроса: 5
• Peaky Blinders
    • Anthony Byrne
    • Colm McCarthy
    • Tim Mielants
    • David Caffrey
    • Otto Bathurst
    • Tom Harper
PS C:\python_projects>
```

Рисунок 1.6.3 – Запрос от класса «Многосерийный фильм»

```
Выберите класс:
1 - Фильм
2 - Многосерийный фильм
3 - Полнометражный фильм
4 - Актёр
5 - Режиссёр
3
Выберите слот:
1 - Актёры
2 - Название
3 - Длительность
2
Выберите функцию запроса:
1 - contains
2 - does not contain
3 - is
4 - is not
5 - begins with
6 - ends with
1
Введите значение запроса: i
• Oppenheimer
    • Christopher Nolan
• Barbie
    • Greta Gerwig
• Inception
    • Christopher Nolan
PS C:\python_projects>
```

Рисунок 1.6.4 – Запрос от класса «Полнометражный фильм»

```
Выберите класс:
1 - Фильм
2 - Многосерийный фильм
3 - Полнометражный фильм
4 - Актёр
5 - Режиссёр
4
Выберите слот:
1 - ФИО
1
Выберите функцию запроса:
1 - contains
2 - does not contain
3 - is
4 - is not
5 - begins with
6 - ends with
1
Введите значение запроса: Mar
• Margot Robbie
    • The Wolf of Wall Street
        • Martin Scorsese
    • Barbie
        • Greta Gerwig
• Marion Cotillard
    • Inception
        • Christopher Nolan
PS C:\python_projects>
```

Рисунок 1.6.5 – Запрос от класса «Актёр»

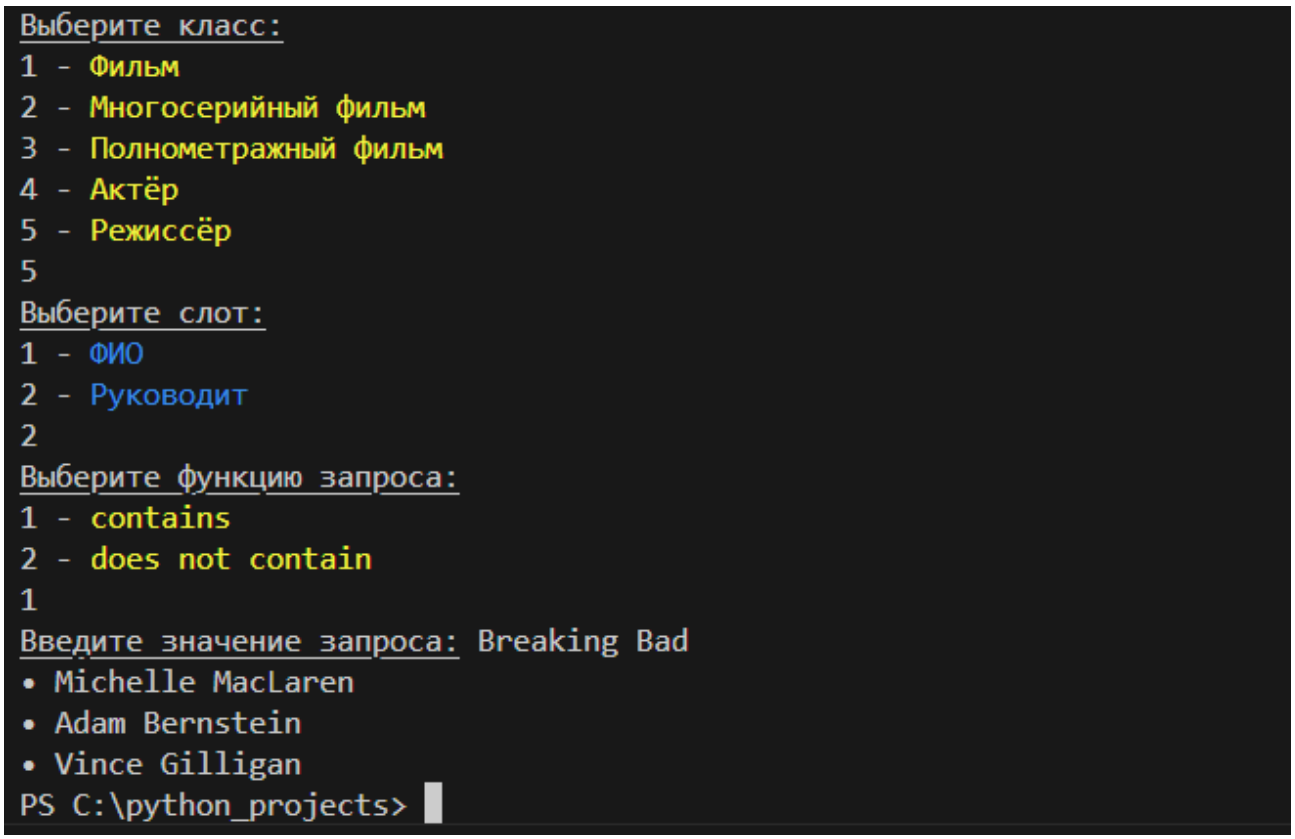

Рисунок 1.6.6 – Запрос от класса «Режиссёр»

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 с.

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()
```