Anwendungen der KI

– Sommersemester 2018 –

Kapitel 11: Linked Open Data

Prof. Adrian Ulges
B.Sc. {Angewandte, Medien-, Wirtschafts-}informatik
Fachbereiche DCSM
Hochschule RheinMain

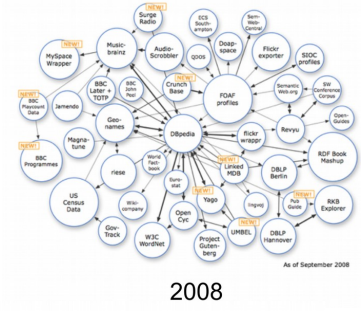
Linked Open Data

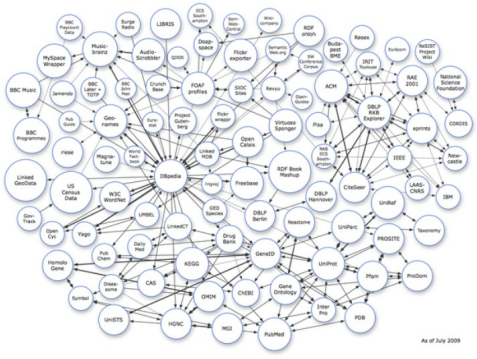
- Wir haben letzte Woche bereits einen **Wissensgraph** kennen gelernt: **WordNet**.
- WordNet war eine Art **Wörterbuch** mit hochqualitativen Information für einen Standard-Vokabular.
- WordNet war **klein** → Kaum **Spezialinformation** für Named Entities.
- Um spezifische Fragen zu beantworten, müssen wir **größer denken...**

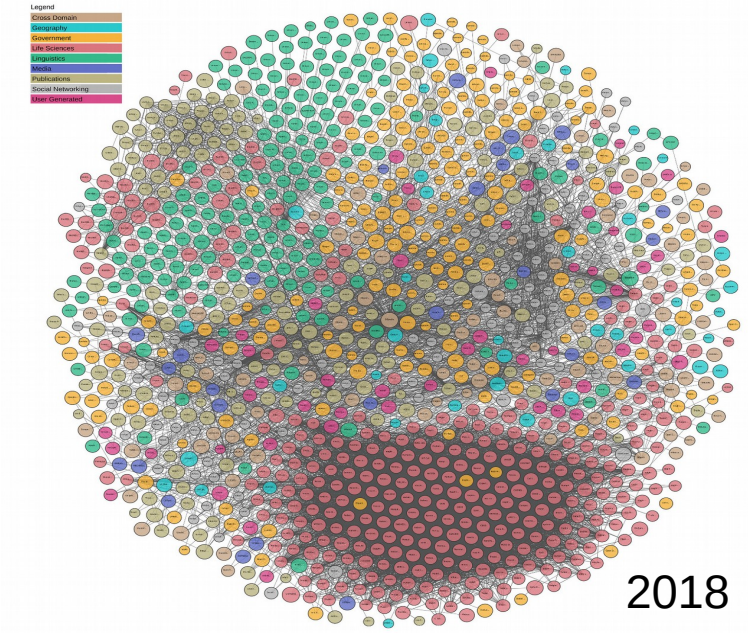
Linked Open Data

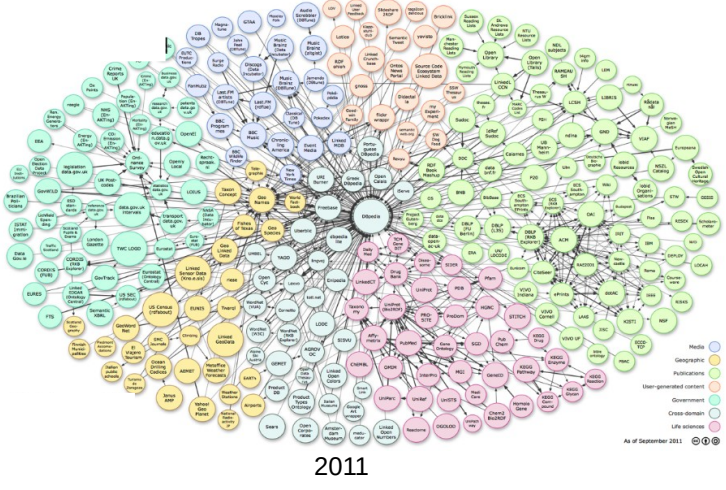
Linked Open Data

In den letzten 15 Jahren ist ein Netz von großen Wissensgraphen entstanden, die sogenannte **Linked Open Data (LOD) Cloud**[1]









[1] https://lod-cloud.net/

Das “Semantic Web”

Woher kommt LOD? → Vision des **Semantic Web**

Das WWW

- Seit den frühen 1990ern, exponentielles Wachstum von **User-generated Content**.
- Zeitalter der **Suchmaschinen**: Computer “rechnen” nicht mehr, sondern sind für ihre User hauptsächlich Portale zu Information.
  - Chats mit Freunden
  - Produkte online recherchieren und kaufen
  - Videos schauen
  - ...
- Aktuell ist Content nur **für Menschen** verständlich, aber **nicht/schwer maschinen-interpretierbar**.

Sind Suchmaschinen “gut genug”?

- Manche Anwendungen erfordern 100% Precision+Recall!
- Manche Anwendungen erfordern **detailiertes** Verständnis

Neue Anwendungsgebiete von KI

- Detailierte Auskünfte erteilen *(Automatische Sachbearbeiter)*
- Personalisierung *(Persönliche Digitale Assistenten)*
- Verhandeln und Optimieren *(Autonome Services)*
- Komplexere Fragestellungen *(Trip nach Rom planen)*

Vision: Das Semantic Web

- **Ziel**: **maschinen-interpretierbarer Content**!
- Vermeide Mehrdeutigkeiten natürlicher Sprache

```
'I am a professor of computer science.' vs.
'I consider myself an AI lecturer.'
```

- Nehme gegebene harte Fakten an, Fokus auf maschinelle Interpretierbarkeit + Inferenz.

Semantic Web: History

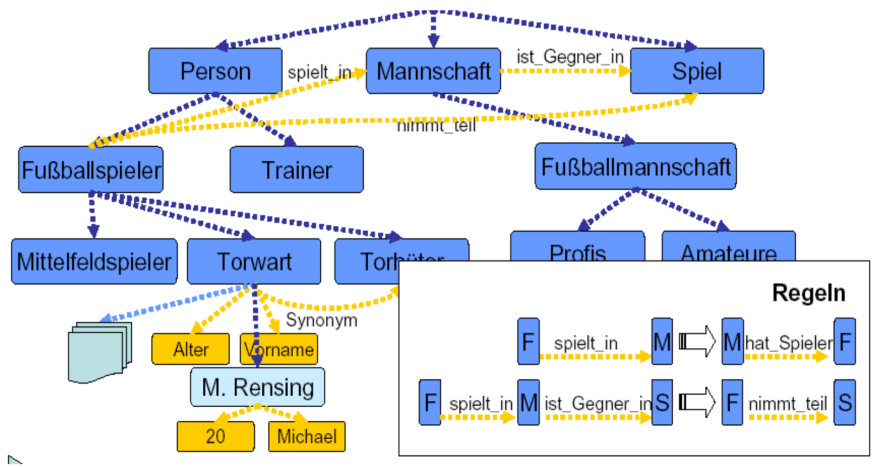
Der bekannteste Verfechter des Semantic Web ist **Tim Berners-Lee**, der auch das **WWW** in den 1980ern mit entwickelte.

> *"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."*
>
> (Tim Berners Lee et al., "The Semantic Web" (Scientific American, 2001))

> *"The Semantic Web will not be a new global information highway parallel to the existing World Wide Web; instead it will gradually evolve out of the existing Web."*
>
> (Antoniou, van Harmelen, "A Semantic Web Primer" (2004))

Grundlage

- Eine gemeinsame Sprache (Symbole, Konzepte, Relationen)
- Begriff der **Ontologie** *(siehe letzte Vorlesung)*

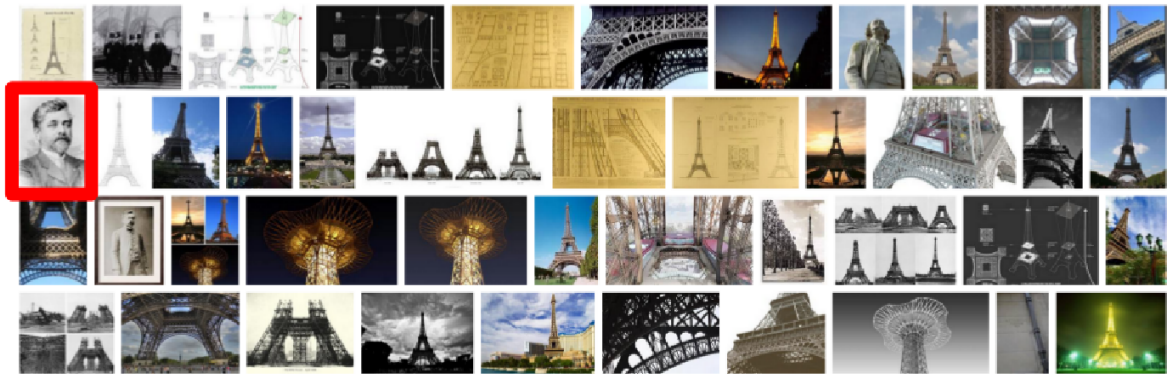


Semantic Web: Beispiel

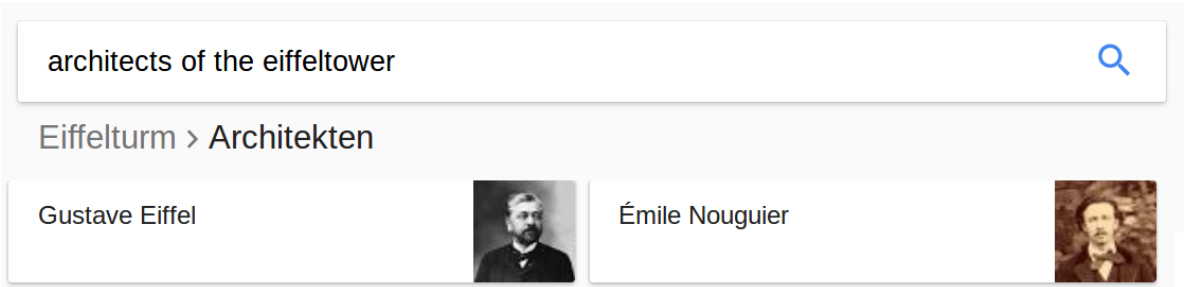
Suche bei einer Web-Suchmaschine Bilder von Gustave Eiffel *(dem Erbauer des Eiffelturms)*

> `'architects of the eiffeltower'`

Ergebnis 2013



Ergebnis 2018



Semantic Web: Beispiel

Was ist hier passiert?

- Die Suchmaschine besitzt eine **Ontologie**, u.a. mit Entitäten `#EiffelTower` und `#GustaveEiffel`, sowie die Relation `HasArchitect(#Eiffeltower,#GustaveEiffel)`.
- Die Instanz `#Eiffeltower` und die Relation `HasArchitect()` wurden in der **Anfrage** identifiziert.
- Die **Zieldokumente / Bilder** besitzen **Metadaten**, z.B. semantische oder Freitext-Annotationen
- Durch einen **formalen Query** wird die Instanz `#GustaveEiffel` gefunden, und hierüber die Zieldokumente/Bilder.

Vorteile dieses "semantischen" Vorgehens?

- Höhere Genauigkeit
- Erklärbarkeit, Transparenz für den Nutzer

Semantische Methoden: Anwendungsfelder

Digitale Personal Assistenten

- Assistenten vereinbaren Arzttermine, planen die Anfahrt, verlegen hierfür weitere Termine, etc.

Handel

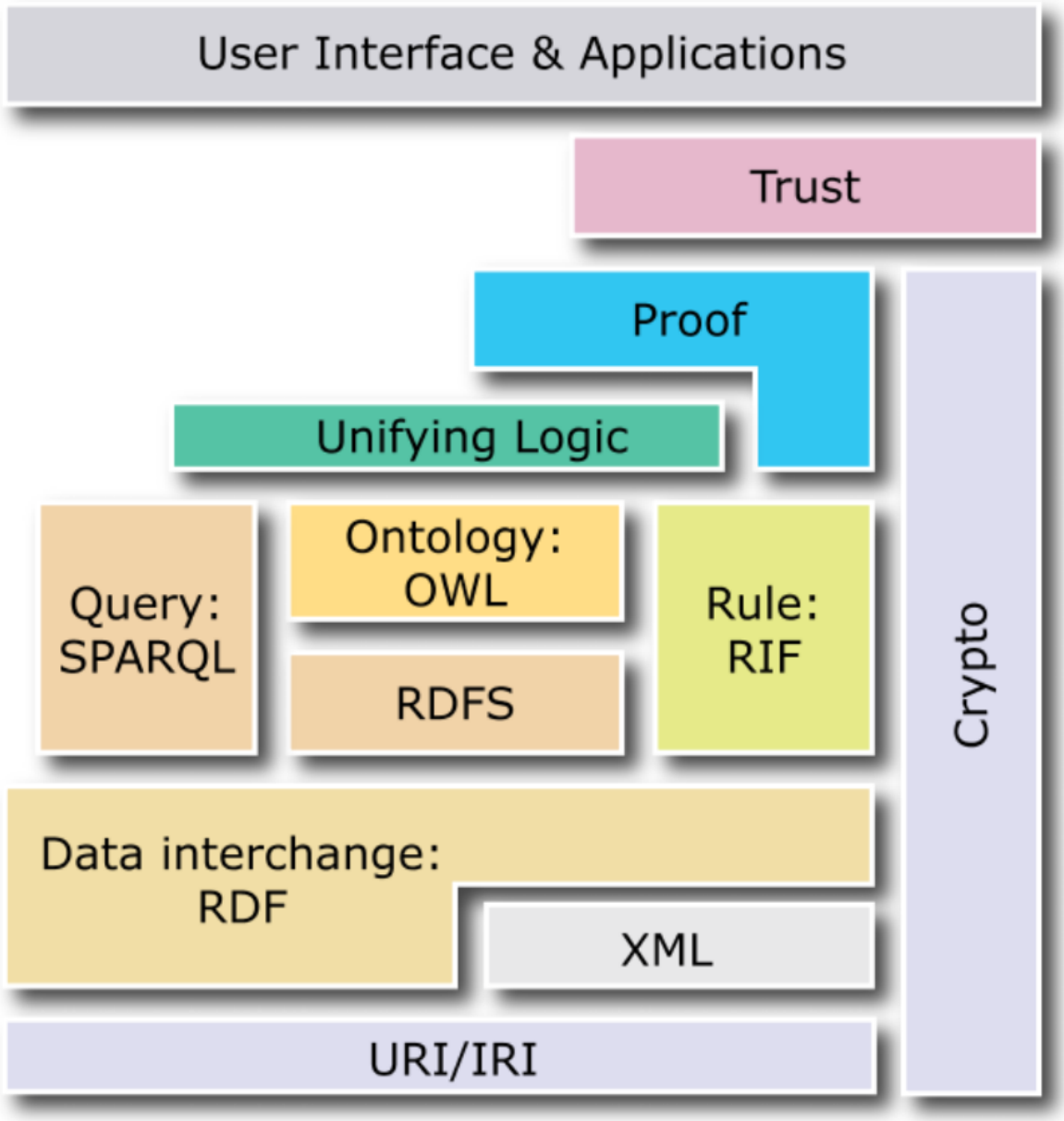
- Informationen über **Produkte**: "Bis zu welcher Temperatur kann ich Schaltung X betreiben?"
- Insbesondere im **B2B-Bereich**: Agenten führen automatisch Preisvergleiche durch, wählen Produkte aus und konfigurieren sie (Was sind die Anforderungen an den Stromkreis? Wie lange ist die Garantie?)
- Agenten interpretieren formal spezifizierte **Verträge**. Beispiel Krankenversicherung: Was wird was bezahlt? Worin unterscheiden sich Tarif A und B?

Dokumenten- und Wissensmanagement

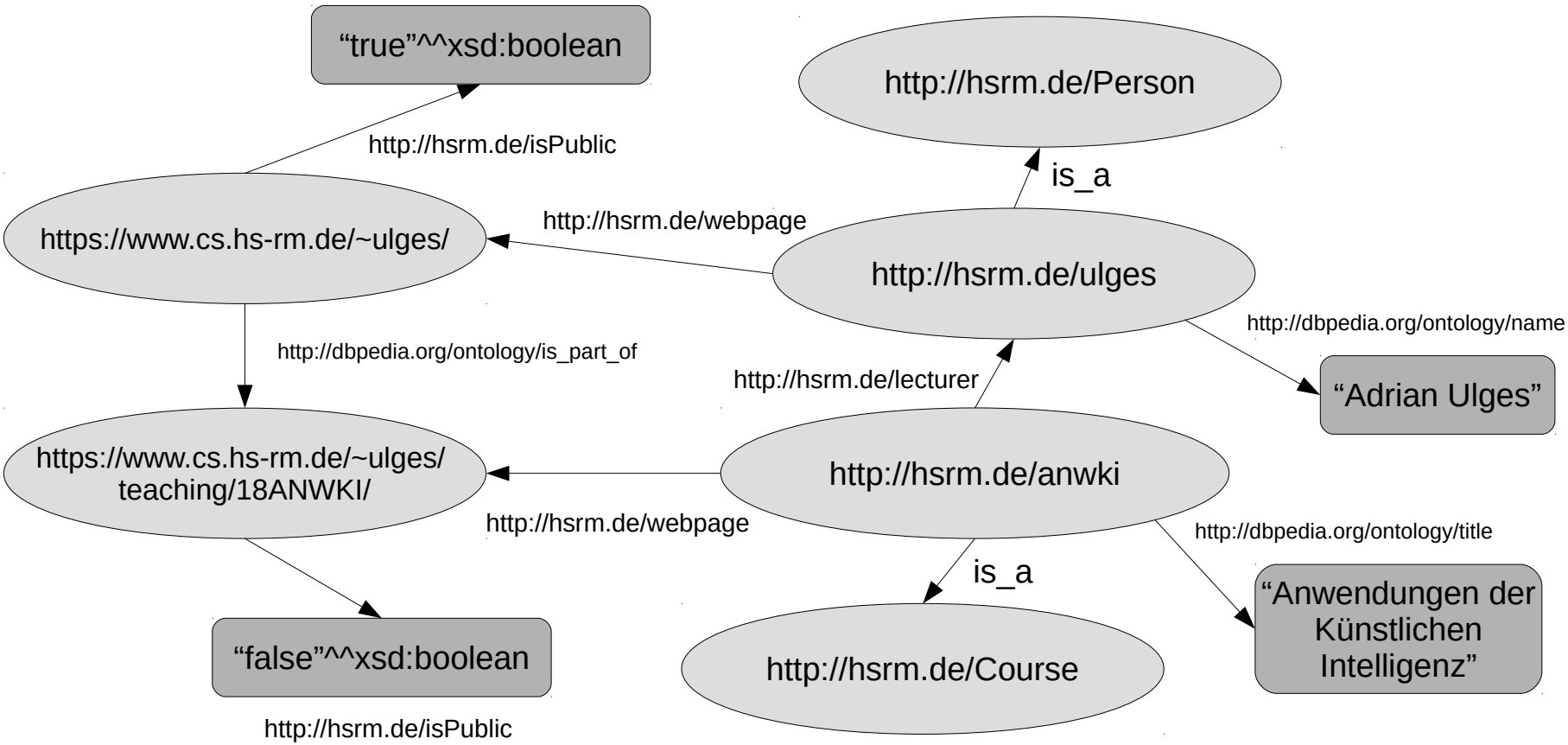
- Wer im Betrieb kennt sich mit welchen Konzepten aus?
  → Reuse von Lösungen, Expertisenmanagement
- Enthalten Dokumente Widersprüche / unterschiedliche Fakten?

Outline

Der W3C Semantic Web Stack



Grundlagen: RDF



Grundlagen: RDF

RDF = Resource Description Framework (vom W3C)

- **Zweck**: Beschreibung von Ressourcen in relationaler Weise
- **Bestandteile**: (1) RDF Data Model *(resource has property with value)*. (2) RDF Syntax (Repräsentation eines RDF Data Models, z.B. in XML oder Turtle).

Ressourcen

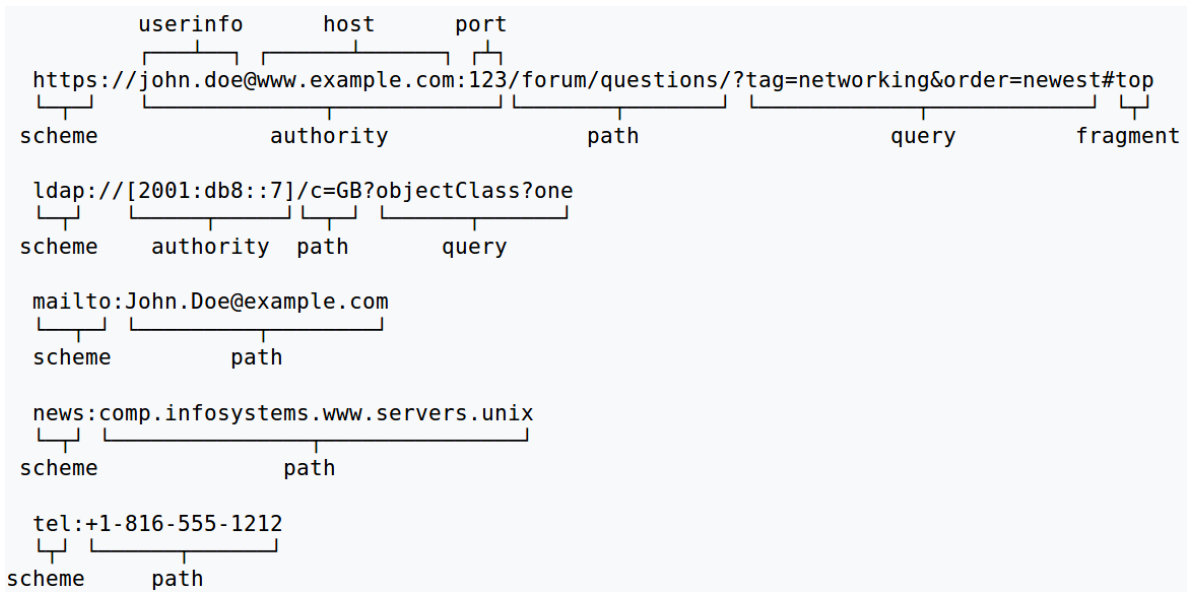
- Eine **Ressource** (oder *Knoten*) ist ein Informationsobjekt das mit Metadaten beschrieben wird.
- Sie wird über einen **Universal Resource Identifier** (**URI**) identifiziert.
- **Leere Knoten** sind erlaubt. Sie erlauben Aussagen über eine Entität ohne diese konkret zu benennen.

Grundlagen: RDF

URIs

```
URI = scheme:[//authority]path[?query][#fragment]
authority = [userinfo@]host[:port]
```

- URIs dienen dazu, Resourcen **eindeutig zu benamen**.
- URIs können sich auf **Webdokumente** beziehen *(d.h., auch URLs sind URIs!)*, müssen aber nicht!



Grundlagen: RDF

Properties

- Eine Ressource wird durch **Property-Wert-Paare** beschrieben
- Die Property ist eine **binäre Relation** zwischen der Ressource und dem Wert.

Literale

- Literale dienen zur Repräsentation von **Datenwerten**
- Sie werden als **Zeichenketten** dargestellt (z.B. ‘‘`42`’’), die zusätzlich **getypt** sein können (‘‘`42`’’`^^integer`).

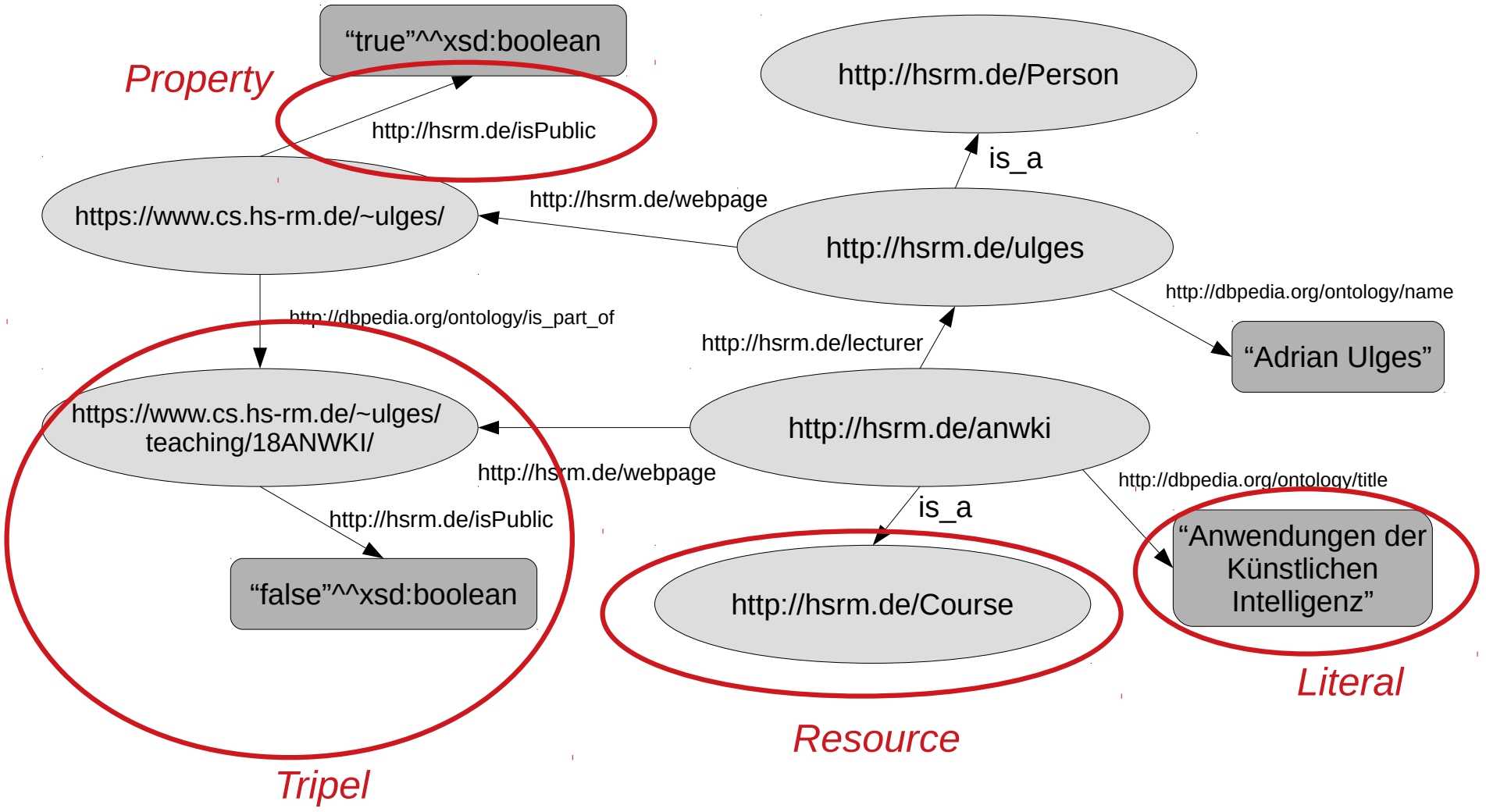
Statements

- Ein Statement ist ein **Triple**:

  (Subjekt, Prädikat, Objekt)

- Das **Subjekt** ist eine URI oder ein leerer Knoten
- Das **Prädikat** *(oder Property)* ist eine URI
- Das **Objekt** ist eine URI / ein leerer Knoten / ein **Literal**.

Grundlagen: RDF



Die Turtle-Syntax

- Um Wissensgraphen zu **serialisieren**, **auszutauschen** und zu **verarbeiten** ist eine **Standardsyntax** erforderlich.
- Häufig wird hier **XML** verwendet, eine kompakte Alternative bietet **Turtle** *("Terse RDF Triple Language")*.
- Turtle – bzw. eine Variante – wird auch von der **Anfragesprache SPARQL** *(später)* verwendet.

Turtle: Grundlagen

- Tripel werden **nacheinander aufgelistet** (Trennung mit ".").
- URIs in **spitzen Klammern**, Literale in Anführungszeichen.
- Leerzeichen+Umbrüche außerhalb Bezeichnern ignorieren.

```
<http://dbpedia.org/resource/Germany>
        <http://dbpedia.org/ontology/areaTotal> ‘‘357168.0’’ .
<http://dbpedia.org/resource/Germany>
        <owl:sameAs> <http://www.wikidata.org/wiki/Q183> .
<http://www.wikidata.org/wiki/Q183>
        <https://www.wikidata.org/wiki/Property:P38>
        <https://www.wikidata.org/wiki/Q4916> .
        (im Klartext: <Germany> <currency> <Euro> .)
```

Die Turtle-Syntax (cont'd)

Wir können Abkürzungen für Präfixe festlegen

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

dbr:Germany dbo:areaTotal ‘‘357168.0’’ .
```

Tripel mit gleichem Subjekt zusammenfassen: ';'

```
dbr:Germany dbo:wineRegion dbr:Riesling .
dbr:Germany dbo:assembly dbr:BMW_i8 .
entspricht

dbr:Germany dbo:wineRegion dbr:Riesling ; dbo:assembly dbr:BMW_i8 .
```

Tripel mit gleichem Subjekt+Prädikat zusammenfassen: ','

```
dbr:Germany dbo:wineRegion dbr:Riesling, dbr:Scheurebe, dbr:Dornfelder ;
        dbo:assembly dbr:BMW_i8 .
```

Turtle: Getypte Literale

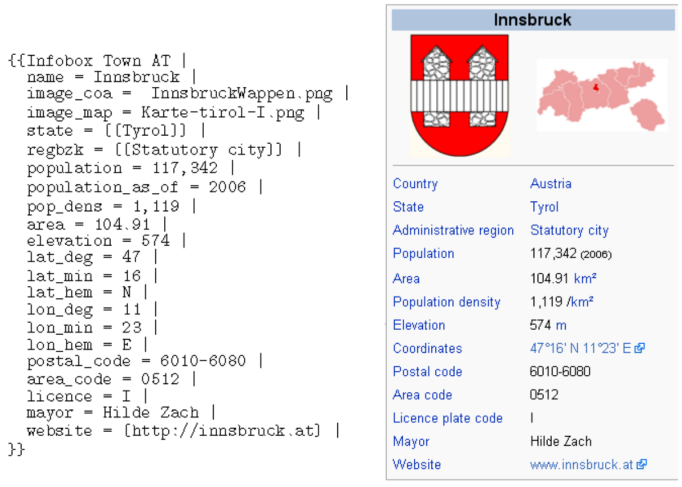
Literale werden generell in Anführungsstrichen angegeben.
Sie können **getypt** sein!

Literale

- Strings (Typ `xsd:string`): `‘‘plain string’’`
- Strings können in verschiedenen **Sprachen** angegeben sein: `‘‘string with language’’@en`
- Integers (Typ tt xsd:integer): `‘‘13.4’’^^xsd:integer`
- Boolean: `‘‘true’’^^xsd:boolean`

Outline

Das DBPedia-Projekt



```
{{Infobox Town AT |
  name = Innsbruck |
  image_coa =  InnsbruckWappen.png |
  image_map = Karte-tirol-I.png |
  state = [[Tyrol]] |
  regbzk = [[Statutory city]] |
  population = 117,342 |
  population_as_of = 2006 |
  pop_dens = 1,119 |
  area = 104.91 |
  elevation = 574 |
  lat_deg = 47 |
  lat_min = 16 |
  lat_hem = N |
  lon_deg = 11 |
  lon_min = 23 |
  lon_hem = E |
  postal_code = 6010-6080 |
  area_code = 0512 |
  licence = I |
  mayor = Hilde Zach |
  website = [http://innsbruck.at] |
}}
```


Bsp.: Deutsche Musiker die in Berlin geboren wurden

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

SELECT ?name ?birth ?description ?person WHERE {
    ?person dbo:birthPlace :Berlin .
    ?person <http://purl.org/dc/terms/subject>
<http://dbpedia.org/resource/Category:German_musicians> .
    ?person dbo:birthDate ?birth .
    ?person foaf:name ?name .
    ?person rdfs:comment ?description .
    FILTER (LANG(?description) = 'en') .
}
ORDER BY ?name
```

- Initiative der Uni Leipzig, der Freien Universität Berlin und der Firma OpenLink Software
- **Regelbasierte Extraktion** formaler semantischer Daten aus **Wikipedia-Infoboxen**
- **Lizenz**: Creative Commons 3.0 and GNU
- Deckt **viele Domänen** ab → Mächtige Breite an Beispiel-Fragen: “Nenne Städte mit mehr als 100,000 Einwohnern in New Jersey”, “Nenne italienische Musiker des 18. Jahrhunderts”.

Das DBPedia-Projekt (cont'd)

Umfang[2]

- 125 Sprachen, **38.3 Mio. Entitäten** und **3 Mrd. Tripel**.
- **Englisch**: **4.5 Mio. Entitäten** und **580 Mio. Tripel**.

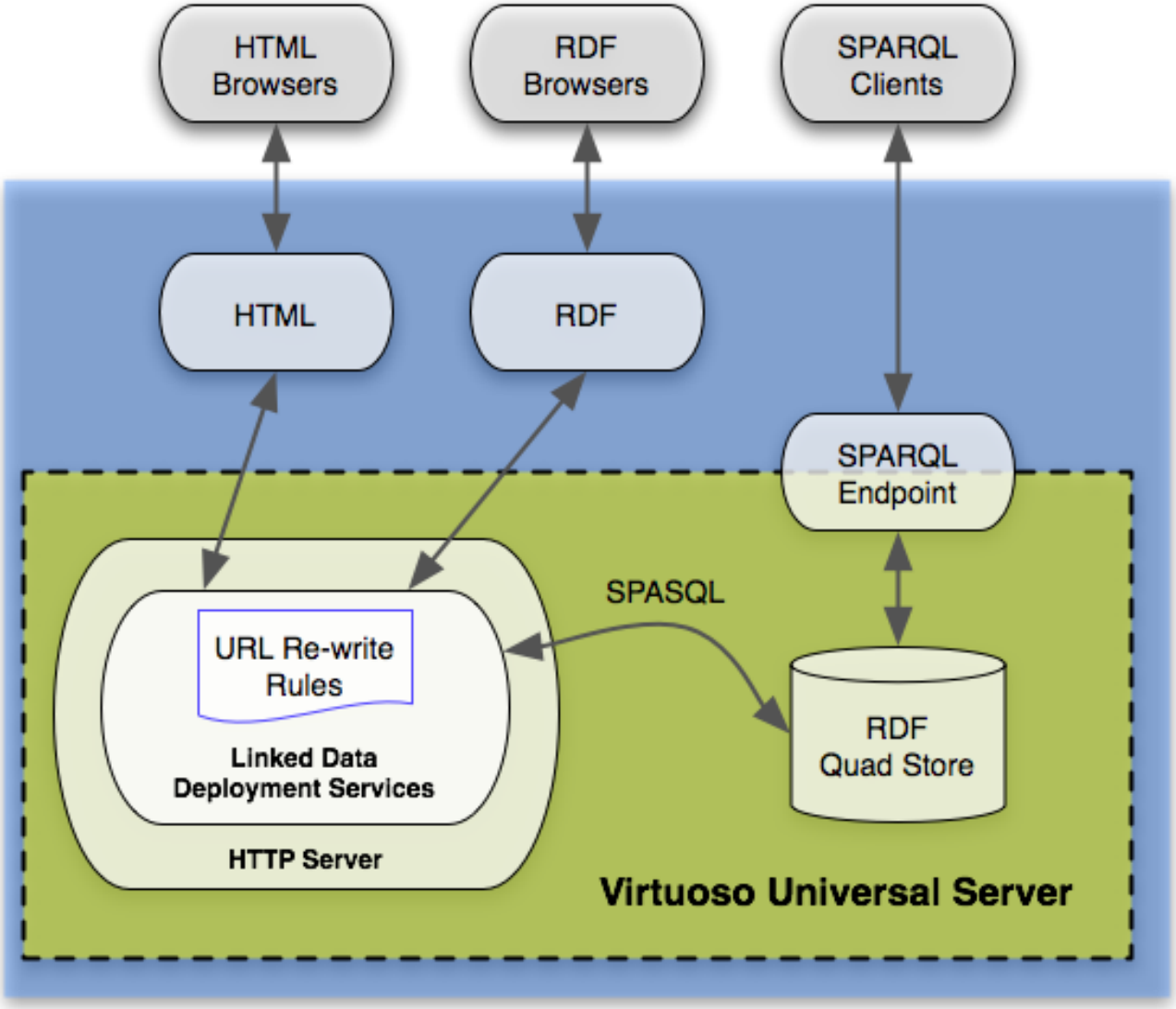
# Personen	1.450.000	# Orte	735.000
# Filme,Songs,...	411.000	#Organisationen	241.000
# Spezies	251.000	# Krankheiten	6.000

Vernetzung

- DBPedia ist eng **vernetzt** mit vielen anderen Ontologien *(ca. 50 mio. Tripeln zu anderen Ontologien)*, und kann deshalb als eine Art **zentraler Hub** der LOD-Cloud verstanden werden.

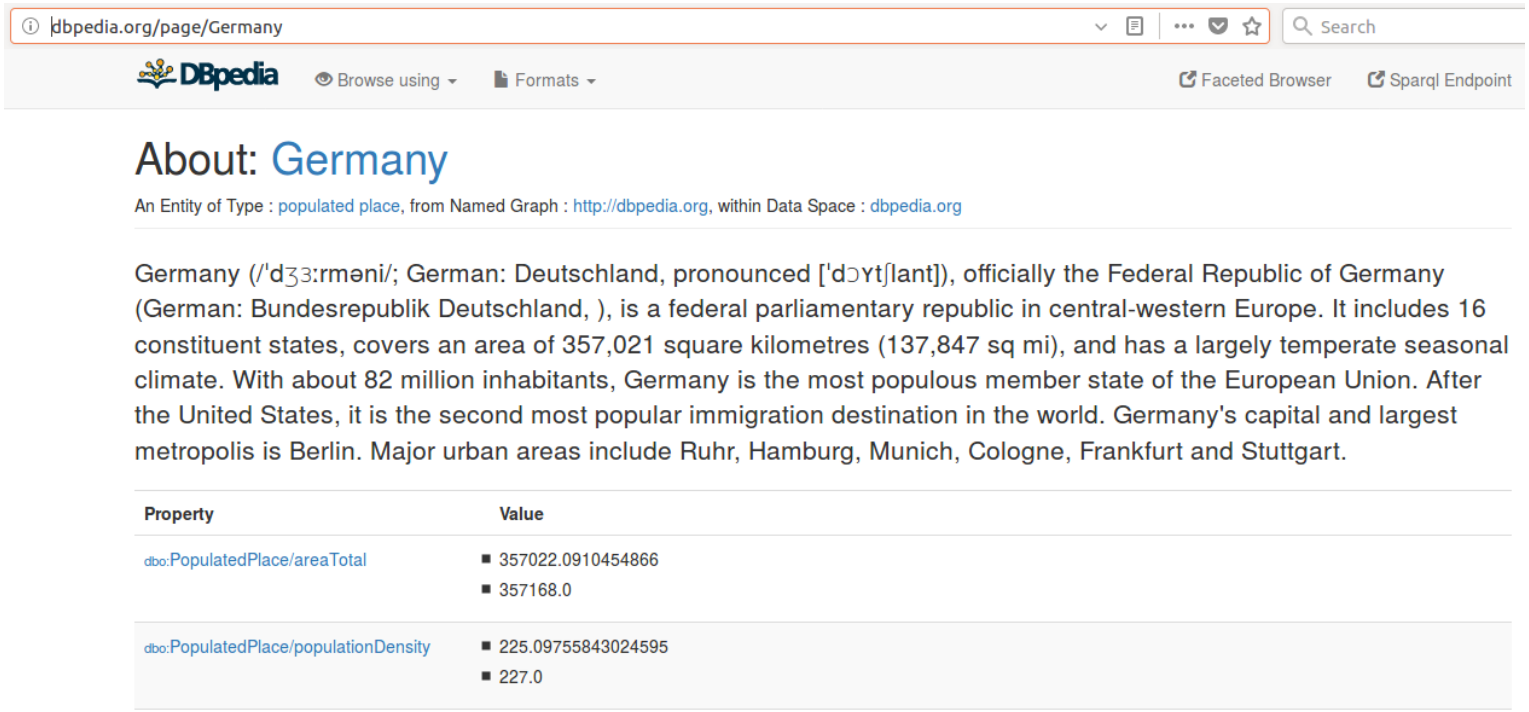
[2] https://wiki.dbpedia.org/about

Zugang zu DBPedia



Zugang zu DBPedia

1. Zugang per Browser



2. Zugang per SPARQL

- Öffentlich zugänglicher Endpoint:
  `https://dbpedia.org/sparql`

Outline

SPARQL *(sprich engl. "sparkle")* [3]

- ... steht für **SPARQL Protocol and RDF Query Language** *(rekursive Namensdefinition)*.
- ... ist eine **Anfragesprache** zur Abfrage von Entitäten aus **RDF-Dokumenten**.
- seit 2013 W3C Recommendation.

Die SPARQL-Spezifikation enthält...

1. Die Anfragesprache *(Thema dieser Vorlesung)*
2. Das Ergebnisformat *(in XML)*
3. Das Protokoll *(Übermittlung von Anfrage und Ergebnissen)*

[3] https://www.w3.org/TR/sparql11-overview/

SPARQL: Einfache Beispiel-Anfrage

```
SELECT ?title ?author
WHERE {
 ?buch ?r <http://sp.com/Verl> .
 ?buch <http://example/org/titel> ?titel .
 ?buch <http://example/org/autor> ?author .
}
```

- **WHERE**: **Anfragemuster** von **Tripeln** in Turtle-Syntax
  (Lösungen müssen auf **alle** *Tripel passen)*
- **Variablen** `?title`, `?buch`, `?r`, `?author`
- Optional Schlüsselwort **PREFIX**: Definiert Abkürzungen, die im Query verwendet werden können
  (`<http://example/org/titel>` → `ex:titel`)

```
PREFIX ex: <http://example/org>
PREFIX mylibrary: <http://mylibrary.org>
SELECT ?title ?author
WHERE {
    ?buch ex:verleger <http://springer.com/Verlag> .
    ?buch mylibrary:titel ?titel .
    ?buch ex:autor ?author .
}
```

SPARQL: Einfache Beispiel-Anfrage

Ergebnis der Anfrage: **Tabelle** mit einer Zeile je Ergebnis

title	**author**
Pattern Recognition and Machine Learning	Chris Bishop
Understanding Cryptography	Christof Paar
Computer Vision	Richard Szeliski
Real-Time C++	Christopher Kormanyos

Syntax

- TURTLE-Abkürzungen mit ';' sind zulässig:

```
?p ?rel ?q . ?p ?rel2 ?q2
```

entspricht

```
?p ?rel ?q ; ?rel2 ?q2
```

- Statt `?variable` können wir auch `$variable` schreiben.

SPARQL und DBPedia

```
1 PREFIX dbo: <http://dbpedia.org/ontology/>
2 PREFIX dbr: <http://dbpedia.org/resource/>
3 SELECT ?city
4 {
5   ?city a dbo:City .
6   ?city dbo:country dbr:Germany
7 }
```

Abkürzungen

- `<http://dbpedia.org/ontology>` (`dbo:`)
  Klassen, **Relationen** (TBox): `dbo:City`, `dbo:country` ...
- `<http://dbpedia.org/property>` (`dbp:`)
  Weitere Properties: `dbp:north` ...
- `<http://dbpedia.org/resource>` (`dbr:`)
  Konkrete Dinge (ABox): `dbr:India` ...

city
http://dbpedia.org/resource/Haaren_(Aachen)
http://dbpedia.org/resource/Albstadt
http://dbpedia.org/resource/Mücheln
http://dbpedia.org/resource/Eppelheim

SPARQL aus Python

```
from SPARQLWrapper import SPARQLWrapper, JSON

sparql = SPARQLWrapper("http://dbpedia.org/sparql")
sparql.setQuery("""
    PREFIX dbo: <http://dbpedia.org/ontology/>
    PREFIX dbr: <http://dbpedia.org/resource/>
    SELECT ?city
    {
      ?city a dbo:City .
      ?city dbo:country dbr:Germany
    }
    """)

sparql.setReturnFormat(JSON)
results = sparql.query().convert()

for result in results['results']['bindings']:
    print(result['city']['value'])

# http://dbpedia.org/resource/Haaren_(Aachen)
# http://dbpedia.org/resource/Albstadt
# http://dbpedia.org/resource/Muecheln
# http://dbpedia.org/resource/Eppelheim
# ...
```

Do-SPARQL-Yourself

Finde Staatschefs in DBPedia, die nicht in dem von ihnen regierten Land geboren sind.

Filter in SPARQL

- Viele Anfragen sind auch mit komplexen Graph-Mustern alleine **nicht möglich**.
- **Lösungsmechanismus**: **Filter**.

Welche Personen sind zwischen 18 und 23 Jahren alt?
Der Nachname welcher Person enthält einen Bindestrich?
Welche Texte in Deutscher Sprache gibt es?

Filter-Beispiel

```
PREFIX ex: <http://example.org/>
SELECT ?buch WHERE {
      ?buch ex:verleger <http://springer.com/Verlag>
      ?buch ex:preis ?preis
      FILTER (?preis < 35) }
```

- Schlüsselwort FILTER, gefolgt von Bedingung in Klammern.
- FILTER und Tripel in beliebiger Reihenfolge.

Filter in SPARQL: Operatoren

- **Arithmetische Operatoren** (+, −, ∗, /)

```
FILTER( ?gewicht / (?groesse * ?groesse) >= 25 )
```

- **Vergleiche** (<, <=, =, >=, >, ! =)
  (Ordnung für String, DateTime, Bollean ist die jeweils *natürlichen Reihenfolge*, ansonsten nur =, ! = verfügbar).
- **Logische Verknüpfung** von Bedingungen (&&, ||, !)
- **Anmerkung**: Konjunktion entspricht der Angabe mehrerer Filter, Disjunktion Filtern in verschiedenen Mustern.

Beispiel: Vor 1953 geborene Deutsche Staatschefs

```
1 PREFIX dbo: <http://dbpedia.org/ontology/>
2 PREFIX dbr: <http://dbpedia.org/resource/>
3 SELECT ?person ?date
4 {
5   ?country a dbo:Country .
6   ?country dbo:leader ?person .
7   ?person dbo:birthDate ?date .
8   FILTER (?country = dbr:Germany && ?date < "1953-00-00T00
      :00:00+00:00"^^xsd:dateTime)
9 }
```

Filter in SPARQL: Operatoren (cont'd)

SPARQL unterstützt weitere **RDF-spezifischer Operatoren**

BOUND(A)	wahr gdw. A eine gebundene Variable ist
isURI(A)	wahr gdw. A eine URI ist
isLITERAL(A)	wahr gdw. A ein RDF-Literal ist
STR(A)	lexikalische Darstellung (xsd:string) von URIs+Literalen
LANG(A)	Sprachcode eines RDF-Literals (z.B. `''en''`) oder leerer String (falls kein Sprachcode vorhanden)
DATATYPE(A)	Datentyp-URI eines RDF-Literals, hilfsweise `xsd:string`
LANGMATCHES(A,B)	wahr gdw. die Sprachangabe von A auf das Muster B passt
REGEX(A,B)	wahr gdw. in der Zeichenkette A der reguläre Ausdruck B gefunden wird.

Beispiel

Deutschsprachige Buchrezensionen

```
PREFIX ex: <http://example.org/>
SELECT ?buch
WHERE {
   ?buch ex:rezension ?text .
   FILTER LANGMATCHES( LANG(?text), "de" )
}
```

SPARQL: Gruppen

Einfache Graph-Muster können durch { ... } **gruppiert** werden. Dies bewirkt erstmal *nichts*, sondern nur unter Verwendung zusätzlicher Konstrukte wie `UNION` oder `OPTIONAL`.

Beispiel

```
1 WHERE {
2   ?buch ex:verleger <http://springer.com/verlag> .
3   ?buch ex:titel ?titel .
4   ?buch ex:autor ?autor .
5 }
```

... ist äquivalent zu ...

```
1 WHERE {
2   { ?buch ex:verleger <http://springer.com/verlag> .
3   ?buch ex:titel ?titel . }
4   ?buch ex:autor ?autor .
5 }
```

SPARQL: Vereinigung

Das Schlüsselwort `UNION` erlaubt die Angabe **alternativer Teile** eines Musters.

Beispiel

- Finde Schauspieler **oder** Regisseure **oder** Producer von "Interstellar".

```
1 PREFIX interstellar:
2        <http://dbpedia.org/resource/Interstellar_(film)>
3 SELECT ?p
4 {
5     { interstellar: dbo:director ?p . } UNION
6     { interstellar: dbo:starring ?p . } UNION
7     { interstellar: dbo:producers ?p . }
8 }
```

SPARQL: Optionale Teil-Patterns

Das Schlüsselwort `OPTIONAL` erlaubt die Angabe **optionaler Teile** eines Musters.

Beispiel

Finde Staatschefs und – **soweit bekannt** – ihre Geburtsorte.

```
1 PREFIX dbo: <http://dbpedia.org/ontology/>
2 PREFIX dbr: <http://dbpedia.org/resource/>
3 SELECT ?person ?place
4 {
5   ?country a dbo:Country .
6   ?country dbo:leader ?person .
7   OPTIONAL { ?person dbo:birthPlace ?place}
8 }
```

person	place
http://dbpedia.org/resource/Alok_Sharma	http://dbpedia.org/resource/India
http://dbpedia.org/resource/Barack_Obama	http://dbpedia.org/resource/Hawaii
http://dbpedia.org/resource/Barack_Obama	http://dbpedia.org/resource/Honolulu
http://dbpedia.org/resource/Ali_Al-Daqbaashi	
http://dbpedia.org/resource/Britt_Bohlin_Olsson	
http://dbpedia.org/resource/Guðbjartur_Hannesson	
http://dbpedia.org/resource/Höskuldur_Þórhallsson	
http://dbpedia.org/resource/Vladimir_Putin	http://dbpedia.org/resource/Soviet_Union
http://dbpedia.org/resource/Vladimir_Putin	http://dbpedia.org/resource/Saint_Petersburg

SPARQL: Sortierung von Ergebnissen

Sortierung von Ergebnissen mit Schlüsselphrase `ORDER BY`

```
1 SELECT ?buch, ?preis
2 WHERE {
3   ?buch <http://example.org/preis> ?preis .
4 }
5 ORDER BY ?preis
```

- Sortierung wie bei Filter-Vergleichsoperatoren, Sortierung von URIs alphabetisch als Zeichenketten
- Reihenfolge **unterschiedlicher Arten** von Elementen: Ungebundene Variable < leere Knoten < URIs < RDF-Literale.
- Weitere mögliche Angaben
  - `ORDER BY DESC (?preis)`: absteigend
  - `ORDER BY ASC (?preis)`: aufsteigend (=Default-Verhalten)
  - `ORDER BY DESC(?preis), ?titel`: Hierarchische Kriterien

SPARQL: LIMIT, OFFSET, DISTINCT

LIMIT

- Beschränke die maximale Anzahl gelieferter Treffer
- **Sehr nützlich**, insbesondere aus **Performance-Gründen**!

OFFSET

- Position des **ersten** gelieferten Ergebnisses
- Ermöglicht in Kombination mit `LIMIT` und `ORDER BY` ein **Paging** durch die Ergebnisse.

SELECT DISTINCT

- entfernt Mehrfach-Treffer.

```
1 SELECT DISTINCT ?buch
2 WHERE {
3   ?buch <http://example.org/preis> ?preis .
4 }
5 ORDER BY ?preis
6 LIMIT 5
7 OFFSET 25
```

SPARQL: Textsuche

Um Knoten/Relationen zu finden *(z.B. wenn in einer Frage 'Angela Merkel' erwähnt wird)* bieten **Triple Stores** *(z.B. Virtuoso)* eine **Volltext-Indexierung** mit verschiedenen Abfragemöglichkeiten:

Exakter Match, z.B. über `rdfs:label`

```
1 SELECT ?person
2 {
3   ?person rdfs:label "Angela Merkel"@de }
```

Per Regexp-Filter

```
1 SELECT ?x
2 {
3   ?x rdfs:label ?label .
4   FILTER( regex(?label, "Angela") ) }
```

Spezieller Operator `bif:contains`[4]

```
1 SELECT ?x
2 {
3   ?x rdfs:label ?label .
4   ?label bif:contains "'Angela '" . }
```

[4] http://docs.openlinksw.com/virtuoso/rdfsparqlrulefulltext/

SPARQL: Typmatching von Literalen

Achtung: Beim Matching von Literalen ist eine **genaue Übereinstimmung** des **Datentyps** gefordert!

Beispiel

```
?entity1 ex:name ‘‘Bob’’ .
?entity2 ex:name ‘‘Bob’’^^xsd:string .
?entity3 ex:name ‘‘Bob’’@en .
?entity4 ex:age ‘‘42’’^^xsd:integer .
```

- Im Beispiel liefert die Anfrage { `?x ex:name ‘‘Bob’’` } **nur** `entity1`, d.h. es wird z.B. unterschieden zwischen Strings und Strings mit Sprach-Tags.

Ausnahmen

- Abkürzungen für Zahlenwerte sind möglich

{ `?x ex:age 42 .` } → *klappt* ✓

- Der Datentyp wird hierbei aus der syntaktischen Form des Literals bestimmt (z.B. `xsd:integer` vs. `xsd:double`).

SPARQL: Vorgestellte Features im Überblick

Grundstruktur
PREFIX
WHERE

Ausgabeformate
SELECT
CONSTRUCT
ASK
DESCRIBE

Graph-Muster
Einfache Graph-Muster
{...}
OPTIONAL
UNION

Filter
BOUND
isURI
isBLANK
isLITERAL
STR
LANG
DATATYPE
sameTERM
langMATCHES
REGEX

Modifikatoren
ORDER BY
LIMIT
OFFSET
DISTINCT

Outline

Semantic Web: Kritik

Kritiker der "Semantic Web" - Strömung führen u.a. folgende Argumente an[5]:

- **People are lazy**: Erheblicher Annotationsaufwand (z.B. **Verdoppelung** des Aufwands für die Content-Erstellung). User sind **Opportunisten**!
- Insbesondere die Erfassung von **common sense knowledge** ist extrem aufwändig *("Wenn's regnet, ist die Wiese nass")*.
- **People are stupid**: Qualität / Genauigkeit / Korrektheit der Metadaten oft fragwürdig. Insbesondere keine Methodik zur Beurteilung der **Signifikanz** von Fakten.

```
dbr:Germany dbo:leader dbr:Angela_Merkel .
dbr:Germany dbo:leader dbr:Andreas_Voßkuhle .
```

- **People lie**: SPAM-Problematik, Kampf um Meinungshoheit.

[5] C. Doctorov (2001): "Metacrap: Putting the Torch to seven Straw-men of the Meta-Utopia"

Semantic Web: Kritik

Limitierte Verbreitung

- Die Vision des **offenen**, **standardisierten**, **geteilten** Semantic Web *(siehe oben)* hat sich **bisher noch nicht durchgesetzt**.
- Tendenz zu **geschlossenen**, **proprietären** Ökosystemen (Facebook, Amazon, Google, ...).

> *" The semantic web idea [...] has turned out to be the opposite of what Tim Berners-Lee anticipated in the early 1990s. [...] TBL hoped for a giant* **structured open web**. *What we got first was a bunch of* **closed structured webs** *and a* **mixed web** *that still isn't very well structured. The open part of the web lives on, but isn't where most web users dwell nowadays."*

(Daniel Lemire. When bad ideas will not die: from classical AI to Linked Data (2014))

> *Ten years go by and* **hardly anything** *(outside academic conferences) comes out of this Semantic Web. Tens of thousands of research papers get written. Hardly anyone ever uses any of the technology produced [...]. Not to worry, Berners-Lee* **rebrands** *whatever remains as 'Linked Data'.*
>
> *Semantics are not going away though you may not always recognize its offspring ... it's in the background as a proud parent.*