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

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. Формальная модель онтологии представлена следующей формулой:

$$O =< X, R, F >$$

где X — конечное множество концептов (понятий, терминов) предметной области, которую представляет онтология;

R — конечное множество отношений между концептами (понятиями, терминами) заданной предметной области;

F — конечное множество функций интерпретации (аксиоматизации), заданных на концептах и/или отношениях онтологии.

1 ОНТОЛОГИЯ

1.1 Постановка задачи

Необходимо разработать онтологию выбранной предметной области — «Музыкальная индустрия». Данная предметная область выбрана в связи с тем, что в современном мире прослушивание музыки стало очень доступным с использованием стриминговых сервисов, и поэтому есть острая необходимость в систематизации музыки, чтобы её было доступно выкладывать в Интернет [2].

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 [1].

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).

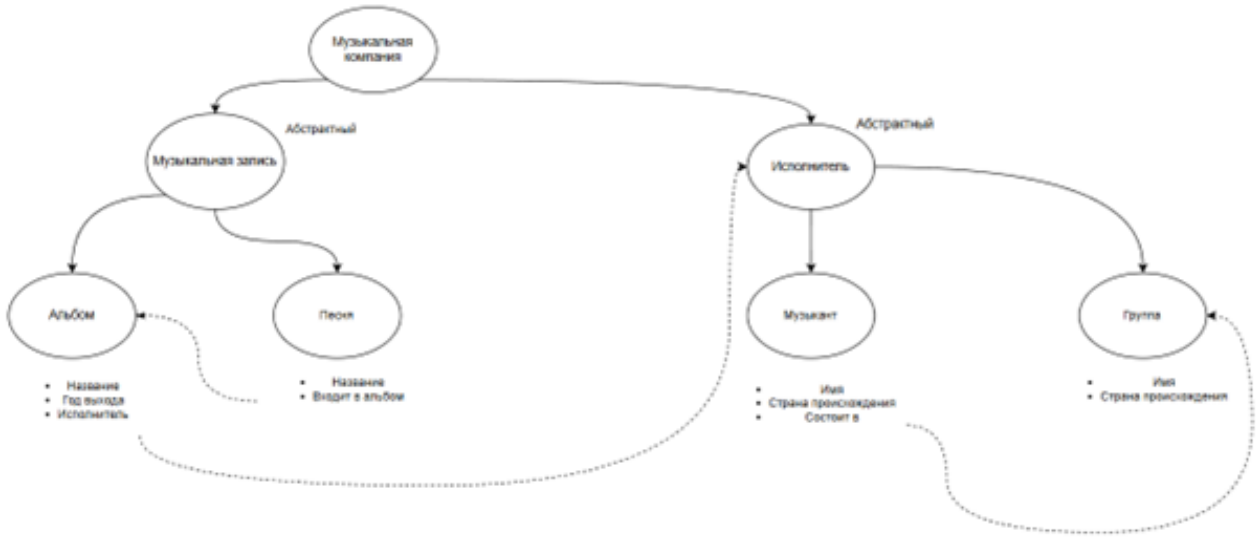

Рисунок 1.2.1 — Схема онтологии «Музыкальная индустрия»

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, представленные на Рисунке 1.3.1.

Рисунок 1.3.1 — Составленная иерархия классов

На Рисунке 1.3.2 представлены слоты класса «Альбом». Слотами данного класса являются: название альбома, исполнитель и год выпуска.

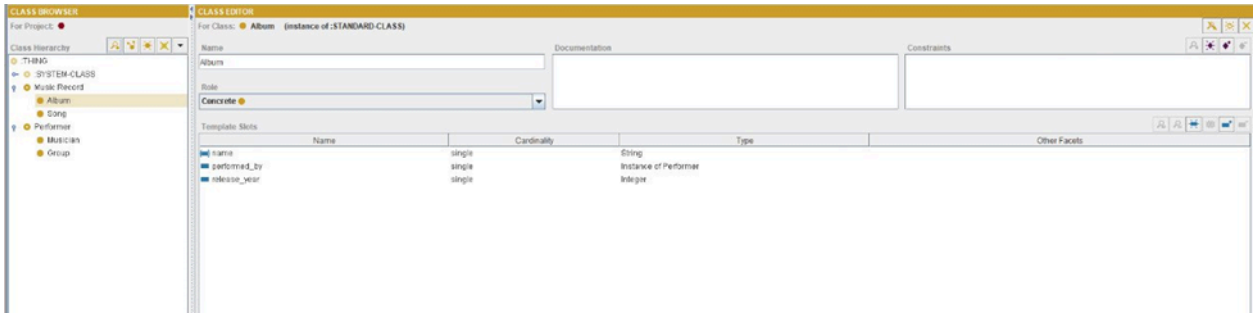

Рисунок 1.3.2 — Слоты класса «Альбом»

На Рисунке 1.3.3 представлены слоты класса «Песня». Слотами данного класса являются: альбом и название.

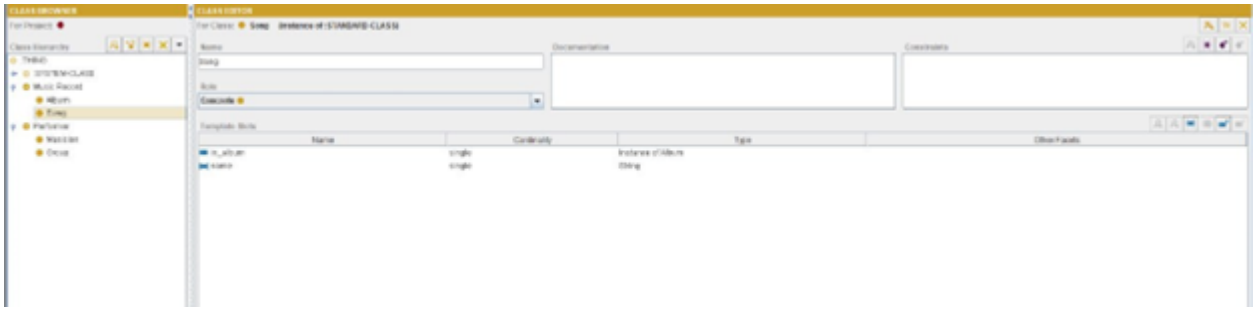

Рисунок 1.3.3 — Слоты класса «Песня»

На Рисунке 1.3.4 представлены слоты класса «Музыкант». Слотами данного класса являются: страна происхождения, группа и имя.

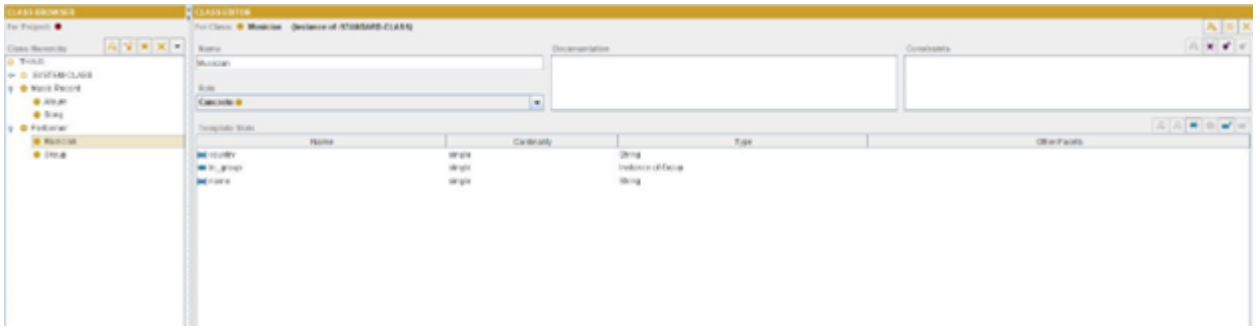

Рисунок 1.3.4 — Слоты класса «Музыкант»

На Рисунке 1.3.5 представлены слоты класса «Группа». Слотами данного класса являются страна и название.

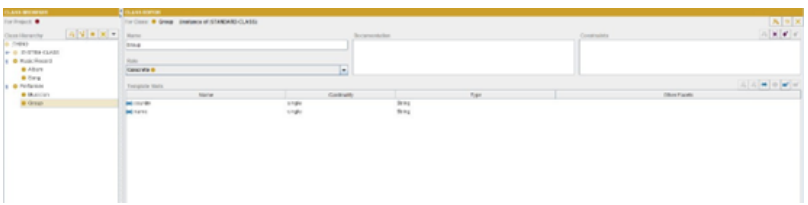

Рисунок 1.3.5 — Слоты класса «Группа»

После составления и описания классов созданы экземпляры каждого из классов. На Рисунке 1.3.6 представлены экземпляры класса «Группа» и значения полей в одном из них.

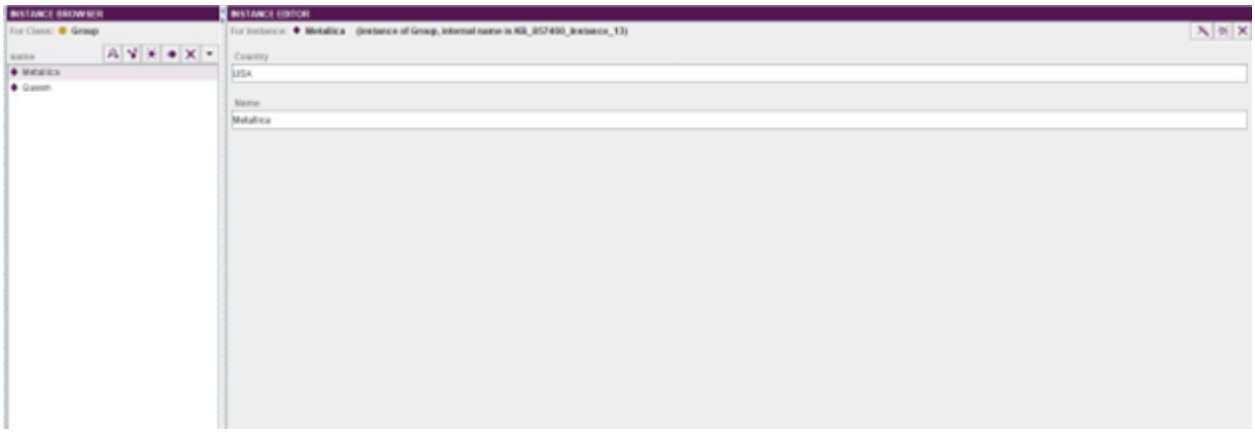

Рисунок 1.3.6 — Экземпляры класса «Группа»

На Рисунке 1.3.7 представлены экземпляры класса «Музыкант» и значения полей в одном из них.

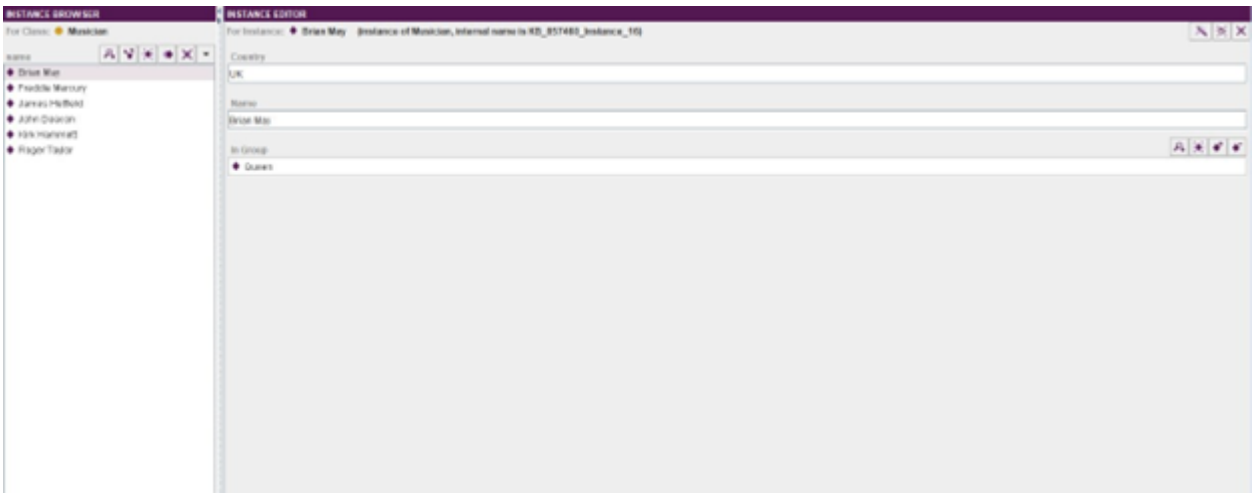

Рисунок 1.3.7 — Экземпляры класса «Музыкант»

На Рисунке 1.3.8 представлены экземпляры класса «Песня» и значения полей в одном из них.

Рисунок 1.3.8 — Экземпляры класса «Песня»

На Рисунке 1.3.8 представлены экземпляры класса «Альбом» и значения полей в одном из них.

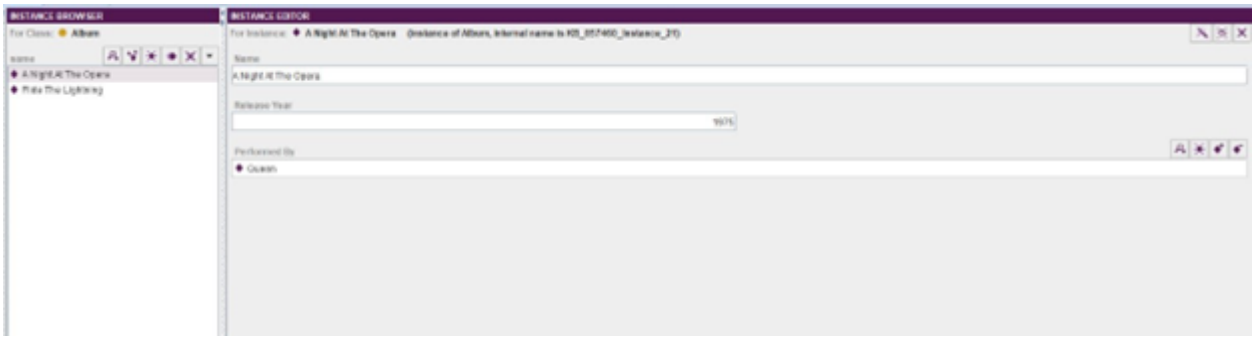

Рисунок 1.3.9 — Экземпляры класса «Альбом»

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. Проделан обычный запрос на получение экземпляров (Рисунок 1.4.1).

Рисунок 1.4.1 – Одинарный запрос на получение песен из альбома На Рисунке 1.4.2 представлен цепной запрос на получение песен, написанных одной группой.

Рисунок 1.4.2 – Цепной запрос на получение песен, написанных одной группой На Рисунке 1.4.3 представлен цепной запрос на получение песен, написанных одним музыкантом.

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. Её код представлен в Листинге А.1 На Рисунке 1.5.1 представлен результат выполнения запроса музыкантов в группе, выполненный в программе.

Рисунок 1.5.1 – Результат выполнения программы На Рисунке 1.5.2 представлен результат получения песен, написанных группой, с помощью запроса, выполненного в программе.

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 ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Карпенко, А. П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой: учебное пособие / А. П. Карпенко — 3-е изд. — Москва: Издательство МГТУ им. Н. Э. Баумана, 2021. — 446 с.

2. Сорокин, А. Б. Введение в роевой интеллект: теория, расчеты и приложения [Электронный ресурс]: Учебно-методическое пособие / А. Б. Сорокин , Л. М. Железняк — М.: Московский технологический университет (МИРЭА), 2019.

ПРИЛОЖЕНИЯ

Приложение А — Код реализации онтологии на языке Python

Приложение А

Код реализации онтологии на языке Python

Листинг А.1 — Код файла main.py

```
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
        Song.instances.append(self)
```

Продолжение Листинга А.1

```
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')]
    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]),
```

Продолжение Листинга А.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(
                                '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
```

Окончание Листинга А.1

```
                    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()
```