Apache Jena

- Home
- Download
- Learn
    - Tutorials
    - Overview
    - RDF core API tutorial
    - SPARQL tutorial
    - Manipulating SPARQL using ARQ
    - Using Jena with Eclipse
    - How-To's
    -
    - References
    - Overview
    - Javadoc
    - RDF API
    - RDF I/O
    - ARQ (SPARQL)
    - RDF Connection - SPARQL API
    - Elephas - tools for RDF on Hadoop
    - Text Search
    - TDB
    - SDB
    - SPARQL over JDBC
    - Fuseki
    - Permissions
    - Assembler
    - Ontology API
    - Inference API
    - Command-line tools
    - Extras
- Javadoc
    - Jena Core
    - ARQ
    - TDB
    - Fuseki
    - Elephas
    - Text Search
    - Spatial Search
    - Permissions
    - JDBC
    - All Javadoc
- Ask
- Get involved
    - Contribute
    - Report a bug
    -
    - Project
    - About Jena
    - Roadmap
    - Architecture
    - Project team

SPARQL Tutorial - Basic Patterns

This section covers basic patterns and solutions, the main building blocks of SPARQL queries.

Solutions

Query solutions are a set of pairs of a variable name with a value. A `SELECT` query directly exposes the solutions (after order/limit/offset are applied) as the result set - other query forms use the solutions to make a graph. The solution is the way the pattern matched - which values the variables must take for a pattern to match.

The first query example had a single solution. Change the pattern to this second query: (q-bp1.rq):

```
SELECT ?x ?fname
WHERE {?x  <http://www.w3.org/2001/vcard-rdf/3.0#FN>  ?fname}
```

This has 4 solutions, one for each VCARD name property triples in the data source

```
----------------------------------------------------
| x                                | name          |
====================================================
| <http://somewhere/RebeccaSmith/> | "Becky Smith" |
| <http://somewhere/SarahJones/>   | "Sarah Jones" |
| <http://somewhere/JohnSmith/>    | "John Smith"  |
| <http://somewhere/MattJones/>    | "Matt Jones"  |
----------------------------------------------------
```

So far, with triple patterns and basic patterns, every variable will be defined in every solution. The solutions to a query can be thought of a table, but in the general case, it is a table where not every row will have a value for every column. All the solutions to a given SPARQL query don't have to have values for all the variables in every solution as we shall see later.

Basic Patterns

A basic pattern is a set of triple patterns. It matches when the triple patterns all match with the same value used each time the variable with the same name is used.

```
SELECT ?givenName
WHERE
  { ?y  <http://www.w3.org/2001/vcard-rdf/3.0#Family>  "Smith" .
    ?y  <http://www.w3.org/2001/vcard-rdf/3.0#Given>  ?givenName .
  }
```

This query (q-bp2.rq) involves two triple patterns, each triple ends in a '.' (but the dot after the last one can be omitted like it was in the one triple pattern example). The variable y has to be the same for each triple pattern match. The solutions are:

```
-------------
| givenName |
=============
| "John"    |
| "Rebecca" |
-------------
```

QNames

There is shorthand mechanism for writing long URIs using prefixes. The query above is more clearly written as the query (q-bp3.rq):

```
PREFIX vcard:      <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?givenName
WHERE
 { ?y vcard:Family "Smith" .
   ?y vcard:Given  ?givenName .
 }
```

This is a prefixing mechanism - the two parts of the URIs, from the prefix declaration and from the part after the ":" in the qname, are concatenated together. This is strictly not what an XML qname is but uses the RDF rule for turning a qname into a URI by concatenating the parts.

Blank Nodes

Change the query just a little to return y as well (q-bp4.rq) :

```
PREFIX vcard:      <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?y ?givenName
WHERE
 { ?y vcard:Family "Smith" .
   ?y vcard:Given  ?givenName .
 }
```

and the blank nodes appear

```
--------------------
| y    | givenName |
====================
| _:b0 | "John"    |
| _:b1 | "Rebecca" |
--------------------
```

as odd looking qnames starting _:. This isn't the internal label for the blank node - it is ARQ printing them out that assigned the _:b0, _:b1 to show when two blank nodes are the same. Here they are different. It does not reveal the internal label used for the blank node although that is available when using the Java API.

Next: Filters