Triple terms and reification in SPARQL

An Introduction to RDF and SPARQL 1.2
ISWC 2025
2 November 2025

Pierre-Antoine Champin
Enrico Franconi
Ora Lassila
Ruben Taelman

https://www.w3.org/Talks/2025/iswc-tutorial-rdfsparql-12/

Outline

- Query syntax
- Functions
- Comparisons
- Results

Outline

- **Query syntax**
- Functions
- Comparisons
- Results

Triple terms use the “<<(...)>>” syntax

Syntax remains aligned with Turtle

```
VERSION "1.2"
PREFIX : <http://example.com/ns#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?person ?authority {
  _:r rdf:reifies <<( ?person :jobTitle "Designer" )>> .
  _:r :accordingTo ?authority .
}
```

Triple term

“<< ... >>” as shorthand for writing reifying triples.

Triple terms are rarely used directly.

```
SELECT ?person ?authority {
  << ?person :jobTitle "Designer" >>
       :accordingTo ?authority .
}
```

Reifying triple

Expands to:

```
SELECT ?person ?authority {
  _:r rdf:reifies <<( ?person :jobTitle "Designer" )>> .
  _:r :accordingTo ?authority .
}
```

Reifier

“<< … ~ :reifier >>” sets a specific reifier

Reifier can be IRIs, blank nodes, or variables.

```
SELECT ?person ?authority {
  << ?person :jobTitle "Designer" ~ :id >> Reifier
      :accordingTo ?authority .
}
```

“{| … |}” reifies and asserts triples

The annotation syntax

```
SELECT ?p ?auth ?date {
  ?p :jobTitle ?title {| :accordingTo ?auth; :recorded ?date |} .
}
```

Expands to:

Annotation block

```
SELECT ?p ?auth ?date {
  ?p :jobTitle ?title .
  << ?p :jobTitle ?title >> :accordingTo ?auth ;
                            :recorded ?date .
}
```

Outline

- Query syntax
- **Functions**
- Comparisons
- Results

Five new functions related to triple terms

To be used within expressions (FILTER, BIND, ...)

`TRIPLE(subj, pred, obj)`	Triple term expression
`SUBJECT(triple-term)`	Get subject of triple term
`PREDICATE(triple-term)`	Get predicate of triple term
`OBJECT(triple-term)`	Get object of triple term
`isTRIPLE(term)`	If this is a triple term

Dynamically *bind* triple terms (1)

Using BIND, triple terms can be bound to a variable.

```
SELECT ?tt ?date {
    ?s ?p ?o .
    BIND( TRIPLE(?s, ?p, ?o) AS ?tt )
    ?reifier rdf:reifies ?tt .
    ?reifier :tripleAdded ?date .
}
```

→*Lookup all asserted reifying triples*

Dynamically *bind* triple terms (2)

TRIPLE can also be written using the "<<(...)>>" shorthand.

```
SELECT ?tt ?date {
    ?s ?p ?o .
    BIND( <<( ?s ?p ?o )>> AS ?tt )
    ?reifier rdf:reifies ?tt .
    ?reifier :tripleAdded ?date .
}
```

Limitation: no sub-expressions allowed within this shorthand.

Allowed: `BIND( TRIPLE(?s, ?p, STR(?o)) AS ?tt )`

Disallowed: `BIND( <<( ?s ?p STR(?o) )>> AS ?tt )`

Retrieve *subjects* of triple terms

Using SUBJECT, the subject of a triple term is returned.

```
SELECT ?s {
    :id rdf:reifies ?tt .
    BIND(SUBJECT(?tt) as ?s)
}
```

Outline

- Query syntax
- Functions
- **Comparisons**
- Results

SPARQL depends on different forms of comparison

Introduction of triple terms has an impact on all of them:

- sameTerm
- sameValue (=)
- Operator mappings (!=, <, <=, >, >=)
- ORDER BY

sameTerm(term1, term2): true if terms are the same RDF term

True if one of the following is true:

- Both IRIs that are the same as IRIs (character-by-character).
- Both literals that are equal as literal terms. (*lexical space*, case-sensitive)
- Both blank nodes that are equal as blank nodes.
- Both triple terms that are equal as triple terms
  - the subject, predicate, and object components are pair-wise the same term.

term1 = term2: true if terms correspond to the same value

A.k.a. sameValue(term1, term) *(was called RDFterm-equal in SPARQL 1.1)*

Similar to sameTerm, but literals are compared in their *value space* (more flexible).

Difference between sameTerm and sameValue (=)

`sameTerm(123, 123)`	`true`
`sameTerm(123, 123.0)`	`false`
`sameTerm(<<(:s :p 123)>>, <<(:s :p 123.0)>>)`	`false`
`123 = 123`	`true`
`123 = 123.0`	`true`
`<<(:s :p 123)>> = <<(:s :p 123.0)>>`	`true`

Operator mappings (<, <=, >, >=) are applied recursively

Pairwise comparisons on triple terms in the order of subject, predicate, object.

`<<(:s :p 123)>>` **<** `<<(:s :p 124)>>`	`true`
`<<(:s :p 123)>>` **>** `<<(:s :p 124)>>`	`false`
`<<(:s :p 123)>>` **<=** `<<(:s :p 123)>>`	`true`

Still under discussion!

ORDER BY now also considers triple terms

“<” operator is used for directly comparable terms.

Otherwise, incomparable terms are sorted in this order:

- No value
- Blank nodes
- IRIs
- Literals
- Triple terms

Outline

- Query syntax
- Functions
- Comparisons
- **Results**

SPARQL results formats also support triple terms

SPARQL JSON

SPARQL XML

SPARQL CSV

SPARQL TSV

SPARQL JSON: “triple” term types

```
{
  "head": { "vars": [ "x", "name", "triple" ] },
  "results": {
    "bindings": [
      {
        "x":    { "type": "bnode",   "value": "r1" },
        "name": { "type": "literal", "value": "Alice" },
        "triple": {
          "type": "triple",
          "value": {
            "subject":   { "type": "uri",     "value": "http://example.org/alice" },
            "predicate": { "type": "uri",     "value": "http://example.org/name" },
            "object":    { "type": "literal", "value": "Alice", "datatype": "..." }
          }
        }
      }
    ]
  }
}
```

SPARQL XML: <triple> tag

```
<?xml version="1.0"?>
<sparql …>
  <head> … </head>
  <results>
    <result>
      <binding name="x"><bnode>r2</bnode></binding>
      <binding name="name"><literal xml:lang="en">Bob</literal></binding>
      <binding name="triple">
        <triple>
            <subject><uri>http://example.org/alice</uri></subject>
            <predicate><uri>http://example.org/name</uri></predicate>
            <object><literal
datatype="http://www.w3.org/2001/XMLSchema#string">Alice</literal></object>
        </triple>
      </binding>
    </result>
  </results>
</sparql>
```

SPARQL CSV: encoding in “<<(...)>>”

x	triple
Alice	<<(http://example/alice http://example/knows http://example/bob)>>
Bob	<<(http://example/bob http://example/knows http://example/alice)>>
Carol	<<(http://example/carol http://example/says "Hello world, my name is Alice.")>>

Warning: SPARQL CSV is lossy! (no datatypes)

SPARQL TSV: encoding in “<<(...)>>”

?x	?triple
"Alice"	<<(<http://example/alice> <http://example/knows> <http://example/bob>)>>
"Bob"	<<(<http://example/bob> <http://example/knows> <http://example/alice>)>>
"Carol"	<<(<http://example/carol> <http://example/says> "Hello world, my name is Alice".)>>

Outline

- Query syntax
- Functions
- Comparisons
- Results