INF3580/4580 – Semantic Technologies – Spring 2017

Lecture 13: SPARQL 1.1

Ernesto Jiménez-Ruiz

8th May 2017

Department of Informatics

University of Oslo

Today's Plan

1. Introduction
2. Recap: SPARQL 1.0
3. SPARQL 1.1 QUERY language
   - Assignment and Expressions
   - Aggregates
   - Subqueries
   - Negation
   - Property paths
4. SPARQL 1.1 Federated Query
5. SPARQL 1.1 UPDATE Language
6. SPARQL 1.1 Entailment Regimes

Outline

Introduction

- Today's lecture

Ernesto Jiménez-Ruiz (ernestoj@ifi.uio.no)
http://www.mn.uio.no/ifi/english/people/aca/ernestoj/
Office hours: from 9:00 to 16:00 at OJD 8165

- Lessons
  - February 6th: SPARQL 1.0
  - April 3rd: OWL loose ends (Profiles and others)
  - **May 8th: More SPARQL (SPARQL 1.1 and entailment regimes)**
  - May 15th: OBDA: Ontology Based Data Access

SPARQL

- SPARQL Protocol And RDF Query Language
- Standard language to query graph data represented as **RDF triples**
- W3C Recommendations
  - **SPARQL 1.0:** W3C Recommendation 15 January 2008
  - **SPARQL 1.1:** W3C Recommendation 21 March 2013

SPARQL

- SPARQL Protocol And RDF Query Language
- Standard language to query graph data represented as **RDF triples**
- W3C Recommendations
  - **SPARQL 1.0:** W3C Recommendation 15 January 2008
  - **SPARQL 1.1:** W3C Recommendation 21 March 2013
- This lecture is about SPARQL 1.1.
- Documentation:
  - SPARQL 1.1 Query Language.
    `https://www.w3.org/TR/sparql11-query/`

Outline

Query with Basic Graph Pattern

Names of people who have published with "Ernesto Jimenez-Ruiz"

```
SELECT DISTINCT ?collab WHERE {
    ?ejr foaf:name "Ernesto Jimenez-Ruiz" .
    ?pub dc:creator ?ejr .
    ?pub dc:creator ?other .
    ?other foaf:name ?collab.
}
```

Query with Basic Graph Pattern

Names of people who have published with "Ernesto Jimenez-Ruiz"

```
SELECT DISTINCT ?collab WHERE {
    ?ejr foaf:name "Ernesto Jimenez-Ruiz" .
    ?pub dc:creator ?ejr .
    ?pub dc:creator ?other .
    ?other foaf:name ?collab.
}
```

`PREFIX` declarations omitted in some examples, use `http://prefix.cc` to find!

Query with Basic Graph Pattern

Names of people who have published with "Ernesto Jimenez-Ruiz"

```
SELECT DISTINCT ?collab WHERE {
    ?ejr foaf:name "Ernesto Jimenez-Ruiz" .
    ?pub dc:creator ?ejr .
    ?pub dc:creator ?other .
    ?other foaf:name ?collab.
}
```

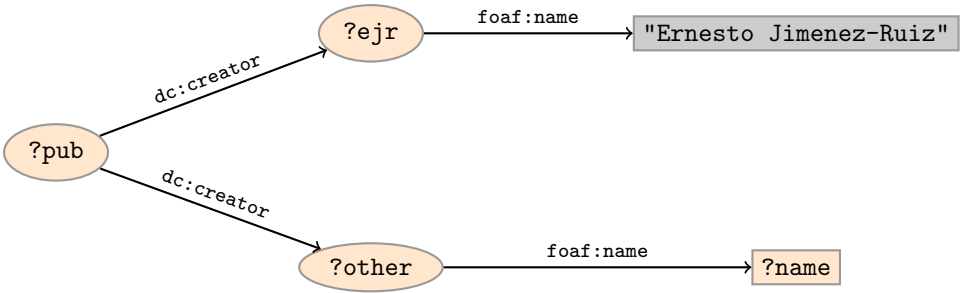
`PREFIX` declarations omitted in some examples, use `http://prefix.cc` to find!

Answer:

?name
"Ernesto Jimenez-Ruiz"
"Jorge Sales"
"Ian Horrocks"
"Bernardo Cuenca Grau"
"Rafael Berlanga Llavori"

Graph Patterns

The previous SPARQL query as a graph:



Pattern matching: assign values to variables to make this a sub-graph of the RDF graph!

SPARQL Query with blank nodes

Names of people who have published with "Ernesto Jimenez-Ruiz"

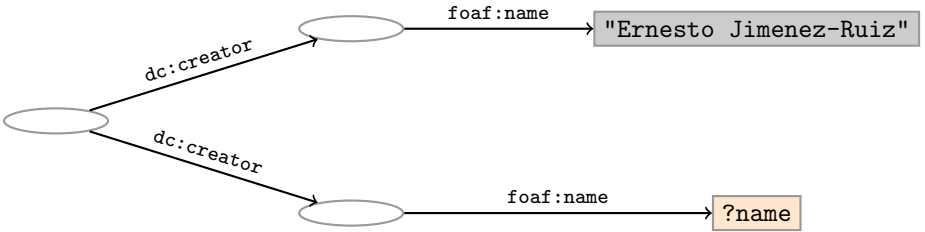
```
SELECT DISTINCT ?name WHERE {
    _:ejr foaf:name "Ernesto Jimenez-Ruiz" .
    _:pub dc:creator _:ejr .
    _:pub dc:creator _:other .
    _:other foaf:name ?name.
}
```

The same with blank node syntax

```
SELECT DISTINCT ?name WHERE {
    [ dc:creator [foaf:name "Ernesto Jimenez-Ruiz"] ,
                 [foaf:name ?name]
    ]
}
```

Graph with blank nodes

Variables not SELECTed can equivalently be blank:



Pattern matching: assign values to variables and blank nodes to make this a sub-graph of the RDF graph!

Components of an SPARQL query

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?collab
FROM <http://dblp_dataset>
WHERE {
    ?ejr foaf:name "Ernesto Jimenez-Ruiz" .
    ?pub dc:creator ?ejr .
    ?pub dc:creator ?other .
    OPTIONAL {
        ?other foaf:name ?collab .
        FILTER (STR(?collab)!="Ernesto Jimenez-Ruiz")
    }
}
ODER BY ?collab
```

Components of an SPARQL query

Prologue: prefix definitions

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?collab
FROM <http://dblp_dataset>
WHERE {
    ?ejr foaf:name "Ernesto Jimenez-Ruiz" .
    ?pub dc:creator ?ejr .
    ?pub dc:creator ?other .
    OPTIONAL {
        ?other foaf:name ?collab .
        FILTER (STR(?collab)!="Ernesto Jimenez-Ruiz")
    }
}
ODER BY ?collab
```

Components of an SPARQL query

Results form specification: (1) variable list, (2) type of query (SELECT, ASK, CONSTRUCT, DESCRIBE), (3) remove duplicates (DISTINCT, REDUCED)

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?collab
FROM <http://dblp_dataset>
WHERE {
    ?ejr foaf:name "Ernesto Jimenez-Ruiz" .
    ?pub dc:creator ?ejr .
    ?pub dc:creator ?other .
    OPTIONAL {
        ?other foaf:name ?collab .
        FILTER (STR(?collab)!="Ernesto Jimenez-Ruiz")
    }
}
ODER BY ?collab
```

Components of an SPARQL query

Dataset specification

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?collab
FROM <http://dblp_dataset>
WHERE {
    ?ejr foaf:name "Ernesto Jimenez-Ruiz" .
    ?pub dc:creator ?ejr .
    ?pub dc:creator ?other .
    OPTIONAL {
        ?other foaf:name ?collab .
        FILTER (STR(?collab)!="Ernesto Jimenez-Ruiz")
    }
}
ODER BY ?collab
```

Components of an SPARQL query

Query pattern: graph pattern to be matched: BGP, FILTER, OPTIONAL, GROUPS, UNION, RDF Datasets

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?collab
FROM <http://dblp_dataset>
WHERE {
    ?ejr foaf:name "Ernesto Jimenez-Ruiz" .
    ?pub dc:creator ?ejr .
    ?pub dc:creator ?other .
    OPTIONAL {
        ?other foaf:name ?collab .
        FILTER (STR(?collab)!="Ernesto Jimenez-Ruiz")
    }
}
ODER BY ?collab
```

Components of an SPARQL query

Solution modifiers: ORDER BY, LIMIT, OFFSET

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?collab
FROM <http://dblp_dataset>
WHERE {
    ?ejr foaf:name "Ernesto Jimenez-Ruiz" .
    ?pub dc:creator ?ejr .
    ?pub dc:creator ?other .
    OPTIONAL {
        ?other foaf:name ?collab .
        FILTER (STR(?collab)!="Ernesto Jimenez-Ruiz")
    }
}
ODER BY ?collab
```

Outline

SPARQL 1.1: new fatures

- The new features in SPARQL 1.1 QUERY language:
  - Aggregates
  - Subqueries
  - Negation
  - Expressions in the SELECT clause
  - Property paths
  - Assignment
  - A short form for CONSTRUCT
  - An expanded set of functions and operators

SPARQL 1.1: new fatures

- The new features in SPARQL 1.1 QUERY language:
  - Aggregates
  - Subqueries
  - Negation
  - Expressions in the SELECT clause
  - Property paths
  - Assignment
  - A short form for CONSTRUCT
  - An expanded set of functions and operators
- SPARQL 1.1 UPDATE Language
- SPARQL 1.1 Federated Queries
- SPARQL 1.1 Entailment Regimes

SPARQL 1.1: new fatures

- The new features in SPARQL 1.1 QUERY language:
  - Aggregates
  - Subqueries
  - Negation
  - Expressions in the SELECT clause
  - Property paths
  - Assignment
  - A short form for CONSTRUCT
  - An expanded set of functions and operators
- SPARQL 1.1 UPDATE Language
- SPARQL 1.1 Federated Queries
- SPARQL 1.1 Entailment Regimes
- Rationale for the extensions of SPARQL 1.0
  `https://www.w3.org/TR/sparql-features/`

Assignment and Expressions

- The value of an expression can be assigned/bound to a new variable
- Can be used in SELECT, BIND or GROUP BY clauses: *(expression AS ?var)*

Assignment and Expressions

- The value of an expression can be assigned/bound to a new variable
- Can be used in SELECT, BIND or GROUP BY clauses: *(expression AS ?var)*

Products with price < 20 taking into account discount

```
SELECT ?title ?price)
{
  { ?x ns:price ?p .
    ?x ns:discount ?discount
    BIND (?p*(1-?discount) AS ?price)
  }
  { ?x dc:title ?title .
    FILTER(?price < 20)
  }
}
```

Assignment and Expressions

- The value of an expression can be assigned/bound to a new variable
- Can be used in SELECT, BIND or GORUP BY clauses: *(expression AS ?var)*

Expressions in SELECT clause

```
  SELECT ?title (?p AS ?fullPrice) (?fullPrice*(1-?discount) AS
?customerPrice))
  {
    ?x ns:price ?p .
    ?x dc:title ?title .
    ?x ns:discount ?discount
  }
```

Aggregates: Grouping and Filtering

- Solutions can optionally be grouped according to one or more expressions.
- To specify the group, use `GROUP BY`.
- If `GROUP BY` is not used, then only one (implicit) group

Aggregates: Grouping and Filtering

- Solutions can optionally be grouped according to one or more expressions.
- To specify the group, use `GROUP BY`.
- If `GROUP BY` is not used, then only one (implicit) group
- To filter solutions resulting from grouping, use `HAVING`.
- `HAVING` operates over grouped solution sets, in the same way that FILTER operates over un-grouped ones.

Aggregates: Example

Counties of Norway with less than 15 municipalities

```
SELECT ?name (count(?kommune) AS ?kcount)
WHERE {
  ?county a gd:Fylke ;
          gn:officialName ?name ;
          gn:hasmunicipality ?kommune .
  ?kommune a gd:Kommune .
}
GROUP BY ?name
HAVING (?kcount < 15)
```

Aggregates: Example

Counties of Norway with less than 15 municipalities

```
SELECT ?name (count(?kommune) AS ?kcount)
WHERE {
  ?county a gd:Fylke ;
          gn:officialName ?name ;
          gn:hasmunicipality ?kommune .
  ?kommune a gd:Kommune .
}
GROUP BY ?name
HAVING (?kcount < 15)
```

Note: Only expressions consisting of aggregates and constants may be projected, together with variables in `GROUP BY`.

Aggregates: functions

- `Count` counts the number of times a variable has been bound.
- `Sum` sums numerical values of bound variables.
- `Avg` finds the average of numerical values of bound variables.
- `Min` finds the minimum of the numerical values of bound variables.
- `Max` finds the maximum of the numerical values of bound variables.
- `Group_Concat` creates a string with the values concatenated, separated by some optional character.
- `Sample` just returns a sample of the values.

Subqueries

- Subqueries are a way to embed SPARQL queries within other queries
- To achieve results which cannot otherwise be achieved, for example, limiting the number of results from some sub-expression within the query.

Subqueries

- Subqueries are a way to embed SPARQL queries within other queries
- To achieve results which cannot otherwise be achieved, for example, limiting the number of results from some sub-expression within the query.

Return the lowest sort order name of Alice's friends

```
SELECT ?y ?minName WHERE {
  :alice :knows ?y .
  {
    SELECT ?y (MIN(?name) AS ?minName) WHERE {
       ?y :name ?name .} GROUP BY ?y
  }
}
```

- Subqueries are evaluated logically first, and the results are projected up to the outer query.
- Only variables projected out of the subquery will be visible, or in scope, to the outer query.

Negation in SPARQL 1.0

COMBINING OPTIONAL, FILTER and !BOUND:

People without names

```
SELECT DISTINCT * WHERE {
    ?person a foaf:Person .
    OPTIONAL {
        ?person foaf:name ?name .
    FILTER (!bound(?name))
    }
}
```

Negation in SPARQL 1.0

COMBINING OPTIONAL, FILTER and !BOUND:

People without names

```
SELECT DISTINCT * WHERE {
    ?person a foaf:Person .
    OPTIONAL {
        ?person foaf:name ?name .
    FILTER (!bound(?name))
    }
}
```

However, this is not very easy to write.

Negation in SPARQL 1.1: MINUS and FILTER NOT EXISTS

Two ways to do negation:

People without names

```
SELECT DISTINCT * WHERE {
    ?person a foaf:Person .
    MINUS { ?person foaf:name ?name }
}
```

People without names, take II

```
SELECT DISTINCT * WHERE {
    ?person a foaf:Person .
    FILTER NOT EXISTS { ?person foaf:name ?name }
}
```

Negation in SPARQL 1.1: MINUS and FILTER NOT EXISTS (cont.)

They may produce different results. Data with `ex:Ernesto a foaf:Person`

```
SELECT DISTINCT * WHERE {
    ?s ?p ?o .
    MINUS { ?x ?y ?z }
}
```

```
SELECT DISTINCT * WHERE {
    ?s ?p ?o .
    FILTER NOT EXISTS { ?x ?y ?z }
}
```

Negation in SPARQL 1.1: MINUS and FILTER NOT EXISTS (cont.)

They may produce different results. Data with `ex:Ernesto a foaf:Person`

```
SELECT DISTINCT * WHERE {
    ?s ?p ?o .
    MINUS { ?x ?y ?z }
}
```

Does not remove solutions (no shared variables!) and returns `ex:Ernesto a foaf:Person`

```
SELECT DISTINCT * WHERE {
    ?s ?p ?o .
    FILTER NOT EXISTS { ?x ?y ?z }
}
```

Returns no solutions. Since there are not shared variables, it removes all solutions.

Open and Closed World Assumptions

Aggregates and negation assume Closed World and Unique names!
The answers are only true with respect to the current dataset.

- "As far as we know, there are 13 municipalities in Vestfold."
- Can't say: "they don't have names", can say: "we don't know their names".
- "As far as we know, no-one has climbed that mountain."
- "Based on the available data, the average fuel price is currently 13.37 NOK/l."

This will have implications when combined with reasoning.

Property paths: basic motivation

- Some queries get needlessly complex.
- Sometimes write `foaf:maker|dct:creator` instead of `UNION`.
- To get friend's name, go { `_:me foaf:knows/foaf:name ?friendsname` }.
- Sum several items:

```
SELECT (sum(?cost) AS ?total) { :order :hasItem/:price ?cost }
```

- etc.
- Adds a small property-oriented query language inside the language.

Property paths: syntax

Syntax Form	Matches
`iri`	An (property) IRI. A path of length one.
`^elt`	Inverse path (object to subject).
`elt1 / elt2`	A sequence path of `elt1` followed by `elt2`.
`elt1 | elt2`	A alternative path of `elt1` or `elt2` (all possibilities are tried).
`elt*`	Seq. of zero or more matches of `elt`.
`elt+`	Seq. of one or more matches of `elt`.
`elt?`	Zero or one matches of `elt`.
`!iri or !(iri`$_1$`| ...|iri`$_n$`)`	Negated property set.
`!^iri or !(^iri`$_1$`| ...|^iri`$_n$`)`	Negation of inverse path.
`!(iri`$_1$`|...|iri`$_j$`|^iri`$_{j+1}$`|...|^iri`$_n$`)`	Negated combination of forward and inverese properties.
`(elt)`	A group path `elt`, brackets control precedence.

* `elt` is a path element, which may itself be composed of path constructs (see Syntax form).

Property paths: example

The names of all my friends of friends

```
SELECT ?name WHRE {
  uio:Ernesto foaf:knows+ ?friend
  ?friend foaf:name|foaf:firstName ?name .
}
```

Outline

Federated query support

- The `SERVICE` keyword instructs a federated query processor to invoke a portion of a SPARQL query against a remote SPARQL service/endpoint.
- SPARQL service: any implementation conforming to the *SPARQL 1.1 Protocol for RDF*

Federated query support

- The `SERVICE` keyword instructs a federated query processor to invoke a portion of a SPARQL query against a remote SPARQL service/endpoint.
- SPARQL service: any implementation conforming to the *SPARQL 1.1 Protocol for RDF*

Combining local file with remote SPARQL service

```
SELECT ?name
FROM <http://example.org/mylocalfoaf.rdf>
WHRE {
  <http://example.org/mylocalfoaf/I> foaf:knows ?person .
  SERVICE <http://people.example.org/sparql> {
    ?person foaf:name ?name .   }
}
```

Outline

SPARQL 1.1 UPDATE

- Do not confuse with CONSTRUCT
- CONSTRUCT is an alternative for SELECT
- Instead of returning a table of result values, CONSTRUCT returns an RDF graph accoding to the template

SPARQL 1.1 UPDATE

- Do not confuse with CONSTRUCT
- CONSTRUCT is an alternative for SELECT
- Instead of returning a table of result values, CONSTRUCT returns an RDF graph accoding to the template
- SPARQL 1.1 UPDATE is a language to modify the given GRAPH
- `https://www.w3.org/TR/2013/REC-sparql11-update-20130321/`

SPARQL 1.1 UPDATE: Inserting and deleting triples

Inserting triples in a graph

```
INSERT DATA {
  GRAPH </graph/courses/> {
    <course/inf3580> ex:taughtBy <staff/ernestoj> .
    <staff/ernestoj> foaf:name "Ernesto Jimenez Ruiz" ;
} }
```

SPARQL 1.1 UPDATE: Inserting and deleting triples

Inserting triples in a graph

```
INSERT DATA {
  GRAPH </graph/courses/> {
    <course/inf3580> ex:taughtBy <staff/ernestoj> .
    <staff/ernestoj> foaf:name "Ernesto Jimenez Ruiz" ;
} }
```

Deleting triples from a graph

```
DELETE DATA {
  GRAPH </graph/courses/> {
    <course/inf3580> ex:oblig <exercise/oblig6> .
    <exercise/oblig6> rdfs:label "Mandatory Exercise 6" .
} }
```

SPARQL 1.1 UPDATE: Inserting and deleting triples

Inserting triples in a graph

```
INSERT DATA {
  GRAPH </graph/courses/> {
    <course/inf3580> ex:taughtBy <staff/ernestoj> .
    <staff/ernestoj> foaf:name "Ernesto Jimenez Ruiz" ;
} }
```

Deleting triples from a graph

```
DELETE DATA {
  GRAPH </graph/courses/> {
    <course/inf3580> ex:oblig <exercise/oblig6> .
    <exercise/oblig6> rdfs:label "Mandatory Exercise 6" .
} }
```

If no `GRAPH` is given, default graph is used.

SPARQL 1.1 UPDATE: Inserting conditionally

Most useful when inserting statements that you already have, but hold true for something else.

Inserting triples for another subject

```
INSERT {
  <http:// .../geo/inndeling/03> a gd:Fylke ;
       gn:name "Oslo" ;
       ?p ?o .
}
WHERE {
  <http:// .../geo/inndeling/03/0301> a gd:Kommune ;
            ?p ?o .
}
```

SPARQL 1.1 UPDATE: Deleting conditionally

From specification:

Deleting old books

```
DELETE {
   ?book ?p ?v .
}
WHERE {
  ?book dc:date ?date .
  FILTER ( ?date < "2000-01-01T00:00:00"^^xsd:dateTime )
  ?book ?p ?v .
}
```

SPARQL 1.1 UPDATE: Deleting conditionally, common shortform

Deleting exactly what's matched by the `WHERE` clause.

```
DELETE WHERE {
  ?s a skos:Concept .
  ?s ?p <http://smil.uio.no/topic/betennelse-i-bihuler> .
}
```

SPARQL 1.1 UPDATE: Delete/Insert full syntax

In most cases, you would delete some triples first, then add new, possibly in the same or other graphs.
From specification:

All the possibilities offered by DELETE/INSERT

```
( WITH IRIref )?
( ( ( DELETE QuadPattern ) ( INSERT QuadPattern )?  )  | (INSERT
QuadPattern) )
( USING ( NAMED )? IRIref )*
WHERE GroupGraphPattern
```

SPARQL 1.1 UPDATE: Delete/Insert simple example

Update user information query from Sublima

```
DELETE {
  <http:// .../user/larshvermannsen> ?p ?o .
}
INSERT {
  <http:// .../user/larshvermannsen> a sioc:User ;
    rdfs:label """Lars Hvermannsen"""@no ;
    sioc:email <mailto:lars@hvermannsen.no> ;
    sioc:has_function <http:// .../role/Administrator> ;
    wdr:describedBy status:inaktiv .
}
WHERE {
  <http:// .../user/larshvermannsen> ?p ?o .
}
```

SPARQL 1.1 UPDATE: Delete/Insert example with named graphs

Update user information query from Sublima

```
DELETE {
  GRAPH </graphs/users/> {
    <http:// .../user/larshvermannsen> ?p ?o .
  }
}
INSERT {
  GRAPH </graphs/users/> {
    <http:// .../user/larshvermannsen> a sioc:User ;
          rdfs:label """Lars Hvermannsen"""@no .
  }
}
USING </graphs/users/>  WHERE {
    <http:// .../user/larshvermannsenno> ?p ?o .
}
```

SPARQL 1.1 UPDATE: Delete/Insert example explained

- `USING` plays the same role as `FROM`.

SPARQL 1.1 UPDATE: Delete/Insert example explained

- `USING` plays the same role as `FROM`.
- `GRAPH` says where to insert or delete.

SPARQL 1.1 UPDATE: Delete/Insert example explained

- `USING` plays the same role as `FROM`.
- `GRAPH` says where to insert or delete.
- This makes it possible to delete, insert and match against different graphs.

SPARQL 1.1 UPDATE: Delete/Insert example with single named graphs

Update user information query from Sublima

```
WITH </graphs/users/>
DELETE {
  <http:// .../user/larshvermannsen> ?p ?o .
}
INSERT {
  <http:// .../user/larshvermannsen> a sioc:User ;
        rdfs:label """Lars Hvermannsen"""@no .
}
WHERE {
  <http:// .../user/larshvermannsenno> ?p ?o .
}
```

Equivalent to the previous query!

SPARQL 1.1 UPDATE: Whole graph operations

From the specification:

`LOAD ( SILENT )? IRIref_from ( INTO GRAPH IRIref_to )?`

Loads the graph at IRIref_from into the specified graph, or the default graph if not given.

SPARQL 1.1 UPDATE: Whole graph operations

From the specification:

`LOAD ( SILENT )? IRIref_from ( INTO GRAPH IRIref_to )?`

Loads the graph at IRIref_from into the specified graph, or the default graph if not given.

`CLEAR ( SILENT )? (GRAPH IRIref | DEFAULT | NAMED | ALL )`

Removes the triples from the specified graph, the default graph, all named graphs or all graphs respectively. Some implementations may remove the whole graph.

SPARQL 1.1 UPDATE: Whole graph operations

From the specification:

`LOAD ( SILENT )? IRIref_from ( INTO GRAPH IRIref_to )?`

Loads the graph at IRIref_from into the specified graph, or the default graph if not given.

`CLEAR ( SILENT )? (GRAPH IRIref | DEFAULT | NAMED | ALL )`

Removes the triples from the specified graph, the default graph, all named graphs or all graphs respectively. Some implementations may remove the whole graph.

`CREATE ( SILENT )? GRAPH IRIref`

Creates a new graph in stores that record empty graphs.

SPARQL 1.1 UPDATE: Whole graph operations

From the specification:

`LOAD ( SILENT )? IRIref_from ( INTO GRAPH IRIref_to )?`

Loads the graph at IRIref_from into the specified graph, or the default graph if not given.

`CLEAR ( SILENT )? (GRAPH IRIref | DEFAULT | NAMED | ALL )`

Removes the triples from the specified graph, the default graph, all named graphs or all graphs respectively. Some implementations may remove the whole graph.

`CREATE ( SILENT )? GRAPH IRIref`

Creates a new graph in stores that record empty graphs.

`DROP ( SILENT )? (GRAPH IRIref | DEFAULT | NAMED | ALL )`

Removes the specified graph, the default graph, all named graps or all graphs respectively. It also removes all triples of those graphs.

SPARQL 1.1 UPDATE: Whole graph operations

From the specification:

`LOAD ( SILENT )? IRIref_from ( INTO GRAPH IRIref_to )?`

Loads the graph at IRIref_from into the specified graph, or the default graph if not given.

`CLEAR ( SILENT )? (GRAPH IRIref | DEFAULT | NAMED | ALL )`

Removes the triples from the specified graph, the default graph, all named graphs or all graphs respectively. Some implementations may remove the whole graph.

`CREATE ( SILENT )? GRAPH IRIref`

Creates a new graph in stores that record empty graphs.

`DROP ( SILENT )? (GRAPH IRIref | DEFAULT | NAMED | ALL )`

Removes the specified graph, the default graph, all named graps or all graphs respectively. It also removes all triples of those graphs.

Also provides shortcuts, `COPY`, `MOVE` and `ADD`.

SPARQL 1.1 UPDATE: Whole graph operations

From the specification:

`LOAD ( SILENT )? IRIref_from ( INTO GRAPH IRIref_to )?`

: Loads the graph at IRIref_from into the specified graph, or the default graph if not given.

`CLEAR ( SILENT )? (GRAPH IRIref | DEFAULT | NAMED | ALL )`

: Removes the triples from the specified graph, the default graph, all named graphs or all graphs respectively. Some implementations may remove the whole graph.

`CREATE ( SILENT )? GRAPH IRIref`

: Creates a new graph in stores that record empty graphs.

`DROP ( SILENT )? (GRAPH IRIref | DEFAULT | NAMED | ALL )`

: Removes the specified graph, the default graph, all named graps or all graphs respectively. It also removes all triples of those graphs.

Also provides shortcuts, `COPY`, `MOVE` and `ADD`.

Usually, `LOAD` and `DROP` are what you want.

Outline

Entailment regimes: overview

- Gives guiadance for SPARQL query engines
- Basic graph pattern by means of subgraph matching: *simple entailment*
- Solutions that implicitly follow from the queried graph: *entailment regimes*
- **RDF entailment**, **RDF Schema entailment**, D-Entailment, **OWL 2 RDF-Based Semantics entailment**, **OWL 2 Direct Semantics entailment**, and RIF-Simple entailment
- `https://www.w3.org/TR/2013/REC-sparql11-entailment-20130321/`

Entailment regimes: example (1)

- `ex:book1 rdf:type ex:Publication .`
- `ex:book2 rdf:type ex:Article .`
- `ex:Article rdfs:subClassOf ex:Publication .`
- `ex:publishes rdfs:range ex:Publication .`
- `ex:MITPress ex:publishes ex:book3 .`

Entailment regimes: example (1)

- `ex:book1 rdf:type ex:Publication .`
- `ex:book2 rdf:type ex:Article .`
- `ex:Article rdfs:subClassOf ex:Publication .`
- `ex:publishes rdfs:range ex:Publication .`
- `ex:MITPress ex:publishes ex:book3 .`

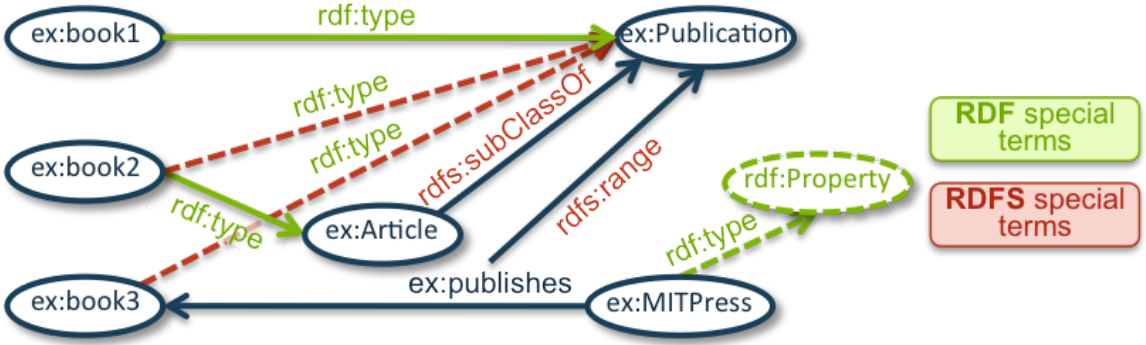
QUERY 1: `SELECT ?prop WHERE  ?prop rdf:type rdf:Property`

Entailment regimes: example (1)

- `ex:book1 rdf:type ex:Publication .`
- `ex:book2 rdf:type ex:Article .`
- `ex:Article rdfs:subClassOf ex:Publication .`
- `ex:publishes rdfs:range ex:Publication .`
- `ex:MITPress ex:publishes ex:book3 .`

QUERY 2: `SELECT ?pub WHERE  ?pub rdf:type ex:Publication`

Entailment regimes: example (2)



Entailment regimes: example (3)

- `ex:book1 rdf:type ex:Publication .`
- `ex:book2 rdf:type ex:Article .`
- `ex:Article rdfs:subClassOf ex:Publication .`
- `ex:publishes rdfs:range ex:Publication .`
- `ex:MITPress ex:publishes ex:book3 .`

Using RDFS entailment regime (new entailed triples):

- `ex:publishes rdf:type rdf:Property .`

Using RDFS entailment regime (new entailed triples):

- `ex:book2 rdf:type ex:Publication .`
- `ex:book3 rdf:type ex:Publication .`

(Graph mathcing is performed over the extended RDF graph)

The OWL Entailment Regimes

- OWL 2 RDF-based Semantics Entailment Regime
- OWL 2 Direct Semantics Entailment Regime

- `https://www.w3.org/TR/2013/REC-sparql11-entailment-20130321/`
- Birte Glimm. **Using SPARQL with RDFS and OWL entailment**. International Conference on Reasoning Web, 2011

OWL 2 Direct Semantics Entailment Regime

Challenges:

- Expressive datatype constructs may lead to infinite answers:
  - i.e., required binding to infinitely many integer values
  - Solution: limit to literals explicitly mentioned in graph
- OWL Direct Semantics defined in terms of OWL objects
  - RDF graph and query must first be translated.
  - Restriction on RDF graphs and SPARQL queries

OWL 2 Direct Semantics Entailment Regime

Challenges:

- Expressive datatype constructs may lead to infinite answers:
  - i.e., required binding to infinitely many integer values
  - Solution: limit to literals explicitly mentioned in graph
- OWL Direct Semantics defined in terms of OWL objects
  - RDF graph and query must first be translated.
  - Restriction on RDF graphs and SPARQL queries

Variable typing:

- In order to have an unambiguous correspondence between BGPs and OWL objects
- e.g., `?x rdf:type TYPE .`
- owl:Class, owl:ObjectProperty, owl:DataProperty, owl:Datatype, or owl:NamedIndividual

OWL 2 RDF-based Semantics Entailment Regime

- Direct extension of the RDFS semantics
- It interprets RDF triples directly without the need of mapping an RDF graph into OWL objects
- Treats classes as individuals that refer to elements of the domain
- This may lead to less consequences than expected (Incompleteness)

OWL 2 Entailment Regimes: example

- Graph: `x:a rdf:type ex:C`
- BGP in query:

```
?x rdf:type
  [
  rdf:type owl:Class ;
  owl:unionOf( ex:C ex:D )
  ]
```

OWL 2 Entailment Regimes: example

- Graph: `x:a rdf:type ex:C`
- BGP in query:

```
?x rdf:type
  [
  rdf:type owl:Class ;
  owl:unionOf( ex:C ex:D )
  ]
```

- `ex:a` not resturned in the solution for `?x` using OWL 2 RDF-Based Semantics
  - G does not include that this union is the class extension of any domain element
  - Solution: add statement `ex:E owl:unionOf ( ex:C ex:D )`

OWL 2 Entailment Regimes: example

- Graph: `x:a rdf:type ex:C`
- BGP in query:

```
?x rdf:type
  [
  rdf:type owl:Class ;
  owl:unionOf( ex:C ex:D )
  ]
```

- `ex:a` not resturned in the solution for `?x` using OWL 2 RDF-Based Semantics
  - G does not include that this union is the class extension of any domain element
  - Solution: add statement `ex:E owl:unionOf ( ex:C ex:D )`
  - These type of statement may lead to undecidability

OWL 2 Entailment Regimes: example

- Graph: `x:a rdf:type ex:C`
- BGP in query:

```
?x rdf:type
  [
  rdf:type owl:Class ;
  owl:unionOf( ex:C ex:D )
  ]
```

- `ex:a` not resturned in the solution for `?x` using OWL 2 RDF-Based Semantics
  - G does not include that this union is the class extension of any domain element
  - Solution: add statement `ex:E owl:unionOf ( ex:C ex:D )`
  - These type of statement may lead to undecidability
- `ex:a` would be a solution for `?x` using OWL 2 Direct Semantics
  - classes denote sets and not domain elements

OWL 2 Entailment Regimes: Complexity and Profiles

- Entailment under OWL 2 (DL) Direct Semantics entailemnt is decidable, but computationally hard.

OWL 2 Entailment Regimes: Complexity and Profiles

- Entailment under OWL 2 (DL) Direct Semantics entailemnt is decidable, but computationally hard.
- Entailment under OWL 2 (DL) RDF-based semantics is incomplete and undecidable for OWL 2 Full.

OWL 2 Entailment Regimes: Complexity and Profiles

- Entailment under OWL 2 (DL) Direct Semantics entailemnt is decidable, but computationally hard.
- Entailment under OWL 2 (DL) RDF-based semantics is incomplete and undecidable for OWL 2 Full.
- No Direct Semantics for OWL 2 Full.

OWL 2 Entailment Regimes: Complexity and Profiles

- Entailment under OWL 2 (DL) Direct Semantics entailemnt is decidable, but computationally hard.
- Entailment under OWL 2 (DL) RDF-based semantics is incomplete and undecidable for OWL 2 Full.
- No Direct Semantics for OWL 2 Full.
- Direct Semantics for OWL 2 QL and EL Profiles have very nice computational properties.

OWL 2 Entailment Regimes: Complexity and Profiles

- Entailment under OWL 2 (DL) Direct Semantics entailemnt is decidable, but computationally hard.
- Entailment under OWL 2 (DL) RDF-based semantics is incomplete and undecidable for OWL 2 Full.
- No Direct Semantics for OWL 2 Full.
- Direct Semantics for OWL 2 QL and EL Profiles have very nice computational properties.
- Entailemnt under OWL 2 QL and EL RDF-based semantics is incomplete as well.

OWL 2 Entailment Regimes: Complexity and Profiles (cont.)

- OWL 2 RL defines a syntactic subset of OWL 2. For RDF graphs that fall into this syntactic subset:

OWL 2 Entailment Regimes: Complexity and Profiles (cont.)

- OWL 2 RL defines a syntactic subset of OWL 2. For RDF graphs that fall into this syntactic subset:
  - Direct Semantics and RDF-based Semantics yield the same (complete and sound) results.

OWL 2 Entailment Regimes: Complexity and Profiles (cont.)

- OWL 2 RL defines a syntactic subset of OWL 2. For RDF graphs that fall into this syntactic subset:
  - Direct Semantics and RDF-based Semantics yield the same (complete and sound) results.
  - Outside OWL 2 RL, the RDF-Based Semantics can still be used, but reasoning can be incomplete.

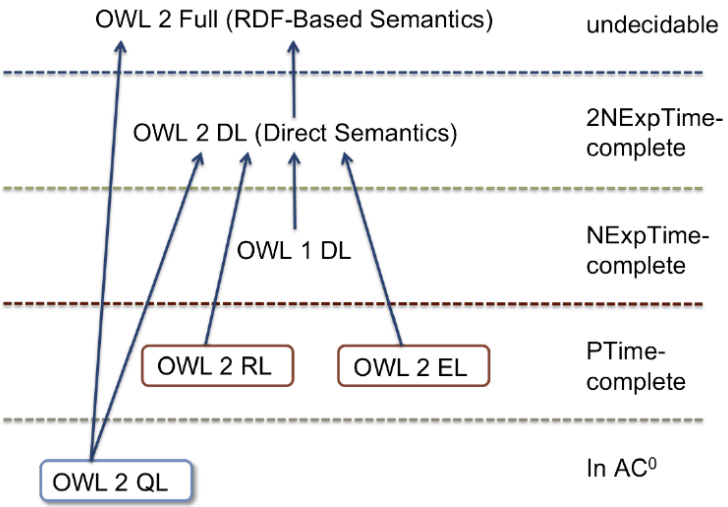
OWL 2 Entailment Regimes: Complexity and Profiles (cont.)

- OWL 2 RL defines a syntactic subset of OWL 2. For RDF graphs that fall into this syntactic subset:
  - Direct Semantics and RDF-based Semantics yield the same (complete and sound) results.
  - Outside OWL 2 RL, the RDF-Based Semantics can still be used, but reasoning can be incomplete.
  - For Direct Semantics the input RDF graph has to satisfy some constrains.

OWL 2 Entailment Regimes: Complexity and Profiles (cont.)

- OWL 2 RL defines a syntactic subset of OWL 2. For RDF graphs that fall into this syntactic subset:
  - Direct Semantics and RDF-based Semantics yield the same (complete and sound) results.
  - Outside OWL 2 RL, the RDF-Based Semantics can still be used, but reasoning can be incomplete.
  - For Direct Semantics the input RDF graph has to satisfy some constrains.
  - The RDF-Based semantics can be use with any RDF graph, but under the OWL 2 RL profile

OWL 2 Entailment Regimes: Complexity and Profiles (cont.)



OWL 2 Entailment Regimes: Systems

- **OWL-BGP:** SPARQL implementation where basic graph patterns are evaluated with OWL 2 Direct Semantics.
  - `https://github.com/iliannakollia/owl-bgp`

OWL 2 Entailment Regimes: Systems

- **OWL-BGP:** SPARQL implementation where basic graph patterns are evaluated with OWL 2 Direct Semantics.
  - `https://github.com/iliannakollia/owl-bgp`
- **RDFox:** highly scalable in-memory RDF triple store that supports parallel datalog reasoning.
  - OWL 2 RL axioms can be directly transformed to datalog rules
  - `https://www.cs.ox.ac.uk/isg/tools/RDFox/`

OWL 2 Entailment Regimes: Systems

- **OWL-BGP:** SPARQL implementation where basic graph patterns are evaluated with OWL 2 Direct Semantics.
  - `https://github.com/iliannakollia/owl-bgp`
- **RDFox:** highly scalable in-memory RDF triple store that supports parallel datalog reasoning.
  - OWL 2 RL axioms can be directly transformed to datalog rules
  - `https://www.cs.ox.ac.uk/isg/tools/RDFox/`
- **ontop:** answering SPARQL queries over databases under OWL 2 QL Entailment regime
  - Ontop is a platform to query relational databases as Virtual RDF Graphs using SPARQL
  - An Ontology in OWL 2 QL and R2RML mappings
  - R2RML: RDB to RDF Mapping Language (more next week!)
  - `http://ontop.inf.unibz.it/`

Entailment Regimes: Service description

- How do we know the Entailment Regime for a SPARQL endpoint?

Entailment Regimes: Service description

- How do we know the Entailment Regime for a SPARQL endpoint?
- SPARQL 1.1 Service Descriptions
- Among other charactersitics, can be used to describe what kind of entailment checkers is used in the backgroud to answer SPARQL queries

Entailment Regimes: Service description

- How do we know the Entailment Regime for a SPARQL endpoint?
- SPARQL 1.1 Service Descriptions
- Among other charactersitics, can be used to describe what kind of entailment checkers is used in the backgroud to answer SPARQL queries
- `sd:defaultEntailmentRegime` or `sd:entailmentRegime`:
  - e.g.: `http://dbpedia.org/sparql sd:entailmentRegime er:OWL-Direct`

Entailment Regimes: Service description

- How do we know the Entailment Regime for a SPARQL endpoint?
- SPARQL 1.1 Service Descriptions
- Among other charactersitics, can be used to describe what kind of entailment checkers is used in the backgroud to answer SPARQL queries
- `sd:defaultEntailmentRegime` or `sd:entailmentRegime`:
  - e.g.: `http://dbpedia.org/sparql sd:entailmentRegime er:OWL-Direct`
- `sd:defaultSupportedEntailmentProfile` or `sd:supportedEntailmentProfile`:
  - e.g.: `http://dbpedia.org/sparql sd:supportedEntailmentProfile owlp:QL`

Entailment Regimes: Service description

- How do we know the Entailment Regime for a SPARQL endpoint?
- SPARQL 1.1 Service Descriptions
- Among other charactersitics, can be used to describe what kind of entailment checkers is used in the backgroud to answer SPARQL queries
- `sd:defaultEntailmentRegime` or `sd:entailmentRegime`:
  - e.g.: `http://dbpedia.org/sparql sd:entailmentRegime er:OWL-Direct`
- `sd:defaultSupportedEntailmentProfile` or `sd:supportedEntailmentProfile`:
  - e.g.: `http://dbpedia.org/sparql sd:supportedEntailmentProfile owlp:QL`
- Unfortunately this information is not always provided

Entailment Regimes: Service description (cont.)

- **Simple Entailment:** http://www.w3.org/ns/entailment/Simple
- **RDF Entailment:** http://www.w3.org/ns/entailment/RDF
- **RDFS Entailment:** http://www.w3.org/ns/entailment/RDFS
- **D Entailment:** http://www.w3.org/ns/entailment/D
- **OWL Entailment with Direct Semantics:** http://www.w3.org/ns/entailment/OWL-Direct
- **OWL Entailment with RDF Based Semantics:** http://www.w3.org/ns/entailment/OWL-RDF-Based
- **RIF Entailment:** http://www.w3.org/ns/entailment/RIF

- **OWL 2 Full entailment checkers**: http://www.w3.org/ns/owl-profile/Full
- **OWL 2 DL entailment checkers**: http://www.w3.org/ns/owl-profile/DL
- **OWL 2 EL entailment checkers**: http://www.w3.org/ns/owl-profile/EL
- **OWL 2 QL entailment checkers**: http://www.w3.org/ns/owl-profile/QL
- **OWL 2 RL entailment checkers**: http://www.w3.org/ns/owl-profile/RL

Questions?

Ernesto Jiménez-Ruiz (ernestoj@ifi.uio.no)

http://www.mn.uio.no/ifi/english/research/groups/logid
http://www.mn.uio.no/ifi/english/people/aca/ernestoj/

Office hours: from 9:00 to 16:00 at OJD 8165