

# Semantic Web Technologies

## Basic SPARQL

Jos de Bruijn  
jos.debruijn@deri.org

Digital Enterprise Research Institute (DERI)  
University of Innsbruck, Austria

May 4, 2006

82/139

## Outline

Turtle Syntax for RDF

### SPARQL

Basic SPARQL Queries  
Query Answer

83/139

## Terse RDF Triple Language

- ▶ RDF/XML language hard to read
- ▶ Notation 3 (N3)
  - ▶ Syntax for RDF
  - ▶ Logical language for RDF
- ▶ Turtle
  - ▶ Refinement of N3
  - ▶ Just RDF representation
- ▶ Basis for parts of SPARQL
  - ▶ Graph patterns

85/139

## Basic Turtle

- ▶ Plain text syntax for RDF
- ▶ Based on **Unicode**
- ▶ Mechanisms for namespace abbreviation
- ▶ Allows grouping of triples according to **subject**
- ▶ Shortcuts for collections
- ▶ In short:
  - ▶ Takes good things of RDF/XML
  - ▶ and leaves out angle brackets

86/139

## Prefixes

- ▶ Mechanism for namespace abbreviation
- ▶ Syntax:
 

```
@prefix abbr: <URI> .
```
- ▶ Example:
 

```
@prefix rdf:
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
```
- ▶ Default:
 

```
@prefix : <URI> .
```
- ▶ Example:
 

```
@prefix : <http://example.org/myOntology#> .
```

87/139

## Identifiers in Turtle

- ▶ URIs: `<URI>`

```
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```
- ▶ QNames: `namespace-abbrev?:localname`

```
rdf:type dc:title :hasName
```
- ▶ Literals: `"string"(@lang)?(^type)?`

```
"John" "Hello"@en-GB "1.4"^^xsd:decimal
```
- ▶ Typed literal shortcuts
  - ▶ integer: 2 45
  - ▶ decimal: 2.4 5.67
  - ▶ boolean: true false

88/139

## Triples in Turtle

- ▶ Simple triple: *subject predicate object* .  
:john rdf:label "John"
- ▶ Grouping triples: *subject predicate object ; predicate object ...* .  
:john  
  rdf:label "John" ;  
  rdf:type ex:Person ;  
  ex:homePage <http://example.org/johnspage/> .

89/139

## Blank Nodes in Turtle

- ▶ Simple blank node: `[]`  
:john ex:hasFather [] .
- ▶ Blank node as subject: [ *predicate object ; predicate object ...* ] .  
[ ex:hasName "John" ] .  
[ ex:authorOf :lotr ;  
  ex:hasName "Tolkien" ] .
- ▶ Collections: ( *object<sub>1</sub> ... object<sub>n</sub>* )
  - ▶ :doc1 ex:hasAuthor (:john :mary) .
  - ▶ is short for:  
:doc1 ex:hasAuthor  
  [ rdf:first :john;  
    rdf:rest [ rdf:first :mary;  
              rdf:rest rdf:nil ] ] .

90/139

## Example

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix : <http://example.org/#> .

<http://www.w3.org/TR/rdf-syntax-grammar>
  dc:title "RDF/XML Syntax Specification (Revised)" ;
  editor [
    :fullName "Dave Beckett";
    :homePage <http://purl.org/net/dajobe/>
  ] .
```

91/139

## Turtle vs. RDF/XML

- ▶ Copies many features from RDF/XML
  - ▶ Namespace abbreviations (QNames)
  - ▶ Grouping of triples
  - ▶ Blank node treatment
- ▶ Differences
  - ▶ No ugly angle brackets
  - ▶ Compact (**T**erse) syntax
  - ▶ Abbreviations for typed literals
- ▶ Standard for SPARQL query patterns
- ▶ Not a standard for RDF

92/139

## Querying RDF

- ▶ SPARQL
  - ▶ RDF Query language
  - ▶ Based on RDQL
  - ▶ Uses SQL-like syntax
- ▶ Example:  
PREFIX dc: <http://purl.org/dc/elements/1.1/>  
  
SELECT ?title  
WHERE { <http://example.org/book/book1> dc:title  
  ?title }

94/139

## SPARQL Queries

- ▶ PREFIX
  - ▶ Prefix mechanism for abbreviating URIs
- ▶ SELECT
  - ▶ Identifies the variables to be returned in the query answer
- ▶ (FROM)
  - ▶ Name of the graph to be queried
- ▶ WHERE
  - ▶ Query pattern as a list of triple patterns

96/139

## URI abbreviation: PREFIX

- ▶ Mechanism for namespace abbreviation
- ▶ Syntax:  
PREFIX *abbr*: <URI>
- ▶ Example:  
PREFIX rdf:  
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
- ▶ Default:  
PREFIX : <URI>
- ▶ Example:  
PREFIX : <http://example.org/myOntology#>

97/139

## Selecting variables: SELECT

- ▶ Filtering variables to return
- ▶ Variables: *?string*  
?x ?title ?name
- ▶ Syntax:  
SELECT *var*<sub>1</sub>, ... , *var*<sub>*n*</sub>  
SELECT ?x, ?title  
SELECT ?name
- ▶ Variables in SELECT are **distinguished** variables

98/139

## Query patterns: WHERE

- ▶ Graph pattern to **match**
- ▶ Set of triples:  
{ (*subject predicate object* .)\* }
  - ▶ Subject: URI, QName, Blank node **Literal**, Variable
  - ▶ Predicate: URI, QName, Blank node, Variable
  - ▶ Object: URI, QName, Blank node **Literal**, Variable
- ▶ Example:  
{  
\_ :author ex:hasName ?name .  
\_ :author ex:authorOf :lotr .  
}
- ▶ Optional triples: OPTIONAL *triple* .  
OPTIONAL :john ont:hasAge ?age

99/139

## Example RDF Dataset (Turtle)

```
@prefix : <http://example.org/data#> .  
@prefix ont: <http://example.org/myOntology#> .  
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .  
  
:john  
  vcard:FN "John Smith" ;  
  vcard:N [  
    vcard:Given "John" ;  
    vcard:Family "Smith" ] ;  
  ont:hasAge 32 ;  
  ont:marriedTo :mary .  
:mary  
  vcard:FN "Mary Smith" ;  
  vcard:N [  
    vcard:Given "Mary" ;  
    vcard:Family "Smith" ] ;  
  ont:hasAge 29 .
```

100/139

## SPARQL Queries: all full names

"Return the full names of all people in the graph"

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>  
SELECT ?fullName  
WHERE {?x vCard:FN ?fullName}
```

result:

```
fullName  
=====  
"John Smith"  
"Mary Smith"
```

101/139

## SPARQL Queries: properties

"Return the relation between John and Mary"

```
PREFIX : <http://example.org/data#>  
SELECT ?p  
WHERE { :john ?p :mary }
```

result:

```
p  
=====  
<http://example.org/myOntology#marriedTo>
```

102/139

## SPARQL Queries: complex patterns

"Return the spouse of a person by the name of John Smith"

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX ont: <http://example.org/myOntology#>
SELECT ?y
WHERE {?x vCard:FN "John Smith".
       ?x ont:marriedTo ?y}
```

result:

```
y
=====
<http://example.org/data#mary>
```

103/139

## SPARQL Queries: blank nodes

"Return the name and the first name of all people in the KB"

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?name, ?firstName
WHERE {?x vCard:N ?name .
       ?name vCard:Given ?firstName}
```

result:

```
name firstName
=====
_:a "John"
_:b "Mary"
```

104/139

## SPARQL Queries: optional patterns (OPTIONAL)

"Return all people and (optionally) their spouses"

```
PREFIX ex: <http://example.org/#>
SELECT ?person, ?spouse
WHERE {?person ex:hasAge ?age .
       OPTIONAL { ?person ex:marriedTo ?spouse } }
```

result:

```
?person                ?spouse
=====
<http://example.org/#mary>
<http://example.org/#john> <http://example.org/#mary>
```

105/139

## Filters in Query Patterns

- ▶ Conditions on literal values
- ▶ Syntax: `FILTER expression`  
`FILTER (?age > 30) FILTER isIRI(?x) FILTER !BOUND(?y)`
- ▶ Different forms
  - ▶ Value comparison, e.g., `>`, `!=`, `>=`
  - ▶ Numeric functions, e.g., `+`, `*`
  - ▶ SPARQL test, e.g., `BOUND(?x)`, `isIRI(?x, isLITERAL(?y))`
  - ▶ **Negation**, e.g., `!BOUND(?x)`

106/139

## SPARQL Tests

- ▶ `BOUND(var)`
  - ▶ **true** if `var` is bound in query answer;
  - ▶ **false**, otherwise
  - ▶ Together with negation `!`, enables **negation-as-failure**
- ▶ Testing types
  - ▶ `isIRI(A)`
  - ▶ `isBLANK(A)`
  - ▶ `isLITERAL(A)`
- ▶ Comparing RDF terms
  - ▶ `A = B`
  - ▶ `A != B`
- ▶ Boolean AND/OR
  - ▶ `A && B`
  - ▶ `A || B`

107/139

## XQuery Functions

- ▶ Numeric, Date comparison
  - ▶ `A = B`
  - ▶ `A != B`
  - ▶ `A <= B`
  - ▶ `A >= B`
  - ▶ `A < B`
  - ▶ `A > B`
- ▶ Basic arithmetic
  - ▶ `A + B`
  - ▶ `A - B`
  - ▶ `A * B`
  - ▶ `A / B`

108/139

## SPARQL Queries: constraints

"Return all people over 30 in the KB"

```
PREFIX ont: <http://example.org/myOntology#>
SELECT ?x
WHERE {?x ont:hasAge ?age .
       FILTER(?age > 30)}
```

result:

```
x
=====
<http://example.org/data#john>
```

109/139

## RDF Datasets: FROM

- ▶ Dataset = RDF Graph
- ▶ Select graph to be queried
- ▶ In case of multiple FROM clauses, graphs are **merged**

110/139

## SPARQL Queries: FROM clause

Graph:

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

Query:

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

111/139

## Named Graphs: FROM NAMED

- ▶ Graphs may be **named**
- ▶ Named graphs not in RDF
- ▶ Associate **name** with a particular graph
- ▶ Specify named graph: FROM NAMED <URI>
- ▶ Allow to query based on name: GRAPH name { triples }

112/139

## SPARQL Queries: FROM NAMED I

Graph <http://example.org/bob>:

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

Graph <http://example.org/alice>:

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

113/139

## SPARQL Queries: FROM NAMED II

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
SELECT ?src ?name
```

```
FROM NAMED <http://example.org/alice>
FROM NAMED <http://example.org/bob>
```

```
WHERE
{ GRAPH ?src { ?x foaf:name ?name } }
```

result:

```
src name
=====
<http://example.org/bob> "Bob"
<http://example.org/alice> "Alice"
```

114/139

## SPARQL Query answers

- ▶ Variable substitution
  - ▶ Assigning values to variables  
e.g. [?x=<http://www.example.org/#john>,?name=:a,  
?firstName="John"]
  - ▶ Not all variables need to be bound  
e.g. [?person=<http://example.org/#mary>,?spouse=]
- ▶ Query answer
  - ▶ Substitute variables in graph pattern  
<http://www.example.org/#> vCard:N \_:a .  
\_:a vCard:Given "John"
  - ▶ If resulting graph pattern is subset of original graph, the variable substitution is a query answer

116/139

## Summary

[Turtle Syntax for RDF](#)[SPARQL](#)[Basic SPARQL Queries](#)  
[Query Answer](#)

117/139

## Required reading

## Further reading

- ▶ Jena SPARQL tutorial:  
<http://jena.sourceforge.net/ARQ/Tutorial/>
- ▶ SPARQL Query Language for RDF:  
<http://www.w3.org/TR/rdf-sparql-query/>
- ▶ Turtle - Terse RDF Triple Language:  
<http://www.dajobe.org/2004/01/turtle/>

118/139