SPARQL
Querying Semantic Web

Erdoğan Doğdu

Notes from "Semantic Web for the Working Ontologist" Book
Some slides are adapted from Dieter Fensel and Federico Facca (University of Innsbruck) notes

Motivation

- Having RDF data available is not enough
  - Need tools to process, transform, and reason with the information
  - Need a way to store the RDF data and interact with it
- Are existing storage systems appropriate to store RDF data?
- Are existing query languages appropriate to query RDF data?

Databases and RDF

- Relational database are a well established technology to store information and provide query support (SQL)
- Relational database have been designed and implemented to store concepts in a predefined (not frequently alterable) schema.
- How can we store the following RDF data in a relational database?

```
<rdf:Description rdf:about="949318">
  <rdf:type rdf:resource="&uni;lecturer"/>
  <uni:name>Dieter Fensel</uni:name>
  <uni:title>University Professor</uni:title>
</rdf:Description>
```

- Several solutions are possible

Databases and RDF

- **Solution 1**: Relational "Traditional" approach

Lecturer		
id	**name**	**title**
949318	Dieter Fensel	University Professor

- **Approach**: We can create a table "Lecturer" to store information about the "Lecturer" RDF Class.
- **Drawbacks**: Every time we need to add new content we have to create a new table -> Not scalable, not dynamic, not based on the RDF principles (TRIPLES)

Databases and RDF

- **Solution 2:** Relational "Triple" based approach

Statement			
Subject	**Predicate**	**ObjectURI**	**ObjectLiteral**
101	102	103	null
101	104		201
101	105		202
103	…	…	null

Resources	
Id	**URI**
101	949318
102	rdf:type
103	uni:lecturer
104	…

Literals	
Id	**Value**
201	Dieter Fensel
202	University Professor
203	…
…	…

- **Approach:** We can create a table to maintain all the triples S P O (and distinguish between URI objects and literals objects).
- **Drawbacks:** We are flexible w.r.t. adding new statements dynamically without any change to the database structure… but what about querying?

Why Native RDF Repositories?

- What happens if I want to find the names of all the lecturers?
- **Solution 1**: Relation "traditional" approach:

```
SELECT NAME FROM LECTURER
```

- We need to query a single table which is easy, quick and performing
- No **JOIN** required (the most expensive operation in a db query)
- **BUT** we already said that Traditional approach is not appropriate

Why Native RDF Repositories?

- What happens if I want to find the names of all the lecturers?
- **Solution 2:** Relational "triple" based approach:

```
SELECT L.Value FROM Literals AS L
  INNER JOIN Statement AS S ON
  S.ObjectLiteral=L.ID
  INNER JOIN Resources AS R ON R.ID=S.Predicate
  INNER JOIN Statement AS S1 ON
  S1.Predicate=S.Predicate
  INNER JOIN Resources AS R1 ON
  R1.ID=S1.Predicate
  INNER JOIN Resources AS R2 ON
  R2.ID=S1.ObjectURI
  WHERE R.URI = "uni:name"
  AND R1.URI = "rdf:type"
  AND R2.URI = "uni:lecturer"
```

Why Native RDF Repositories?

Solution 2

- The query is quite complex: 5 JOINS!
- This require a lot of optimization specific for RDF and triple data storage, that it is not included in Relational DB
- For achieving efficiency a layer on top of a database is required. More, SQL is not appropriate to extract RDF fragments
- *Do we need a new query language?*

Query Languages

- *Querying* and inferencing is the very purpose of information representation in a machine-accessible way
- A query language is a language that allows a user to retrieve information from a "**data source**"
  - E.g. data sources
    - A simple text file
    - XML file
    - A database
    - The "Web"
- Query languages usually includes **insert** and **update** operations

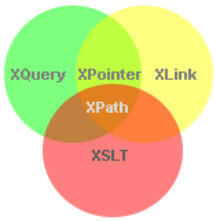
Example of Query Languages

- *SQL*
  - Query language for relational databases
- *XQuery*, *XPointer* and *XPath*
  - Query languages for XML data sources
- *SPARQL*
  - Query language for RDF graphs
- *RDQL*
  - Query language for RDF in Jena models

XPath: a simple query language for XML trees

- The basis for most XML query languages
  - Selection of document parts
  - Search context: ordered set of nodes
- Used extensively in XSLT
  - XPath itself has non-XML syntax
- Navigate through the XML Tree
  - Similar to a file system (“/“, “../“, “ ./“, etc.)
  - Query result is the final search context, usually a set of nodes
  - Filters can modify the search context
  - Selection of nodes by element names, attribute names, type, content, value, relations
- Several pre-defined functions
- Version 1.0, **W3C Recommendation 16 November 1999**
- Version 2.0, **W3C Recommendation 23 January 2007**

Other XML Query Languages



- XQuery
  - Building up on the same functions and data types as XPath
  - With XPath 2.0 these two languages get closer
  - XQuery is not XML based, but there is an XML notation (XQueryX)
  - XQuery 1.0, W3C Recommendation 23 January 2007
- XLink 1.0, W3C Recommendation 27 June 2001
  - Defines a standard way of creating hyperlinks in XML documents
- XPointer 1.0, W3C Candidate Recommendation
  - Allows the hyperlinks to point to more specific parts (fragments) in the XML document
- XSLT 2.0, W3C Recommendation 23 January 2007

Why a New Language?

- RDF description (1):

```
<rdf:Description rdf:about="949318">
    <rdf:type rdf:resource="&uni;lecturer"/>
    <uni:name>Dieter Fensel</uni:name>
    <uni:title>University Professor</uni:title>
</rdf:Description>
```

- XPath query:

```
/rdf:Description[rdf:type=
"http://www.mydomain.org/uni-ns#lecturer"]/uni:name
```

Why a New Language?

- RDF description (2):

```
<uni:lecturer rdf:about="949318">
    <uni:name>Dieter Fensel</uni:name>
    <uni:title>University Professor</uni:title>
</uni:lecturer>
```

- XPath query:

```
//uni:lecturer/uni:name
```

Why a New Language?

- RDF description (3):

```
<uni:lecturer rdf:about="949318"
              uni:name="Dieter Fensel"
                   uni:title="University Professor"/>
```

- XPath query:

```
//uni:lecturer/@uni:name
```

Why a New Language?

- What is the difference between these three definitions?
- RDF description (1):

```
<rdf:Description rdf:about="949318">
     <rdf:type rdf:resource="&uni;lecturer"/>
     <uni:name>Dieter Fensel</uni:name>
     <uni:title>University Professor</uni:title>
</rdf:Description>
```

- RDF description (2):

```
<uni:lecturer rdf:about="949318">
     <uni:name>Dieter Fensel</uni:name>
     <uni:title>University Professor</uni:title>
</uni:lecturer>
```

- RDF description (3):

```
<uni:lecturer rdf:about="949318"
                      uni:name="Dieter Fensel"
                           uni:title="University Professor"/>
```

Why a New Language?

- All three description denote the same thing:

```
⟨#949318, rdf:type, <uni:lecturer>⟩
⟨#949318, <uni:name>, "Dieter Fensel"⟩
⟨#949318, <uni:title>, "University Professor"⟩
```

- But the queries are different depending on a particular serialization:

```
/rdf:Description[rdf:type=
"http://www.mydomain.org/uni-ns#lecturer"]/uni:name

//uni:lecturer/uni:name

//uni:lecturer/@uni:name
```

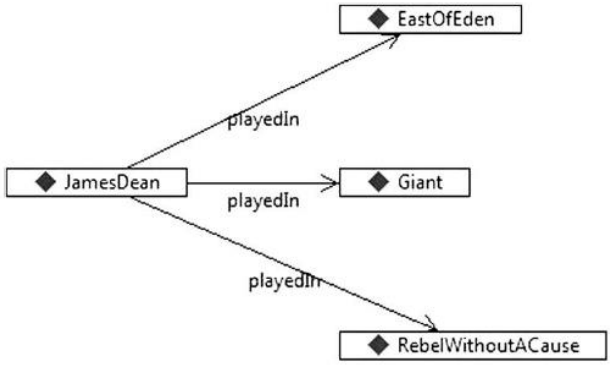
SPARQL

- The standard query language of SemWeb
- SPARQL Protocol And RDF Query Language
- Similar to Xquery and SQL
- Based on specifying RDF triple patterns

- See: http://www.w3.org/TR/rdf-sparql-query for all features of SPARQL

Data

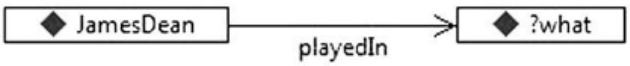
- Movies “James Dean” played in…



19

Query

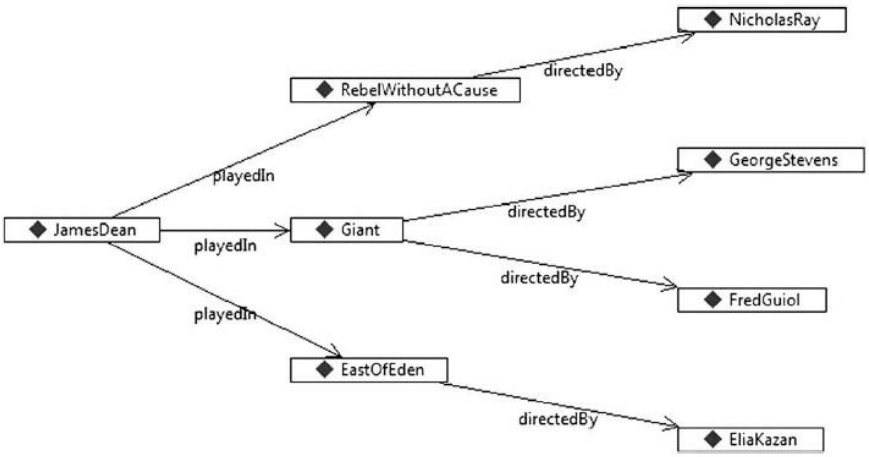
- **What** did James Dean played in?
- Query pattern:



- :JamesDean :playedIn ?what

20

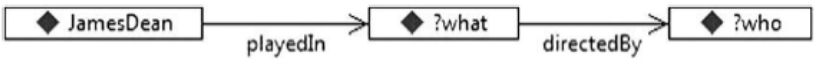
Data



21

Query

- Who directed the movies that James Dean played in?



- :JamesDean :playedIn ?what
- ?what :directedBy ?who

22

Queries in SPARQL

- **What** did James Dean played in?

  SELECT ?what
  WHERE { :JamesDean :playedIn ?what }

  Answer:
  :Giant, :EastOfEden, :RebelWithoutaCause

- **Who** directed the movies that James Dean played in?

  SELECT ?who
  WHERE { :JamesDean :playedIn ?what .
  ?what :directedBy ?who .}

  Answer:
  :GeorgeStevens, :EliaKazan, :NicholasRay, :FredGuiol

23

Query

- Movies and their directors in which James Dean played in?

  SELECT **?what ?who**
  WHERE { :JamesDean :playedIn ?what .
  ?what :directedBy ?who .}

Answer:

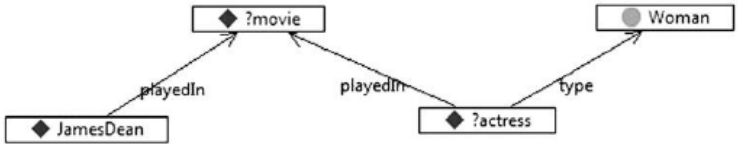
?what	?who
:Giant	:GeorgeStevens
:Giant	:FredGuiol
:EastOfEden	:EliaKazan
:RebelWithoutaCause	:NicholasRay

24

Query

- Actresses who played with James Dean in the same movies

```
SELECT ?actress ?movie
WHERE { :JamesDean :playedIn ?movie .
        ?actress :playedIn ?movie .
        ?actress rdf:type :Woman }
```



25

Query

- What is the following query for?

```
SELECT ?actress ?movie
WHERE { :JamesDean :playedIn ?movie .
        ?actress :playedIn ?movie .
        ?actress a :Woman .
        ?actress :playedIn ?anotherMovie .
        ?anotherMovie :directedBy :JohnFord .}
```

26

Properties

- SELECT ?property ?value
  WHERE {:JamesDean ?property ?value}

?property	?value
bornOn	1931-02-08
diedOn	1955-09-30
playedIn	RebelWithoutaCause
playedIn	EastOfEden
playedIn	Giant
rdf:type	Man
rdfs:label	James Dean

27

Properties

- SELECT ?property
  WHERE {:JamesDean ?property ?value}

?property
bornOn
diedOn
playedIn
playedIn
playedIn
rdf:type
rdfs:label

28

Properties

- SELECT **DISTINCT** ?property
  WHERE { :JamesDean ?property ?value}

?property
bornOn
diedOn
playedIn
rdf:type
rdfs:label

29

Querying schema

- What do Actors do?

SELECT DISTINCT ?property
WHERE { ?q0 **a** :Actor .
?q0 **?property** ?object .}

?property
bornOn
diedOn
playedIn
rdf:type
rdfs:label
produced
sang
wrote

30

Querying schema

- Find all classes

  SELECT DISTINCT ?class WHERE {?q0 a ?class}
- Find all properties

  SELECT DISTINCT ?property

  WHERE {?q0 ?property ?q1}

31

FILTER

- Excluding some results
- Actors who played in East of Eden, who were born in 1930 or later?

  SELECT ?actor

  WHERE {?actor :playedIn :EastOfEden .

  FILTER (?birthday > "1930-01-01"^^xsd:date)}

  Answer: (none)

  Why?

32

FILTER

- Correct query

```
SELECT ?actor
WHERE {
   ?actor :playedIn :EastOfEden .
   ?actor :bornOn ?birthday .
   FILTER (?birthday > "1930-01-01"^^xsd:date)
}
```

- Another one

```
SELECT ?person
WHERE { ?person a :Person .
   ?person :bornOn ?birthday .
   FILTER (?birthday > "Jan 1, 1960"^^xsd:date)
   FILTER (?birthday < "Dec 31, 1969"^^xsd:date) }
```

33

OPTIONAL

- Actors who played in Giant and their death date

```
SELECT ?actor ?deathdate
WHERE {  ?actor :playedIn :Giant .
   ?actor :diedOn ?deathdate .}
```

actor	deathdate
RockHudson	1985-10-02
JamesDean	1955-10-30
. . .	

- If they did not die (yet)

```
SELECT ?actor ?deathdate
WHERE {?actor :playedIn :Giant .
OPTIONAL {
   ?actor :diedOn ?deathdate .}}
```

Actor	deathdate
RockHudson	1985-10-02
JamesDean	1955-10-30
Elizabeth Taylor	*(no binding)*
. . .	

34

Negation

- SPARQL 1.1
- By specifying that certain triples do not exist
- UNSAID
  - Find a matching graph for which UNSAID pattern does not exist
- All of the living actors who played in Giant.

```
SELECT ?actor
WHERE { ?actor :playedIn :Giant .
        UNSAID {?actor :diedOn ?deathdate .} }
```

35

ASK query

- Yes/no questions
- Is Elizabeth Taylor alive?

```
ASK WHERE {:ElizabethTaylor :diedOn ?any}
```

Answer: Yes

- Correct query

```
ASK WHERE {
  UNSAID { :ElizabethTaylor :diedOn ?any}
}
```

Answer: No

36

ASK query

- Was any actor in Giant born after 1950?

```
ASK WHERE {
    ?any :playedIn :Giant.
    ?any :bornOn ?birthday .
    FILTER (?birthday > "1950-01-01"^^xsd:date)
}
```

37

CONSTRUCT query

- Query that returns a graph (triples)

```
CONSTRUCT {  ?d rdf:type :Director .
             ?d rdfs:label ?name . }
WHERE { ?any :directedBy ?d .
        ?d rdfs:label ?name . }
```

38

Using SPARQL as a rule language

- CONSTRUCT {?q1 :hasSon :q2 .}
  WHERE {?q2 :hasFather ?q1}
- No
  - CONSTRUCT {?q1 :hasSon :q2 .}
    WHERE {?q2 :hasFather ?q1. **?q2 a :Man.** }
  - CONSTRUCT {?q1 :hasDaughter :q2 .}
    WHERE {?q2 :hasFather ?q1. **?q2 a :Woman.** }

39

Using SPARQL as a rule language

- CONSTRUCT {?q1 a :Mortal}
  WHERE {?q1 a :Man}
- CONSTRUCT {?q1 :hasUncle ?q2}
  WHERE {?q2 :hasSister ?s .
  ?q1 :hasMother ?s .}
- CONSTRUCT {?q1 :**hasSibling** ?q2} WHERE {?q1 :hasBrother ?q2}
- CONSTRUCT {?q1 :**hasSibling** ?q2} WHERE {?q1 :hasSister ?q2}
- CONSTRUCT {?q1 :**hasParent** ?q2} WHERE {?q1 :hasFather ?q2}
- CONSTRUCT {?q1 :**hasParent** ?q2} WHERE {?q1 :hasMother ?q2}

40

Transitive queries

- (SPARQL 1.1)
- Children of Joe
  - SELECT ?member
    WHERE {?member :hasParent :Joe}
- Grandchildren of Joe
  - SELECT ?member
    WHERE { ?int :hasParent :Joe .
    ?member :hasParent ?int .}

41

Transitive queries

- All children, grandchildren, great-grandchildren, etc.
  - SELECT ?member
    WHERE {?member **:hasParent*** :Joe .}
    - The result includes Joe as well
  - SELECT ?member
    WHERE {?member **:hasParent+** :Joe .}
    - At least one triple in the chain should exist, Joe is excluded (correct query)

42

Federating SPARQL Queries

- Querying from more than one data source
  - Via SPARQL endpoints (SERVICE)
  - Or via named graphs (GRAPH)

```
SELECT ?entry
WHERE {  ?actor :playedIn :Giant .
         ?actor rdfs:label ?name .
         SERVICE <http://dbpedia.org/sparql>
                 { ?entry rdfs:label ?name .}
}
```

43

ORDER

- Order by date

```
SELECT ?title ?date
WHERE { :JamesDean :playedIn ?movie.
  ?movie rdfs:label ?title .
  ?movie dc:date ?date . }
ORDER BY ?date
```

44

LIMIT

- Earliest James Dean movie

```
SELECT ?title
WHERE { :JamesDean :playedIn ?m.
   ?m rdfs:label ?title .
   ?m dc:date ?date . }
ORDER BY ?date
LIMIT 1
```

45

Aggregates and grouping

- (SPARQL 1.1)
- SELECT (**COUNT** (?movie) AS ?howmany)
  WHERE {:JamesDean ?playedIn ?movie .}
- SELECT (**SUM** (?val) AS ?total)
  WHERE { ?s a :Sale .
  ?s :amount ?val }
- SELECT ?year (SUM (?val) AS ?total)
  WHERE { ?s a :Sale .
  ?s :amount ?val . ?s :year ?year }
  GROUP BY ?year

46

Subqueries (SPARQL 1.1)

```
SELECT ?company
WHERE {
    { SELECT ?company ((SUM(?val)) AS ?total09)
      WHERE {
      ?s a :Sale .
      ?s :amount ?val .
      ?s :company ?company .
      ?s :year 2009 . }
      GROUP BY ?company } .
      {SELECT ?company ((SUM(?val)) AS ?total10)
      WHERE {
      ?s a :Sale .
      ?s :amount ?val .
      ?s :company ?company .
      ?s :year 2010 .}
      GROUP BY ?company } .
      FILTER (?total10 > ?total09) . }
```

47

UNION

- All the actors who played either in Rebel Without a Cause **or** Giant.

```
SELECT ?actor
WHERE {
   { ?actor :playedIn :Giant . }
   UNION
   { ?actor :playedIn :RebelWithoutaCause . }
}
```

48

Assignment

- SELECT (fn:concat (?first, " ", ?last) **AS** ?fullname)

  WHERE {:WorkingOntologist dc:creator ?author .

  ?author :firstName ?first .

  ?author :lastName ?last .

  }

49

References

- http://www.w3.org/TR/rdf-sparql-query
- http://dig.csail.mit.edu/2010/Courses/6.898/resources/sparql-tutorial.pdf
- Semantic Web for the Working Ontologist, 2nd Ed., Wiley

50