Introduction to Semantic Web Databases

Prepared By:

Amgad Madkour
Ph.D. Candidate
Purdue University

http://amgadmadkour.github.io

Last Updated: November 19, 2018
Semantic Web – Motivation

• Represents the next generation of the world wide web (Web 3.0)

• Aims at converting the current web into a web of data

• Intended for realizing the machine-understandable web

• Allows combining data from several applications to arrive at new information
What is the Semantic Web?

• A set of standards

• Defines best practices for sharing data over the web for use by applications

• Allows defining the semantics of data
  • Example:
    • Spouse is a symmetric relations (if A spouse of B then B spouse of A)
    • zip codes are a subset of postal codes
    • “sell” is the opposite of “buy”
Semantic Web – Standardization

• The World Wide Web Consortium (W3C) developed a number of standards around the Semantic Web:

1. Data Model (RDF)
2. Query languages (SPARQL)
3. Ontology languages (RDF Schema and OWL variants)
Semantic Web – Use Cases

• Many Semantic Web components (e.g. RDF and SPARQL) are used in various domains:
  • Semantic Search (Google, Microsoft, Amazon)
  • Smart Governments (data.gov.us, data.gov.uk)
  • Pharmaceutical Companies (AstraZeneca)
  • Automation (Siemens)
  • Mass Media (Thomson Reuters)
Semantic Web – Technology Stack

• **Hypertext Web Technologies**
  • **IRI**: Generalization of URI
  • **Unicode**: Language support
  • **XML**: Create documents of structured data

• **Standardized Semantic Web Technologies**
  • **RDF**: Creating statements (triples)
  • **RDFS**: RDF Schema of classes and properties
  • **OWL**: Extends RDFS by adding constructs
  • **SPARQL**: Query RDF-based data
  • **RIF**: Rule interchange format, goes beyond OWL
Resource Description Framework (RDF)

• Is the **standard** for **representing knowledge**

• RDF expresses information as a list of **statements** known as **triples**

• A **triple** consists of:
  
  **SUBJECT**, **PREDICATE**, and an **OBJECT**
  
  • **Example**: (“Muhammad Ali”, “isA”, “Boxer”)
RDF Model
Triple Structure

• Subjects, predicates, and objects are represented by resources or literals

• A resource is represented by a URI and denotes a named thing

• Literals represent a string or a number

• Literals representing values other than strings may have an attached datatype
RDF Model
Anonymous Resources

- RDF allows one special case of resources where the **URI is not known**

- An **anonymous resource** is represented as having a blank identity or a **blank node/bnode**

- A blank node can only be used as **subject** or **object** of a triple
RDF Model
Namespaces

• URI’s allow defining distinct identities to RDF resources

• Each RDF dataset provider can define common RDF resources using its own namespace
  • Example:
    • http://dbpedia.org/resource/Muhammad_Ali
    • http://www.wikipedia.org/Muhammad_Ali

• URI’s representing the namespace can be replaced with a prefix
  • Example:
    • dbp:Muhammad_Ali
    • wiki:Muhammad_Ali

• The namespace can be defined in an RDF document using @prefix
  • Example:
    • @prefix dbp: http://dbpedia.org/resource/
    • @prefix wiki: http://www.wikipedia.org/
RDF Model
Storing RDF Files

• RDF can be serialized using
  • N-Triple
  • Notation 3/Turtle
  • RDF/XML

• The standardized formats by W3C are RDF/XML and Turtle

• Notation 3 is similar to Turtle but includes more enhanced features

• Notation 3 is being developed by Tim Berners-Lee
RDF Model
Storing RDF Files - N-Triple Format

RDF Model
Storing RDF Files - Notation 3/Turtle Format

@prefix dbp: <http://dbpedia.org/resource> .
@prefix dbo: <http://dbpedia.org/ontology> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

dbp:Muhammed_Ali
  dbo:birthPlace dbp:Louisville,_Kentucky ;
  dbo:birthDate "1942-01-17"^^xsd:date ;
  foaf:name "Muhammad Ali"@en .

@prefix dbp: <http://dbpedia.org/resource> .
@prefix dbo: <http://dbpedia.org/ontology> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/2001/12/ rdf-syntax-ns#>

dbp:Muhammed_Ali rdf:type
  foaf:Person ;
  dbo:Boxer ,
  dbo:Agent .

Representing multiple predicate, object per subject
Representing multiple objects per predicate of a subject
RDF Model

Data Typing

• Non-URI values are called literals

• Literals have a datatype assigned to them

@prefix dbp: <http://dbpedia.org/resource>  .
@prefix dbo: <http://dbpedia.org/ontology>  .
@prefix dbpr: <http://dbpedia.org/property/>  .

dbp:Muhammed_Ali  dbo:birthDate “1942-01-17”^^xsd:date  .
RDF Model
Labeling and Tagging

• RDF Queries can be narrowed down to literals tagged in a particular language

• One of RDF best practices is to assign a label (i.e. rdfs:label) values to resources and tag them with a language

```rdf
@prefix dbp: <http://dbpedia.org/resource> .
@prefix rdf: <http://www.w3.org/2001/XMLSchema#> .

dbp:Muhammed_Ali  rdf:label
    “Muhammad Ali”@en ,
    “モハメド・アリ”@ja ,
    "محمد علي" @ar .
```
RDF Model
Blank Nodes

• Blank nodes have **no permanent identity**

• Used to **group** together a set of **values**

• Used as a **placeholder** in case other triples need to refer to a blank node grouping

```reasoning
@prefix dbp: <http://dbpedia.org/resource> .
@prefix ex:    <http://example.org/>

dbp:Muhammed_Ali ex:info _:b1 .
_:b1 ex:firstName "Muhammad" ;
ex:lastName  "Ali" .
```
RDF Model
Vocabularies

- **Vocabulary** (i.e. new URI’s) can be **created** or **resused**

- Existing vocabularies (e.g. Friend of a Friend - FOAF) are stored using, e.g., **RDF schema (RDFS)**

- The **RDF Vocabulary Description Language** (RDF Schema) allows describing vocabularies

- RDF Schema allows defining **properties** or new **classes** of resources
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

dc:creator
    rdf:type rdf:Property ;
    rdfs:comment "Makes a URI"@en-US ;
    rdfs:label "Creator"@en-US .

**Tip:** Another way of specifying rdf:type is using “a”

dc:creator a rdf:Property
@prefix ex: <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

ex:Athlete
    rdf:type rdfs:Class ;
    rdfs:label "Athlete" .

ex:Sport
    a rdfs:Class ;
    rdfs:label "Sport" .
RDF Model

RDF Schema Example

@prefix ex: <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

ex:playsSport
    rdf:type rdf:Property ;
    rdfs:domain ex:Athlete ;
    rdfs:range ex:Sport .

- **rdf:domain**: If a property is `ex:playSport` in a triple then the **subject** is an `ex:Athlete`
- **rdf:range**: If the property is `ex:playSport` in a triple then the **object** is a `ex:Sport`

A query engine can retrieve all resources (e.g. Muhammad Ali) of a specific class (e.g., Athlete) even though there are **no explicit triples** indicating a resource **membership in a class**
Web Ontology Language (OWL)

• A **key technology** for defining **semantics** for RDF data

• OWL extends RDFS to define **ontologies**

• An **ontology** is a **formal definition** of set of vocabulary that define **relationships** between vocabulary **terms** and **class** members

• Ontologies are used to describe **domain knowledge** (e.g. biology) so that users are able to more formally share and understand data

• An ontology defined with OWL is a **collection of triples**
Web Ontology Language (OWL) Example

```w3c-owl
@prefix ex: <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

ex:opponent
   rdf:type owl:SymmetricProperty ;
   rdfs:comment "Identify someone's opponent" .

:Muhammad_Ali
   ex:opponent :Joe_Frazier
```

- :Muhammad_Ali is now known to have an opponent :Joe_Frazier
- No triples for :Joe_Frazier are required to be defined for ex:opponent relation
Linked Data

• RDF allows interlinking datasets either on the **data level** or the **query level**

• **On the data level**: RDF dataset creators can provide “sameAs” dataset that interlinks the same resources across datasets

• **On the query level**: The query engine can be used to merge results from multiple sources

Figure: Linked RDF Data Cloud, containing thousands of datasets
Linked Data
Principles

• Use **URIs** as **names for things**

• Use **HTTP URIs** so that people can **look up** those names

• When someone looks up a URI, **provide** useful **information**, using the standards (RDF*, SPARQL)

• Include **links** to **other URIs** so that they can discover more things
SPARQL Query Language

Overview

• SPARQL (pronounced "sparkle") is an acronym for SPARQL Protocol and RDF Query Language

• SPARQL is an RDF/semantic query language for databases that store RDF data

• SPARQL query can consist of triple patterns, conjunctions, disjunctions, and optional patterns
The conditions of a SPARQL query is specified using **triple patterns**.

Triple patterns are similar to RDF triples but contain **variables**.

Variables add **flexibility** to the triple patterns matching.

---

**Query:** Get the birth date of Muhammad Ali

```sparql
SELECT ?bd
WHERE {
  :Muhammad_Ali dbo:birthDate ?bd .
}
```

**Result:**

```
<table>
<thead>
<tr>
<th>bd</th>
</tr>
</thead>
<tbody>
<tr>
<td>1942-01-17</td>
</tr>
</tbody>
</table>
```
**SPARQL Query Language**

**Multiple Triple Patterns**

**Query:** Get names of all Boxers

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

SELECT ?name
WHERE {
  ?uri rdf:type dbo:Boxer.
}
```

Results include labels in multiple languages as they all match the query triple patterns

<table>
<thead>
<tr>
<th>name</th>
<th>language</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Muhammad Ali&quot;</td>
<td>en</td>
</tr>
<tr>
<td>&quot;محمد علي&quot;</td>
<td>ar</td>
</tr>
<tr>
<td>&quot;Mike Tyson&quot;</td>
<td>ja</td>
</tr>
<tr>
<td>&quot;مايكل تايسيون&quot;</td>
<td>ar</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Two triple patterns joined by ?uri variable
SPARQL Query Language

**Query:** Get names of all Boxers in English

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

SELECT ?name
WHERE {
  ?uri rdf:type dbo:Boxer.
  FILTER ( lang(?name) = 'en' )
}
```

Results are filtered based on the language tag assigned to the label

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Muhammad Ali&quot;@en</td>
</tr>
<tr>
<td>&quot;Mike Tyson&quot;@en</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
SPARQL Query Language

**OPTIONAL**

**Query:** Get names of all Boxers and show nicknames if exists

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
PREFIX foaf: <http://xmlns.com/foaf/0.1/> .

SELECT ?resource ?label ?nickname
WHERE {
  OPTIONAL { ?resource foaf:nick ?nickname . }
  FILTER(lang(?label) = 'en')
}
```

**Note:** The order of the OPTIONAL graph patterns matters in case multiple OPTIONAL patterns exist

**RESULT**

<table>
<thead>
<tr>
<th>lbl</th>
<th>nickname</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Lennox Lewis&quot;@en</td>
<td>&quot;The Lion&quot;@en</td>
</tr>
<tr>
<td>&quot;Mike Tyson&quot;@en</td>
<td>&quot;Iron&quot;@en</td>
</tr>
<tr>
<td>&quot;Mike Tyson&quot;@en</td>
<td>&quot;Kid Dynamite&quot;@en</td>
</tr>
<tr>
<td>&quot;Barbados Joe Walcott&quot;@en</td>
<td>&quot;Barbados Demon&quot;@en</td>
</tr>
<tr>
<td>&quot;Chris Arreola&quot;@en</td>
<td>&quot;The Nightmare&quot;@en</td>
</tr>
<tr>
<td>&quot;Giulian Ilie&quot;@en</td>
<td>&quot;The Dentist&quot;@en</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
SPARQL Query Language
MINUS

**Query:** Get names of all Boxers that do not have a nickname

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
PREFIX foaf: <http://xmlns.com/foaf/0.1/> .

SELECT ?label
WHERE {
  MINUS { ?resource foaf:nick ?nickname . }
}
```

RESULT

<table>
<thead>
<tr>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Franciszek Szymura&quot;@en</td>
</tr>
<tr>
<td>&quot;Victor McLaglen&quot;@en</td>
</tr>
<tr>
<td>&quot;Anders Petersen (boxer)&quot;@en</td>
</tr>
<tr>
<td>&quot;Dick Turpin (boxer)&quot;@en</td>
</tr>
<tr>
<td>&quot;Edward Flynn (boxer)&quot;@en</td>
</tr>
<tr>
<td>&quot;Frederick Wedge&quot;@en</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
SPARQL Query Language
Property Paths – Alternative Paths ( | )

**Query:** Get name or titles of Muhammad Ali

```sparql
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
PREFIX dbp: <http://dbpedia.org/property/> .

SELECT ?var
WHERE {
}
```

**RESULT**

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Muhammad Ali&quot;@en</td>
</tr>
<tr>
<td>&quot;WBA heavyweight champion&quot;^^rdf:langString</td>
</tr>
<tr>
<td>&quot;WBC heavyweight champion&quot;^^rdf:langString</td>
</tr>
<tr>
<td>&quot;Lineal heavyweight champion&quot;^^rdf:langString</td>
</tr>
<tr>
<td>&quot;NABF heavyweight champion&quot;^^rdf:langString</td>
</tr>
<tr>
<td>&quot;The Ring heavyweight champion&quot;^^rdf:langString</td>
</tr>
<tr>
<td>&quot;Undisputed heavyweight champion&quot;^^rdf:langString</td>
</tr>
</tbody>
</table>
QUERY: Get all heavy weight champions before Muhammad Ali

PREFIX dbp: <http://dbpedia.org/property/> .

SELECT ?champions
WHERE {
}

Recursively get all Boxing Heavy-weight Champions before Muhammad Ali

+ → One or more
* → Zero or more
SPARQL Query Language
Property Paths – Using Defined Paths

**Query:** Get all heavy weight champions before Muhammad Ali that are *two links away*

```sparql
PREFIX dbp: <http://dbpedia.org/property/>
SELECT ?s
WHERE {
}
```

**RESULT**

champions

: :Floyd_Patterson
**Query:** Get all heavy weight champions *before* Muhammad Ali

PREFIX dbp: <http://dbpedia.org/property/>

SELECT ?champions
WHERE {
}

Recursively (+) get all Boxing Heavy-weight Champions *before* Muhammad Ali

**RESULT**

<table>
<thead>
<tr>
<th>champions</th>
</tr>
</thead>
<tbody>
<tr>
<td>:John_Tate_(boxer)</td>
</tr>
<tr>
<td>:Leon_Spinks</td>
</tr>
<tr>
<td>:Jimmy_Ellis_(boxer)</td>
</tr>
</tbody>
</table>
**Query:** Get all heavy weight champions that Muhammad Ali is *not before* them

```
PREFIX dbp: <http://dbpedia.org/property/> .

SELECT ?champions
WHERE {
}
```

**Switching** the subject & object and **negating** the predicate achieves the same result as previous query
**SPARQL Query Language**

**DISTINCT - Eliminating Redundant Output**

**Query:** Get all **unique** predicates/relations for the Muhammad Ali

```

SELECT DISTINCT ?predicate
WHERE {
}
```

**RESULT**

<table>
<thead>
<tr>
<th>predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf:type</td>
</tr>
<tr>
<td>rdfs:label</td>
</tr>
<tr>
<td>rdfs:comment</td>
</tr>
<tr>
<td>rdfs:seeAlso</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
**Query**: Get the champion before and after Muhammed Ali

```
PREFIX dbp: <http://dbpedia.org/property/> .

SELECT ?champion
WHERE {
  {?champion dbp:before :Muhammad_Ali .} 
  UNION 
  {?champion dbp:after :Muhammad_Ali .} 
}
```

RESULT

<table>
<thead>
<tr>
<th>Champion</th>
</tr>
</thead>
<tbody>
<tr>
<td>John_Tate_(boxer)</td>
</tr>
<tr>
<td>Leon_Spinks</td>
</tr>
<tr>
<td>Jimmy_Ellis_(boxer)</td>
</tr>
<tr>
<td>Ernie_Terrell</td>
</tr>
<tr>
<td>Joe_Frazier</td>
</tr>
<tr>
<td>Sonny_Liston</td>
</tr>
<tr>
<td>Antonio_Rebollo</td>
</tr>
</tbody>
</table>
SPARQL Query Language
FILTER on Condition - regexp

**Query:** Get matches of Muhammed Ali that contain the word "Undisputed"

```sparql
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX : <http://dbpedia.org/resource/>

SELECT ?title
WHERE {
  FILTER(regex(?title, 'Undisputed', 'i'))
}
```

Filter the results by the word ‘Undisputed’ in a case insensitive fashion (‘i’)

**RESULT**

```
<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Undisputed heavyweight champion&quot;^^rdf:langString</td>
</tr>
<tr>
<td>:List_of_undisputed_boxing_champions</td>
</tr>
</tbody>
</table>
```
**Query:** Get matches of Muhammed Ali that contain the word “Undisputed” and is not a URI

```sparql
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX : <http://dbpedia.org/resource/>

SELECT ?title
WHERE {
  FILTER(regex(?title, 'Undisputed', 'i'))
  FILTER(!isURI(?title))
}
```

**RESULT**

"Undisputed heavyweight champion"^^rdf:langString
SPARQL Query Language
LIMIT and OFFSET

**Query:** Get two titles after the second returned title of Muhammed Ali

```sparql
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX : <http://dbpedia.org/resource/>

SELECT ?title
WHERE {
  FILTER(!isURI(?title))
} OFFSET 1 LIMIT 2
```

RESULT

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;WBC heavyweight champion&quot;^^rdf:langString</td>
</tr>
<tr>
<td>&quot;Lineal heavyweight champion&quot;^^rdf:langString</td>
</tr>
</tbody>
</table>

Skip the first result and limit to 2 following result
SPARQL Query Language
ORDER BY – Sorting Results

**Query:** Get all sorted titles of Muhammed Ali

```
PREFIX dbp: <http://dbpedia.org(property)/>
PREFIX : <http://dbpedia.org/resource/>

SELECT ?title
WHERE {
}
ORDER BY(?title)
```

ORDER BY DESC(?title) can also be used to sort results in a descending order.

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Lineal heavyweight champion&quot;^^rdf:langString</td>
</tr>
<tr>
<td>&quot;NABF heavyweight champion&quot;^^rdf:langString</td>
</tr>
<tr>
<td>&quot;The Ring heavyweight champion&quot;^^rdf:langString</td>
</tr>
<tr>
<td>&quot;Undisputed heavyweight champion&quot;^^rdf:langString</td>
</tr>
<tr>
<td>&quot;WBA heavyweight champion&quot;^^rdf:langString</td>
</tr>
<tr>
<td>&quot;WBC heavyweight champion&quot;^^rdf:langString</td>
</tr>
</tbody>
</table>

RESULT
Semantic Web: Case Study

• Solid (Social Linked Data) is a web decentralization project led by Tim Berners-Lee

• The objective of Solid is to create true data ownership and improved privacy

• Applications and data are separate, allowing users to store personal data where they want

• A user stores personal data in "pods" (personal online data stores)

• Applications are authenticated by Solid and are given access to pods based on the application permission
References

• **Learning SPARQL**, Second Edition
• **RDF Basic Concepts**
  • [https://jena.apache.org/documentation/rdf/index.html](https://jena.apache.org/documentation/rdf/index.html)
• **RDF Tutorial**
  • [https://jena.apache.org/tutorials/rdf_api.html](https://jena.apache.org/tutorials/rdf_api.html)
• **SPARQL Tutorial**
  • [https://jena.apache.org/tutorials/sparql.html](https://jena.apache.org/tutorials/sparql.html)
• **SPARQL Recommendation (W3C)**
  • [https://www.w3.org/TR/sparql11-query/](https://www.w3.org/TR/sparql11-query/)