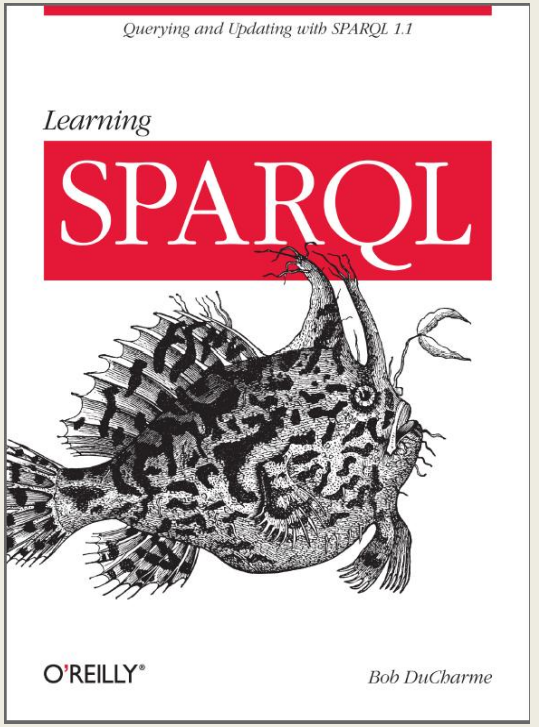
ОСНОВЫ SPARQL

Интеграция данных

SPARQL

SPARQL – SPARQL Protocol And RDF Query Language
Описывается спецификациями комитета W3C.

Язык запросов SPARQL для RDF хранилищ данных



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/

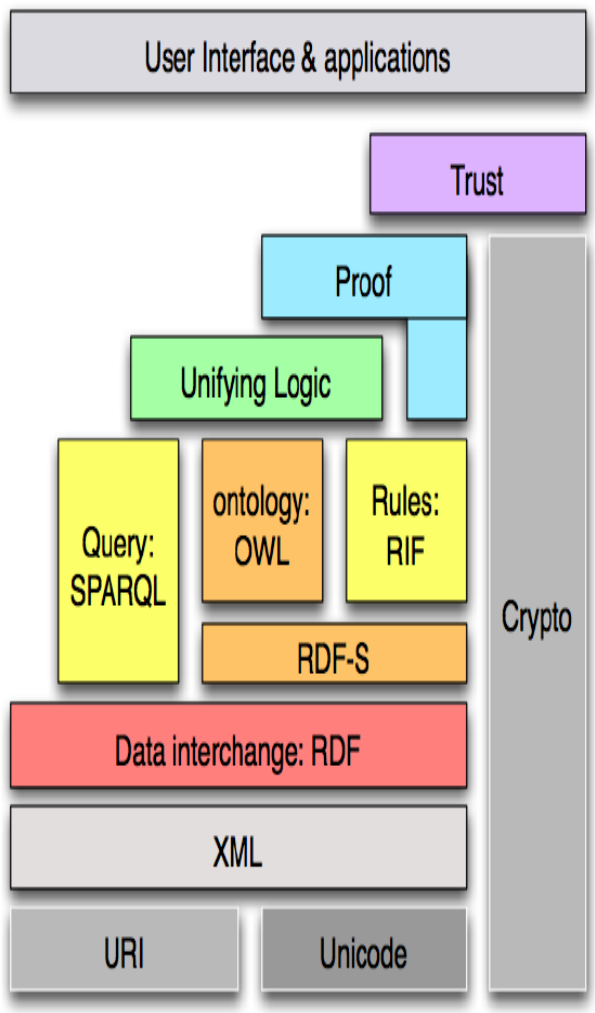
Хранение информации является важнейшей составляющей современных информационных технологий

SPARQL

(65 стр., © Губин А.Н., Филиппов Ф.В.
© ФГОБУВПО СПб ГУТ
им.проф. М.А. Бонч-Бруевича

Хранение информации является важнейшей составляющей **современных информационных технологий**

SPARQL



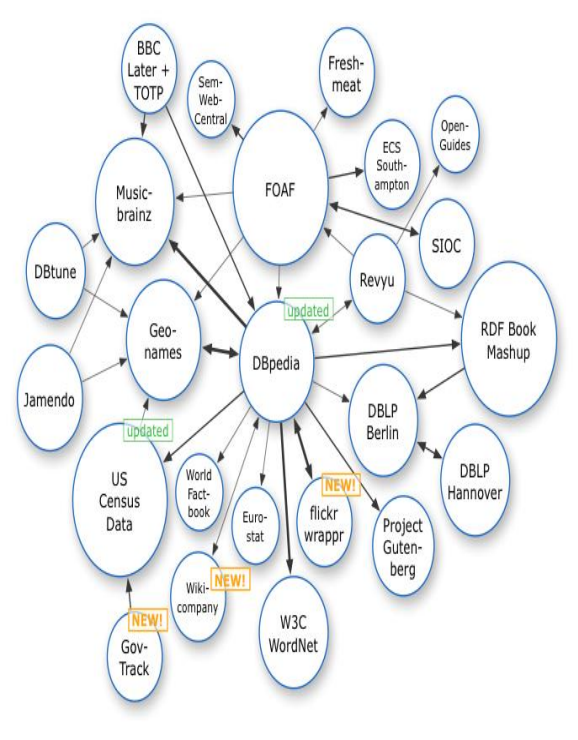
Общие вопросы. RDF. SPARQL

1) **RDF (Resource Description Framework**
2) **UNION** — позволяет объединять результаты различных шаблонов.
3) **REDUCED** — сокращает количество результатов.
4) **DISTINCT** — обеспечивает уникальность решений в ответе на запрос.
5) **ORDER BY** — позволяет отсортировать результаты.
6) **LIMIT** — задает максимальное количество выводимых результатов.
7) **FILTER** — позволяет выделить в результате интересующие данные.
8) **OFFSET** — опускает в результате первые n решений.

SPARQL

http://www.sut.ru/

RDF (Resource Description Framework) – это модель описания связанных данных, которая позволяет технологии Семантического веба **интерпретировать информацию, представленную в сети**

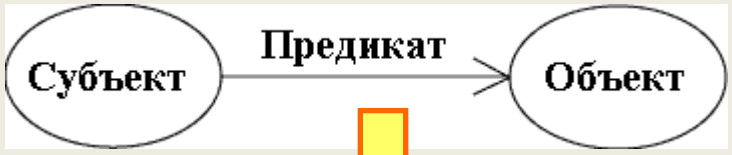


SPARQL

Общая идея, на которой основана модель RDF, заключается в следующем: всё, что существует в мире (будь то физический предмет или абстрактное понятие), имеет определенные свойства, а любое свойство имеет конкретные значения. Значит, описать любую сущность можно с помощью элементарных выражений, которые перечисляют эти свойства и определяют их значения.

SPARQL

Структура, лежащая в основе любых выражений в RDF, это коллекция триплетов, каждый из которых состоит из субъекта, предиката и объекта.



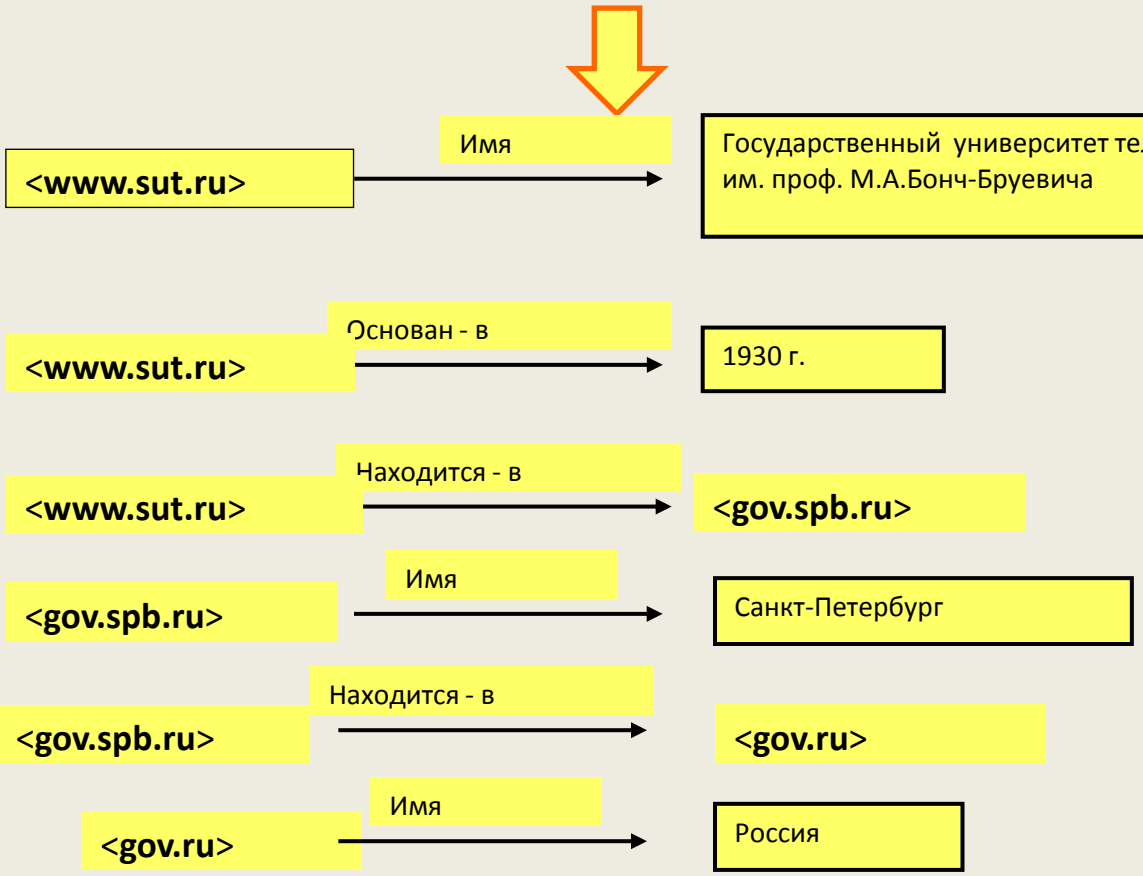
Набор таких триплетов называется RDF графом

Направление дуги имеет значение: оно всегда идет к объекту.

Субъект в данном случае является описываемым ресурсом, предикат – это наименование свойства ресурса, а объект – значение свойства описываемого ресурса

SPARQL

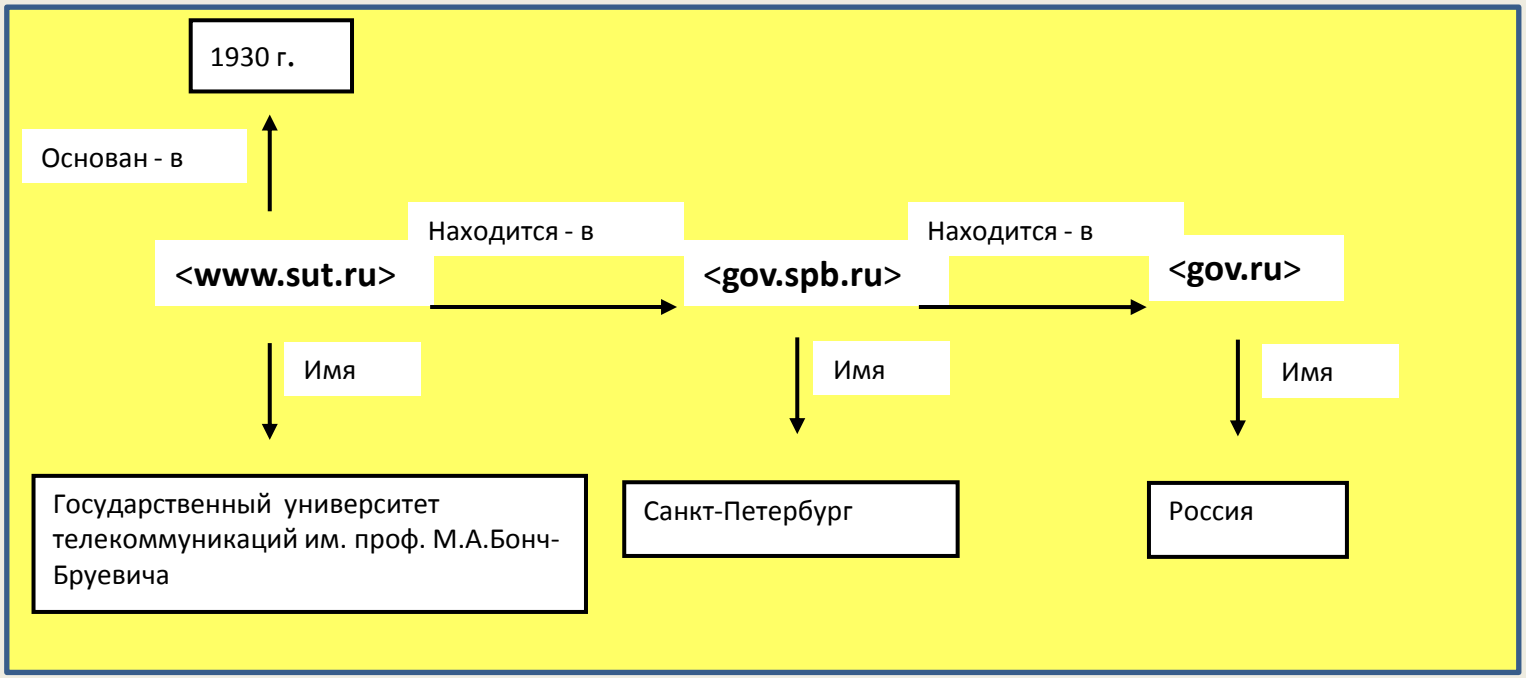
Описание реального объекта представляет собой совокупность триплетов а



Описание объекта СПбГУТ в виде совокупности триплетов

SPARQL

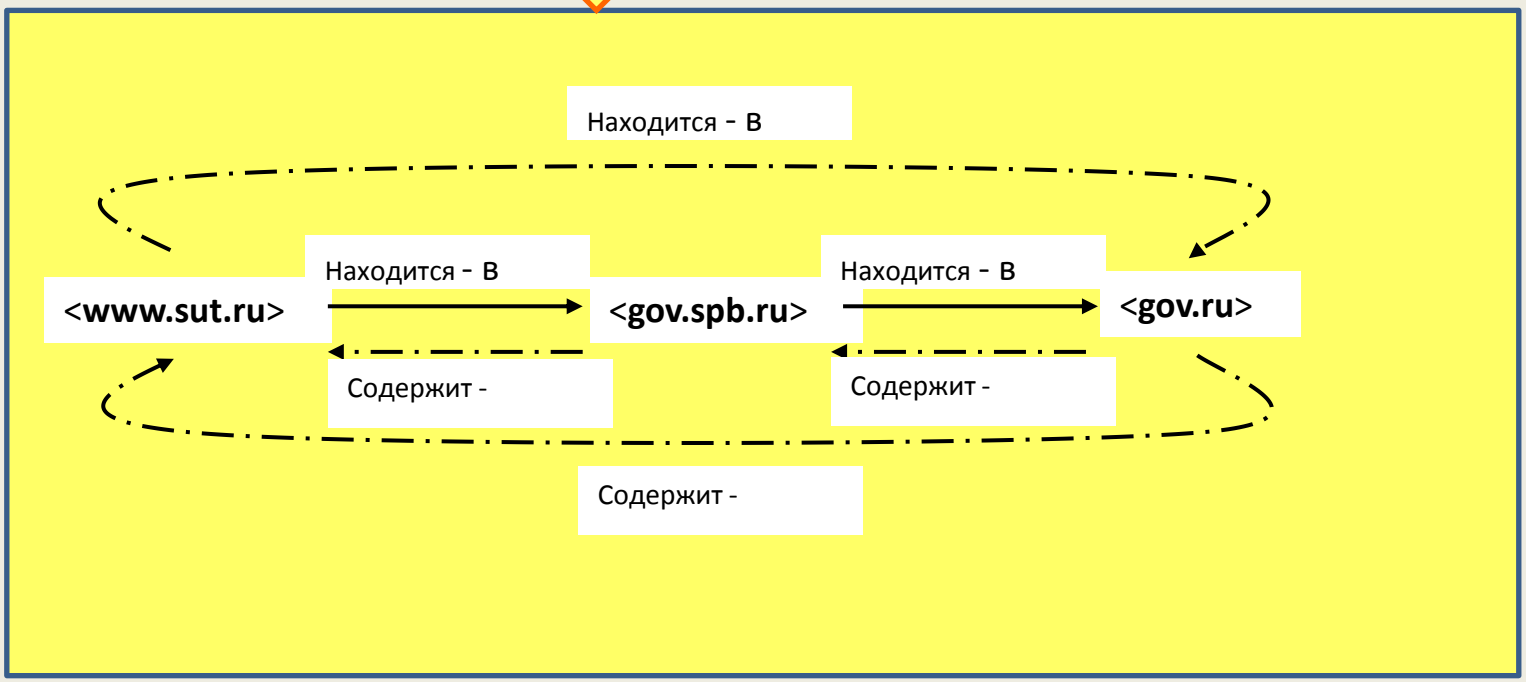
Все триплеты RDF документа объединяются в общий RDF – граф



Описание объекта СПбГУТ в виде RDF-графа

SPARQL

Объединенные RDF-графы позволяют с помощью логических выводов генерировать новые, несуществующие до этого триплеты



— · → новые свойства субъектов

Состав RDF – триплетов

SPARQL

Основными элементами RDF-триплета является **ресурс и его свойства**.
Наиболее рациональным способом указания вида необходимого клиенту ресурса является, например, ***имя файла, каталог,*** в котором хранится этот файл и ***имя сервера***, на котором хранится данная файловая система.

Очевидно, что **URL** могут быть использованы только по отношению к **электронным ресурсам.**

Комбинацию ***сервер-каталог-ресурс*** называют унифицированным локатором ресурса (**Uniform Resource Locator) или URL.** Например, http://www.learningsparql.com/resources/index.html является URL

Состав RDF – триплетов

SPARQL

Для обозначения ресурсов **реального мира** была разработана спецификация, определяющая универсальное имя ресурса (**Universal Resource Names) URN**, которое не связано с адресом расположения ресурса в интернете

Например, ***urn:isbn:9660313837*** – является URN ресурса **«Книга "Капитал. Том I"»,** определённого по коду ISBN, а ***urn:oid: 1.2.643*** – является URN ресурса **«Страна Россия»**, определённого по коду OID.

Недостатки

- не для всех типов ресурсов созданы схемы кодирования;
- нет данных о месте расположения ресурса, то есть, невозможно перейти по коду как по ссылке и получить информацию о ресурсе.

Состав RDF – триплетов

SPARQL

Для того, чтобы получить возможность использования как **электронных ресурсов**, так и **ресурсов мира вещей** в дальнейшем предложено использовать универсальный идентификатор ресурса **(Uniform Resource Identifier) URI**.

В общем случае, **URI** представляет собой **некую последовательность символов**, идентифицирующую абстрактный или физический ресурс

Недостатки
- необходимость использования при их записи **ограниченного набора только латинских символов и знаков препинания.**

Состав RDF – триплетов

SPARQL

Эту проблему позволяет решить **стандарт IRI** (Internationalized Resource Identifier) — международный идентификатор ресурсов, в котором можно использовать **символы Юникода**.

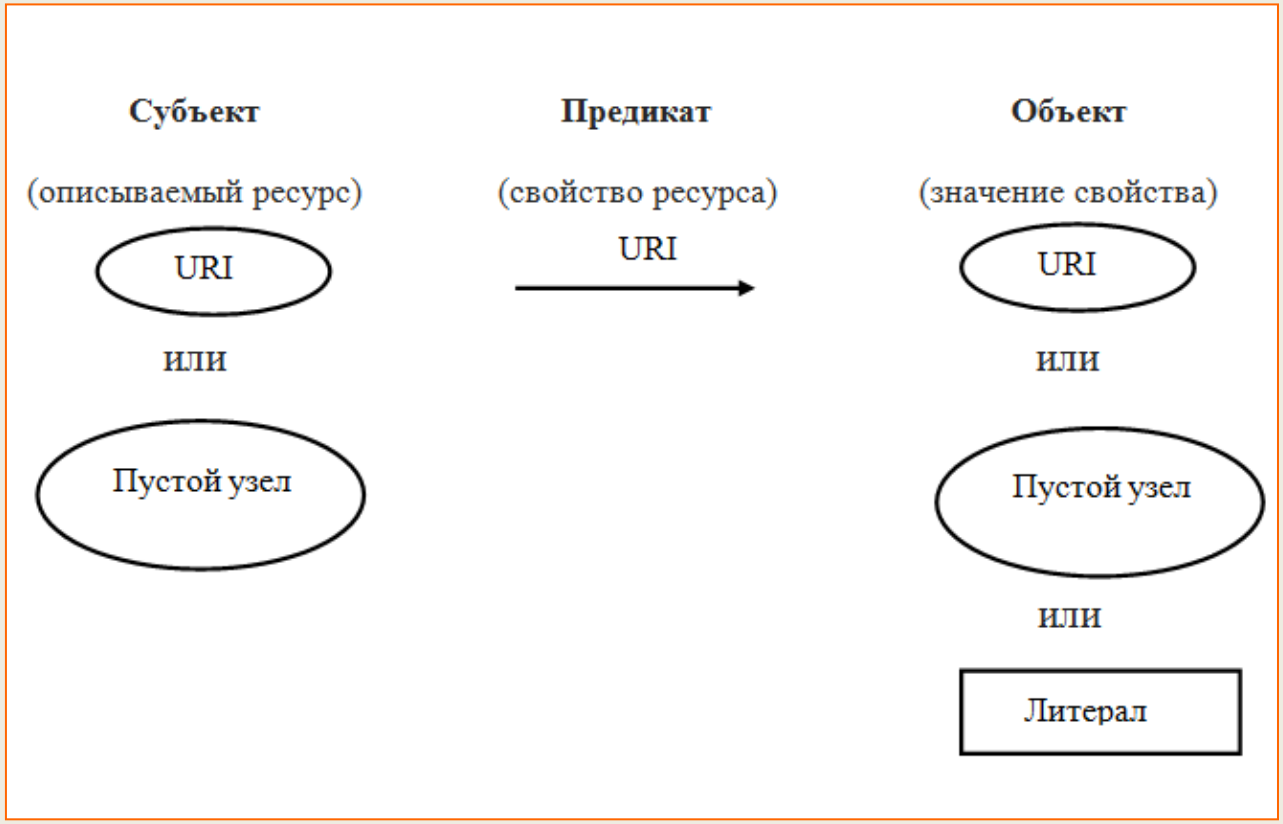
Язык запросов **SPARQL** ориентирован на использование **IRI,**

Достоинство позволяет использовать при формировании запросов и ответов **многоязычные формы описания ресурсов**, **не ущемляют права других языков.**

Состав RDF – триплетов

SPARQL

Возможные значения элементов триплета



Состав RDF – триплетов

SPARQL W3C

Пустые узлы в RDF документе, как правило, имеют "**локальные" идентификаторы,** на которые можно ссылаться только внутри данного документа..

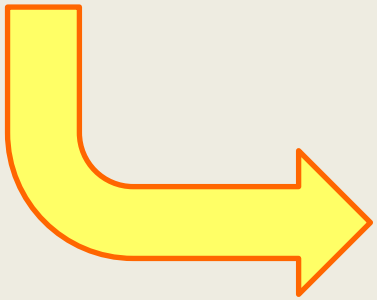
субъект **ab:i0432** имеет предикат **ab:address**, значение которого имеет особенный префикс пространства имен: подчеркивание. Это значение имеет свои собственные значения, описывающие отдельные компоненты адреса Ричарда.

```
# filename: ex041.ttl

@prefix ab: <http://learningsparql.com/ns/addressbook#> .


ab:i0432 ab:firstName "Richard" ;
         ab:lastName "Mutt" ;
         ab:homeTel "(229) 276---5135" ;
         ab:email "richard49@hotmail.com" ;
         ab:address _:b1 .


_:b1 ab:postalCode "49345" ;
     ab:city "Springfield" ;
     ab:streetAddress "32 Main St." ;
     ab:region "Connecticut"
```

Состав RDF – триплетов

SPARQL

Префикс с подчеркиванием означает, что это пустой узел. Его цель в данном случае состоит в выделении определенной группы значений предикатов и объектов.

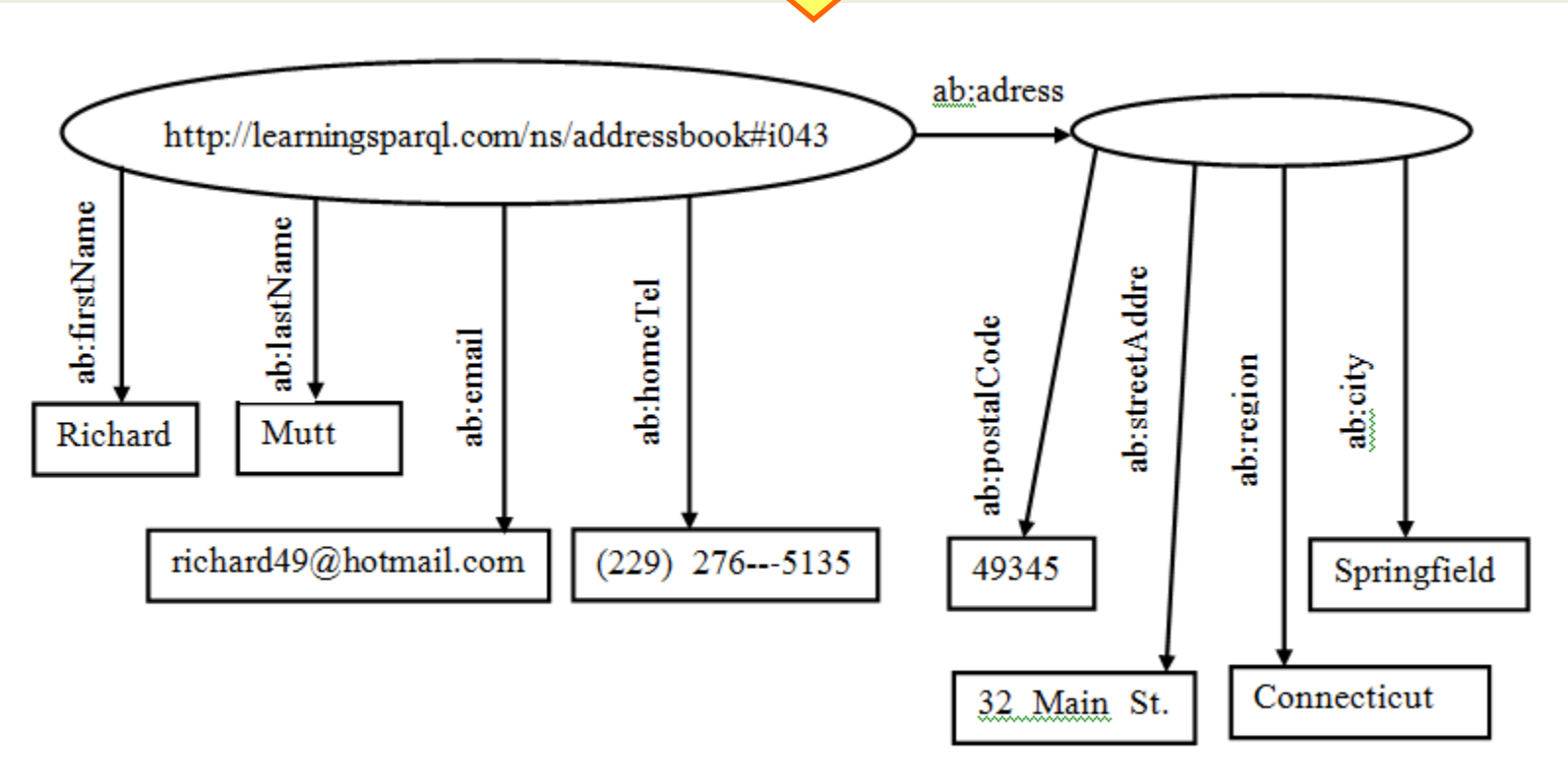


```
# filename:  ex041.ttl

@prefix  ab:  <http://learningsparql.com/ns/addressbook#>  .


ab:i0432  ab:firstName        "Richard"  ;
                  ab:lastName        "Mutt"  ;
                  ab:homeTel         "(229)  276---5135"  ;
                  ab:email         "richard49@hotmail.com"  ;
                  ab:address         _:b1  .

_:b1    ab:postalCode        "49345"  ;
                  ab:city         "Springfield"  ;
                  ab:streetAddress    "32  Main  St."  ;
                  ab:region         "Connecticut"
```

Состав RDF – триплетов

SPARQL

Граф RDF записи адресной книги с пустым узлом



Состав RDF – триплетов

SPARQL

Литеральные значения представляют собой просто текст, который может быть использован вместо объектов в RDF-триплетах. В отличии от имен (то есть URI), которые заменяют собой сущности в реальном мире, литеральные значения — это просто текстовые данные вставленные в граф.

Литеральные значения как правило дополняется одним из двух возможных элементов метаданных.

Первый это дескриптор языка, который указывает, на каком языке представлено текстовое значение литерала.
Выражение **'chat'@en** представляет собой литеральное значение **'chat'** с английским дескриптором языка, или **'chat'@fr** то же самое значение литерала, но с французским дескриптором языка.

Состав RDF – триплетов

SPARQL

Литералы с одинаковым текстом, но разными кодами языка не равны друг другу

Chat @en (болтовня) ≠ Chat @fr (кошка)

Состав RDF – триплетов

SPARQL

Кроме того, литеральные значения могут быть помечены с помощью URI, идентифицирующего тип данных (типизированные литералы). Тип данных указывает, как интерпретировать текст литерала: как число, URI, дату или время, и др. Типом данных может быть любой URI, но обычно для этого используются типы данных, определенные в XML Schema.

Состав RDF – триплетов

SPARQL

Типизированные литералы с разным текстом могут быть равны, если соотносятся с одним и тем же значением

45 ≠ +45 ≠ +45.00

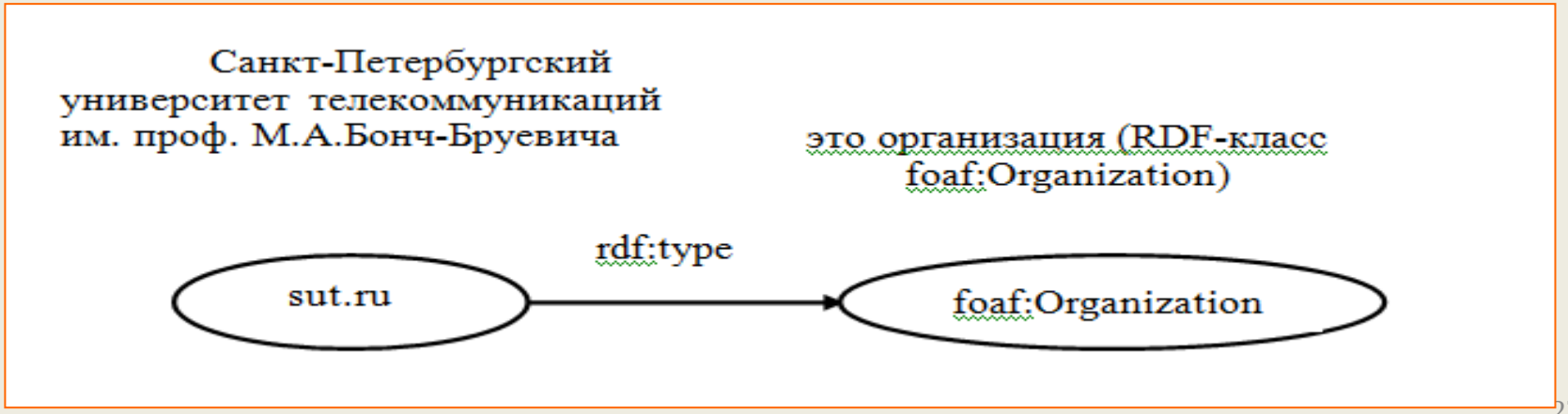
45 ^^xsd:date = +45 ^^xsd:date = +45.00 ^^xsd:date

RDF – классы

SPARQL

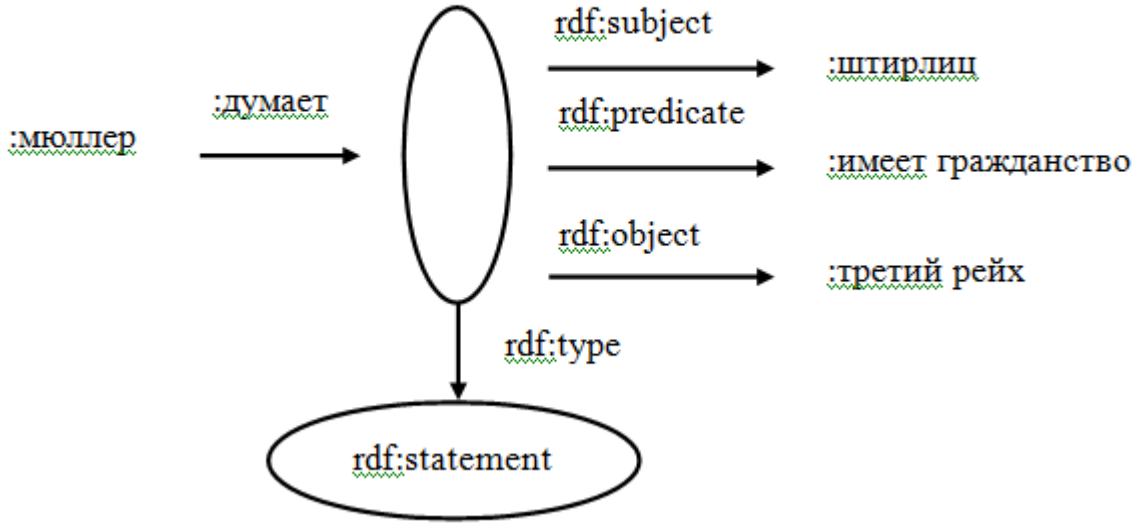
RDF содержит **несколько встроенных классов**, таких как ***rdf:Statement***- для задания **реификации**; *rdf:Bag, rdf:Seq, rdf:Alt, rdf:List* - для создания раличных контейнеров и списков и др.

Принадлежность ресурса классу задается встроенным предикатом ***rdf:type***



SPARQL

Реификация является **утверждением об утверждении**.
Для создания утверждения об утверждении создается ресурс, относящийся к классу ***rdf:statement*** и имеющий три свойства:
rdf:subject - субъект утверждения;
rdf:predicate - предикат;
rdf:object - объект.



Утверждение о том, что "Мюллер думает, что Штирлиц является гражданином Третьего рейха"

SPARQL

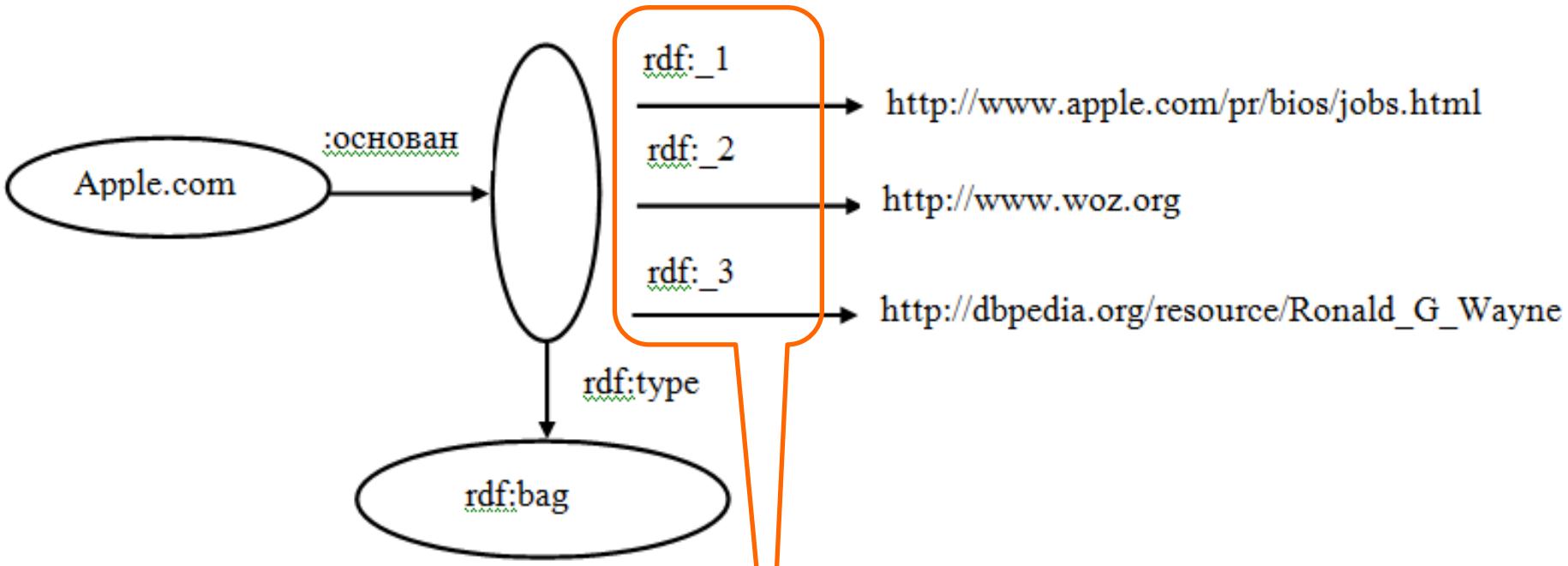
В RDF поддерживаются два вида базовых структур данных – **контейнеры** и **коллекции**. Эти средства RDF позволяют **группировать ресурсы** (и литералы) и использовать их в высказываниях как единое целое.

В RDF предусмотрена возможность использования следующих трех типов **контейнеров**

- **bag**— неупорядоченный набор (Apple основал Стив Джобс, Стив Возняк и Рональд Уэйн.);
- **alt**— набор возможных альтернатив (столицей XXIII Зимних Олимпийских игр будет Анси, Мюнхен или Пхёнчхан.);
- **seq**— упорядоченный набор (в случае отставки президента США его замещают по порядку: вице-президент, спикер Палаты представителей,

Использование контейнера типа Bag

SPARQL



Для указания принадлежности некоторых сущностей (элементов) к контейнеру в RDF используется специальное обозначение свойств ресурса-контейнера, имеющее общий вид **«rdf:_n»,** где n – произвольное десятичное положительное целое, большее нуля.

RDF- коллекции

SPARQL

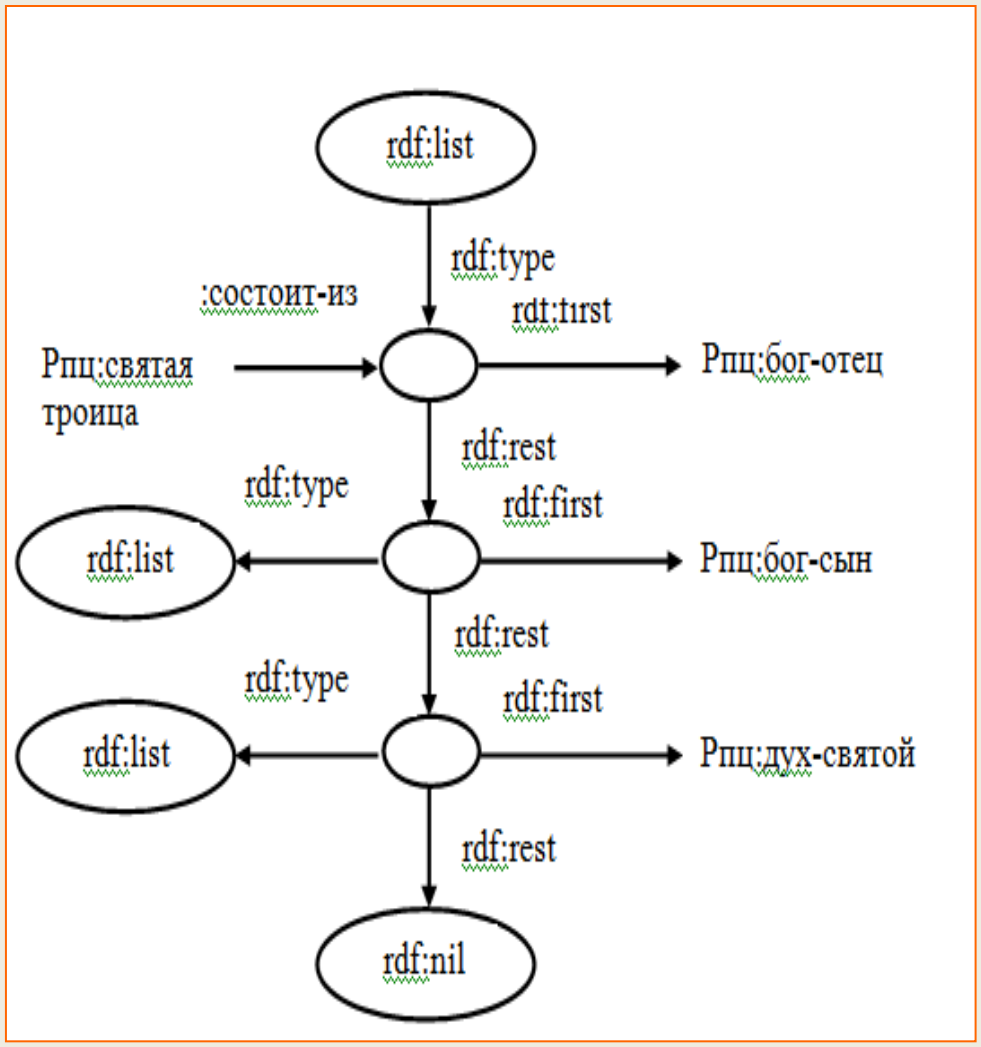
Коллекции являются **базовыми структурами** данных в RDF и представляют собой фактически линейную списочную **структуру, связывающую** (через специальные свойства пустых узлов-субъектов) **в единую последовательность различные триплеты.**

Коллекции являются **закрытыми группами**, т.е. они содержат только те элементы, которые были перечислены и больше никаких других по определению. (**Согласно канонической христианской традиции, Троица состоит из Бога-отца, Бога-Сына и Святого Духа и ни из кого больше по определению**).

В отличие от контейнеров, в которых один ресурс-субъект (обозначающий контейнер в целом) связан различными свойствами с различными объектами, **коллекции связывают узлы одними и теми же стандартными свойствами, определяющими порядок следования элементов.**

RDF- коллекции

SPARQL



Из примера видно, что в RDF-графе **отсутствуют явные указания на то, что в модели использована коллекция.** Здесь все элементы коллекции являются объектами одного и того же предопределенного свойства (предиката) – **«rdf:first»** и имеют в качестве субъекта своего триплета пустой узел, который связывается со следующим аналогичным пустым узлом и другим предопределенным свойством **«rdf:rest»**. Агрегирование всех элементов в единую структуру данных происходит за счет того, что все пустые узлы, формирующие основной список, неявно (без отображения в графе) получают зарезервированное в RDF свойство **«rdf:type»**, значение которого становится равным предопределенному символу **«rdf:list»**. **Признак окончания списка** задается в коллекциях с использованием предопределенного символа – **«rdf:nil»**.

SPARQL

Сериализация - это технический термин обозначающий **технологию сохранения RDF** в виде строки байтов, которая в дальнейшем может быть сохранена на магнитном диске или другом запоминающем устройстве . Использование термина "сериализация", а не "файла", объясняется тем, что существуют операционные системы, которые не используют термин "файл"

на практике все RDF сериализации представляют собой **текстовые файлы,**

Сериализация RDF

SPARQL

В текстовых файлах тройки могут быть представлены в следующих форматах (синтаксис)

- **RDF/XML—стандартный формат на базеXML.**
- **N-Triples—простой формат.**
- **Turtle—удобный формат.**
- **JSON-LD—формат на базе JSON.**
- **RDF и Microdata—формат RDF-разметки HTML-страниц**

SPARQL

Turtle (**Terse RDF Triple Language**) - формат для сериализации графов RDF (модель описания ресурсов).

Синтаксис.

URI в формате Turtle записываются в угловых скобках.

<http://dbpedia.org/resource/Leipzig>

SPARQL

Turtle (**Terse RDF Triple Language**) - формат для сериализации графов RDF (модель описания ресурсов).

Синтаксис.

Литералы заключаются в кавычки

"Лейпциг" @ ru
"51.333332" ^ ^ xsd:float

Интеграция данных

Сериализация RDF

SPARQL

Turtle (**Terse RDF Triple Language**) - формат для сериализации графов RDF (модель описания ресурсов).

Синтаксис.

В записях RDF моделей тройки (субъект – предикат – объект) разделяются точками.

```
<http://dbpedia.org/resource/Leipzig>
<http://www.w3.org/2000/01/rdf-schema#label>
"Leipzig"@de .
<http://dbpedia.org/resource/Leipzig>
<http://dbpedia.org/property/hasMayor >
<http://dbpedia.org/resource/Burkhard_Jung>.
```

Сериализация RDF

SPARQL

Turtle (**Terse RDF Triple Language**) - формат для сериализации графов RDF (модель описания ресурсов).

Синтаксис.

Пробелы и переносы строк за пределами идентификаторов ресурсов **игнорируются.**

Для сокращения записей URI обычно используются **префиксные объявления**

SPARQL

Turtle (**Terse RDF Triple Language**) - формат для сериализации графов RDF (модель описания ресурсов).

Синтаксис. Использование **префиксных объявлений**

@prefix dbr: <http://dbpedia.org/resource/>.

SPARQL

Использование префиксных объявлений

Синтаксис.

```
<http://dbpedia.org/resource/Leipzig>
<http://www.w3.org/2000/01/rdf-schema#label>  "Leipzig"@de .
```

```
@prefix dbr:  <http://dbpedia.org/resource/> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema> .
dbr:Leipzig  rdfs:label  "Leipzig"@de .
```

SPARQL

SPARQL (SPARQL Protocol and RDF Query Language), разработанный консорциумом W3C, определяет протокол для передачи запросов клиентов обработчикам запросов и язык формирования запросов на выборку и модификацию данных в RDF хранилищах данных.

Запрос SPARQL является текстовым документом в кодировке Unicode, либо в другой кодировке, поддерживаемым программным средством обработки запросов. Файлы с запросами SPARQL имеют расширение **.rq**.

Язык запросов SPARQL

SPARQL

Для языка SPARQL определены следующие элементы:

1. IRI;
2. Уточненные имена;
3. Пробельные символы;
4. Литералы;
5. Переменные запроса;
6. Пустые узлы;
7. Шаблоны триплетов;
8. Базовые шаблоны графов;
9. Ключевые слова;
10. Комментарии.

Язык запросов SPARQL

SPARQL

В языке SPARQL используется **интернационализированный идентификатор ресурса – IRI** (Internationalized Resource Identifier),

В SPARQL определено следующее **ограничение** для IRI: в идентификаторе не должны использоваться символы
"<", ">", "", пробел, "{", "}", "|", "\", "^" и **"`"**

В языке SPARQL IRI заключаются в угловые скобки "<" и ">", например:
<http://purl.org/dc/elements/1.1/>.

SPARQL

Уточненные имена задаются в виде: префикс-пространства-имен:локальное-имя, например, **dc:creator**.
Если для локального имени задано пространство имен по умолчанию, то префикс пространства имен не указывается, например, **:name**.

SPARQL

В **SPARQL** используются следующие **пробельные символы**: символ пробела (код "20"), символ горизонтальной табуляции (код "09"), символ возврата каретки (код "0D") и символ перехода на новую строку (код "0A").

SPARQL

Литералы в SPARQL заключаются в **двойные или одиночные апострофы**, за которыми может следовать метка языка, образуемая символом **"@"**, за которым задается код языка.

"abcd", **'Строка 1'@ru**, **"String 1"@en**.
Если литералы содержат апострофы или символ перевода на новую строку, то они заключаются в три двойных или одиночных апострофа, например:
'''Символ '<' задает начало IRI а символ '>' – его окончание.''',

SPARQL

В SPARQL можно также задавать **типизированные литералы**, т.е. литералы, содержащие данные определенного типа. Для этого **после литерала вставляются символы "^^"**, после которых либо задается **ссылка на IRI** для типа данных, либо задается **уточненное имя типа данных**

"ТИ-81"^^<http://www.someInstitute.edu/ns/groupDatatype>
или
"2008-09-01"^^xsd:date.

SPARQL

В SPARQL в качестве типов литералов можно использовать следующие типы схемы XML: **xsd:integer, xsd:decimal, xsd:float, xsd:double, xsd:string, xsd:boolean, xsd:dateTime и xsd:date.**

Для **целых чисел** (тип **xsd:integer**), **чисел с десятичной точкой** (тип **xsd:decimal**), **чисел в экспоненциальной форме** (тип **xsd:double**), а также **булевских значений true or false** (тип **xsd:boolean**) можно просто задать значение литерала без указания типа и без ограничивающих апострофов, например, **15, 2.72, 4.7e-3** или **true.**

В SPARQL определен также **нулевой литерал** (ресурс **rdf:nil**), который в сокращенной форме записывается как "( )".

Язык запросов SPARQL

SPARQL

Переменные запроса задаются в SPARQL как идентификаторы XML, перед которыми задается префикс **"$"** или префикс **"?"**, например **$x1** или **?x1** (это одна и та же переменная).

Переменные в SPARQL имеют **глобальную область видимости**, т.е. доступны в любом месте запроса.

SPARQL

Пустые узлы задаются в следующем виде **_:метка-узла**, например,
_:emptyNode1 dc:creator "Иванов И.И."^^xsd:token .

Если пустой узел в запросе один, его метку можно не указывать, заключив предикат и объект в квадратные скобки, например,
[dc:creator "Иванов И.И."^^xsd:token]

либо указав скобки "**[ ]**" перед предикатом и объектом, например,
[] dc:creator "Иванов И.И."^^xsd:token .

SPARQL

Пустой узел без метки может задаваться как субъект следующего шаблона триплета,

```
[ dc:creator "Иванов И.И."^^xsd:token ]
dc:contributor "Петров П.П."^^xsd:token .
```

что эквивалентно следующим двум триплетам:

```
_:emptyNode1 dc:creator "Иванов И.И."^^xsd:token.
_:emptyNode1 dc:contributor "Петров П.П."^^xsd:token
.
```

SPARQL

Пустой узел без метки может также задаваться как **объект предыдущего шаблона** триплета

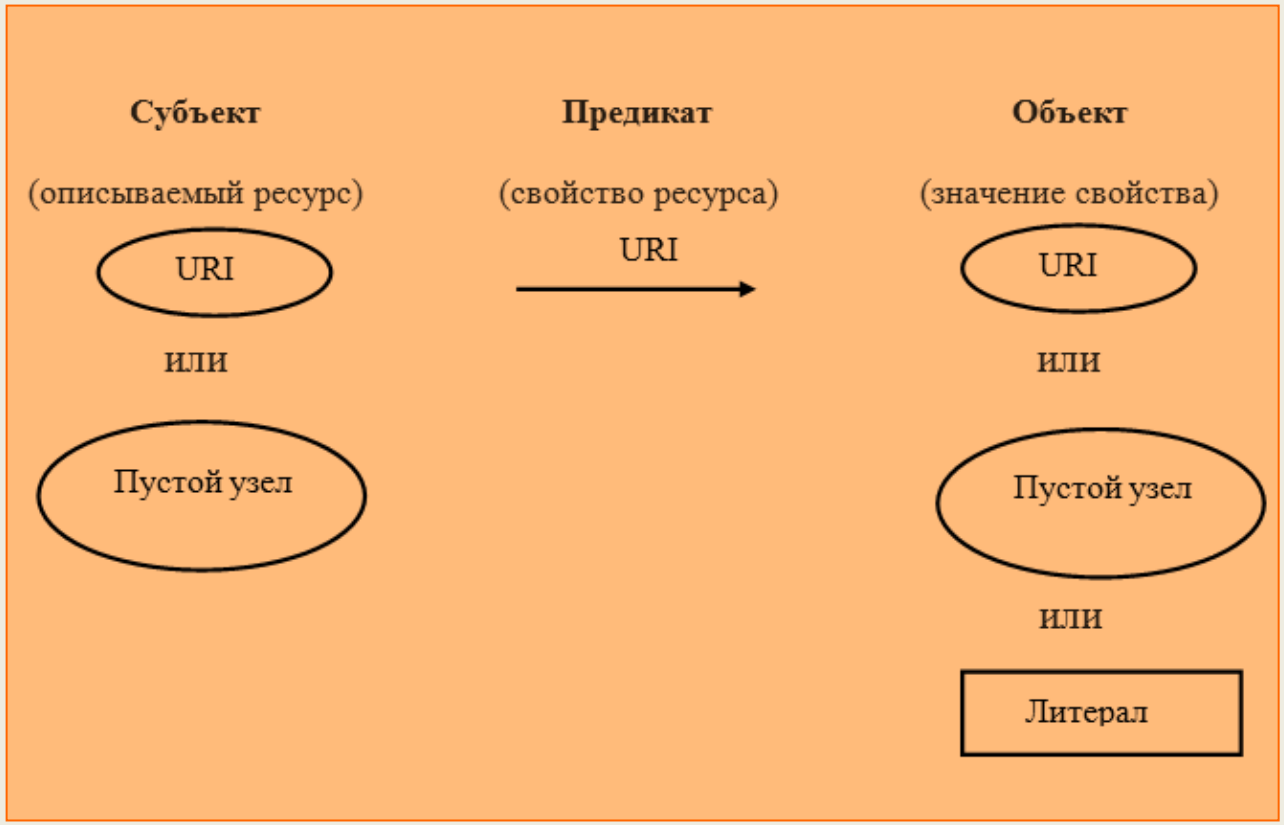
```
prog:statProg dc:subject
[ dc:creator "Иванов И.И."^^xsd:token ] .
```

что эквивалентно следующим двум триплетам:

```
prog:statProg dc:subject _:emptyNode1
_:emptyNode1 dc:creator "Иванов И.И."^^xsd:token .
```

SPARQL

Шаблоны триплетов состоят из трех компонент: **субъекта, предиката и объекта.**



SPARQL

При записи шаблонов триплетов можно использовать сокращения:

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

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

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

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

комбинированный

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

⟺

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

SPARQL

В SPARQL определены следующие **ключевые слова**

Ключевые слова, за исключением **a**, могут задаваться в любом регистре, т.е. как прописными, так и строчными буквами.

Комментарии в SPARQL начинаются с символа "#" и заканчиваются концом строки.

BASE	**SELECT**	**ORDER BY**	**FROM**	**GRAPH**	**STR**	**isURI**
PREFIX	**CONSTRUCT**	**LIMIT**	**FROM NAMED**	**OPTIONAL**	**LANG**	**isIRI**
	DESCRIBE	**OFFSET**	**WHERE**	**UNION**	**LANGMATCHES**	**isLiteral**
	ASK	**DISTINCT**		**FILTER**	**DATATYPE**	**REGEX**
		REDUCED		**a**	**BOUND**	**true**
					sameTERM	**false**

SPARQL

Общая структура SPARQL запроса

пролог,
запрос-SPARQL.

Пролог, содержит одно или несколько объявлений префиксов для пространств имен, а также может содержать одно предложение объявления базового IRI.

- запрос **SELECT;**
- запрос **CONSTRUCT**;
- запрос **ASK**;
- запрос **DESCRIB**E

SPARQL

Объявление префиксов сопоставляет префикс заданному пространству имен и имеет следующий синтаксис

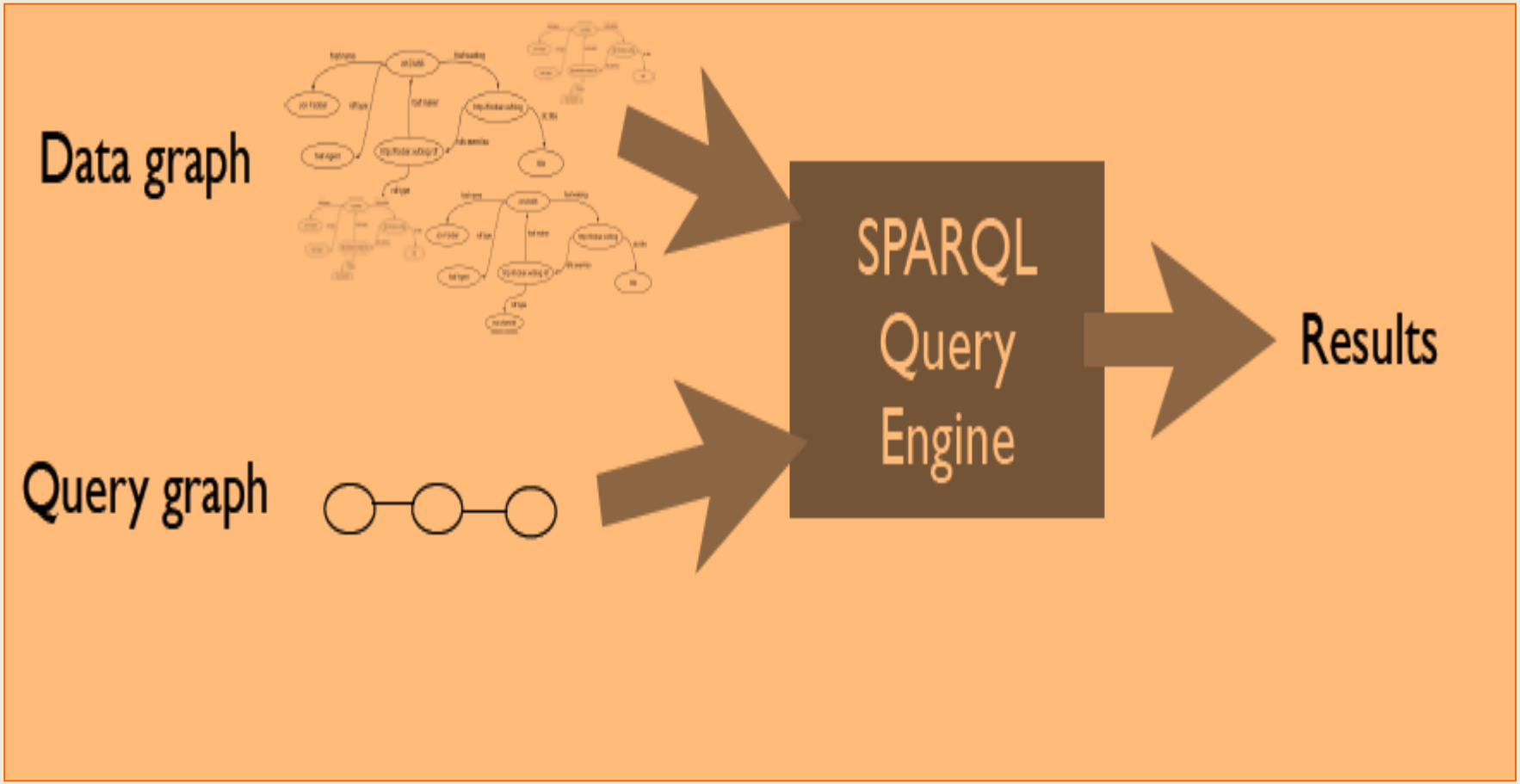
PREFIX префикс: <IRI>, например,
PREFIX dc: <http://purl.org/dc/elements/1.1/>.

Обозначение префикса

SPARQL

Исходными данными, которые обрабатывает запрос (файл с расширением **.rq**), являются файлы документов RDF, RDFS или OWL, записанные с использованием либо нотации N3 (файлы с расширением .n3), либо нотации Turtle (файлы с расширением .ttl), либо на языке RDF/XML (файлы с расширениями **.rdf** или **.owl**).

SPARQL



Запрос SELECT

SPARQL

PREFIX foo: <http://example.com/resources/>

префиксные объявления (пролог)

FROM …

источники запроса

SELECT …

состав ответа на запрос (список переменных)

WHERE { …}

шаблон запроса

ORDER BY …

модификаторы запроса

Шаблон запроса

SPARQL

Условие поиска в запросе задается шаблоном группы графов

WHERE *шаблон-группы-графов*

шаблон-группы-графов заключен в фигурные скобки ("{" и "}") и может содержать следующие компоненты:

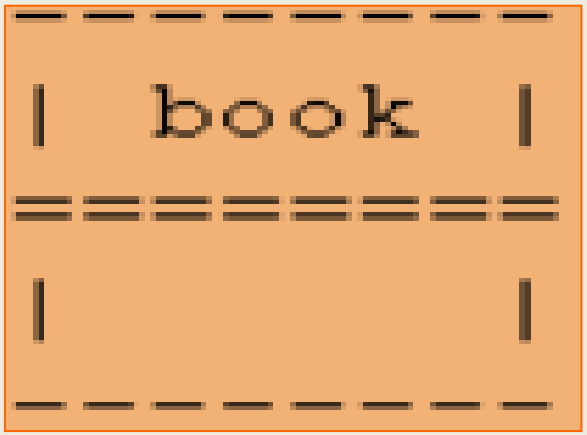
- пустой шаблон;
- базовый шаблон графа;
- вложенные шаблоны группы графов;
- дополнительный шаблон группы графов;
- объединение шаблонов группы графов;
- шаблон **GRAPH**;
- фильтр.

SPARQL

Пустой шаблон запроса

Пустой шаблон соответствует любому графу, включая пустой граф, с только одним решением, не связывающим переменную запроса.

```
SELECT $book
{}
```



Ключевое слово **WHERE** в условии может быть пропущено.

SPARQL

Исходный документ в формате Turtle

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix : <http://www.libRegistry.org/ns/lib#> .
# Экземпляр book1 класса Book
:book1 a <http://www.libRegistry.org/ns/lib#Book>;
# Идентификатор ISBN книги
dc:identifier "02419-0";
# Название книги
dc:title "Десять негритят"@ru;
# Автор книги
dc:creator "Кристи Агата"@ru;
# Жанр книги
dc:type "детектив"^^xsd:string;
```

SPARQL

Исходный документ в формате Turtle

```
# Издатель книги

dc:publisher "Эксмо"@ru;

# Дата издания книги

dc:date "2003-03-09"^^xsd:date .

# Экземпляр book2 класса Book

:book2 a <http://www.libRegistry.org/ns/lib#Book>;

dc:identifier "966-00-0785-5";

dc:title "Кобзар"@uk;

dc:creator "Шевченко Тарас"@uk;

dc:type "классика"^^xsd:string;

dc:publisher "Наукова думка"@uk;

dc:date "2002-10-16"^^xsd:date .
```

Исходный документ в формате Turtle

SPARQL

```
# Экземпляр book3 класса Book

:book3 a <http://www.libRegistry.org/ns/lib#Book>;

dc:identifier "0-14-062015-3";

dc:title "Robinson Crusoe"@en;

dc:creator "Defoe Daniel"@en;

dc:type "классика"^^xsd:string;

dc:publisher "Penguin"@en;

dc:date "2005-06-19"^^xsd:date .

# Экземпляр book4 класса Book

:book4 a <http://www.libRegistry.org/ns/lib#Book>;

dc:identifier "5-367-00019-3";

dc:title "Убийство в доме викария"@ru;
```

Исходный документ в формате Turtle

SPARQL

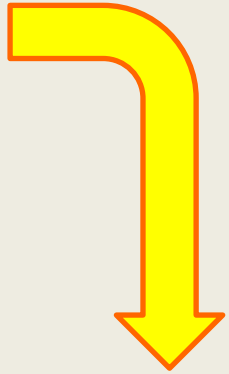
```
dc:creator "Кристи Агата"@ru;

dc:type "детектив"^^xsd:string;

dc:publisher "Амфора"@ru;

dc:date "2006-01-28"^^xsd:date .

# Экземпляр book5 класса Book

:book5 a <http://www.libRegistry.org/ns/lib#Book>;

dc:identifier "966-7047-42-3";

dc:title "Приключения Робинзона Крузо"@ru;

dc:creator "Дефо Даниэл"@ru;

dc:type "классика"^^xsd:string;

dc:publisher "Эксмо"@ru;

dc:date "2007-03-28"^^xsd:date .
```

Примеры построения SPARQL запросов к данным представленным в исходном документе в нотации Turtle

SPARQL

Запрос-1. Найти и вывести в таблицу ответов для экземпляра книги book5 название книги (переменная- **bookTitle**) и автора (переменная- **bookAuthor**)

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://www.libRegistry.org/ns/lib#>
SELECT ?bookTitle ?bookAuthor
WHERE {
:book5 dc:creator ?bookAuthor;
dc:title ?bookTitle;
dc:type ?bookGenre .
}
```



bookTitle	bookAuthor
"Приключения Робинзона Крузо"@ru	"Дефо Даниэл"@ru

Примеры построения SPARQL запросов к данным представленным в исходном документе в нотации Turtle

SPARQL

Запрос-2. Найти и вывести в таблицу ответов для экземпляра книги book1 название книги (переменная- **bookTitle**), автора (переменная- **bookAuthor**) и жанр произведения (переменная- **bookGenre**).)

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://www.libRegistry.org/ns/lib#>
SELECT *
WHERE {
:book1 dc:creator ?bookAuthor;
dc:title ?bookTitle;
dc:type ?bookGenre .
}
```

bookAuthor	bookTitle	bookGenre
"Кристи Агата"@ru	"Десять негритят"@ru	"детектив"^^xsd:string

SPARQL

Примеры построения SPARQL запросов к данным представленным в исходном документе в нотации Turtle

Запрос – 3. Найти и вывести в таблицу ответов все экземпляры книг, описанных в исходном документе (переменная- **book)** и указать наименование этих книг (переменная- **bookTitle**).

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>

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

PREFIX : <http://www.libRegistry.org/ns/lib#>

SELECT ?book ?bookTitle

WHERE {

?book dc:title ?bookTitle .

}
```

book	bookTitle
:book5	"Приключения Робинзона Крузо"@ru
:book4	"Убийство в доме викария"@ru
:book3	"Robinson Crusoe"@en
:book2	"Кобзар"@uk
:book1	"Десять негритят"@ru

Примеры построения SPARQL запросов к данным представленным в исходном документе в нотации Turtle

SPARQL

Запрос – 4. Найти и вывести в таблицу ответов обозначения (переменная - **bookChar**) и значения (переменная- **bookValue**) параметров книги 1 , описанной в исходном документе.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>

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

PREFIX : <http://www.libRegistry.org/ns/lib#>

SELECT ?bookChar ?bookValue

WHERE {

:book1 ?bookChar ?bookValue .

}
```

bookChar	bookValue
dc:date	"2003-03-09"^^xsd:date
dc:publisher	"Эксмо"@ru
dc:type	"детектив"^^xsd:string
dc:creator	"Кристи Агата"@ru
dc:title	"Десять негритят"@ru
dc:identifier	"02419-0"
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	:Book

SPARQL

Вид операции	Функции и операторы	Пример
Logical & Comparisons	!, &&, ||, =, !=, <, <=, >, >=, IN, NOT IN	?hasPermit || ?age < 25
Conditionals *(SPARQL 1.1)*	EXISTS, NOT EXISTS, IF, COALESCE	NOT EXISTS { ?p foaf:mbox ?email }
Math	+, -, *, /, abs, round, ceil, floor, RAND	?decimal * 10 > ?minPercent
Strings *(SPARQL 1.1)*	STRLEN, SUBSTR, UCASE, LCASE, STRSTARTS, CONCAT, STRENDS, CONTAINS, STRBEFORE, STRAFTER	STRLEN(?description) < 255

SPARQL

Вид операции	Функции и операторы	Пример
Date/time *(SPARQL 1.1)*	now, year, month, day, hours, minutes, seconds, timezone, tz	month(now()) < 4
SPARQL tests	isURI, isBlank, isLiteral, isNumeric, bound	isURI(?person) || !bound(?person)
Constructors *(SPARQL 1.1)*	URI, BNODE, STRDT, STRLANG, UUID, STRUUID	STRLANG(?text, "en") = "hello"@en
Accessors	str, lang, datatype	lang(?title) = "en"

SPARQL

Вид операции	Функции и операторы	Пример
Hashing *(1.1)*	MD5, SHA1, SHA256, SHA512	BIND(SHA256(?email) AS ?hash)
Miscellaneous	isURI, isBlank, isLiteral, isNumeric, bound	regex(?ssn, "\\d{3}-\\d{2}-\\d{4}")

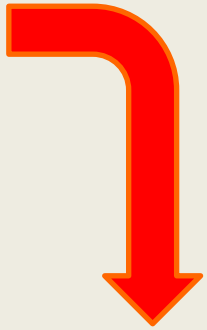
SPARQL

В выражениях можно использовать операции отношения: равно ("**=**"), **не равно ("!=")**, **больше (">")**, **меньше ("<")**, **больше или равно (">=")**, **меньше или равно ("<=")**, **а также логические операции: И ("&&")**, **ИЛИ ("||")** и **НЕ ("!")**.

Для числовых литералов определены также одноместные арифметические операции: плюс ("**+**") **и минус ("-")**, **а также двуместные арифметические операции: сложение ("+")**, **вычитание ("-")**, **умножение ("*")** и **деление ("/")**.

SPARQL

В языке SPARQL определены встроенные функции следующих типов: (используются при конструировании фильтров)

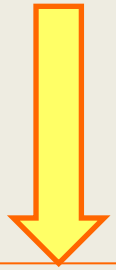


1. функции проверки;
2. функция преобразования;
3. функция определения типа данных;
4. языковые функции;
5. функции выполнения логических операций;
6. функция поиска с использованием регулярного выражения.

SPARQL

К **функциям проверки относятся :**
bound(),
isIRI(),
isURI(),
isBlank(),
isLiteral(),
RDFterm-equal(),
sameTerm().

Функция
bound(*переменная*)



проверяет, является ли переменная, заданная в качестве аргумента функции, связанной. Если это условие выполняется, функция возвращает true, иначе возвращает false.

SPARQL

Данные:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
 @prefix dc:<http://purl.org/dc/elements/1.1/> .
 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

 _:a foaf:givenName "Alice".

_:b foaf:givenName "Bob" .
_:b dc:date "2005-04-04T04:04:04Z"^^xsd:dateTime .
```

```
SELECT ?givenName
WHERE
{
?x foaf:givenName ?givenName .
OPTIONAL
{ ?x dc:date ?date } .
 FILTER ( bound(?date) )
}
```

givenName
"Bob"

SPARQL

Данные:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
 @prefix dc:<http://purl.org/dc/elements/1.1/> .
 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

 _:a foaf:givenName "Alice".

_:b foaf:givenName "Bob" .
_:b dc:date "2005-04-04T04:04:04Z"^^xsd:dateTime .
```

```
SELECT ?givenName
WHERE
{
?x foaf:givenName ?givenName .
OPTIONAL
{ ?x dc:date ?date } .
 FILTER ( !bound(?date) )
}
```

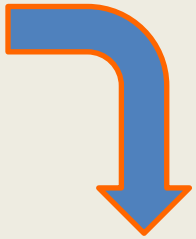
givenName
"Alice"

Этот запрос выбирает людей с name, но без date:

SPARQL

Функции

isIRI(*аргумент*)
isURI(*аргумент*)

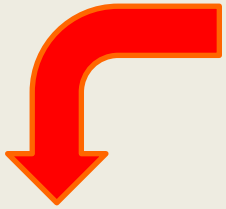


Проверяют, является ли аргумент IRI(URI). Если это условие выполняется, функция возвращает true, иначе возвращает false.
В качестве аргумента могут быть заданы :
IRI(URI),
литерал,
переменная,
пустой узел.

SPARQL

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

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



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

SPARQL

Функция
isBlank(*аргумент*)

проверяет, является ли аргумент пустым узлом.

Если это условие выполняется, функция возвращает true, иначе возвращает false.

В качестве аргумента могут быть заданы IRI(URI), литерал, переменная или пустой узел.

SPARQL

```
@prefix a: <http://www.w3.org/2000/10/annotation-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a a:annotates <http://www.w3.org/TR/rdf-sparql-query/> .
_:a dc:creator "Alice B. Toeclips" .
_:b a:annotates <http://www.w3.org/TR/rdf-sparql-query/> .
_:b dc:creator _:c .
_:c foaf:given "Bob".
_:c foaf:family "Smith".
```

```
SELECT ?given ?family
WHERE
{
?annot a:annotates <http://www.w3.org/TR/rdf-sparql-query/> .
?annot dc:creator ?c .
OPTIONAL { ?c foaf:given ?given ; foaf:family ?family } .
FILTER isBlank(?c)
```

given	family
"Bob"	"Smith"

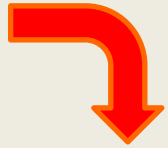
SPARQL

Функция isLiteral(*аргумент)*

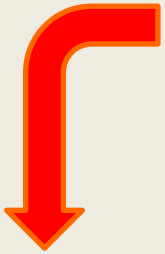
проверяет, является ли аргумент литералом.
Если это условие выполняется, функция возвращает true, иначе возвращает false.
В качестве аргумента могут быть заданы
IRI(URI),
литерал,
переменная,
пустой узел.

SPARQL

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

 _:a foaf:name "Alice".
 _:a foaf:mbox <mailto:alice@work.example> .
_:b foaf:name "Bob" .
 _:b foaf:mbox "bob@work.example" .
```



```
SELECT ?name ?mbox
WHERE
{
?x foaf:name ?name ;
 foaf:mbox ?mbox .
 FILTER isLiteral(?mbox)
}
```



name	mbox
"Bob"	"bob@work.example"

SPARQL

Функция str(аргумент)

возвращает строковое представление аргумента.
В качестве аргумента могут быть заданы
литерал,
переменная,
IRI(URI).

SPARQL

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

Этот запрос выбирает набор людей, которые используют свой адрес work.example в своем профиле foaf:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
 WHERE
 {
 ?x foaf:name ?name ;
  foaf:mbox ?mbox .
  FILTER regex(str(?mbox),
"@work.example")
}
```

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

SPARQL

Функция lang(аргумент)

Возвращает тег языка (language tag) для ltrl при наличии такового.
Если тег языка для ltrl отсутствует, возвращается ""

SPARQL

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Robert"@EN.
 _:a foaf:name "Roberto"@ES.
 _:a foaf:mbox <mailto:bob@work.example> .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
 WHERE
 {
 ?x foaf:name ?name ;
 foaf:mbox ?mbox .
 FILTER ( lang(?name) = "ES" )
}
```

name	mbox
"Roberto"@ES	<mailto:bob@work.example>

SPARQL

Функция **datatype(аргумент)**

Возвращает datatype IRI для typedLit; возвращает xsd:string, если параметр является простым (simple) литералом.

SPARQL

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix eg: <http://biometrics.example/ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>
. _:a foaf:name "Alice".
_:a eg:shoeSize "9.5"^^xsd:float .
 _:b foaf:name "Bob".
 _:b eg:shoeSize "42"^^xsd:integer .
```

Этот запрос находит foaf:name и foaf:shoeSize всех, у кого размер обуви (shoeSize) является целым числом:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX eg: <http://biometrics.example/ns#>
SELECT ?name ?shoeSize
 WHERE
{
?x foaf:name ?name ;
eg:shoeSize ?shoeSize .
FILTER ( datatype(?shoeSize) = xsd:integer )
 }
```

name	shoeSize
"Bob"	42

SPARQL

Функция RDFterm-equal(*аргумент-1, аргумент-2)*

Возвращает *возвращает true, если оба аргумента являются эквивалентными, иначе возвращает false.*
В качестве аргументов могут быть заданы
литерал,
переменная,
пустой узел,
IRI(URI).
Два литерала являются эквивалентными при выполнении следующих условий: они посимвольно равны; у обоих либо есть, либо нет меток языков; если метки языков есть, то они одинаковые; у обоих либо есть, либо нет типов данных.
Два пустых узла считаются эквивалентными, если они указывают на один и тот же узел.
Два IRI(URI) считаются эквивалентными, если они посимвольно равны.

SPARQL

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

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name1 ?name2
WHERE
{
?x foaf:name ?name1 ;
foaf:mbox ?mbox1 .
 ?y foaf:name ?name2 ;
foaf:mbox ?mbox2 .
FILTER (?mbox1 = ?mbox2 && ?name1 != ?name2)
}
```

name1	name2
"Alice"	"Ms A."
"Ms A."	"Alice"

SPARQL

Функция **datatype(аргумент)**

Возвращает datatype IRI для typedLit; возвращает xsd:string, если параметр является простым (simple) литералом.

SPARQL

SPARQL

проверяет, является ли аргумент литералом. Если это условие выполняется, функция возвращает true, иначе возвращает false. В качестве аргумента могут быть заданы IRI(URI), литерал, переменная или пустой узел.

Функция RDFterm-equal(*аргумент-1, аргумент-2) возвращает true, если оба аргумента являются эквивалентными, иначе возвращает false. В качестве аргументов могут быть заданы литерал, переменная, пустой узел или IRI(URI). Два литерала являются эквивалентными при выполнении следующих условий: они посимвольно равны; у обоих либо есть, либо нет меток языков; если метки языков есть, то они одинаковые; у обоих либо есть, либо нет типов данных. Два пустых узла считаются эквивалентными, если они указывают на один и тот же узел. Два IRI(URI) считаются эквивалентными, если они посимвольно равны.*

SPARQL

Функция sameTerm(*аргумент-1, аргумент-2) возвращает true, если оба аргумента являются одним и тем же данным, иначе возвращает false. В качестве аргументов могут быть заданы литерал, переменная, пустой узел или IRI(URI).*
Функция преобразования str(*аргумент) возвращает строковое представление аргумента. В качестве аргумента могут быть заданы литерал, переменная или IRI(URI).*
Функция определения типа datatype(*аргумент) возвращает уточненный тип данных схемы XML для аргумента. Если аргумент задает простое (нетипизированное) данное, возвращается тип данных xsd:string. В качестве аргумента могут быть заданы простой или типизированный литерал, а также переменная.*

SPARQL

К языковым функциям относятся функции lang() и langMatches().
Функция lang(аргумент) возвращает строку кода языка аргумента. Если код языка для аргумента не задан, возвращается пустая строка. В качестве аргумента могут быть заданы литерал или переменная.

SPARQL

Функция langMatches(*метка-языка, диапазон-меток) возвращает true, если метка-языка входит в диапазон-меток, заданный во втором аргументе, иначе возвращает false. Аргумент диапазон-меток содержит либо список меток языков, отделенных друг от друга символом "-", либо символ "*".*
Логические функции logical-or() и logical-and()выполняют логические операции над аргументами.
Функция logical-or(*аргумент-1, аргумент-2) возвращает результат выполнения операции аргумент-1 || аргумент-2 (true или false). Оба аргумента должны иметь булевский тип.*
Функция logical-and(*аргумент-1, аргумент-2) возвращает результат выполнения операции аргумент-1 && аргумент-2 (true или false). Оба аргумента должны иметь булевский тип.*

SPARQL

Функция поиска с использованием регулярного выражения regex(*строка, шаблон, флажки) вызывает функцию fn:matches()языка XPath 2.0 (см. 5.1.1.4.3.5). В аргументе строка задается строковый литерал или строковая переменная, а в аргументе шаблон – шаблон поиска в строке (в соответствии с правилами задания регулярных выражений). В необязательном аргументе задаются флажки "s", "m", "i" и "x" (один флажок или строка, содержащая комбинацию флажков). Функция возвращает true, если в строке найдено соответствие шаблону и false – в противном случае.*

SPARQL

```
PREFIX dc: http://purl.org/dc/elements/1.1/
PREFIX : <http://www.libRegistry.org/ns/lib#>
SELECT ?bookAuthor ?bookTitle
WHERE {
?book dc:title ?bookTitle;
dc:creator ?bookAuthor .
FILTER (regex(str(?bookAuthor), "Defoe") ||
regex(str(?bookAuthor), "Дефо"))
}
```

bookAuthor	bookTitle
"Дефо Даниэл"@ru	"Приключения Робинзона Крузо"@ru
"Defoe Daniel"@en	"Robinson Crusoe"@en

SPARQL

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <http://www.libRegistry.org/ns/lib#>
SELECT ?bookAuthor ?bookTitle
WHERE {
?book dc:title ?bookTitle;
dc:creator ?bookAuthor .
FILTER (lang(?bookTitle) = "uk")
```

bookAuthor	bookTitle
"Шевченко Тарас"@uk	"Кобзар"@uk

SPARQL

```
SELECT ?bookAuthor ?bookTitle ?bookPubDate
WHERE {
?book dc:title ?bookTitle;
dc:creator ?bookAuthor;
dc:publisher ?bookPublisher;
dc:date ?bookPubDate .
FILTER (?bookPublisher = "Эксмо"@ru &&
?bookPubDate > "2005-01-01"^^xsd:date)
}
```

bookAuthor	bookTitle	bookPubDate
"Дефо Даниэл"@ru	"Приключения Робинзона Крузо"@ru	"2007-03-28"^^xsd:date

SPARQL

Проверка даты: Анализируемые даты должны находится между 1 января. 1-й (включительно), 2015 и 1 января. 2016 годов

```
FILTER("2015-01-01"^^xsd:dateTime <= ?dob
&& ?dob < "2016-01-01"^^xsd:dateTime).
```

SPARQL

FILTER on values in Labels

```
SELECT ?human ?humanLabel
WHERE
{
?human wdt:P31 wd:Q15632617;  # fictional human
rdfs:label ?humanLabel.
FILTER(LANG(?humanLabel) = "en").
FILTER(STRSTARTS(?humanLabel, "Mr. ")).
}
```

SPARQL

FILTER NOT EXIST

```
SELECT ?human ?humanLabel ?image
WHERE
{
?human wdt:P31 wd:Q15632617;  # fictional human
rdfs:label ?humanLabel.
FILTER(LANG(?humanLabel) = "en").
FILTER(STRSTARTS(?humanLabel, "Mr. ")).
FILTER NOT EXISTS{ ?human wdt:P18 ?image.}
}  # without images
```

SPARQL

MINUS

```
SELECT ?human ?humanLabel ?image
WHERE
{  ?human wdt:P31 wd:Q15632617;  # fictional human
rdfs:label ?humanLabel.
FILTER(LANG(?humanLabel) = "en").
FILTER(STRSTARTS(?humanLabel, "Mr. ")).
MINUS{ ?human wdt:P18 ?image. }  # without images
}
```

SPARQL

СРАВНЕНИЕ ТЕКСТОВЫХ СТРОК

```
SELECT ?workflow
WHERE
{
?workflow rdf:type mecontrib:Workflow ;
mebase:has-content-type ?ct .
 ?ct dcterms:title ?ct_title
FILTER regex(?ct_title,'^taverna','i')
}
```

Сравнивается значение **?ct_title** с текстовой строкой **'taverna'**. Знак **caret (^)** используется, чтобы указать, что строка для **?ct_title** должен начинаться с' **taverna',** а не просто иметь его где-то внутри строки.

SPARQL

СРАВНЕНИЕ ТЕКСТОВЫХ СТРОК

Анализируется значение **результата преобразования ресурса в** текстовую строку **на вхождение в эту строку** литерала **«Group»**

```
SELECT ?membership ?requester
WHERE
{
?membership rdf:type mebase:Membership ;
mebase:has-requester ?requester ;
mebase:has-accepter ?accepter ;
mebase:accepted-at ?accepted_at
FILTER regex(str(?requester),'Group','i')
}
```

SPARQL

ОПЕРАЦИИ С ЧИСЛАМИ

```
SELECT DISTINCT ?workflow ?ct_title
WHERE
{
?workflow rdf:type mecontrib:Workflow ;
mebase:has-content-type ?ct .
?ct dcterms:title ?ct_title .
?rating rdf:type meannot:Rating ;
mebase:annotates ?workflow;
meannot:rating-score ?score
FILTER (?score >= 4)
}
```

Выдача всех рабочих процессов, которые имеют рейтинг, равный 4 или больше:

SPARQL

ФИЛЬТРАЦИЯ ПО НЕСКОЛЬКИМ КРИТЕРИЯМ

```
SELECT DISTINCT ?workflow ?ct_title
WHERE
{
?workflow rdf:type mecontrib:Workflow ;
mebase:has-content-type ?ct .
?ct dcterms:title ?ct_title .
?rating rdf:type meannot:Rating ;
mebase:annotates ?workflow ;
 meannot:rating-score ?score
 FILTER (?score >= 4 &&
regex(?ct_title,'^Taverna 1'))
}
```

Выдача всех рабочих процессов, которые имеют рейтинг, равный 4 или больше и имеют наименование Taverna 1

SPARQL

ФИЛЬТРАЦИЯ ПО ДАТЕ ОБНОВЛЕНИЯ ДАННЫХ

```
SELECT ?workflow ?added
WHERE
 {
?workflow rdf:type mecontrib:Workflow ;
dcterms:created ?added
FILTER ( ?added >= xsd:dateTime('2009-09-01T00:00:00Z') )
}
```

Выдача всех рабочих процессов,, которые были добавлены с начала сентября 2009 года: