QUERYING THE SEMANTIC WEB- SPARQL

Dr Edlira Vakaj Edlira.Vakaj@bcu.ac.uk - CMP7173

QUERYING THE SEMANTIC WEB

- Resource Description Framework (RDF) provides a simple way to represent distributed data.
- A triple is the simples way to represent a named connection between two notes.
- Triple stores are generated and some means of accessing that data is needed.
- SPARQL is the standard **query language to access RDF data**.
- **SPARQL** Protocol and RDF Query Language

SPARQL SYNTAX

- Version 1.0 released in 2008
- Version 1.1 released in 2013
- SPARQL query patterns are represented in a variant of Turtle
- Shares many features with other query languages like XQUERY AND SQL

SELECT

- A SPARQL SELECT has two parts: a set of question words, and a question pattern.
- WHERE indicates the selection pattern and can be seen as a graph pattern matched against the data graph.

```
SELECT ?what
FROM file:JamesDean.ttl
WHERE {jd6:JamesDean jd6:playedIn ?what}
```

SELECT EXAMPLE- MOVIES JAMES DEAN PLAYED IN

Pattern with one triple:
:JamesDean as subject
:playedIn as the predicate
?what as the object

Any word

```
SELECT ?what
FROM file:JamesDean.ttl
WHERE {jd6:JamesDean jd6:playedIn ?what}
```

Results table

	what
1	jd6:EastOfEden
2	jd6:Giant
3	jd6:RebelWithoutaCause

```
PREFIX file: <https://swwo.linked.data.world/d/chapter-6-examples/file/>
PREFIX jd6:     <http://www.workingontologist.org/Examples/Chapter6/JamesDean#>
```

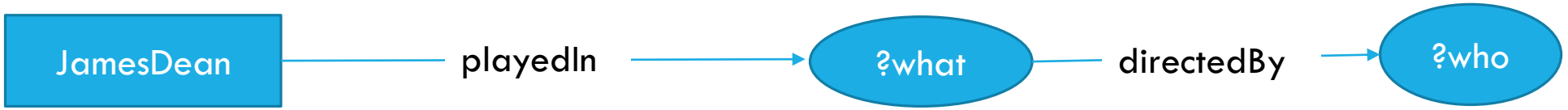
Link to query in DataWorld

SELECT EXAMPLE- WHO DIRECTED THE MOVIES JAMES DEAN PLAYED IN

Graph pattern with two triples:

```
SELECT ?what ?who
FROM file:JamesDean.ttl
WHERE {jd6:JamesDean jd6:playedIn ?what .
       ?what jd6:directedBy ?who .}
```

what	who
jd6:EastOfEden	jd6:EliaKazan
jd6:Giant	jd6:FredGuiol
jd6:Giant	jd6:GeorgeStephens
jd6:RebelWithoutaCause	jd6:NicholasRay



SELECT EXAMPLE- WOMAN WHO WORKED WITH JAMES DEAN AND JOHN FORD

```
SELECT ?actress
FROM file:JamesDean.ttl
WHERE {jd6:JamesDean jd6:playedIn ?q1 .
       ?actress jd6:playedIn ?q1 .
       ?actress rdf:type jd6:Woman .
       ?actress jd6:playedIn ?q2 .
       ?q2 jd6:directedBy jd6:JohnFord . }
```

	actress
1	jd6:CarrollBaker
2	jd6:NatalieWood

QUERYING FOR PROPERTIES AND SCHEMA

- SPARQL pattern – matching allows predicates to be matched as well.
- This ability to query for properties in the data distinguishes SPARQL from many other query languages!

```
SELECT ?property
FROM file:JamesDean.ttl
WHERE { jd6:JamesDean ?property ?value }
```

	property	value
1	rdf:type	jd6:Actor
2	rdf:type	jd6:Man
3	rdfs:label	James Dean
4	jd6:bornOn	1931-02-08
5	jd6:diedOn	1955-10-30
6	jd6:playedIn	jd6:EastOfEden
7	jd6:playedIn	jd6:Giant
8	jd6:playedIn	jd6:RebelWithoutaCause

QUERYING FOR PROPERTIES AND SCHEMA

- DISTINCT to filter out the duplicate results.

What do you know about any actor?

```
SELECT DISTINCT ?property
FROM file:JamesDean.ttl
WHERE { ?q0 a jd6:Actor .
        ?q0 ?property ?value }
```

	property
1	rdf:type
2	rdfs:label
3	jd6:bornOn
4	jd6:diedOn
5	jd6:playedIn

VARIABLES, BINDINGS, AND FILTERS

- ?deathdate, ?actor - variables
- FILTER – to define a condition on one or more variables under which rows will be excluded from the result.
- Uses a Boolean test.

Which of the actors who played in Giant lived more than five years after the movie was made?

```
SELECT ?actor ?deathdate
FROM file:JamesDean.ttl
WHERE { ?actor jd6:playedIn jd6:Giant ;
               jd6:diedOn ?deathdate .
            FILTER (?deathdate  > "1961-11-24"^^xsd:date)}
```

OPTIONAL PATTERNS (2)

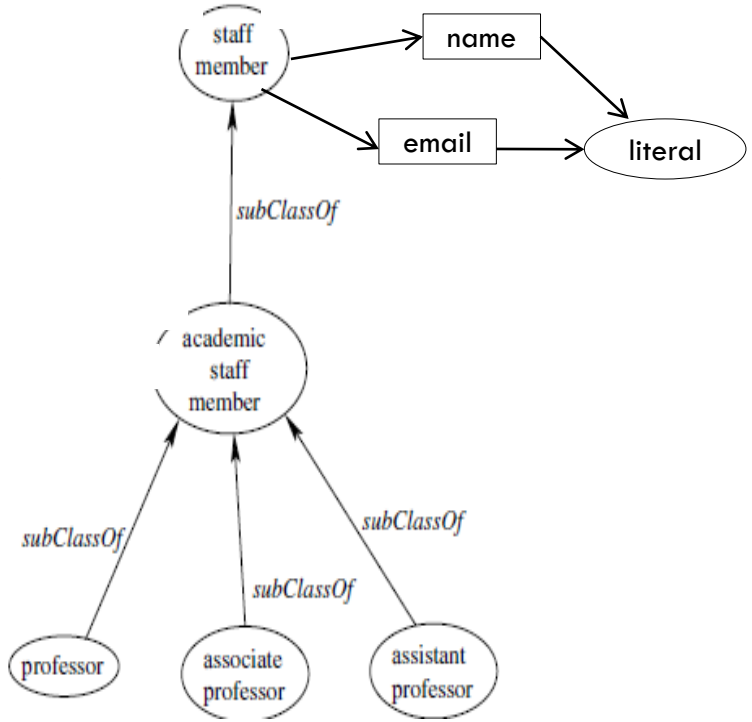
All lecturers names and their email addresses:

```
SELECT ?name ?email
WHERE
{ ?x rdf:type uni:academicStaffMember ;
        uni:name ?name ;
        uni:email ?email .
}
```

- The result of the previous query would be:

?name	?email
David Billington	**david@work.example.org**

Grigoris Antoniou is listed as a lecturer, but he has no e-mail address




```
<uni:assistanceProfessor  rdf:about="949352">
  <uni:name>Grigoris Antoniou</uni:name>
</uni:assistanceProfessor>

<uni:professor rdf:about="94318">
  <uni:name>David Billington</uni:name>
  <uni:email>david@work.example.org</uni:email>
</uni:professor>
```

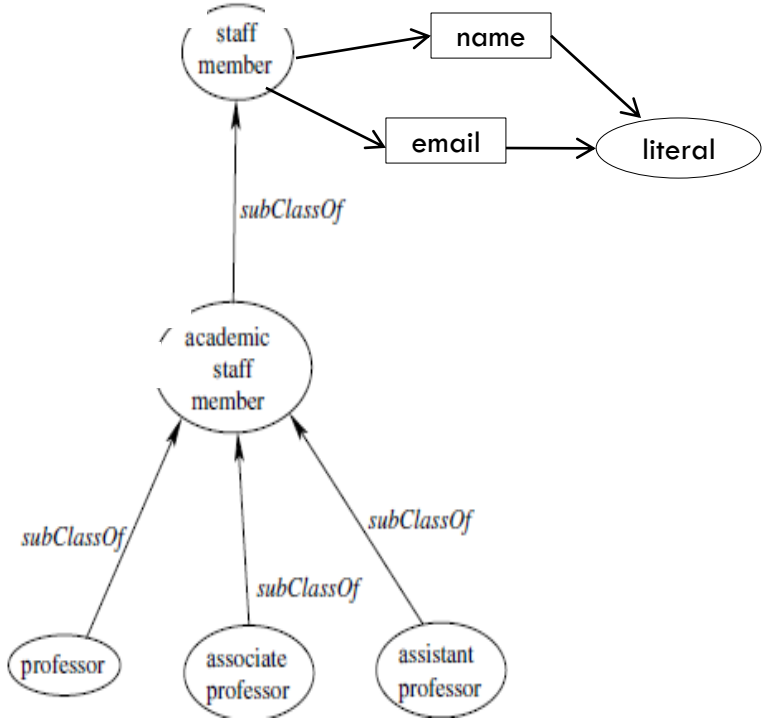
OPTIONAL PATTERNS (2)

As a solution we can adapt the query to use an optional pattern:

```
SELECT ?name ?email
WHERE
{ ?x rdf:type uni:academicStaffMember ;
        uni:name ?name .
        OPTIONAL { x? uni:email ?email }
}
```

- The meaning is roughly "give us the names of lecturers, and if known also their e-mail address"
- The result looks like this:

?name	?email
Grigoris Antoniou	
David Billington	**david@work.example.org**




```
<uni:assistanceProfessor  rdf:about="949352">
  <uni:name>Grigoris Antoniou</uni:name>
</uni:assistanceProfessor>

<uni:professor rdf:about="94318">
  <uni:name>David Billington</uni:name>
  <uni:email>david@work.example.org</uni:email>
</uni:professor>
```

NEGATION- SPARQL 1.1

- What if we want to specify that there are certain triples that are not it the dataset?!
- MINUS
- FILTER NOT EXISTS
- *Find all of the living actors (no death date recorded in the data) who played in the East of Eden.*

```
SELECT ?actor ?deathdate
FROM file:JamesDean.ttl
WHERE { ?actor jd6:playedIn jd6:Giant .
        FILTER NOT EXISTS { ?actor jd6:diedOn ?deathdate .  } }
```

YES/NO QUERIES

SPARQL uses the keyword ASK at the beginning of the query and it return either *true* of *false*.

- *Is Elizabeth Is Elizabeth Taylor dead?*

```
ASK
FROM file:JamesDean.ttl
WHERE { jd6:ElizabethTaylor jd6:diedOn ?any }
```

- *Is Elizabeth Is Elizabeth Taylor alive?*

```
ASK
FROM file:JamesDean.ttl
WHERE {FILTER NOT EXISTS { jd6:ElizabethTaylor jd6:diedOn ?any }}
```

Combined with FILTER NOT EXIST

CONSTRUCT

- CONSTRUCT uses the expressive power of RDF in the answer to a query, as well as in the match pattern.
- The result is not a table or a single bit but an RDF graph

Directors are people who direct movies

```
CONSTRUCT {?d rdf:type jd6:Director .
           ?d rdfs:label ?name . }
FROM file:JamesDean.ttl
WHERE {?any jd6:directedBy ?d .
       ?d rdfs:label ?name . }
```

Result in Turtle

```
jd6:JohnFord  rdf:type  jd6:Director ;
        rdfs:label  "John Ford" .

jd6:EliaKazan  rdf:type  jd6:Director ;
        rdfs:label  "Elia Kazan" .

jd6:FredGuiol  rdf:type  jd6:Director ;
        rdfs:label  "Fred Guiol " .

jd6:GeorgeStephens  rdf:type  jd6:Director ;
        rdfs:label  "George Stephens" .

jd6:NicholasRay  rdf:type  jd6:Director ;
        rdfs:label  "Nicholas Ray" .
```

USING RESULTS OF CONSTRUCT QUERIES

Sophisticated RDF query systems offer a variety of options for what to do with the constructed triples:

- Insert the constructed triples back into the original data source that the query was run against
- Store the constructed triples as a separate graph
- Store the constructed triples into a new dataset or database
- Serialise the results in some standard format and save them to a file.

RESOURCES

Running Examples:

http://workingontologist.org/index.html

Dataset:

https://data.world/swwo/chapter-6-examples/workspace/file?filename=JamesDean.ttl