SPARQL 1.0

Irini Fundulaki
Institute of Computer Science
FORTH
Overview

1. Introduction to SPARQL
2. SPARQL Basics
3. Updating Linked Data with SPARQL 1.1
4. SPARQL Protocol
INTRODUCTION TO SPARQL
Parts of the SPARQL Specification

- **SPARQL Query Language** *(discussed here)*
  - declarative query language for RDF data
  - SPARQL 1.0: W3C Specification since January 2008
    - [http://www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/)
  - SPARQL 1.1: W3C Specification since March 2013
    - [http://www.w3.org/TR/sparql11-query/](http://www.w3.org/TR/sparql11-query/)

- **SPARQL Update Language** *(discussed here)*
  - Declarative manipulation language for RDF data
  - [http://www.w3.org/TR/sparql11-update/](http://www.w3.org/TR/sparql11-update/)

- **SPARQL Protocol** *(discussed here)*
  - Standard for issuing SPARQL queries and receiving results
  - Used for the communication between SPARQL services and clients
  - [http://www.w3.org/TR/sparql11-protocol/](http://www.w3.org/TR/sparql11-protocol/)

- **SPARQL Query Results XML Format** *(not discussed here)*
  - [http://www.w3.org/TR/rdf-sparql-XMLres/](http://www.w3.org/TR/rdf-sparql-XMLres/)
SPARQL 1.1

- **SPARQL 1.0** only allows accessing the data (**query**)
- **SPARQL 1.1** introduces:

  **Query extensions**
  - Aggregates, subqueries, negation, expressions in the SELECT clause, property paths, assignment, short form for CONSTRUCT, expanded set of functions and operators
  - **Data management**: Insert, Delete, Delete/Insert
  - **Graph management**: Create, Load, Clear, Drop, Copy, Move, Add

  **Updates**
  - Service, values, service variables (informative)
SPARQL BASICS
Data representation

• Data will be represented in the Turtle format
• Turtle also used in SPARQL queries (in the representation of triple pattern- *to see later*)

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>dbpedia:Help!_(album) dbpedia-owl:artist dbpedia:The_Beatles .</td>
</tr>
<tr>
<td>T2</td>
<td>dbpedia:Yellow_Submarine_(album) dbpedia-owl:artist dbpedia:The_Beatles .</td>
</tr>
<tr>
<td>T4</td>
<td>dbpedia:The_Beatles dbprop:alt “A square quartered into four head shots of young men with moptop haircuts. All four wear white shirts and dark coats.” .</td>
</tr>
</tbody>
</table>
SPARQL Basics

- **RDF triple**: Basic building block of the form \((subject, predicate, object)\)

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1: dbpedia:The_Beatles</td>
<td>foaf:name</td>
<td>&quot;The Beatles&quot;</td>
</tr>
<tr>
<td>T2: dbpedia:Help!_(album)</td>
<td>dbpedia-owl:artist</td>
<td>dbpedia:The_Beatles</td>
</tr>
<tr>
<td>T3: dbpedia:The_Beatles</td>
<td>dbpedia-owl:genre</td>
<td>dbpedia:Rock_music</td>
</tr>
</tbody>
</table>
SPARQL Basics

• SPARQL is based on Pattern Matching
  – Queries describe subgraphs of the queried graph
  – SPARQL graph patterns describe the subgraphs to match

• SPARQL Graph Patterns
  – Triple Pattern: triple with variables
  – Group Graph Patterns: composition of Group Graph Patterns using SPARQL operators
    • Basic Graph Patterns: sets of triple patterns
    • Filters: complex conditions/constraints
    • Union Graph Patterns
    • Optional Graph Patterns
  – Patterns on Named Graphs
• **Triple Pattern**
  – An RDF triple with **variables** in the place of *subject, predicate, object*
  – **Variables** are **prefixed** with “?”

| TP1 | ?album dbpedia-owl:artist dbpedia:The_Beatles |

– ?album, ?property, ?object are **variables**
SPARQL Basic Graph Pattern

- A Basic Graph Pattern (BGP) is a set of triple patterns
- A BGP is understood as a conjunction of the triple patterns it consists of

<table>
<thead>
<tr>
<th>Triple Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP₁</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Basic Graph Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGP₁</td>
</tr>
</tbody>
</table>
SPARQL Group Graph Pattern

- A group graph pattern: a set of graph patterns delimited by {}
SPARQL Group Graph Patterns

- Group Graph Patterns
  - built through **inductive construction** combining smaller patterns into more complex ones using SPARQL operators (*to define later*)
  - Variables in group graph patterns have a **global scope**
  - **Empty Group Graph Pattern:** special case of graph pattern denoted by `{ }`

**Note:** There is *no keyword* for **conjunction** in SPARQL. Basic graph patterns (BGPs) are simply juxtaposed and then enclosed in “{" and “}” to form a group graph pattern.
SPARQL Query Results

- **Subgraphs** of the data that **match** the SPARQL Query
- **Matching** based on **variable bindings/mappings**
- **Binding/Mapping:**
  - substitution of query variables with RDF terms
  - perceived as **partial function** from variables to RDF terms
  - represents one possible way of variables substitution
  - **not all variables need to be bound in a mapping**
  - tabular representation: each row represents one mapping

<table>
<thead>
<tr>
<th>Mapping</th>
<th>?v1</th>
<th>?v2</th>
<th>?v3</th>
<th>....</th>
<th>?vk</th>
</tr>
</thead>
<tbody>
<tr>
<td>μ₁</td>
<td>val₁</td>
<td>val₂</td>
<td>val₃</td>
<td>....</td>
<td>valₖ</td>
</tr>
</tbody>
</table>

**Solution: bags of variable bindings**

<table>
<thead>
<tr>
<th>Mapping</th>
<th>?v1</th>
<th>?v2</th>
<th>?v3</th>
<th>....</th>
<th>?vk</th>
</tr>
</thead>
<tbody>
<tr>
<td>μ₁</td>
<td>v₁</td>
<td>v₂</td>
<td>v₃</td>
<td>....</td>
<td>vₖ</td>
</tr>
<tr>
<td>μ₂</td>
<td>u₁</td>
<td>u₂</td>
<td>u₃</td>
<td>....</td>
<td>um</td>
</tr>
</tbody>
</table>
Matching Triple Patterns (1)

- Result: the *triples* that match the *variables in the triple pattern*

<table>
<thead>
<tr>
<th>Triple Pattern</th>
<th>Mapping</th>
<th>?work</th>
<th>?artist</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP₁:</td>
<td>μ₁</td>
<td>dbpedia:Good_Day_Sunshine</td>
<td>dbpedia:The_Beatles</td>
</tr>
<tr>
<td></td>
<td>μ₂</td>
<td>dbpedia:Help!_(album)</td>
<td>dbpedia:The_Beatles</td>
</tr>
<tr>
<td></td>
<td>μ₃</td>
<td>dbpedia:Yellow_Submarine_(album)</td>
<td>dbpedia:The_Beatles</td>
</tr>
</tbody>
</table>

RDF Triples:
- T₅: dbpedia:The_Beatles dbprop:alt “A square quartered into four head shots of young men with moptop haircuts. All four wear white shirts and dark coats.”.
Matching Basic Graph Patterns (2)

• Result:
  – *all the triple patterns should match* (conjunction)
  – *all variables must be bound*

**BGP1:**

```
  ?artist dbpedia-owl:genre ?genre . }
```

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>T1</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>T2</strong></td>
<td></td>
</tr>
</tbody>
</table>

|     | dbpedia:The_Beatles dbprop:alt “A square quartered into four head
       shots of young men with moptop haircuts. All four wear white
       shirts and dark coats.” . |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>T4</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>T3</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mappings</th>
<th>?artist</th>
<th>?genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$</td>
<td>dbpedia:The_Beatles</td>
<td>dbpedia:Pop_music</td>
</tr>
</tbody>
</table>
Matching Basic Graph Patterns (3)

BGP1:  
  ?artist dbpedia-owl:genre ?genre . }

| T1   | dbpedia:Good_Day_Sunshine dbpedia-owl:artist dbpedia:The_Beatles . |
| T3   | dbpedia:Yellow_Submarine_(album) dbpedia-owl:artist dbpedia:The_Beatles . |

<table>
<thead>
<tr>
<th>Mappings</th>
<th>?artist</th>
<th>?genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>μ₁</td>
<td>dbpedia:The_Beatles</td>
<td>dbpedia:Pop_music</td>
</tr>
<tr>
<td>μ₂</td>
<td>dbpedia:The_Beatles</td>
<td>dbpedia:Rock_music</td>
</tr>
</tbody>
</table>

Irini Fundulaki, ESWC 2014 Summer School
Filters

• *Impose constraints* on group graph patterns not only on variables

• Constraints involve:
  
  – *Arithmetic operators* (+, -, *, /)
  
  – *RDF specific functions* (isLiteral(), Lang(), Bound(),…)
  
  – *Comparisons* (=, <, >, <=, >=, !=)
  
  – *Logical Connectives*: negation (!), disjunction (||), conjunction (&&)

• Expressed using the **FILTER** keyword

<table>
<thead>
<tr>
<th>F1.</th>
<th>FILTER (isLiteral(?name))</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2.</td>
<td>FILTER (?name = “The Beatles” )</td>
</tr>
<tr>
<td>F3.</td>
<td>FILTER (?name = “The Beatles</td>
</tr>
<tr>
<td>F4.</td>
<td>FILTER(REGEX(?name, “<em>beat</em>”, i))</td>
</tr>
<tr>
<td>F5.</td>
<td>FILTER ( ?runtime&gt; 1720</td>
</tr>
</tbody>
</table>

• **Matching**: compute solutions and then **filter out those** that do not satisfy the constraint
Matching Filters

- A **FILTER** condition is a *restriction on solutions over the whole group in which the filter appears*.
- A group graph pattern can have **multiple filters**. These *can be rewritten as a single filter with conjoined filter conditions*.

FILTER(?name="The Beatles") .
FILTER (?runtime > 1720 ||
  ?runtime < 1200) .

---

**Equivalent Graph Patterns**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>F1.</strong></td>
<td><code>?artist dbpprop:name ?name . FILTER (?name = “The Beatles”)</code></td>
</tr>
<tr>
<td><strong>F2.</strong></td>
<td><code>?artist dbpprop:name “The Beatles” .</code></td>
</tr>
</tbody>
</table>
Filter Expressions

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Function</th>
<th>Result type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Forms</td>
<td>bound</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td></td>
<td>IF</td>
<td>rdfTerm</td>
</tr>
<tr>
<td></td>
<td>COALESCE</td>
<td>rdfTerm</td>
</tr>
<tr>
<td></td>
<td>NOT EXISTS, EXISTS</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td></td>
<td>or, and</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td></td>
<td>RDFTerm-equal (=), sameTerm</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td></td>
<td>IN, NOT IN</td>
<td>boolean</td>
</tr>
<tr>
<td>Functions on RDF Terms</td>
<td>isIRI, isBlank, isLiteral, isNumeric</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td></td>
<td>str, lang, datatype</td>
<td>simple literal</td>
</tr>
<tr>
<td></td>
<td>IRI</td>
<td>iri</td>
</tr>
<tr>
<td></td>
<td>BNODE</td>
<td>iri</td>
</tr>
<tr>
<td>Functions on Numerics</td>
<td>ABS, ROUND, CEIL, FLOOR, RAND</td>
<td>numeric</td>
</tr>
<tr>
<td></td>
<td>RAND</td>
<td>xsd:double</td>
</tr>
</tbody>
</table>

Source: http://www.w3.org/TR/sparql11-query/#SparqlOps

Irini Fundulaki, ESWC 2014 Summer School
# Filter Expressions

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Function</th>
<th>Result type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions on Strings</td>
<td>STRLEN</td>
<td>xsd:integer</td>
</tr>
<tr>
<td></td>
<td>SUBSTR, UCASE, LCASE</td>
<td>string literal</td>
</tr>
<tr>
<td></td>
<td>STRSTARTS, STREND, CONTAINS</td>
<td>literal</td>
</tr>
<tr>
<td></td>
<td>STRBEFORE, STRAFTER</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td></td>
<td>ENCODE_FOR_URI</td>
<td>string literal</td>
</tr>
<tr>
<td></td>
<td>CONCAT</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td></td>
<td>langMatches</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td></td>
<td>REGEX</td>
<td>string literal</td>
</tr>
<tr>
<td></td>
<td>REPLACE</td>
<td></td>
</tr>
<tr>
<td>Functions on Dates and Times</td>
<td>now</td>
<td>xsd:dateTime</td>
</tr>
<tr>
<td></td>
<td>year, month, day, hours, minutes</td>
<td>xsd:integer</td>
</tr>
<tr>
<td></td>
<td>seconds</td>
<td>xsd:decimal</td>
</tr>
<tr>
<td></td>
<td>timezone</td>
<td>xsd:dayTimeDuration</td>
</tr>
<tr>
<td></td>
<td>tz</td>
<td>simple literal</td>
</tr>
</tbody>
</table>

Source: [http://www.w3.org/TR/sparql11-query/#SparqlOps](http://www.w3.org/TR/sparql11-query/#SparqlOps)
Matching String Filters (1)

F1:  
?artist dbpprop:name ?name .  
FILTER(?name="The Beatles") }  

T1  dbpedia:Good_Day_Sunshine dbpedia-owl:artist dbpedia:The_Beatles.  
T3  dbpedia:Rain_Dogs dbpedia-owl:artist dbpedia:Tom_Waits  
T4  dbpedia:Tom_Waits dbpprop:name “Tom Waits”.  
T5  dbpedia:The_Beatles dbpprop:name “The Beatles”.  

M | ?album | ?artist | ?name  
---|--------|--------|-------  
μ₁ | dbpedia:Help!_(album) | dbpedia:The_Beatles | “The Beatles”  
μ₂ | dbpedia:Good_Day_Sunshine | dbpedia:The_Beatles | “The Beatles”  
μ₃ | dbpedia:Rain_Dogs | dbpedia:Tom_Waits | “Tom Waits”  

Matching Arithmetic Filters (2)

F1: \[
\{ ?album \text{ dbpedia-owl:runtime } ?\text{runtime} . \\
\text{FILTER(?runtime > 1720 || ?runtime < 1200)} \}
\]

<table>
<thead>
<tr>
<th>M</th>
<th>?album</th>
<th>?runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mu_1)</td>
<td>dbpedia:Help!_(album)</td>
<td>1723.000</td>
</tr>
<tr>
<td>(\mu_2)</td>
<td>dbpedia:Help!_(album)</td>
<td>2060.000</td>
</tr>
<tr>
<td>(\mu_3)</td>
<td>dbpedia:Good_Day_Sunshine</td>
<td>1290.000</td>
</tr>
</tbody>
</table>

T1 dbpedia:Good_Day_Sunshine dbpedia-owl:runtime 1290.000 .

T2 dbpedia:Help!_(album) dbpedia-owl:runtime 1723.000 .

T3 dbpedia:Help!_(album) dbpedia-owl:runtime 2060.000 .
Union Graph Patterns

- **What:** allow the forming of *disjunction of* graph patterns
- **Why:** information is expressed in different ways in an RDF graph
- **How:** Expressed using the **UNION** keyword

\[
\text{UGP}_1: \quad \{ \{ \text{?artist dbpedia-owl:genre } \text{?genre} \} \text{ UNION } \{ \text{?artist mo:kind } \text{?genre} \} \}
\]

- **Matching:**
  - More than one patterns may match **match**
  - If more than one patterns match, then *all possible solutions are returned*
  - Patterns are matched independently and results are combined using **set theoretic union**
  - *Patterns do not necessarily use the same variables!*

Irini Fundulaki, ESWC 2014 Summer School
Matching Union Graph Patterns (1)

UGP1: \[
\{ \{ \text{?artist} \text{ dbpedia-owl:genre } \text{dbpedia:Pop_music} \} \ \text{UNION} \ \\
\{ \text{?artist} \text{ dbpedia-owl:genre } \text{dbpedia:Rock_music} \} \}
\]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>dbpedia:The_Beatles dbpedia-owl:genre dbpedia:Pop_music .</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mappings</th>
<th>?artist</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mu_1)</td>
<td>dbpedia:The_Beatles</td>
</tr>
<tr>
<td>(\mu_2)</td>
<td>dbpedia:Tom_Waits</td>
</tr>
<tr>
<td>(\mu_3)</td>
<td>dbpedia:The_Beatles</td>
</tr>
</tbody>
</table>
Matching Union Graph Patterns (2)


<table>
<thead>
<tr>
<th>M</th>
<th>?album</th>
<th>?artist</th>
<th>?genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>μ₁</td>
<td>dbpedia:Help!_(album)</td>
<td>dbpedia:The_Beatles</td>
<td></td>
</tr>
<tr>
<td>μ₂</td>
<td></td>
<td>dbpedia:The_Beatles</td>
<td>dbpedia:Pop_music</td>
</tr>
<tr>
<td>μ₃</td>
<td></td>
<td>dbpedia:Tom_Waits</td>
<td>dbpedia:Rock_music</td>
</tr>
</tbody>
</table>
Matching Union Graph Patterns (3)

- Use of variables in the alternatives depends on what we want to compute

UGP₁: \[
\{ \{ \text{?album} \text{ dbpedia-owl:artist } \text{?artist1} \} \text{ UNION } \\
\{ \text{?artist2} \text{ dbpedia-owl:genre } \text{?genre} \} \} \]

T₁ \text{ dbpedia:The_Beatles dbpedia-owl:genre } \text{ dbpedia:Pop_music} .

T₂ \text{ dbpedia:Tom_Waits dbpedia-owl:genre } \text{ dbpedia:Rock_music} .

T₃ \text{ dbpedia:Help!_(album) dbpedia-owl:artist } \text{ dbpedia:The_Beatles} .

\[\begin{array}{|c|c|c|c|}
\hline
\text{M} & \text{?album} & \text{?artist1} & \text{?artist2} & \text{?genre} \\
\hline
\mu₁ & \text{ dbpedia:Help!_(album)} & \text{ dbpedia:The_Beatles} & \text{} & \text{} \\
\hline
\mu₂ & \text{} & \text{ dbpedia:The_Beatles} & \text{ dbpedia:Pop_music} & \text{} \\
\hline
\mu₃ & \text{} & \text{ dbpedia:Tom_Waits} & \text{ dbpedia:Rock_music} & \text{} \\
\hline
\end{array}\]
Semantics of Union Graph Patterns

- Union Graph Pattern UGP₁ returns the same result as the set-theoretic union of the results of Graph Patterns GP₁ and GP₂

<table>
<thead>
<tr>
<th>UGP₁:</th>
<th>{ { ?artist dbpedia-owl:genre dbpedia:Pop_music } UNION { ?artist dbpedia-owl:genre dbpedia:Rock_music } }</th>
</tr>
</thead>
<tbody>
<tr>
<td>GP₁:</td>
<td>{ ?artist dbpedia-owl:genre dbpedia:Pop_music . }</td>
</tr>
<tr>
<td>GP₂:</td>
<td>{ ?artist dbpedia-owl:genre dbpedia:Rock_music . }</td>
</tr>
</tbody>
</table>
Properties of UNION

- binary operator
- commutative operator
  - \( \text{GP1 UNION GP2} \) equivalent to \( \text{GP2 UNION GP1} \)
- left-associative
- has higher precedence than conjunction
- \( \{ \text{GP1 UNION GP2} \} \) UNION \( \text{GP3} \) equivalent to
  - \( \text{GP1 UNION \{ GP2 UNION GP3 \}} \)
Optional Graph Patterns

• **What:** allows the *specification of optional parts* of a query

• **Why:** *regular and/or complete information* cannot be assumed in all RDF graphs

• **How:** Expressed using the **OPTIONAL** keyword
  
  – **GP₁** OPTIONAL { **GP₂** }

<table>
<thead>
<tr>
<th>OGP₁:</th>
<th>{ ?album dbpedia-owl:artist ?artist }</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>OPTIONAL</strong></td>
</tr>
<tr>
<td></td>
<td>{ ?album foaf:img ?img }</td>
</tr>
</tbody>
</table>

• **Matching:**
  
  – If the optional part is not matched, no solutions for this part are created, but the solutions computed are not eliminated
### Optional Graph Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Query</th>
</tr>
</thead>
</table>
| **OGP₁:** | `{ { ?album dbpedia-owl:artist ?artist } .  
  OPTIONAL { ?artist dbpprop:birthDate ?date. } }` |
| **GP₁:** | `{ ?album dbpedia-owl:artist ?artist .  
  ?artist dbpprop:birthDate ?date. FILTER (!BOUND(?date)) }` |
| **GP₂** | `{ ?album dbpedia-owl:artist ?artist .  
  OPTIONAL { ?artist dbpprop:birthDate ?date.  
  FILTER (!BOUND(?date)) } }` |

**OGP₁:** returns all mappings independently whether ?date is bound to some value.

**GP₁:** returns all mappings where ?date is not bound.
### Matching Optional Graph Patterns (1)

OGP1:

\[
\{ \text{?album } \text{dbpedia-owl:artist } \text{?artist} \}
\]

**OPTIONAL**

\[
\{ \text{?album } \text{foaf:img } \text{?img} \}
\]

<table>
<thead>
<tr>
<th>T1</th>
<th>dbpedia:Help!_(album) dbpedia-owl:artist dbpedia:The_Beatles .</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Mappings</th>
<th>?album</th>
<th>?artist</th>
<th>?img</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mu_1)</td>
<td>dbpedia:Help!_(album)</td>
<td>dbpedia:The_Beatles</td>
<td></td>
</tr>
</tbody>
</table>
Properties of **OPTIONAL**

- binary operator
- left associative

\[
\text{GP1} \ \text{OPTIONAL} \ \{ \ \text{GP2} \ \} \ \text{OPTIONAL} \ \{ \ \text{GP3} \ \}
\]
equivalent to

\[
\{ \text{GP1} \ \text{OPTIONAL} \ \{ \ \text{GP2} \ \} \} \ \text{OPTIONAL} \ \{ \ \text{GP3} \ \}
\]

– But

\[
\{ \text{GP1} \ \text{OPTIONAL} \ \text{GP2} \} \ \text{OPTIONAL} \ \text{GP3}
\]
not equivalent to

\[
\text{GP1} \ \text{OPTIONAL} \ \{ \ \text{GP2} \ \text{OPTIONAL} \ \text{GP3} \ \}
\]

- not commutative
- \{ \ \text{OPTIONAL} \ \{ \ \text{GP1} \ \} \ \} \\
  equivalent to

\[
\{ \ \text{\{} \ \text{OPTIONAL} \ \{ \ \text{GP1} \ \} \ \}
\]

- higher precedence than conjunction
### OPTIONALs and FILTERs


| T1   | dbpedia:Good_Day_Sunshine dbpedia-owl:runtime 1000.000 |
| T2   | dbpedia:Help!_(album) dbpedia-owl:runtime 1723.000 |
| T3   | dbpedia:Help!_(album) dbpedia-owl:artist dbpedia:The_Beatles |
| T4   | dbpedia:Rain_Dogs dbpedia-owl:artist dbpedia:Tom_Waits |
| T4   | dbpedia:Good_Day_Sunshine dbpedia-owl:artist dbpedia:The_Beatles |

<table>
<thead>
<tr>
<th>Mapping</th>
<th>?album</th>
<th>?artist</th>
<th>?runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>μ₁</td>
<td>dbpedia:Help!_(album)</td>
<td>dbpedia:The_Beatles</td>
<td>1723.000</td>
</tr>
<tr>
<td>μ₂</td>
<td>dbpedia:Rain_Dogs</td>
<td>dbpedia:Tom_Waits</td>
<td></td>
</tr>
</tbody>
</table>

The OPTIONAL pattern **does not generate any bindings** when either
- **there is no** dbpedia-owl:runtime property (dbpedia:RainDogs)
- **there is a** dbpedia-owl:runtime property **but the constraint is not satisfied** (dbpedia:Good_Day_Sunshine)
Multiple OPTIONALs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mu_1)</td>
<td><code>dbpedia:Help!_(album)</code></td>
<td><code>dbpedia:The_Beatles</code></td>
<td>1723.000</td>
<td><img src="http://upload.wikimedia.org/wikipedia/commons/4/46TomWaitsRainDogs.jpg" alt="TomWaitsRainDogs.jpg" /></td>
</tr>
<tr>
<td>(\mu_2)</td>
<td><code>dbpedia:Rain_Dogs</code></td>
<td><code>dbpedia:Tom_Waits</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\mu_3)</td>
<td><code>dbpedia:Bone_Machine</code></td>
<td><code>dbpedia:Tom_Waits</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Combining UNION and OPTIONAL

• What does the following mean?

```{ ?album dbpedia-owl:releaseDate "01-01-1960" .
{ ?album dbpedia-owl:artist ?artist. } UNION
{ ?album dbpedia-ont:leadSinger ?artist. } OPTIONAL
{ ?artist foaf:name ?name . }
}
```

1. UNION of two patterns with an OPTIONAL condition
2. UNION of two patterns where the 2\textsuperscript{nd} has an OPTIONAL condition

Solution (1):

```{ ?album dbpedia-owl:releaseDate "01-01-1960" .
{ { ?album dbpedia-owl:artist ?artist. } UNION
{ ?album dbpedia-ont:leadSinger ?artist. } } }
OPTIONAL
{ ?artist foaf:name ?name . }
}```
Combining UNION and OPTIONAL

**General Rules:**

1. OPTIONAL refers exactly to one grouped pattern to the right.
2. OPTIONAL and UNION refer to all expressions to their left since both operators are left associative.
3. UNION and OPTIONAL have the same precedence.

\[
\begin{align*}
\{\{s1 \ p1 \ o1\} \ UNION \ \{s2 \ p2 \ o1\} \ OPTIONAL \ \{s3 \ p3 \ o3\}\}
\end{align*}
\]

*Equivalent to*

\[
\begin{align*}
\{\{s1 \ p1 \ o1\} \ UNION \ \{s2 \ p2 \ o1\}\} \ OPTIONAL \ \{s3 \ p3 \ o3\}
\end{align*}
\]
Combining UNION and Conjunction

\[
\{\{s_1 \ p_1 \ o_1\} \ \text{UNION} \ \{s_2 \ p_2 \ o_1\} \ \{s_3 \ p_3 \ o_3\}\}
\]

Equivalent to

\[
\{\{\{s_1 \ p_1 \ o_1\} \ \text{UNION} \ \{s_2 \ p_2 \ o_1\}\} \ \{s_3 \ p_3 \ o_3\}\}
\]

Irini Fundulaki, ESWC 2014 Summer School
SPARQL Query Components: PROLOGUE

@prefix dbpedia: <http://dbpedia.org/resource/>
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix dppedia-owl: <http://dbpedia.org/resource/>

SELECT ?artist, ?genre
FROM <http://dbpedia.org>
WHERE { dbpedia:Help!_(album) dppedia-owl:artist ?artist.
   ?artist foaf:name ?name }
ORDER BY ?name

• PREFIX
  – Definition of prefix labels for URIs
  – Syntax: Subtly different from Turtle syntax – no period (".") character used as separator

Irini Fundulaki, ESWC 2014 Summer School
SPARQL Query Components: QUERY FORMS

@prefix dbpedia: <http://dbpedia.org/resource/>
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix dppedia-owl: <http://dbpedia.org/resource/>

SELECT ?artist, ?genre
FROM <http://dbpedia.org>
  ?artist foaf:name ?name }
ORDER BY ?name

- Query Forms
  - SELECT, ASK, DESCRIBE, CONSTRUCT
    (to explain in detail later)
SPARQL Query Components: DATASET SPECIFICATION

@prefix dbpedia: <http://dbpedia.org/resource/>
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix dppedia-owl: <http://dbpedia.org/resource/>

SELECT ?artist, ?genre
FROM <http://dbpedia.org>
  ?artist foaf:name ?name }
ORDER BY ?name

• **FROM** or **FROM NAMED**
  – *Optional clause*
  – Specify the data sources against which the queries will be evaluated
  – One default graph (**FROM** clause)
  – Zero or more named graphs (**FROM NAMED** clause)
SPARQL Query Components: QUERY PATTERN

@prefix dbpedia: <http://dbpedia.org/resource/>
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix dppedia-owl: <http://dbpedia.org/resource/>

SELECT ?artist, ?genre
FROM <http://dbpedia.org>
WHERE { dbpedia:Help!_(album) dppedia-owl:artist ?artist.
  ?artist foaf:name ?name }
ORDER BY ?name

- WHERE
  - Defines the *patterns* to match against the data
  - Computation of the mappings/bindings
SPARQL Query Components: SOLUTION MODIFIERs

@prefix dbpedia: http://dbpedia.org/resource/
@prefix dbpprop: http://dbpedia.org/property

SELECT ?artist, ?genre
FROM <http://dbpedia.org>
WHERE {
  ?artist foaf:name ?name
}
ORDER BY ?name

• Solution Modifiers are used to modify the result set
  – ORDER BY, LIMIT, OFFSET to reorganize the results
  – GROUP BY to combine the results
  – DISTINCT to eliminate duplicates
**SPARQL Query Forms**

SPARQL supports different query forms:

- **SELECT** returns *ordered multi-set of variable bindings*

- **ASK** checks *whether a graph pattern has at least one solution*; returns a *Boolean* value (true/false)

- **CONSTRUCT** *returns a new RDF graph* as specified by the graph template using the computed bindings from the query’s **WHERE** clause

- **DESCRIBE** *returns the RDF graph* containing the RDF data about the requested resource
Query Form: SELECT (1)

- The variables in the SELECT clause are returned along with their values according to the computed bindings
- Similar to the SELECT clause in SQL queries
- “*” means all components should be returned

```
Return the artists that made album identified by URI dbpedia:Help!_(album) and the artist’s music kind
```

```
SELECT ?artist, ?genre
WHERE
```
Return the name of the albums and tracks of albums that “The Beatles” made.

@prefix dbpedia: <http://dbpedia.org/resource/>
@prefix dbpprop: <http://dbpedia.org/property/>

SELECT ?album_name, ?track_name
WHERE {
  ?artist dbpprop:name "The Beatles" .
  ?album dbpprop:name ?album_name .
}
Query Form: SELECT (3)

Return the name of the albums of “The Beatles” with runtimes between 5400 and 3600 seconds and the tracks of albums whose runtimes are more than 900 seconds.

@prefix dbpedia: <http://dbpedia.org/resource/>
@prefix dbpprop: <http://dbpedia.org/property/>

SELECT ?album_name, ?track_name
  ?album dbpprop:name ?album_name .
  FILTER (?album dbpedia-owl:runtime > 3600 &&
             ?album dbpedia-owl:runtime < 5400 &&
             ?track dbpedia-owl:runtime > 900 )}
Query Form: SELECT (4) Elimination of Duplicates

Return the name of the albums of The Beatles that have at least two different tracks.

```
SELECT SOLUTION_MODIFIER ?name
WHERE {
  ?artist dbpprop:name "The Beatles" .
  ?album dbpprop:name ?album_name .
  FILTER (?track1!=?track2)
}
```

RESULTS:

REDUCED

DISTINCT

```
?album_name

“Revolver”
“Revolver”
“Revolver”
“Revolver”
“Sessions”
“Abbey Road”
“Abbey Road”
```

Irini Fundulaki, ESWC 2014 Summer School
AGGREGATES

• Aggregates specify expressions over *groups of solutions*

• Used when the result is computed over a *group of solutions rather than a single solution*
  
  – *example*: average value of a set of values a variable takes

• Aggregates defined in SPARQL 1.1 are `COUNT`, `SUM`, `MIN`, `MAX`, `AVG`, `GROUP_CONCAT`, and `SAMPLE`.

• Solutions are *grouped* using the `GROUP BY` clause

• *Pruning at group level* is performed with the `HAVING` clause
Query Form: SELECT (5) AGGREGATES

Return the number of albums that The Beatles have recorded.

SELECT count(?album) as ?albums

Return the number of tracks per album recorded by The Beatles

SELECT ?album (count(?track) as ?tracks)
Return the total duration of the albums recorded by The Beatles where the duration of all tracks per album is greater than 3600000 seconds

```
SELECT ?album (SUM(?track_duration) AS ?album_duration)
WHERE {
  ?artist dbpprop:name "The Beatles" .
}
GROUP BY ?album
HAVING (SUM(?track_duration) > 3600000))
```
Query Form: SELECT (7) SOLUTION MODIFIERS

- ORDER BY: establishes the order of a solution sequence:
  - ASC (ascending) - default, DESC (descending)

Return the most recent albums of “The Beatles” with duration more than one hour. Return the albums with higher duration first.

```
SELECT ?album
WHERE
{?al dbpedia-owl:artist ?ar. ?ar dbpprop:name "The Beatles".
 ?al dbpedia-owl:releaseDate ?date.?al dbpedia-owl:runtime ?rt
FILTER(?rt > 3600)}
ORDER BY DESC(?date)
```
Query Form: SELECT (8)  
SOLUTION MODIFIERS

- LIMIT: specifies the number of solutions to be returned
  - It should always be preceded by the ORDER BY modifier

Return the 10 most recent albums made by “The Beatles” that last more than one hour.

SELECT ?album  
WHERE  
{  
  ?artist dbpprop:name “The Beatles”.  
  ?album dbpedia-owl:releaseDate ?date .  
  ?album dbpedia-owl:runtime ?runtime  
  FILTER(?runtime > 3600)}  
ORDER BY DESC(?date)  
LIMIT 10
Query Form: SELECT (9)

SOLUTION MODIFIERS

• OFFSET: causes the solutions generated to start after the specified number of solutions
  – index of the first reported item in the sequence

Return the 5 most recent albums made by “The Beatles” with duration higher than one hour, starting from the tenth most recent album

SELECT ?album
WHERE

  ?artist dbpprop:name “The Beatles”.
  ?album dbpedia-owl:releaseDate ?date .
  ?album dbpedia-owl:runtime ?rt . FILTER(?rt > 3600)}
ORDER BY DESC(?date)
LIMIT 5
OFFSET 10
Query Form: DESCRIBE

• Takes as input a resource and returns the RDF graph containing information about the resource
• A resource is identified
  1. Through *explicit IRIs*
  2. Through *bindings of variables* in the WHERE clause

Return all information about “The Beatles”

(1) DESCRIBE dbpedia:The_Beatles

(2) DESCRIBE ?group
WHERE {
  ?group dbpedia:name "The_Beatles" .}

Possibly different results

Irini Fundulaki, ESWC 2014 Summer School
Query Form: ASK

• Takes as input a graph pattern and returns true if there exist at least one solution and false otherwise.

Is Paul McCartney a member of “The Beatles”?

ASK
WHERE
{?artist foaf:name “Paul Mc Cartney” .
?group dbpprop:name “The Beatles” . }

Is Elvis Presley a member of ‘The Beatles’?

ASK
WHERE
{?artist foaf:name “Elvis Presley” .
?artist dbpedia-owl:associatedBand dbpedia:The_Beatles }.
Query Form: Construct

CONSTRUCT GP1
WHERE { GP2 }
......

- Returns a new graph specified by an optional pattern
- If the pattern is missing, then the graph pattern in the WHERE clause is considered
- Pattern uses variables from the query’s WHERE clause
- Graph is obtained from the variable bindings in the query’s WHERE clause
Return a graph containing all the information about “The Beatles” except type information

CONSTRUCT
FILTER(?property!=rdf:type)}

T1  dbpedia:The_Beatles rdf:type dbpedia-owl:Organisation.
T2  dbpedia:The_Beatles rdf:type dbpedia-owl:Band.
T4  dbpedia:The_Beatles dbpedia-owl:hometown dbpedia:Liverpool.
T5  dbpedia:The_Beatles dbprop:name “The Beatles”.

Result:
C1  dbpedia:The_Beatles dbpedia-owl:genre dbpedia:Pop_music.
C2  dbpedia:The_Beatles dbpedia-owl:hometown dbpedia:Liverpool.
C3  dbpedia:The_Beatles dbprop:name “The Beatles”
Query Form: Construct (2)

```
CONSTRUCT {?track dc:creator ?artist_name}
```

<table>
<thead>
<tr>
<th>RDF Triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
</tr>
<tr>
<td>T2</td>
</tr>
<tr>
<td>T3</td>
</tr>
<tr>
<td>T4</td>
</tr>
<tr>
<td>T5</td>
</tr>
<tr>
<td>T6</td>
</tr>
<tr>
<td>T7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>C4</td>
</tr>
</tbody>
</table>
Matching RDF Literals

• **Language Tags**
  – “Athens”@en
  – “Αθήνα”@gr
  – “Atene”@it

• Queries Q1, Q2 have different results

• **Typed Literals**
  1 = “1”^^xsd:integer
  1.5 = “1.5”^^xsd:decimal
  1.0E6 = “1”^^xsd:double
  true = “true”^^xsd:boolean

Q1: SELECT ?v WHERE {?v ?p “Athens”@en}
Q2: SELECT ?v WHERE {?v ?p “Atene”@it}

• The @en language tag means that the string is an English word and will match differently than any other language tag (similarly for the other language tags).
Assigning Variables

- The *value of an expression* can be added to a solution mapping by *binding a new variable* (which can be further used and returned)

*Calculate the duration of the tracks from ms to s, and store the value using the dbpedia-ont:runtime property.*

CONSTRUCT { ?track dbpedia-ont:runtime ?secs .} WHERE {
  dbpedia:The_Beatles foaf:made ?album .
  BIND((?duration/1000) AS ?secs) .
}
Sub-queries & Aggregate Values

• To combine the CONSTRUCT query form with aggregate values, a sub-query should be created inside the WHERE clause

CONSTRUCT {?album music-ont:duration ?album_duration .}
WHERE {
    SELECT ?album (SUM(?track_duration) AS ?album_duration)
    {
        dbpedia:The_Beatles foaf:made ?album .
    }
    GROUP BY ?album
    HAVING (SUM(?track_duration) > 3600000)}
UPDATING LINKED DATA WITH SPARQL 1.1
Data Management

SPARQL 1.1 provides data update operations:

• **INSERT data:** adds some triples, given inline in the request, into a graph

• **DELETE data:** removes some triples, given inline in the request, if the respective graphs contain those

• **DELETE/INSERT data:** uses in parallel INSERT and DELETE
Data Management

• INSERT

Insert the following triples into the graph <http://musicbrainz.org/20130302>

```
INSERT DATA { GRAPH { <http://musicbrainz.org/20130302>
    <http://musicbrainz.org/artist/a1> foaf:made
    <http://musicbrainz.org/release/r1> ,

    <http://musicbrainz.org/release/r1> dc:title "r1’s title" .
    <http://musicbrainz.org/release/r2> dc:title "r2’s title" } }
```
Data Management

**DELETE**

*Delete all the information about the album “Casualties” of “The Beatles.”*

DELETE { ?album ?predicate ?object . }
WHERE {
  ?album dbpprop:name "Casualties".
  ?album ?predicate ?object . }
Data Management

- DELETE/INSERT data

*Delete* the status of “Peter Best” as current member of “The Beatles”, and *insert* his status as former member of the band in graph <http://musicbrainz.org/20130302>

```
DELETE {  dbpedia:The Beetles db-ont:currentMember ?x . }

INSERT {  GRAPH <http://musicbrainz.org/20130302>
             {  dbpedia:The Beetles dbpedia-owl:pastMembers ?x .} }

WHERE {  dbpedia:The Beetles db-ont:currentMember ?member .
             ?member foaf:name "Peter Best" .}
```
SPARQL 1.1 provides graph update operations:

- **CREATE**: creates an empty graph in the Graph Store
- **LOAD**: reads the content of a document into a graph in the Graph Store
- **CLEAR**: removes all triples in one or more graphs
- **DROP**: removes the graph from the Graph Store
- **Other operations**: COPY, MOVE, ADD
Graph Management (2)

CREATE

• Creates a new named graph

CREATE GRAPH <http://musicbrainz.org/20130302>

LOAD

• An RDF graph can be loaded from a URL

LOAD <http://xmlns.com/foaf/spec/20100809.rdf>

LOAD <http://xmlns.com/foaf/spec/20100809.rdf>
INTO <http://xmlns.com/foaf/0.1/>

Named graph
Graph Management (3)

CLEAR

• Removes all triples in the graph (it is emptied but not deleted!)

• The graph(s) can be specified with the following keywords: DEFAULT, NAMED, ALL, GRAPH

DROP

• The given graph is removed from the Graph Store, including its content

• Can be used with the DEFAULT and ALL keywords
SPARQL 1.1 provides other graph management operations:

- **COPY ... TO ...**

  COPY GRAPH <http://musicbrainz.org/20130302>
  TO GRAPH <http://musicbrainz.org/20130303>

- **MOVE ... TO ...**

  MOVE GRAPH <http://musicbrainz.org/temp>
  TO GRAPH <http://musicbrainz.org/20130303>

- **ADD ... TO ...**

  ADD GRAPH <http://musicbrainz.org/20130302>
  TO GRAPH <http://musicbrainz.org/20130303>
SPARQL PROTOCOL FOR RDF
SPARQL 1.1. Protocol

- Consists of two **operations**: query and update

- An operation defines:
  - The HTTP method (GET or POST)
  - The HTTP query string parameters
  - The message content included in the HTTP request body
  - The message content included in the HTTP response body

---

Irini Fundulaki, ESWC 2014 Summer School
Query Operation

• Sends a SPARQL query to a service and receives the results of the query

• The response is:
  - XML, JSON, CSV/TSV from a **SELECT** query
  - RDF/XML, Turtle from a **CONSTRUCT** query

• May be invoked using **HTTP GET** or **HTTP POST**. The method POST with URL encoding is mostly used when the query string is too long

Irini Fundulaki, ESWC 2014 Summer School
SPARQL Query:

PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-ont: <http://dbpedia.org/ontology/>
SELECT ?album
WHERE { ?album dbpedia-owl:artist dbpedia:The_Beatles .}
LIMIT 3

HTTP GET request:

```
GET /sparql?default-graph-uri=http%3A%2F%2Fdbpedia.org&query=PREFIX+dbpedia%3A+3Http
%3A%2F%2Fdbpedia.org%2Fresource%2F%3E%0D%0A PREFIX+dbpedia-ont%3A+3Http%3A%2F%2Fdbpedia
.org%2Fontology%2F%3E%0D%0A SELECT+3Falbum+0D%0A WHERE+7B+%3Falbum+dbpedia-ont%3Aart
ist+dbpedia%3A+3AThe_Beatles+7D+%0D%0A LIMIT+3%0D%0A&format=application%2F%0D%0A sparql-results%2Bxml&timeout=0&debug=on HTTP/1.1
Host: dbpedia.org
```

Try this query!  
(Click on the hurl icon)
Update Operation

• Sends a **SPARQL update** request to a service

• Should be invoked using the **HTTP PATCH/POST** method

• The response consists of a HTTP response status code, which indicates success or failure of the operation
Acknowledgements

• **Content from**
  – Knowledge Technologies Course, University of Athens, Greece
    • Available at http://cgi.di.uoa.gr/~pms509/2012-2013/lectures/SPARQL%20-%20lecture%201.pdf
  – ESWC Summer School 2013 Slideshare
  – SPARQL, Martin Svoboda
    • Available at: http://www.ksi.mff.cuni.cz/~svoboda/teaching/2011-S1-NSWL144/seminar/05-sparql-1.pdf
  – W3C Candidate Recommendation “SPARQL Query Language for RDF”
    • Available at: http://www.w3.org/TR/rdf-sparql-query/
• Part of this work has been done with the support of **Linked Data Benchmark Council Project (FP7- 317548)**