

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

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА №1**

**«Онтология»**

**по дисциплине**

**«Системный анализ данных СППР»**

Студент группы: ИКБО-04-22 Егоров Л.А. *(Ф. И.О.студента)*

Преподаватель \_\_Железняк Л.М.\_\_ *(Ф.И.О. преподавателя)*

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

Москва 2024

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc177928544)

[1 ОНТОЛОГИЯ 4](#_Toc177928545)

[1.1 Постановка задачи 4](#_Toc177928546)

[1.2 Описание онтологии 4](#_Toc177928547)

[1.3 Построение онтологии в Protégé 5](#_Toc177928548)

[1.4 Выполнение запросов в Protege 8](#_Toc177928549)

[1.5 Результаты выполнения программного кода 9](#_Toc177928550)

[ЗАКЛЮЧЕНИЕ 11](#_Toc177928551)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИК 12](#_Toc177928552)

[ПРИЛОЖЕНИЯ 13](#_Toc177928553)

# ВВЕДЕНИЕ

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, и т.п.);
* 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.

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

Существует много видов онтологий, однако одним из самых широко применяемых видов являются онтологии предметных областей, содержащие понятия определённой области знаний или входящих в неё областей.

# 1 онтология

## 1.1 Постановка задачи

Необходимо разработать онтологию выбранной предметной области – «Музыкальная индустрия». Данная предметная область выбрана из-за личного интереса к теме.

## 1.2 Описание онтологии

Основным продуктом звукозаписывающих компаний являются музыкальные записи – наиболее распространёнными из них являются песни и альбомы, являющиеся сборниками песен. Авторами альбомов выступают либо группы, либо отдельные музыканты, и обе эти категории также связаны между собой – группы состоят из музыкантов.

На основе этого описания можно составить онтологию, состоящую из следующих классов:

* «Музыкальная индустрия» — общий базовый класс для всех классов;
* «Музыкальная запись» — базовый класс для разных видов музыкальных записей, содержит общий слот «Название»;
* «Альбом» — класс для описания альбомов, содержит слоты «Год выхода» и «Исполнитель», ссылающийся на экземпляр класса «Исполнитель»;
* «Песня» — класс для описания песен, содержит слот «Входит в альбом», ссылающийся на экземпляр класса «Альбом»;
* «Исполнитель» — базовый класс для всех видов исполнителей, содержит общие слоты «Имя» и «Страна происхождения»;
* «Группа» — класс для описания групп, не содержит своих слотов;
* «Музыкант» — класс для описания музыкантов, содержит слот «Входит в группу», ссылающийся на экземпляр класса «Группа»

Данное описание использовано для построения графической схемы онтологии (Рисунок 1.2.1).

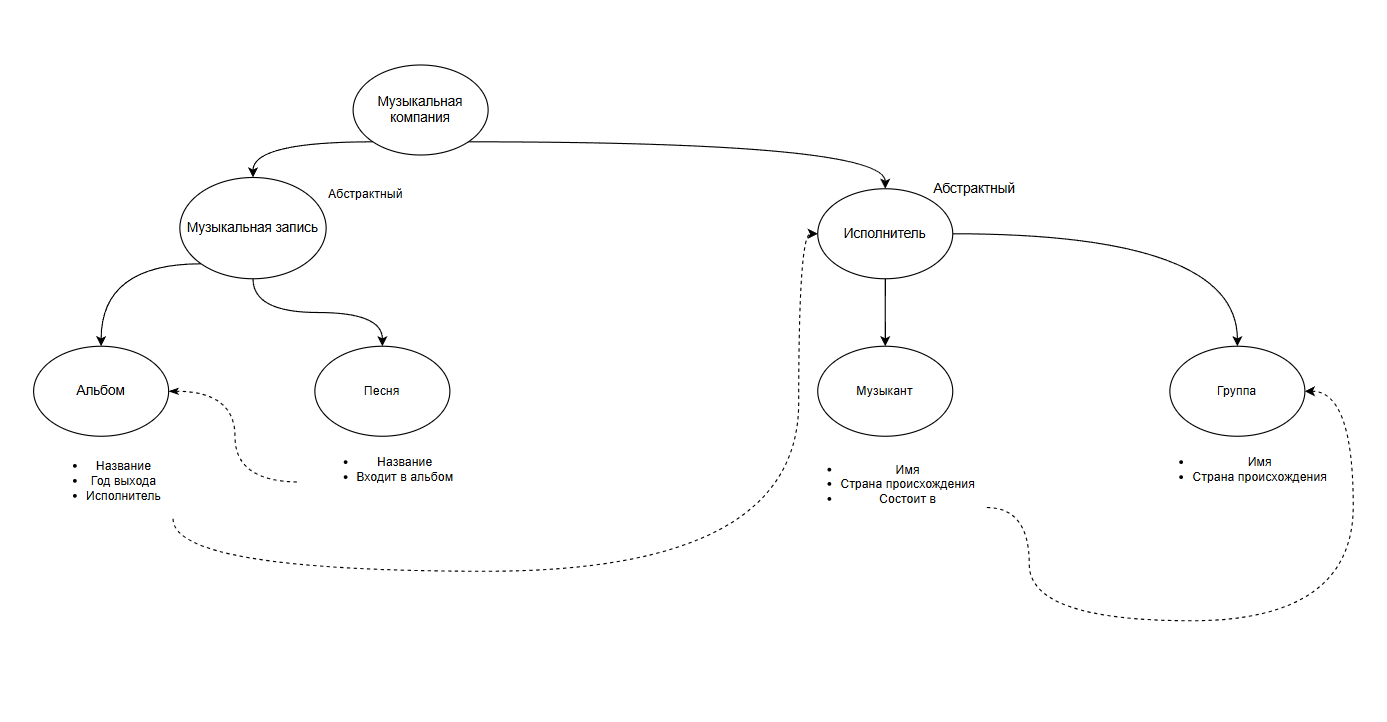


Рисунок 1.2.1 – Схема онтологии «Музыкальная индустрия»

## 1.3 Построение онтологии в Protégé

Для подробного изучения составленной онтологии использован инструмент для построения, редактирования онтологий и работы с ними Protégé. Сначала созданы классы (Рисунок 1.3.1), а затем в них описаны слоты (Рисунок 1.3.2-1.3.5).



Рисунок 1.3.1 – Составленная иерархия классов

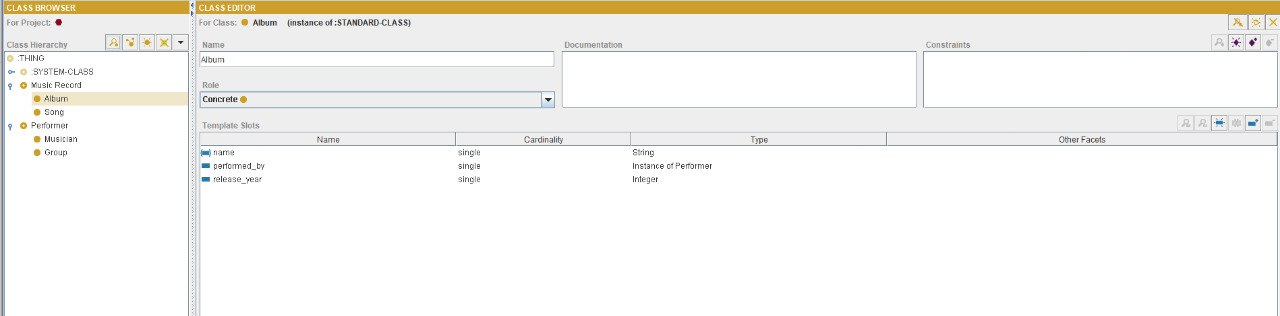


Рисунок 1.3.2 – Слоты класса «Альбом»

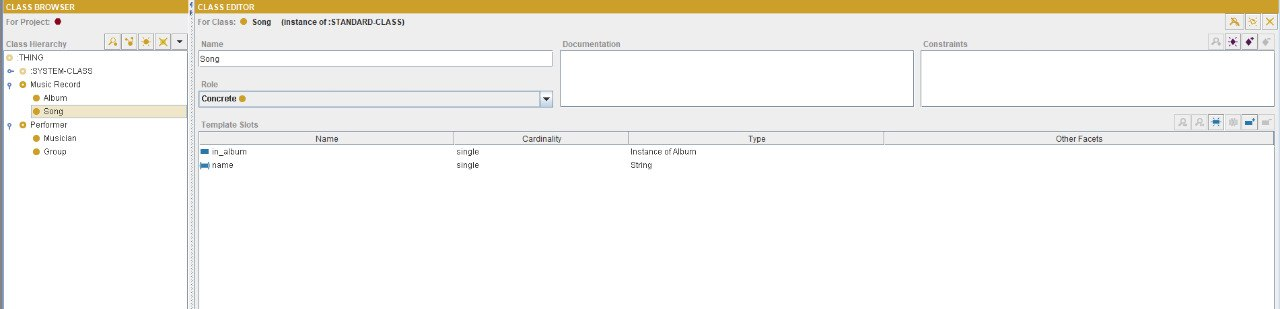


Рисунок 1.3.3 – Слоты класса «Песня»

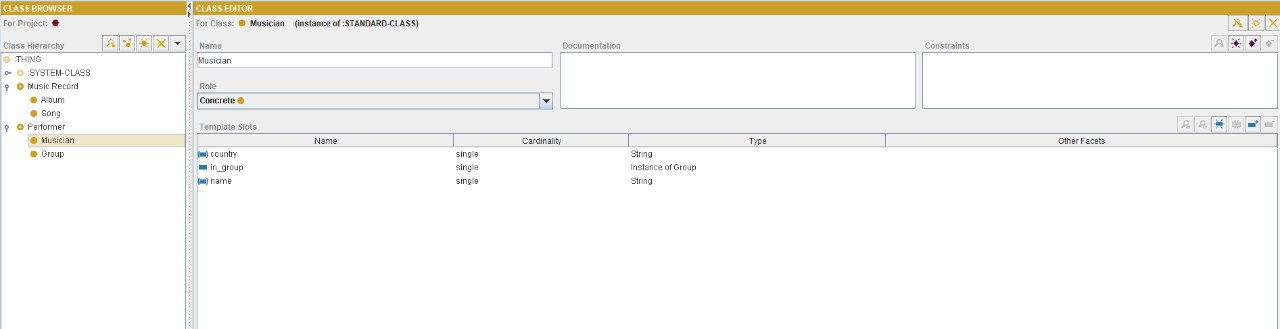


Рисунок 1.3.4 – Слоты класса «Музыкант»

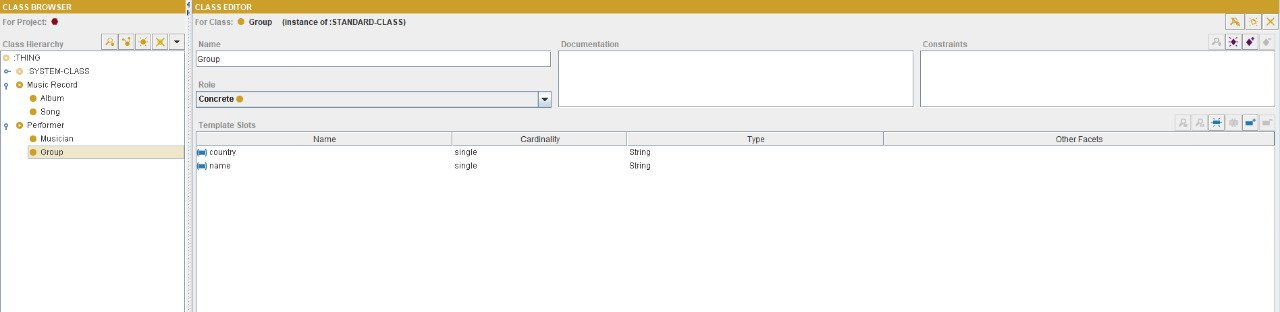


Рисунок 1.3.5 – Слоты класса «Группа»

После составления и описания классов созданы экземпляры каждого из классов (Рисунок 1.3.6-1.3.9).

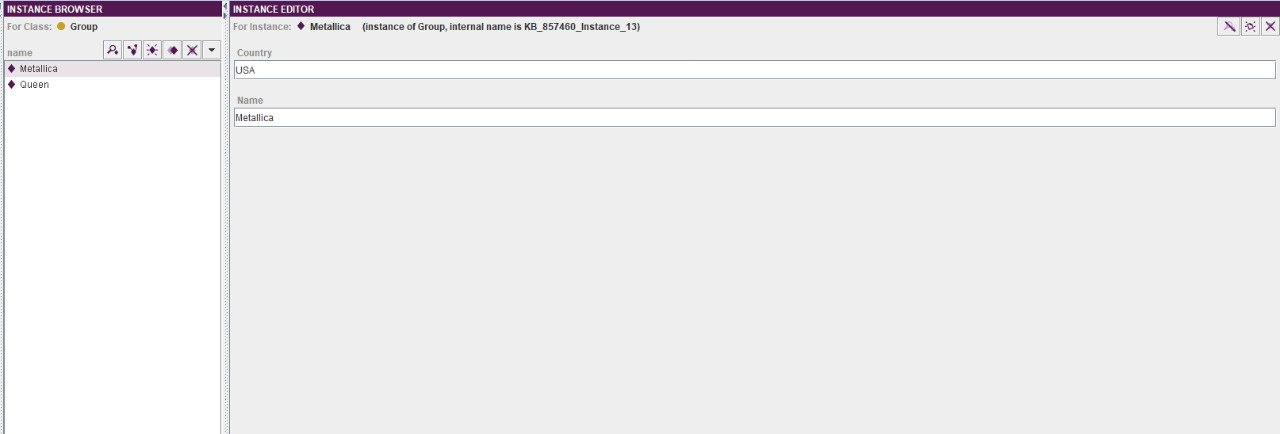


Рисунок 1.3.6 – Экземпляры класса «Группа»

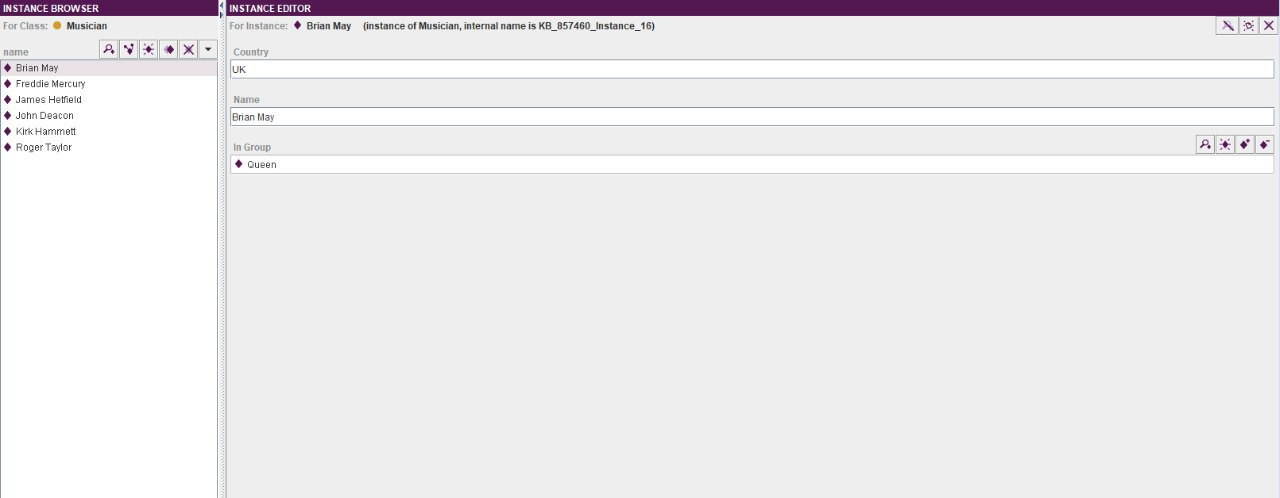


Рисунок 1.3.7 – Экземпляры класса «Музыкант»

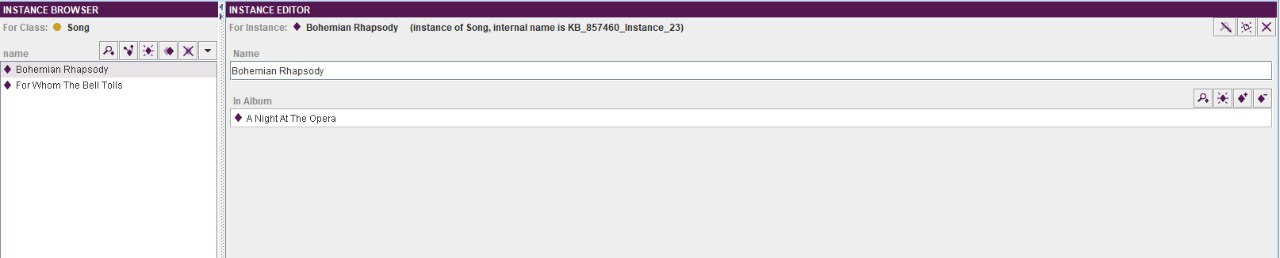


Рисунок 1.3.8 – Экземпляры класса «Песня»

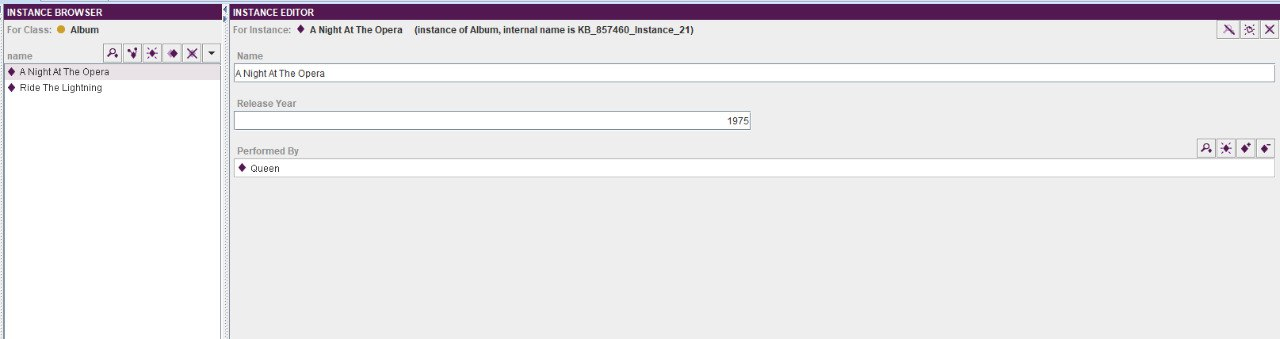


Рисунок 1.3.9 – Экземпляры класса «Альбом»

## 1.4 Выполнение запросов в Protege

Программа Protégé позволяет составлять запросы на получение объектов по определённым условиям, а также вытаскивать связанные объекты для уже полученных объектов. Проделаны обычные запросы на получение экземпляров (Рисунок 1.4.1), а также сделаны цепные запросы на получение связанных объектов (Рисунок 1.4.3).

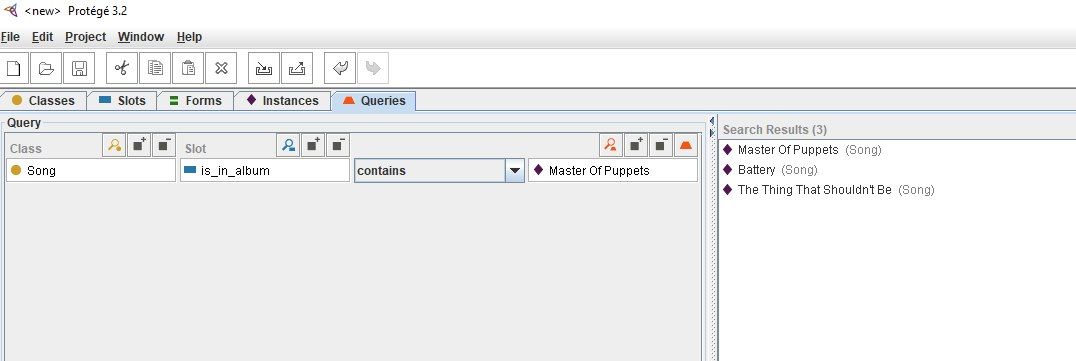


Рисунок 1.4.1 – Одинарный запрос на получение песен из альбома

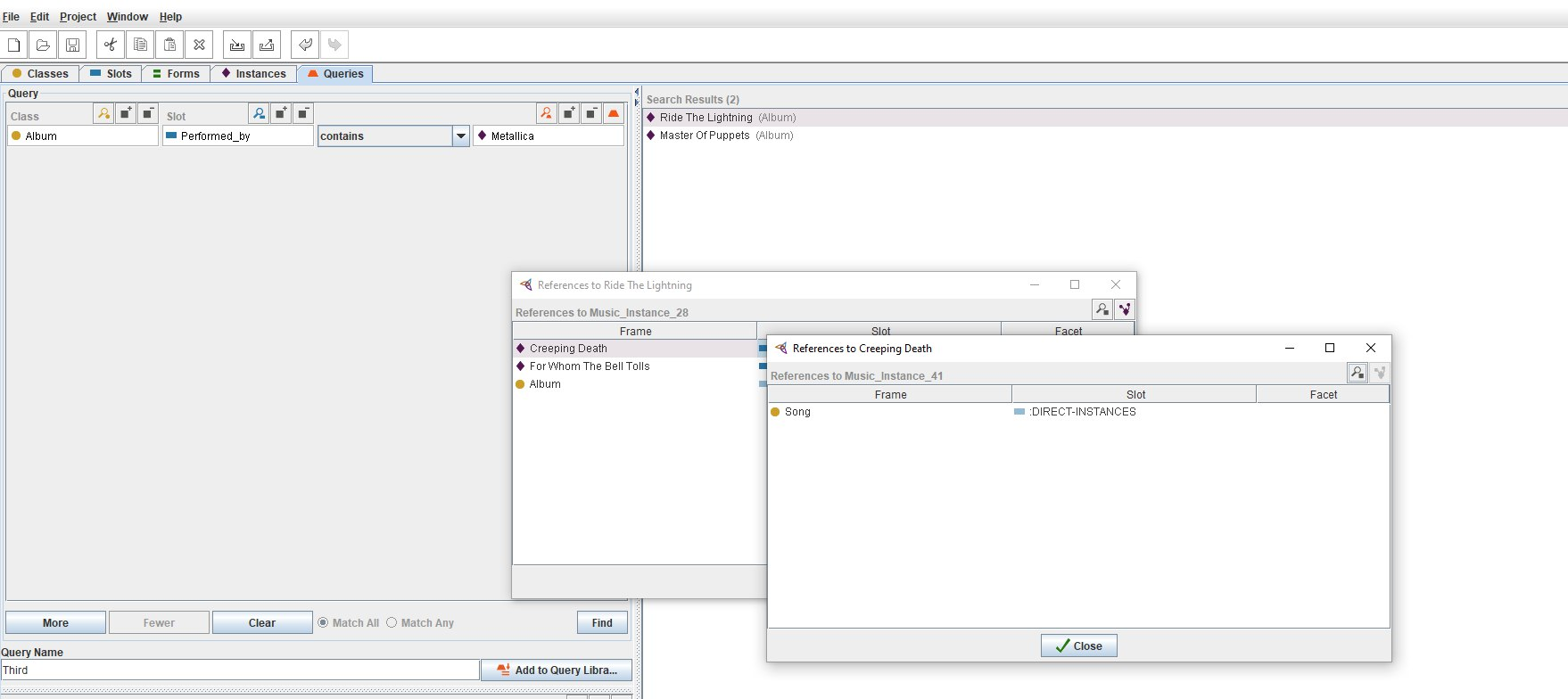


Рисунок 1.4.2 – Цепной запрос на получение песен, написанных одной группой

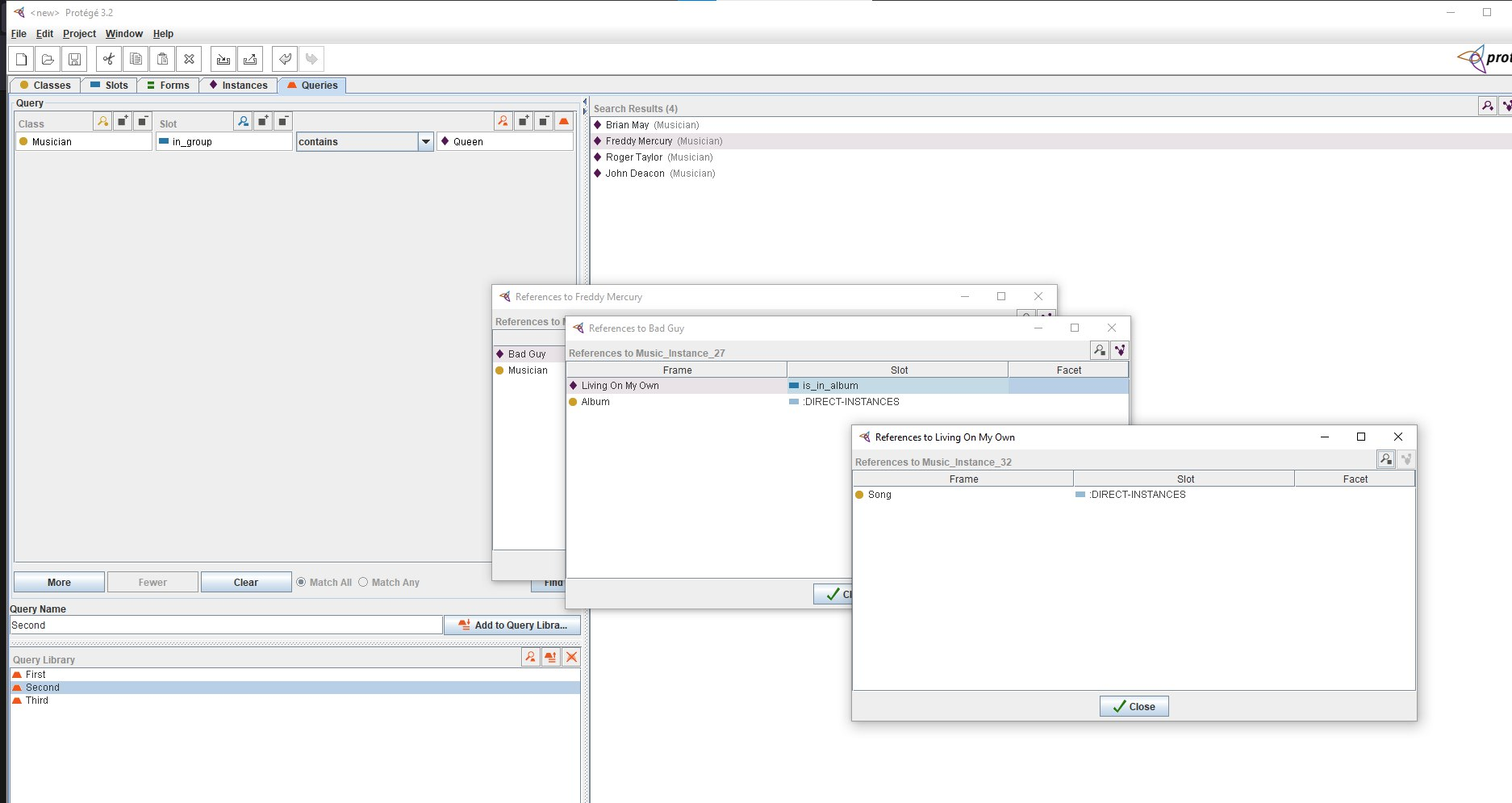


Рисунок 1.4.3 – Цепной запрос на получение песен, написанных одним музыкантом

## 1.5 Результаты выполнения программного кода

Для работы с онтологиями написана программа на языке Python, которая запускается в консоли и поддерживает выполнение запросов на получение экземпляров. Работы программы продемонстрирована на Рисунках 1.5.1-1.5.2.

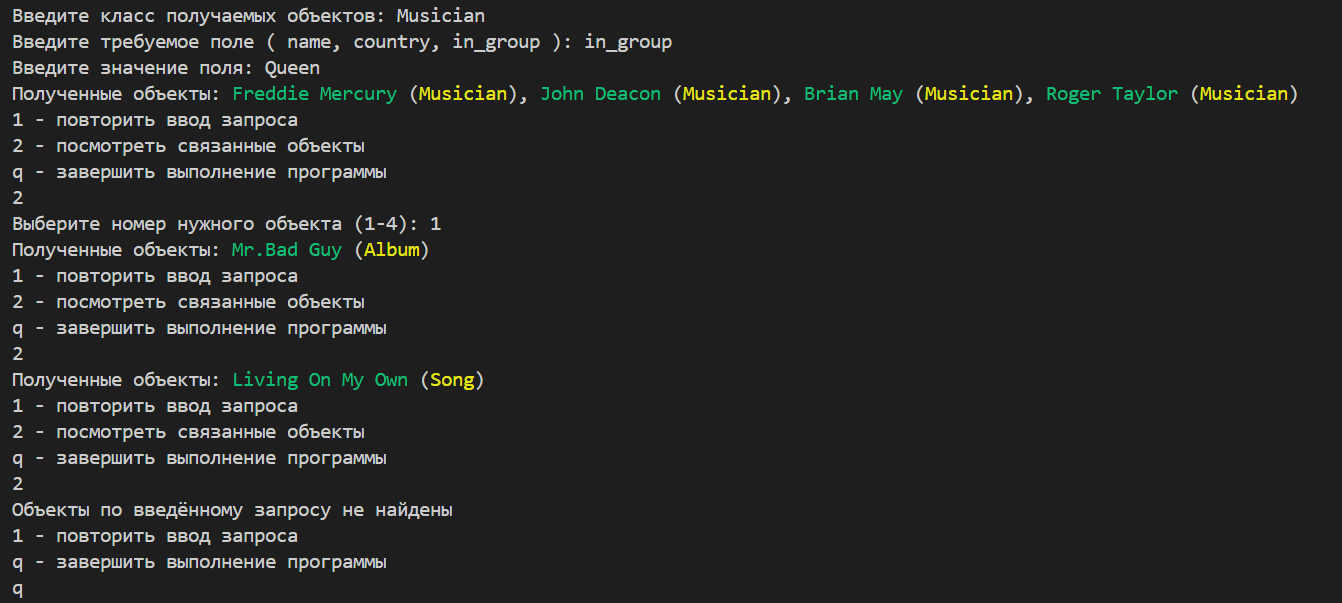


Рисунок 1.5.1 – Результат выполнения программы



Рисунок 1.5.2 – Результат выполнения программы

# 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. Сорокин А.Б. Введение в роевой интеллект: теория, расчеты и приложения [Электронный ресурс]: Учебно-методическое пособие — М.: Московский технологический университет (МИРЭА), 2019.
2. Лапшин В. А. Онтологии в компьютерных системах. — М.: Научный мир, 2010.
3. Карпенко А. П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой: учебное пособие / А. П. Карпенко. – Москва: Издательство МГТУ им. Н. Э. Баумана, 2014. – 448 c.
4. Noy, Natalya F.; [McGuinness, Deborah L.](https://en.wikipedia.org/wiki/Deborah_McGuinness) (March 2001). ["Ontology Development 101: A Guide to Creating Your First Ontology"](https://web.archive.org/web/20100714172301/http:/www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html). Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, Stanford Medical Informatics Technical Report SMI-2001-0880.
5. Добров Б. В., Иванов В.В., Лукашевич Н.В., Соловьев В.Д. [Онтологии и тезаурусы: модели, инструменты, приложения](http://www.intuit.ru/department/expert/ontoth/). — М.: Бином. Лаборатория знаний, 2009. — 173 с.

# приложения

Приложение А — Код реализации онтологии на языке Python

### Приложение А

Код реализации онтологии на языке Python

Листинг А.1 – Реализация онтологии

from typing import Any

import random

import re

class OntologyObject:

    shown\_field = 'name'

    def \_\_str\_\_(self) -> str:

        return getattr(self, self.shown\_field)

class Performer(OntologyObject):

    instances = []

    def \_\_init\_\_(self, name: str, country: str) -> None:

        self.name = name

        self.country = country

        Performer.instances.append(self)

class MusicRecord(OntologyObject):

    instances = []

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

        self.name = name

        MusicRecord.instances.append(self)

class Group(Performer):

    instances = []

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

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

        Group.instances.append(self)

class Musician(Performer):

    instances = []

    def \_\_init\_\_(self, name: str, country: str, in\_group: Group):

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

        self.in\_group = in\_group

        Musician.instances.append(self)

class Album(MusicRecord):

    instances = []

    def \_\_init\_\_(self, name: str, release\_year: int, performed\_by: Performer):

        super().\_\_init\_\_(name)

        self.release\_year = release\_year

        self.performed\_by = performed\_by

        Album.instances.append(self)

class Song(MusicRecord):

    instances = []

    def \_\_init\_\_(self, name: str, in\_album: Album):

        super().\_\_init\_\_(name)

        self.in\_album = in\_album

Продолжение Листинга А.1

        Song.instances.append(self)

def find\_related\_objects\_by\_value(cls: OntologyObject.\_\_class\_\_,

                                  lookup\_field: str, value: Any):

    instances = cls.instances

    result = []

    for instance in instances:

        if isinstance(getattr(instance, lookup\_field), OntologyObject):

            if not isinstance(value, OntologyObject):

                value = find\_object\_by\_name(

                    getattr(instance, lookup\_field).\_\_class\_\_, value)

                if value is None:

                    return None

        if getattr(instance, lookup\_field) == value:

            result.append((instance, type(instance).\_\_name\_\_))

    return result

def get\_class(class\_name: str):

    classes = {

        'musician': Musician,

        'group': Group,

        'album': Album,

        'song': Song

    }

    class\_name = class\_name.lower().strip()

    if class\_name in classes:

        return classes[class\_name]

    else:

        return None

def find\_object\_by\_name(cls: OntologyObject.\_\_class\_\_, name: str):

    instances = cls.instances

    for instance in instances:

        if instance.name == name:

            return instance

    return None

def get\_random\_class\_instance(cls: OntologyObject.\_\_class\_\_):

    instances = cls.instances

    return random.choice(instances)

def get\_related\_class(obj: OntologyObject):

    classes = [Album, Song, Musician, Group]

    class\_types = set()

    for cls in classes:

        cls\_instances = cls.instances

        for instance in cls\_instances:

            fields = [

                key for key in instance.\_\_dict\_\_.keys()

                if not re.match(r"\_\_\w\*\_\_", key)

            ]

            for field in fields:

                field\_value = getattr(instance, field)

                if field\_value == obj:

                    class\_types.add((cls, field))

    return list(class\_types)

def main():

    groups = [Group('Queen', 'Great Britain'), Group('Metallica', 'USA')]

Продолжение Листинга А.1

    musicians = [

        Musician('Freddie Mercury', 'Zanzibar', groups[0]),

        Musician('John Deacon', 'UK', groups[0]),

        Musician('Brian May', 'UK', groups[0]),

        Musician('Roger Taylor', 'UK', groups[0]),

        Musician('James Hetfield', 'USA', groups[1]),

        Musician('Kirk Hammett', 'USA', groups[1]),

        Musician('Lars Ulrich', 'USA', groups[1]),

        Musician('Robert Trujilio', 'USA', groups[1])

    ]

    albums = [

        Album('Mr.Bad Guy', 1985, musicians[0]),

        Album('A Night At The Opera', 1975, groups[1]),

        Album('Innuendo', 1991, groups[0]),

        Album('Ride The Lightning', 1984, groups[1]),

        Album('Master Of Puppets', 1986, groups[1])

    ]

    songs = [

        Song('Living On My Own', albums[0]),

        Song('Bohemiarn Rhapsody', albums[1]),

        Song('Love Of My Life', albums[1]),

        Song('Innuendo', albums[2]),

        Song('Ride The Lightning', albums[3]),

        Song('For Wthom The Bell Tolls', albums[3]),

        Song('Master Of Puppets', albums[4]),

        Song('Battery', albums[4])

    ]

    while True:

        while True:

            class\_name = input('Введите класс получаемых объектов: ')

            cls: OntologyObject.\_\_class\_\_ | None = get\_class(class\_name)

            if cls is None:

                print('Такого класса не существует\n\n')

            else:

                break

        while True:

            instance = get\_random\_class\_instance(cls)

            available\_fields = [

                key for key in instance.\_\_dict\_\_.keys()

                if not re.match(r"\_\_\w\*\_\_", key)

            ]

            field = input(

                f'Введите требуемое поле ( {", ".join(available\_fields)} ): ')

            try:

                getattr(instance, field)

                break

            except Exception:

                print('Такого поля в классе не существует\n\n')

        value = input('Введите значение поля: ')

        res = find\_related\_objects\_by\_value(cls, field, value)

        while True:

            flag = False

            if res is None or len(res) == 0:

                print('Объекты по введённому запросу не найдены')

                while True:

                    \_type = input(

Окончание Листинга А.1

                        '1 - повторить ввод запроса\nq - завершить выполнение программы\n'

                    )

                    if \_type == 1:

                        break

                    elif \_type == 'q':

                        exit(0)

            else:

                str\_objects = [

                    "\033[32m" + str(obj) + "\033[0m (\033[33m" +

                    str(obj\_type) + "\033[0m)" for obj, obj\_type in res

                ]

                print(f'Полученные объекты: {", ".join(str\_objects)}')

                while True:

                    \_type = input('1 - повторить ввод запроса\n'

                                  '2 - посмотреть связанные объекты\n'

                                  'q - завершить выполнение программы\n')

                    if \_type == '1':

                        flag = False

                        break

                    elif \_type == '2':

                        flag = True

                        break

                    elif \_type == 'q':

                        exit(0)

                if flag:

                    if len(res) == 1:

                        obj\_number = 1

                    else:

                        while True:

                            obj\_number = int(

                                input(

                                    f'Выберите номер нужного объекта (1-{len(res)}): '

                                ))

                            if obj\_number not in range(1, len(res) + 1):

                                print('Введён неправильный номер')

                            else:

                                break

                    instance = res[obj\_number - 1][0]

                    related\_classes = get\_related\_class(instance)

                    res = []

                    for rel\_class, field\_name in related\_classes:

                        res.extend(

                            find\_related\_objects\_by\_value(

                                rel\_class, field\_name, instance))

                else:

                    break

            if not flag:

                break

main()