SPARQL: Fortgeschrittene Themen

Markus Stocker

28. Mai 2018

Rekapitulation

- Was ist SPARQL und warum benötigt man sowas?
- Was ist ein *triple pattern*? Wie unterscheidet es sich von einem *triple*?
- Wozu dienen Variabeln in SPARQL?
- Was ist ein *basic graph pattern*?
- Womit kann man Resultatsmengen einschränken?

Rekapitulation: Erklären Sie diese Abfrage

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

Übersicht

- Resultatformate
- Modifizierer
- SPARQL Update
- SPARQL Endpoints
- Abfragenoptimierung

Resultatformate

- `SELECT`: Variabeln und deren Ersetzungen
- `CONSTRUCT`: RDF Dokument
- `ASK`: Wahr oder falsch
- `DESCRIBE`: RDF Ressourcen beschreiben

Resultatformate: `SELECT`

- Das `SELECT` Format haben wir bereits gesehen
- Angabe einer oder mehrerer Variabeln
- Oder $*$ als Kurzform für alle Variabeln die in der Abfrage vorkommen
- Die Variabelnersetzungen (Resultate) werden als Tabelle angezeigt
- Die Tabellendarstellung hat auch Nachteile
- Zum Beispiel, nicht unbedingt geeignet für Weiterverarbeitung
- Aber auch ungewünschte Redundanz (Werte in einer Spalte)

Resultatformate: `CONSTRUCT`

- Resultate werden als RDF Dokument zurückgegeben
- `CONSTRUCT` erwartet ein *template* für das zurückgegebene RDF
- Ein *template* kann flexibel gestaltet werden
- Neue Tripel oder spezifische Werte einführen, Prädikate ersetzen
- Das resultierende RDF kann direkt weiterverarbeitet werden

CONSTRUCT: 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#>

CONSTRUCT {
  ?planet rdf:type ex:Planet .
  ?planet rdf:type ex:TerrestrialPlanet .
  ?planet rdfs:label ?label .
  ?planet ex:radius ?radius .
}
WHERE {
  ?planet rdf:type ex:Planet .
  ?planet rdfs:label ?label .
  ?planet ex:radius ?radius .
  FILTER (?radius > 2000 & ?radius < 8000)
}
```

Resultatformate: `ASK`

- Resultat ist entweder *true* oder *false*
- *True* wenn die Abfrage zutreffende Resultate liefert, sonst *false*
- Kann benutzt werden um zu testen ob es Resultate gibt

ASK: Beispiel

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

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

Resultatformate: `DESCRIBE`

- Damit kann man RDF navigieren ohne die Struktur zu kennen
- Die einfachste `DESCRIBE` Abfrage ist für eine einzelne Ressource
- `DESCRIBE <http://example.org#Earth>`
- Variabelersetzte Ressourcen können auch beschrieben werden

DESCRIBE: Beispiel

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

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

Modifizierer

- Oft ist nicht bekannt wie gross eine Resultatsmenge sein wird
- Oft ist die Menge viel zu gross und für Benutzer nicht brauchbar
- Abhilfe bieten Operatoren die die Ergebnissequenz steuern und ändern
- Zum Beispiel, Sortierung steuern und Ergebnismenge limitieren
- Man kann so z.B. die *top 10* Resultate erhalten

Modifizierer: ORDER BY

- Sortierung in auf- (ASC) oder absteigender (DESC) Reihenfolge
- Sinnvol nur im Zusammenhang mit SELECT Abfragen

```
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 ?radius
WHERE {
  ?planet rdf:type ex:Planet .
  ?planet rdfs:label ?label .
  ?planet ex:radius ?radius .
}
ORDER BY DESC(?radius)
```

Modifizierer: LIMIT und OFFSET

- Ermöglicht die Adressierung eines Ausschnitts der Resultatsmenge
- Stückweise Verarbeitung der Resultatsmenge
- Folgendes Beispiel sind die *top 10* Resultate

```
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 ?radius
WHERE {
  ?planet rdf:type ex:Planet .
  ?planet rdfs:label ?label .
  ?planet ex:radius ?radius .
}
ORDER BY DESC(?radius) LIMIT 10 OFFSET 0
```

SPARQL Update

- Bis anhin haben wir nur Abfragen gestellt
- Also, deklarativer Zugriff auf Information in RDF
- Man kann Information in RDF mittels SPARQL auch ändern
- Insbesondere neue Information hinzufügen oder löschen
- Wurde mit SPARQL 1.1 eingeführt

SPARQL Update: INSERT (DELETE) DATA

```
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#>

INSERT DATA
{
      ex:Mars rdf:type ex:Planet .
      ex:Mars rdfs:label "Mars" .
}
```

SPARQL Update: DELETE/INSERT

```
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#>

DELETE { ?planet rdf:type ex:Planet }
INSERT { ?planet rdf:type ex:TerrestrialPlanet }
WHERE {
  ?planet rdf:type ex:Planet .
  ?planet ex:radius ?radius .
  FILTER (?radius > 2000 & ?radius < 8000)
}
```

SPARQL Endpoints

- Web Service für SPARQL Abfragen auf RDF Datenbank
- Üblicherweise mit graphischer und Programmierschnittstelle
- Endpoints implementieren SPARQL Protokoll
- Das Übermittlungsprotokoll für Abfragen und Resultate
- Übermittlung von Resultaten auch in verschiedene Formate
- Zum Beispiel XML oder CSV

SPARQL Endpoints: Beispiel DBpedia

- Gehen Sie mal auf `http://dbpedia.org/sparql/`
- Und kopieren Sie die folgende Abfrage ins Textfeld
- Führen Sie die Abfrage dann aus (*Run Query*)

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?populationMetro ?populationTotal ?federalState ?country
WHERE {
  [] rdfs:label "Hannover"@de ;
     dbo:populationMetro ?populationMetro ;
     dbo:populationTotal ?populationTotal ;
     dbo:federalState ?federalState ;
     dbo:country ?country
}
```

Etwas aus der Eigenen Küche: Abfragenoptimierung

SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation

Markus Stocker
HP Laboratories
Bristol
United Kingdom
markus.stocker@gmail.com

Andy Seaborne
HP Laboratories
Bristol
United Kingdom
andy.seaborne@hp.com

Abraham Bernstein
Department of Informatics
University of Zurich
Switzerland
bernstein@ifi.uzh.ch

Christoph Kiefer
Department of Informatics
University of Zurich
Switzerland
kiefer@ifi.uzh.ch

Dave Reynolds
HP Laboratories
Bristol
United Kingdom
dave.reynolds@hp.com

ABSTRACT

In this paper, we formalize the problem of *Basic Graph Pattern* (BGP) optimization for SPARQL queries and *main memory* graph implementations of RDF data. We define and analyze the characteristics of heuristics for selectivity-based static BGP optimization. The heuristics range from simple triple pattern variable counting to more sophisticated selectivity estimation techniques. Customized summary statistics for RDF data enable the selectivity estimation of *joined* triple patterns and the development of efficient heuristics. Using the Lehigh University Benchmark (LUBM), we evaluate the performance of the heuristics for the queries provided by the LUBM and discuss some of them in more details.

Query optimization is a fundamental and crucial subtask of query execution in database management systems. We focus on *static* query optimization, i.e. a join order optimization of triple patterns performed before query evaluation. The optimization goal is to find the execution plan which is expected to return the result set fastest without actually executing the query or subparts. This is typically solved by means of heuristics and summaries for statistics about the data.

The problem we are going to tackle in this paper is best explained by a simple example. Consider the BGP displayed in Listing 1 which represents a BGP of a SPARQL query executed over RDF data describing the university domain. Typically, there are a number of different subjects working, teaching, and studying at a university (e.g. staff members,

https://doi.org/10.1145/1367497.1367578

Abfragenoptimierung

- Die *triple patterns* werden der Reihe nach ausgeführt (naiv)
- Jedes ergibt eine Menge als Zwischenresultat
- Zudem gibt es auch *joins* zwischen *triple patterns*
- Daraus können sich sehr grosse Zwischenresultate ergeben
- Es macht Sinn, die selektivsten *triple patterns* zuerst auszuführen
- Sprich die die möglichst kleine Zwischenresultatsmengen ergeben
- So kann man die Abfrage optimieren
- Also die Abfragegeschwindigkeit potentiell drammatisch verbessern
- Die meisten Systeme implementieren heute solche Optimierungen

Zusammenfassung

- Zusätzlich zu `SELECT` gibt es auch `CONSTRUCT`
- Und `ASK` und `DESCRIBE` Abfragen (vielleicht weniger benutzt)
- Resultatsmengen können geordnet und limitiert werden
- SPARQL Update ist eine wichtige Erweiterung
- SPARQL Endpoints unterstützen Abfragen über das Internet (Web)