RSAFG

Research Studio **SAT**

194.083 Hackathon

Kooperative Open Source Entwicklung mit Web of Needs

Before we continue…

https://blazegraph.com/

1. Download
2. Start blazegraph.jar (java –jar blazegraph.jar)
3. Check ip/port combination and access via webbrowser
4. Keep it in the back for later ☺

RSAFG

Research Studio **SAT**

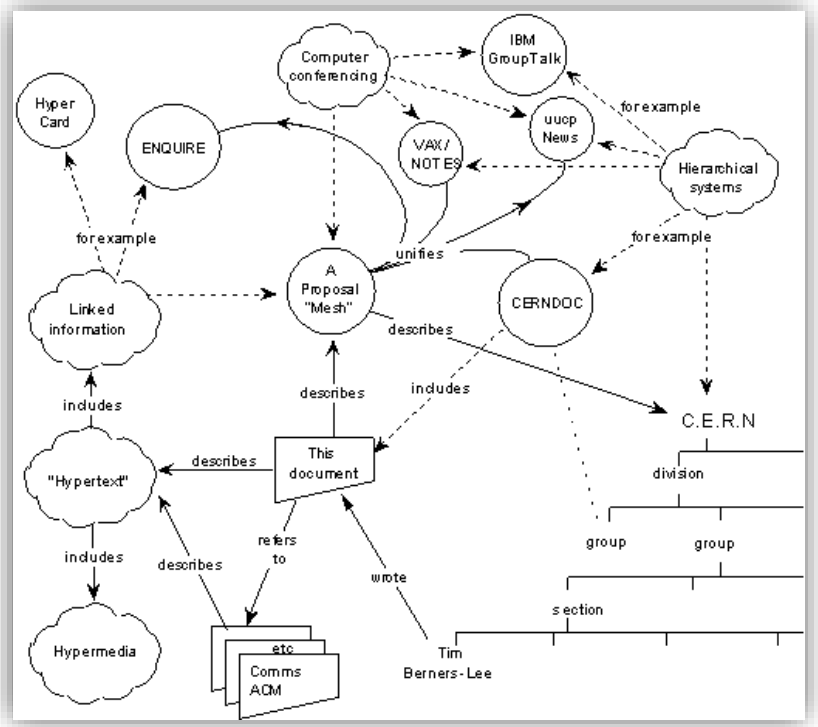
194.083 Hackathon – Semantic Web

Kooperative Open Source Entwicklung mit Web of Needs

Semantic Web

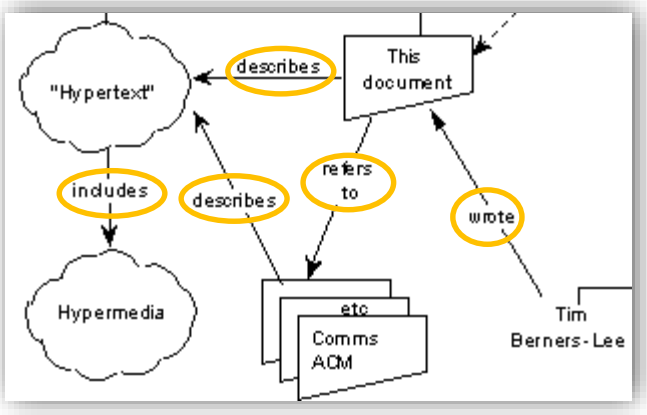
- Definition: „a machine-processable web of smart data […], we can further define smart data that is application-independent, composeable, classified, and part of a larger information ecosystem (ontology).“ [4]
- Term:
  - Web 3.0 – John Markoff
  - Linked Open Data
  - Web of Data – W3C
  - Giant Global Graph (GGG) – Tim Berners Lee

Semantic Web



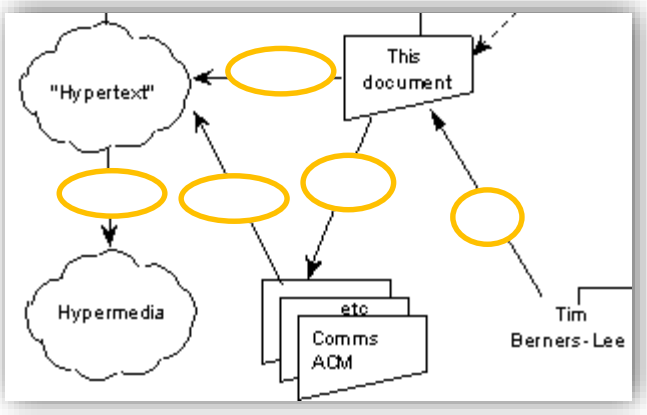
Tim Berners-Lee, CERN
March 1989, May 1990

Semantic Web



Tim Berners-Lee, CERN
March 1989, May 1990

Semantic Web



Tim Berners-Lee, CERN
March 1989, May 1990

Semantic Web

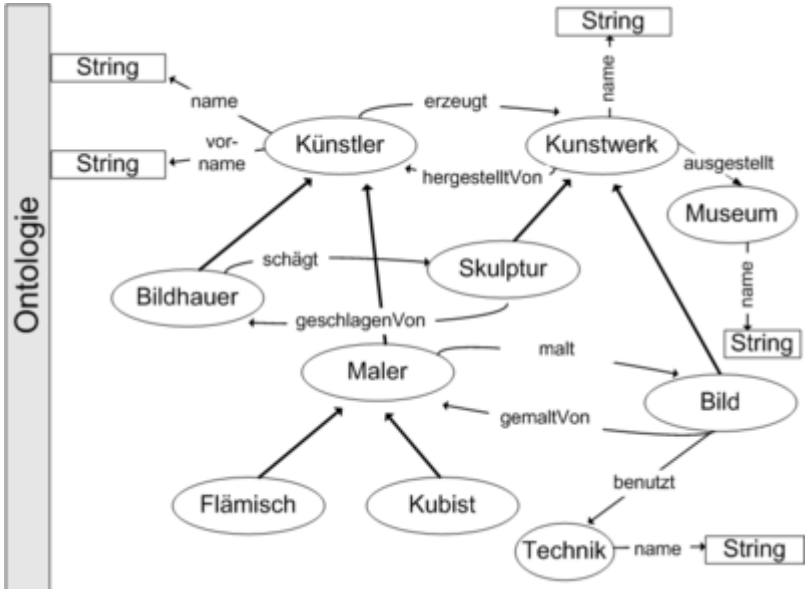
- Datamodel: RDF
- Ontologie Language: OWL
- Query Language: SPARQL
- Serialization Formats : Turtle, RDF/XML, JSON-LD, RDFa

- More:
  - https://www.w3.org/2013/data/

Semantic Web - Ontology:

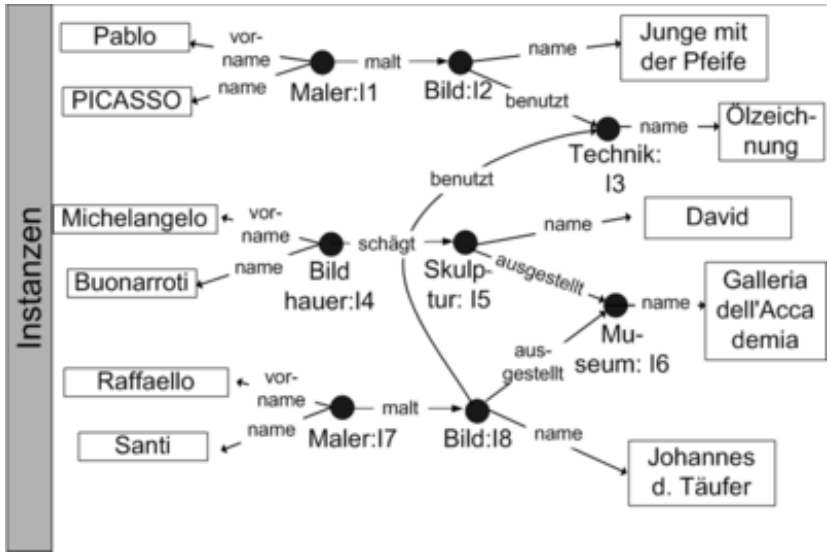
- „Describing and representing an area of knowledge“ [4]
- „Defining the common words and concepts of a description“[4]
- „An ontology models the vocabulary and meaning of domains of interests:
  - the object (things) in domains
  - the relationship among those things
  - the properties, functions & processes involving those things
  - and constraints on and rules about those thins“[4]

Semantic Web - Ontology



https://de.wikipedia.org/wiki/Ontologie_(Informatik)

Semantic Web - Ontology



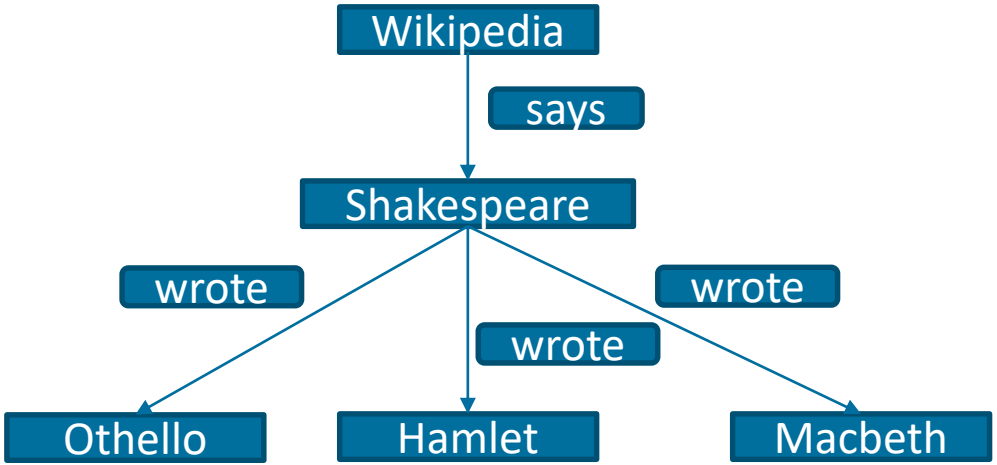
https://de.wikipedia.org/wiki/Ontologie_(Informatik)

Resource Description Framework (RDF)

- https://www.w3.org/RDF/
- Keywords:
  - **Knowledge Graphs**
  - **Triples:** ([Subject] [Predicate] [Object]) .
  - **URIS:** Uniform Resource Identifiers
  - **TTL:** Language Turtle = Terse RDF Triple Language
  - **SPARQL:** Query Language: **S**PARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage

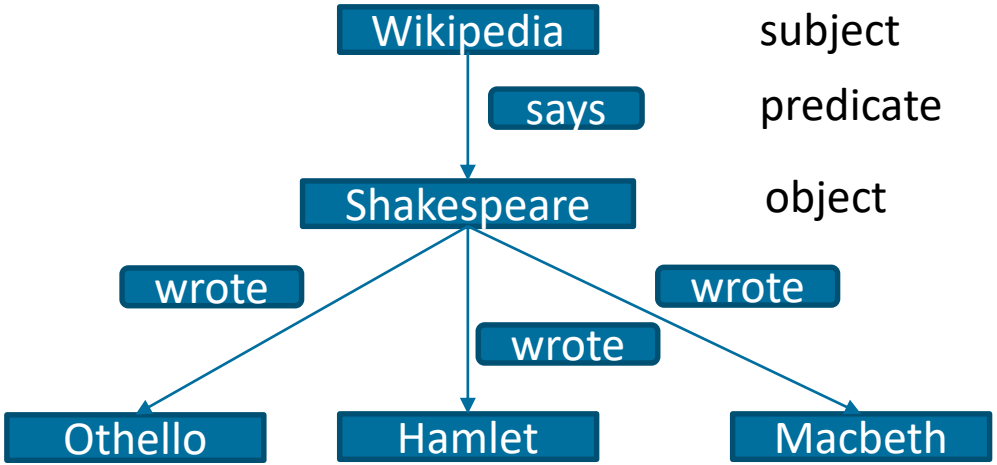
RDF - Triples

- Triples: ([Subject] [Predicate] [Object]) .
- Example: "*Wikipedia says Shakespeare wrote Hamlet*"



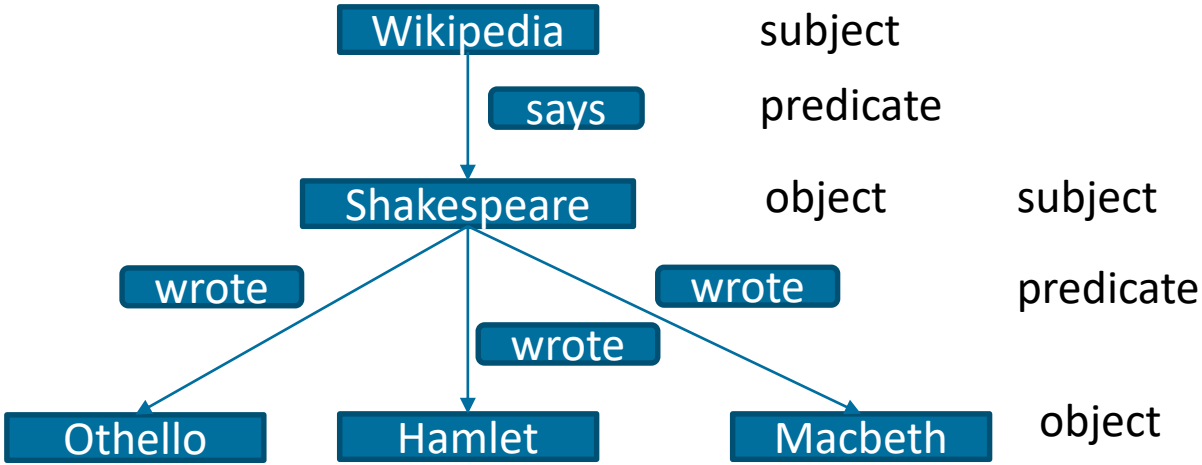
RDF - Triples

- rdf:subject, rdf:predicate, rdf:object .
- Example: "*Wikipedia says Shakespeare wrote Hamlet*"



RDF - Triples

- rdf:subject, rdf:predicate, rdf:object .
- Example: "*Wikipedia says Shakespeare wrote Hamlet*"



RDF Serialization with N-Triples

- URI = main component of triples in W3C Knowledge Graphs
- Using fully unabbreviated URIs
- Triple statement: Sequence of (subject, predicate, object) terms, separated by whitespace and terminated by '.'

“Spiderman is an enemy of Green Goblin”

<http://example.org/#spiderman> <http://www.perceive.net/schemas/relationship/enemyOf> <http://example.org/#green-goblin> .

RDF – N-Quads

- Adding the graph URI to the triple

<http://example.org/#spiderman> <http://www.perceive.net/schemas/relationship/enemyOf>
<http://example.org/#green-goblin> <http://example.org/graphs/spiderman> .

RDF – TTL

- Based on N-Triples
- Name Space Prefixes
- abbreviations

@prefix ex: <http://example.org/#> .
@prefix per: <http://www.perceive.net/schemas/relationship/> .

ex:spiderman per:enemyOf ex:green-goblin .

RDF - Data model

([Subject] [Predicate] [Object]) :

- [Subject]: resource, blank node
  - <http://example.org/#spiderman>, _:x
- [Predicate]: resource
  - <http://www.perceive.net/schemas/relationship/enemyOf>
- [Object]: resource, blank node, literal
  - <http://example.org/#green-goblin>, _:x, 25

RDF – Data types

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix s: <http://schema.org/> .
@prefix won: <https://w3id.org/won/core#> .

won:Atom1 rdf:type won:Atom;
        s:title "Test"^^s:text;
        s:dateCreated "2019-10-16T13:37:12.886Z"^^s:dateTime  .
```

rdf:type ⇔ a (is a):

```
won:Atom1 a won:Atom .
```

RDF – Blank Nodes

```
@prefix ex: <http://example.org/data#> .
<http://example.org/web-data> ex:title "Web Data" ;
                    ex:professor [    ex:fullName "Alice Carol" ;
                                      ex:homePage <http://example.net/alice-carol>  ] .
```

RDF – Blank Nodes

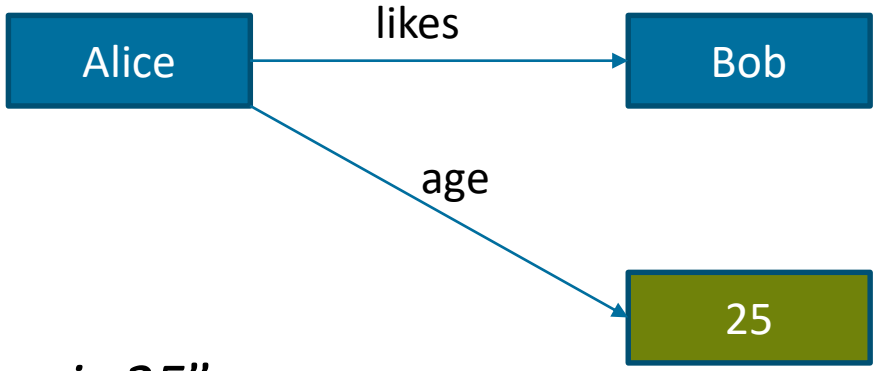
```
@prefix ex: <http://example.org/data#> .

<http://example.org/web-data> ex:title "Web Data" ;
            ex:professor _:entity ;

_:entity ex:fullName "Alice Carol" ;
            ex:homePage <http://example.net/alice-carol> .
```

RDF Example: Triples

- *"Alice likes Bob"*



- *"Alice is 25"*
- Predicate = [age]

25 = Literal: Item of data, not resource
Typically not Subject

RDF Example: Namespaces

"Alice likes Bob"

<http://tempuri.com/people/Alice> <http://example.com/relationships/likes> <http://tempuri.com/people/Bob> .

@prefix p: <http://tempuri.com/people/> .

@prefix r: <http://example.com/relationships/> .

p:Alice r:like p:Bob .

RDF Example: Multiple Statements

```
@prefix p: <http://tempuri.com/people/> .
@prefix r: <http://example.com/relationships/> .

p:Alice rdf:type p:Person;
    r:like p:Bob;
    r:givenName "Alice";
    r:familyName "Brown";
    r:age 25 .


p:Alice r:like p:Bob, p:Charlie, p:David, p:Eddie, p:Fred .
```

RDF Example: SPARQL

```
SELECT ?person
WHERE {
    p:Alice r:like ?person .
}
```

> Bob

“All persons Alice likes”

```
SELECT ?person
WHERE {
    _:liker r:like ?person ;
            r:age 25 .
}
```

> Bob

“Persons liked by 25 years old persons”

RDF Example: SPARQL

```
p:Alice r:like p:Bob, p:Charlie, p:David;
    r:givenName "Alice";
    r:familyName "Brown";
    r:age 25 .
p:Bob r:like p:Alice, p:Charlie, p:Eddie;
    r:givenName "Bob";
    r:familyName "Carter";
    r:age 31 .
p:Charlie r:like p:Alice, p:Bob, p:Eddie, p:Fred;
    r:givenName "Charles";
    r:familyName "David";
    r:age 27 .
p:David r:like p:Alice, p:Bob, p:Eddie, p:Fred;
    r:givenName "David";
    r:familyName "Eddings";
    r:age 28 .
p:Eddie r:like p:Alice, p:Bob, p:Eddie, p:Fred;
    r:givenName "Edward";
    r:familyName "Foster";
    r:age 28 .
p:Fred r:like p:Alice, p:Bob, p:Eddie, p:Fred;
    r:givenName "Frederick";
    r:familyName "Groves";
    r:age 25 .
```

```
SELECT DISTINCT ?person
WHERE {
   ?person r:age 27 .
   ?x r:like ?person;
      r:age 25.
}

>
```

RDF Example: SPARQL

```
p:Alice r:like p:Bob, p:Charlie, p:David;
    r:givenName "Alice";
    r:familyName "Brown";
    r:age 25 .
p:Bob r:like p:Alice, p:Charlie, p:Eddie;
    r:givenName "Bob";
    r:familyName "Carter";
    r:age 31 .
p:Charlie r:like p:Alice, p:Bob, p:Eddie, p:Fred;
    r:givenName "Charles";
    r:familyName "David";
    r:age 27 .
p:David r:like p:Alice, p:Bob, p:Eddie, p:Fred;
    r:givenName "David";
    r:familyName "Eddings";
    r:age 28 .
p:Eddie r:like p:Alice, p:Bob, p:Eddie, p:Fred;
    r:givenName "Edward";
    r:familyName "Foster";
    r:age 28 .
p:Fred r:like p:Alice, p:Bob, p:Eddie, p:Fred;
    r:givenName "Frederick";
    r:familyName "Groves";
    r:age 25 .
```

```
SELECT DISTINCT ?person
WHERE {
   ?person r:age 27 .
   ?x r:like ?person;
      r:age 25.
}
```

> p:Charlie

“Persons who are 27 and liked by 25 years old persons”.

RDF Example: Blazegraph

https://blazegraph.com/

1. Download
2. Start blazegraph.jar (java –jar blazegraph.jar)
3. Check ip/port combination and access via webbrowser

RDF Example: 1) TTL Input

http://ip:port/blazegraph/#update

Type: RDF Data Format: Turtle

```
@prefix p: <http://tempuri.com/people/> .
@prefix r: <http://example.com/relationships/> .

p:Alice r:like p:Bob;
   r:givenName "Alice";
   r:familyName "Brown";
   r:age 25 .
p:Bob r:like p:Alice, p:Charlie;
   r:givenName "Bob";
   r:familyName "Carter";
   r:age 31 .
p:Charlie r:like p:Alice;
   r:givenName "Charles";
   r:familyName "David";
   r:age 27 .
```

RDF Example: 2) SPARQL Query

http://ip:port/blazegraph/#query

```
PREFIX p: <http://tempuri.com/people/>
PREFIX r: <http://example.com/relationships/>

SELECT ?person
WHERE {
   _:liker r:like ?person;
        r:givenName "Bob".
   ?person r:age 25 .
}
```

> p:Alice

DBpedia

- https://wiki.dbpedia.org/
- Open Knowledge Graph (OKG)
- “4.58 million things, out of which 4.22 million are classified in a consistent ontology”

DBpedia - Vienna

- http://dbpedia.org/data/Vienna.ttl
- 7728 RDF triples

DBpedia – 3) SPARQL

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>

SELECT DISTINCT ?person
WHERE {
        ?person dbo:birthPlace dbr:Vienna .
}
```

DBpedia – 4) SPARQL: Multiple Statements

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>

SELECT DISTINCT ?person
WHERE {
    ?person dbo:birthPlace dbr:Vienna .
    ?person dbo:deathPlace dbr:Vienna .
}
```

DBpedia – 5) SPARQL: Projection

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT DISTINCT ?name ?person
WHERE {
    ?person dbo:birthPlace dbr:Vienna .
    ?person dbo:deathPlace dbr:Vienna .
    ?person foaf:name ?name .
}
```

DBpedia – 6) SPARQL: Projection

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT distinct ?name ?birth ?p
WHERE {
      ?p dbo:birthPlace dbr:Vienna .
      ?p dbo:deathPlace dbr:Vienna .
      ?p foaf:name ?name .
      ?p dbo:birthDate ?birth
}
```

DBpedia – 7) SPARQL: Order

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT distinct ?name ?birth ?p
WHERE {
      ?p dbo:birthPlace dbr:Vienna .
      ?p dbo:deathPlace dbr:Vienna .
      ?p foaf:name ?name .
      ?p dbo:birthDate ?birth
}  ORDER BY ?name
```

DBpedia – 8) SPARQL: Filter

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT distinct ?name ?birth ?p
WHERE {
      ?p dbo:birthPlace dbr:Vienna .
      ?p foaf:name ?name .
      ?p dbo:birthDate ?birth .
       FILTER (?birth < “2000-01-01”^^xsd:date) .
}  ORDER BY ?name
```

DBpedia – 9) SPARQL: Count

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT (COUNT(?p) AS ?personen)
WHERE {
      ?p dbo:birthPlace dbr:Vienna .
      ?p foaf:name ?name .
      ?p dbo:birthDate ?birth .
      FILTER (?birth < "2000-01-01"^^xsd:date) .
}
```

DBpedia: Hands On

Amount of People who were born in Berlin before 1900

DBpedia: 10) Hands On

Amount of People who were born in Berlin before 1900

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX : <http://dbpedia.org/resource/>


SELECT DISTINCT (COUNT (?person) AS ?amount) WHERE {
        ?person dbo:birthPlace :Berlin .
        ?person dbo:birthDate ?birth .
        ?person foaf:name ?name .
        ?person dbo:deathDate ?death .
        FILTER (?birth < "1900-01-01"^^xsd:date) .
}
```

DBpedia: 11) Hands On

- Why is the outcome of this query “438336346” ?

```
SELECT (COUNT(?s) AS ?triples) WHERE { ?s ?p ?o }
```

RDF Implementation

- JSON-LD
  - "JSON-based format to serialize Linked Data" [12]
  - https://json-ld.org/playground/
- Jena
  - "A free and open source Java framework for building Semantic Web and Linked Data applications."
  - https://jena.apache.org/

Where to go next?

- Schema.org
  - https://schema.org/
- Linked Open Vocabularies
  - https://lov.linkeddata.es/dataset/lov

GitHub

- https://github.com/WoN-Hackathon-2019
- Create 1 Team for every Hackathon team (https://github.com/orgs/WoN-Hackathon-2019/teams)
- Add a repository to your team
- Clone the bot-skeleton: https://github.com/researchstudio-sat/bot-skeleton
- Link this repository with a short description of your ideas in the “Information” repository in “WoN-Hackathon-2019”
- Commit your changes, they will be reviewed and merged into the main branch by your colleagues

Maven

- Using maven to build the bot application from the command line
  1. cd bot-skeleton
  2. Define the Node URI your bot connects to:
     Export WON_NODE_URI="https://hackathonnode.matchat.org/won"
  3. Compile and Build: mvn clean package
  4. Start the bot: java -jar target/bot.jar

Resources:

- https://github.com/researchstudio-sat/webofneeds
- https://researchstudio-sat.github.io/webofneeds/ontologies/matching/
- https://json-ld.org/playground/

References

- https://de.wikipedia.org/wiki/Semantic_Web [1]
- Andrew. „Knowledge Graphs 101“. *Industrial Inference* (blog), 15. Oktober 2019. https://aabs.wordpress.com/2019/09/12/knowledge-graphs-101/. [2]
- Dean Allemang and James Hendler. 2011. Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL (2 ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. [3]
- Daconta, Michael & Obrst, Leo & Smith, Kevin. 2003. The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management. [4]
- Florian Kleedorfer, Christina Maria Busch. 2013. Beyond Data: Building a Web of Needs
- https://github.com/researchstudio-sat/webofneeds [5]
- https://wiki.dbpedia.org/ [7]
- http://sites.linkeddata.center/help/devop/examples/sparql-examples [8]
- https://www.w3.org/TR/n-quads/ [9]
- https://www.w3.org/TR/n-triples/ [10]
- https://www.ietf.org/rfc/rfc3987.txt [11]
- https://www.w3.org/TR/json-ld11/ [12]