

Shapes for Knowledge Graphs

Presenter: **Jose Emilio Labra Gayo**
WESO Research group
University of Oviedo, Spain



About me...

Main researcher at WESO (Web Semantics Oviedo)

Some books:

"*Web semántica*" (in Spanish), 2012

"*Validating RDF data*", 2017

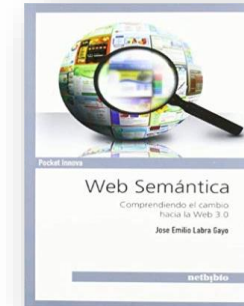
"*Knowledge Graphs*", 2021

...and some software:

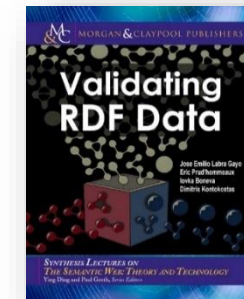
RDFShape (RDF playground)

rudof 

<http://labra.weso.es>

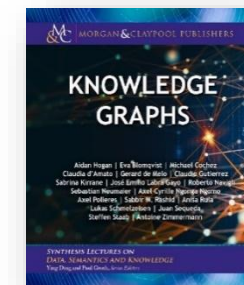


2012



2017 HTML version:

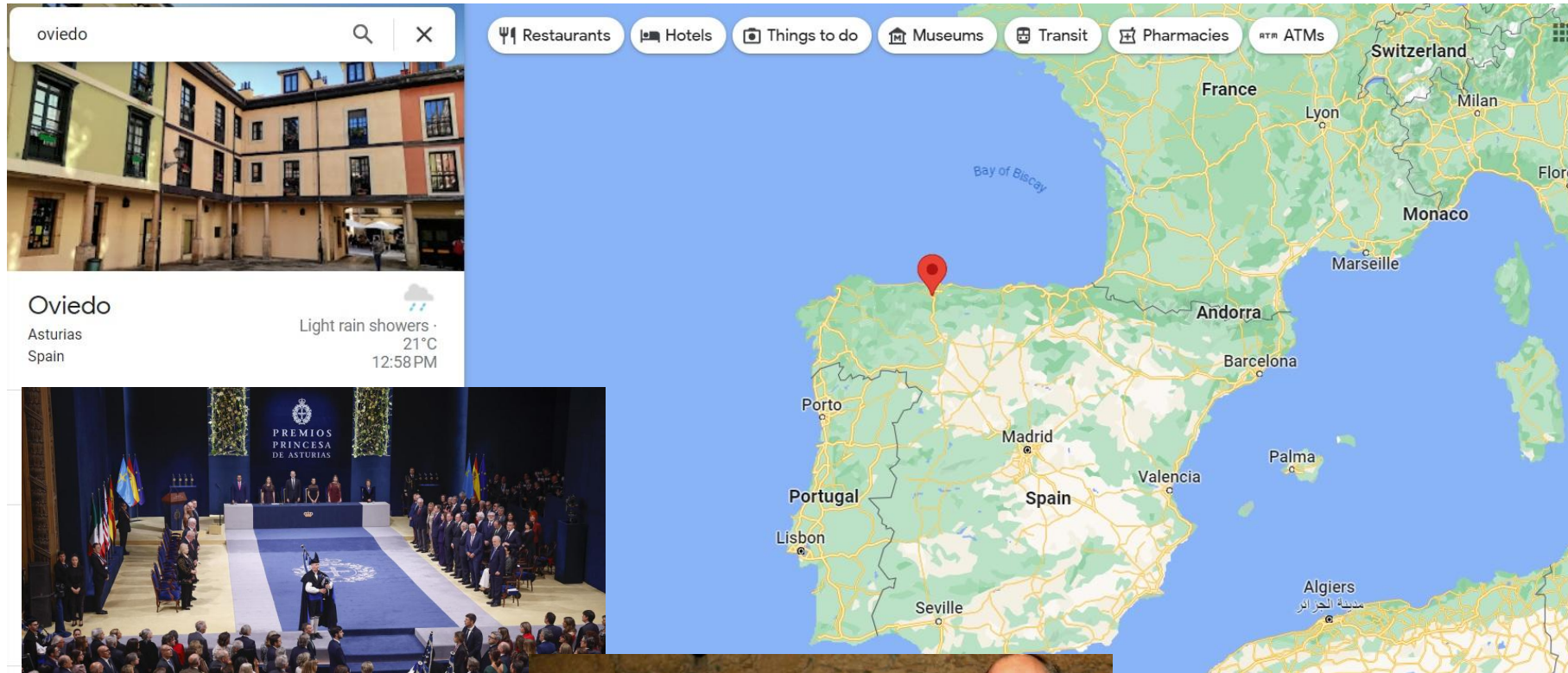
<http://book.validatingrdf.com>



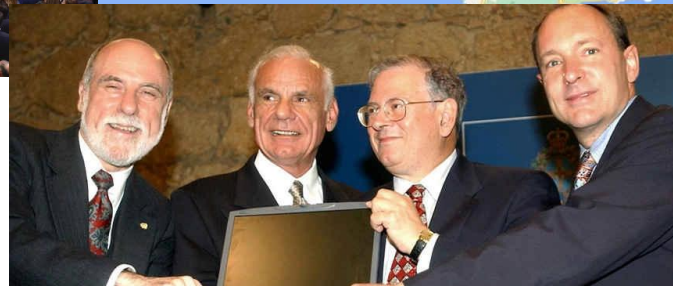
2021, HTML version

<https://kgbook.org/>

About Oviedo, Asturias



Princess of Asturias Awards



2002, Tim Berners-Lee in Asturias receiving the Prince of Asturias Award
Together with: Vinton Cerf, Lawrence Roberts and Robert Kahn

If you want to play...



<https://rudof-project.github.io>

What is rudof?

Knowledge graphs Library

Different models: RDF & RDF 1.2, Wikibase, Property graphs

Shapes or Schemas

ShEx, SHACL, DCTAP, PGSchema, ...

Other tools:

Querying (SPARQL)

Visualizations (UML, HTML, SVG,...)

Generation

...

Available as:

Web demo (WebAssembly)

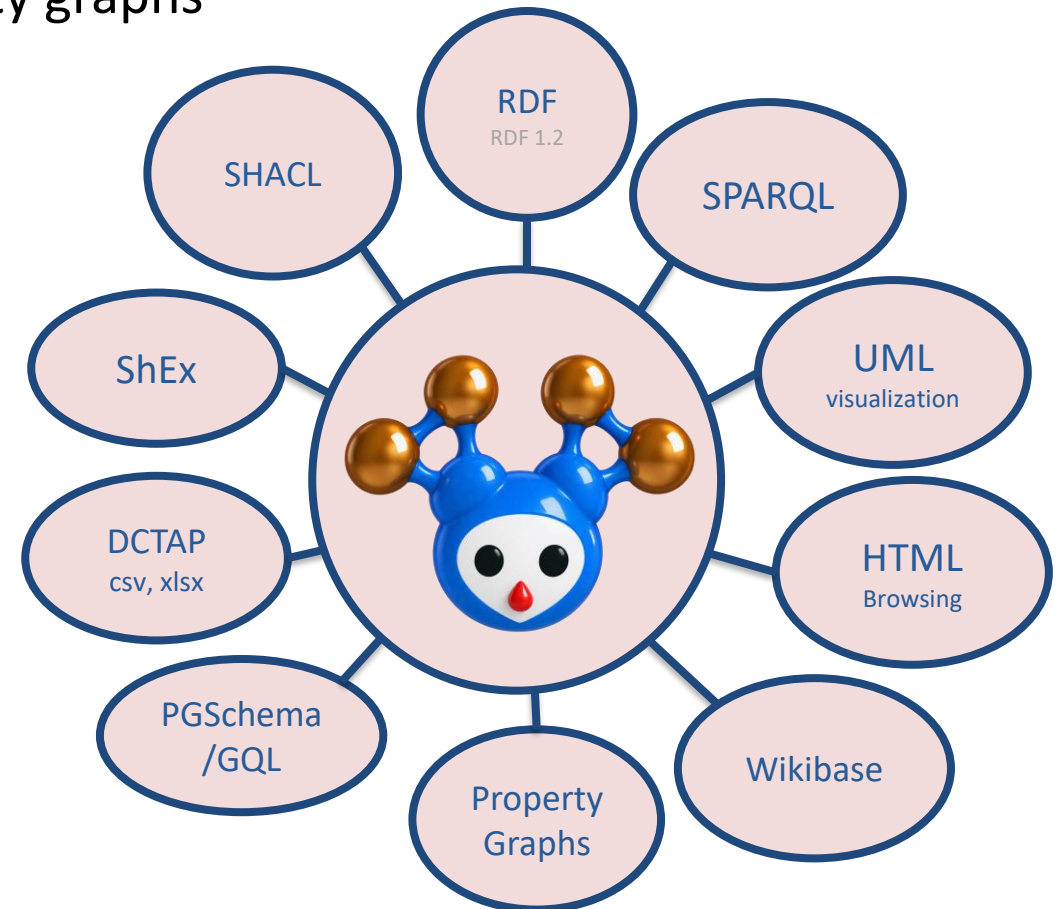


Command line and Interactive Shell

[Windows, Linux, MacOS](#)

Python Bindings: [Jupyter Notebooks](#)

MCP server





rudof CLI

Windows, Linux and macOS binaries

<https://github.com/rudof-project/rudof/releases>

```
> rudof --help
RDF and Knowledge Graphs processing tool

Usage: rudof [OPTIONS] [COMMAND]

Commands:
  mcp          Export rudof as an MCP server
  config       Dump the effective configuration rudof is using as TOML
  shell        Start an interactive shell (REPL) session
  . . .
  help        Print this message or the help of the given subcommand(s)
Options:
  -V, --version  Print version
>
```

Config files are used to customize most of the commands



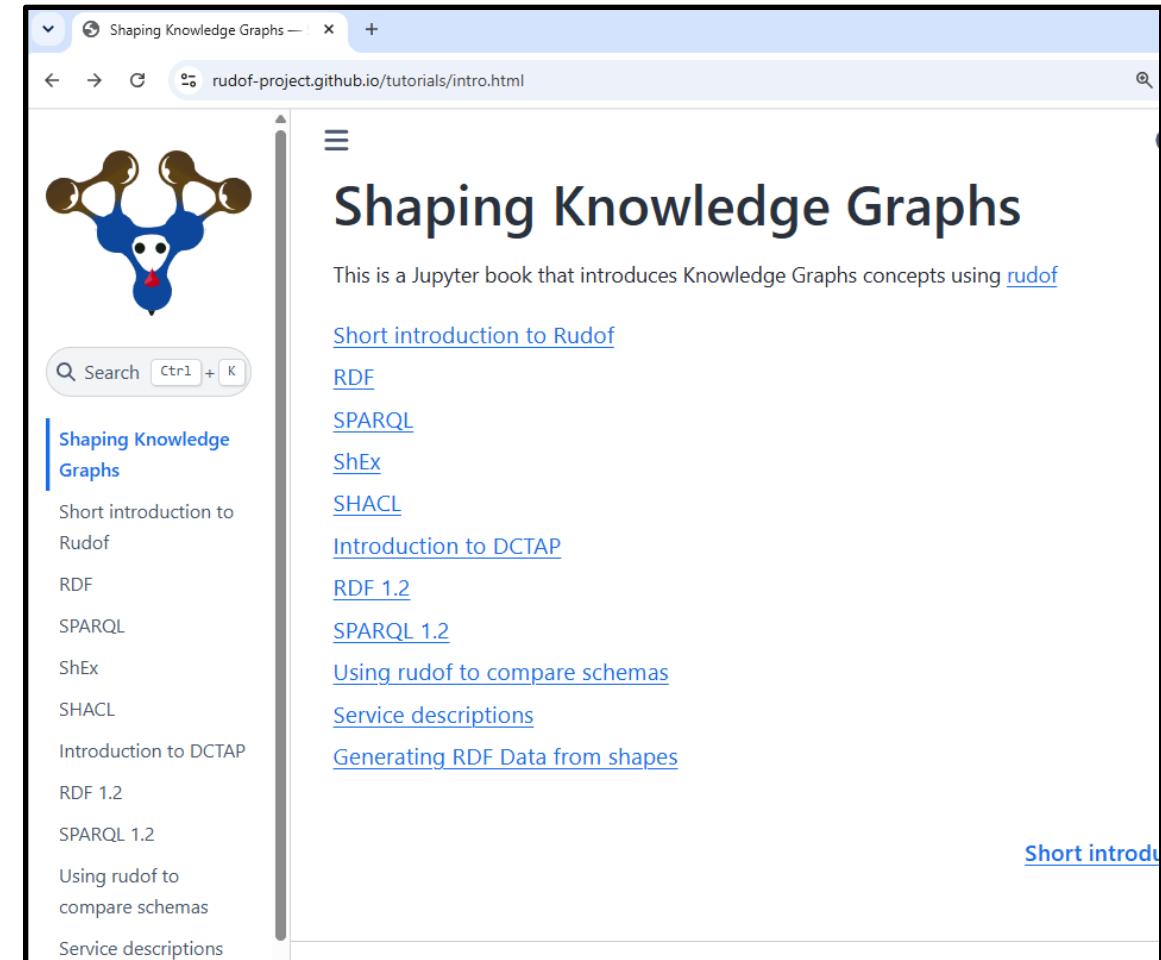
rudof with Python

Python bindings for rudof library available at pypi.org

<https://pypi.org/project/pyrudof/>

Example tutorials available as Jupyter notebooks:

<https://rudof-project.github.io/tutorials/intro.html>



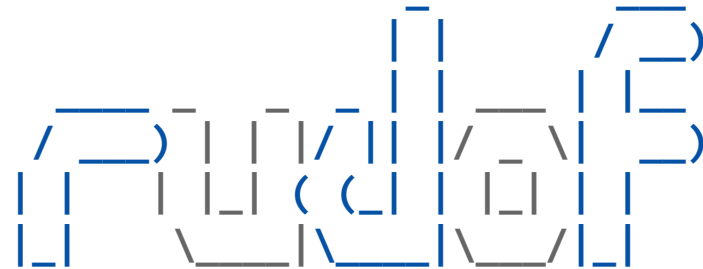


rudof shell

Command line interface

Interactive Shell

```
> rudof shell
```



```
Type 'help' for available commands, 'exit' to quit.
```

```
rudof>
```



rudof online demo

Online demo based
on Web Assembly

 rudof online demo

Examples ▾ Help About ▾  

Convert RDF data and property graphs between formats, validate them with **ShEx**, **SHACL**, **DCTAP** or **PGSchema**, and query RDF data with **SPARQL**. Everything runs in your browser: rudof is compiled to WebAssembly, and nothing you type is sent anywhere, except the queries to the SPARQL endpoint you give. rudof 0.3.28 ready, with endpoints

Data

Schema

Validate

Query

Layout

Side by side ▾

Permalink

ShEx

SHACL

DCTAP

PGSchema

Endpoint

None: the RDF data below ▾

None: validate the RDF data below

The endpoint must allow other sites to query it (CORS).

RDF data

Turtle ▾ 

ShEx schema

ShExC ▾ 

```
1 prefix : <http://example.org/>
2 prefix xsd: <http://www.w3.org/2001/XMLSchema#>
3
4 :alice :name "Alice" ;
5       :birthDate "1995-03-12"^^xsd:date ;
6       :knows :carol .
7
8 :bob   :name "Robert", "Bob" ;
9       :birthDate "unknown" .
10
11 :carol :name "Carol" .
```

```
1 prefix : <http://example.org/>
2 prefix xsd: <http://www.w3.org/2001/XMLSchema#>
3
4 :Person {
5   :name      xsd:string ;
6   :birthDate xsd:date ? ;
7   :knows     @:Person *
8 }
```

Contents



Knowledge graphs & Types of Knowledge Graphs:

RDF, Property graphs, Wikibase, RDF-1.2

Shapes for RDF: ShEx & SHACL, DCTAP

Shapes for other types of Knowledge graphs:

Shaping Wikibase and Wikidata graphs

Shaping Property Graphs

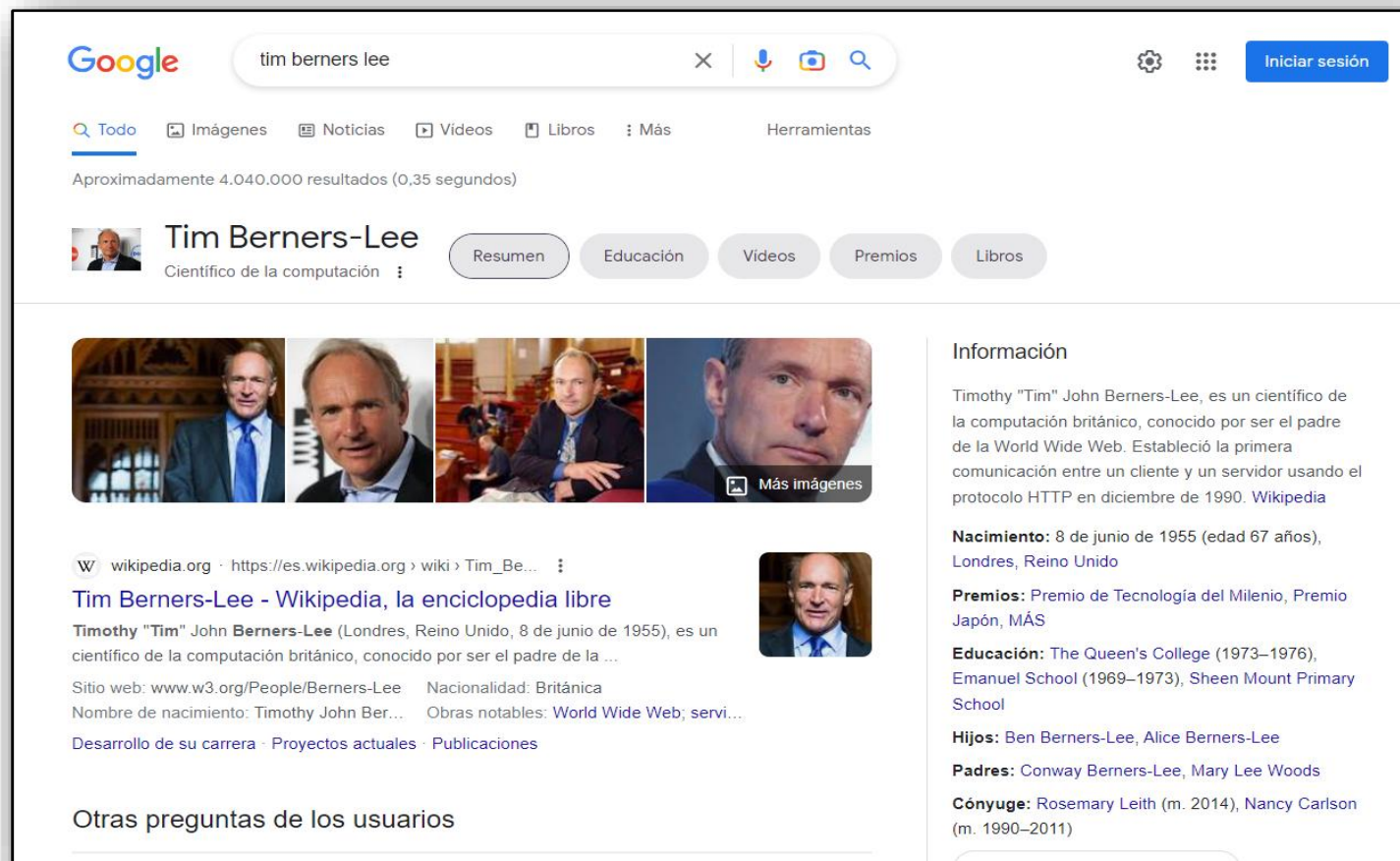
Shaping RDF 1.2

Some ideas for future work

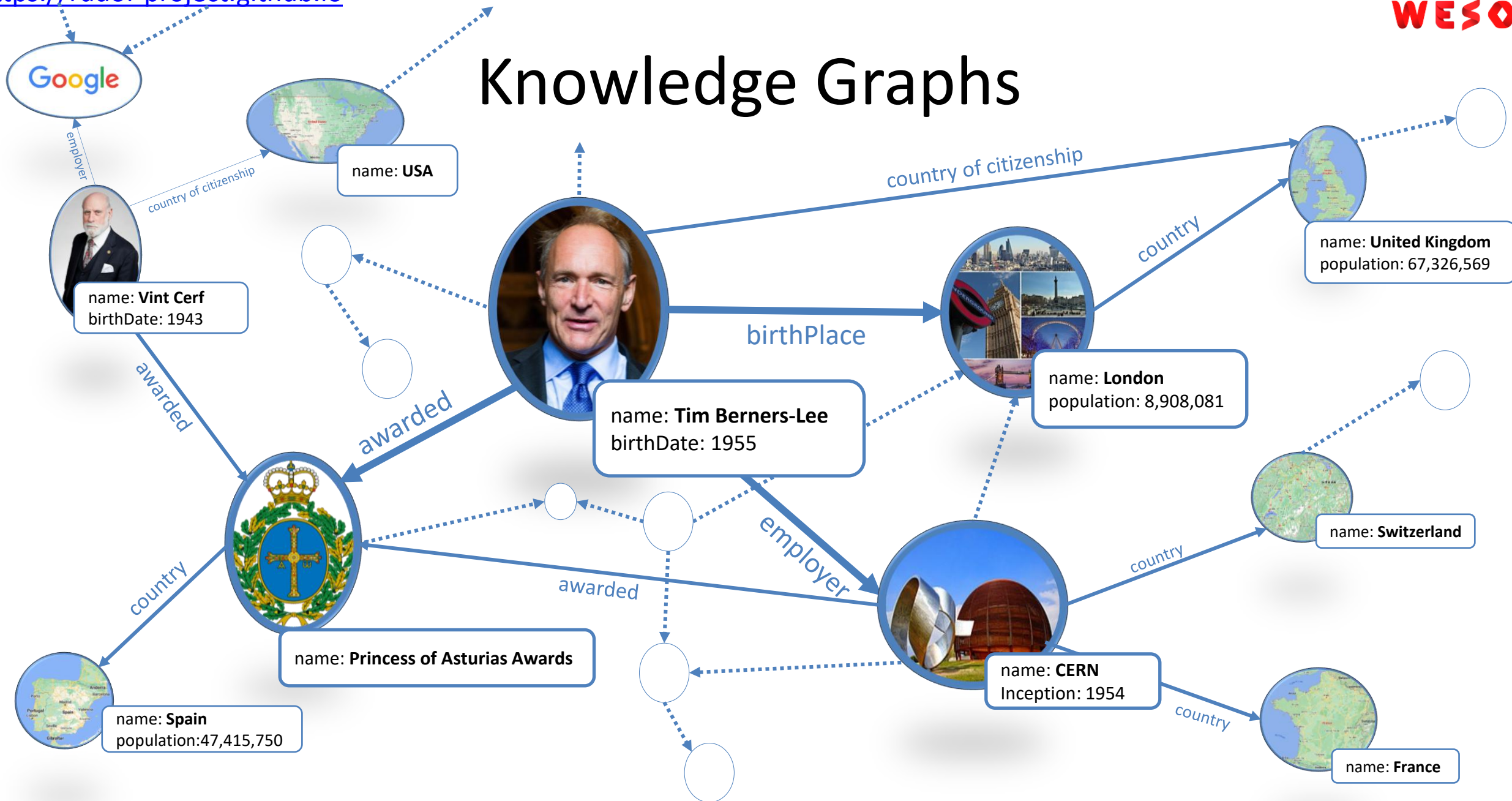


Knowledge Graphs

Current notion of Knowledge Graphs, popular after Google, 2012*



Knowledge Graphs



Types of Knowledge Graphs

- Traditional RDF
- Labeled Property graphs
- Wikibase graphs
- RDF 1.2



RDF graphs

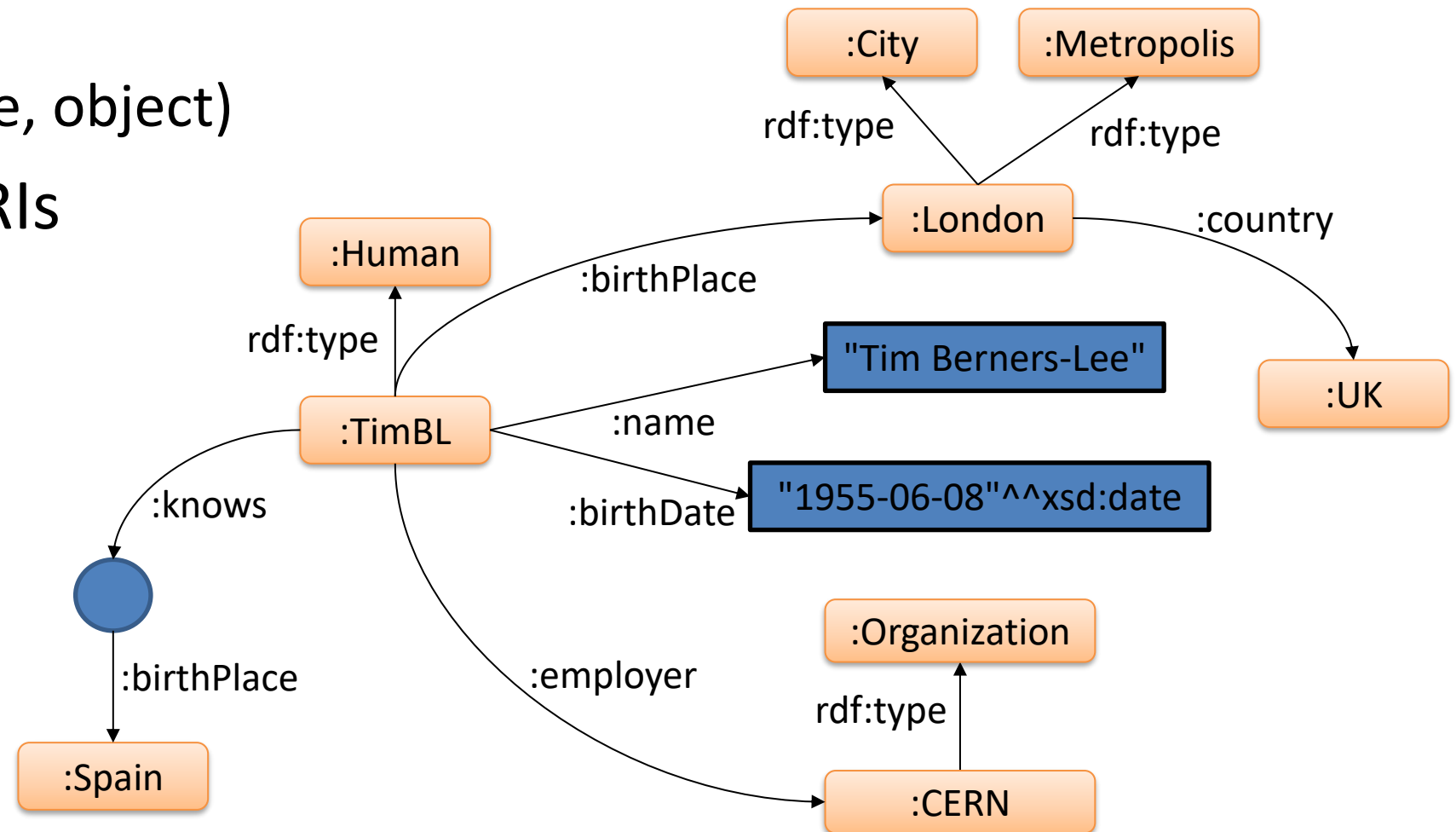
Based on triples

(subject, predicate, object)

Most nodes are URIs

Interoperability

Simple & flexible





RDF ecosystem

One data model, several syntaxes: Turtle, N-Triples, JSON-LD

Vocabularies: RDF Schema, OWL, SKOS, etc.

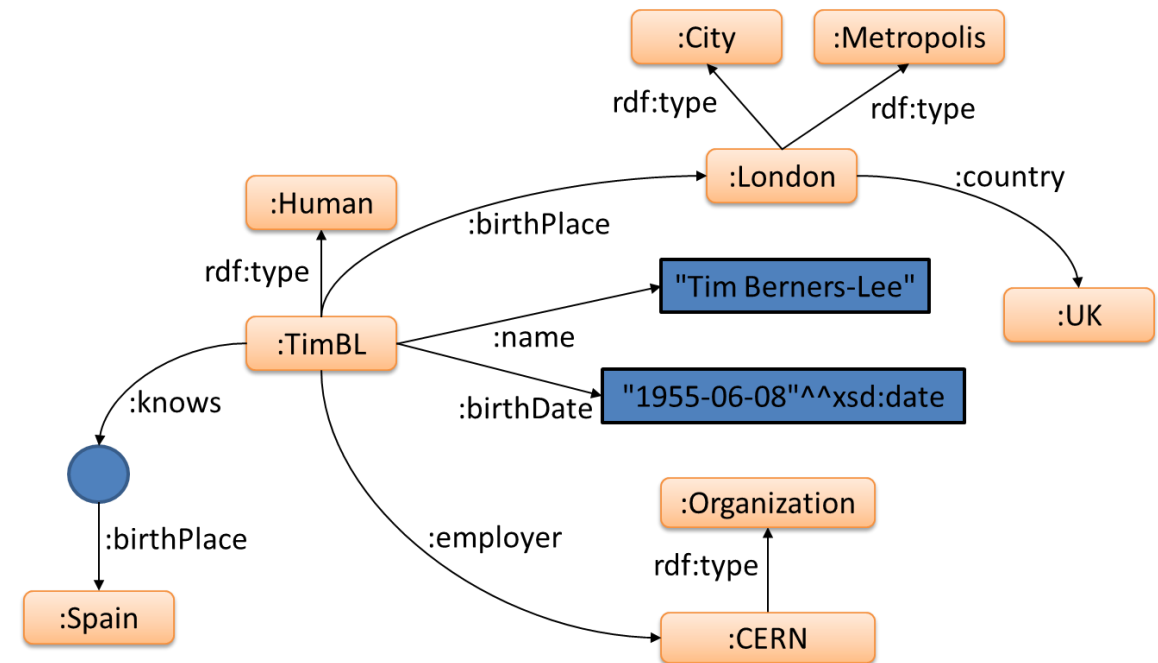
Turtle

```
prefix :      <http://example.org/>
prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
prefix xsd:   <http://www.w3.org/2001/XMLSchema#>
prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
:timbl rdfs:type      :Human ;
       :birthPlace   :london ;
       :name         "Tim Berners-Lee" ;
       :birthDate    "1955-06-08"^^xsd:date ;
       :employer     :CERN ;
       :knows        _:1 .

:london rdfs:type     :City, :Metropolis ;
       :country      :UK .

:CERN   rdfs:type     :Organization .
_:1     :birthPlace   :Spain .
```



Try it: <https://tinyurl.com/mr5eub96>



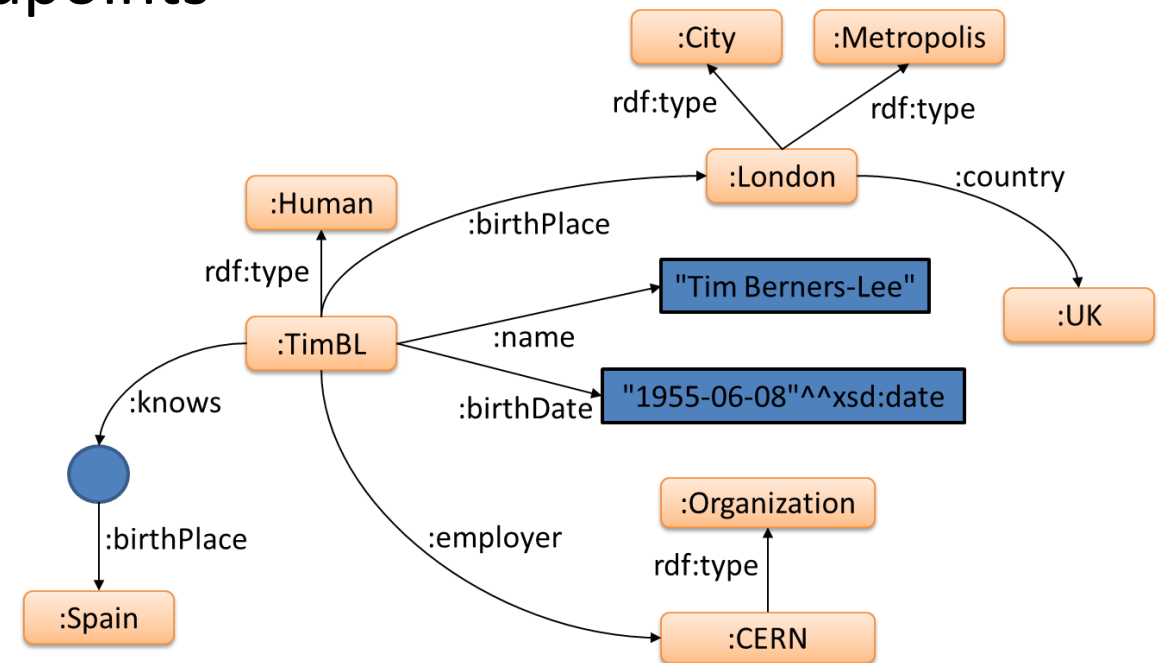
RDF ecosystem: SPARQL

SPARQL = RDF query language and protocol

Enables the creation of SPARQL endpoints

```
SELECT ?person ?date ?country WHERE {  
  ?person :birthDate ?date .  
  ?person :birthPlace ?place .  
  ?place :country ?country  
}
```

?person	?date	?country
:timbl	1955-06-08	:UK



Labeled Property graphs

Popular model in industry

neo4j, Amazon Neptune, Microsoft, etc

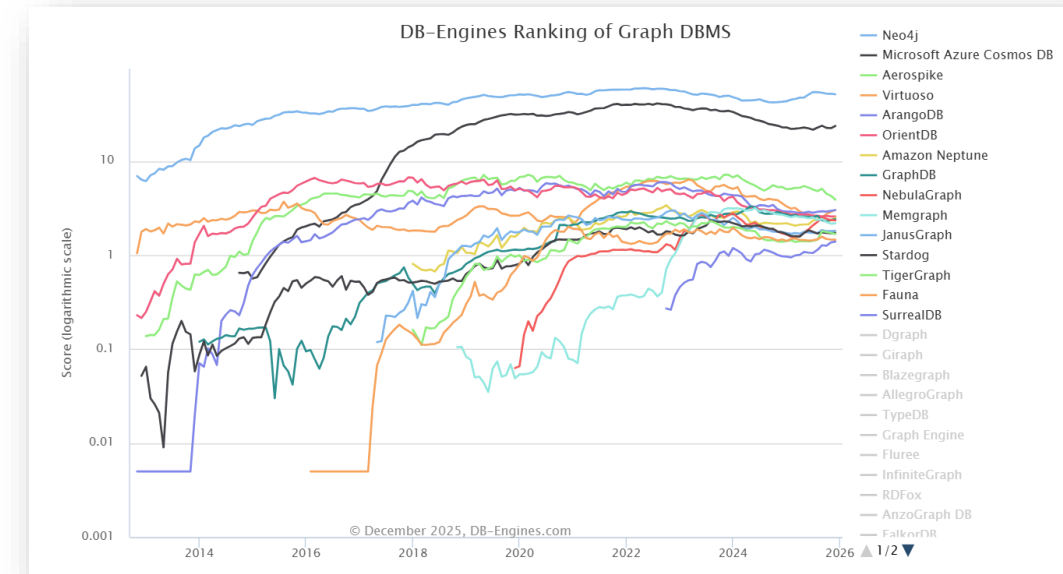
GQL (Graph Query Language)

Published in 2024: ISO/IEC FDIS 39075

Developed by the “SQL” committee

Influenced by Cypher, PGSchema, etc.

Also adopted to implement Knowledge Graphs



Source: https://db-engines.com/en/ranking_trend/graph+dbms

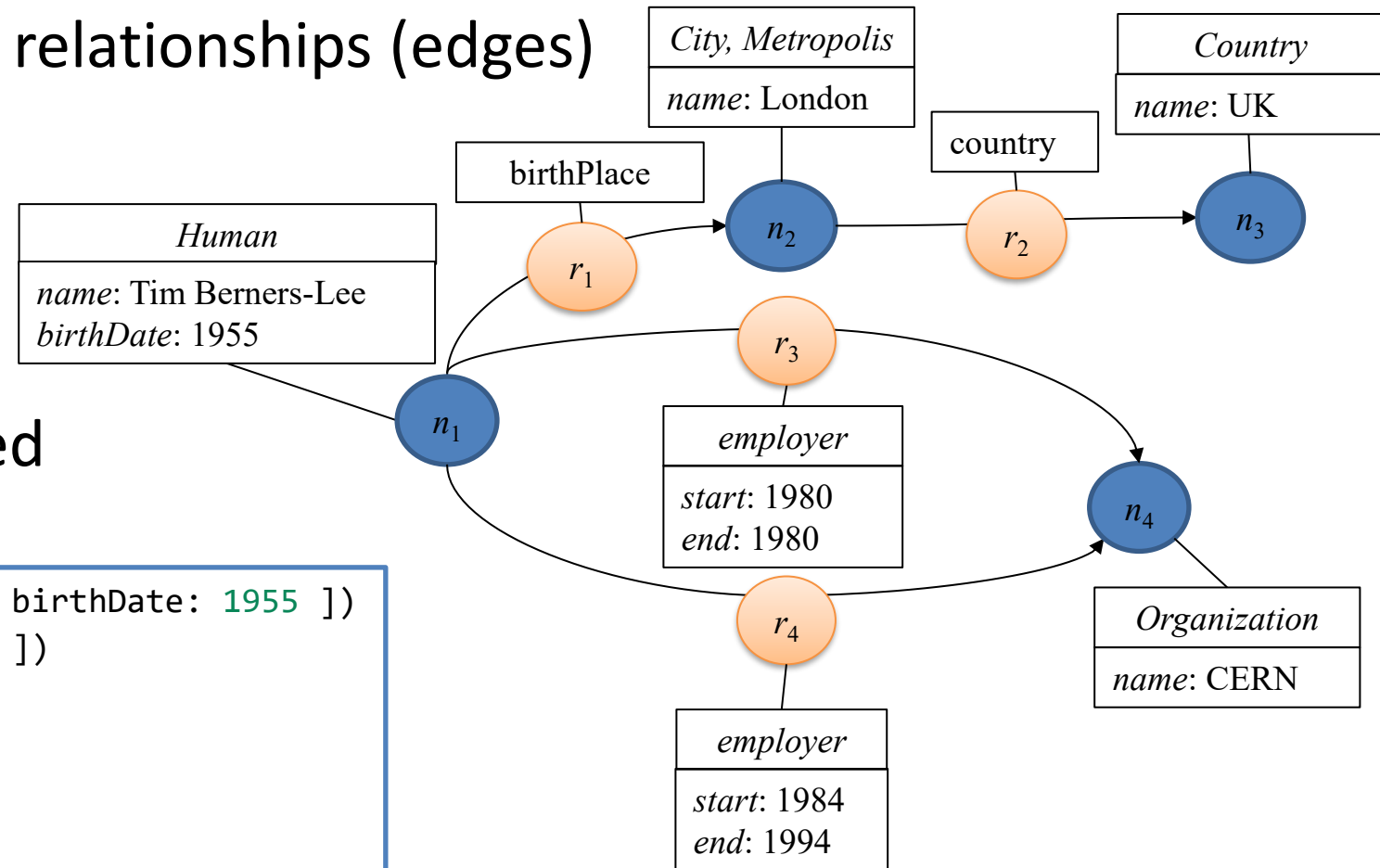
Property graphs

Graph structure with nodes and relationships (edges)

Nodes and edges can have:

- Labels
- A set of property-value pairs

Edges can be directed/undirected



```
(n1 {Person} [ name: "Tim Berners-Lee", birthDate: 1955 ] )
(n2 {City, Metropolis} [ name: "London" ] )
(n3 {Country} [ name: "UK" ] )
(n4 {Organization} [ name: "CERN" ] )
```

```
(n1) - (r1 {birthPlace}) -> (n2)
(n2) - (r2 {country}) -> (n3)
(n1) - (r3 {employer} [ start: 1980, end: 1980 ]) -> (n4)
(n1) - (r4 {employer} [ start: 1984, end: 1994 ]) -> (n4)
```

YarsPG
notation

Try it: <https://tinyurl.com/yw53uxtm>

Wikidata based Knowledge Graphs

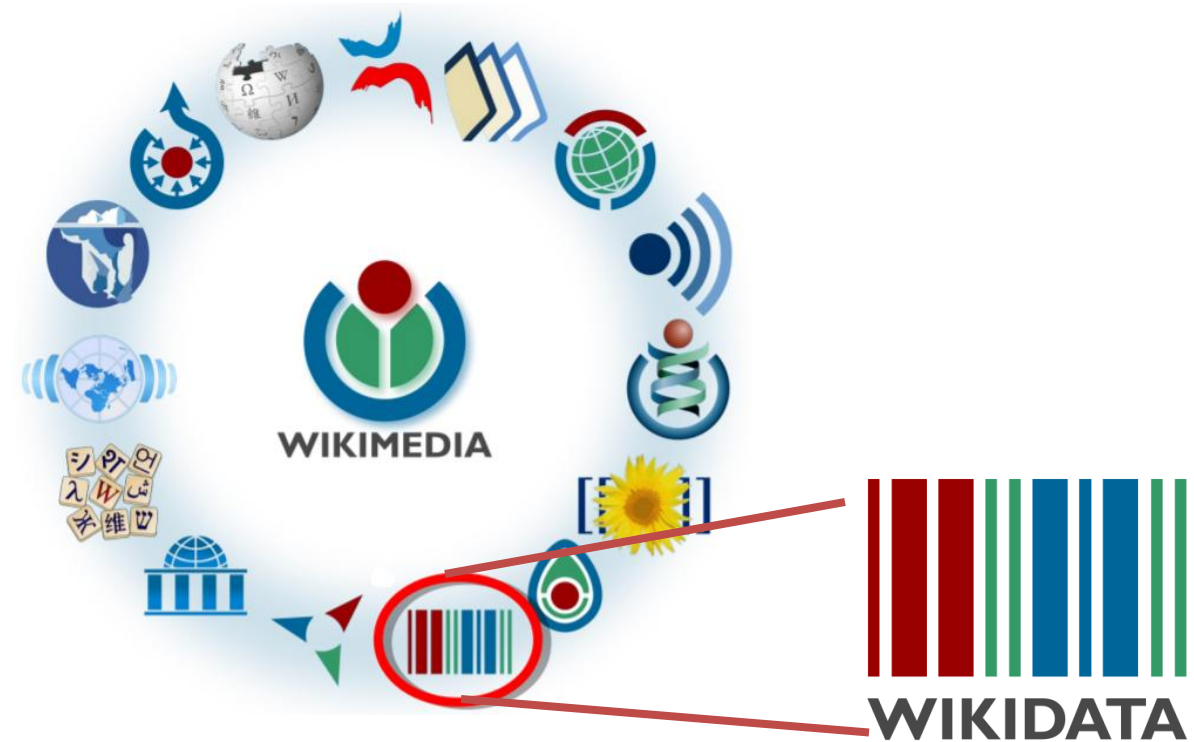
Wikidata was created in 2012 as a collaborative knowledge graph

<https://www.wikidata.org/>

Developed and supported by Wikimedia Deutschland

Initial goal:

Support multilingual infoboxes in Wikipedia



English Wikipedia page of Tim Berners-Lee

<http://www.wikidata.org/entity/Q80>

Bihari Wikipedia of Tim Berners-Lee

Wikidata

Wikidata as a commons Knowledge Graph

123,011,796 items, 2,535,483,822 edits (08/2026):

<https://www.wikidata.org/wiki/Wikidata:Statistics>

Free and open license: CC0

Multilingual

Collaborative (Humans and bots)

Public SPARQL endpoint

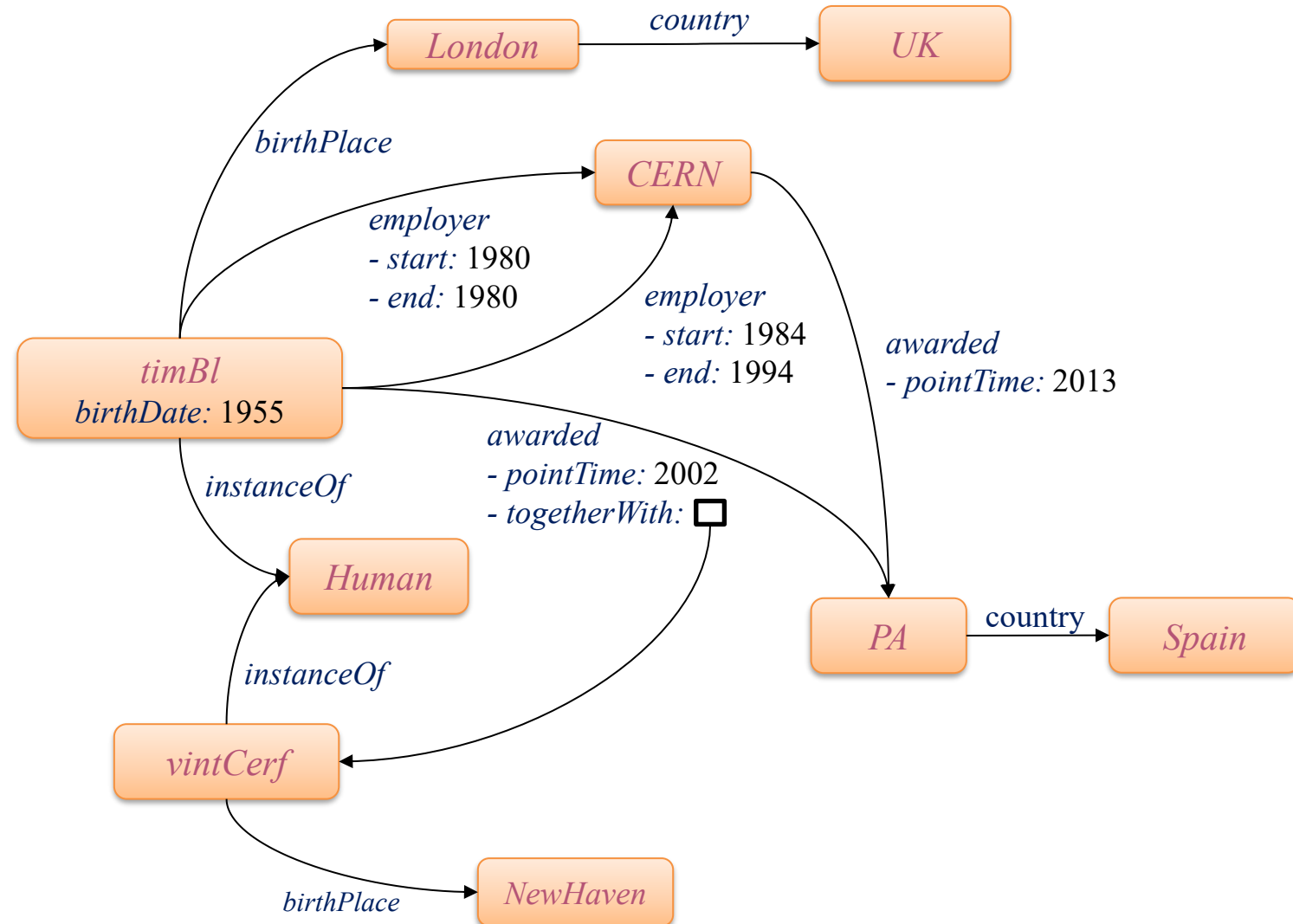
Software that supports Wikidata = Wikibase



Example from Wikidata

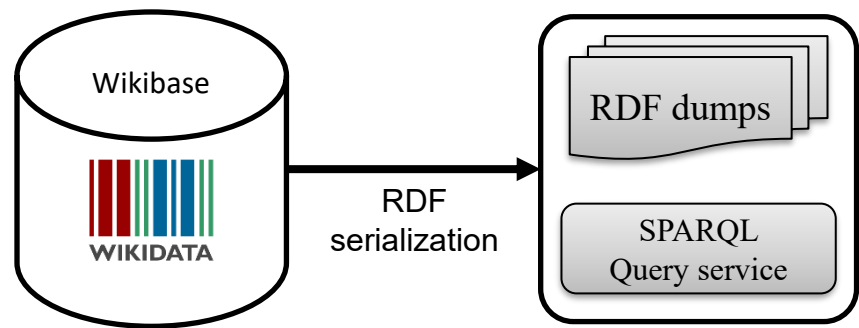
Information about Tim Berners Lee

<http://www.wikidata.org/entity/Q80>



Wikibase and RDF

Query service available: SPARQL endpoint



```
SELECT ?name ?date?country WHERE {  
  wd:Q80 wdt:P1559 ?name .  
  wd:Q80 wdt:P569 ?date .  
  wd:Q80 wdt:P19 ?place .  
  ?place wdt:P17 ?country  
}
```

?name	?date	?country
Tim Berners-lee	1955-06-08	:UK

RDF 1.2

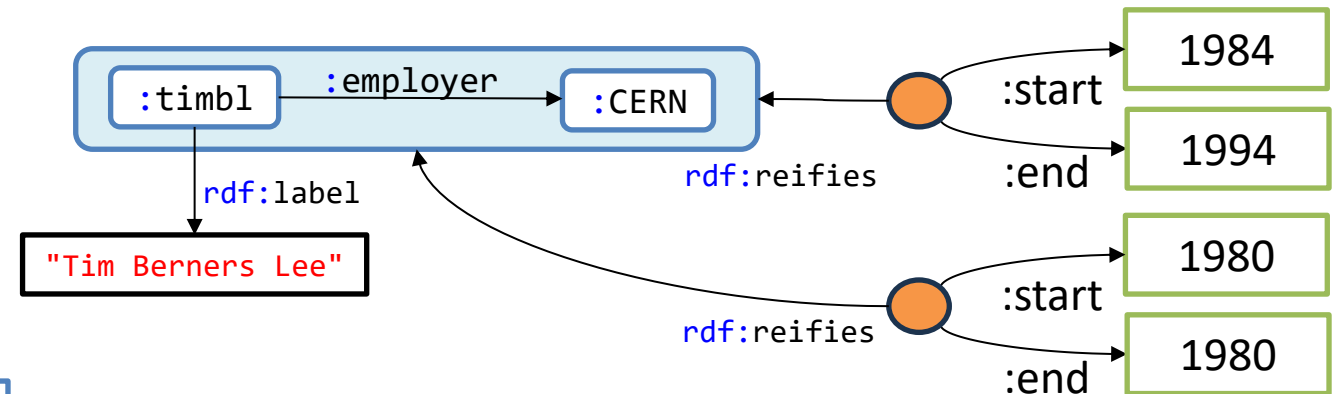
Under development (almost done): <https://www.w3.org/TR/rdf12-concepts/>
Add statements about triples (called triple terms)

Reifiers

```
:timbl rdfs:label "Tim Berners Lee" ;  
      :employer :CERN  
      { | :start "1984" ;  
          :end   "1994" | }  
      { | :start "1980" ;  
          :end   "1980" | } .
```



```
:timbl rdfs:label "Tim Berners Lee" ;  
      :employer :CERN .  
_:r rdf:reifies <<( :timbl :employer :CERN )>> .  
_:r :start "1984" ;  
    :end   "1994" .  
_:s rdf:reifies <<( :timbl :employer :CERN )>> .  
_:s :start "1980" ;  
    :end   "1980" .
```



Note:

In this example, we are asserting that Tim's employer is CERN

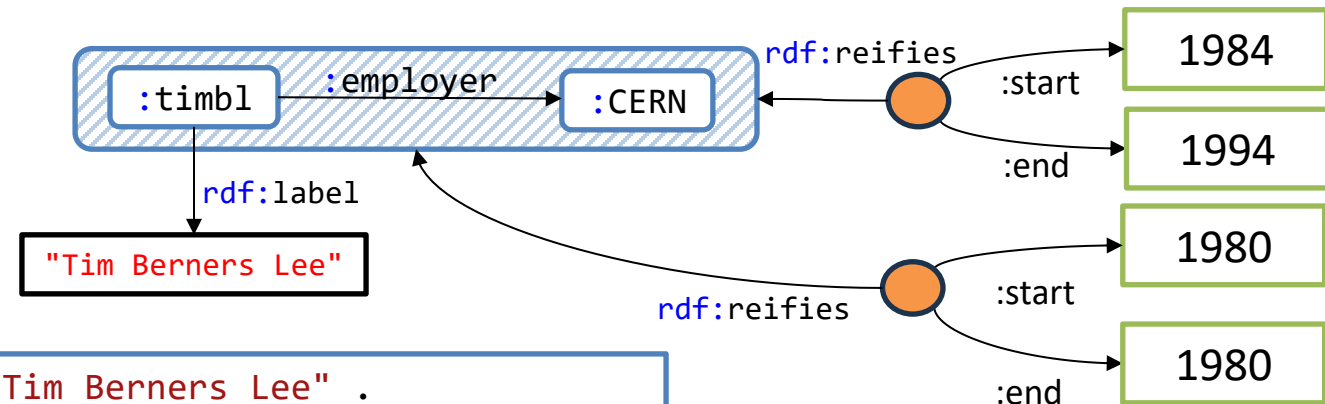
RDF 1.2: Non-asserted triple terms

Triple terms can be reified without being asserted

We can talk about a triple that says Tim's employer is the CERN

But we don't need to assert whether Tim's employer is the CERN or not

```
:timbl rdfs:label "Tim Berners Lee" .  
<< :timbl :employer :CERN >>  
  :start "1984" ;  
  :end   "1994" .  
<< :timbl :employer :CERN >>  
  :start "1980" ;  
  :end   "1980" .
```



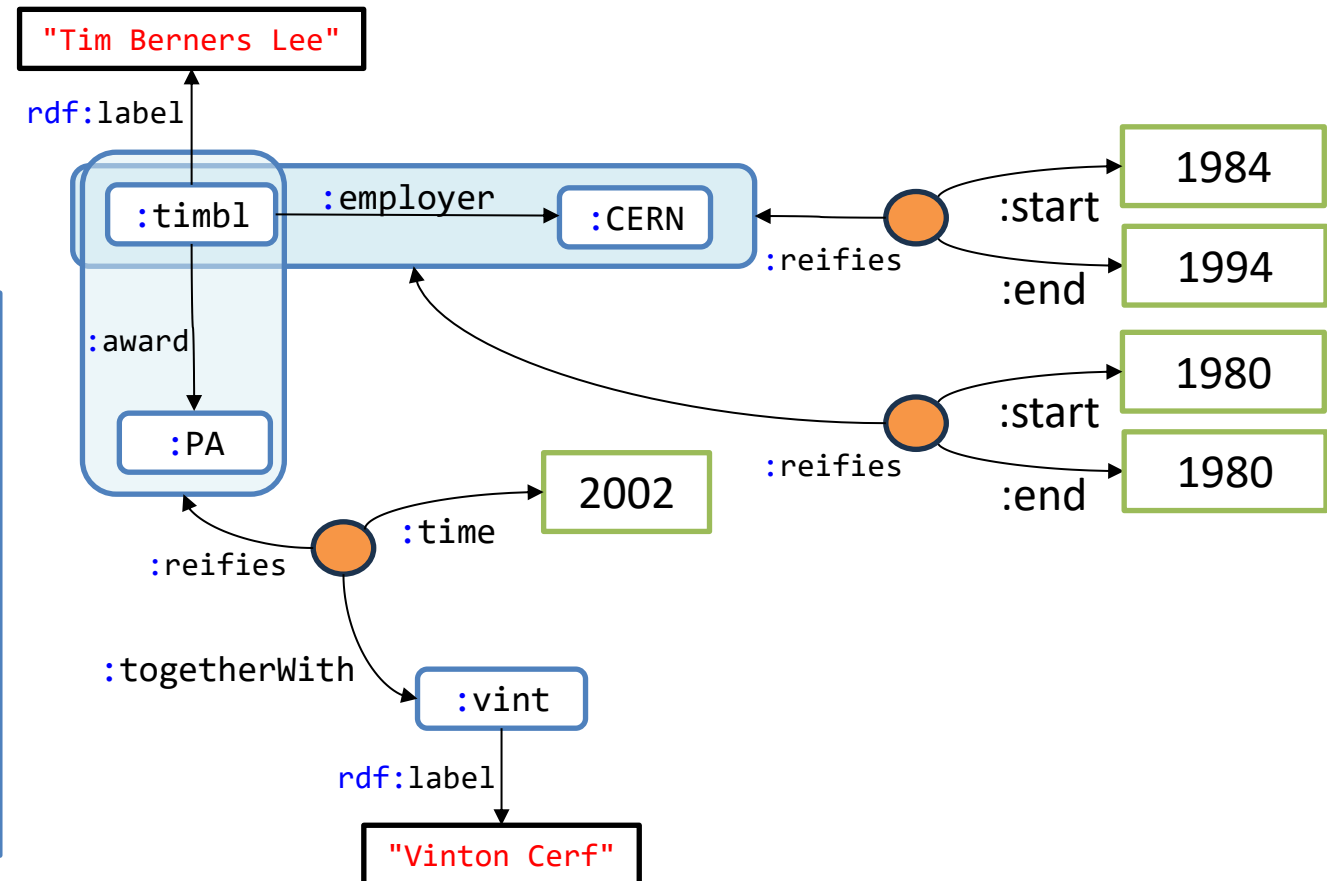
```
:timbl rdfs:label "Tim Berners Lee" .  
_:1 rdf:reifies <<( :timbl :employer :CERN )>> ;  
  :start "1984" ;  
  :end   "1994" .  
_:2 rdf:reifies <<( :timbl :employer :CERN )>> ;  
  :start "1980" ;  
  :end   "1980" .
```

RDF 1.2

RDF 1.2 also supports Wikibase style models

```
:timbl rdfs:label "Tim Berners Lee" ;  
      :employer :CERN { | :start      1984 ;  
                          :end        1994 | }  
                        { | :start      1980 ;  
                          :end        1980 | } ;  
      :award :PA { | :time      2002 ;  
                    :togetherWith :vint | } .  
:vint rdfs:label "Vinton Cerf" .
```

```
:timbl rdfs:label "Tim Berners Lee" ;  
      :employer :CERN ; :award :PA.  
_:r rdf:reifies <<(:timbl :employer :CERN )>> ;  
   :start      "1984" ;  
   :end        "1994" .  
_:s rdf:reifies <<(:timbl :employer :CERN )>> ;  
   :start      "1980" ;  
   :end        "1980" .  
_:t rdf:reifies <<(:timbl :award :PA )>> ;  
   :time      "2002" ;  
   :togetherWith :vint .  
:vint rdfs:label "Vinton Cerf" .
```

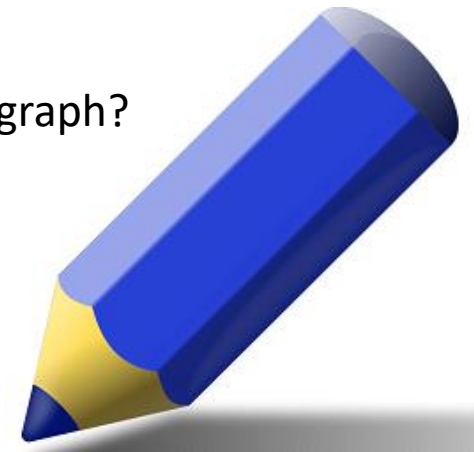


RDF 1.2

And things that Wikibase can not support, like qualifying qualifiers...

```
:timbl rdfs:label "Tim Berners Lee" ;
  :employer :CERN { | :start "1984"          { | :accordingTo :wikidata { | :statedIn "2024-05-02"^^xsd:date | } | } ;
                  | :end  "1994"          { | :accordingTo :wikidata { | :statedIn "2024-05-02"^^xsd:date | } | } ;
                  | }
  { | :start "1980"          { | :accordingTo :wikidata { | :statedIn "2024-05-01"^^xsd:date | } | } ;
    | :end  "1980"          { | :accordingTo :wikidata { | :statedIn "2024-05-01"^^xsd:date | } | } ;
    | }
  :award      :PA { | :year 2002          { | :accordingTo :wikidata { | :statedIn "2024-05-01"^^xsd:date | } | } ;
                  | :together_with :vint { | :accordingTo :wikidata { | :statedIn "2024-05-01"^^xsd:date | } | } ;
                  | }
.
```

Do you want to draw that graph?



Try it: <https://tinyurl.com/y29rbt5y>

SPARQL 1.2

Current draft: <https://www.w3.org/TR/sparql12-query/>

rudof supports RDF 1.2 and SPARQL 1.2 (thanks to Oxigraph)

Example query:

```
PREFIX : <http://example.org/>

SELECT ?person ?employer ?start ?end WHERE {
  ?person :employer ?employer { | :start ?start ;
                                   :end   ?end
                                   | }
}
```

Contents

Introduction to Knowledge graphs

Types of Knowledge Graphs:

RDF, Property graphs, Wikibase, RDF-Star

Shaping RDF: ShEx & SHACL, DCTAP

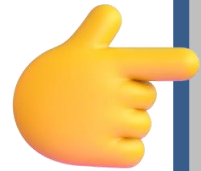
Shaping other types of Knowledge graphs:

Shaping Wikibase and Wikidata graphs

Shaping Property Graphs

Shaping RDF 1.2

Some ideas for future work





RDF, the good parts...

As an integration language

RDF as a *lingua franca* for semantic web and linked data

Basis for knowledge representation

Flexibility

Data can be adapted to multiple environments

Reusable data by default

Ecosystem

RDF data stores & SPARQL

Multiple serializations: Turtle, JSON-LD, RDF/XML...

Inference and ontologies





RDF, the other parts

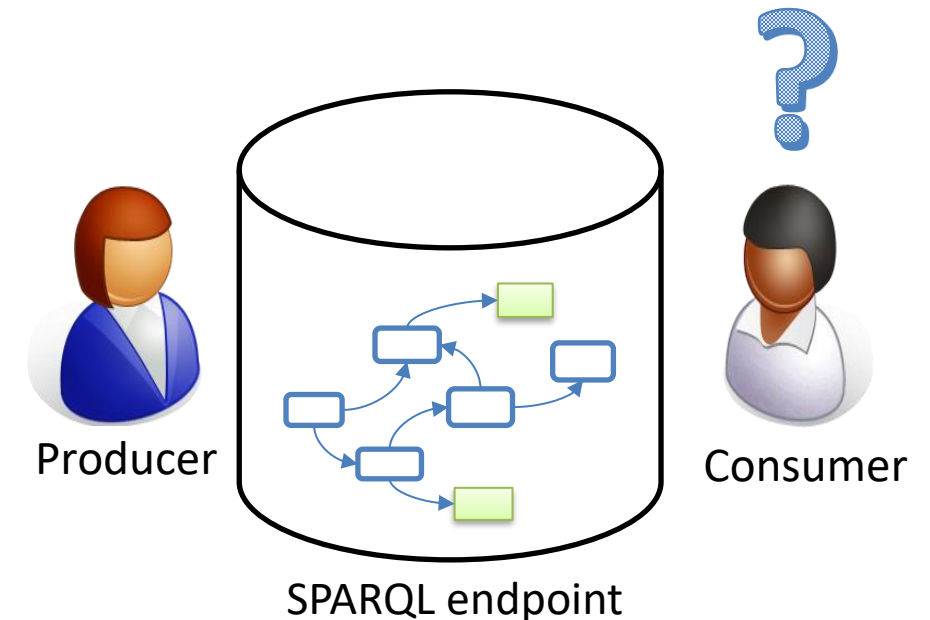
Consuming & producing RDF

Describing and validating RDF content

SPARQL endpoints are not well documented

Typical documentation = set of SPARQL queries

Difficult to know where to start doing queries





Why describe & validate RDF?

For producers

Developers can understand the contents they are going to produce

They can ensure they produce the expected structure

Advertise and document the structure

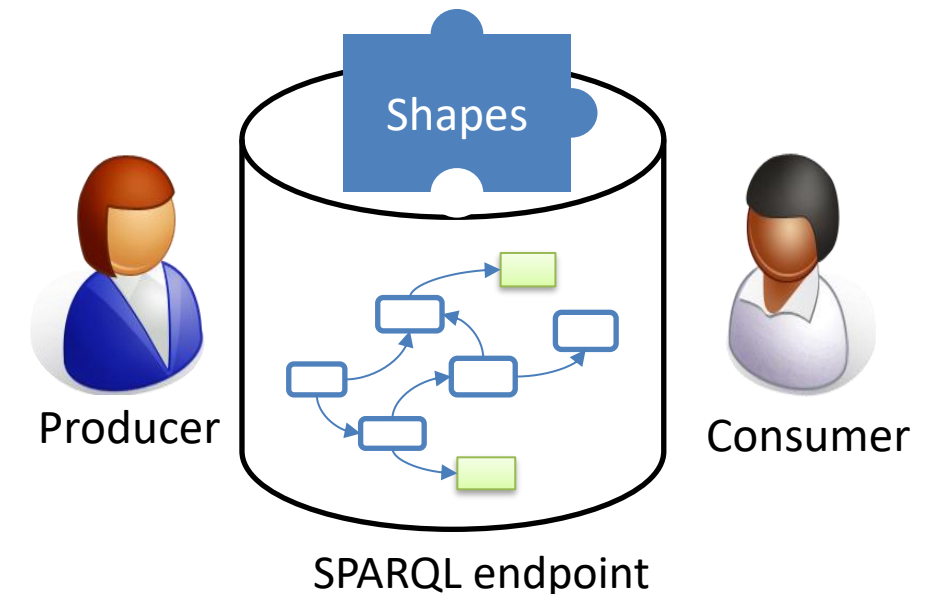
Generate interfaces

For consumers

Understand the contents

Verify the structure before processing it

Query generation & optimization



Running example: some errors?

Well formed RDF can contain mistakes

Usually, RDF processors don't complain

RDF is very flexible

A nice feature *"sometimes"*

```
:timbl a          :Human          ;  
       :name      "Tim Berners-Lee" ;  
       :knows     :bob             .  
  
:alice a          :Human          ;  
       :name      234              .  
  
:bob   a          :Human          ;  
       :name      "Bob", "Robert"  ;  
       :knows     :CERN            .  
  
:carol a          :Human          ;  
       :birthPlace :Dog            .  
  
:CERN  a          :Organization    ;  
       :name      "CERN"           .
```

Try it: <https://tinyurl.com/mscaxdkf>

ShEx & SHACL

2013 RDF Validation Workshop

Conclusions of the workshop:

There is a need of a higher level, concise language for RDF Validation

ShEx initially proposed (v 1.0)

2014 W3c Data Shapes WG chartered

2017 SHACL accepted as W3C recommendation

2017 ShEx 2.0 released as W3C Community group draft

2019 ShEx adopted by Wikidata

2025 IEEE ShEx (*work in progress*)

2025 Data Shapes WG chartered again for SHACL 1.2 (*work in progress*)

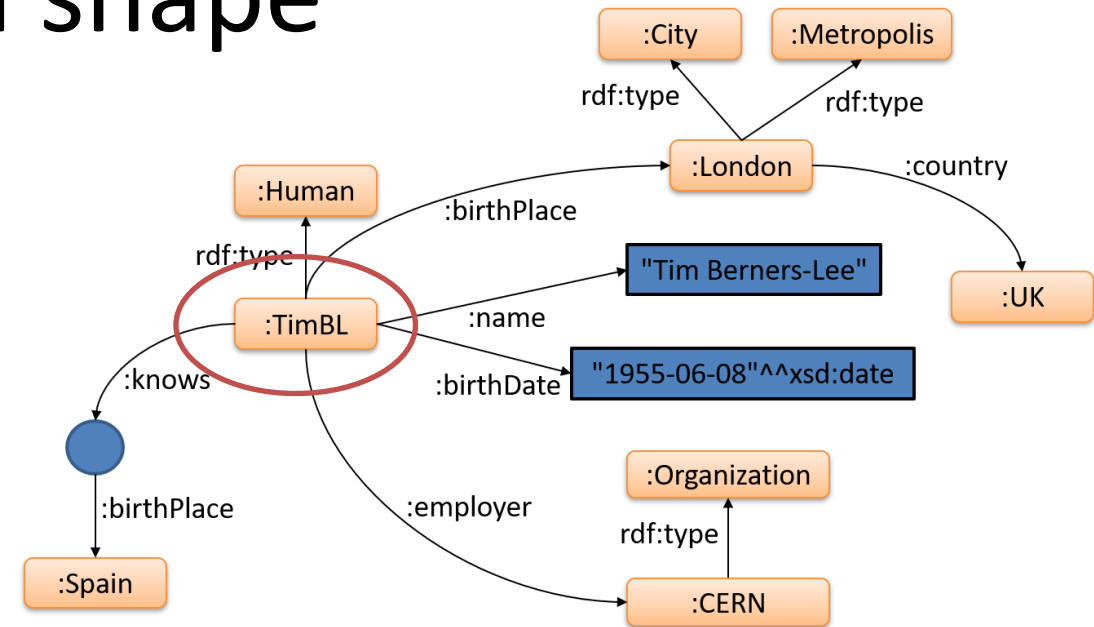
Example of a shape

A shape describes

The form of a node (node constraint)

Incoming/outgoing arcs from a node

Possible values associated with those arcs



RDF Node

```
:timbl :name "Tim Berners-Lee" ;  
       :birthPlace :london ;  
       :birthDate "1955-06-08"^^xsd:date ;  
       :employer :CERN .
```

ShEx

```
<Researcher> {  
  :name      xsd:string      ;  
  :birthPlace @<Place>      ? ;  
  :birthDate xsd:date        ? ;  
  :employer  @<Organization> * ;  
}
```



Try it: <https://tinyurl.com/5mvnp47r>



Shape Expressions - ShEx

Goal: Concise and human-readable language

Semantics inspired by regular expressions

3 syntaxes:

ShExC: Compact syntax, similar to Turtle or SPARQL

ShExJ: JSON(-LD), for the spec

ShExR: RDF, based on JSON-LD

Note: Round tripping is possible, convert from one to the other



ShEx libraries and demos

Implementations & libraries:

[shex.js](#): Javascript

[Jena-ShEx](#): Java

[SHaclEX](#): Scala (Jena/RDF4j)

[PyShEx](#): Python

[shex-java](#): Java

[Ruby-ShEx](#): Ruby

[RDF-Elixir](#): Elixir

[rudof](#): Rust

Online demos & playgrounds

[ShEx-simple](#)

[RDFShape](#)

[Wikishape](#)

[ShEx-Java](#)

[ShExValidata](#)

More info: <http://shex.io>

ShEx in rudof

rudof supports latest ShEx version of ShEx

3 syntaxes (Compact, RDF, JSON) can roundtrip

ShEx schemas visualization as UML like diagrams

Validation with RDF data in-memory or with SPARQL endpoints

Some special modes for Wikibase

Support for RDF 1.2

Recursion semantics:

Greatest Fixpoint (Permissive)



Simple example

Prefix
declarations
as in
Turtle/SPARQL

```
prefix :      <http://example.org/>
prefix xsd:    <http://www.w3.org/2001/XMLSchema#>
prefix schema: <http://schema.org/>

:Person {
  :name      xsd:string      ;
  :knows     @:Person       *
}
```

Nodes conforming to `:Person` must

- Have property `:name` with a value of type `xsd:string` (exactly one)
- Have property `:knows` with values conforming to `:Person` (zero or more)

RDF Validation using ShEx

RDF Data

Schema

```
:Person {  
  :name      xsd:string      ;  
  :knows @:Person      *  
}
```

Shape map

```
:a@:Person, ✓  
:b@:Person, ✓  
:c@:Person, ✗  
:d@:Person, ✗  
:e@:Person, ✗  
:f@:Person  ✗
```

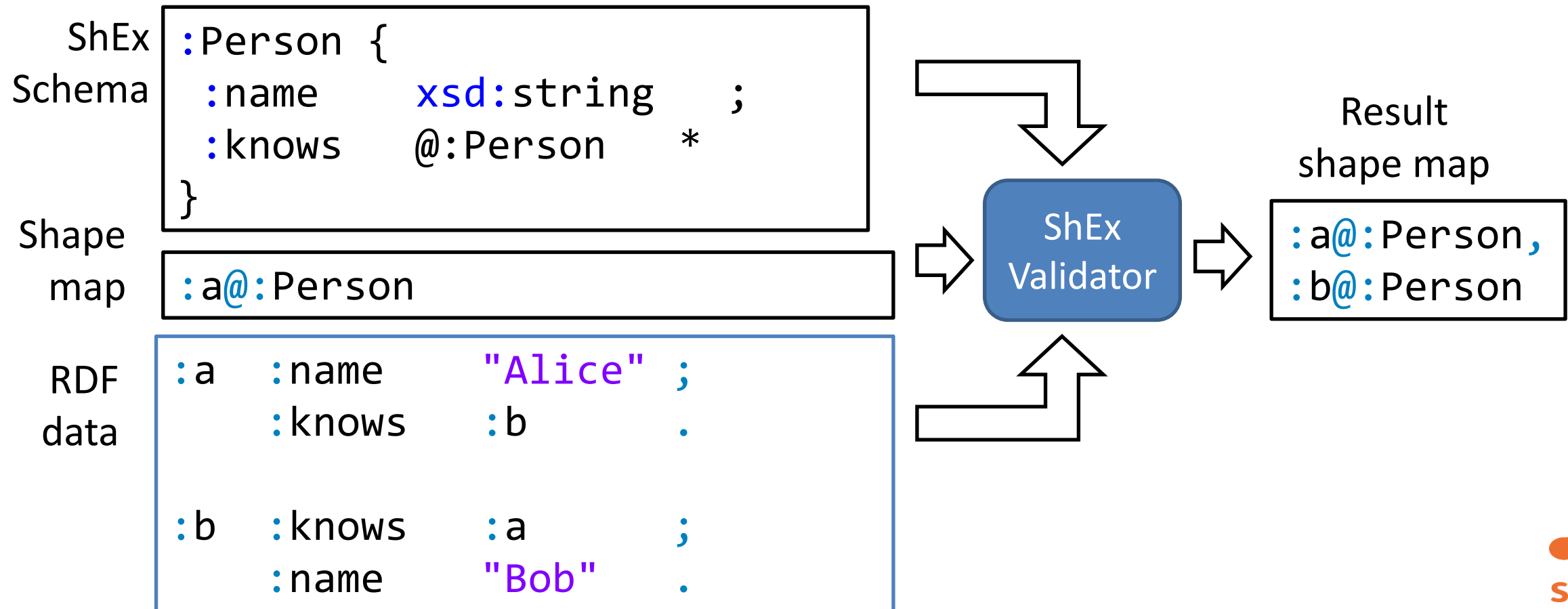
```
:a  :name      "Alice"      ;  
    :knows     :b           .  
:b  :knows     :a           ;  
    :name      "Bob"        .  
:c  :name      "Carol", "Carole" .  
:d  :name      234          .  
:e  :namme     "Emily"      .  
:f  :name      "Frank"      ;  
    :email     "frank@mail.com" ;  
    :knows     :a, :c       .
```

Try it ([rudof](#))Try it ([Simple ShExDemo](#))

Validation process

Input: RDF data, ShEx schema, Shape map

Output: Result shape map



Node constraints

Constraints over a node (without considering its neighbourhood)

```
:Book {  
  :name          xsd:string  
  :datePublished xsd:date  
  :numberOfPages MinInclusive 1  
  :author        @:Person  
  :genre          [ :Fiction :NonFiction ]  
  :isbn           /isbn:[0-9X]{10}/  
  :publisher      IRI  
  :audio          .  
  :maintainer     @:Person OR  
                  @:Organization  
}  
:Person {}  
:Organization {}
```

```
:item23  
  :name          "Weaving the Web" ;  
  :datePublished "2012-03-05"^^xsd:date ;  
  :numberOfPages 272 ;  
  :author        :timbl ;  
  :genre          :NonFiction ;  
  :isbn           "isbn:006251587X" ;  
  :publisher      <http://www.harpercollins.com/> ;  
  :audio          <http://audio.com/item23> ;  
  :maintainer     :alice .
```

Cardinalities

Inspired by regular expressions: +, ?, *, {m,n}

By default {1,1}

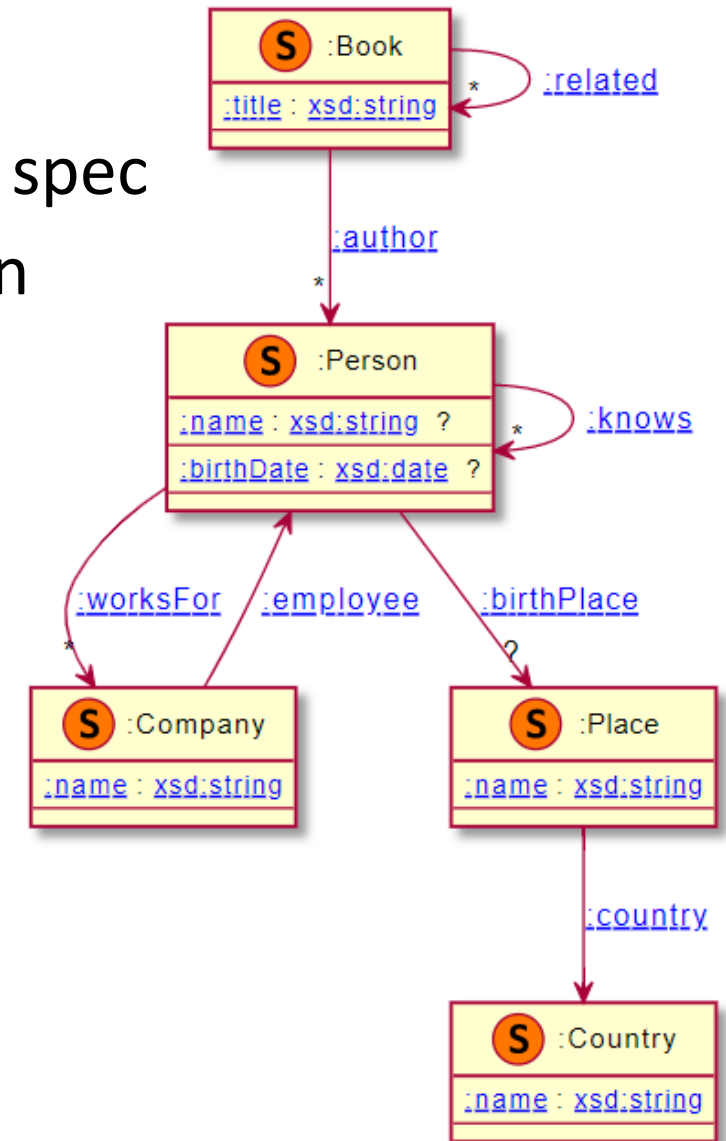
```
:Book {  
  :name          xsd:string          ;  
  :numberOfPages xsd:integer  ?      ;  
  :author        @:Person    +      ;  
  :publisher     IRI          ?      ;  
  :maintainer    @:Person    {1,3}  ;  
  :related       @:Book      *      ;  
}  
:Person {}  
:Organization {}
```

```
:item23  
  :name          "Weaving the Web"    ;  
  :numberOfPages 272                  ;  
  :author        :timbl, :markFischetti ;  
  :maintainer     :alice, :bob        .
```

Recursive schemas

Support for recursive (cyclic) data models is part of the spec
Well formed semantics based on stratified negation

```
:Book    { :title      xsd:string    ;  
           :author     @:Person     * ;  
           :related     @:Book       * }  
:Person  { :name       xsd:string    ? ;  
           :birthDate   xsd:date     ? ;  
           :birthPlace  @:Place      ? ;  
           :knows       @:Person     * ;  
           :worksFor    @:Company    * }  
:Place   { :name       xsd:string    ;  
           :country     @:Country    }  
:Country { :name       xsd:string    }  
:Company { :name       xsd:string    ;  
           :employee    @:Person     }
```



Open/Closed content models

RDF semantics mostly presume open content models

Shape expressions are open by default

Enable extensibility

But...some use cases require closed content models

Added CLOSED keyword

```
:Book {  
  :name    xsd:string    ;  
}
```

```
:Book CLOSED {  
  :name    xsd:string    ;  
}
```



```
:a :name "Weaving the web" ;  
   :isbn "006251587X" .
```

Open/Closed properties

Property values are closed by default (closed properties)

```
:Book {  
  :code /isbn:[0-9X]{10}/ ;  
}
```



```
:item23 :code "isbn:006251587X" .
```



```
:item23 :code 23 .
```

Properties can be repeated

```
:Book {  
  :code /isbn:[0-9X]{10}/ ;  
  :code /isbn:[0-9]{13}/  
}
```



```
:item23 :code "isbn:006251587X" ,  
        :code "isbn:9780062515872" .
```

EXTRA declares properties as open

```
:Book EXTRA :code {  
  :code /isbn:[0-9X]{10}/ ;  
}
```



```
:item23 :code "isbn:006251587X" ,  
        :code 23 .
```

Triple expressions

“Unordered” regular expressions: Regular bag expressions

```
:Person {  
  ( :name      xsd:string |  
    :firstName xsd:string + ;  
    :lastName  xsd:string  
  ) ;  
  :birthDate  xsd:date    ?  
}
```

(name |
 firstName + ;
 lastName
) ;
birthDate ?

(n | f + ; l) ; b?

```
:alice :name      "Alice Cooper"      ;  
       :birthDate "2010-02-23"^^xsd:date .  
  
:bob   :firstName "Robert"            ;  
       :lastName  "Smith"             .  
  
:carol :firstName "Carol"             ;  
       :lastName  "King"              ;  
       :firstName "Carole"            .
```

```
:alice → n  
:bob   → f l  
:carol → f l f
```

```
:dave → n f
```

```
:dave :name      "Dave Navarro"      ;  
      :firstName "Dave"              .
```

Logical operators

Shape Expressions can be combined with **AND**, **OR**, **NOT**

```
:Book {  
  :name    xsd:string ;  
  :author  @:Person OR @:Organization ;  
}  
  
:AudioBook @:Book AND {  
  :name           MaxLength 20 ;  
  :readBy         @:Person      ;  
} AND NOT {  
  :numberOfPages . +  
}  
  
:Person {}  
:Organization {}
```

```
:item24 :name      "Weaving the Web" ;  
        :author    :timbl             ;  
        :readBy    :timbl             .
```

```
:item23 :name      "Weaving the Web" ;  
        :author    :timbl             ;  
        :numberOfPages 272            ;  
        :readBy    :timbl             .
```

ShEx allows **NOT** combined with recursion (semantics based on stratified negation)

Importing schemas

`import` statement can be used to import schemas

<http://validatingrdf.com/tutorial/examples/book.shex>

```
:Book {  
  :title   xsd:string   ;  
}  
:Person {  
  :name    xsd:string ? ;  
}
```

```
import <https://www.validatingrdf.com/examples/book.shex>
```

```
:AudioBook @:Book AND {  
  :title      MaxLength 20 ;  
  :readBy     @:Person    ;  
}
```

```
:item24 :name      "Weaving the Web" ;  
        :author    :timbl             ;  
        :readBy     :timbl             .
```

Machine processable annotations

Annotations based on RDF

Lots of applications, e.g. generate forms from shapes

```
prefix schema: <http://schema.org/>
prefix : <http://example.org/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix ui: <http://www.w3.org/ns/ui#>

start = @:User

:User {
  schema:name      xsd:string           // ui:label "Name"      ;
  schema:birthDate xsd:dateTime         // ui:label "Birth date" ;
  schema:gender    [ schema:Male schema:Female ] // ui:label "Gender"
}
```

A screenshot of a web form generated from the SHEx shape. The form is titled ":User" and contains three input fields: "Name:" with the value "Alice", "Birth date:" with the value "23/04/2010", and "Gender:" with a dropdown menu showing "schema:male". A blue "Check" button is at the bottom.

Try it:
[Eric's demo](#)

3 syntaxes: ShExC, ShExJ, ShExR

ShExC

```
prefix :      <http://example.org/>
prefix xsd:   <http://www.w3.org/2001/XMLSchema#>
prefix schema: <http://schema.org/>

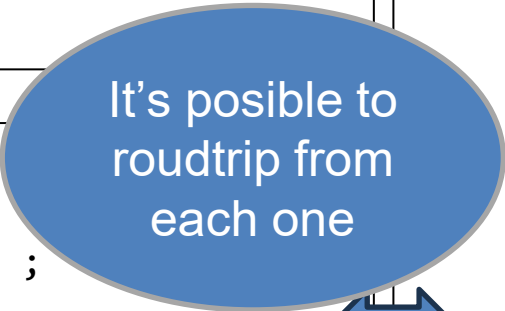
:Book {
  schema:name    xsd:string    ;
  :related      @:Book      *
}
```

ShExR (RDF, Turtle)

```
:Book a sx:ShapeDecl ;
sx:shapeExpr [ a sx:Shape ;
  sx:expression [ a sx:EachOf ;
    sx:expressions (
      [ a sx:TripleConstraint ;
        sx:predicate schema:name ;
        sx:valueExpr [ a sx:NodeConstraint ;
          sx:datatype xsd:string
        ] ]
      [ a sx:TripleConstraint ;
        sx:predicate :related ;
        sx:valueExpr [ a sx:NodeConstraint ;
          sx:valueExpr :Book ] ] ) ] ] ] .
```

ShExJ (JSON LD)

```
{ "type" : "Schema",
  "@context" : "http://www.w3.org/ns/shex.jsonld",
  "shapes" : [ {
    "type" : "Shape",
    "id" : "http://example.org/Book",
    "expression" : {
      "type" : "EachOf",
      "expressions" : [ {
        "type" : "TripleConstraint",
        "predicate" : "http://schema.org/name",
        "valueExpr" : {
          "type" : "NodeConstraint",
          "datatype" : "http://www.w3.org/2001/XMLSchema#string"
        }
      },
      { "predicate" : "http://example.org/related",
        "valueExpr" : "http://example.org/Book",
        "min" : 0,
        "max" : -1,
        "type" : "TripleConstraint"
      }
    ]
  } ] }
```



More complex Shape Maps

Shape Maps define which nodes validate with which shapes

Examples:

```
{FOCUS a :Person}@:Person
```

```
{FOCUS :name _}@:Person
```

```
SPARQL ""  
  PREFIX : <http://example.org/>  
  SELECT ?person WHERE {  
    ?person :name ?name;  
             :knows :a  
  }""@:Person
```

Inheritance model for ShEx

extends allows to reuse existing shapes adding new content

Handles closed properties and shapes

```
:Book {  
  :name      xsd:string ;  
  :author    @:Person   ;  
  :code      /isbn:[0-9]{13}/ ;  
  :code      /isbn:[0-9X]{10}/  
}  
  
:LibraryBook extends @:Book {  
  :code      /internal:[0-9]*/ ;  
}
```

```
:item23 :name      "Weaving the Web" ;  
        :author    :timbl             ;  
        :code      "isbn:006251587X"  ;  
        :code      "isbn:9780062515872" ;  
        :code      "internal:234"     .
```

Other features

Multiple inheritance

Abstract shapes

More details:

Shape Expressions with Inheritance, Iovka Boneva, Jose Emilio Labra Gayo, Eric Prud'hommeaux, Katherine Thornton, Andra Waagmeester, *Extended Semantic Web Conference*, Portoroz, Slovenia, 2025, https://labra.weso.es/publication/2025_shex_inheritance/

Other shexy features

Semantic actions

ShEx is an RDF description, validation and transformation language

New extensions and work in progress

RDF 1.2

Arrays

Graphs

SHACL

W3C recommendation: <https://www.w3.org/TR/shacl/> (July 2017)

Inspired by SPIN, OSLC & ShEx

2 parts: SHACL-Core, SHACL-SPARQL

Syntax = RDF vocabulary

Some SHACL implementations

Name	Parts	Language - Library	Comments
Topbraid SHACL API	SHACL Core, SPARQL	Java (Jena)	Used by TopBraid composer
SHACL playground	SHACL Core	Javascript (rdflib.js)	http://shacl.org/playground/
SHACL-S (SHacLEX)	SHACL Core	Scala (Jena, RDF4j)	http://rdfshape.weso.es
pySHACL	SHACL Core, SPARQL	Python (rdflib)	https://github.com/RDFLib/pySHACL
Corese SHACL	SHACL Core, SPARQL	Java (STTL)	http://wimmics.inria.fr/corese
RDFUnit	SHACL Core, SPARQL	Java (Jena)	https://github.com/AKSW/RDFUnit
Jena SHACL	SHACL Core, SPARQL	Java (Jena)	https://jena.apache.org/
RDF4j SHACL	SHACL Core	Java (RDF4J)	https://rdf4j.org
Stardog	SHACL Core, SPARQL	Java	https://www.stardog.com
Zazuko SHACL	SHACL Core	Javascript	https://github.com/zazuko/rdf-validate-shacl
maplib	SHACL Core	Rust	https://github.com/DataTreehouse/maplib
rudof 	SHACL core (in progress)	Rust	https://rudof-project.github.io/

SHACL in rudof

rudof implements SHACL 1.0 (SHACL Core and SHACL-SPARQL)

Validation in-memory and with SPARQL endpoints

SHACL 1.2

RDF 1.2 related features implemented

Node expressions (Work in progress)

Rules (pending)

Information about positive results (evidences)

3 semantics for SHACL recursion

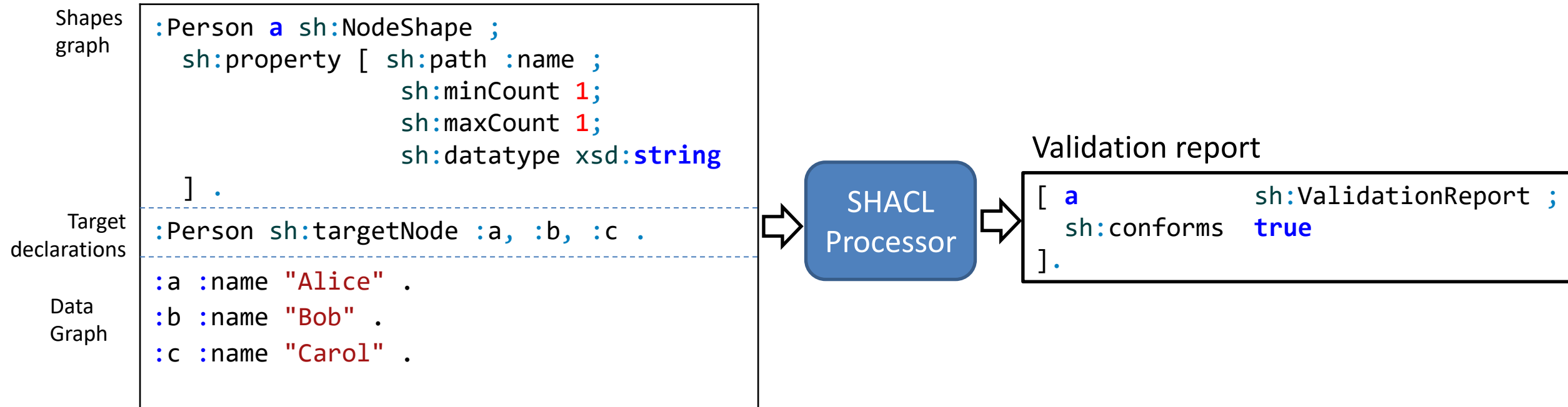
no recursion

lfp (least fixpoint, conservative)

gfp (greatest fixpoint, permissive)



SHACL processor



Validation Report

The output of the validation process is a list of violation errors

No errors $\Rightarrow$ RDF conforms to shapes graph

```
[ a          sh:ValidationReport ;  
  sh:conforms true  
].
```

```
[ a          sh:ValidationReport ;  
  sh:conforms false ;  
  sh:result  [  
    a          sh:ValidationResult ;  
    sh:focusNode :b ;  
    sh:message  
      "MinCount violation. Expected 1, obtained: 0" ;  
    sh:resultPath schema:name ;  
    sh:resultSeverity sh:Violation ;  
    sh:sourceConstraintComponent  
      sh:MinCountConstraintComponent ;  
    sh:sourceShape :hasName  
  ] ;  
  ...
```

Basic example

```
prefix :      <http://example.org/>
prefix sh:    <http://www.w3.org/ns/shacl#>
prefix xsd:   <http://www.w3.org/2001/XMLSchema#>
prefix schema: <http://schema.org/>
```

```
:UserShape a sh:NodeShape ;
  sh:targetNode :alice, :bob, :carol ;
  sh:nodeKind sh:IRI ;
  sh:property :hasName,
              :hasEmail .
:hasName sh:path schema:name ;
  sh:minCount 1;
  sh:maxCount 1;
  sh:datatype xsd:string .
:hasEmail sh:path schema:email ;
  sh:minCount 1;
  sh:maxCount 1;
  sh:nodeKind sh:IRI .
```

Shapes graph

```
:alice schema:name "Alice Cooper" ;
        schema:email <mailto:alice@mail.org> .

:bob    schema:firstName "Bob" ;
        schema:email <mailto:bob@mail.org> . ☹️

:carol  schema:name "Carol" ;
        schema:email "carol@mail.org" . ☹️
```

Data graph

Try it. RDFShape <https://tinyurl.com/y46b2f8q>

Same example with blank nodes

```
prefix :      <http://example.org/>
prefix sh:    <http://www.w3.org/ns/shacl#>
prefix xsd:   <http://www.w3.org/2001/XMLSchema#>
prefix schema: <http://schema.org/>
```

```
:UserShape a sh:NodeShape ;
  sh:targetNode :alice, :bob, :carol ;
  sh:nodeKind sh:IRI ;
  sh:property [
    sh:path      schema:name ;
    sh:minCount  1; sh:maxCount 1;
    sh:datatype  xsd:string ;
  ] ;
  sh:property [
    sh:path      schema:email ;
    sh:minCount  1; sh:maxCount 1;
    sh:nodeKind  sh:IRI ;
  ] .
```

```
:alice schema:name "Alice Cooper" ;
       schema:email <mailto:alice@mail.org> .

:bob   schema:firstName "Bob" ;
       schema:email <mailto:bob@mail.org> . ☹️

:carol schema:name "Carol" ;
       schema:email "carol@mail.org" . ☹️
```

Data graph

Shapes graph

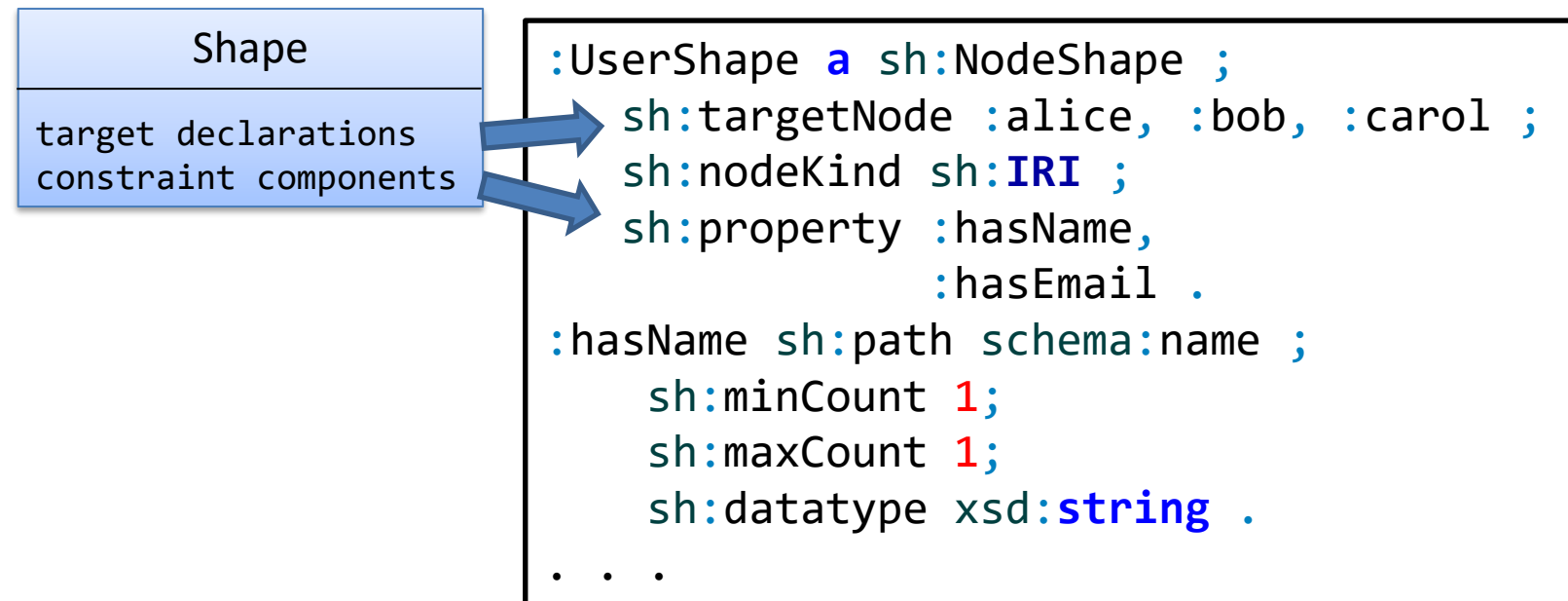
Try it. RDFShape <https://tinyurl.com/y4ycv2vn>

Some definitions about SHACL

Shape: collection of targets and constraints components

Targets: specify which nodes in the data graph must conform to a shape

Constraint components: Determine how to validate a node



Shapes graph and data graph

Conceptually: 2 graphs

Shapes graph: an RDF graph that contains shapes

Data graph: an RDF graph that contains data to be validated

Note: They can be the same

```
:UserShape a sh:NodeShape ;  
  sh:targetNode :alice, :bob, :carol ;  
  sh:nodeKind sh:IRI ;  
  sh:property :hasName,  
              :hasEmail .  
:hasName sh:path schema:name ;  
  sh:minCount 1;  
  sh:maxCount 1;  
  sh:datatype xsd:string .  
. . .
```

Shapes graph

```
:alice schema:name "Alice Cooper" ;  
       schema:email <mailto:alice@mail.org> .  
  
:bob   schema:firstName "Bob" ;  
       schema:email <mailto:bob@mail.org> .  
  
:carol schema:name "Carol" ;  
       schema:email "carol@mail.org" .
```

Data graph

Validation Report

The output of the validation process is a list of violation errors

No errors $\Rightarrow$ RDF conforms to shapes graph

```
[ a          sh:ValidationReport ;  
  sh:conforms true  
].
```

```
[ a          sh:ValidationReport ;  
  sh:conforms false ;  
  sh:result  [  
    a          sh:ValidationResult ;  
    sh:focusNode :bob ;  
    sh:message  
      "MinCount violation. Expected 1, obtained: 0" ;  
    sh:resultPath schema:name ;  
    sh:resultSeverity sh:Violation ;  
    sh:sourceConstraintComponent  
      sh:MinCountConstraintComponent ;  
    sh:sourceShape :hasName  
  ] ;  
  ...
```

Importing shapes graphs

SHACL processors follow `owl:imports` declarations

It extends the current shapes graph with the imported shapes

```
:UserShape a sh:NodeShape ;  
  sh:targetNode :alice, :bob, :carol ;  
  sh:nodeKind sh:IRI ;  
  sh:property :hasName .  
:hasName sh:path schema:name ;  
  sh:minCount 1;  
  sh:maxCount 1;  
  sh:datatype xsd:string .
```

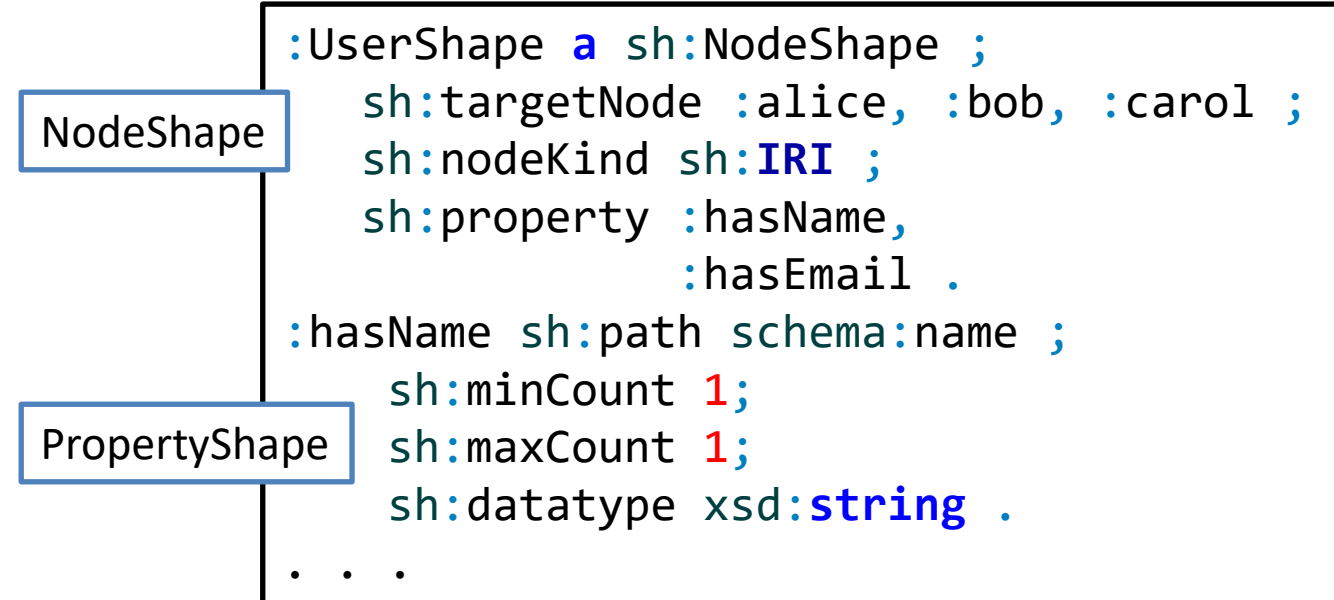
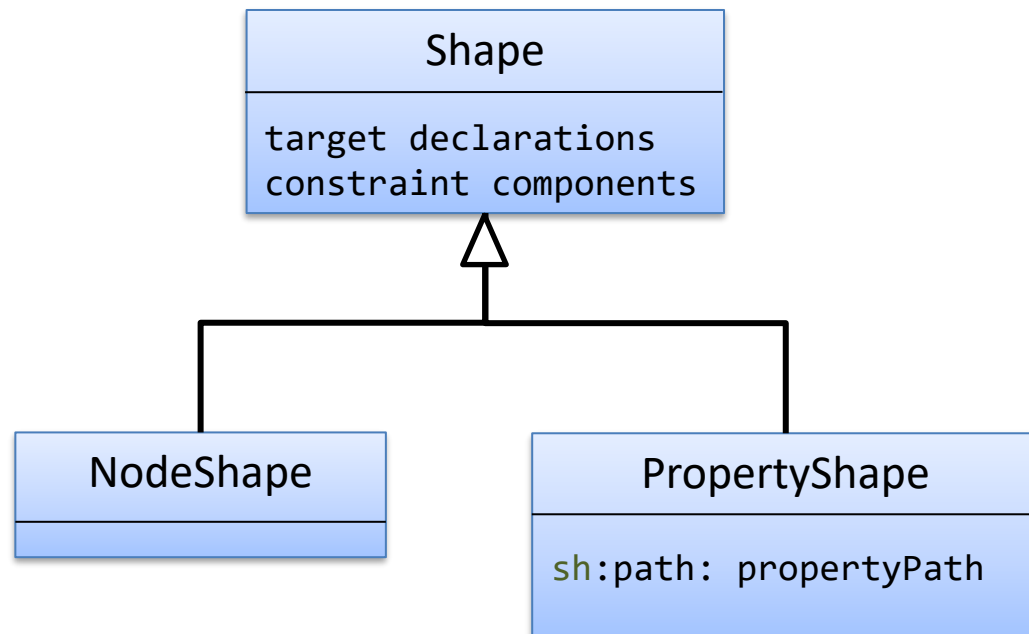
```
<> owl:imports <http://example.org/UserShapes> .  
  
:TeacherShape a sh:NodeShape;  
  sh:targetClass :Teacher ;  
  sh:node :UserShape ;  
  sh:property [  
    sh:path :teaches ;  
    sh:minCount 1;  
    sh:datatype xsd:string;  
  ] .
```

Node and property shapes

2 types of shapes:

NodeShape: constraints about shapes of nodes

PropertyShapes: constraints on property path values of a node



Node Shapes

Constraints about a focus node

```
:UserShape a sh:NodeShape ;  
  sh:nodeKind sh:IRI ;  
  sh:targetClass :User .
```

```
:alice a :User .  
  
<http://example.org/bob> a :User .  
  
_:1 a :User .
```



Try it: <https://tinyurl.com/y6qyqo5g>

Property shapes

Constraints about a given property and its values for the focus node

`sh:property` associates a shape with a property shape

`sh:path` identifies the path

```
:User a sh:NodeShape ;  
  sh:property [  
    sh:path      schema:email ;  
    sh:nodeKind  sh:IRI  
  ] .
```

```
:alice a :User ;  
       schema:email <mailto:alice@mail.org> .  
  
:bob   a :User;  
       schema:email <mailto:bob@mail.org> . ☹️  
  
:carol a :User;  
       schema:email "carol@mail.org" . ☹️
```

Paths in property shapes

Subset of SPARQL property paths using the following names:

inversePath

alternativePath

zeroOrMorePath

oneOrMorePath

zeroOrOnePath

```
:User a sh:NodeShape, rdfs:Class ;  
  sh:property [  
    sh:path [sh:inversePath schema:follows ];  
    sh:nodeKind sh:IRI ;  
  ] .
```

```
:alice a :User;  
       schema:follows :bob .  
  
:bob   a :User .  
  
:carol a :User;  
       schema:follows :alice .  
  
_:1 schema:follows :bob .
```



Constraint components

Constraints associated with shapes

They have parameters whose values specify the constraints

SHACL-core provides a list of predefined constraint components

Most of them have one parameter which identifies them

Convention:

Parameter: `sh:xx`

Constraint component: `sh:xxConstraintComponent`

```
:UserShape a sh:NodeShape ;  
sh:nodeKind sh:IRI .
```

Constraint component
`sh:nodeKindConstraintComponent`

Parameter
`sh:nodeKind`

Value of Parameter
`sh:IRI ;`

NOTE: Custom constraint components
can be defined in SHACL-SPARQL

Repeated parameter

Each value of the parameter declares a different constraint

```
:UserShape a sh:NodeShape;  
sh:class foaf:Person ;  
sh:class schema:Person .
```

```
:alice a schema:Person, foaf:Person .  
:bob a schema:Person .
```



SHACL Core constraint components

Type	Constraints
Cardinality	minCount, maxCount
Types of values	class, datatype, nodeKind
Values	node, in, hasValue, property
Range of values	minInclusive, maxInclusive minExclusive, maxExclusive
String based	minLength, maxLength, pattern
Language based	languageIn, uniqueLang
Logical constraints	not, and, or, xone
Closed shapes	closed, ignoredProperties
Property pair constraints	equals, disjoint, lessThan, lessThanOrEquals
Non-validating constraints	name, description, order, group
Qualified shapes	qualifiedValueShape, qualifiedValueShapesDisjoint qualifiedMinCount, qualifiedMaxCount

See later



Human friendly messages

Message declares the message that will appear in the validation report in case of violation

```
:UserShape a sh:NodeShape ;  
  sh:targetClass :User ;  
  sh:property [  
    sh:path      schema:name ;  
    sh:minCount  1 ;  
    sh:message   "Where is the name?"  
  ] .
```



```
:bob  a :User ;  
      schema:alias "Bob" . ☹️
```



```
:report a :ValidationReport ;  
sh:conforms false ;  
sh:result [ a sh:ValidationResult ;  
  sh:resultSeverity      sh:Violation ;  
  sh:sourceConstraintComponent sh:MinCountConstraintComponent ;  
  sh:sourceShape        ... ;  
  sh:focusNode           :bob ;  
  sh:resultPath          schema:name ;  
  sh:resultMessage       "Where is the name?" ;  
] .
```

Severities

Declare the level of the violation

3 predefined levels: Violation (default), Warning, Info

```
:UserShape a sh:NodeShape ;  
sh:targetClass :User ;  
sh:property [  
  sh:path      schema:name ;  
  sh:datatype  xsd:string ;  
  sh:severity  sh:Warning  
] .
```

```
:bob a :User ;  
schema:alias "Bob" .
```

```
:report a :ValidationReport ;  
sh:conforms false ;  
sh:result [ a sh:ValidationResult ;  
  sh:resultSeverity      sh:Warning ;  
  sh:sourceConstraintComponent sh:MinCountConstraintComponent ;  
  sh:sourceShape          ... ;  
  sh:focusNode             :bob ;  
  sh:resultPath            schema:name ;  
  sh:resultMessage         "MinCount Error" ;  
] .
```

Deactivating shapes

Deactivate a shape

Useful when importing shapes

UserShapes

```
:UserShape a sh:NodeShape;  
  sh:targetClass :User ;  
  sh:property :HasName ;  
  sh:property :HasEmail .  
  
:HasName sh:path schema:name ;  
  sh:datatype xsd:string .  
  
:HasEmail sh:path schema:email ;  
  sh:minCount 1;  
  sh:nodeKind sh:IRI .
```

```
<> owl:imports <UserShapes> .  
  
:TeacherShape a sh:NodeShape;  
  sh:targetClass :Teacher ;  
  sh:node :UserShape ;  
  sh:property [ sh:path :teaches ;  
    sh:minCount 1;  
    sh:datatype xsd:string;  
  ] .  
  
:HasEmail sh:deactivated true .
```

Target declarations

Targets specify nodes that must be validated against the shape

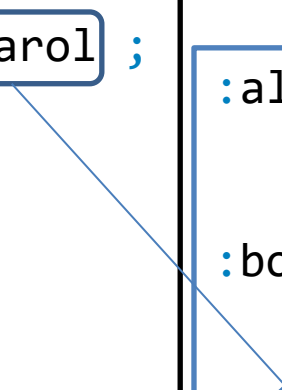
Several types

Value	Description
targetNode	Directly point to a node
targetClass	All nodes that have a given type
targetSubjectsOf	All nodes that are subjects of some predicate
targetObjectsOf	All nodes that are objects of some predicate

Target node

Directly declare which nodes must validate the against the shape

```
:UserShape a sh:NodeShape ;  
  sh:targetNode :alice, :bob, :carol ;  
  sh:property [  
    sh:path schema:name ;  
    sh:minCount 1;  
    sh:maxCount 1;  
    sh:datatype xsd:string ;  
  ] ;  
  sh:property [  
    sh:path schema:email ;  
    sh:minCount 1;  
    sh:maxCount 1;  
    sh:nodeKind sh:IRI ;  
  ] .
```



```
:alice schema:name "Alice Cooper" ;  
      schema:email <mailto:alice@mail.org> .  
  
:bob  schema:givenName "Bob" ;  
      schema:email <mailto:bob@mail.org> .  
  
:carol schema:name "Carol" ;  
       schema:email "carol@mail.org" .
```

Target class

Selects all nodes that have a given class

Looks for `rdf:type` declarations*

```
:UserShape a sh:NodeShape ;  
  sh:targetClass :User ;  
  sh:property [  
    sh:path schema:name ;  
    sh:minCount 1 ;  
    sh:maxCount 1 ;  
    sh:datatype xsd:string ;  
  ] ;  
  sh:property [  
    sh:path schema:email ;  
    sh:minCount 1 ;  
    sh:maxCount 1 ;  
    sh:nodeKind sh:IRI ;  
  ] .
```

```
:alice a :User ;  
  schema:name "Alice Cooper" ;  
  schema:email <mailto:alice@mail.org> .  
  
:bob a :User ;  
  schema:givenName "Bob" ;  
  schema:email <mailto:bob@mail.org> .  
  
:carol a :User ;  
  schema:name "Carol" ;  
  schema:email "carol@mail.org" .
```

* Also looks for `rdfs:subClassOf`*/`rdf:type` declarations

Implicit class target

A shape with type `sh:Shape` and `rdfs:Class` is a scope class of itself
The `targetClass` declaration is implicit

```
:User a sh:NodeShape, rdfs:Class ;  
  sh:property [  
    sh:path schema:name ;  
    sh:minCount 1;  
    sh:maxCount 1;  
    sh:datatype xsd:string ;  
  ] ;  
  sh:property [  
    sh:path schema:email ;  
    sh:minCount 1;  
    sh:maxCount 1;  
    sh:nodeKind sh:IRI ;  
  ] .
```

```
:alice a :User;  
  schema:name "Alice Cooper" ;  
  schema:email <mailto:alice@mail.org> .  
  
:bob a :User;  
  schema:givenName "Bob" ;  
  schema:email <mailto:bob@mail.org> .  
  
:carol a :User;  
  schema:name "Carol" ;  
  schema:email "carol@mail.org" .
```

targetSubjectsOf

```
:UserShape a sh:NodeShape;  
sh:targetSubjectsOf :teaches ;  
sh:property [  
  sh:path schema:name ;  
  sh:minCount 1;  
  sh:maxCount 1;  
  sh:datatype xsd:string ;  
] .
```

```
:alice :teaches :Algebra ;      #Passes as :UserShape  
      schema:name "Alice" .  
  
:bob   :teaches :Logic ;        #Fails as :UserShape  
      foaf:name "Robert" .  
  
:carol foaf:name 23 .           # Ignored
```

targetObjectsOf

```
:UserShape a          sh:NodeShape;  
sh:targetObjectsOf :isTaughtBy ;  
sh:property [  
  sh:path          schema:name ;  
  sh:minCount 1;  
  sh:maxCount 1;  
  sh:datatype xsd:string ;  
] .
```

```
:alice schema:name "Alice" . #Passes as :UserShape  
:bob    foaf:name "Robert" . #Fails as :UserShape  
:carol  foaf:name 23 .        # Ignored  
:algebra :isTaughtBy :alice, :bob .
```

Core constraint components

Type	Constraints
Cardinality	minCount, maxCount
Types of values	datatype, class, nodeKind
Values	node, in, hasValue
Range of values	minInclusive, maxInclusive minExclusive, maxExclusive
String based	minLength, maxLength, pattern, stem, uniqueLang
Logical constraints	not, and, or, xone
Closed shapes	closed, ignoredProperties
Property pair constraints	equals, disjoint, lessThan, lessThanOrEquals
Non-validating constraints	name, value, defaultValue
Qualified shapes	qualifiedValueShape, qualifiedMinCount, qualifiedMaxCount

Cardinality constraints

Constraint	Description
minCount	Restricts minimum number of triples involving the focus node and a given predicate. Default value: 0
maxCount	Restricts maximum number of triples involving the focus node and a given predicate. If not defined = unbounded

```
:User a sh:NodeShape ;
  sh:property [
    sh:path      schema:follows ;
    sh:minCount  2 ;
    sh:maxCount  3 ;
  ] .
```

```
:alice schema:follows :bob,
                                     :carol .

:bob   schema:follows :alice . ☹️

:carol schema:follows :alice,
                                     :bob,
                                     :carol,
                                     :dave . ☹️
```

Datatypes of values

Constraint	Description
datatype	Restrict the datatype of all value nodes to a given value

```
:User a sh:NodeShape ;  
  sh:property [  
    sh:path      schema:birthDate ;  
    sh:datatype  xsd:date ;  
  ] .
```

```
:alice schema:birthDate "1985-08-20"^^xsd:date .  
:bob   schema:birthDate "Unknown"^^xsd:date .  
:carol schema:birthDate 1990 .
```



Class of values

Constraint	Description
class	Verify that each node in an instance of some class It also allows instances of subclasses*

(*) The notion of SHACL instance is different from RDFS
It is defined as `rdfs:subClassOf*/rdf:type`

```
:User a sh:NodeShape, rdfs:Class ;  
  sh:property [  
    sh:path schema:follows ;  
    sh:class :User  
  ] .
```

```
:Manager rdfs:subClassOf :User .  
  
:alice a :User;  
       schema:follows :bob .  
:bob   a :Manager ;  
       schema:follows :alice .  
:carol a :User;  
       schema:follows :alice, :dave .  
:dave  a :Employee .
```



Kind of values

Constraint	Description
nodeKind	Possible values: BlankNode, IRI, Literal, BlankNodeOrIRI, BlankNodeOrLiteral, IRIOrLiteral

```
:User a sh:NodeShape, rdfs:Class ;
  sh:property [
    sh:path      schema:name ;
    sh:nodeKind  sh:Literal ;
  ];
sh:property [
  sh:path      schema:follows ;
  sh:nodeKind  sh:BlankNodeOrIRI
];
sh:nodeKind sh:IRI .
```

```
:alice a :User;
      schema:name      _:1 ;
      schema:follows   :bob .

:bob   a :User;
      schema:name      "Robert";
      schema:follows   [ schema:name "Dave" ] .

:carol a :User;
      schema:name      "Carol" ;
      schema:follows   "Dave" .

_:1 a :User .
```







Constraints on values

Constraint	Description
hasValue	Verifies that the focus node has a given value
in	Enumerates the value nodes that a property may have

```
:User a sh:NodeShape, rdfs:Class ;
  sh:property [
    sh:path      schema:affiliation ;
    sh:hasValue  :OurCompany ;
  ];
sh:property [
  sh:path schema:gender ;
  sh:in   (schema:Male schema:Female)
] .
```

```
:alice a :User;
       schema:affiliation :OurCompany ;
       schema:gender schema:Female .

:bob   a :User;
       schema:affiliation :AnotherCompany ; ☹️
       schema:gender schema:Male .

:carol a :User;
       schema:affiliation :OurCompany ;
       schema:gender schema:Unknown . ☹️
```

Constraints on values with another shape

Constraint	Description
node	All values of a given property must have a given shape Recursion is not allowed in current SHACL

```
:User a sh:NodeShape, rdfs:Class ;  
  sh:property [  
    sh:path schema:worksFor ;  
    sh:node :Company ;  
  ] .
```

```
:Company a sh:Shape ;  
  sh:property [  
    sh:path schema:name ;  
    sh:datatype xsd:string ;  
  ] .
```

```
:alice a :User;  
       schema:worksFor :OurCompany .
```

```
:bob   a :User;  
       schema:worksFor :Another .
```



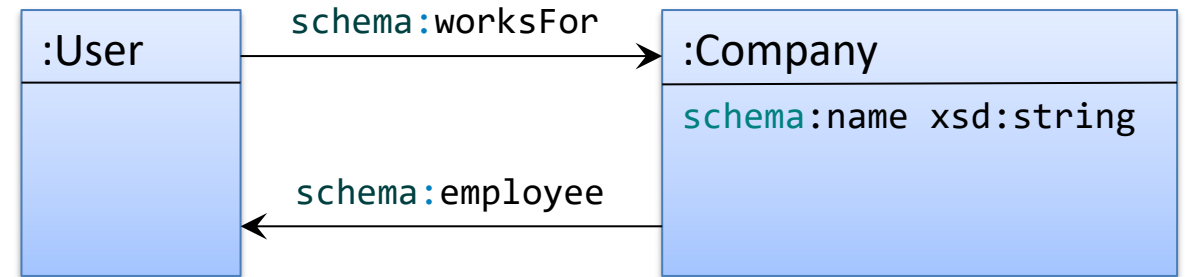
```
:OurCompany  
  schema:name "OurCompany" .
```

```
:Another  
  schema:name 23 .
```

Value shapes and recursion

Can we define cyclic data models as the following?

```
:User a sh:NodeShape ;  
  sh:property [  
    sh:path schema:worksFor ;  
    sh:node :Company ;  
  ] .  
  
:Company a sh:Shape ;  
  sh:property [  
    sh:path schema:name ;  
    sh:datatype xsd:string ;  
  ] ;  
  sh:property [  
    sh:path schema:employee ;  
    sh:node :User ;  
  ] .
```



```
:alice schema:worksFor :OneCompany .  
:bob schema:worksFor :OneCompany .  
:carol schema:worksFor :OneCompany .  
  
:OneCompany schema:name "One" ;  
  schema:employee :alice, :bob, :carol .
```

SHACL spec leaves this as implementation dependent

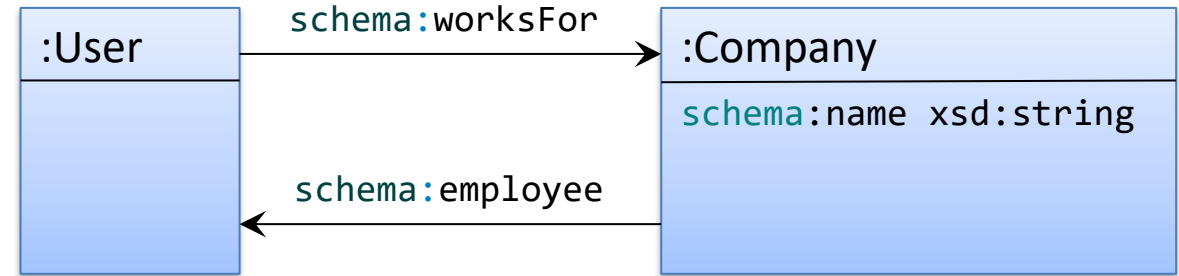
Try it: <https://tinyurl.com/y3hkka6s>

An approach to avoid recursion

Add `rdf:type` arcs for every resource and use `sh:class`

```
:User a sh:NodeShape ;  
  sh:property [  
    sh:path    schema:worksFor ;  
    sh:class   :Company ;  
  ] .
```

```
:Company a sh:Shape ;  
  sh:property [  
    sh:path      schema:name ;  
    sh:datatype  xsd:string ;  
  ] ;  
  sh:property [  
    sh:path      schema:employee ;  
    sh:class     :User ;  
  ] .
```



```
:alice a :User ;  
       schema:worksFor :OneCompany .  
:bob   a :User ;  
       schema:worksFor :OneCompany .  
:carol a :User ;  
       schema:worksFor :Something .  
  
:OneCompany a :Company ;  
            schema:name "One" ;  
            schema:employee :alice, :bob, :carol .
```



Logical Operators

Constraint	Description
and	Conjunction of a list of shapes
or	Disjunction of a list of shapes
not	Negation of a shape
xone	Exactly one (similar XOR for 2 arguments)

Conjunction: and

Although it can be declared explicitly, it is the default behavior

```
:User a sh:NodeShape ;
  sh:and (
    [ sh:property [
      sh:path      schema:name;
      sh:minCount 1;
    ]
    [ sh:property [
      sh:path      schema:affiliation;
      sh:minCount 1;
    ]
  ]
) .
```

≡

```
:User a sh:Shape ;
  [ sh:property [
    sh:path      schema:name;
    sh:minCount 1;
  ]
  [ sh:property [
    sh:path      schema:affiliation;
    sh:minCount 1;
  ]
]
.
```

Disjunction: or

```
:User a sh:NodeShape ;  
  sh:or (  
    [ sh:property [  
      sh:predicate foaf:name;  
      sh:minCount 1;  
    ]  
    [  
      sh:property [  
        sh:predicate schema:name;  
        sh:minCount 1;  
      ]  
    ]  
  ) .
```

```
:alice schema:name "Alice" .
```

```
:bob foaf:name "Robert" .
```

```
:carol rdfs:label "Carol" .
```



Negation: not

```
:NotFoaf a sh:NodeShape ;  
  sh:not [ a sh:Shape ;  
    sh:property [  
      sh:predicate foaf:name ;  
      sh:minCount 1 ;  
    ] ;  
  ] .
```

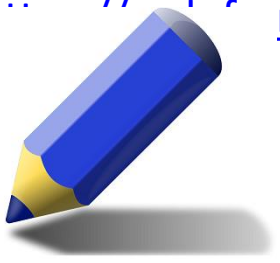
```
:alice schema:name "Alice" .  
:bob    foaf:name "Robert" .  
:carol  rdfs:label "Carol" .
```



Exactly one: xone

```
:UserShape a sh:NodeShape ;  
  sh:targetClass :User ;  
  sh:xone (  
    [ sh:property [  
      sh:path      foaf:name;  
      sh:minCount  1;  
    ]  
    [  
      sh:property [  
        sh:path      schema:name;  
        sh:minCount  1;  
      ]  
    ]  
  ) .
```

```
:alice a :User ;           #Passes as :User  
      schema:name "Alice" .  
  
:bob   a :User ;           #Passes as :User  
      foaf:name   "Robert" .  
  
:carol a :User ;           #Fails as :User  
      foaf:name   "Carol";  
      schema:name "Carol" .  
  
:dave  a :User ;           #Fails as :User  
      rdfs:label  "Dave" .
```



Exercise

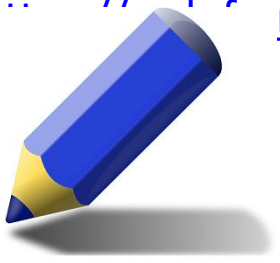
IF-THEN pattern

All products must have `:productID` and, if a product has `rdf:type schema:Vehicle` then it must have the properties `schema:vehicleEngine` and `schema:fuelType`

```
:p1 a :Book;                # Conforms
    schema:productID        "P1" .

:p2 a schema:Vehicle ;      # Conforms
    schema:productID        "P2" ;
    schema:fuelType         "Gasoline" ;
    schema:vehicleEngine    "X2" .

:p3 a schema:Vehicle ;      # Fails
    schema:productID        "P3" .
```



Exercise

IF-THEN-ELSE pattern

All products must have `:productID` and, if a product has `rdf:type schema:Vehicle` then it must have the properties `schema:vehicleEngine` and `schema:fuelType` else it must have `schema:category` with a string value.

Value ranges

Constraint	Description
minInclusive	$\leq$
maxInclusive	$\geq$
minExclusive	$<$
maxExclusive	$>$

```
:Rating a sh:NodeShape ;  
  sh:property [  
    sh:path          schema:ratingValue ;  
    sh:minInclusive  1 ;  
    sh:maxInclusive  5 ;  
    sh:datatype      xsd:integer  
  ] .
```

```
:bad          schema:ratingValue 1 .  
:average      schema:ratingValue 3 .  
:veryGood     schema:ratingValue 5 .  
:zero         schema:ratingValue 0 . ☹️
```

String based constraints

Constraint	Description
minLength	Restricts the minimum string length on value nodes
maxLength	Restricts the maximum string length on value nodes
pattern	Checks if the string value matches a regular expression

minLength/maxLength

Checks the string representation of the value

This cannot be applied to blank nodes

If minLength = 0, no restriction on string length

```
:User a sh:NodeShape ;  
  sh:property [  
    sh:path      schema:name ;  
    sh:minLength 4 ;  
    sh:maxLength 10 ;  
  ] .
```

```
:alice schema:name "Alice" .  
:bob schema:name "Bob" .  
:carol schema:name :Carol .  
:strange schema:name _:strange .
```



?



pattern

Checks if the values matches a regular expression

It can be combined with sh:flags

```
:Product a sh:NodeShape ;  
  sh:property [  
    sh:path      schema:productID ;  
    sh:pattern    "^P\\d{3,4}" ;  
    sh:flags      "i" ;  
  ] .
```

```
:car      schema:productID "P2345" .  
:bus      schema:productID "p567" .  
:truck    schema:productID "P12" .  
:bike     schema:productID "B123" .
```



Language based constraints

Constraint	Description
languageIn	Declares the allowed languages of a literal
uniqueLang	Specifies that no pair of nodes can have the same language tag

languageIn

Specifies the allowed language that a literal can have

```
:ProductShape a sh:NodeShape;  
  sh:targetClass :Product ;  
  sh:property [  
    sh:path      rdfs:label ;  
    sh:languageIn ("es" "en" "fr")  
  ] .
```

```
:p234 a :Product ;  
  rdfs:label "jamón"@es, "ham"@en .
```

```
:p235 a :Product ;  
  rdfs:label "milk"@en .
```

```
:p236 a :Product ;  
  rdfs:label "Käse"@de .
```



```
:p237 a :Product ;  
  rdfs:label "patatas"@es ,  
            "kartofeln"@de .
```



uniqueLang

Checks that no pair of nodes use the same language tag

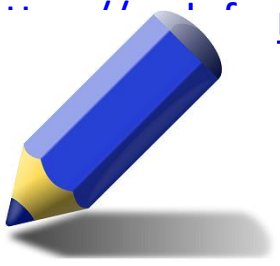
```
:CountryShape a sh:NodeShape ;  
  sh:targetClass :Country ;  
  sh:property [  
    sh:path      skos:prefLabel ;  
    sh:uniqueLang true  
  ] .
```

```
:spain a :Country;  
  skos:prefLabel "Spain"@en,  
                 "España"@es .  
  
:france a :Country;  
  skos:prefLabel "France",  
                 "France"@en,  
                 "Francia"@es .
```

```
:italy a :Country .
```

```
:usa a :Country;  
  skos:prefLabel "USA"@en,  
                 "United States"@en.
```





Exercise

Nodes must have exactly one literal per language in English and Spanish for property `skos:prefLabel`

Property pair constraints

Constraint	Description
equals	The sets of values of both properties at a given focus node must be equal
disjoint	The sets of values of both properties at a given focus node must be different
lessThan	The values must be smaller than the values of another property
lessThanOrEquals	The values must be smaller or equal than the values of another property

```
:User a sh:NodeShape ;
sh:property [
  sh:path schema:givenName ;
  sh>equals foaf:firstName
];
sh:property [
  sh:path schema:givenName ;
  sh:disjoint schema:lastName
] .
```

```
:alice schema:givenName "Alice";
       schema:lastName  "Cooper";
       foaf:firstName    "Alice" .

:bob   schema:givenName "Bob";
       schema:lastName  "Smith" ;
       foaf:firstName    "Robert" .

:carol schema:givenName "Carol";
       schema:lastName  "Carol" ;
       foaf:firstName    "Carol" .
```




Closed shapes

Constraint	Description
closed	Valid resources must only have values for properties that appear in <code>sh:property</code>
ignoredProperties	Optional list of properties that are also permitted

```
:User a sh:NodeShape ;
  sh:closed true ;
  sh:ignoredProperties ( rdf:type ) ;
  sh:property [
    sh:path schema:givenName ;
  ];
  sh:property [
    sh:path schema:lastName ;
  ] .
```

```
:alice schema:givenName "Alice";
        schema:lastName "Cooper" .

:bob    a :Employee ;
        schema:givenName "Bob";
        schema:lastName "Smith" .

:carol  schema:givenName "Carol";
        schema:lastName "King" ;
        rdfs:label "Carol" .
```



Qualified value shapes

Problem with repeated properties

Example: Books have two IDs (an isbn and an internal code)

```
:Book a sh:NodeShape ;  
  sh:property [  
    sh:path      schema:productID ;  
    sh:minCount  1;  
    sh:datatype  xsd:string ;  
    sh:pattern    "^isbn"  
  ];  
  sh:property [  
    sh:path      schema:productID ;  
    sh:minCount  1;  
    sh:datatype  xsd:string ;  
    sh:pattern    "^code"  
  ] .
```

```
:b1 schema:productID "isbn:123-456-789" ;  
    schema:productID "code234" .
```

It fails!!

Qualified value shapes

Qualified value shapes verify that certain number of values of a given property have a given shape

```
:Book a sh:NodeShape;  
  sh:property [  
    sh:path schema:productID ;  
    sh:minCount 2; sh:maxCount 2; ];  
  sh:property [  
    sh:path schema:productID ;  
    sh:qualifiedMinCount 1 ;  
    sh:qualifiedValueShape [  
      sh:pattern "^isbn"  
    ]];  
  sh:property [  
    sh:path schema:productID ;  
    sh:qualifiedMinCount 1 ;  
    sh:qualifiedValueShape [  
      sh:pattern "^code"  
    ]]
```

```
:b1 schema:productID "isbn:123-456-789" ;  
    schema:productID "code234" .
```

Non-validating constraints

Can be useful to annotate shapes or design UI forms

Constraint	Description
name	Provide human-readable labels for a property
description	Provide a description of a property
order	Relative order of the property
group	Group several constraints together

```
:User a sh:NodeShape ;
sh:property [
  sh:path schema:url ;
  sh:name "URL";
  sh:description "User URL";
  sh:order 1
];
sh:property [
  sh:path schema:name ;
  sh:name "Name";
  sh:description "User name";
  sh:order 2
] .
```

Non-validating constraints

```
:User a sh:NodeShape ;  
  sh:property [ sh:path schema:url ;  
    sh:name "URL";  
    sh:group :userDetails  
  ] ;  
  sh:property [ sh:path schema:name ;  
    sh:name "Name"; sh:group :userDetails  
  ] ;  
  sh:property [ sh:path schema:address ;  
    sh:name "Address"; sh:group :location  
  ] ;  
  sh:property [ sh:path schema:country ;  
    sh:name "Country"; sh:group :location  
  ] .
```

```
:userDetails a sh:PropertyGroup ;  
  sh:order 0 ;  
  rdfs:label "User details" .  
  
:location a sh:PropertyGroup ;  
  sh:order 1 ;  
  rdfs:label "Location" .
```

An agent could generate a form like:

User details

URL: _____

Name: _____

Location

Address: _____

Country: _____

SHACL-SPARQL

SPARQL constraints

Constraints based on SPARQL code.

When the SPARQL query return validation errors a violation is reported

SPARQL constraints have type `sh:SPARQLConstraint`

Constraint	Description
message	Message in case of error
sparql	SPARQL code that is run
prefixes	Points to namespace prefix declarations defined by <code>sh:declare</code> : Each one has: <code>sh:prefix</code> : Prefix alias <code>sh:namespace</code> : namespace IRI

SPARQL constraints

Special variables are pre-bound by the SHACL-SPARQL processor

Constraint	Description
<code>\$this</code>	Focus Node
<code>\$shapesGraph</code>	Can be used to query the shapes graph in named graphs Similar to: <code>GRAPH \$shapesGraph { ... }</code>
<code>\$currentShape</code>	Current shape

SPARQL constraints

Mappings between result rows and error validation information

Constraint	Description
sh:focusNode	Value of <code>\$this</code> variable
sh:subject	Value of <code>?subject</code> variable
sh:predicate	Value of <code>?predicate</code> variable
sh:object	Value of <code>?object</code> variable
sh:message	Value of <code>?message</code> variable
sh:sourceConstraint	The constraint that was validated against
sh:sourceShape	The shape that was validated against
sh:severity	sh:ViolationError by default or the value of sh:severity

SPARQL constraints

Example: Name must be the concatenation of singleName and familyName

```
:UserShape a sh:NodeShape ;
sh:targetClass :User ;
sh:sparql [ a sh:SPARQLConstraint ;
  sh:message "schema:name must equal schema:givenName+schema:familyName";
  sh:prefixes [ sh:declare [
    sh:prefix "schema" ;
    sh:namespace "http://schema.org/"^^xsd:anyURI ;
  ] ] ;
sh:select
  """SELECT $this (schema:name AS ?path) (?name as ?value)
  WHERE {
    $this schema:name ?name .
    $this schema:givenName ?givenName .
    $this schema:familyName ?familyName .
    FILTER (!isLiteral(?value) ||
      !isLiteral(?givenName) || !isLiteral(?familyName) ||
      concat(str(?givenName), ' ', str(?familyName))!=?name )
  }""" ;
] .
```

```
:alice a :User ;
  schema:givenName "Alice" ;
  schema:familyName "Cooper" ;
  schema:name "Alice Cooper" .

:bob a :User ;
  schema:givenName "Bob" ;
  schema:familyName "Smith" ;
  schema:name "Robert Smith" .
```



SPARQL constraint components

SHACL-SPARQL allows to declare custom constraint components
Once defined, they can be used like built-in constraint components

```
:ProductShape a sh:NodeShape ;  
  sh:targetClass :Product ;  
  sh:property [  
    sh:path      :color ;  
    :size        3 ;  
    sh:minCount  1 ;  
  ] .
```



```
:c1 :color (255 0 255) .
```

```
:c2 :color (255 0 210 345) . ☹️
```

```
:c3 :color (255 0) . ☹️
```

SPARQL constraint components

```
:FixedListConstraintComponent
a sh:ConstraintComponent ;
sh:parameter [
  sh:path          :size ;
  sh:name           "Size of list" ;
  sh:description   "The size of the list" ;
] ;
sh:labelTemplate "Size of values: \"{$size}\"" ;
sh:propertyValidator :fixedLengthValidator .
```

Two types of validators:
SPARQLSelectValidator
SPARQLASKValidator

```
:fixedLengthValidator a sh:SPARQLSelectValidator ;
sh:message
  "{$PATH} must have length {?size}, not {?count}" ;
sh:prefixes [ sh:declare [
  sh:prefix "rdf" ;
  sh:namespace
    "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
] ] ;
sh:select """SELECT $this ?value $count WHERE {
  $this $PATH ?value .
  { SELECT $this ?value
    (COUNT(?member) AS ?count)
    $size WHERE {
      ?value rdf:rest*/rdf:first ?member
    } GROUP BY $this ?value $size
  }
  FILTER (!isBlank(?value) || ?count != $size)
}"""
.
```

SPARQL constraint components

Property	Description
sh:parameter	Declares the parameters of the constraint component The values are subclasses of property shapes sh:path declares the parameter name sh:optional declares if the parameter is optional
sh:labelTemplate	Suggests how constraints are rendered. Can refer to parameter names using: \$varName
sh:nodeValidator	Associates a node shape validator
sh:propertyValidator	Associates a property shape validator

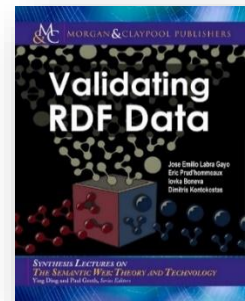
SPARQL based validators can be SELECT or ASK based validators

ShEx and SHACL: Several common features...

	ShEx	SHACL
Both employ the word "shape"	✓	✓
Can be used to validate RDF graphs	✓	✓
Node constraints	✓	✓
Constraints on incoming/outgoing arcs	✓	✓
Cardinalities on properties	✓	✓
RDF syntax	✓	✓
Extension mechanism	✓	✓

ShEx and SHACL compared

More information in chapter 7 of Validating RDF data book



2017 HTML version:

<http://book.validatingrdf.com>

Or in some talks like:

https://labra.weso.es/talk/2018_validatingrdfdata_stanford/

ShEx and SHACL

ShEx

```
:Researcher {  
  :name      xsd:string      ;  
  :birthPlace @:Place        ? ;  
  :birthDate  xsd:date        ? ;  
  :employer   @:Organization * ;  
  :knows      @:Person        *  
}
```

SHACL

```
:Researcher a sh:NodeShape ;  
  sh:targetClass :Human ;  
  sh:property [ sh:path rdfs:label ;  
                sh:minCount 1; sh:maxCount 1;  
                sh:datatype xsd:string ;  
              ] ;  
  sh:property [ sh:path :birthPlace ;  
                sh:node :Place  
              ] ;  
  sh:property [ sh:path :birthDate ;  
                sh:maxCount 1;  
                sh:datatype xsd:date  
              ] ;  
  sh:property [ sh:path :employer ;  
                sh:node :Organization  
              ] ;  
  sh:property [ sh:path :knows ;  
                sh:node :Researcher  
              ] .
```

Note:

Cyclic data models are implementation dependent in SHACL

Try it: <https://tinyurl.com/yh28fsct>

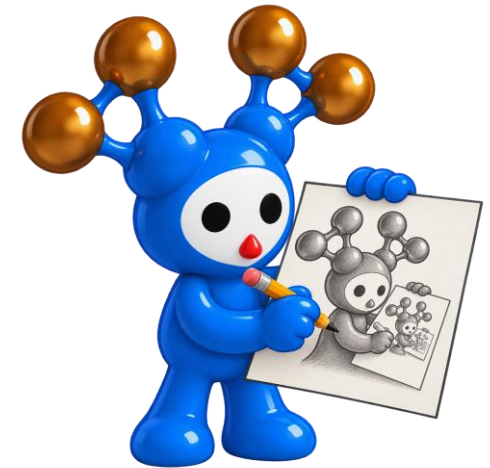
ShEx, SHACL and recursive shapes

ShEx adopts Greatest fixpoint semantics

SHACL left recursion as implementation dependent

SHACL 1.0: Implementations with incompatible behaviour

SHACL 1.2: [Issue 1199](#)



More information:

[sheval: An RDF data shapes evaluation tool and test-suite for recursive shapes](#), Shqiponja Ahmetaj, Iovka Boneva, Jan Hidders, Jose Emilio Labra Gayo, Wim Martens, Filip Murlak, Cem Okulmus, Ognjen Savković, Mantas Šimkus, Dominik Tomaszuk, 2nd International Workshop on Data Management for Knowledge Graphs, ISWC 2026.

[Common Foundations for Recursive Shape Languages](#), Shqiponja Ahmetaj, Iovka Boneva, Jan Hidders, Jose Emilio Labra Gayo, Wim Martens, Fabio Mogavero, Filip Murlak, Cem Okulmus, Ognjen Savković, Mantas Šimkus, Dominik Tomaszuk, [23rd International Conference on Principles of Knowledge Representation and Reasoning](#), 2026.

This evening

ShEx and SHACL

Comparing ShEx (compact syntax) with SHACL (compact syntax*)

ShEx

```
:Researcher {  
  :name      xsd:string      ;  
  :birthPlace @:Place        ? ;  
  :birthDate  xsd:date        ? ;  
  :employer   @:Organization * ;  
  :knows      @:Researcher   *  
}
```

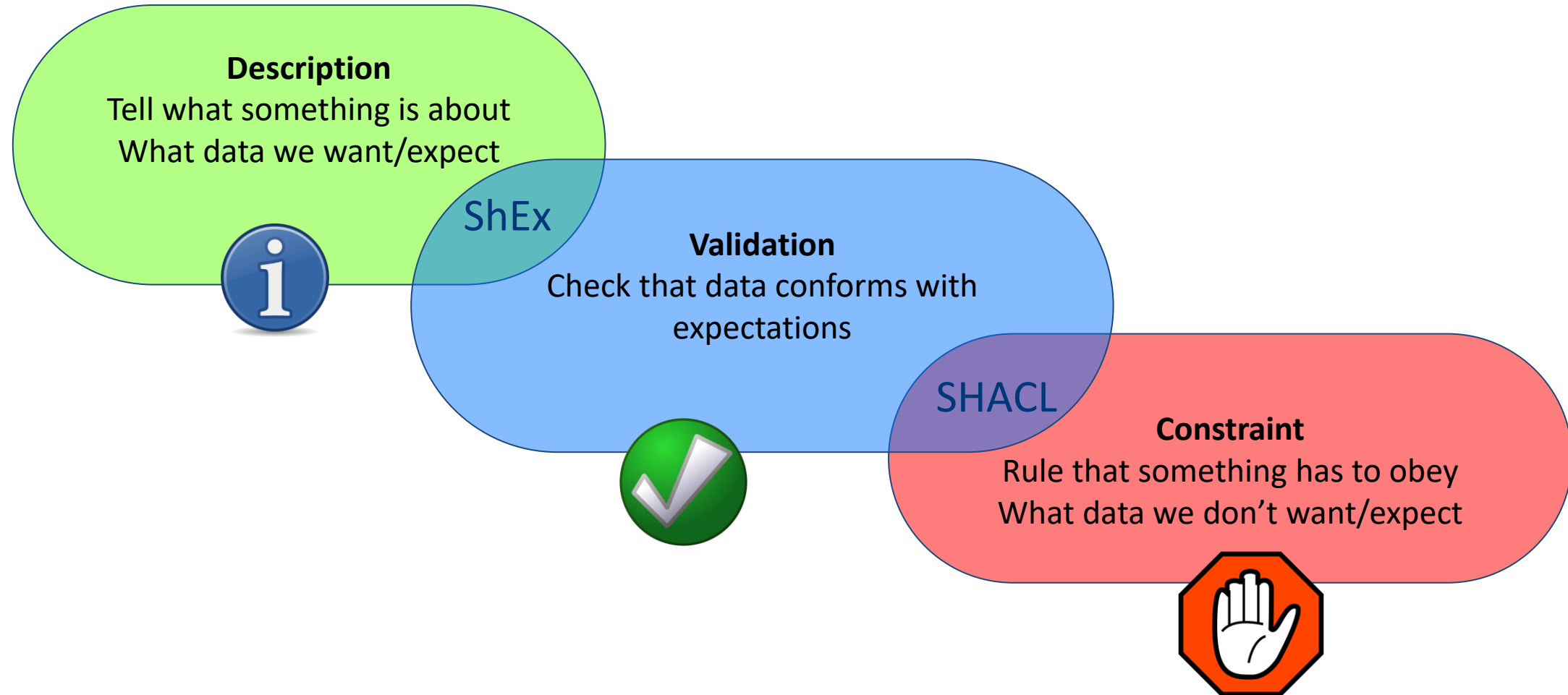
SHACL

```
shape @:Researcher -> :Human {  
  rdfs:label xsd:string [1..1] ;  
  :birthPlace @:Place        ;  
  :birthDate  xsd:date        [1..] ;  
  :employer   @:Organization  ;  
  :knows      @:Researcher  
}
```

Note: SHACL Compact syntax is being discussed for SHACL 1.2
More info: <https://w3c.github.io/shacl/shacl-compact-syntax/>

ShEx and SHACL

description - validation - constraints



DCTAP

Metadata model:

<https://www.dublincore.org/specifications/dctap/>

Tabular: CSV, Spreadsheets

Easy to understand and edit by domain experts

It can be used as a template and a *single-source-of-truth*

DCTAP can be converted to Shapes (ShEx/SHACL/etc.)



DCTAP example



shapeID	propertyID	propertyLabel	mandatory	repeatable	valueDataType	valueShape
Researcher	name	Label	TRUE	FALSE	xsd:string	
	birthPlace	Place of birth	FALSE	FALSE		Place
	birthDate	Date of birth	FALSE	FALSE	xsd:date	
	employer	Employer	FALSE	TRUE		Organization
Place						
Organization						



```
<Researcher> {  
  :name      xsd:string      ;  
  :birthPlace @<Place>      ? ;  
  :birthDate  xsd:date       ? ;  
  :employer   @<Organization> * ;  
}  
:Place> {}  
:Organization {}
```

In ShEx

Try it: <https://tinyurl.com/5t6ppr63>

Contents

Introduction to Knowledge graphs

Types of Knowledge Graphs:

RDF, Property graphs, Wikibase, RDF-1.2

Shaping RDF: ShEx & SHACL, DCTAP

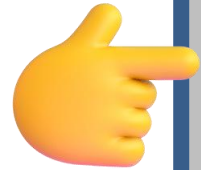
Shaping other types of Knowledge graphs:

Shaping Wikibase and Wikidata graphs

Shaping Property Graphs

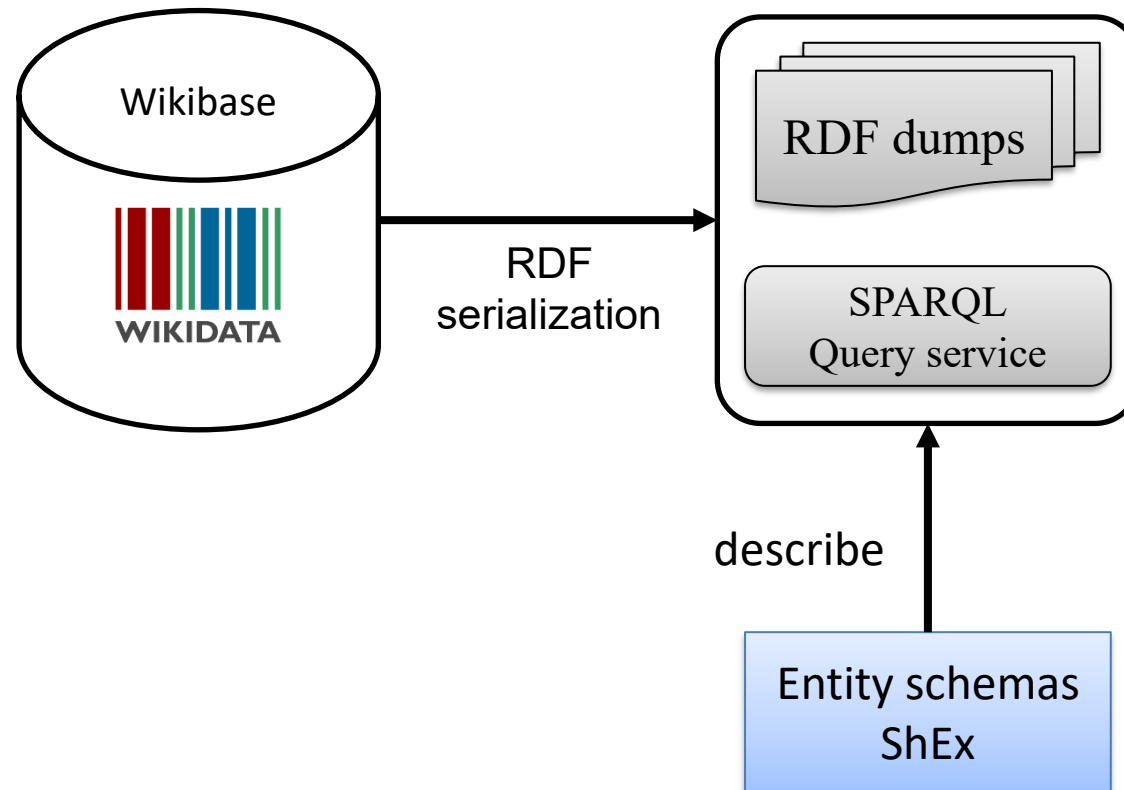
Shaping RDF 1.2

Some ideas for future work



Wikibase and RDF: Entity Schemas

If we have RDF, we can use ShEx to describe and validate entities



Entity Schemas

They can be used to describe Wikidata entities

Adopted in 2019 as a new Wikidata namespace Exxx

Example:

<https://www.wikidata.org/wiki/EntitySchema:E10>

Directory of entity schemas:

https://www.wikidata.org/wiki/Wikidata:Database_reports/EntitySchema_directory

Example of an Entity Schema

Q80.ttl

```
wd:Q80 rdf:type wikibase:Item ;
  wdt:P31 wd:Q5 ;
  wdt:P19 wd:Q84 ;
  wdt:P569 "1955-06-08T00:00:00Z"^^xsd:dateTime ;
  wdt:P108 wd:Q42944 ;
  p:P108 s:Q80-fcba864c, :Q80-4fe7940f
#...
.

:Q80-4fe7940f rdf:type wikibase:Statement ;
  wikibase:rank wikibase:NormalRank ;
  ps:P108 wd:Q42944 ;
  pq:P580 "1984-01-01T00:00:00Z"^^xsd:dateTime ;
  pq:P582 "1994-01-01T00:00:00Z"^^xsd:dateTime .

s:Q80-fcba864c a wikibase:Statement ;
  wikibase:rank wikibase:NormalRank ;
  ps:P108 wd:Q42944 ;
  pq:P580 "1980-06-01T00:00:00Z"^^xsd:dateTime ;
  pq:P582 "1980-12-01T00:00:00Z"^^xsd:dateTime .
```

Entity-schema - ShEx

E371

```
PREFIX pq: <.../prop/qualifier/>
PREFIX ps: <.../prop/statement/>
PREFIX p: <.../prop/>
PREFIX wdt: <.../prop/direct/>
PREFIX wd: <.../entity/>
PREFIX xsd: <...XMLSchema#>

<Researcher> {
  wdt:P31 [ wd:Q5 ] ;
  wdt:P19 @<Place> ;
  wdt:P569 xsd:dateTime ;
  wdt:P108 @<Employer> * ;
  p:P108 { ps:P108 @<Employer> ;
    pq:P580 xsd:dateTime ? ;
    pq:P582 xsd:dateTime *
  } *
}
<Place> { }
<Employer> { }
```

Using Entity Schemas for validation

Example: <https://www.wikidata.org/wiki/EntitySchema:E371>

wikidata.org/wiki/EntitySchema:E371

WIKIMANIA

Click here to Apply now
This application is open until Sunday 8th December, 2024

[Help with translations!]

Researcher (test) (E371)

language code	label	description	aliases	edit
en	Researcher (test)	Schema for researcher created as a test for a paper about WShEx	researcher	edit

```
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX pq: <http://www.wikidata.org/prop/qualifier/>
PREFIX ps: <http://www.wikidata.org/prop/statement/>
PREFIX p: <http://www.wikidata.org/prop/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

# Example SPARQL query (Tim Berners Lee)
# select ?p { values ?p { wd:Q80 }}
# This schema has been created as a simple demo for ShEx and WShEx

start = @<Researcher>

<Researcher> {
  wdt:P31 [ wd:Q5 ] ;
  wdt:P19 @<Place> ;
  wdt:P569 xsd:dateTime ? ;
  wdt:P108 @<Organization> * ;
  p:P108 {
    ps:P108 @<Organization> ;
    pq:P580 xsd:dateTime ? ;
    pq:P582 xsd:dateTime ?
  } * ;
}
```

check entities against this Schema [edit](#)

ShEx2 — Simple Online Validator

```
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX pq: <http://www.wikidata.org/prop/qualifier/>
PREFIX ps: <http://www.wikidata.org/prop/statement/>
PREFIX p: <http://www.wikidata.org/prop/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

# Example SPARQL query (Tim Berners Lee)
# select ?p { values ?p { wd:Q80 }}
# This schema has been created as a simple demo for ShEx and WShEx

start = @<Researcher>

<Researcher> {
  wdt:P31 [ wd:Q5 ] ;
  wdt:P19 @<Place> ;
  wdt:P569 xsd:dateTime ? ;
  wdt:P108 @<Organization> * ;
  p:P108 {
    ps:P108 @<Organization> ;
    pq:P580 xsd:dateTime ? ;
    pq:P582 xsd:dateTime ?
  } * ;
  wdt:P166 @<Award> * ;
  p:P166 {
```

validate (ctrl-enter)

Query

Entities to check

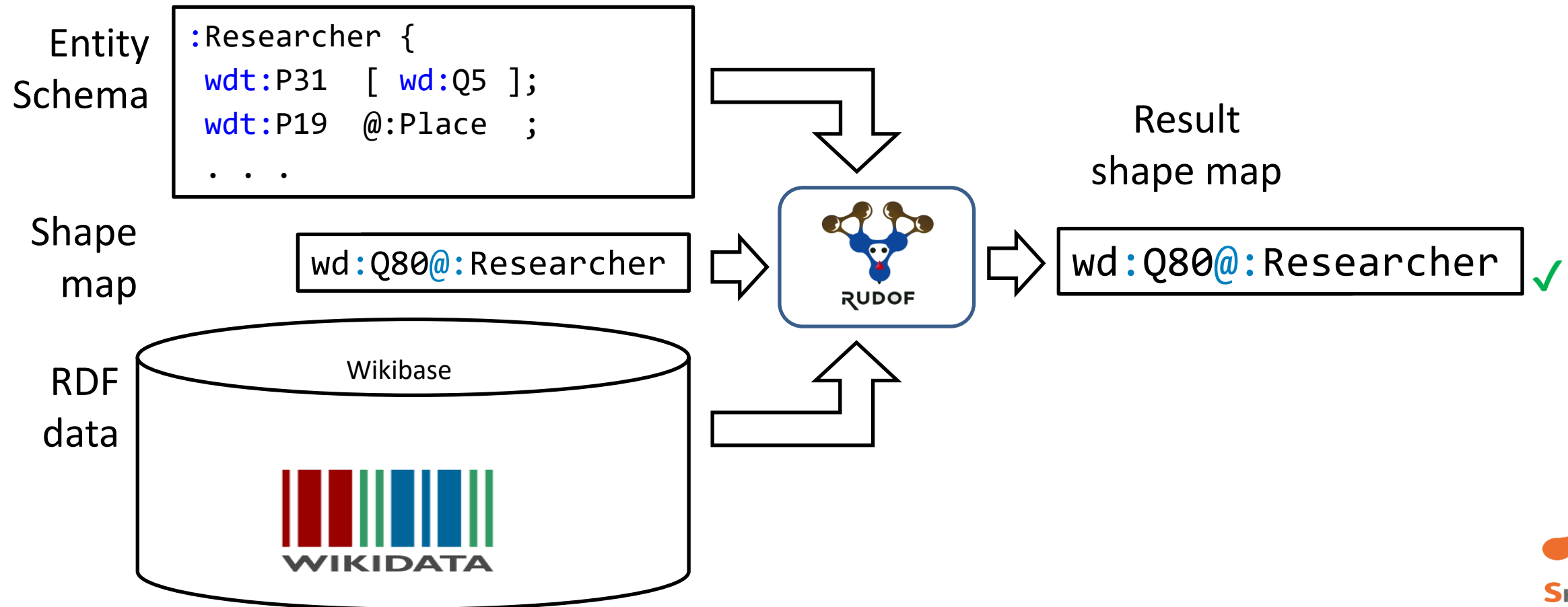
<http://www.wikidata.org/entity/Q80> @ START

✓

✓wd:Q80@START

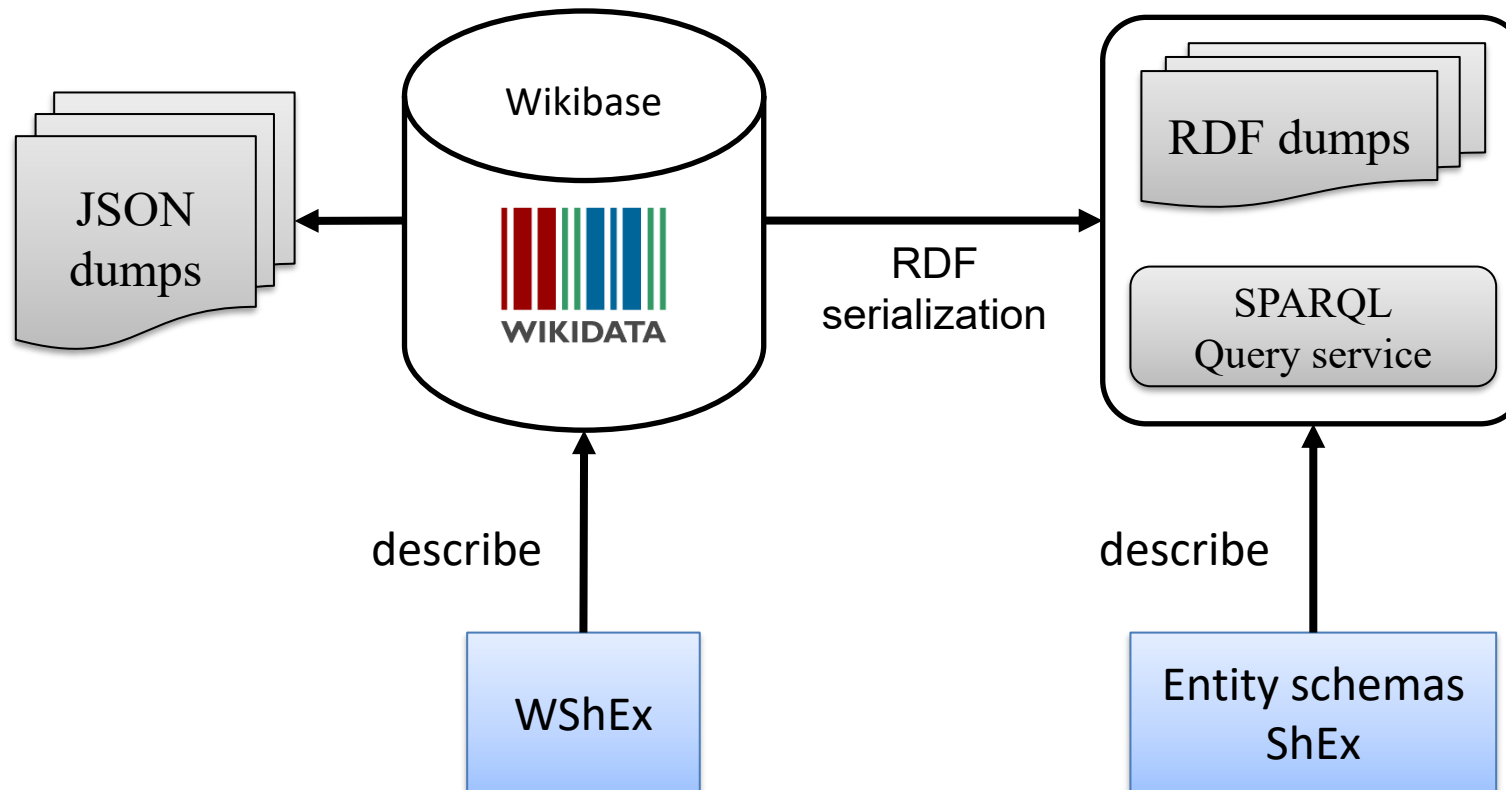
Using rudof with Wikibase

rudof supports validating against SPARQL endpoints



WShEx

Although Entity Schemas and ShEx just work, they describe Wikibase indirectly
WShEx has been proposed as a language to describe the Wikibase data model?



<https://www.weso.es/WShEx/>

Wikibase RDF representation



Item Discussion

Tim Berners-Lee (Q80)

employer (P108)



CERN (Q42944)

start time (P580)

1984

end time (P582)

1994

RDF

```
wd:Q80 wdt:P108 wd:Q42944 ;  
p:P108 s:Q80-fcba864c  
...  
s:Q80-fcba864c a wikibase:Statement ;  
ps:P108 wd:Q42944 ;  
pq:P580 1984 ;  
pq:P582 1994 .
```

Could be represented as

```
:Q80 :P108 :Q42944 { |  
:P580 1984 ;  
:P582 1994  
| }
```

Inspired by RDF-1.2 annotated triples syntax

ShEx vs WShEx

Entity-schema - ShEx

```
PREFIX pq: <.../prop/qualifier/>
PREFIX ps: <.../prop/statement/>
PREFIX p: <.../prop/>
PREFIX wdt: <.../prop/direct/>
PREFIX wd: <.../entity/>
PREFIX xsd: <...XMLSchema#>

<Researcher> {
  wdt:P31 [ wd:Q5 ] ;
  wdt:P19 @<Place> ;
  wdt:P569 xsd:dateTime ;
  wdt:P108 @<Employer> * ;
  p:P108 { ps:P108 @<Employer> ;
            pq:P580 xsd:dateTime ? ;
            pq:P582 xsd:dateTime ?
          } *
}
<Place> { }
<Employer> { }
```

WShEx

```
<Researcher> {
  :P31 [ :Q5 ] ;
  :P108 @<Employer> { | :P580 Time ? ,
                     | :P582 Time ?
                   } *
}
<Award> { :P17 @<Country> }
<Country> { }
```

Contents

Introduction to Knowledge graphs

Types of Knowledge Graphs:

RDF, Property graphs, Wikibase, RDF-1.2

Shaping RDF: ShEx & SHACL, DCTAP

Shaping other types of Knowledge graphs:

Shaping Wikibase and Wikidata graphs

Shaping Property Graphs

Shaping RDF 1.2

Some ideas for future work



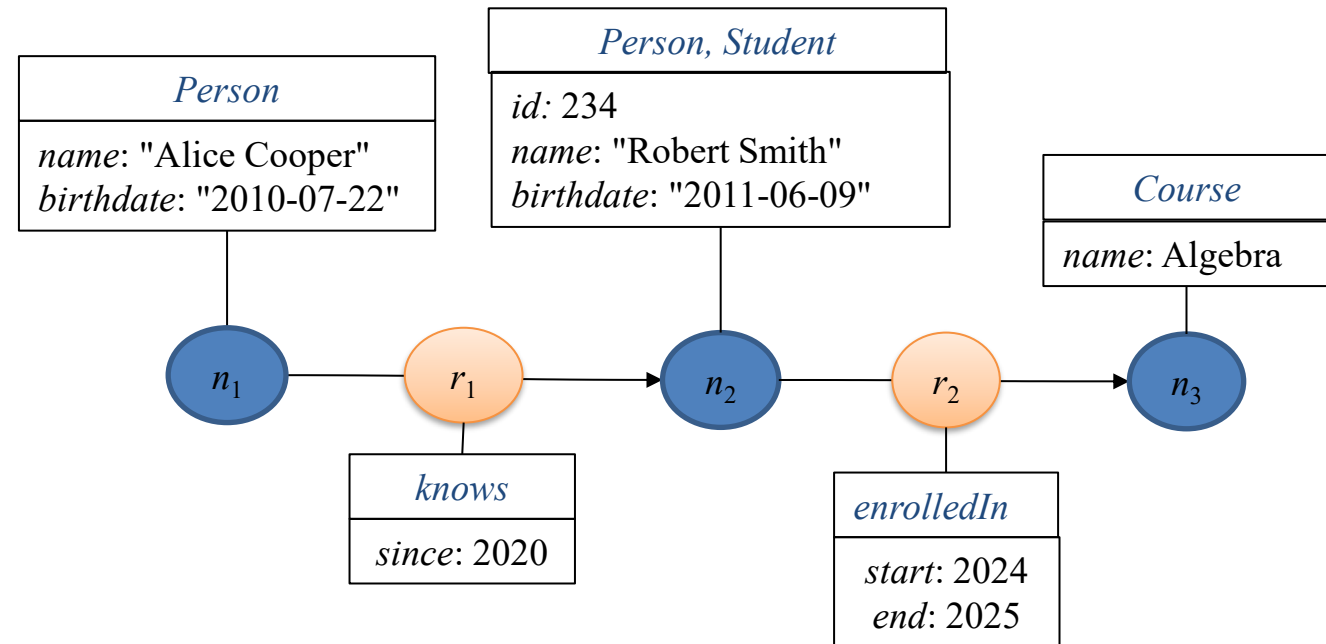
Labeled Property graphs

Graph structure with nodes and relationships (edges)

Nodes and edges can have:

- Labels
- A set of property-value pairs

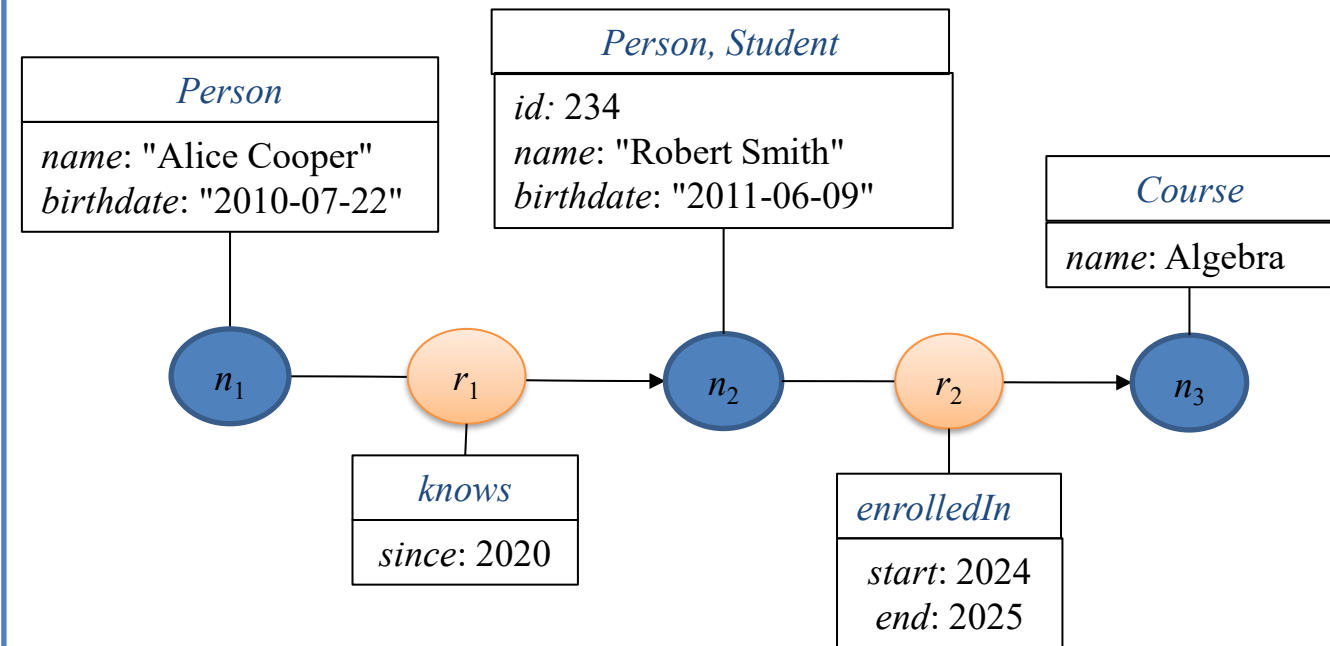
Edges can be directed/undirected



PGSchema

Paper: *PG-Schema: Schemas for property graphs*, <https://doi.org/10.1145/3589778>

```
CREATE GRAPH TYPE CourseSchema {  
  ( PersonType: Person {  
    name STRING, birthdate: DATE })  
  
  ( StudentType: @PersonType & Student {  
    id INTEGER  
  })  
  
  ( courseType: Course ) { name STRING }  
  
  (:personType)  
    -[KnowsEdgeType : knows  
      { since DATE }]-> (:personType)  
  
  (:studentType)  
    -[EnrolledEdgeType : enrolledIn {  
      start DATE,  
      end DATE } ]-> (:courseType)  
}
```



PGSchema already supported:

- Nodes & Edge type declarations
- Label declarations with conjunctions & disjunctions
- Inheritance between types

Extending PG-Schema adding property constraints

We extended PG-Schema with 3 types of constraints

- Structural constraints
- Cardinality constraints
- Range constraints

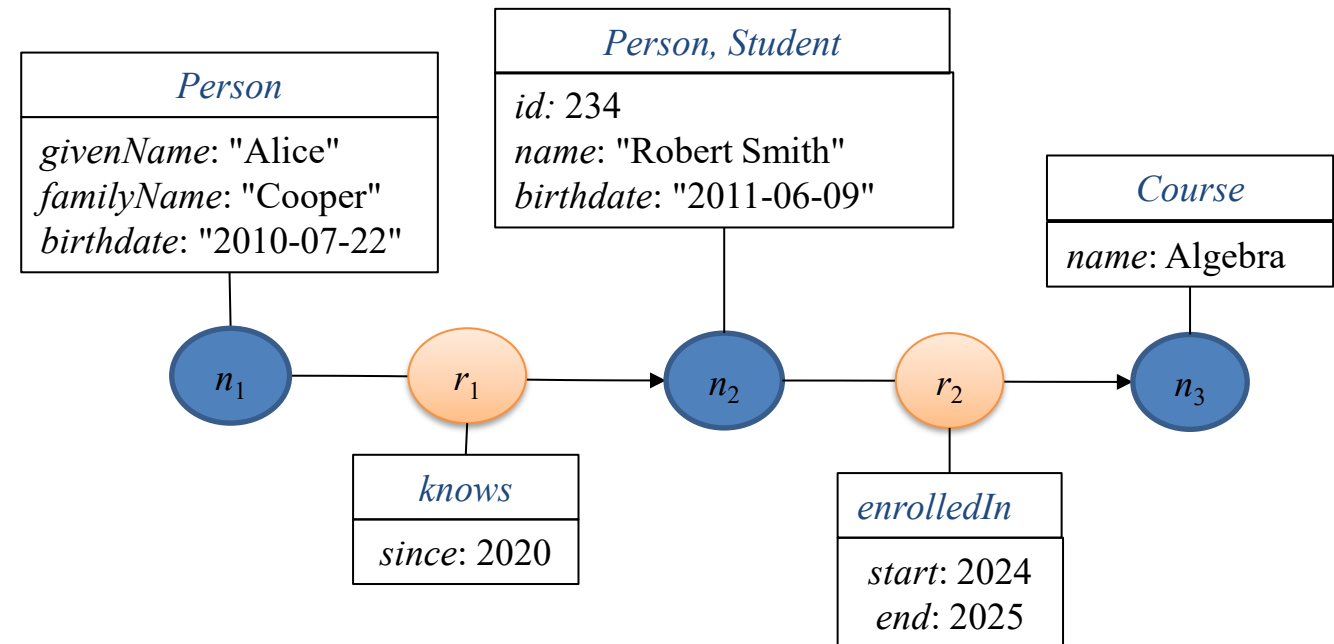
Structural constraints

Adding constraints at property-level

Alternative properties "||"

Value disjunction "|"

```
( PersonType: Person {  
  name STRING ||  
  ( givenName STRING,  
    lastName STRING  
  ),  
  birthdate: DATE  
})  
  
( StudentType: @PersonType & Student {  
  id STRING | INTEGER  
})
```



Cardinality constraints

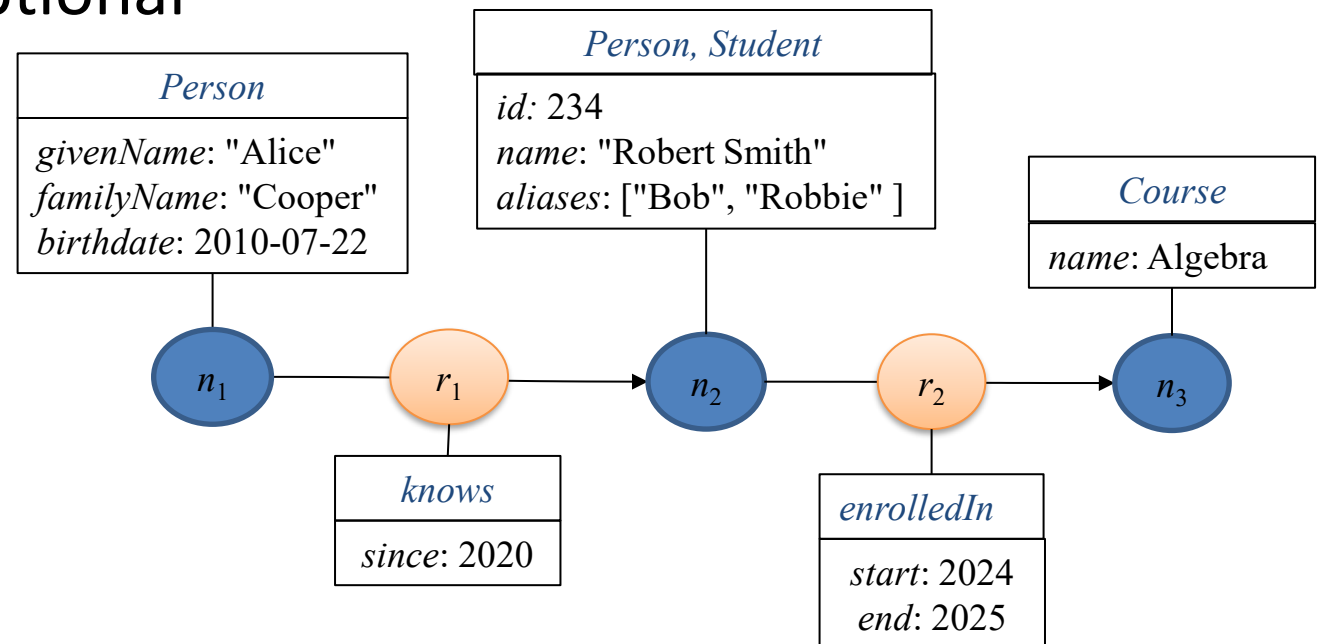
Define how many times a property type participates

$\{m, n\}$ = can be repeated between m and n times ($*$ = unbounded)

Default cardinality = no repetition: $\{1, 1\}$

OPTIONAL = the property is optional

```
(StudentType : @PersonType & Student {  
    id          STRING | INTEGER,  
    name        STRING,  
    aliases     LIST STRING {0 ,*},  
    OPTIONAL birthDate DATE  
})
```



Range constraints

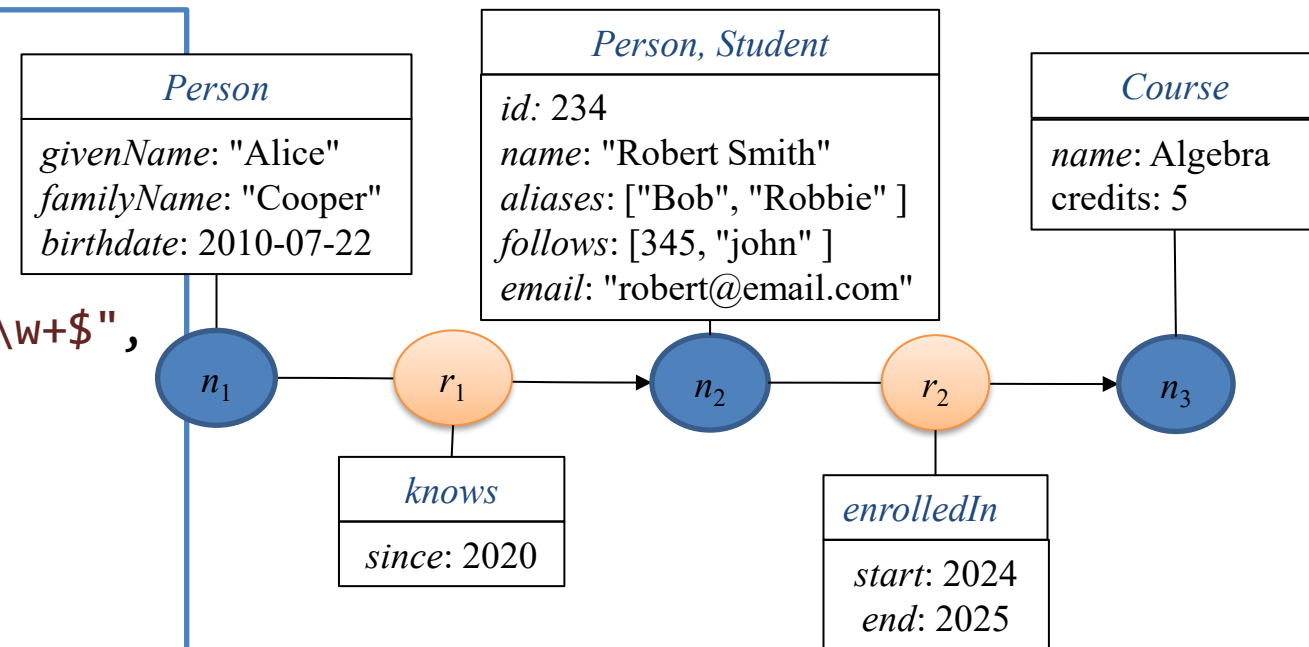
Restrict admissible property values

ANY = no restriction

CHECK cond = Datatype facets like:

Length limits, numeric bounds, regular expressions, enumerations, etc.

```
( StudentType : @PersonType & Student {  
  id      STRING | INTEGER,  
  aliases LIST STRING {0,*},  
  follows ANY,  
  email   STRING  
          CHECK REGEX "^[\\w.-]+@[\\w.-]+\\.\\w+$",  
})  
( courseType: Course ) {  
  name     STRING  
  credits  INTEGER CHECK (> 3)  
})
```



PGSchema_PC abstract grammar

Property values specification

pvs	$::=$	$\{ pv \} [OPEN]$	property-value specifier
pv	$::=$	$[OPTIONAL] p \ ts$	optional property p with ts
	$ $	pv_1, pv_2	Each of pv_1 and pv_2
	$ $	$pv_1 \mid pv_2$	One of pv_1 and pv_2
	$ $	ϵ	Empty content
ts	$::=$	$t [card]$	type t with cardinality $card$
t	$::=$	$t_1 \ \& \ t_2$	conjunction of t_1 and t_2
	$ $	$t_1 \mid t_2$	disjunction of t_1 and t_2
	$ $	$\Delta [cond]$	Datatype Δ satisfying cond
$card$	$::=$	$\{ n, max \}$	$n \in \mathbb{N}$
max	$::=$	$n \mid *$	

Semantic changes to PGSchema

Property values defined as sets of values instead of single values

By default, the values are singleton sets (cardinality = {1,1})

Record types: $\tau : \mathcal{K} \mapsto \text{Set}(\mathcal{V})$

We use Δ_m^n = a subset $\Delta' \in \text{Set}(\mathcal{V})$ where $m \leq |\Delta'| \leq n$

We adapted the semantics from the PGSchema paper

...and implemented it in Rust

PGSchema + Property Constraints in practice

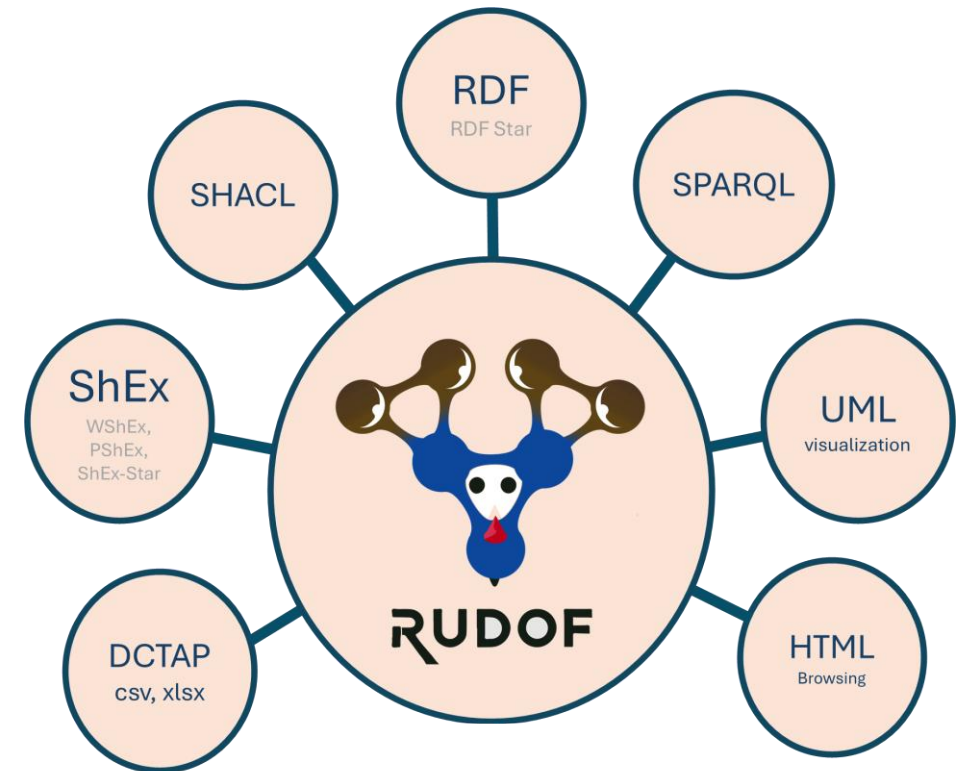
Prototype implementation in rudof

3 inputs:

- Property graph using YarsPG notation
- PGSchema declaration
- Map declaration $\approx$ shapemap

Output:

- Validation result



Running example

demo.pg

```
(n1 {Person} [
  name: "Alice",
  birthdate: DATE "2010-07-22"
])
(n2 {Person, Student} [
  id: 234,
  name: "Robert Smith",
  aliases: ["Bob", "Robbie"],
  email: "bob@mail.com"
])
(n3 {Course} [
  name: "Algebra",
  credits: 6
])
(n1) -
  (e1 {knows}
    [ since: 2020 ]) -> (n2)
(n2) -
  (e2 {EnrolledIn}
    [ start: 2024,
      end: 2025 ]) -> (n3)
```

demo.pgs

```
CREATE NODE TYPE ( PersonType : Person {
  name: STRING ||
  ( givenName: STRING, familyName: STRING ),
  OPTIONAL birthdate: DATE
}) ;
CREATE NODE TYPE ( StudentType : @PersonType & Student {
  id: INTEGER | STRING,
  email: STRING CHECK REGEX "^[\\w.-]+@[\\w.-]+\\.\\w+$",
  aliases: STRING *
}) ;
CREATE NODE TYPE ( CourseType: Course {
  name: STRING,
  credits: INTEGER CHECK (> 3)
});
CREATE EDGE TYPE
  (:PersonType)
    -[ KnowsType : Knows {
      since: INTEGER }]-> (:PersonType);
CREATE EDGE TYPE
  (:StudentType)
    -[ EnrolledInType : EnrolledIn {
      start: INTEGER,
      end: INTEGER }]-> (:CourseType)
```

demo.map

```
n1:PersonType,
n2:StudentType,
n3:CourseType,
e1:KnowsType,
e2:EnrolledInType
```

```
$ rudof validate --mode pgschema --schema demo.pgs --shapemap demo.map demo.pg
Result - valid?: true
n1:PersonType: true
n2:StudentType: true
. . .
```

Future work (1)

Adding cardinality constraints to Edge types

Express more complex topology definitions $\rightarrow$ ShEx|SHACL

Example:

```
CREATE EDGE TYPE
(:PersonType)
-[ KnowsType : Knows
  { since: INTEGER }]-> (:PersonType) {0,*}
```

```
CREATE EDGE TYPE
(:StudentType)
-[ EnrolledInType : EnrolledIn
  { start: INTEGER,
    end : INTEGER }]-> (:CourseType) {1,5}
```

Future work (2): Practical adoption

Integrate with existing Labeled property graph implementations

Neo4j, Amazon Neptune, Fabric, etc.

Looking for a common access protocol or API

Increase the expressivity of current maps

Selection of nodes/edges $\approx$ query

Performance & benchmarks

Contents

Introduction to Knowledge graphs

Types of Knowledge Graphs:

RDF, Property graphs, Wikibase, RDF-1.2

Shaping RDF: ShEx & SHACL, DCTAP

Shaping other types of Knowledge graphs:

Shaping Wikibase and Wikidata graphs

Shaping Property Graphs

Shaping RDF 1.2

Some ideas for future work



ShEx for RDF 1.2

Note: work in progress...(rudof's roadmap)

```
:timbl :name "Tim Berners Lee";  
      :employer :CERN {  
        :start 1980;  
        :end 1980  
      } {  
        :start 1984;  
        :end 1994  
      } .
```

```
<Researcher> {  
  :name xsd:string ;  
  :employer @<Org> {  
    :start xsd:date ,  
    :end xsd:date ?  
  } *  
}  
<Org> {  
  :name xsd:string ;  
}
```

SHACL 1.2

Currently under development at W3C: <https://www.w3.org/TR/shacl12-core/>

```
:timbl :name "Tim Berners Lee";  
      :employer :CERN {  
        :start 1980;  
        :end 1980  
      } {  
        :start 1984;  
        :end 1994  
      } .
```

```
:UserShape a sh:NodeShape ;  
  sh:targetClass :User ;  
  sh:property [  
    sh:path :employer ;  
    sh:reificationRequired true ;  
    sh:reifierShape [  
      sh:property [  
        sh:path :start ;  
        sh:datatype xsd:integer ;  
        sh:minCount 1 ; sh:maxCount 1  
      ] ;  
      sh:property [  
        sh:path :end ;  
        sh:datatype xsd:integer ;  
        sh:minCount 0 ; sh:maxCount 1  
      ] ] .
```

New
The example is already supported in rudof

Try it: <https://tinyurl.com/34pthj95>

Contents

Introduction to Knowledge graphs

Types of Knowledge Graphs:

RDF, Property graphs, Wikibase, RDF-Star

Shaping RDF: ShEx & SHACL, DCTAP

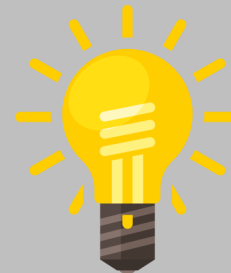
Shaping other types of Knowledge graphs:

Shaping Wikibase and Wikidata graphs

Shaping Property Graphs

Shaping RDF 1.2

Some ideas for future work



ShEx/SHACL in practice



Static analysis and performance

Benchmarking and optimizing rudof's SHACL implementation

Extending and Optimizing the rudof SHACL validator, Á. García, S. Bustamante, Jose E. Labra, O. Corcho

DMKG2026, 2nd International Workshop on Data Management for Knowledge Graphs, ISWC Conference 2026, Bari, Italy

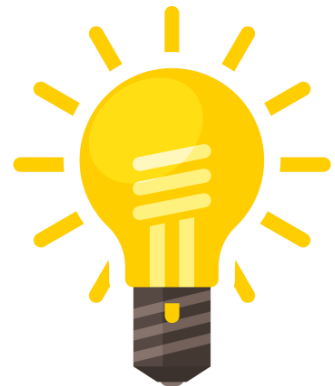
Hypothesis: Some subsets of ShEx/SHACL can be very performant

Improve validation reporting and messages

Hypothesis: Validation results $\approx$ proof tree

Incremental and streaming validation

Hypothesis: Derivatives algorithm can work in streaming mode



Property Graphs

Adding cardinality constraints to Edge types

Express more complex topology definitions $\rightarrow$ ShEx|SHACL

Integrate with existing implementations

Neo4j, Amazon Neptune, Microsoft Fabric, etc.

Increase the expressivity of current type maps

Selection of nodes/edges $\approx$ GQL query

Performance & benchmarks



Wikibase & Wikidata

Performance and validation in practice

Leverage on QLever (https://rudof-project.github.io/rudof/cli_usage/backend.html)

Caching validation results & parallel validation

Representing property constraints as Shapes

WShEx

Implement WShEx in rudof

Subsetting approaches based on shapes

Port wdsup to rudof



Convergence on Graph Schema languages

Common Foundations for SHACL, ShEx, and PG-Schema, S. Ahmetaj, I. Boneva, J. Hidders, K. Hose, M. Jakubowski, J. Labra, W. Martens, F. Mogavero, F. Murlak, C. Okulmus, A. Polleres, O. Savković, M. Šimkus, D. Tomaszuk, International World Wide Web Conference, Sidney, Australia, 2025

https://labra.weso.es/publication/2025_common_foundations_shacl_shex_pgschema/

Common Foundations for Recursive Shape Languages, S. Ahmetaj, I. Boneva, J. Hidders, M. Jakubowski, J. Labra, W. Martens, F. Mogavero, F. Murlak, C. Okulmus, O. Savković, M. Šimkus, D. Tomaszuk, 23rd International Conference on Principles of Knowledge Representation and Reasoning, Lisbon, 2026

sheval: A Framework for Evaluating Shape Languages, S. Ahmetaj, I. Boneva, J. Hidders, M. Jakubowski, J. Labra, W. Martens, F. Murlak, C. Okulmus, O. Savković, M. Šimkus, D. Tomaszuk, DMKG2026, 2nd International Workshop on Data Management for Knowledge Graphs, ISWC Conference 2026, Bari, Italy

Translating RDF 1.2 Graphs to Property Graphs, M. Jakubowski, D. Tomaszuk, D. Fernández, R. Taelman, R. Dedecker, J. Labra, K. Hose, DMKG2026, 2nd International Workshop on Data Management for Knowledge Graphs, ISWC Conference 2026, Bari, Italy



Combination with other technologies

Shapes + Rules

Combining shapes and rules

SHACL 1.2 rules vs SPARQL Rules

Inferring/Extracting Shapes from existing graphs

Implement sheXer in Rust, extend sheXer to Property graphs

Knowledge Graph Construction

Implement ShExML in Rust



Conclusions

Several types of Knowledge Graphs

Different models

Could they converge in the future?

Shapes can increase quality of KGs

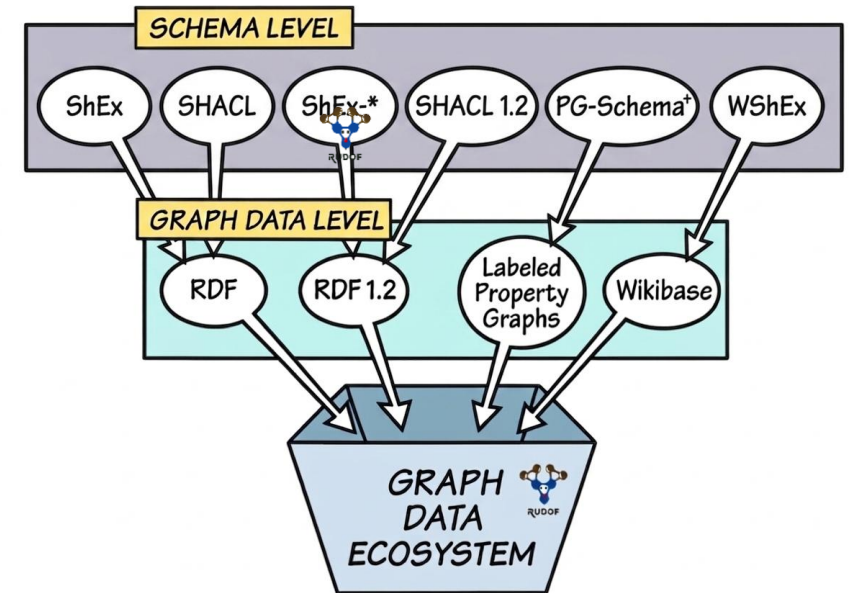
Different types of KGs

Wikibase graphs: ShEx, WShEx

Property graphs: PGSchema, GQL, ...

RDF 1.2: ShEx (next), SHACL 1.2

Could the shapes languages converge in the future?



End of presentation



ShEx vs SPARQL

SPARQL pros:

Expressive

Ubiquitous

SPARQL cons:

Expressive

Idiomatic - many ways to encode the same constraint

Non recursive

```
<User> {  
  schema:name    xsd:string ;  
  schema:gender [ schema:Female schema:Male]  
}
```

```
ASK {{ SELECT ?Person {  
          ?Person schema:name ?o .  
        } GROUP BY ?Person HAVING (COUNT(*)=1)  
}  
{ SELECT ?Person {  
          ?Person schema:name ?o .  
          FILTER ( isLiteral(?o) &&  
                   datatype(?o) = xsd:string )  
        } GROUP BY ?Person HAVING (COUNT(*)=1)  
}  
{ SELECT ?Person (COUNT(*) AS ?c1) {  
          ?Person schema:gender ?o .  
        } GROUP BY ?Person HAVING (COUNT(*)=1)}  
{ SELECT ?Person (COUNT(*) AS ?c2) {  
          ?S schema:gender ?o .  
          FILTER ((?o = schema:Female ||  
                   ?o = schema:Male))  
        } GROUP BY ?Person HAVING (COUNT(*)=1)}  
FILTER (?c1 = ?c2)  
}
```