SPARQL: Die RDF Abfragesprache

Markus Stocker

14. Mai 2018

Rekapitulation

- Wozu wird das Prädikat `rdf:type` verwendet?
- Nennen Sie einige XSD Datentypen
- Erläutern Sie warum diese ungleich sind
  - `"14.5"`
  - `"14.5"^^xsd:string`
  - `"14.5"^^xsd:decimal`
- Welche RDF Listen gibt es und wozu verwendet man diese?
- Was ist Reifizierung?

Übersicht

- Einführung in SPARQL
- Konzepte
- Beispiele

Was ist SQL
und wozu verwendet man die Sprache?

Eine Abfragesprache für RDF

- Mit RDF kann man Information maschinenlesbar beschreiben
- Zum Beispiel “Erde ist ein Planet mit Radius 6371 km”
- Nun möchte man auf solche Information zugreifen können
- Zum Beispiel “Alle Planeten mit Radius grösser als 6000 km”
- Dazu benötigt man eine Abfragesprache

SPARQL

- **SPARQL** **P**rotocol **A**nd **R**DF **Q**uery **L**anguage
- Ermöglicht Zugang auf in RDF beschriebene Information
- Erinnert etwas an die Structured Query Language (SQL)
- Natürlich zugeschnitten auf RDF Graphen
- W3C Recommendation seit 2008
- Neuere Version SPARQL 1.1 seit 2013

SPARQL: Beispiel

```
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#>

SELECT ?label
WHERE {
  ?planet rdf:type ex:Planet .
  ?planet rdfs:label ?label .
}

=>

| label                |
| -------------------- |
| "Earth"              |
| -------------------- |
```

Triple Pattern

- Wie besprochen, ist das Tripel ein zentrales Konstrukt in RDF
- Ein RDF Dokument ist eine Tripelmenge
- Mit Tripelmuster (*triple pattern*) kann man solche Mengen abfragen

Triple Pattern: Beispiel

Ein RDF Dokument mit einem Tripel

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

ex:Earth rdf:type ex:Planet .
```

Ein (passendes) *triple pattern*

```
?planet rdf:type ex:Planet .
```

Triple Pattern: Variabel

- Zentral für das Tripelmuster ist die Variabel
- Ein *triple pattern* kann eine oder mehrere Variabeln enthalten
- Beliebig in Subjekt-, Prädikat- oder Objektposition
- Variabeln beginnen mit einem Fragezeichen (`?`)
- Variabelname kann frei gewählt werden
- Sollte allerdings sinnvoll sein: `?planet` besser als `?p45t`

Variabel: Beispiele

```
?planet rdf:type ex:Planet .

ex:Earth rdf:type ?type .

ex:Earth ?predicate ?object .

?subject rdf:type ?object .

?subject ?predicate ?object .

?s ?p ?o .
```

Abfrage Ausführen: Tripelmuster über -menge Auswerten

- In einer Abfrage wird das Muster über eine Menge ausgewertet
- Es werden Variabeln mit RDF Knoten oder Kanten ersetzt
- Diese Ersetzung ergibt das Resultat der Abfrage

Muster über Menge Auswerten: Beispiel

Auswertung Tripelmuster

```
?planet rdf:type ex:Planet .
```

über die Tripelmenge (Tripel die auf Muster passen in Blau)

```
ex:Earth rdfs:label "Earth" .
ex:Earth rdf:type ex:Planet .
ex:Mars rdfs:label "Mars" .
ex:Earth ex:radius "6371"^^xsd:decimal .
ex:Mars rdf:type ex:Planet .
```

Resultat der Abfrage mit zwei Variabelersetzungen

1. `?planet <- ex:Earth`
2. `?planet <- ex:Mars`

Basic Graph Pattern

- Mit einem *triple pattern* kann man keine komplexe Abfragen stellen
- Dazu benötigt man eine *triple pattern* Menge
- Auch *basic graph pattern* (BGP) genannt
- Ein BGP kann natürlich aus einem einzigen *triple pattern* bestehen

Basic Graph Pattern: Beispiel

```
?planet rdf:type ex:Planet .
?planet ex:radius ?radius .
```

Verbindungen (*Joins*)

- Variabelnamen definieren Verbindungen zwischen *triple patterns*
- Können beliebig über Subjekte, Prädikate, Objekte definiert werden
- Gleichbenannte Variabeln müssen mit gleichen Knoten ersetzt werden

```
?planet rdf:type ex:Planet .
?planet ex:radius ?radius .
?planet ex:satellite ?satellite .
?satellite rdfs:label ?label .
```

Group Graph Pattern

- *Basic graph patterns* können gruppiert werden
- Dazu verwendet man geschweifte Klammern, { und }
- Bedingungen (z.B. `FILTER`) können so auf Gruppen limitiet werden

```
{
  ?planet rdf:type ex:Planet .
  ?planet ex:radius ?radius .
}
```

```
{
  { ?planet rdf:type ex:Planet . }
  { ?planet ex:radius ?radius . }
}
```

FILTER

- FILTER ermöglicht Angabe von Bedingungen
- Variabelnersetzungen müssen diese erfüllen
- Somit eine Einschränkung der Resultatsmenge
- Geltungsbereich ist das *group graph pattern* welches FILTER definiert

```
{
  ?planet rdf:type ex:Planet .
  ?planet ex:radius ?radius .
  FILTER (?radius > 6000)
}
```

FILTER: Operatoren

- Vergleichs, boolesche, arithmetische und spezielle Operatoren
- Vergleichsoperatoren: `=`, `>`, `<`, `>=`, `<=`, `!=`
- Definiert für Datentypen mit natürlicher Ordnung, inklusive Zeit
- Boolesche Operatoren: `&&`, `||`, `!`
- Arithmetische Operatoren: `+`, `-`, `*`, `/`
- Definiert für numerische Werte
- Einige spezielle Operatoren: `LANG()`, `DATATYPE()`, `REGEX()`

FILTER Operatoren: Beispiele

```
?planet rdf:type ex:Planet .
?planet rdfs:label ?label .
?planet ex:radius ?radius .

FILTER (?radius > 6000 && REGEX(?label, "^E"))
FILTER (DATATYPE(?radius) = xsd:decimal)
FILTER (LANG(?label) = "en")
```

OPTIONAL

- Ermöglicht die Definition von optionalen *group graph patterns*
- Optionale *patterns* müssen Variabeln nicht zwingend ersetzen
- Fehlt die Ersetzung wird das Resultat nicht aus der Menge genommen
- Variabeln in optionalen *patterns* können somit ungebunden sein

```
{
  ?planet rdf:type ex:Planet .
  OPTIONAL { ?planet ex:radius ?radius }
}
```

```
| planet    | radius                |
| ----------| --------------------- |
| ex:Earth  | "6371"^^xsd:decimal   |
| ex:Mars   |                       |
| ----------| --------------------- |
```

OPTIONAL: Beispiele

```
{
  ?planet rdf:type ex:Planet .
  OPTIONAL {
    ?planet ex:radius ?radius .
    ?planet ex:satellite ?satellite .
  }
}

{
  ?planet rdf:type ex:Planet .
  OPTIONAL { ?planet ex:radius ?radius . }
  OPTIONAL { ?planet ex:satellite ?satellite . }
}
```

UNION

- Ermöglicht die Definition von alternativen *group graph patterns*
- Bildet die Vereinigungsmenge der unabhängigen Resultatsmengen

```
{
  ?planet rdf:type ex:Planet .
  { ?planet ex:radius ?radius . }
  UNION
  { ?planet ex:satellite ?satellite . }
}
```

Literale in SPARQL

- RDF unterscheidet typisierte und untypisierte Literale
- Somit sind `"6371"` und `"6371"^^xsd:decimal` nicht gleich
- Datentypen in Abfragen könnnen die Resultatsmenge verändern

```
?planet ex:radius "6371" .

?planet ex:radius "6371"^^xsd:decimal .

?planet ex:radius "6371.0"^^xsd:decimal .

?planet ex:radius "6371"^^xsd:string .

?planet ex:radius 6371 .
```

Blank Nodes

- Nicht-ausgezeichnete (*non-distinguished*) Variabeln
- Keine Referenzen auf spezifische RDF *blank nodes*
- Können in Form `_:abc` oder `[]` geschrieben werden
- Werden in der Abfrageverarbeitung ersetzt
- Sind aber nicht Teil der Resultatsmenge
- Der gleiche *blank node* kann nicht in zwei BGPs vorkommen

Blank Nodes: Beispiele

```
# Mit ausgezeichneten Variabeln
?planet rdf:type ex:Planet .
?planet ex:radius ?radius .
?planet ex:satellite ?satellite .
?satellite rdfs:label ?label .

# Mit blank nodes (nicht-ausgezeichneten Variabeln)
_:a1 rdf:type ex:Planet .
_:a1 ex:radius ?radius .
_:a1 ex:satellite _:a2 .
_:a2 rdfs:label ?label .

# Mit blank nodes (verkuerzte Form)
[] rdf:type ex:Planet ;
   ex:radius ?radius ;
   ex:satellite [ rdfs:label ?label ]
```

Zusammenfassung

- Um RDF Daten zu verarbeiten benötigt es einer Abfragesprache
- Mittels SPARQL flexibel auf RDF Daten zugreifen
- Indem die gewünschte Resultatsmenge deklariert wird
- Das *triple pattern* als zentrales SPARQL Konstrukt
- Subjekt, Prädikat, Objekt dürften Variabel sein
- Eine Abfrage wertet solche Muster über Tripelmenge aus