DL - Query в Protége

Для построения запросов к онтологии в Protégé можно использовать плагин DL-Query, предоставляющий мощные и легкие в использовании средства поиска. Плагин поддерживает язык запросов, основанный на манчестерском синтаксисе языка OWL (Manchester OWL syntax). Этот синтаксис разработан для диалекта OWL DL, является простым и интуитивно понятным. Информация о некотором классе, свойстве или индивиде собирается в одну конструкцию, называемую фреймом.

Для формирования запроса необходимо записать выражение класса (class expression) – выражение, в котором упоминаются классы, ограничения, конъюнкции или дизъюнкции классов и ограничений.

Далее, опционально выбирая в панели выбора сведения, которые необходимо получить, можно составлять разные запросы о классе, например:

− какие экземпляры заявлены в классе;

− какие суперклассы есть у искомого класса;

− какие эквивалентные классы есть у искомого класса и др.

Для выполнения запросов в Protégé необходимо выбрать пункт главного меню Windows – Tabs – DL-Query. Появится вкладка, показанная на рис. 1. Перед тем, как выполнять запрос, необходимо запустить машину логического вывода: выбрать пункт Reasoner и отметить, например, Hermit – Start Reasoner. Для выполнения запроса необходимо заполнить поле запроса Query и нажать кнопку execute. Результаты выполнения запроса отображаются в поле Query Results.

Рассмотрим простейший запрос DL-Query:

Человек and фамилия value "Одинцов"

Результатом выполнения запроса является список URI всех объектов класса Человек, для которых свойство «фамилия» равно «Одинцов». Результаты поиска можно фильтровать, выделяя надклассы искомого класса, подкласс, эквивалентные и несовместимые классы, а также индивидов класса, удовлетворяющих запросу.

Запросы экземпляров

Пусть имеется онтология, содержащая:

• Class:

o Person

• Data Properties:

o hasGivenName

o hasSurname

Кроме того, онтология содержит несколько сотен экземпляров класса Сlass. Чтобы найти экземпляр с именем "Matthew":

hasGivenName value "Matthew"

Чтобы уточнить, что именно необходимо найти, надо отметить пункт "Individuals".

Следующий запрос эквивалентен предыдущему: Person and hasGivenName value "Matthew" Список всех индивидов будет получен после выполнения запроса :

Person

Манчестерский синтаксис предоставляет богатые выразительные возможности, позволяющие составлять сложные запросы.

Краткая спецификация синтаксиса может быть найдена по ссылке: http://www.co-ode.org/resources/reference/manchester\_syntax/.

Другие примеры запросов, использующие литералы и ограничения кардинальности:

hasAge value "21"^^long

hasAge some int

hasChild some Man

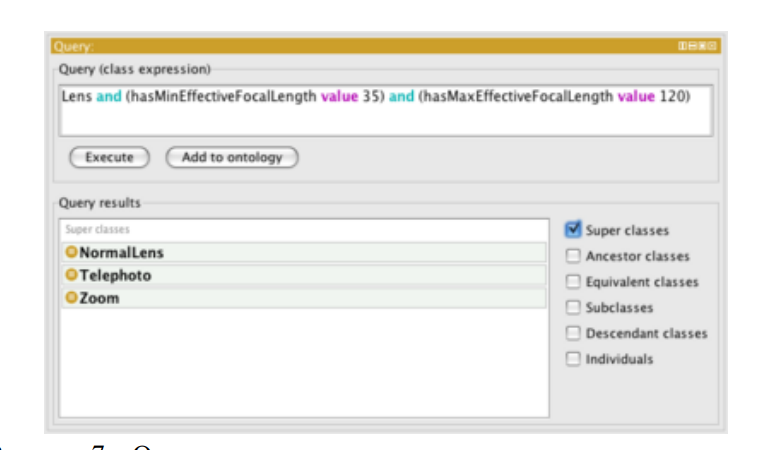
hasSibling only Woman

hasCountryOfOrigin value England

hasChild min 3

hasChild exactly 3

hasChild max 3



Выполнение SPARQL-запросов в Protégé.

По аналогии с базами данных для [RDF](http://sites.google.com/site/anisimovkhv/learning/iis/lecture/tema11#p113) и [OWL](http://sites.google.com/site/anisimovkhv/learning/iis/lecture/tema11#p113) онтологических моделей возможно выполнение запросов с целью извлечения знаний, содержащихся в них.

Запрос начинается с ключевого слова «SELECT», после которого указываются:

- переменные, отображаемые в результатах запроса;

- условие «WHERE», задающее ограничения (правила) на выборку результатов.

Переменные, используемые в запросе должны начинаться со знака «?».

Условие может включать в себя несколько [RDF-троек](http://sites.google.com/site/anisimovkhv/learning/iis/lecture/tema11#p113) (подусловий), после каждой из которых ставится «.». Считается, что такие подусловия соединены конъюнкцией (логическим И). В условие могут включаться дополнительные конструкции (**графовые шаблоны**):

- FILTER – ограничение на значения переменных;

- OPTIONAL – допущение отсутствия значения для переменной в [RDF-тройке](http://sites.google.com/site/anisimovkhv/learning/iis/lecture/tema11#p113);

- UNION – объединение результатов нескольких частей запроса (объединение дизъюнкцией (логическим ИЛИ)).

В запросе можно указать дополнительные параметры (**модификаторы**).

Таблица 1

Модификаторы

| Модификатор | Положение | Назначение модификатора |
| --- | --- | --- |
| DISTINCT | после слова SELECT | гарантирует уникальность решений (исключает дублирование результатов) |
| REDUCED | после слова SELECT | предписывает выдачу всех решений, включая повторы |
| ORDER BY | после раздела WHERE {} | сортировка решений (результатов) |
| OFFSET <n> | после раздела WHERE {} | предписывает исключить из выборки n первых решений |
| LIMIT <n> | после раздела WHERE {} | ограничивает количество выдаваемых решений |

Выполнение запросов осуществляется на вкладке «SPARQL Query».

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Рис.1. Вкладка «SPARQL Query»

Перед выполнением запросов рекомендуется указать сокращенное имя (префикс) онтологии, чтобы в дальнейшем не было необходимости указания полного [IRI](http://sites.google.com/site/anisimovkhv/learning/iis/lecture/tema11#p112) для ее элементов (классов, свойств, индивидов).

На рис. 1 показан стандартный запрос, отображаемый при первом открытии данной вкладки. Он предписывает вывести все подклассы (переменная ?subject) для всех классов (переменная ?object), определенных в онтологии.

Ниже приводятся несколько запросов с использованием модификаторов и графовых шаблонов.

Показать фамилию, имя и группу 3 студентов (LIMIT 3), пропустив первых 2 (OFFSET 2), в виде отсортированного списка по фамилии (ORDER BY ASC(?fam)).

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

Автоматически созданное описание

Рис.2. Запрос с модификаторами

Показать фамилию, имя и номер квартиры студентов, у которых этот номер меньше 40 (FILTER (?number < 40)).

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

Автоматически созданное описание

Рис.3. Запрос с графовым шаблоном

ИЛИ

Объекты, связанные свойством типа isInputOf (isControlOf, isOutputOf, isMechanismOf)

PREFIX db: <http://www.semanticweb.org/boris/ontologies/2014/3/untitled-ontology-126#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?x ?property ?y

            WHERE { ?property rdfs:subPropertyOf\* db:isInputOf .

            ?property rdfs:domain ?x .

          ?property rdfs:range ?y .

            }

|  |
| --- |
|  |  |

Результат:

В данном запросе следует обратить внимание на описание префиксов, определяющих место описания объектов и предикатов, используемых для описания запросов к данной онтологии.

Стандартный набор префиксов должен быть дополнен префиксами, связанными с конкретной предметной областью. В данном случае это префикс db.

Описание этого префикса можно взять из адресной строки конкретной онтологии. Остальные префиксы являются стандартными. Они всегда присутствуют в стандартном запросе появляющемся при первом открытии каждого экземпляра вкладки SPARQL Query.

|  |
| --- |
|  |  |

Подготовленный запрос записывается или копируется во вкладку SPARQL Query. Затем следует кликнуть на кнопку Execute в нижней части вкладки. Результат запроса оформляется в виде таблицы.

**Запрос 2** Значения полей Дата таблицы Заказы

PREFIX db: <http://www.semanticweb.org/boris/ontologies/2014/3/untitled-ontology-126#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT  ?t  ?y

            WHERE {

?x db:isДата\_заказаOf ?t.

?x db:Дата\_заказа ?y

}

  
Результат имеет вид:

Данный запрос иллюстрирует поиск информации на основе использования двух типов отношений (между экземплярами классов и конкретными характеристиками данных).

SPARQL (SPARQL Protocol and RDF Query Language) — это язык запросов для работы с данными в формате RDF (Resource Description Framework). Он позволяет извлекать и манипулировать данными, хранящимися в триплетах (субъект, предикат, объект).

▎Основные конструкции SPARQL

1. SELECT — для выборки данных.

2. CONSTRUCT — для создания новых графов.Чтобы вывести результаты, где значение предиката отсутствует (т.е. пустое значение или не задано), вы можете использовать фильтр FILTER NOT EXISTS в вашем SPARQL-запросе. Вот пример запроса, который находит все объекты, у которых отсутствует определённый предикат:

PREFIX ex: <http://example.org/>

SELECT ?subject

WHERE {

?subject a ex:Person .

FILTER NOT EXISTS { ?subject ex:hasName ?name }

}

▎Объяснение запроса:

1. PREFIX: Устанавливает пространство имен для сокращения URI.

2. SELECT: Указывает переменную ?subject, которую мы хотим получить в результате.

3. WHERE: Определяет условия для выборки данных:

• ?subject a ex:Person: выбирает всех индивидов, которые являются экземплярами класса ex:Person.

• FILTER NOT EXISTS { ?subject ex:hasName ?name }: фильтрует результаты, оставляя только тех субъектов, у которых отсутствует предикат ex:hasName.

Таким образом, этот запрос вернет всех людей, у которых нет имени.

3. ASK — для проверки наличия данных.

4. DESCRIBE — для получения описания ресурса.

▎Синтаксис

Запросы SPARQL обычно начинаются с объявления префиксов, затем идут ключевые слова (например, SELECT), после чего указывается, какие данные нужно выбрать или создать.

▎Пример 1: Простой SELECT запрос

Запрос: Найти все имена авторов.

PREFIX ex: <http://example.org/>

SELECT ?authorName

WHERE {

?author ex:name ?authorName.

}

▎Пример 2: SELECT с фильтрацией

Запрос: Найти имена авторов, родившихся после 1980 года.

PREFIX ex: <http://example.org/>

SELECT ?authorName

WHERE {

?author ex:name ?authorName.

?author ex:birthYear ?birthYear.

FILTER(?birthYear > 1980)

}

▎Пример 3: JOIN запрос

Запрос: Найти названия произведений и имена их авторов.

PREFIX ex: <http://example.org/>

SELECT ?artworkTitle ?authorName

WHERE {

?artwork ex:title ?artworkTitle.

?artwork ex:createdBy ?author.

?author ex:name ?authorName.

}

▎Пример 4: Использование OPTIONAL

Запрос: Найти названия произведений и имена авторов, если есть информация о годе создания.

PREFIX ex: <http://example.org/>

SELECT ?artworkTitle ?authorName ?creationYear

WHERE {

?artwork ex:title ?artworkTitle.

?artwork ex:createdBy ?author.

?author ex:name ?authorName.

OPTIONAL { ?artwork ex:creationYear ?creationYear. }

}

▎Пример 5: CONSTRUCT запрос

Запрос: Создать новый граф, который содержит информацию о произведениях и их авторах.

PREFIX ex: <http://example.org/>

CONSTRUCT {

?artwork ex:title ?artworkTitle.

?artwork ex:createdBy ?author.

}

WHERE {

?artwork ex:title ?artworkTitle.

?artwork ex:createdBy ?author.

}

▎Пример 6: ASK запрос

Запрос: Проверить, существует ли автор с именем "Ван Гог".

PREFIX ex: <http://example.org/>

ASK {

?author ex:name "Ван Гог".

}

▎Пример 7: DESCRIBE запрос

Запрос: Получить описание автора с именем "Пикассо".

PREFIX ex: <http://example.org/>

DESCRIBE ?author

WHERE {

?author ex:name "Пикассо".

}

▎Заключение

SPARQL предоставляет мощные инструменты для работы с RDF-данными, позволяя формулировать запросы различной сложности. Вы можете комбинировать различные конструкции для достижения необходимых результатов, используя фильтры, опциональные данные и другие возможности языка.

SPARQL (SPARQL Protocol and RDF Query Language) — это язык запросов, используемый для работы с данными в формате RDF (Resource Description Framework). Он позволяет извлекать и манипулировать данными, хранящимися в триплетах (субъект, предикат, объект).

▎Основные конструкции SPARQL

1. SELECT: используется для извлечения данных.

2. ASK: возвращает булевое значение, указывающее, существуют ли данные, соответствующие запросу.

3. DESCRIBE: возвращает описание ресурсов.

4. CONSTRUCT: создает новые триплеты на основе заданных шаблонов.

▎Пример синтаксиса

PREFIX ex: <http://example.org/>

SELECT ?subject ?predicate ?object

WHERE {

?subject ?predicate ?object .

}

▎Примеры запросов

1. Простой SELECT-запрос

SELECT ?s WHERE { ?s a ex:Person }

2. Запрос с фильтром

SELECT ?name WHERE { ?person ex:name ?name . FILTER(?name = "Alice") }

3. Использование нескольких предикатов

SELECT ?s ?o WHERE { ?s ex:hasFriend ?o }

4. Запрос с ORDER BY

SELECT ?name WHERE { ?person ex:name ?name }

ORDER BY ?name

5. LIMIT и OFFSET

SELECT ?name WHERE { ?person ex:name ?name } LIMIT 10 OFFSET 5

6. Запрос с OPTIONAL

SELECT ?person ?name ?age WHERE {

?person ex:name ?name .

OPTIONAL { ?person ex:age ?age }

}

7. Группировка данных

SELECT (COUNT(?person) AS ?count) WHERE { ?person a ex:Person }

GROUP BY ?count

8. Использование UNION

SELECT ?s WHERE {

{ ?s ex:hasPet ?pet . }

UNION

{ ?s ex:hasPlant ?plant . }

}

9. Запрос с подзапросом

SELECT ?s WHERE {

{ SELECT ?s WHERE { ?s ex:age ?age . FILTER(?age > 30) } }

}

10. ASK-запрос

ASK WHERE { ?s ex:name "Alice" }

11. DESCRIBE-запрос

DESCRIBE ex:Alice

12. CONSTRUCT-запрос

CONSTRUCT {

?person ex:hasAge ?age .

} WHERE {

?person ex:age ?age .

}

13. Запрос с использованием BIND

SELECT ?name (STR(?name) AS ?stringName) WHERE {

?person ex:name ?name .

BIND(CONCAT(?name, " is a person") AS ?description)

}

14. Запрос с фильтрацией по регулярному выражению

SELECT ?name WHERE {

?person ex:name ?name .

FILTER regex(?name, "^A", "i")

}

15. Работа с датами

SELECT ?event WHERE {

?event ex:date ?date .

FILTER(?date > "2023-01-01"^^xsd:date)

}

16. Запрос с анонимными ресурсами

SELECT ?s WHERE {

[] ex:hasFriend ?s .

}

17. Запрос на получение уникальных значений

SELECT DISTINCT ?name WHERE { ?person ex:name ?name }

18. Запрос с использованием VALUES

SELECT ?s WHERE {

VALUES ?color { ex:red ex:green ex:blue }

?s ex:color ?color .

}

19. Сложный запрос с несколькими фильтрами

SELECT ?s WHERE {

?s a ex:Person .

FILTER(?age > 30 && regex(?name, "Smith"))

}

20. Запрос с использованием GROUP BY и HAVING

SELECT ?city (COUNT(?person) AS ?count) WHERE {

?person ex:livesIn ?city .

} GROUP BY ?city HAVING (?count > 5)

▎Пример запроса с DESCRIBE

DESCRIBE <http://example.org/Alice>

Этот запрос вернет описание ресурса "Alice", включая все связанные триплеты, которые могут быть полезны для понимания этого ресурса в контексте данных RDF.

Если у вас есть дополнительные вопросы или нужна помощь с конкретными аспектами SPARQL, дайте знать!

▎Справка по языку запросов SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) — это язык запросов, предназначенный для работы с данными, представленными в формате RDF (Resource Description Framework). Он позволяет извлекать и манипулировать данными, хранящимися в триплетах (субъект, предикат, объект).

▎Основные конструкции SPARQL

1. SELECT: используется для извлечения данных.

2. WHERE: определяет шаблон для поиска данных.

3. FILTER: позволяет добавлять условия к запросам.

4. OPTIONAL: добавляет необязательные части к запросу.

5. UNION: объединяет результаты нескольких запросов.

6. GROUP BY: группирует результаты по определённому полю.

7. ORDER BY: сортирует результаты.

▎Примеры запросов SPARQL

▎1. Простой запрос на получение всех субъектов

SELECT ?s WHERE { ?s ?p ?o }

▎2. Получение всех объектов для конкретного субъекта

SELECT ?o WHERE { <http://example.org/resource/Subject1> ?p ?o }

▎3. Получение всех предикатов для конкретного субъекта

SELECT ?p WHERE { <http://example.org/resource/Subject1> ?p ?o }

▎4. Получение всех субъектов с определённым объектом

SELECT ?s WHERE { ?s ?p <http://example.org/resource/Object1> }

▎5. Запрос с фильтрацией по значению объекта

SELECT ?s WHERE { ?s ?p ?o FILTER(?o = "SomeValue") }

▎6. Использование OPTIONAL для получения дополнительных данных

SELECT ?s ?o WHERE {

?s ?p ?o .

OPTIONAL { ?s <http://example.org/property/Property1> ?o2 }

}

▎7. Группировка и подсчёт количества субъектов по предикату

SELECT ?p (COUNT(?s) AS ?count) WHERE {

?s ?p ?o

} GROUP BY ?p

2

▎8. Сортировка результатов по субъектам

SELECT ?s WHERE { ?s ?p ?o } ORDER BY ASC(?s)

▎9. Запрос с использованием UNION для объединения результатов

SELECT ?s WHERE {

{ ?s <http://example.org/property/Type1> ?o }

UNION

{ ?s <http://example.org/property/Type2> ?o }

}

▎10. Использование VALUES для задания фиксированных значений

SELECT ?s WHERE {

VALUES ?type { <http://example.org/type/Type1> <http://example.org/type/Type2> }

?s a ?type

}

▎11. Запрос с несколькими фильтрами

SELECT ?s WHERE {

?s ?p ?o .

FILTER(?o > 10 && ?o < 100)

}

▎12. Получение всех предикатов для конкретного субъекта и объекта

SELECT ?p WHERE {

<http://example.org/resource/Subject1> ?p <http://example.org/resource/Object1>

}

▎13. Использование GROUP\_CONCAT для объединения объектов в строку

SELECT ?s (GROUP\_CONCAT(?o; separator=", ") AS ?objects) WHERE {

?s ?p ?o

} GROUP BY ?s

▎14. Запрос с использованием подзапроса

SELECT ?s WHERE {

{ SELECT ?s WHERE { ?s a <http://example.org/type/Type1> } }

}

▎15. Использование FILTER NOT EXISTS для исключения результатов

SELECT ?s WHERE {

?s a <http://example.org/type/Type1> .

FILTER NOT EXISTS { ?s <http://example.org/property/Property1> ?o }

}

▎16. Запрос на получение всех субъектов с определённым классом

SELECT ?s WHERE {

?s a <http://example.org/class/Class1>

}

▎17. Запрос с использованием MIN и MAX

SELECT (MIN(?value) AS ?minValue) (MAX(?value) AS ?maxValue) WHERE {

?s <http://example.org/property/value> ?value.

}

▎18. Запрос на получение уникальных объектов для конкретного предиката

SELECT DISTINCT ?o WHERE {

?s <http://example.org/property/Property1> ?o.

}

▎19. Запрос на получение данных с использованием HAVING

SELECT ?p (COUNT(?s) AS ?count) WHERE {

?s ?p ?o

} GROUP BY ?p HAVING (?count > 5)

▎20. Запрос на получение данных с использованием FILTER и регулярных выражений

SELECT ?s WHERE {

?s <http://example.org/property/name> ?name.

FILTER regex(?name, "^A.\*")

}

\_\_\_\_\_\_

Чтобы вывести результаты, где значение предиката отсутствует (т.е. пустое значение или не задано), вы можете использовать фильтр

FILTER NOT EXISTS

в вашем SPARQL-запросе. Вот пример запроса, который находит все объекты, у которых отсутствует определённый предикат:

PREFIX ex: <http://example.org/>

SELECT ?subject

WHERE {

?subject a ex:Person .

FILTER NOT EXISTS { ?subject ex:hasName ?name }

}

▎**Объяснение запроса:**

1. **PREFIX**: Устанавливает пространство имен для сокращения URI.

2. **SELECT**: Указывает переменную

?subject

, которую мы хотим получить в результате.

3. **WHERE**: Определяет условия для выборки данных:

•

?subject a ex:Person

: выбирает всех индивидов, которые являются экземплярами класса

ex:Person

.

•

FILTER NOT EXISTS { ?subject ex:hasName ?name }

: фильтрует результаты, оставляя только тех субъектов, у которых отсутствует предикат

ex:hasName

.

Таким образом, этот запрос вернет всех людей, у которых нет имени.

\_\_\_\_\_\_

Если вы хотите заменить name на предикат, который представляет собой object property, то вам нужно будет использовать другой класс или ресурс в качестве значения. Например, предположим, что у вас есть объектное свойство ex:hasName, которое связывает экземпляр класса ex:Person с экземпляром класса ex:Name.

Вот как может выглядеть запрос SPARQL для получения всех людей и их имен, если имя представлено как объект:

PREFIX ex: <http://example.org/>

SELECT ?person ?name

WHERE {

?person a ex:Person .

?person ex:hasName ?nameResource .

?nameResource ex:value ?name .

}

▎Объяснение запроса:

1. PREFIX: Определяет пространство имен для сокращения URI.

2. SELECT: Указывает, какие переменные мы хотим получить в результате.

3. WHERE: Определяет условия для выборки данных:

• ?person a ex:Person: выбирает всех индивидов, которые являются экземплярами класса ex:Person.

• ?person ex:hasName ?nameResource: связывает каждого человека с ресурсом имени.

• ?nameResource ex:value ?name: извлекает значение имени из ресурса.

Этот запрос предполагает, что имена представлены как отдельные ресурсы с некоторым свойством (например, ex:value), которое содержит строковое значение имени.