Previously on “Knowledge Graphs”

• We have got to know
  – The RDF and RDFS languages
  – The Linked Open Data paradigm

• We have accessed KGs and Linked Open Data
  – with browsers and via programming frameworks
  – jumping from node to node in the graph

• ...let us have a closer look!
An Example RDF Graph

Question: in which states are the five biggest cities of Germany located?
Information Retrieval on Linked Open Data

- Question: in which states are the five biggest cities of Germany located?

- So let's try...

```
Germany
lies in 0001
lies in 0002
lies in 0003
lies in 0004
lies in ...

0020
state
inhabitants
HTTP GET
3.501.872

0001
inhabitants
HTTP GET
691.518

0002
inhabitants
HTTP GET
1.378.176

0003
inhabitants
HTTP GET
1.798.836

0004
inhabitants
HTTP GET

...```

10/12/22 Heiko Paulheim
Information Retrieval on Linked Open Data

• Observations
  – Navigation across derefencable URIs ultimately leads to a goal
  – But it is tedious
  – A lot of useless data is potentially retrieved

• Different information needs
  – Good for retrieving simple facts
  – Less efficient for more complex questions
Semantic Web and Linked Data


Berners-Lee (2009): Semantic Web and Linked Data

here be dragons...

Semantic Web Technologies (This lecture)

Technical Foundations
What Would We Like to Have?

Question: in which states are the five biggest cities of Germany located?
Wanted: A Query Language for the KGs

• ...just like SQL is for relational databases

```
SELECT  name, birthdate FROM customers
WHERE    id = '00423789'
```

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>birthdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>00183283</td>
<td>Stephen Smith</td>
<td>23.08.1975</td>
</tr>
<tr>
<td>00423782</td>
<td>Julia Meyer</td>
<td>05.09.1982</td>
</tr>
<tr>
<td>00789534</td>
<td>Sam Shepherd</td>
<td>31.03.1953</td>
</tr>
<tr>
<td>00423789</td>
<td>Herbert King</td>
<td>02.04.1960</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Wanted: A Query Language for the KGs

- **SPARQL**: "SPARQL Query Language for RDF"
  - a recursive acronym

- A W3C Standard since 2008
- Allows for querying RDF graphs
SPARQL: General Idea

- Look for specific subgraphs in a knowledge graph
  - Subgraphs may contain variables
SPARQL Basics: Graph Pattern Matching

- Subgraph: :Peter :knows :Stephen
  - Found in graph: no
- Subgraph: :Peter :knows :Julia
  - Found in graph: yes
SPARQL Basics: Graph Pattern Matching

- Subgraph with variable: \(?x :\text{knows} :\text{Julia}\).
  - Result: \(\{ ?x = :\text{Peter} \}\)
SPARQL Basics: Graph Pattern Matching

- Subgraph with variable: :Julia :knows ?x .
  - Result: { ?x = :Stephen, ?x = :Ellen }
SPARQL Basics: Graph Pattern Matching

- Subgraph with two variables: :x :knows ?y.
SPARQL Basics: Graph Pattern Matching

  - Result: { (?x = :Julia, ?y = :Stephen) ; (?x = :Julia, ?y = :Ellen)}
Hello SPARQL!

- Example:
  ```sparql
  SELECT ?child
  WHERE { :Stephen :fatherOf ?child }
  ```

Expressions with ? denote variables
SPARQL Syntax

• Basic structure:

    SELECT <list of variables>
    WHERE { <pattern> }

• Variables denoted with ?

• Prefixes as in RDF/N3:

    PREFIX foaf: <http://xmlns.com/foaf/0.1/>
    SELECT ?person ?name
    WHERE { ?person foaf:name ?name }
SPARQL Syntax

• The <pattern> in the WHERE clause is like N3
  – with variables

• {?p foaf:name ?n }
• {?p foaf:name ?n; foaf:homepage ?hp }
• {?p foaf:knows ?p1, ?p2 }
SPARQL: Pattern Matching on RDF Graphs

• A person who has a daughter and a son:
  
  \{ ?p :hasDaughter ?d ; :hasSon ?s . \}

• A person knowing two persons who know each other
  

• A person who has two children:
  
  \{ ?p :hasChild ?c1, ?c2 . \}
• A person who has two children:
  \{ ?p :hasChild ?c1, ?c2 . \}

• ResultSet:

Observation: different variables are not necessarily bound to different resources!
SPARQL: Blank Nodes

- **WHERE clause: an RDF graph with variables**

  ```sparql
  SELECT ?person1 ?person2 ?otherPerson
  WHERE {
    ?person1 :knows ?otherPerson .
  }
  ```

- **Result:**
  - ?person1 = :Peter, ?person2 = :Julia; ?otherPerson = _:x1

- **Note:** Blank Node IDs are only unique within one result set!
SPARQL: Matching Literals

- **Strings**
  
  \{ ?country :name "Germany" . \}

- **Watch out for language tags!**
  
  \{ ?country :name "Germany"@en . \}

  → The Strings "Germany" and "Germany"@en are different!

- **Numbers:**
  
  \{ ?person :age "42"^^xsd:int . \}

  Short hand notation:

  \{ ?person :age 42 . \}
SPARQL: Filters

• Used for further restricting results
  
  `{?person :age ?age . FILTER(?age < 42) }`

• Operators for comparisons:
  
  `=    !=    <    >    <=    >=`

• Logical operations:
  
  `&&    ||    !`
SPARQL: Filters

- Persons with younger siblings
  
  \[
  \{ \quad \text{?p1 :siblingOf ?p2 .} \\
  \quad \text{?p1 :age ?a1 .} \\
  \quad \text{?p2 :age ?a2 .} \\
  \quad \text{FILTER(?a2 < ?a1)} \}
  \]

- Persons that have both younger and older siblings
  
  \[
  \{ \quad \text{?p1 :siblingOf ?p2,?p3 .} \\
  \quad \text{?p1 :age ?a1 .} \\
  \quad \text{?p2 :age ?a2 .} \\
  \quad \text{?p3 :age ?a3 .} \\
  \quad \text{FILTER(?a2 < ?a1 && ?a3 > ?a1)} \}
  \]

Question: why do we get different persons for p2 and p3?
SPARQL: Filters

- Second try: a person with two children
  
  ```sparql
  { ?p :hasChild ?c1, ?c2 . FILTER( ?c1 != ?c2) }
  ```

- A slight improvement
  
  → Variables are now bound to different resources

- But: we still have the Non-Unique Naming Assumption
  
  → i.e., given that
  
  ```sparql
  :Peter :hasChild :Julia .
  :Peter :hasChild :Stefan .
  ```

  we still cannot conclude that Peter has two children!

- Furthermore, there is still the Open World Assumption
  
  → i.e., Peter could also have more children
Filters for Strings

• Searching in Strings: using regular expressions

• People called “Ann”
  
  `{?person :name ?n . FILTER(regex(?n,"^Ann$")) }`
  `{?person :name ?n . FILTER(regex(?n,"Ann")) }`
  → the second variant would also find, e.g., “Mary-Ann”

• `str`: URIs and Literals as strings
  • allows for, e.g., searching for literals across languages
    
    `{?country :name ?n . FILTER(str(?n) = "Tyskland") }`
Further Built-In Features

- Querying the type of a resource
  - isURI
  - isBLANK
  - isLITERAL

- Querying for the data type and language tags of literals
  - DATATYPE(?v)
  - LANG(?v)

- Comparing the language of two literals
  - langMATCHES(?v1, ?v2)
  - Caution: given ?v1 = "Januar"@DE, ?v2 = "Jänner"@DE-at
    LANG(?v1) = LANG(?v2) → false
    langMATCHES(?v1, ?v2) → true
Combining Patterns

• Find the private and work phone number

\[
\{ \ ?p :\text{privatePhone} \ ?nr \ \} \\
\text{UNION} \ \{ \ ?p :\text{workPhone} \ ?nr \ \}
\]

• UNION creates a set union

\[\text{?p = :peter, ?nr = 123;} \]
\[\text{?p = :john, ?nr = 234;} \]
\[\text{?p = :john, ?nr = 345;} \]
\[\ldots \]

That happens if John has both a private and a work phone
Interlude: A Real-World Example


Kopf der Runde ist nach der endgültigen
Interlude: A Real-World Example

Who is this Walter K.?"
Interlude: A Real-World Example

SELECT DISTINCT(?x) WHERE {
  ?x a dbo:SoccerPlayer .
  ?x dbo:careerStation ?s. ?s dbo:team dbr:Germany_national_football_team.
}

Who is this Walter K.?
Interlude: A Real-World Example

Who is this Walter K.?


We get one result:

<http://dbpedia.org/resource/Walter_Kelsch>
Auf schiefere Bahn geraten
Ex-Nationalspieler Kelsch sitzt in U-Haft


Der ehemalige Fußball-Nationalspieler Walter Kelsch sitzt wegen Drogenhandels im Internet in Stuttgart-Stammheim in Untersuchungshaft. Dies bestätigte die Staatsanwaltschaft im niedersächsischen Verden.


Quelle: n-tv.de, whoisl
Optional Patterns

• Find a person's phone number and fax number, **if existing**

```reason
{ ?p :phone ?tel }
OPTIONAL { ?p :fax ?fax }
```

• **OPTIONAL** also creates unbound variables

```
?p = :julia, ?nr = 978; ?fax = 349;
...
```

**Unbound variable:** John does not have a fax number (as far as we know)
Unbound Variables

- Variables can remain unbound
- We can test this with BOUND

- Everybody who has a phone or a fax (or both):

```sparql
OPTIONAL { ?p :phone ?tel . }
OPTIONAL { ?p :fax ?fax . }
FILTER ( BOUND(?tel) || BOUND(?fax) )
```
Negation

• This is a common question w.r.t. SPARQL

• How do I do this:
  – "Find all persons who do not have siblings."

• This is left out of SPARQL intentionally!

• Why?

• Open World Assumption
  – we cannot know!

• For the same reason, there is no \texttt{COUNT}
  – at least not in standard SPARQL
Negation – Hacking SPARQL

• However, there is a possibility
  – try with caution!

• Using `OPTIONAL` and `BOUND`

• Find all persons without siblings
  
  ```sparql
  OPTIONAL {?p :hasSibling ?s . }
  FILTER ( !BOUND(?s) )
  ```

• This works

• However, you should know what you are doing
  – ...and how to interpret the results!
Negation – Hacking SPARQL

• How does that work?

• Results before FILTER:

  OPTIONAL { ?p :hasSibling ?s . }

  ?p = :peter, ?s = :julia
  ?p = :julia, ?s = :peter
  ?p = :mary, ?s =
  ?p = :paul, ?s =

• Applying the FILTER

  − FILTER(!BOUND(?s))

  ?p = :mary, ?s =
  ?p = :paul, ?s =

Unbound variables
Sorting and Paging Results

- **Sorting**: ORDER BY ?name
- **Limitations**: LIMIT 100
- **Lower Bounds**: OFFSET 200

Example: persons 101-200, ordered by name
- ORDER BY ?name LIMIT 100 OFFSET 100

**LIMIT/OFFSET without ORDER BY:**
- Result orderings are not deterministic
- There is no default ordering
Sorting and Paging Results

• Application scenarios:
  – Some SPARQL services limit their result set sizes
  – Pre-loading in applications

• Application example:
  – let the user browse cities
  – it is more likely that users want to see the big cities
  – display 100 biggest cities on one page, show more on demand

• SELECT ?city ?population
  WHERE {?city hasPopulation ?population}
  ORDER BY DESC(?population)
  LIMIT 100
Filtering Duplicates

• SELECT DISTINCT ?person
  WHERE { ?person :privatePhone ?nr }
  UNION { ?person :workPhone ?nr }

• This means: all results with identical variable bindings are filtered

• This does not mean: persons identified by ?person are actually different

• Why?
  – Non-unique naming assumption
Custom Built-Ins

• Some SPARQL engines allow special constructs
• also known as *Custom Built-Ins*
• Example: geographic processing
  – Knowledge Graph: DBpedia
  – Runs on Virtuoso
Custom Built-Ins

- Querying for coordinates
  - simple:
    ```sql
    SELECT ?x
    WHERE { ?x geo:long ?long; geo:lat ?lat .
    FILTER (?long>8.46 && ?long<8.47 &&
              ?lat>49.48 && ?lat<49.49)}
    ```
<table>
<thead>
<tr>
<th>SPARQL</th>
<th>HTML5 table</th>
</tr>
</thead>
<tbody>
<tr>
<td>×</td>
<td><a href="http://dbpedia.org/resource/Action_at_Mannheim_(1795)">http://dbpedia.org/resource/Action_at_Mannheim_(1795)</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/GESIS_-_Leibniz_Institute_for_the_Social_Sciences">http://dbpedia.org/resource/GESIS_-_Leibniz_Institute_for_the_Social_Sciences</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Reiss_Engelhorn_Museum">http://dbpedia.org/resource/Reiss_Engelhorn_Museum</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/University_of_Mannheim_University_of_Mannheim_1">http://dbpedia.org/resource/University_of_Mannheim_University_of_Mannheim_1</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/University_of_Mannheim">http://dbpedia.org/resource/University_of_Mannheim</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/University_of_Mannheim_Business_School">http://dbpedia.org/resource/University_of_Mannheim_Business_School</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Jesuit_Church_Mannheim">http://dbpedia.org/resource/Jesuit_Church_Mannheim</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Battle_of_Mannheim_(1799)">http://dbpedia.org/resource/Battle_of_Mannheim_(1799)</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Battle_of_Seckenheim">http://dbpedia.org/resource/Battle_of_Seckenheim</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Klapsm%C3%BChle_am_Rathaus">http://dbpedia.org/resource/Klapsmühle_am_Rathaus</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Mannheim">http://dbpedia.org/resource/Mannheim</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Mannheim_Business_School">http://dbpedia.org/resource/Mannheim_Business_School</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Mannheim_University_Library">http://dbpedia.org/resource/Mannheim_University_Library</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Zentrum_f%C3%BCr_Europ%C3%A4ische_Wirtschaftsforschung">http://dbpedia.org/resource/Zentrum_für_Europäische_Wirtschaftsforschung</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Mannheim_Observatory">http://dbpedia.org/resource/Mannheim_Observatory</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Palais_Bretzenheim">http://dbpedia.org/resource/Palais_Bretzenheim</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Graduate_School_of_Economic_and_Social_Sciences">http://dbpedia.org/resource/Graduate_School_of_Economic_and_Social_Sciences</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Mannheim_Palace">http://dbpedia.org/resource/Mannheim_Palace</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dbpedia.org/resource/Mannheim_Palace_Church">http://dbpedia.org/resource/Mannheim_Palace_Church</a></td>
</tr>
</tbody>
</table>
Custom Built-Ins

- More complex queries
  - all museums within a 10km radius of a given point

```sparql
SELECT ?x
WHERE { ?x rdf:type dbo:Museum; geo:long ?long;
  geo:lat ?lat
FILTER (bif:st_intersects(
    bif:st_point(?long, ?lat),
    bif:st_point(8.466, 49.488), 10))}
```

```
http://dbpedia.org/resource/Automuseum_Dr_Carl_Benz
http://dbpedia.org/resource/Reiss_Engelhorn_Museum
http://dbpedia.org/resource/Kunsthalle_Mannheim
http://dbpedia.org/resource/Technoseum
```
Further Query Types: ASK

- So far, we have only looked at SELECT queries
- ASK allows for yes/no queries:
  - e.g., are there persons with siblings?
    
      ASK { ?p :hasSibling ?s . }
    
- Often faster than SELECT queries

- The answer is true or false
  - false means: there are no matching sub graphs
  - do not misinterpret (Open World Assumption!)
Further Query Types: DESCRIBE

- All properties of a resource

  DESCRIBE <http://dbpedia.org/resource/Berlin>

- Can be combined with a WHERE clause

  DESCRIBE ?city WHERE { :Peter :livesIn ?city . }

- Allows for exploration of a dataset with unknown structure

- Caution: types of results are not standardized, results vary from implementation to implementation!
Further Query Types: CONSTRUCT

• Creates a new RDF graph

CONSTRUCT
{ ?x rdfs:seeAlso <http://dbpedia.org/resource/Berlin> . } WHERE
{ <http://dbpedia.org/resource/Berlin> ?y ?x . FILTER (isURI(?x)) } 

• CONSTRUCT returns complete RDF graphs
  – e.g., for further processing
Query Federation

- Queries can be answered over *multiple* SPARQL endpoints
- Example

```sparql
SELECT ?name ?lat ?long WHERE {
    ?x a foaf:Person .
    ?x foaf:name ?name .
    ?city owl:sameAs ?dbpediacity .
    SERVICE <http://dbpedia.org/sparql> {
        ?dbpediacity geo:lat ?lat .
    }
}
```

:John

```
:London
```

"John Smith"

```
:534789
```

"51.50939"

"-0.11832"
SPARQL: Wrap-Up

• SPARQL is a query language for the semantic web
• Basic principle: pattern matching on graphs
• SPARQL allows for directed search for information instead of navigating the graph from node to node

• Results follow the semantic principles of RDF!
  – Open World Assumption
  – Non-unique naming assumption
Example: Jena + SPARQL

• Querying models with SPARQL

```java
String queryString = "SELECT ?x ...";
Query query = QueryFactory.create(queryString);
QueryExecution qe =
    QueryExecutionFactory.create(query, model);
ResultSet results = qe.execSelect();
while(results.hasNext()) {
    QuerySolution sol = results.next();
    String s = sol.get("x").toString();
    ...
}
```
SPARQL and Reasoning

- Important note:
  - By default, SPARQL does **not** use any reasoning

- Given the following T-box
  :Museum rdfs:subClassOf :Building .
  
  and A-box
  :Technoseum rdf:type :Museum .

- The result of
  
  ```sparql
  SELECT ?x WHERE {?x rdf:type :Building}
  ```
  
  is empty!
SPARQL and Reasoning

• Important note:
  – By default, SPARQL does **not** use any reasoning

• Given the following T-box
  
  :Museum rdfs:subClassOf :Building .

  and A-box

  :Technoseum rdf:type :Museum .

• We would have to use something like

  ```sparql
  SELECT ?x WHERE {
    {?x rdf:type :Building } 
    UNION  {?x rdf:type ?t . 
    ?t rdfs:subClassOf :Building }
  ```
Recap: Reasoning with Jena

- Given: a schema and some data

```java
Model schemaModel = ModelFactory.createDefaultModel();
InputStream IS = new FileInputStream("data/example_schema.rdf");
schemaModel.read(IS);

Model dataModel = ModelFactory.createDefaultModel();
IS = new FileInputStream("data/example_data.rdf");
dataModel.read(IS);

Model reasoningModel =
    ModelFactory.createRDFSModel(schemaModel, dataModel);
```

- Now, reasoningModel contains all derived facts
Example: Jena + SPARQL + Reasoning

• Derived facts can also be queries with SPARQL

• Given the reasoningModel

Query q = QueryFactory.create(
   "SELECT ?t WHERE
   { <http://example.org/Madrid>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?t .}");
QueryExecution qexec =
   QueryExecutionFactory.create(q, reasoningModel);
ResultSet rs = qexec.execSelect();
   while(rs.hasNext())
      String type = rs.next().get("t");

• Here, the query produces two solutions
  - http://example.org/City
  - http://www.w3.org/2000/01/rdf-schema#Resource
Accessing Public SPARQL Endpoints

- SPARQL Endpoints are an important building block of the Semantic Web tool stack

- Access using Jena:

```java
String query = "SELECT ...";
String endpoint = "http://dbpedia.org/sparql";
Query q = QueryFactory.create(strQuery);
QueryExecution qexec =
    QueryExecutionFactory.sparqlService(endpoint, q);
ResultSet RS = qexec.executeSelect();
```
Accessing Public SPARQL Endpoints

• Recap:
  – Jena uses the iterator pattern quite frequently

• Observation:
  – SPARQL ResultSets are also like iterators
  – Data can be retrieved from the server little by little
Triple Pattern Fragments

• Observation:
  – Operating SPARQL endpoints is costly
    • Hence, there are often downtimes
    • Maintenance often ends as project funding runs out
  – Accessing data via dumps or dereferencing is time consuming
    • See initial experiment

• Triple Pattern Fragments provide a middle ground solution

http://linkeddatafragments.org
Triple Pattern Fragments

- Only allow simple restrictions
  - i.e., only {?s ?p ?o}
- Provide results in a paged fashion
  - Estimated count
  - Links to further pages

Data Portal @ linkeddatafragments.org

DBpedia 2014
Query DBpedia 2014 by triple pattern

subject: http://www.w3.org/1999/02/22-rdf-syntax-ns#text
predicate: http://www.w3.org/1999/02/22-rdf-syntax-ns#type
object: http://dbpedia.org/ontology/Restaurant

Find matching triples

Matches in DBpedia 2014 for {?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://...}
Showing triples 1 to 100 of ±1,213 with 100 triples per page.  next
- _t_Brouwerskolkje type Restaurant.
- _t_Fornuis type Restaurant.
- _t_Ganzenest type Restaurant.
- _t_Koetschuis type Restaurant.
- _t_Wilverstant type Restaurant.
- _t_Nonnefje type Restaurant.
- _t_Schuiten_Huys type Restaurant.
- _t_Veerhuis type Restaurant.
- 1321_Downtown_Taproom_Bistro type Restaurant.
- 21_Club type Restaurant.
- 36_on_the_Quay type Restaurant.
- 5_North_St type Restaurant.
- 65-85_Bak_and_Restaurant type Restaurant.
- Aan_De_Poel type Restaurant.
- AD_Hoc_Restaurant type Restaurant.
- Aekloere type Restaurant.
- Alain_Ducasse_at_the_Dorchester type Restaurant.
- Alba_Italian_Restaurant type Restaurant.

http://linkeddatafragments.org
Triple Pattern Fragments

- Most SPARQL queries can be solved by iteratively retrieving TPFs
  - Successively issuing new *selectors*
  - More targeted, i.e., less calls, than dereferencing individual URIs

http://linkeddatafragments.org
Triple Pattern Fragments

- Example: astronauts born in capital countries

  ```
  select ?x ?y ?z where {
    ?x a dbpedia-owl:Astronaut .
    ?x dbpedia-owl:birthPlace ?y .
    ?z a dbpedia-owl:Country
  }
  ```

- Algorithm:
  - retrieve pattern: ?x a dbpedia-owl:Astronaut .  
  - for each result ?x: retrieve ?x dbpedia-owl:birthPlace ?y .  
  - for each result ?z: check ?z a dbpedia-owl:Country .
Triple Pattern Fragments

• Middle ground between
  – setting up a SPARQL server (costly for the server)
  – providing a full RDF dump (costly for the client)

• In our example, a SPARQL query was broken down into ~3k HTTP GET requests
  – Using clever index structures, this might still be faster
  – Results may also be streamed – allows for early stopping
Triple Pattern Fragments vs. SPARQL

• All SPARQL constructs can be translated to a TPF query plan
• Some are quite fast
  – e.g., typical star-shaped queries
• Some are rather slow
  – e.g., regex queries for labels
Semantic Web Technology Stack

here be dragons...

Semantic Web Technologies (This lecture)

Technical Foundations

Berners-Lee (2009): Semantic Web and Linked Data
Questions?
Knowledge Graphs
Introduction to Student Projects

Heiko Paulheim, Sven Hertling
Student Projects

• Goals
  – Gain more practical experience with KGs
  – Become familiar with existing datasets
  – Understand possibilities and limitations of existing KGs

• Expectation
  – Choose one or more (preferably more) KGs
  – Build an interesting application with it
Interesting Applications

• Just a few possible examples
  – Quiz applications
  – Mobile apps with local information
  – Expert systems for a special domain
  – ...
Procedure

• Teams of two students
  1. realize a semantic web project
  2. write a 10 to 12 page summary of the project and the methods employed in the project
  3. present the project results to the other students

• Finding a team
  – see previous e-mails (ILIAS forum, Google Sheet)

• Final mark for the course
  ▪ will be entirely based on the exam
  ▪ the project, report, and presentation are a mandatory requirement!
Requirements

- The project you develop should
  - solve a real world task for end users
  - use one or more Knowledge Graphs
  - involve some processing beyond mere display of the data
Project Outlines

- 2-3 pages (sharp!) without title and TOC pages, DWS master thesis layout
- due Sunday, October 16th, 23:59
- send by e-mail to Sven
- answer the following questions:
  - What is the goal of the application you are going to build?
  - What are the example results you expect?
  - What datasets are you planning to use?
  - What techniques are you going to use?
  - How do you plan to evaluate your results?
Project Reports

• 10-12 pages (sharp!) without title and toc pages
• due Friday, December 9\textsuperscript{th}, 23:59
• send by e-mail to Sven and Heiko
• describe your solution including the steps to get there:
  1. Application domain and goals
  2. Datasets used
  3. Techniques used
  4. Example results
  5. Known limitations
  6. Lessons learned

• Requirements
  ■ Use the DWS master thesis layout
  ■ Please cite sources properly
Project Reports

• Application domain and goals
  – Which users are targeted?
  – Which user problems are solved?
  – Which user information needs are addressed?

• Datasets used
  – Which KGs does the application use?
  – How are they accessed (SPARQL, local)?
  – How do you combine information from different sources?

• Techniques used, e.g.
  – Reasoning
  – Search
  – external services
Example results
- What outcomes does the application provide?
- How is some user queries answered?

Known limitations
- In which domains does the application not work?
- Are there queries which cannot be answered?
- Why?
- How could you overcome those limitations, given more time?

Lessons learned
- Which challenges did you face?
- What were the biggest obstacles?
- What would you do differently next time?
Deadlines at a Glance

• Submission of project work proposal
  – Sunday, October 16\textsuperscript{th}, 23:59

• Submission of final project work report
  – Friday, December 9\textsuperscript{th}, 23:59

• Final presentations
  – Tuesday, December 6\textsuperscript{th}, lecture slot
Questions?