SPARQL NUTS & BOLTS (HTTPS://CAMBRIDGESEMANTICS.COM/BLOG/SEMANTIC-UNIVERSITY/LEARN-SPARQL/SPARQL-NUTS-BOLTS/)

SPARQL BY EXAMPLE (HTTPS://CAMBRIDGESEMANTICS.COM/BLOG/SEMANTIC-UNIVERSITY/LEARN-SPARQL/SPARQL-BY-EXAMPLE/)

SPARQL VS SQL (HTTPS://CAMBRIDGESEMANTICS.COM/BLOG/SEMANTIC-UNIVERSITY/LEARN-SPARQL/SPARQL-VS-SQL/)

Learn SPARQL

SPARQL (pronounced “sparkle”) is the query language for the Semantic Web. Along with RDF and OWL, it is one of the three core technologies of the Semantic Web.

This lesson introduces the SPARQL query language, starting with simple queries. Future lessons will build on this material with more advanced SPARQL concepts.

Today’s Lesson

SPARQL is a recursive acronym, which stands for SPARQL Protocol and RDF Query Language.

SPARQL consists of two parts: query language and protocol. The *query* part of that is pretty straightforward. SQL is used to query relational data. XQuery is used to query XML data. SPARQL is used to query RDF data. Despite this similarity, SPARQL differs in that it was designed to operate over disconnected sources over a network in addition to a local database.

In particular, the *SPARQL protocol* allows transmitting SPARQL queries and results between a client and a SPARQL engine via HTTP. We can take advantage of that fact to query live, public SPARQL endpoints, as we’ll see later in this tutorial. A SPARQL endpoint is simply a server that exposes its data via the SPARQL protocol.

We’ll cover the importance of the SPARQL protocol later in the lesson, after introducing some more basic SPARQL concepts.

SPARQL Fundamentals

At its most basic, a SPARQL query is an RDF graph with variables. For example, consider the following RDF graph:

ex:juan foaf:name "Juan Sequeda" .
ex:juan foaf:based_near ex:Austin .

Now consider a version of the previous RDF graph that has variables instead of values:

?x foaf:name ?y .
?x foaf:based_near ?z .

Note that variables in SPARQL queries start with a question mark (?).

At first blush, this is not very different from RDF itself, and that's intentional. SPARQL queries are based on the concept of *graph pattern matching*. A basic SPARQL query is simply a *graph pattern* with some variables. Data that is returned via a query is said to *match* the pattern.

In the example above, we can see that foaf:name and foaf:based_near can be *matched* against the actual RDF data from the real graph. In doing so, we *bind* ?x to ex:juan, ?y to "Juan Sequeda" and ?z to ex:Austin to get the actual RDF graph we showed above.

Before we go on to show a full SPARQL query, let's summarize the vocabulary:

- **Graph pattern**. Specifying a *graph pattern*, which is just RDF using some variables.
- **Matching**. When RDF data matches a specific graph pattern.
- **Binding**. When a specific value in RDF is bound to a variable in a graph pattern.

What does a SPARQL query look like?

The following SPARQL query has all the major components from SPARQL:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM <http://example.com/dataset.rdf>
WHERE {
?x foaf:name ?name .
}
ORDER BY ?name
```

Let's look at each component in turn.

The **PREFIX** keyword describes prefix declarations for abbreviating URIs. Without a prefix, you would have to use the entire URI in the query (<http://xmlns.com/foaf/0.1/name> (http://xmlns.com/foaf/0.1/name%3E)). Create a prefix by using a string (foaf) to reference a part of the URI (<http://xmlns.com/foaf/0.1/> (http://xmlns.com/foaf/0.1/%3E)). When you use the abbreviation (foaf:name), it appends the string after the colon (:) to the URI that is referenced by the prefix string.

The **SELECT** keyword is the most popular of the 4 possible return clauses (more on the others later). If you've used SQL, SELECT serves very much the same function in SPARQL, which is simply to return data matching some conditions. In particular, SELECT queries return data represented in a simple table, where each matching result is a row, and each column is the value for a specific variable. Using our SPARQL query above in which we SELECT ?name, the result would be a table with one column and as many rows as match the query. The variable ?x is not returned.

The **FROM** keyword defines the RDF dataset which is being queried. There is an optional clause, FROM NAMED, which is used when you want to query a named graph.

The **WHERE** clause specifies the query graph pattern to be matched. This is the heart of the query. A graph pattern, as mentioned above, is, in essense, RDF with variables.

Finally, **ORDER BY** is one of the several possible solution modifiers, which are used to rearrange the query results. Other solution modifiers are LIMIT and OFFSET.

Return Clauses

In addition to SELECT, there are three other very important return clauses that you can use: ASK, DESCRIBE, and CONSTRUCT.

ASK queries check if there is at least one result for a given query pattern. The result is true or false.

DESCRIBE queries returns an RDF graph that describes a resource. The implementation of this return form is up to each query engine, so you won't see it used nearly as often as the other return clauses.

CONSTRUCT queries returns an RDF graph that is created from a template specified as part of the query itself. That is, a new RDF graph is created by taking the results of a query pattern and filling in the values of variables that occur in the construct template. CONSTRUCT is used to transform RDF data (for example into a different graph structure and with a different vocabulary than the source data).

CONSTRUCT queries are useful if you have RDF data that was automatically generated and would like to transform it using well-known vocabularies, or if you have RDF data using vocabulary from one ontology but need to translate it to another ontology. After SELECT this is the most common type of query in practice, and a major reason why agreeing on every aspect of an OWL ontology ahead of time is not necessary. Translation using CONSTRUCT is relatively cheap.

In the next SPARQL lesson, we'll show a number of examples of each kind of query.

SPARQL Protocol

The SPARQL protocol enables SPARQL queries over simple HTTP requests. A SPARQL endpoint is simply a service that implements the SPARQL protocol.

For example, if you do a curl on the following:

curl –I http://dbpedia.org/sparql?query=PREFIX%20rdf%3A%20%3Chttp%3A%2F%2Fwww.w3… (http://dbpedia.org/sparql?query=PREFIX%20rdf%3A%20%3Chttp%3A%2F%2Fwww.w3.org%2F1999%2F02%2F22-rdf-syntax-ns%23%3E%0ASELECT%20*%20WHERE%20%7B%0A%3Fcity%20rdf%3Atype%20%3Chttp%3A%2F

The response is the following:

```
HTTP/1.1 200 OK
Date: Mon, 21 May 2012 23:43:38 GMT
Content-Type: application/sparql-results+xml; charset=UTF-8
Connection: keep-alive
Server: Virtuoso/06.04.3132 (Linux) x86_64-generic-linux-glibc25-64  VDB
Content-Length: 96743
Accept-Ranges: bytes
```

This means that SPARQL is basically an API!

SPARQL as Federation

What this means is that data exposed via SPARQL *on any server* can be queried by any SPARQL client. This is a fundamental difference between SPARQL and other query languages, such as SQL, which assume that all data being queried is local and conforms to a single model. With SPARQL—and especially when using CONSTRUCT—data from multiple places can be combined dynamically, as needed, to create new forms of information.

Going back to the basic SPARQL syntax shown above, the FROM clause can be used to specify named graphs that sit on any server. We'll show exactly how to merge data from multiple sources in more advanced SPARQL lessons.

So how do you expose data via SPARQL? RDF databases typically include a SPARQL endpoint by default. Even non-RDF data sources can be exposed using a SPARQL endpoint. Future lessons will present ways to turn existing relational databases into SPARQL endpoints, making them part of the Semantic Web.

Note: we won't go into the details of the SPARQL protocol query encoding here since you're much more likely to be writing actual SPARQL and using tools to issue the queries over HTTP for you.

Conclusion

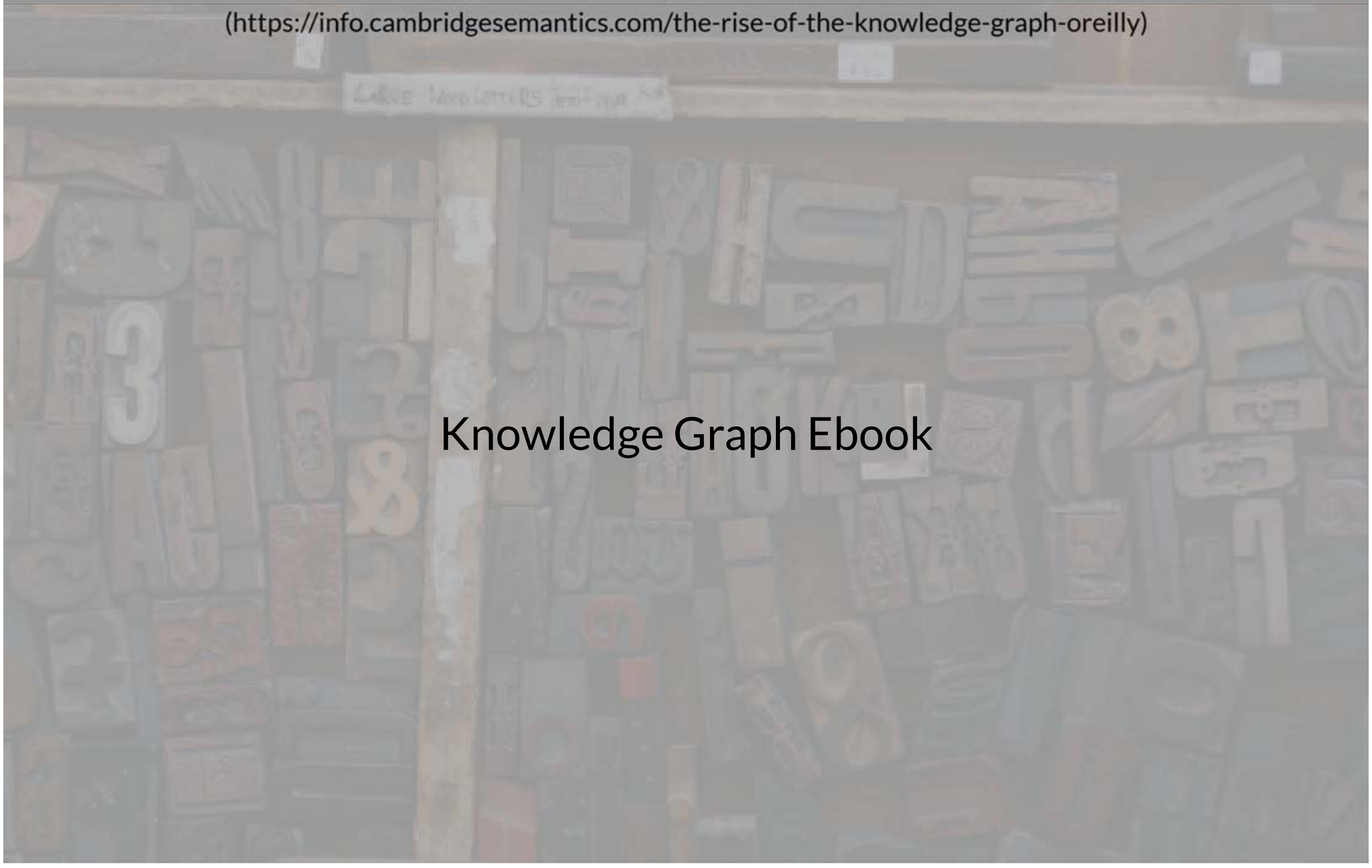
This is a brief introduction to the SPARQL query language. In the next lesson, SPARQL Nuts & Bolts (https://cambridgesemantics.com/semantic-university/sparql-nuts-bolts) we will look at each of the return clauses in the context of real-world queries that you can run on a public SPARQL endpoint.

< PREVIOUS LESSON (HTTPS://CAMBRIDGESEMANTICS.COM/BLOG/SEMANTIC-UNIVERSITY/LEARN-OWL-RDFS/OWL-REFERENCES-HUMANS/)

NEXT LESSON > (HTTPS://CAMBRIDGESEMANTICS.COM/BLOG/SEMA UNIVERSITY/LEARN-SPARQL/SPARQL-NUTS-BOLTS/)

(https://blog.cambridgesemantics.com/sparql-one-standard-to-rule-them-all)

SPARQL - One Standard to Rule them All



(https://info.cambridgesemantics.com/the-rise-of-the-knowledge-graph-oreilly)

Knowledge Graph Ebook

(https://info.cambridgesemantics.com/what-is-data-fabric-design)

(https://info.cambridgesemantics.com/an-integrated-data-enterprise)

CAMBRIDGE SEMANTICS

(https://www.

1 Beacon St | 15th Floor | Boston, MA 02108