Преподаватель: **ГУБИН Александр Николаевич**

к.т.н., доц.

каф. ИУС (ауд. 209/2) Тел. 3051278

Состав курса (1-ый сем):

Лекции - ___ ч.

Лаб.р. - ___ ч.

Практика- ____ ч.

Зачет

Лекция №7

SPARQL

Язык запросов SPARQL для RDF

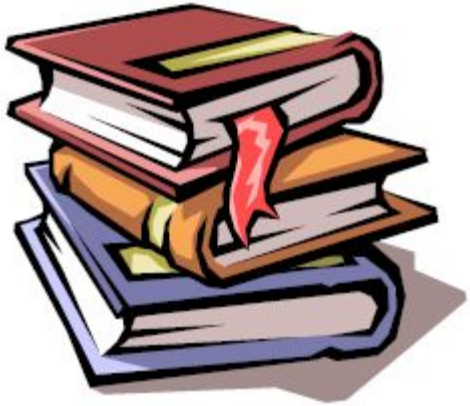
Рекомендация W3C, 15 января 2008

Текущая версия:

http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

Последняя версия:

http://www.w3.org/TR/rdf-sparql11-query/



SPARQL

Basic Structure
PREFIX
WHERE

Graph Patterns
Basic Graph Patterns
{...}
OPTIONAL
UNION

Filter
BOUND
isURI
isBLANK
isLITERAL
STR
LANG
DATATYPE
sameTERM
langMATCHES
REGEX

Modifiers
ORDER BY
LIMIT
OFFSET
DISTINCT

Output Formats
SELECT
CONSTRUCT
ASK
DESCRIBE

SPARQL

Пространства имен

Предполагаются следующие привязки префикса пространства имен, если не оговорено иного:

Префикс	IRI
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
xsd:	http://www.w3.org/2001/XMLSchema#
fn:	http://www.w3.org/2005/xpath-functions#

SPARQL

Описания результатов

Результирующие наборы представляются в табличном виде.

В показанном ниже результирующем наборе имеется три переменных: x, y и z (показаны заголовками столбцов).

Каждое решение - это строка таблицы.

В данном случае решение будет единственным, где переменная **x связана с "Alice",** переменная **y - с <http://example/a>**, и переменная **z не связана с RDF-термином**. Переменные не обязательно связаны в решении.

x	y	z
"Alice"	<http://example/a>	

SPARQL

Синтаксис IRI

Префиксные имена

Ключевое слово **PREFIX** связывает метку префикса с **IRI**. Префиксное имя представляет собой метку префикса с локальным элементом (local part), разделенными двоеточием ":". Префиксное имя преобразовывается в IRI путем конкатенации IRI, связанного с префиксом, и локального элемента.

Метка префикса либо локальный элемент могут быть пустыми. Обратите внимание, что локальные имена SPARQL могут начинаться с цифры, в то время как локальные имена XML нет.

Синтаксис IRI

Относительные IRI

Относительные IRI составляются из базовых IRI

Ключевое слово BASE определяет базовый IRI, используемый для разрешения относительных IRI

SPARQL

Синтаксис IRI

Ниже указано несколько способов написания одного и того же IRI:

http://example.org/book/book1

BASE <http://example.org/book/>
<book1>

PREFIX book: <http://example.org/book/>
book:book1

SPARQL

Синтаксис литералов

Общий синтаксис литералов представляет собой строку (заключенную либо в двойные, либо в одинарные кавычки - **"..."**, или, **'...'**), либо с необязательным тегом языка (который начинается с @), либо с необязательным типом данных (**datatype**) IRI или префиксным именем (которое начинается с **^^**).

Для удобства целые числа можно записывать непосредственно (без кавычек и явных типов данных IRI). Они интерпретируются как типизированные литералы со значением datatype, равным **xsd:integer.**

Десятичные числа — в которых присутствует дробная часть, отделенная '.', но не представленные в экспоненциальной форме — интерпретируются как **xsd:decimal.**

А числа, представленные в экспоненциальной форме — как **xsd:double**.

Значения типа **xsd:boolean** можно записать как **true** или **false.**

Синтаксис литералов

Примеры синтаксиса литералов в SPARQL:

- "chat"
- 'chat'@fr с тегом языка "fr"
- **"xyz"^^<http://example.org/ns/userDatatype>**
- **"abc"^^appNS:appDataType**
- **'"The librarian said, "Perhaps you would enjoy 'War and Peace'."'"‘**

- 1, что то же самое, что и"1"^^xsd:integer
- 1.3, что то же самое, что и "1.3"^^xsd:decimal
- 1.300, что то же самое, что и "1.300"^^xsd:decimal
- 1.0e6, что то же самое, что и "1.0e6"^^xsd:double
- true, что то же самое, что и "true"^^xsd:boolean
- false, что то же самое, что и "false"^^xsd:boolean

SPARQL

Синтаксис переменных запроса

Переменные запроса в SPARQL-запросах имеют **глобальную область действия.**
Использование заданного имени переменной где-либо в запросе указывает на одну и туже переменную.
Переменные имеют префиксы "?" или "$";
"?" или "$" не являются составными частями имени переменной. **$abc** и **?abc** обозначают в запросе одинаковые переменные.

Синтаксис безымянных вершин

Безымянные вершины в шаблонах графа воспринимаются как неизвестные переменные, а не как ссылки на отдельные безымянные вершин в данных, к которым выполняется запрос.
Безымянные вершины обозначаются либо с помощью меток, например,"**_:abc**", либо посредством сокращенной формы **"[]"**.
Безымянная вершина, которая используется в синтаксисе запроса только один раз, может обозначаться с помощью **[]**.
Одинаковые метки безымянных вершин не могут быть использованы в двух различных основных графовых шаблонах в одном и том же запросе.
Конструкция**[:p :v]** может быть использована в шаблонах триплетов. Она создает метку безымянной вершины, которая в свою очередь применяется в качестве субъекта для всех заключенных внутри пар предикат-объект.
Созданная безымянная вершина может также использоваться в последующих шаблонах триплетов на месте субъекта и объекта

SPARQL

Синтаксис безымянных вершин

Следующие две формы
[:p "v"] . [] :p "v" .
задают уникальную метку безымянной вершины, что эквивалентно записи:
_:b57 :p "v" .
Заданная метка вершины может быть использована в качестве субъекта или объекта для последующих шаблонов триплетов. Например, в качестве субъекта:
[:p "v"] :q "w" .
эквивалентного двум триплетам:
_:b57 :p "v" . _:b57 :q "w" .
и объекта:
:x :q [:p "v"] .
эквивалентного двум триплетам:
:x :q _:b57 . _:b57 :p "v" .

SPARQL

Синтаксис шаблонов триплетов

Шаблоны триплетов записываются в виде разделенного пробелами списка:
субъект, предикат и объект.
Существуют варианты сокращенной записи некоторых простых конструкций шаблонов триплетов.

Синтаксис шаблонов триплетов

Следующие примеры демонстрируют один и тот же запрос:

```
PREFIX dc: http://purl.org/dc/elements/1.1/
 SELECT ?title
WHERE { <http://example.org/book/book1> dc:title ?title }
```

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <http://example.org/book/>
SELECT $title
WHERE { :book1 dc:title $title }
```

```
BASE <http://example.org/book/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT $title WHERE
{ <book1> dc:title ?title }
```

Списки предикат-объект

Шаблоны триплетов с простым субъектом могут быть записаны таким образом, что субъект упоминается в записи лишь единожды и используется для более чем одного шаблона триплета.
Для этих целей применяется нотация ";".

```
?x foaf:name ?name ;
   foaf:mbox ?mbox .
```

Это эквивалентно записи шаблонов триплетов:

```
?x foaf:name ?name .
?x foaf:mbox ?mbox .
```

SPARQL

Списки объектов

Если шаблоны триплетов используют общий и субъект, и предикат, объекты могут быть отделены посредством ",".

?x foaf:nick "Alice" , "Alice_" .

Это эквивалентно записи шаблонов триплетов:

?x foaf:nick "Alice" .
?x foaf:nick "Alice_" .

Списки объектов могут быть объединены со списками предикат-объект:

?x foaf:name ?name ;
foaf:nick "Alice" , "Alice_" .

что эквивалентно следующей записи:

?x foaf:name ?name .
?x foaf:nick "Alice" .
?x foaf:nick "Alice_" .

SPARQL

rdf:type

Ключевое слово "**a**"может быть использовано в качестве предиката в шаблоне триплета и является альтернативой для IRI **http://www.w3.org/1999/02/22-rdf-syntax-ns#type.** Данное ключевое слово чувствительно к регистру

```
?x a :Class1 .
[ a :appClass ] :p "v" .
```

является синтаксическим сахаром для:

```
?x rdf:type :Class1 .
_:b0 rdf:type :appClass .
_:b0 :p "v" .
```

SPARQL

Групповые графовые шаблоны

Групповой графовый шаблон в строке запроса SPARQL ограничен фигурными скобками: **{}**.
Например, запросный шаблон этого запроса представлен групповым шаблоном запроса, состоящим из одного основного графового шаблона.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
 {
?x foaf:name ?name .
?x foaf:mbox ?mbox .
 }
```

SPARQL

Групповые графовые шаблоны

Те же решения могут быть получены из запроса, который группирует шаблоны триплетов в два основных графовых шаблона.
К примеру, структура следующего запроса отличается от предыдущего, однако результаты получатся теми же самыми:

```
PREFIX foaf:    http://xmlns.com/foaf/0.1/
 SELECT ?name ?mbox
WHERE
{
{ ?x foaf:name ?name . }
 { ?x foaf:mbox ?mbox . }
  }
```

SPARQL

Область действия фильтров

Ограничение, выраженное с помощью использования ключевого слова **FILTER**, накладывает определенные рамки на решения.
Действие ограничения распространяется на всю группу, в которой присутствует фильтр.
Все следующие шаблоны имеют одинаковые решения:

```
{
?x foaf:name ?name .
?x foaf:mbox ?mbox .
 FILTER regex(?name, "Smith")
}
```

```
{
FILTER regex(?name, "Smith")
?x foaf:name ?name .
?x foaf:mbox ?mbox .
}
```

SPARQL

Область действия фильтров

```
{
 ?x foaf:name ?name .
 FILTER regex(?name, "Smith")
 ?x foaf:mbox ?mbox .
 }
```

SPARQL

Включение необязательных значений

Необязательные составляющие графового шаблона могут быть определены синтаксически при помощи ключевого слова **OPTIONAL**, применимого к графовому шаблону:

Синтаксическая форма:
{ OPTIONAL { *pattern* } }

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
 WHERE
 {
 ?x foaf:name ?name .
 OPTIONAL
{
 ?x foaf:mbox ?mbox
}
}
```

SPARQL

Включение необязательных значений

Данные

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
@prefix rdf:        <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
_:a   rdf:type      foaf:Person .
 _:a       foaf:name    "Alice" .
 _:a       foaf:mbox    <mailto:alice@example.com> .
_:a  foaf:mbox    <mailto:alice@work.example> .
_:b  rdf:type     foaf:Person .
_:b  foaf:name    "Bob" .
```

SPARQL

Включение необязательных значений

Результат

name	mbox
"Alice"	<mailto:alice@example.com>
"Alice"	<mailto:alice@work.example>
"Bob"	

В решении отсутствует значение **mbox** там, где **name** равно **"Bob".**

SPARQL

Ограничения в необязательном шаблоне

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title ?price
WHERE
 { ?
x dc:title ?title .
OPTIONAL
{ ?x ns:price ?price . FILTER (?price < 30) }
 }
```

SPARQL

Данные

```
@prefix dc:  <http://purl.org/dc/elements/1.1/> .
 @prefix :      <http://example.org/book/> .
 @prefix ns:  <http://example.org/ns#> .
:book1   dc:title   "SPARQL Tutorial" .
:book1   ns:price       42 .
:book2    dc:title  "The Semantic Web" .
 :book2   ns:price      23 .
```

Результат

title	price
"SPARQL Tutorial"	
"The Semantic Web"	23

Значение **price** для книги с **title**, соответствующим "**SPARQL Tutorial**", не отображается, потому как необязательный графовый шаблон не приводит к решению, которое бы содержало переменную **"price".**

SPARQL

Множественные необязательные графовые шаблоны

Запрос

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox ?hpage
 WHERE
{
?x foaf:name ?name .
OPTIONAL { ?x foaf:mbox ?mbox } .
OPTIONAL { ?x foaf:homepage ?hpage }
}
```

SPARQL

Данные

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
 _:a       foaf:name     "Alice" .
 _:a  foaf:homepage  <http://work.example.org/alice/> .
 _:b  foaf:name     "Bob" .
 _:b       foaf:mbox     <mailto:bob@work.example> .
```

Результат

name	mbox	hpage
"Alice"		<http://work.example.org/alice/>
"Bob"	<mailto:bob@work.example>	

SPARQL

Соответствие альтернативам

SPARQL предлагает средства компоновки графовых шаблонов так, чтобы поиск соответствия мог вестись для нескольких альтернативных графовых шаблонов.
Альтернативы шаблонов синтаксически определяются при помощи ключевого слова **UNION.**

```
PREFIX dc10:    <http://purl.org/dc/elements/1.0/>
PREFIX dc11:    <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE
 {
 { ?book dc10:title ?title }
UNION
{ ?book dc11:title ?title }
}
```

SPARQL

Данные

```
@prefix dc10:  <http://purl.org/dc/elements/1.0/> .
@prefix dc11:  <http://purl.org/dc/elements/1.1/> .
_:a  dc10:title  "SPARQL Query Language Tutorial" .
_:a  dc10:creator  "Alice" .
 _:b  dc11:title  "SPARQL Protocol Tutorial" .
 _:b  dc11:creator  "Bob" .
 _:c  dc10:title  "SPARQL" .
_:c  dc11:title  "SPARQL (updated)" .
```

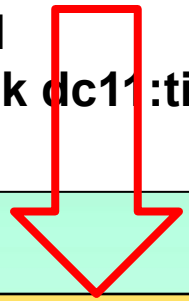
Результат

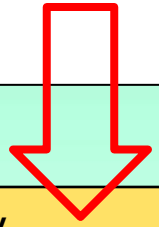
title
"SPARQL Protocol Tutorial"
"SPARQL"
"SPARQL (updated)"
"SPARQL Query Language Tutorial"

Данный запрос находит все названия (title) книг из исходных данных, где эти названия записаны с помощью свойств Dublin Core версии 1.0 или 1.1. Для того, чтобы определить, как именно была записана информация, запрос должен использовать различные переменные для двух альтернатив:

SPARQL

```
PREFIX dc10:    <http://purl.org/dc/elements/1.0/>
PREFIX dc11:    <http://purl.org/dc/elements/1.1/>
SELECT ?x ?y
WHERE
{
 { ?book dc10:title ?x }
UNION
{ ?book dc11:title ?y }
}
```





x	y
	"SPARQL (updated)"
	"SPARQL Protocol Tutorial"
"SPARQL"	
"SPARQL Query Language Tutorial"	

Этот запрос вернет результаты с переменной **x**, связанной с решениями из левой части от UNION, и **y**, связанной с решениями из правой части.

33

SPARQL

Шаблон **UNION** объединяет графовые шаблоны, причем каждая альтернатива может содержать более одного шаблона триплета:

```
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?title ?author
WHERE
 {
 { ?book dc10:title ?title .
 ?book dc10:creator ?author}
 UNION
{ ?book dc11:title ?title .
 ?book dc11:creator ?author }
}
```

SPARQL

Результат

author	title
"Alice"	"SPARQL Protocol Tutorial"
"Bob"	"SPARQL Query Language Tutorial"

Этот запрос будет соответствовать книге лишь в том случае, если ее предикаты, обозначающие как автора, так и название, будут взяты из одной версии Дублинского ядра.

SPARQL

Модификаторы последовательности решений

Шаблоны запроса генерируют неупорядоченную коллекцию решений.
Каждое решение является частичной функцией переменных от RDF-терминов.
Любые модификаторы последовательности используются для создания другой последовательности.
И эта последняя последовательность применяется для генерации результатов работы формы запроса SPARQL.

SPARQL

Модификаторы последовательности

- Модификатор последовательности (**Order by**): упорядочивает решения
- Модификатор проекции (**Projection**): выбирает определенные переменные
- Модификатор уникальности (**Distinct**): гарантирует уникальность решений в последовательности
- Модификатор сокращения (**Reduced**): разрешает элиминацию некоторых неуникальных решений
- Модификатор сдвига (**Offset)**: контролирует, где начинаются решения во всей последовательности решений
- Модификатором предела (**Limit**): ограничивает число решений

ORDER BY

Условие **ORDER BY** устанавливает порядок последовательности решений

За условием **ORDER BY** расположены компараторы последовательности, состоящие из выражения и необязательного модификатора последовательности **(ASC()** либо **DESC()**).
Каждый компаратор последовательности является либо **восходящим** (**ascending**), — обозначается модификатором **ASC()** или отсутствием модификатора — либо **нисходящим (descending)** — обозначается модификатором **DESC().**

SPARQL

ORDER BY

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
{
 ?x foaf:name ?name.
}
ORDER BY ?name
```

```
PREFIX : <http://example.org/ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?name
 WHERE { ?x foaf:name ?name ;
                :empId ?emp }
ORDER BY DESC(?emp)
```

SPARQL

ORDER BY

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
 WHERE
{
?x foaf:name ?name ;
    :empId ?emp
}
 ORDER BY ?name DESC(?emp)
```

SPARQL

Повторные решения

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
{ ?x foaf:name ?name }
```

```
@prefix foaf:      <http://xmlns.com/foaf/0.1/> .
 _:x       foaf:name     "Alice" .
 _:x       foaf:mbox    mailto:alice@example.com.

 _:y  foaf:name   "Alice" .
_:y  foaf:mbox    <mailto:asmith@example.com> .

_:z  foaf:name     "Alice" .
_:z  foaf:mbox    <mailto:alice.smith@example.com> .
```

SPARQL

Повторные решения

name
"Alice"
"Alice"
"Alice"

Модификатор решений **DISTINCT** позволяет ликвидировать повторные решения.
В частности, каждое решение, которое связывает одинаковые переменные и одинаковые RDF-термины в качестве еще одного решения, из набора решений удаляются.

SPARQL

DISTINCT

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?name
WHERE
{
?x foaf:name ?name
}
```

name
"Alice"

SPARQL

LIMIT

Условие **LIMIT** задает верхнюю границу числа возвращаемых решений.

Если фактическое число решений превышает лимит, то число возвращаемых решений не превысит заданный лимит.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
{
?x foaf:name ?name
}
LIMIT 20
```

Равенство **LIMIT 0** не возвратит никаких результатов.

Значение **LIMIT** не может быть отрицательным.

SPARQL

OFFSET

OFFSETпозволяет сгенерированным решениям начинаться после определенного числа решений. Равенство **OFFSET** нулю не возымеет никакого эффекта.
Использование **LIMIT** и **OFFSET** для выбора разных подмножеств решений запроса не имеет никакого смысла, если последовательность решений не предсказуема и не применен модификатор **ORDER BY**.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
 {
 ?x foaf:name ?name
}
ORDER BY ?name
LIMIT 5
OFFSET 10
```

SPARQL

Формы запросов

В **SPARQL** существует четыре формы запросов.
Эти формы запроса используют решения исходя из соответствия шаблону для формирования результирующих наборов или RDF-графов.

SELECT

Возвращает все или подмножество переменных, связанных с соответствующим шаблоном запроса.

CONSTRUCT

Возвращает RDF-граф, созданный путем замены переменных в наборе шаблонов триплетов.

ASK

Возвращает булево значение, которое указывает, соответствует ли шаблон запроса.

DESCRIBE

Возвращает RDF-граф, описывающий найденные ресурсы.

SPARQL

CONSTRUCT

Форма запроса **CONSTRUCT** возвращает **один RDF-граф**, определенный графовым шаблоном.

В результате получается **RDF-граф**, сформированный путем взятия каждого решения запроса из последовательности решений.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:    <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT { <http://example.org/person#Alice> vcard:FN ?name }
WHERE { ?x foaf:name ?name }
```

создает свойства vcard из информации FOAF:

```
@prefix vcard:    <http://www.w3.org/2001/vcard-rdf/3.0#> .
<http://example.org/person#Alice>    vcard:FN    "Alice" .
```

SPARQL

CONSTRUCT

```
@prefix foaf:      <http://xmlns.com/foaf/0.1/> .
_:a   foaf:name    "Alice" .
 _:a       foaf:mbox    <mailto:alice@example.org> .
```

Данные для предыдущего запроса:

SPARQL

ASK

Приложения могут использовать форму **ASK** для тестирования, **имеет ли шаблон запроса решение**.
Никакая информация о возможных решениях запроса не возвращается, просто выполняется проверка наличия решения.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
ASK
{
?x foaf:name "Alice"
}
```

```
prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Alice" .
_:a foaf:homepage <http://work.example.org/alice/> .
_:b foaf:name "Bob" .
_:b foaf:mbox <mailto:bob@work.example> .
```

yes

SPARQL

```
PREFIX foaf:     <http://xmlns.com/foaf/0.1/>
ASK
 {
?x   foaf:name    "Alice" ;
     foaf:mbox    mailto:alice@work.example.
}
```

С теми же данными этот запрос возвращает отсутствие соответствия (**no**) , так как **mbox** Алисы в данных отсутствует.

SPARQL

DESCRIBE

Форма **DESCRIBE** возвращает единственный результирующий **RDF-граф**, который содержит **RDF-данные о ресурсах**. Описание определяется сервисом запросов**.**

Синтаксис **DESCRIBE *** является сокращением и обозначает использование всех переменных в запросе**.**

Самый простой запрос **DESCRIBE** представляет собой просто **IRI** в условии **DESCRIBE:**

DESCRIBE <http://example.org/>

SPARQL

DESCRIBE

Идентификация ресурсов

Ресурсы, которые необходимо описать, могут быть взяты из привязок к переменной запроса в результирующем наборе.
Это делает возможным описание ресурсов, каким бы образом они ни были определены (с помощью IRI или безымянных вершин в наборе данных):

```
PREFIX foaf:      <http://xmlns.com/foaf/0.1/> DESCRIBE
?x
WHERE { ?x  foaf:mbox   <mailto:alice@org> }
```

Этот запрос возвратит информацию максимум об одном человеке.

SPARQL

DESCRIBE

Идентификация ресурсов

Однако, если шаблон запроса имеет множество решений, тогда **RDF-данными** для каждого будет объединение всех описаний **RDF-графов.**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
DESCRIBE ?x
 WHERE { ?x foaf:name "Alice" }
```

Может быть задано более одной переменной или IRI:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
DESCRIBE ?x ?y <http://example.org/>
WHERE {?x foaf:knows ?y}
```

SPARQL

Описания ресурсов

Возвращаемые **RDF-данные** определяются тем, кто публикует информацию.
Это та полезная информация о ресурсе, которой обладает сервис.
Она может включать информацию о других ресурсах

Простой запрос об **ID** сотрудника

```
PREFIX ent:      <http://org.example.com/employees#>
DESCRIBE ?x
WHERE { ?x ent:employeeId "1234" }
```

может возвратить описание работника и некоторые другие потенциально полезные детали:

SPARQL

Описания ресурсов

```
@prefix foaf:       http://xmlns.com/foaf/0.1/ .
 @prefix vcard:  <http://www.w3.org/2001/vcard-rdf/3.0> .
 @prefix exOrg: <http://org.example.com/employees#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> . @prefix owl:
http://www.w3.org/2002/07/owl#.
 _:a       exOrg:employeeId   "1234" ;
      foaf:mbox_sha1sum "ABCD1234" ;
       vcard:N [ vcard:Family "Smith" ;
                 vcard:Given "John" ] .
foaf:mbox_sha1sum rdf:type owl:InverseFunctionalProperty .
```

SPARQL

Запрос источника данных общего пользования

companyURL	corporation	founderURL	founderBirth	founderName
http://dbpedia.org/resource/Aircel_Comics	"Aircel Comics"@en	http://dbpedia.org/resource/Barry_Blair	"1954"^^<http://www.w3.org/2001/XMLSchema#gYear>	"Blair, Barry"@en
http://dbpedia.org/resource/Aircel_Comics	"Aircel Comics"@en	http://dbpedia.org/resource/Barry_Blair	"1954"^^<http://www.w3.org/2001/XMLSchema#gYear>	"Blair, Barry"@en

Запрос источника данных общего пользования

founderDescription	founderPicture
"Barry Blair (Ottawa, 1954 - 3 de janeiro de 2010) foi um quadrinista e escritor canadense, conhecido por fundar a Aircel Comics (editora de titulos como Samurai, Elflord, Dragonforce, e Men in Black) em 1980.== §Referências =="@pt	http://commons.wikimedia.org/wiki/Special:FilePath/Barryblair90s.png
"Barry Blair (1954 – January 3, 2010) was a Canadian comics publisher, artist and writer, known for launching Aircel Comics (publisher of titles such as Samurai, Elflord, Dragonforce, and Men in Black) in the 1980s. From early on, Blair's art style was influenced by the comics he had seen living in East Asia, at a time when manga and other Asian comics were largely unknown in North America. His art was typically characterized by childlike figures, and	http://commons.wikimedia.org/wiki/Special:FilePath/Barryblair90s.png

Запрос источника данных общего пользования

Последний запрос выдает нам все URL'ы компаний, их названия, URL-ы основателей, их имена, описания портреты и даты рождения.
Есть поля которые можно не выводить, например URL-ы компаний и URL-адреса основателей.
Можно указать имена переменных, значения которых необходимо вывести на экран. Для этого используется ключевое слово SELECT.

Запрос источника данных общего пользования

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbpprop: <http://dbpedia.org/property/>

SELECT ?corporation ?founderName ?founderPicture ?founderBirth ?founderDescription
WHERE
{
  ?companyURL rdf:type              dbpedia-owl:Company     ;
              dbpprop:companyName   ?corporation            ;
              dbpedia-owl:foundedBy ?founderURL             .
?founderURL dbpedia-owl:birthDate ?founderBirth           ;
              foaf:name             ?founderName      ;
              dbpedia-owl:abstract  ?founderDescription ;
              foaf:depiction        ?founderPicture     .

FILTER (?founderBirth > "1940-03-10"^^xsd:date )          .
  { ?founderURL rdf:type  dbpedia-owl:Artist. }
  UNION
  { ?founderURL rdf:type dbpedia-owl:Actor. }
  UNION
  { ?founderURL rdf:type dbpedia-owl:MartialArtist. }
  UNION
  { ?founderURL dbpedia-owl:occupation dbp:Martial_arts.}
}
LIMIT 100
```


60

Запрос источника данных общего пользования

corporation	founderName	founderPicture	founderBirth	founderDescription
"Aircel Comics"@en	"Blair, Barry"@en	http://commons.wikimedia.org/wiki/Special:FilePath/Barryblair90s.png	"1954"^^<http://www.w3.org/2001/XMLSchema#gYear>	"Barry Blair (Ottawa, 1954 - 3 de janeiro de 2010) foi um quadrinista e escritor canadense, conhecido por fundar a Aircel Comics (editora de títulos como Samurai, Elflord, Dragonforce, e Men in Black) em 1980.== §Referências =="@pt
"Aircel Comics"@en	"Blair, Barry"@en	http://commons.wikimedia.org/wiki/Special:FilePath/Barryblair90s.png	"1954"^^<http://www.w3.org/2001/XMLSchema#gYear>	"Barry Blair (1954 – January 3, 2010) was a Canadian comics publisher, artist and writer, known for launching Aircel Comics (publisher of titles such as Samurai, Elflord, Dragonforce, and Men in Black) in the 1980s. From early on, Blair's art style was influenced by the comics he had seen living in East Asia, at a time when manga and other Asian comics were largely unknown in North America. His art was typically characterized by

Запрос источника данных общего пользования

Фильтрация строковых литералов по языку

Запустив этот запрос на выполнение, можно увидеть большое количество дубликатов. Дублирование связано, в первую очередь, с тем, что одна и та же компания может иметь разные языковые тэги (информация дублируется на французском немецком и др. языках).
Для того, чтобы оставить в ответе на запрос только один язык используем оператор фильтрации для переменных `?corporation` и `?founderDescription` (только английский язык)

62

Запрос источника данных общего пользования

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbpprop: <http://dbpedia.org/property/>

SELECT ?corporation ?founderName ?founderPicture ?founderBirth ?founderDescription
WHERE
{
  ?companyURL rdf:type              dbpedia-owl:Company      ;
              dbpprop:companyName   ?corporation             ;
              dbpedia-owl:foundedBy ?founderURL              .
?founderURL dbpedia-owl:birthDate ?founderBirth              ;
              foaf:name             ?founderName       ;
              dbpedia-owl:abstract  ?founderDescription ;
              foaf:depiction        ?founderPicture     .

FILTER (?founderBirth > "1940-03-10"^^xsd:date )          .
  FILTER langMatches( lang(?corporation), "EN" )        .
  FILTER langMatches( lang(?founderDescription), "EN" )  .

  { ?founderURL rdf:type  dbpedia-owl:Artist. }
  UNION
  { ?founderURL rdf:type dbpedia-owl:Actor. }
  UNION
  { ?founderURL rdf:type dbpedia-owl:MartialArtist. }
  UNION
  { ?founderURL dbpedia-owl:occupation dbp:Martial_arts.}
}
LIMIT 100
```

Запрос источника данных общего пользования

corporation	founderName	founderPicture	founderBirth	founderDescription
"Aircel Comics"@en	"Barry Blair"@en	http://commons.wikimedia.org/wiki/Special:FilePath/Barryblair90s.png	"1954"^^<http://www.w3.org/2001/XMLSchema#gYear>	"Barry Blair (1954 – January 3, 2010) was a Canadian comics publisher, artist and writer, known for launching Aircel Comics (publisher of titles such as Samurai, Elflord, Dragonforce, and Men in Black) in the 1980s. From early on, Blair's art style was influenced by the comics he had seen living in East Asia, at a time when manga and other Asian comics were largely unknown in North America. His art was typically characterized by childlike figures, and included nudity and partial nudity. This continued into the erotica which became his main focus later in his career, and these attributes were a common criticism of his work."@en
				"Barry Blair (1954 – January 3,

Запрос источника данных общего пользования

```
PREFIX bpedia-owl: <http://dbpedia.org/ontology/>
SELECT ?Cat
{
  <http://dbpedia.org/resource/Cat> dbpedia-owl:abstract ?Cat
  FILTER(langMatches(lang(?Cat), "RU"))
}
```

Домашнее задание (контрольная работа)

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT *
WHERE {
        ?subject rdfs: свойство ?x
       }
```

На базе представленного SPARQL-запроса изучите различные свойства RDFS

Составьте SPARQL-запросы для изучения конструкций языка OWL Lite

http://dbpedia.org/sparql

Обеспечение читабельности результатов запроса

Как было указано ранее, запрос, который спрашивает, кто в данных адресной книги имеет телефонный номер (229) 276-5135:

```
# filename: ex008.rq
PREFIX  ab: <http://learningsparql.com/ns/addressbook#>
SELECT ?person
WHERE
{
?person ab:homeTel "(229)276-5135".
}
```

При запуске этого запроса к данным:

Обеспечение читабельности результатов запроса

```
# filename: ex012.ttl
@prefix ab: <http://learningsparql.com/ns/addressbook#> .
@prefix d: <http://learningsparql.com/ns/data#> .

d:i0432 ab:firstName   "Richard" .
d:i0432 ab:lastName    "Mutt" .
d:i0432 ab:homeTel     "(229) 276-5135" .
d:i0432 ab:email       "richard49@hotmail.com" .

d:i9771 ab:firstName   "Cindy" .
d:i9771 ab:lastName    "Marshall" .
d:i9771 ab:homeTel     "(245) 646-5488" .
d:i9771 ab:email       "cindym@gmail.com" .

d:i8301 ab:firstName   "Craig" .
d:i8301 ab:lastName    "Ellis" .
d:i8301 ab:email       "craigellis@yahoo.com" .
d:i8301 ab:email       "c.ellis@usairwaysgroup.com" .
```

Обеспечение читабельности результатов запроса

Обеспечивает получение следующего результата

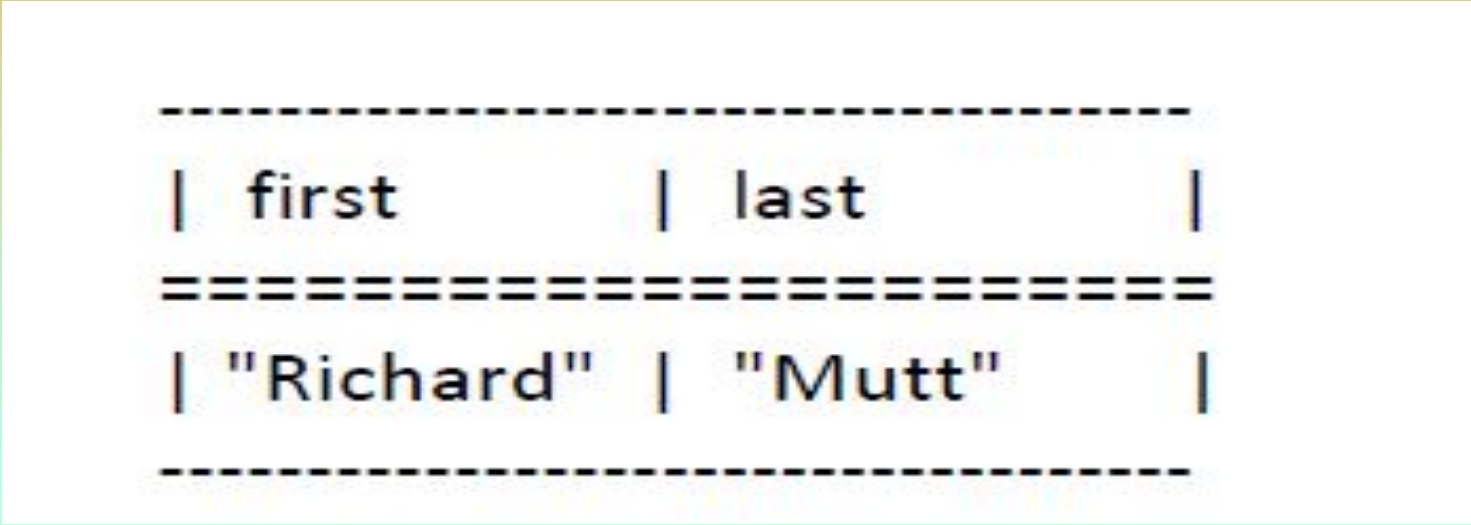
person
<http://learningsparql.com/ns/data#i0432>

Очевидно, что это не очень хороший ответ. Откорректируем запрос, для того, чтобы получить имя и фамилию человека с этим номером телефона:

Обеспечение читабельности результатов запроса

```
# filename: ex017.rq
PREFIX ab: <http://learningsparql.com/ns/addressbook#>
SELECT ?first ?last
WHERE
{
 ?person ab:homeTel "(229) 276-5135" .
 ?person ab:firstName ?first .
 ?person ab:lastName ?last .
}
```

Обеспечение читабельности результатов запроса



```
-----------------------------------
| first        | last             |
===================================
| "Richard"    | "Mutt"           |
-----------------------------------
```

Результат соответствует требованиям читабельности

Обеспечение читабельности результатов запроса

Точка с запятой в SPARQL запросе означает, следующее:

"Вот еще предикат и объект для данного субъекта".

Используя точку с запятой, построим следующий запрос, который будет работать точно так же, как в предыдущем случае :

```
# filename: ex047.rq
PREFIX ab: <http://learningsparql.com/ns/addressbook#>
SELECT ?first ?last
WHERE
{
 ?person ab:homeTel "(229) 276-5135" ;
         ab:firstName ?first ;
         ab:lastName ?last .
}
```

Использование меток, предусмотренных DBpedia

Ранее, мы рассматривали запрос на получение списка альбомов, произведенных Timbaland и исполнителей.

Результаты запроса фактически перечислили идентификаторы URI, которые представляли эти альбомы и исполнителей, т.е. результаты можно считать плохо читаемыми.

Можно улучшить читабельность результатов, отредактировав запрос с использованием значения меток rdfs:label:

Использование меток, предусмотренных DBpedia

```
# filename: ex048.rq
 PREFIX d: <http://dbpedia.org/ontology/>
SELECT ?artistName ?albumName
WHERE
{
album d:producer :Timbaland .
?album d:musicalArtist ?artist .
?album rdfs:label ?albumName .
?artist rdfs:label ?artistName .
}
```

Использование меток, предусмотренных DBpedia

У каждого исполнителя и названия альбома имеется несколько значений rdfs:label с разными языковыми тегами, назначенными каждому значению, например "en" для английского и "de" (Deutsch) для немецкого языка.
(Название альбома отображается на английском языке, потому что он был выпущен под таким названием в этих странах).
Когда вводится этот запрос в SNORQL DBpedia интерфейсе, то результат выдает каждую комбинацию из них, начиная с нескольких для Мисси Эллиот "Back in the day".

Использование меток, предусмотренных DBpedia

SPARQL Explorer for http://dbpedia.org/sparql

SPARQL:

```
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#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbpedia2: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
```

```
PREFIX d: <http://dbpedia.org/ontology/>

SELECT ?artistName ?albumName WHERE {
  ?album d:producer :Timbaland .
  ?album d:musicalArtist ?artist .
  ?album rdfs:label ?albumName .
  ?artist rdfs:label ?artistName .
}
```

Results: Browse Go! Reset

SPARQL results:

artistName	albumName
"Missy Elliott"@en	"Back in the Day (Missy Elliott song)"@en
"Missy Elliott"@en	"Back in the Day"@pl
"Missy Elliott"@de	"Back in the Day (Missy Elliott song)"@en
"Missy Elliott"@de	"Back in the Day"@pl
"Missy Elliott"@fi	"Back in the Day (Missy Elliott song)"@en
"Missy Elliott"@fi	"Back in the Day"@pl
"Missy Elliott"@es	"Back in the Day (Missy Elliott song)"@en
"Missy Elliott"@es	"Back in the Day"@pl
"Missy Elliott"@fr	"Back in the Day (Missy Elliott song)"@en
"Missy Elliott"@fr	"Back in the Day"@pl
"Missy Elliott"@it	"Back in the Day (Missy Elliott song)"@en
"Missy Elliott"@it	"Back in the Day"@pl

Использование меток, предусмотренных DBpedia

Использование Ключевого слова FILTER позволит получить значения меток и названия альбомов только на английском языке:

```
# filename: ex049.rq PREFIX d: <http://dbpedia.org/ontology/>
SELECT ?artistName ?albumName
WHERE
{
?album d:producer :Timbaland .
?album d:musicalArtist ?artist .
?album rdfs:label ?albumName .
?artist rdfs:label ?artistName .
FILTER ( lang(?artistName) = "en" )
FILTER ( lang(?albumName) = "en" )
}
```

Использование меток, предусмотренных DBpedia

SPARQL results:

artistName	albumName
"Missy Elliott"@en	"Back in the Day (Missy Elliott song)"@en
"Bobby Valentino"@en	"Anonymous (Bobby Valentino song)"@en
"Justin Timberlake"@en	"Cry Me a River (Justin Timberlake song)"@en
"Jay-Z"@en	"Big Pimpin"@en
"Brandy (entertainer)"@en	"Who Is She 2 U"@en
"Jay-Z"@en	"Dirt off Your Shoulder"@en
"Björk"@en	"Earth Intruders"@en
"Missy Elliott"@en	"One Minute Man"@en
"Ginuwine"@en	"Pony (Ginuwine song)"@en
"Björk"@en	"[illegible] (Björk song)"@en

SPARQL. Запрос данных

Переменная указывает поисковой машине, что ее устроит триплет удовлетворяющей шаблону с любым значением в этой позиции.

Все эти значения будут запомнены в переменной ?craigEmail, так что можно использовать их в другом месте этого запроса.

SPARQL. Запрос данных

```
# filename: ex003.rq
PREFIX ab: http://learningsparql.com/ns/addressbook#
SELECT ?craigEmail
WHERE
{ ab:craig ab:email ?craigEmail . }
```

Данный запрос запрашивает любой ab:email, связанный с ресурсом ab:craig.
Это означает поиск любых электронных адресов Крейга.

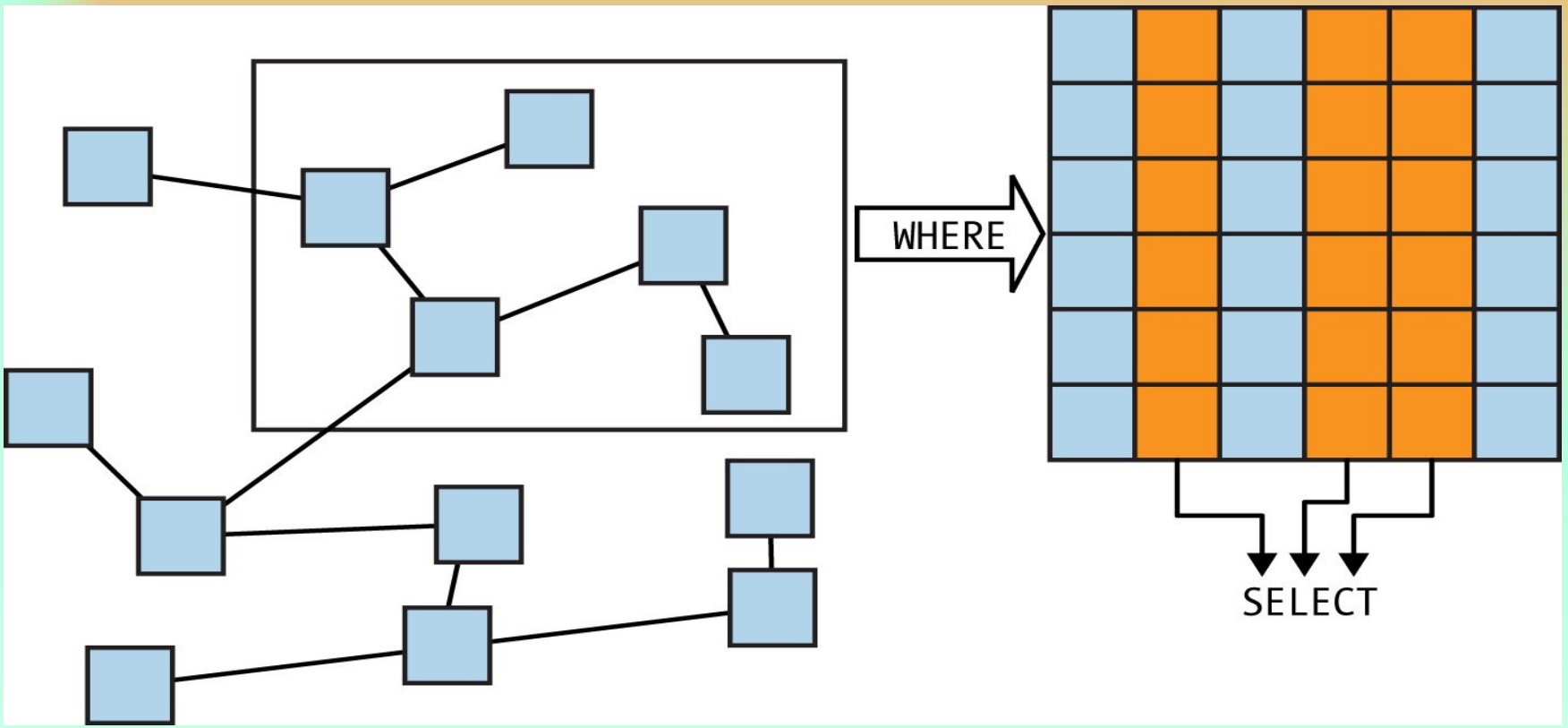
В наборе триплетов данных или триплетов запросов, точка после последнего не является обязательной. Тем не менее ставить ее – хорошая привычка (потом проще добавлять новые шаблоны).

SPARQL. Запрос данных

```
#  filename:  ex003.rq
PREFIX  ab:  http://learningsparql.com/ns/addressbook#
SELECT  ?craigEmail
WHERE
{  ab:craig  ab:email  ?craigEmail  .  }
```

В запросе SPARQL, ключевое слово WHERE указывает «извлечь эти данные из набора данных», а SELECT «выбрать для извлекаемых данных указанные имена».

SPARQL. Запрос данных



WHERE специфицирует извлекаемые данные;
SELECT выбирает
данные для отображения

SPARQL. Запрос данных

```
#  filename:  ex003.rq
PREFIX  ab:  http://learningsparql.com/ns/addressbook#
SELECT  ?craigEmail
WHERE
{  ab:craig  ab:email  ?craigEmail  .  }
```

Какую информацию запрос приведенный выше извлечет из троек данных?
Все, что соответствует (удовлетворяет) требованиям его триплета.
Вся информация, что будет получена в качестве результата выполнения запроса помещается в переменную ?CraigEmail.

SPARQL. Запрос данных

Как и в любым языке программирования или языке запросов, имя переменной должно давать подсказку о назначении переменной. Вместо имени ?CraigEmail, можно было выбрать имя
?Zxzwzyx, что сделало бы запрос более трудным для его понимания человеком.

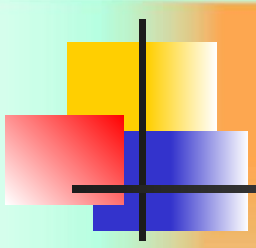
Как этот запрос будет выполняться?
Существует SPARQL – процессор (или SPARQL – движок), который применяет запрос к информационному файлу и визуализирует результат работы.

SPARQL процессор

Термины SPARQL процессор и SPARQL движок, имеют в виду одно и то же - программу, которая может применить SPARQL запрос к набору данных и выдать результат.

Для запросов к файлу данных на вашем жестком диске, можно использовать свободно распространяемую Java-программу ARQ. ARQ является частью Apache Jena framework, поэтому ее можно загрузить с домашней страницы по адресу http://jena.apache.org/documentation/query и скачать бинарный файл, имя которого Apache-jena- *.zip.

Разархивирование создаст подкаталог с именем, похожим на имя файла ZIP; это ваш домашний каталог Jena.



SPARQL процессор

Пользователи Windows, найдут там arq.bat и sparql.bat скрипты в bat подкаталоге, а пользователи с системами Linux - arq и sparql shell сценарии в подкаталоге bin. (Первый из каждой пары позволяет использовать расширения ARQ).

Либо на Windows, либо на Linux, можно запустить ex003.rq запрос к данным ex002.ttl с помощью следующей команды:

arq - data ex002.ttl - query ex003.rq

SPARQL процессор

Формат вывода по умолчанию - ARQ показывает имя каждой выбранной переменной в верхней строке и линиями вокруг значений каждой переменной, с использованием дефиса, знаков равно и вертикальная линия:

```
-----------------------------------
-------------
| craigEmail                      |
=================================
| "c.ellis@usairwaysgroup.com"    |
| "craigellis@yahoo.com"
| ---------------------------------
---------------
```

SPARQL процессор

Следующий вариант ex003.rq запроса использует полные URI, для записи субъекта и предиката в триплете запроса, вместо того чтобы использовать префикс.
По сути, это тот же запрос, и получает такой же ответ от ARQ, что и в предыдущем случае.

```
# filename: ex006.rq
SELECT ?craigEmail
WHERE
{
   <http://learningsparql.com/ns/addressbook#craig>
   <http://learningsparql.com/ns/addressbook#email>
   ?craigEmail .
}
```

SPARQL процессор

Различия между этим запросом и первым демонстрируют две вещи:

- **использовать префиксы в запросе необязательно, но они могут сделать запрос более компактным и легким для чтения, чем тот, который использует полные URI. Когда используется полный URI, необходимо заключить его в скобки, чтобы показать, процессору что это URI.**
- **Пробелы не влияют на синтаксис SPARQL. Новый запрос использует возврат каретки, для разделения трех частей триплета и все равно работает отлично.**

SPARQL процессор

```
# filename: ex003.rq
PREFIX ab: http://learningsparql.com/ns/addressbook#
SELECT ?craigEmail
WHERE
{ ab:craig ab:email ?craigEmail . }
```

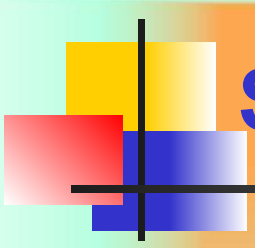
```
# filename: ex006.rq
SELECT ?craigEmail
WHERE
{
   <http://learningsparql.com/ns/addressbook#craig>
   <http://learningsparql.com/ns/addressbook#email>
   ?craigEmail .
}
```

Команда ARQ приведенная выше указывает набор данных для запроса в командной строке.

SPARQL процессор

SPARQL запрос с ключевым словом FROM позволяет указать набор данных для запроса как часть самого запроса.
Если опустить параметр --data ex002.ttl показанный в командной строки ARQ,
arq - data ex002.ttl - query ex003.rq
и использовать следующий запрос, вы получите тот же результат:

```
# filename: ex007.rq
PREFIX ab: <http://learningsparql.com/ns/addressbook
#>
SELECT ?craigEmail FROM <ex002.ttl>
WHERE
{ ab:craig ab:email ?craigEmail . }
```



SPARQL процессор

Угловые скобки вокруг "ex002.ttl" говорят процессору SPARQL использовать его в качестве URI.

Но, поскольку это просто имя файла, а не полный URI, ARQ предполагает, что это файл находящийся в той же папке, что и сам запрос.

SPARQL процессор

В запросах, которые рассмотрены до сих пор переменная была в позиции объекта триплета (третья позиция), но ее можно поместить в любой из трех позиций.

Например, кто-то позвонил мне по телефону (229) 276-5135, и я ничего не ответил. Я хочу знать, кто пытался позвонить мне, для этого необходимо создать следующий запрос для набора данных адресной книги, поставив переменную в субъектной позиции, вместо позиции объекта:

SPARQL процессор

```
# filename: ex008.rq
PREFIX ab: <http://learningsparql.com/ns/addressbook#>
SELECT ?person
WHERE
{ ?person ab:homeTel "(229) 276-5135" . }
```

При запуске этого запроса в ARQ к данным ex002.ttl адресной книги, получим следующий результат:

```
-----------------------
| person              |
=======================
| ab:richard          |
-----------------------
```

SPARQL процессор

Шаблоны триплетов в запросах часто имеют более чем одну переменную.
Например, можно перечислить все, что имеется в адресной книге о Синди с помощью следующего запроса.
В этом запросе переменная ?PropertyName помещена на позицию предиката, а переменная ?PropertyValue - на позицию объекта:

```
# filename: ex010.rq
PREFIX ab: <http://learningsparql.com/ns/addressbook
#>
SELECT ?propertyName ?propertyValue
 WHERE
{ ab:cindy ?propertyName ?propertyValue . }
```

SPARQL процессор

Пункт SELECT запроса спрашивает значения переменных ?propertyName и ?propertyValue, и ARQ показывает их в виде таблицы со столбцами для каждого:

```
-----------------------------------------------
---------------
| propertyName  |   propertyValue             |
=================================
| ab:email          |   "cindym@gmail.com"   |
| ab:homeTel    |   "(245) 646 5488"     |
 --------------------
```

SPARQL процессор

В большинстве данных RDF, субъектам троек не принято выступать в роли названий понятных для человеческого глаза, как например в ex002.ttl ab:richard и ab:cindy (имена ресурсов). Они должны играть роль идентификаторов, аналогично значениям уникального ID поля таблицы реляционной базы данных.

Вместо того чтобы хранить чье-то имя, как часть URI субъекта, как наш первый набор данных, более типично для RDF иметь для субъектов значения, которые составляют не удобочитаемый смысл, а выполняют роль уникальных идентификаторов.
В этом случае firstName и lastName будут храниться с использованием отдельных троек, как значения homeTel и email.

SPARQL процессор

Другая нереалистичная деталь ex002.ttl состоит в том, что идентификаторы ресурсов, такие как ab:richard и имена свойств, такие как ab:homeTel, берутся из одного и того же пространства имен http://learningsparql.com/ns/addressbook#, которое представляет префикс ab.
Словарь имен свойств, как правило, имеет свое собственное пространство имен, чтобы упростить его использование его с другими наборами данных.

Если пересмотреть данный пример, чтобы использовать реалистичные
идентификаторы ресурсов, чтобы сохранить имена и фамилии в качестве значений свойств и поставить значения данных в собственном пространстве имен отдельно от http://learningsparql.com/ns/data#, получим следующий набор данных:

SPARQL процессор

```
# filename: ex012.ttl
@prefix ab: <http://learningsparql.com/ns/addressbook#>
 .
@prefix d: <http://learningsparql.com/ns/data#> .
d:i0432 ab:firstName "Richard" .
d:i0432 ab:lastName "Mutt" .
d:i0432 ab:homeTel "(229) 276--5135" .
d:i0432 ab:email "richard49@hotmail.com" .
d:i9771 ab:firstName "Cindy" .
d:i9771 ab:lastName "Marshall" .
d:i9771 ab:homeTel "(245) 646--5488" .
d:i9771 ab:email "cindym@gmail.com" .
d:i8301 ab:firstName "Craig" .
d:i8301 ab:lastName "Ellis" .
d:i8301 ab:email "craigellis@yahoo.com" .
d:i8301 ab:email "c.ellis@usairwaysgroup.com" .
```

SPARQL процессор

Запрос, для нахождения адреса электронной почты Крейга будет выглядеть так:

```
# filename: ex013.rq
PREFIX ab: <http://learningsparql.com/ns/addressbook#>
SELECT ?craigEmail
WHERE
{
 ?person ab:firstName "Craig" .
 ?person ab:email ?craigEmail .
}
```

SPARQL процессор

Предположим, что процессор SPARQL посмотрел все строки адресной книги и нашел совпадение в {ab:i8301 ab:firstName "Craig"}.
Это будет связывать ab: i8301 с переменной ?person, так как обе находятся в субъектной позиции и остальные части триплетов совпадают.

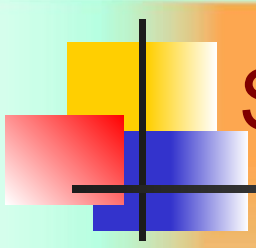
Для запросов этого типа (ex013.rq, которые имеют более одного триплета), когда процессор находит одно совпадение он переходит к другим триплетам, чтобы найти другие совпадения, с учетом предыдущего.
Так в нашем случае выясняется, что две тройки в ex012.ttl имеющие в качестве субъекта d:i8301, а в качестве предиката ab:email возвратят два значения для переменной ?craigEmail:

SPARQL процессор

```
-----------------------------------------------------
---------
|  craigEmail                                      |
====================================
|  "c.ellis@usairwaysgroup.com"                    |
|  "craigellis@yahoo.com"                          |
 ------------------
```

SPARQL процессор

Множество триплетов записанных между фигурными скобками в запросе SPARQL называется графовым шаблоном. Граф в данном случае является техническим термином для набора RDF троек.
Существуют утилиты набора RDF троек в «графовую картинку», это не относится к графике в визуальном смысле. В «графовой картинке» RDF, узлы представляют субъект или объект ресурсов, и предикаты соединения между этими узлами.



SPARQL процессор

В ex013.rq запросе переменная ?person использована в двух разных триплетах, чтобы найти связанные тройки в данных.
В более сложных запросах, это техника с использованием переменной для связывания различных троек данных становится все более распространенной.
В случаях, когда необходимо составление запросов к данным, которые поступают из нескольких источников, эта способность, находить связи между тройками из различных источников является одним из лучших особенностей SPARQL.

SPARQL процессор

Если бы адресная книга включала не одного абонента по имени Крейг, причем необходимо найти адреса электронной почты именно Крейг Эллис, то к шаблону запросу необходимо добавить еще один триплет :

```
# filename: ex015.rq
PREFIX ab: <http://learningsparql.com/ns/addressbook#>
SELECT ?craigEmail
WHERE
{
 ?person ab:firstName "Craig" .
 ?person ab:lastName "Ellis" .
 ?person ab:email ?craigEmail .
}
```

SPARQL процессор

Теперь предположим, что телефон показал, что кто-то звонит с номера телефона "(229) 276-5135".
Для определения имени звонившего используем запрос ex008.rq, который использовали прежде, но на этот раз к данным из ex012.ttl. Результат покажет субъект тройки у которой предикат ab:homeTel и объект "(229) 276-5135":

```
# filename: ex008.rq
PREFIX ab: <http://learningsparql.com/ns/addressbook#>
SELECT ?person
WHERE
{ ?person ab:homeTel "(229) 276-5135" . }
```

SPARQL процессор

```
# filename: ex012.ttl
@prefix ab: <http://learningsparql.com/ns/addressbook#>
 .
@prefix d: <http://learningsparql.com/ns/data#> .
d:i0432 ab:firstName "Richard" .
d:i0432 ab:lastName "Mutt" .
d:i0432 ab:homeTel "(229) 276--5135" .
d:i0432 ab:email "richard49@hotmail.com" .
d:i9771 ab:firstName "Cindy" .
d:i9771 ab:lastName "Marshall" .
d:i9771 ab:homeTel "(245) 646--5488" .
d:i9771 ab:email "cindym@gmail.com" .
d:i8301 ab:firstName "Craig" .
d:i8301 ab:lastName "Ellis" .
d:i8301 ab:email "craigellis@yahoo.com" .
d:i8301 ab:email "c.ellis@usairwaysgroup.com" .
```

SPARQL процессор

```
-------------------------------------------------------------------------
| person                                                                |
=========================================================================
| <http://learningsparql.com/ns/data#i0432>                             |
-------------------------------------------------------------------------
```

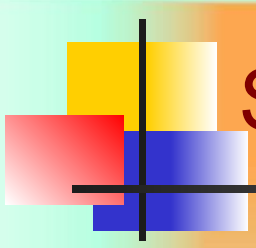
Чтобы получить имя и фамилию звонившего с этого номера телефона, необходимо сформировать следующий запрос :

SPARQL процессор

```
# filename: ex017.rq
PREFIX ab: <http://learningsparql.com/ns/addressbook#>
SELECT ?first ?last
WHERE
{
 ?person ab:homeTel "(229) 276-5135" .
 ?person ab:firstName ?first .
 ?person ab:lastName ?last .
}
```

ARQ вернет в этом случае то, что необходимо:

```
-------------------------
| first       | last      |
=========================
| "Richard"   | "Mutt"    |
-------------------------
```



SPARQL процессор

Рассмотрим формирование запроса на получение всех сведений о Синди в данных ex012.ttl: запросим все предикаты и объекты (хранимые в переменных ?propertyName и ?propertyValue), связанных с субъектом, который имеет ab:firstName “Cindy” и ab:lastName “Marshall”:

SPARQL процессор

```
# filename: ex019.rq
PREFIX a: <http://learningsparql.com/ns/addressbook#>
SELECT ?propertyName ?propertyValue
WHERE
{
?person a:firstName "Cindy" .
?person a:lastName "Marshall" .
?person ?propertyName ?propertyValue .
}
```

Обратите внимание, что значения ab:firstName и ab:lastName из файла ex012.ttl появляются в колонке PropertyValue. Другими словами, они получили связь с переменной PropertyValue, также как ab:email и ab:homeTel:

SPARQL процессор

```
-------------------------------------
| propertyName   | propertyValue                |
=====================================
| a:email              | cindym@gmail.com          |
| a:homeTel          | "(245)  646--5488"           |
| a:lastName         | "Marshall"                   |
| a:firstName        | "Cindy"                       |
-------------------------------------
```

Поиск подстрок

Что делать, если необходимо, проверить часть данных, но неизвестно, какой субъект или предикат может ее включать?
Следующий запрос имеет только один триплет и все три части переменные, так что он будет соответствовать каждой тройки в наборе данных. Однако, в результате не будут присутствовать все тройки, потому что сюда добавлен FILTER, который инструктирует процессор запросов пройти только вдоль троек,
которые отвечают определенному условию. В этом фильтре, условие задается с помощью регулярных выражений функцией regex(), которая проверяет соответствие строк, определенному в ней шаблону.

Поиск подстрок

```
# filename: ex021.rq
PREFIX ab: <http://learningsparql.com/ns/addressbook#>
SELECT *
WHERE
{
?s ?p ?o .
FILTER (regex(?o, "yahoo","i"))
}
```

Этот вызов функции regex() проверяет, имеет ли объект подстроку "yahoo" в любом месте.

Это общее соглашение SPARQL использовать имя переменной ?s для субъекта, ?p для предиката, а ?o для объекта.

В этом запросе использована звездочка вместо перечень конкретных переменных в SELECT. Это сокращенный способ сказать "выбрать все переменные, использованные в этом запросе."

Поиск подстрок

Процессор запросов найдет единственный триплет, имеющий "yahoo" в значении объекта:

```
-------------------------------------------------------------------------------
| s                                                 | p           | o                       |
===============================================================================
| <http://learningsparql.com/ns/data#i8301>   |   ab:email  |   "craigellis@yahoo.com"  |
-------------------------------------------------------------------------------
```

Возможные ошибки

Давайте изменим запрос ex015.rq, запросив кроме адреса электронной почты Крейга Эллиса также его домашний телефон. (Если просмотреть данные ex012.ttl, то можно увидеть, что именно у Крейг его нет.)

```
# filename: ex023.rq???
PREFIX ab: <http://learningsparql.com/ns/addressbook#>
SELECT ?craigEmail ?homeTel
WHERE
{???
?person ab:firstName "Craig" .
?person ab:lastName "Ellis" .
?person ab:email ?craigEmail .
?person ab:homeTel ?homeTel .
}
```

Возможные ошибки

Когда ARQ применит этот запрос к данным ex012.ttl, результат б включать заголовки для переменных, но данные будут отсутствовать:

```
-----------------------------
| craigEmail | homeTel |
=============================
-----------------------------
```

Почему? Запрос спросил процессор SPARQL об адресе электронной почты и номере телефона Крэйга Эллис, т.е. поставил задачу найти ресурс, который удовлетворяет всем четырем условиям, перечисленным в шаблоне графа. Но такого ресурса не нашлось, поэтому процессор SPARQL не возвращает никаких данных.