RDF Triples Storage and Query

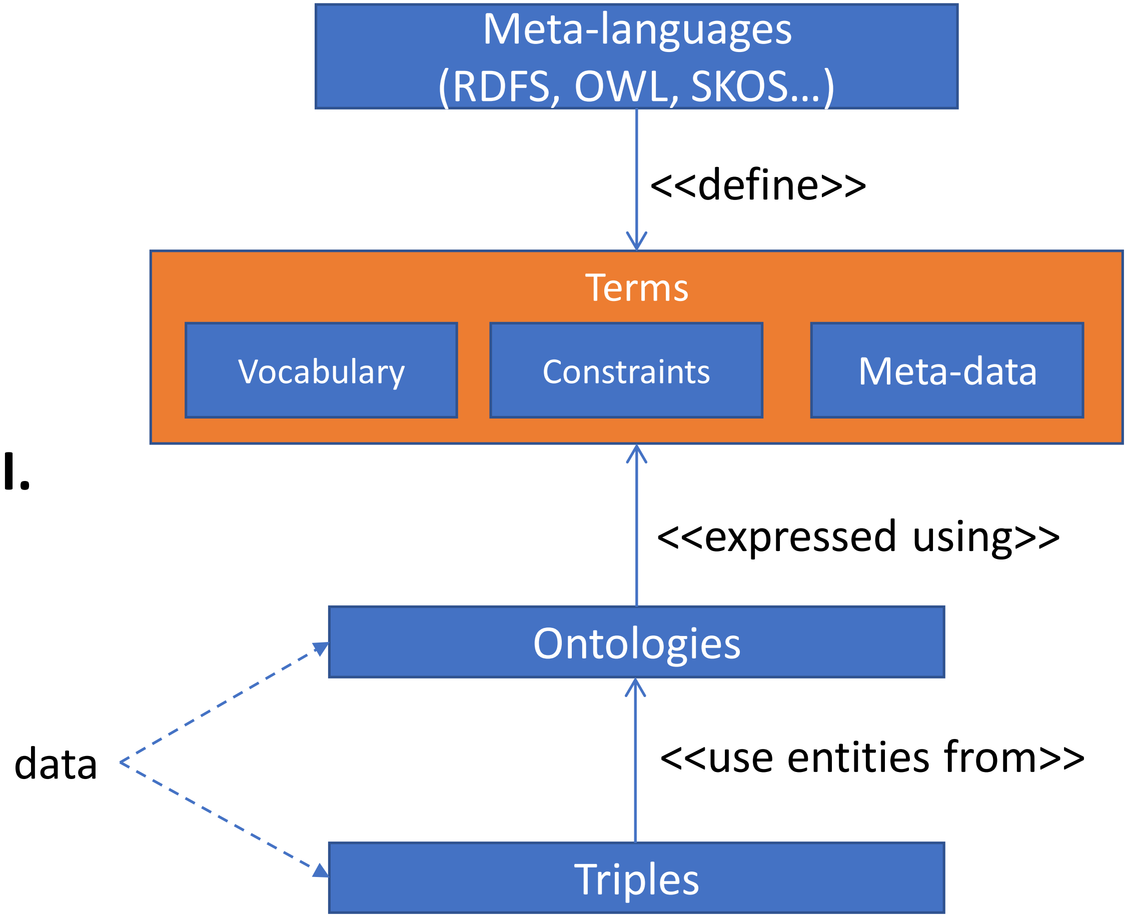
Chin Zi Hau

Introduction

- **Triple** is the basic element of knowledge representation

  <**subject**> <**predicate**> <**object**>

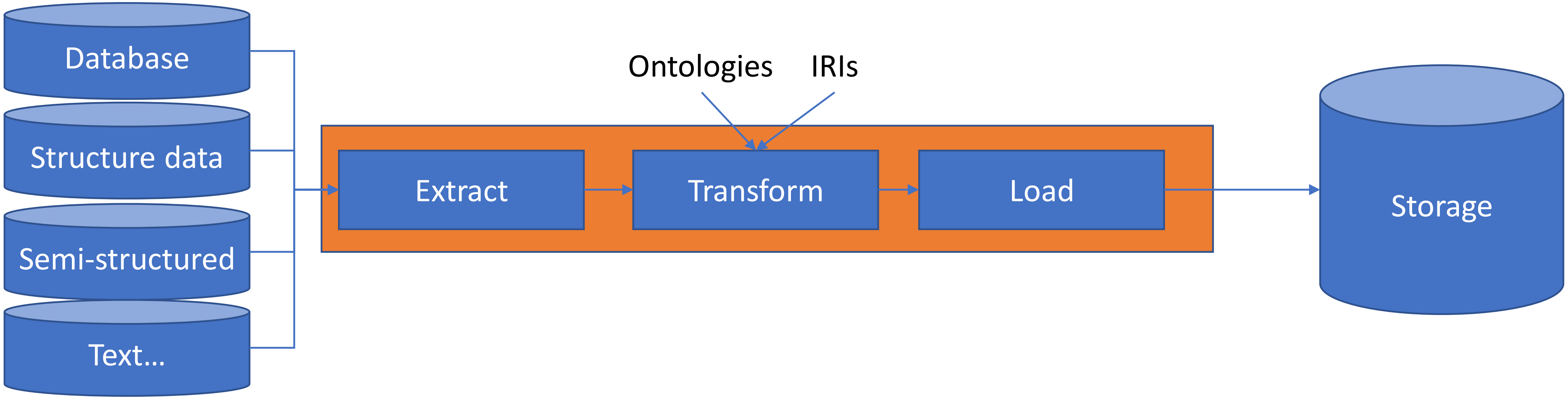
  - Such triples are called **RDF statements.**
- Subjects, predicates and objects are **resources.**
  - Subjects and predicates are represented by **IRI.**
  - Object are represented by IRI/**literals/bnode.**
- An **ontology** defines the entities used for resources (subjects, predicates, objects…).
- Ontologies are expressed using formal **metalanguages** such as RDFS and OWL.
- Metalanguages define *vocabularies* and *constraints* used to express ontologies.



Creating Triples

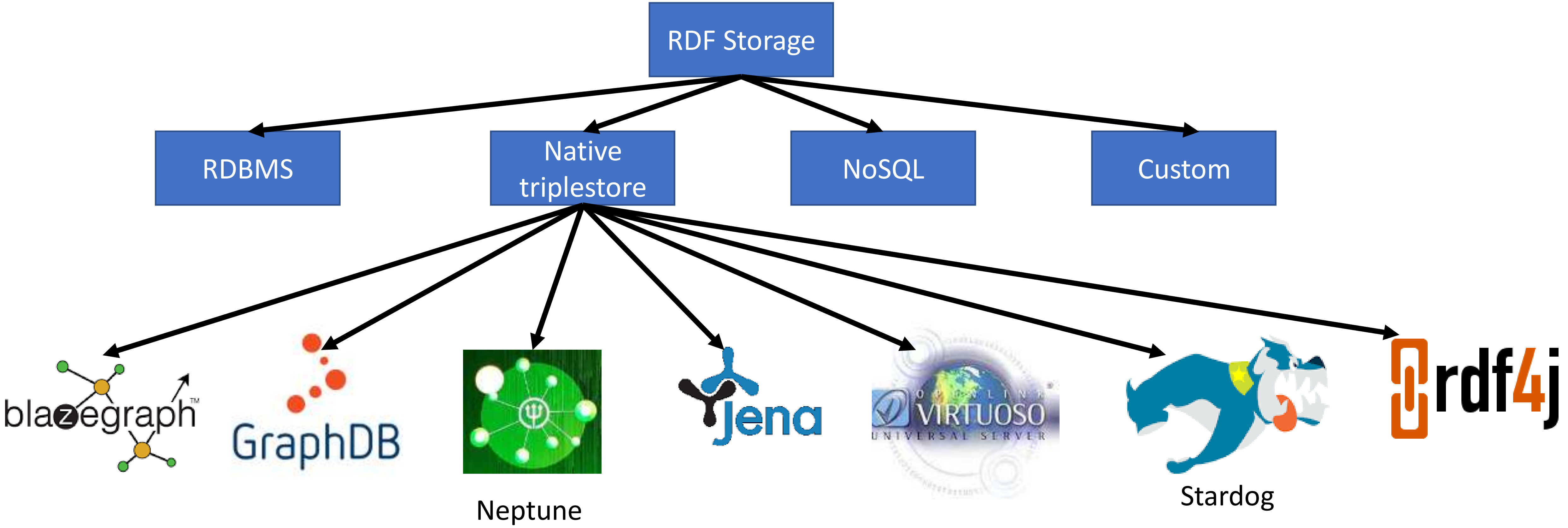
- Having an ontology, the objective is usually to "populate it" with ***triples***.
- Creating triples manually is possible but is **slow** and **inefficient**, so automated tools are used to convert data into RDF.
- These tools follow an **Extract, Transform, Load (ETL)** pattern, where data from different sources is mapped to the ontology and then loaded as triples.

Data Source	Tools
RDB	R2RML, triplify, D2RQ, ODEMapster, Datalift…
XML	GRDDL, Xsparql…
xls,csv…	Google refine, Any23, QuidiCRC, Lionel…
Text	Dog4dag, Gate, Fred, OntoLing, LexOnt
frameworks	Coeus, marimba, DataTank,



Storing Triples

- Triples are graphs.
- They can be stored using different technologies.



RDF Storage

RDBMS

Native triplestore

NoSQL

Custom

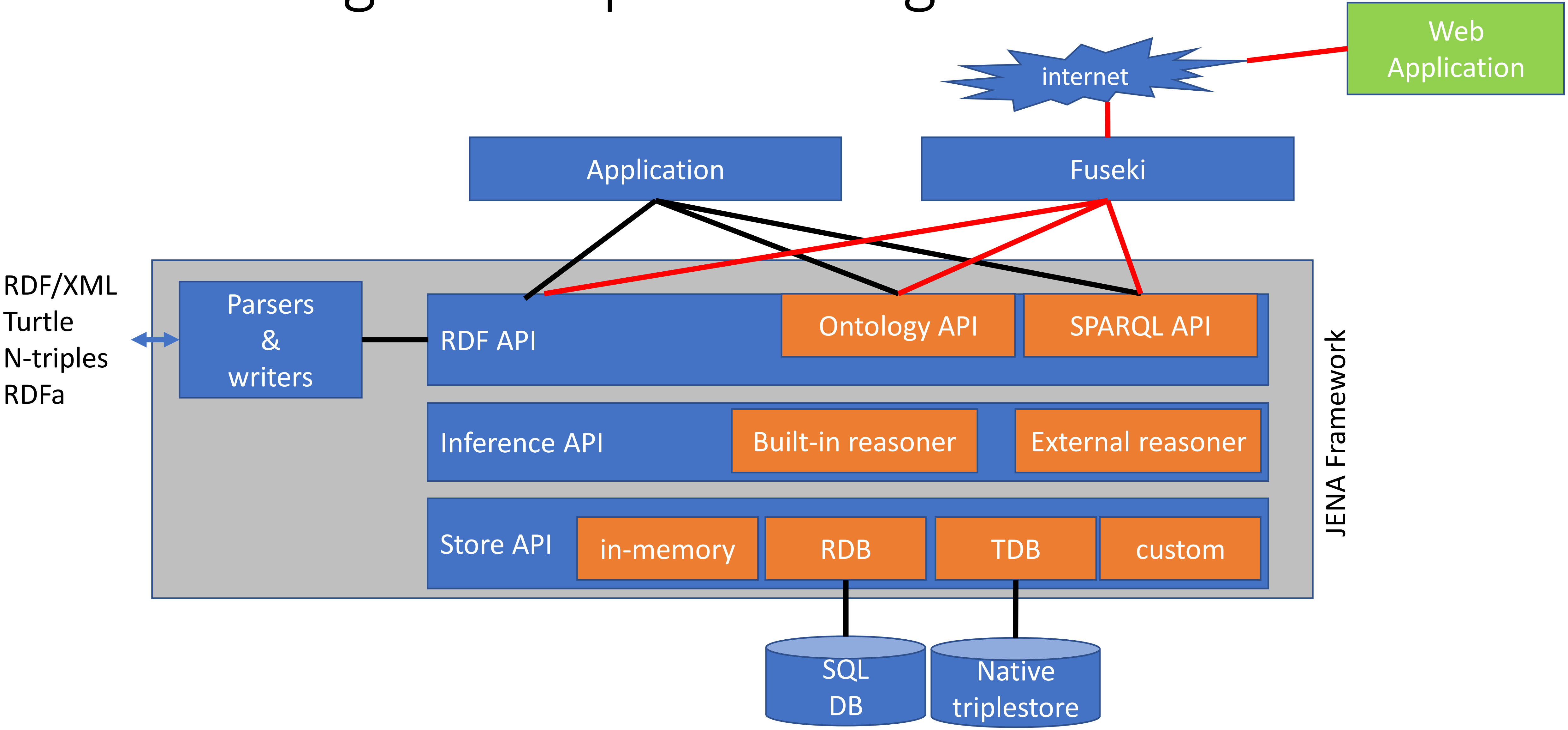
Neptune

Stardog

Triplestore (RDF store)

- Used to **store RDF data**
- Can **manage multiple datasets**
  - Each dataset contains **one or more graphs**
- Supports **SPARQL** queries
- Allows **inferencing** through built in or external reasoners
- Can **query across multiple graphs**

Accessing RDF Triples through Jena

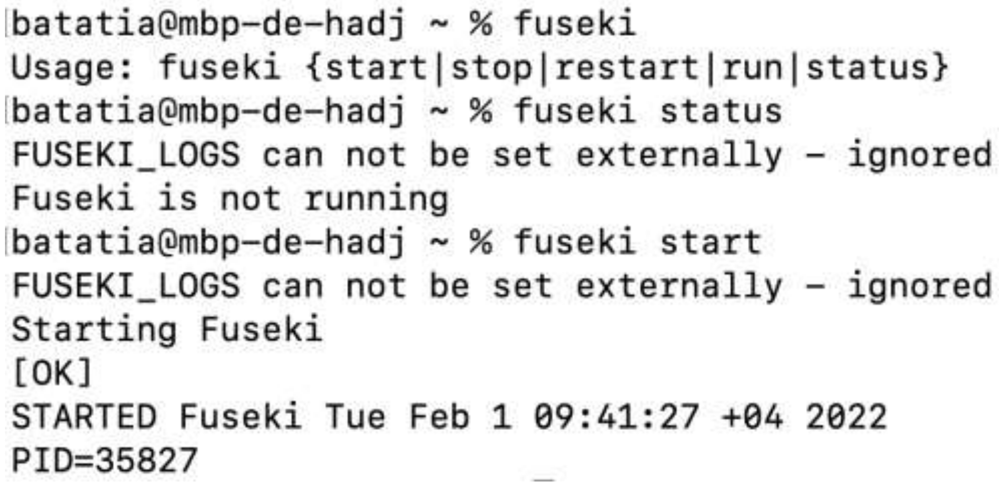


Fuseki

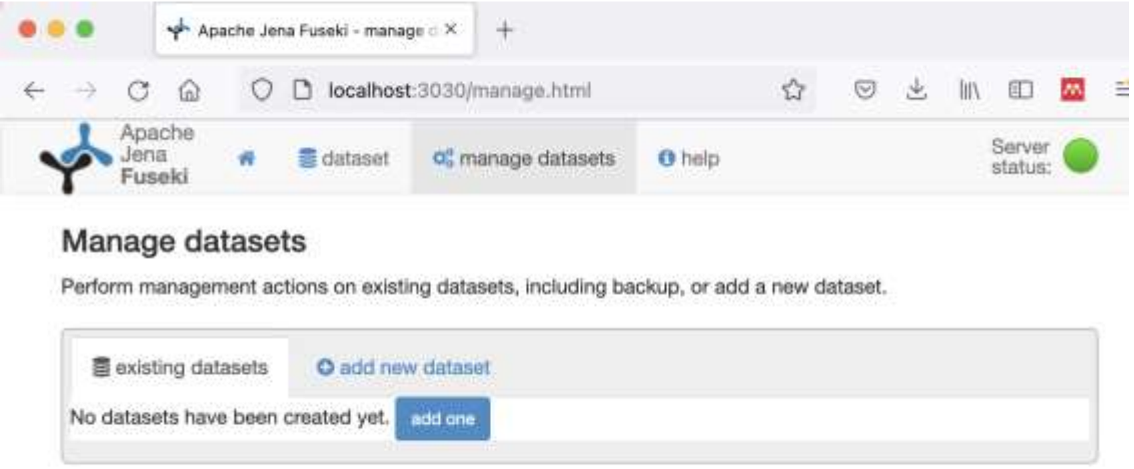
- A SPARQL server based on JENA.
- Can run as: a **system service**, a standalone server, or a web application.
- Based on Jetty (embedded web server and java servlet container).
- Comes with TDB2 (triplestore).
- Can be installed as a system service.

Running Fuseki

- Fuseki is a server
- Start/stop it using a terminal window
  - Example MacOS: fuseki start, fuseki stop, fuseki restart
- Interact with Fuseki from a web page by connecting to
  - http://localhost:3030



```
batatia@mbp-de-hadj ~ % fuseki
Usage: fuseki {start|stop|restart|run|status}
batatia@mbp-de-hadj ~ % fuseki status
FUSEKI_LOGS can not be set externally - ignored
Fuseki is not running
batatia@mbp-de-hadj ~ % fuseki start
FUSEKI_LOGS can not be set externally - ignored
Starting Fuseki
[OK]
STARTED Fuseki Tue Feb 1 09:41:27 +04 2022
PID=35827
```



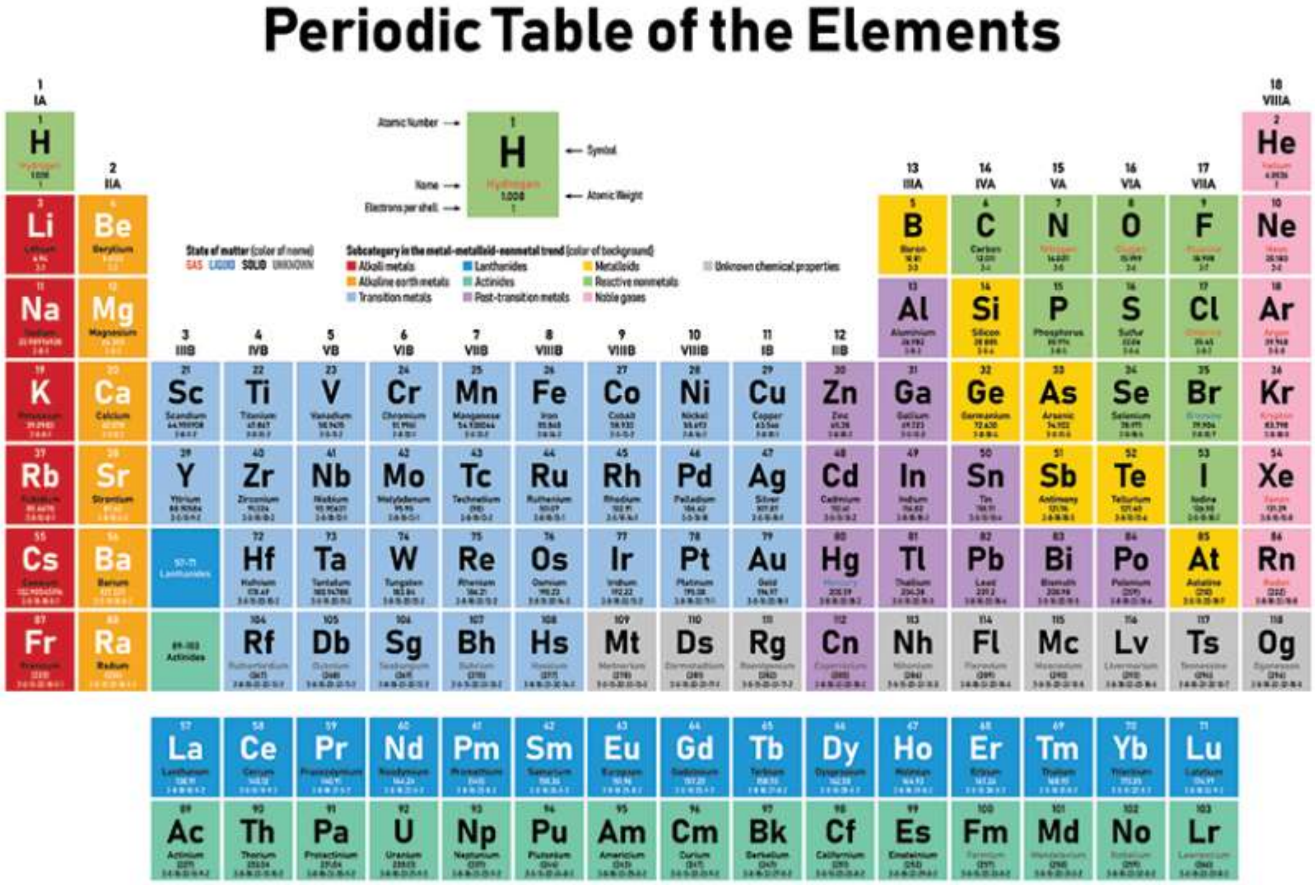
Manage datasets

Perform management actions on existing datasets, including backup, or add a new dataset.

No datasets have been created yet.

Experimental Dataset

- Periodic table is a dataset that describes basic chemical materials
- All chemical elements are described with name, class, atomic number, atomic weight…



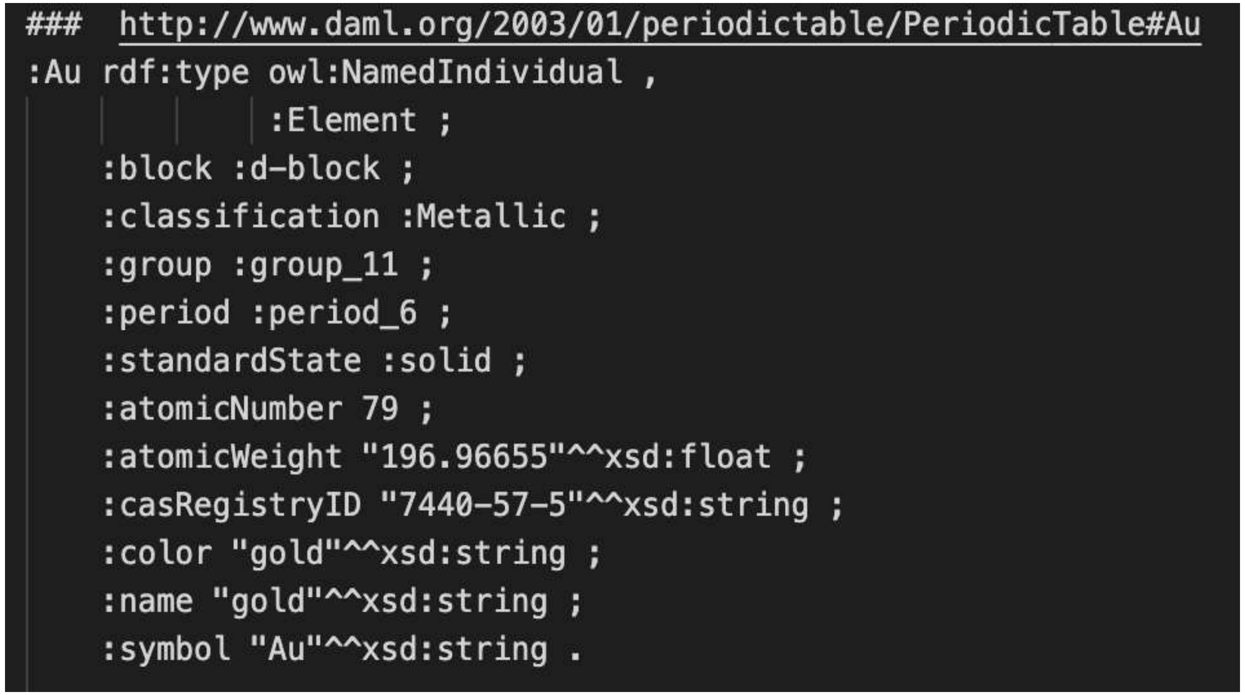
Periodic Table of the Elements
Atomic Number
Symbol
Name
Atomic Weight
Electrons per shell
State of matter (color of name)
GAS LIQUID SOLID UNKNOWN
Subcategory in the metal-metalloid-nonmetal trend (color of background)
Alkali metals
Alkaline earth metals
Transition metals
Lanthanides
Actinides
Post-transition metals
Metalloids
Reactive nonmetals
Noble gases
Unknown chemical properties
1 IA
2 IIA
3 IIIB
4 IVB
5 VB
6 VIB
7 VIIB
8 VIIIB
9 VIIIB
10 VIIIB
11 IB
12 IIB
13 IIIA
14 IVA
15 VA
16 VIA
17 VIIA
18 VIIIA
H He
Li Be B C N O F Ne
Na Mg Al Si P S Cl Ar
K Ca Sc Ti V Cr Mn Fe Co Ni Cu Zn Ga Ge As Se Br Kr
Rb Sr Y Zr Nb Mo Tc Ru Rh Pd Ag Cd In Sn Sb Te I Xe
Cs Ba 57-71 Lanthanides Hf Ta W Re Os Ir Pt Au Hg Tl Pb Bi Po At Rn
Fr Ra 89-103 Actinides Rf Db Sg Bh Hs Mt Ds Rg Cn Nh Fl Mc Lv Ts Og
La Ce Pr Nd Pm Sm Eu Gd Tb Dy Ho Er Tm Yb Lu
Ac Th Pa U Np Pu Am Cm Bk Cf Es Fm Md No Lr

Exploring the Dataset

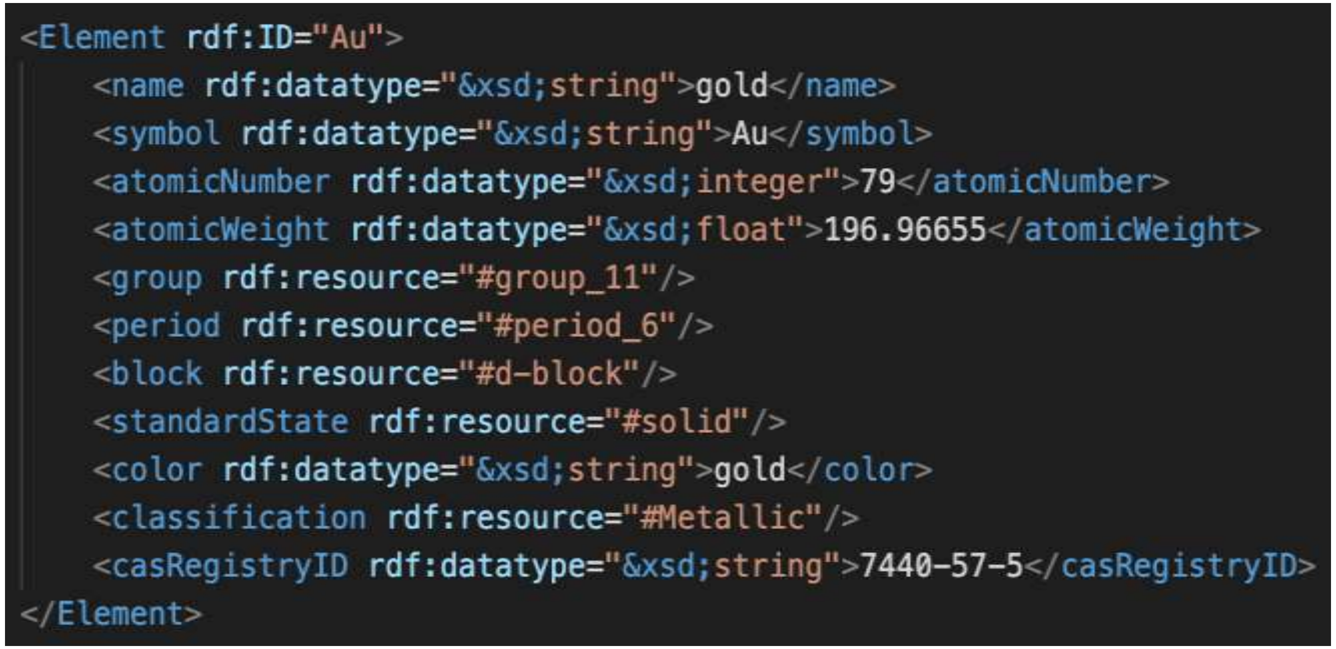
- Download the **PeiordicTable.owl**

http://www.daml.org/2003/01/periodictable/

- Display the file using a text editor, like VS code and inspect the content
- Notice the entity IRI and the value of its data type symbol look the same
  - The IRI is the identifier
  - The symbol is a string



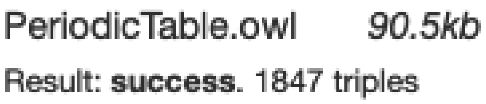
```
###  http://www.daml.org/2003/01/periodictable/PeriodicTable#Au
:Au rdf:type owl:NamedIndividual ,
             :Element ;
    :block :d-block ;
    :classification :Metallic ;
    :group :group_11 ;
    :period :period_6 ;
    :standardState :solid ;
    :atomicNumber 79 ;
    :atomicWeight "196.96655"^^xsd:float ;
    :casRegistryID "7440-57-5"^^xsd:string ;
    :color "gold"^^xsd:string ;
    :name "gold"^^xsd:string ;
    :symbol "Au"^^xsd:string .
```



```
<Element rdf:ID="Au">
    <name rdf:datatype="&xsd;string">gold</name>
    <symbol rdf:datatype="&xsd;string">Au</symbol>
    <atomicNumber rdf:datatype="&xsd;integer">79</atomicNumber>
    <atomicWeight rdf:datatype="&xsd;float">196.96655</atomicWeight>
    <group rdf:resource="#group_11"/>
    <period rdf:resource="#period_6"/>
    <block rdf:resource="#d-block"/>
    <standardState rdf:resource="#solid"/>
    <color rdf:datatype="&xsd;string">gold</color>
    <classification rdf:resource="#Metallic"/>
    <casRegistryID rdf:datatype="&xsd;string">7440-57-5</casRegistryID>
</Element>
```

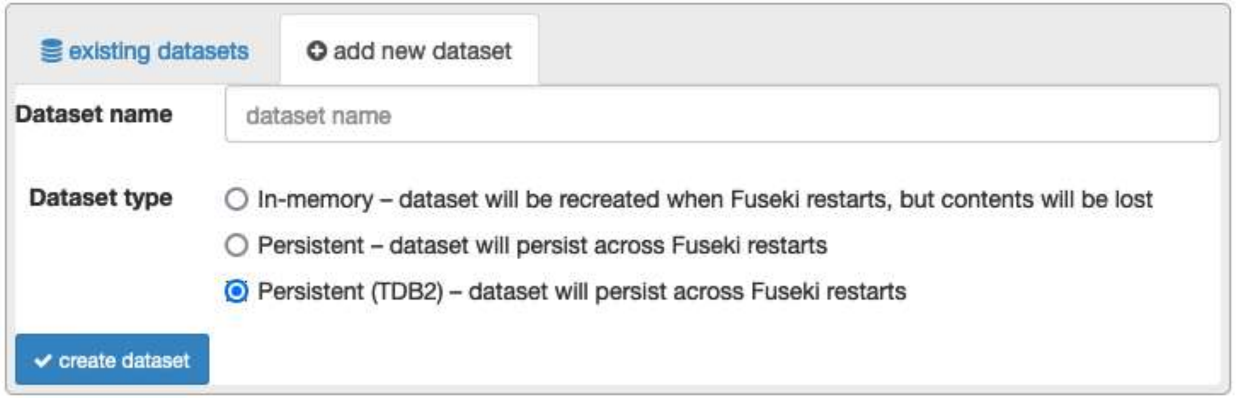
Fuseki: create dataset

- Fuseki can manage several datasets
- Datasets can be added/removed…
- You can add a new dataset by clicking add new dataset
- Upload the Periodic Table ontology upload data then upload now
- Check the metrics



PeriodicTable.owl *90.5kb*

Result: **success.** 1847 triples



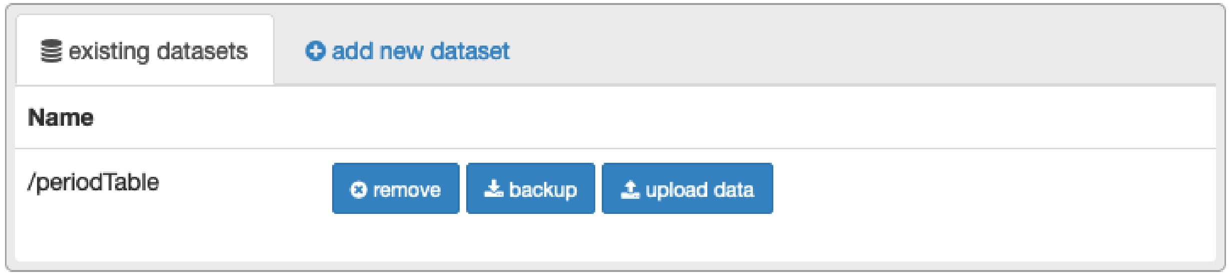
Dataset name dataset name

Dataset type

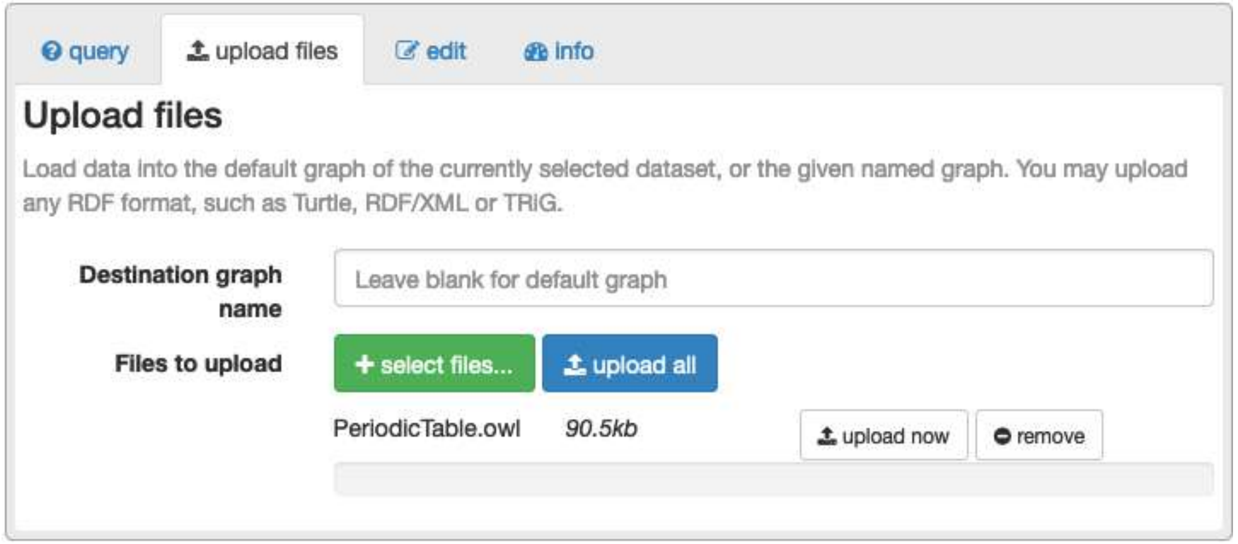
○ In-memory – dataset will be recreated when Fuseki restarts, but contents will be lost

○ Persistent – dataset will persist across Fuseki restarts

● Persistent (TDB2) – dataset will persist across Fuseki restarts



Name	
/periodTable	remove backup upload data



Upload files

Load data into the default graph of the currently selected dataset, or the given named graph. You may upload any RDF format, such as Turtle, RDF/XML or TRIG.

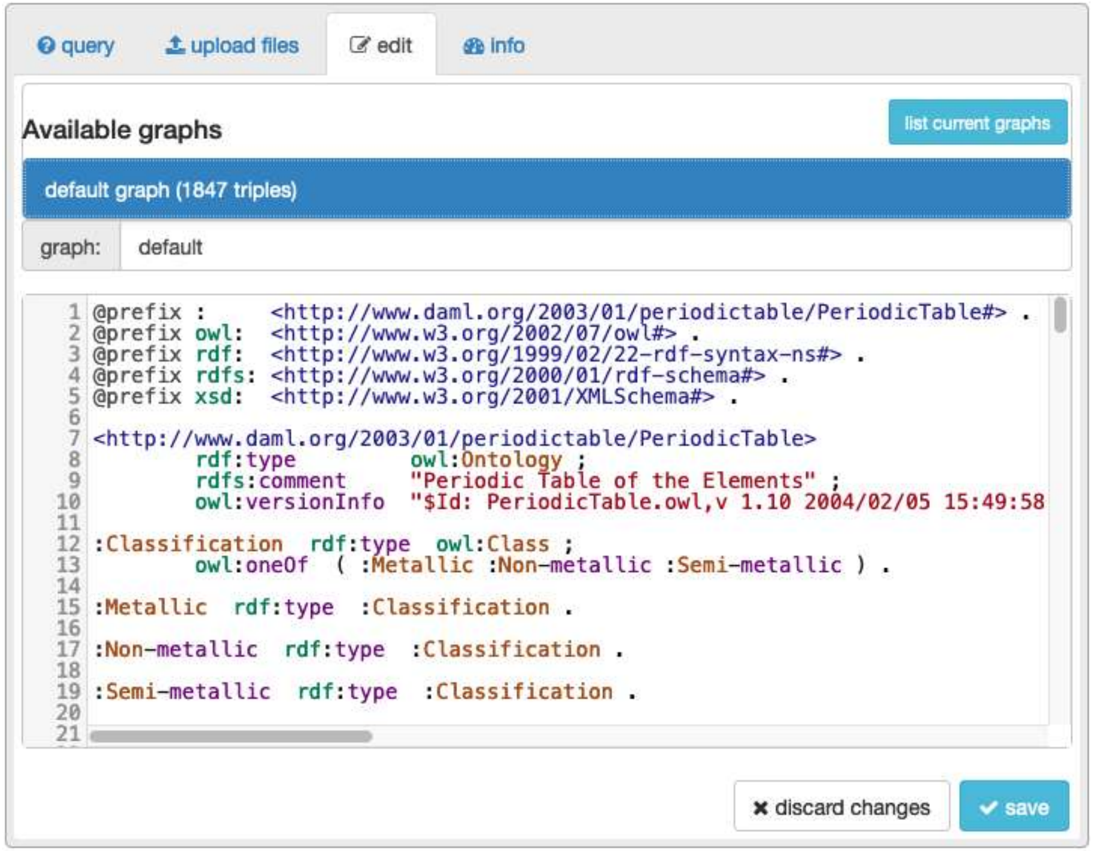
Destination graph name Leave blank for default graph

Files to upload

PeriodicTable.owl *90.5kb*

Fuseki: editing

- Fuseki allows editing the ontology
- Click edit
- And list current graphs



Available graphs

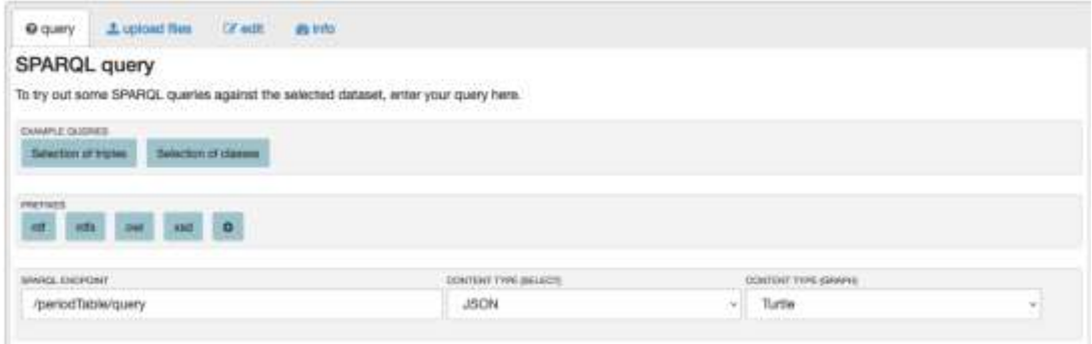
default graph (1847 triples)

graph: default

```
@prefix :     <http://www.daml.org/2003/01/periodictable/PeriodicTable#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .

<http://www.daml.org/2003/01/periodictable/PeriodicTable>
        rdf:type          owl:Ontology ;
        rdfs:comment      "Periodic Table of the Elements" ;
        owl:versionInfo   "$Id: PeriodicTable.owl,v 1.10 2004/02/05 15:49:58

:Classification  rdf:type  owl:Class ;
        owl:oneOf  ( :Metallic :Non-metallic :Semi-metallic ) .

:Metallic  rdf:type  :Classification .

:Non-metallic  rdf:type  :Classification .

:Semi-metallic  rdf:type  :Classification .

```

Fuseki: Querying

- Fuseki is a triple store server.
- It allows querying the triples using SPARQL
- The window shows examples queries
- It allows adding prefixes
- Typing queries and running them
- Viewing and manipulating results (formatting, exporting, searching…)



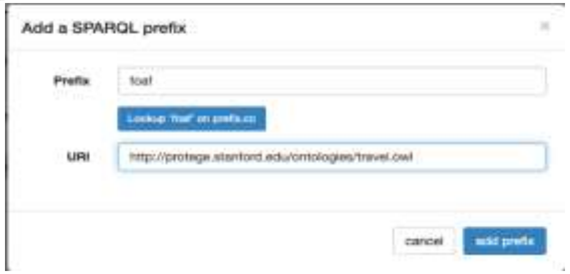
SPARQL query

To try out some SPARQL queries against the selected dataset, enter your query here.

SPARQL ENDPOINT: /periodTable/query

CONTENT TYPE (SELECT): JSON

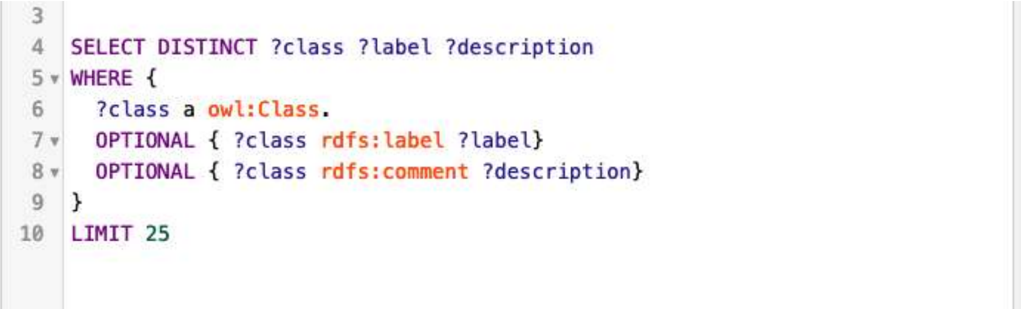
CONTENT TYPE (GRAPH): Turtle



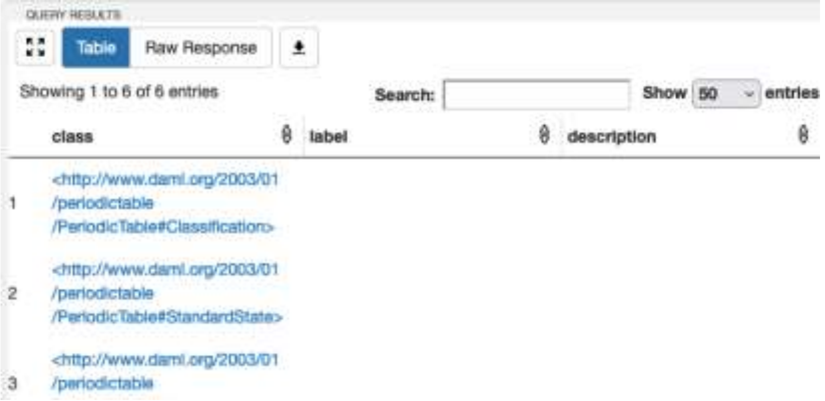
Add a SPARQL prefix

Prefix: foaf

URI: http://protege.stanford.edu/ontologies/travel.owl



```
SELECT DISTINCT ?class ?label ?description
WHERE {
  ?class a owl:Class.
  OPTIONAL { ?class rdfs:label ?label}
  OPTIONAL { ?class rdfs:comment ?description}
}
LIMIT 25
```



Showing 1 to 6 of 6 entries

Search:

Show 50 entries

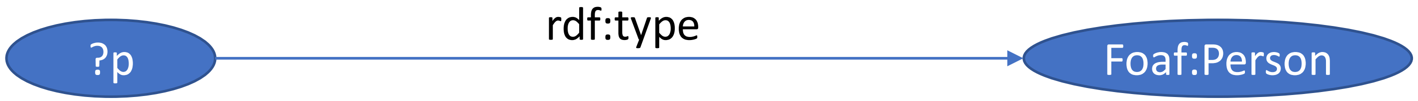
	class	label	description
1	<http://www.daml.org/2003/01/periodictable/PeriodicTable#Classification>		
2	<http://www.daml.org/2003/01/periodictable/PeriodicTable#StandardState>		
3	<http://www.daml.org/2003/01/periodictable		

SPARQL

- ***SPARQL*** is the ***query language*** for ***RDF*** data**.**
- **It's like SQL, but for graph data.**
  - Syntax is similar, but *SPARQL works with triples and relationships*, *not tables*.
- SPARQL Standard: Four Key Parts
  - **Query Language:** retrieve data.
  - **Protocol:** send queries to a server and receives results.
  - **Query Results:** The format of the data returned by the query (e.g., XML, JSON, CSV, …)
  - **Update Language:** Add, modify or delete data in RDF graph**.**
- SPARQL Endpoint
  - Web service that implements the SPQARL protocol
    - (server with triplestore that accepts remove SPQARL queries)

Principle of SPARQL Queries

- SPARQL is based on ***pattern matching***.
  - You ***describe the pattern*** you want**.** This is your **SPARQL query**.
  - The SPARQL engine ***searches the graph*****.**
  - The ***results*** are the ***parts of the graph*** that ***match the pattern*****.**
- Example:
  - *Goal:* Find all people (foaf:Person) in the graph.
    - ***?p** is a variable.*
    - ***rdf:type** = "is a type of"*
    - ***foaf:Person** = specific type/pattern*



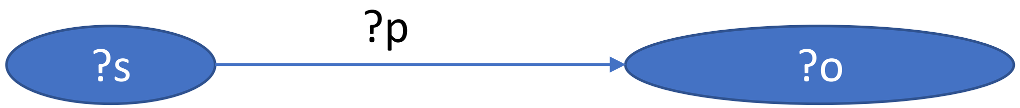
Example 1

```
PREFIX pep: <http://com.intrinsec//ontology#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?p
WHERE {
  ?p rfd:type owl:Person.
}
```

Example 2: Select All Triples

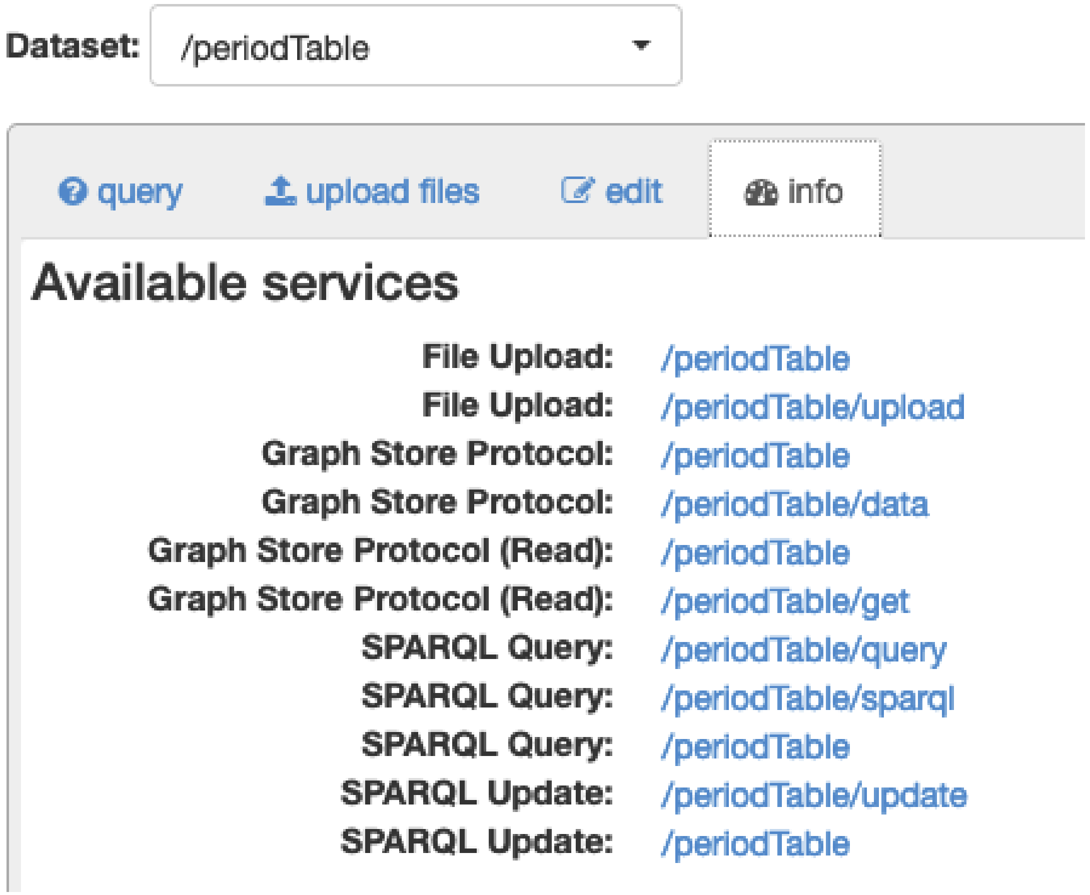
```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

prefix owl: <http://www.w3.org/2002/07/owl#>


SELECT ?subject ?predicate ?object

WHERE {

  ?subject ?predicate ?object

}
```

Very general pattern.
Matches all triples.
Typical first query when querying a new endpoint



Fuseki SPARQL endpoint (web services)

Fuseki exposes your RDF dataset as a remote SPARQL service, allowing anyone to send SPARQL queries over the web.

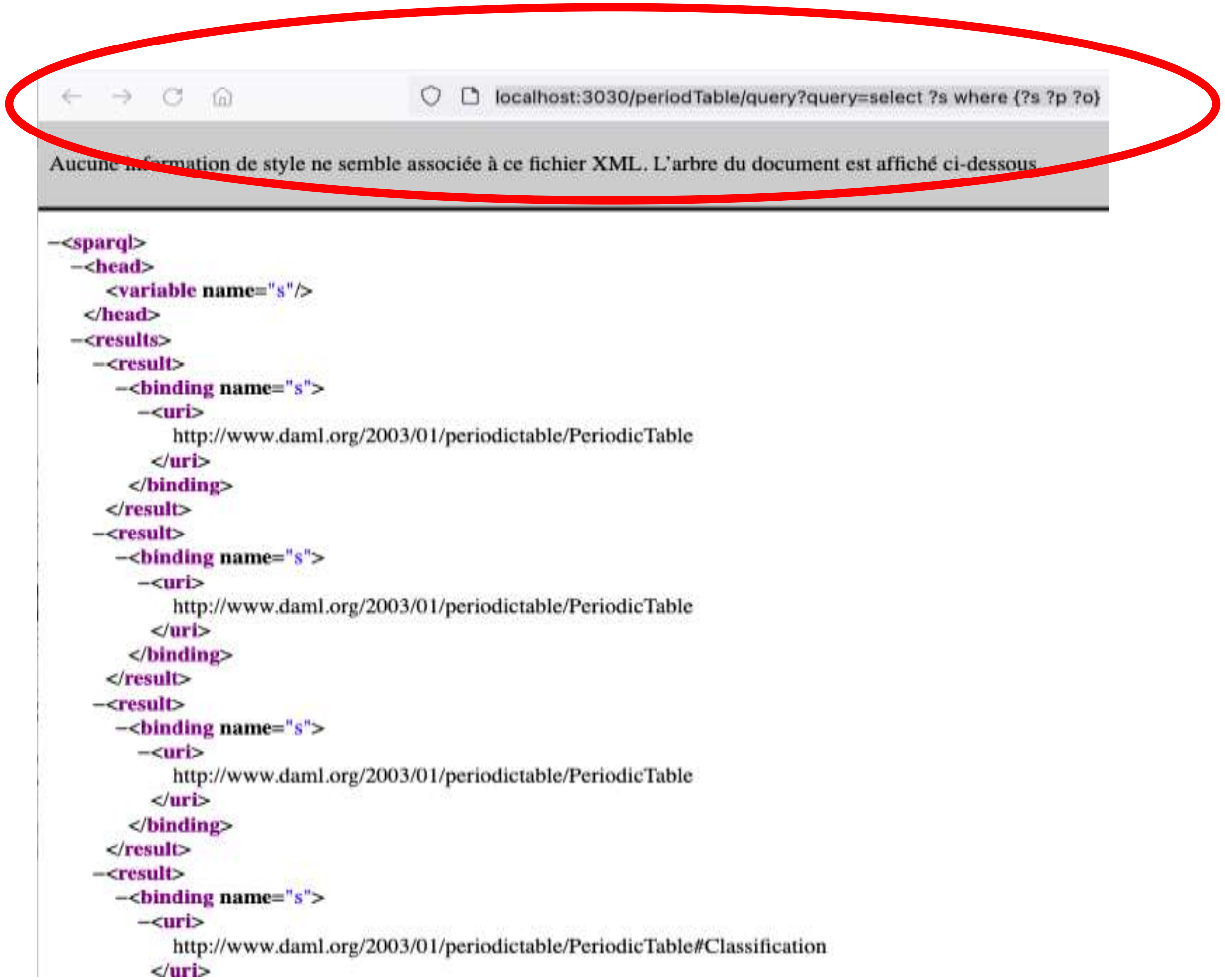


Dataset: /periodTable

Available services

File Upload:	/periodTable
File Upload:	/periodTable/upload
Graph Store Protocol:	/periodTable
Graph Store Protocol:	/periodTable/data
Graph Store Protocol (Read):	/periodTable
Graph Store Protocol (Read):	/periodTable/get
SPARQL Query:	/periodTable/query
SPARQL Query:	/periodTable/sparql
SPARQL Query:	/periodTable
SPARQL Update:	/periodTable/update
SPARQL Update:	/periodTable

Query your SPARQL end point



localhost:3030/periodTable/query?query=select ?s where {?s ?p ?o}

Aucune information de style ne semble associée à ce fichier XML. L'arbre du document est affiché ci-dessous.

```
−<sparql>
  −<head>
      <variable name="s"/>
    </head>
  −<results>
    −<result>
      −<binding name="s">
        −<uri>
            http://www.daml.org/2003/01/periodictable/PeriodicTable
          </uri>
        </binding>
      </result>
    −<result>
      −<binding name="s">
        −<uri>
            http://www.daml.org/2003/01/periodictable/PeriodicTable
          </uri>
        </binding>
      </result>
    −<result>
      −<binding name="s">
        −<uri>
            http://www.daml.org/2003/01/periodictable/PeriodicTable
          </uri>
        </binding>
      </result>
    −<result>
      −<binding name="s">
        −<uri>
            http://www.daml.org/2003/01/periodictable/PeriodicTable#Classification
          </uri>
```

SPARQL Queries

Four types of queries depending on the data they return

Query	Return/Output	Description
SELECT	variables	To retrieve specif
CONSTRUCT	RDF graph	Builds new triples The template is de bound by the WH *information from* *different structure.*
DESCRIBE	RDF graph	Retrieves **ALL** informatio *comprehensive view of a* *entities in the RDF graph.*
ASK	Boolean	Returns ***true*** if the pattern ma ed.

```
SELECT ?name ?age
WHERE {
```

```
CONSTRUCT {
        ?person foaf:name ?name .
}
WHERE {
        ?person foaf:name ?name .
}
```

```
DESCRIBE
<http://example.org/person/123>
```

```
ASK
WHERE {
        ?person foaf:age 30 .
}
```

SPARQL query structure

PREFIX foaf: <http://xmlns.com/foaf/0.1/>

```
PREFIX …
…
PREFIX …
```

Each prefix defines a namespace used in the query

```
QUERY_TYPE …
```

The query type (SELECT, CONSTRUCT…) and the variables

```
FROM …
```

Which graph(s) to query (possibly union of graphs)

```
WHERE {
…
}
```

Patterns to match written in turtle syntax (with FILTER, ORDER…)

Feature

- Negation
- Transitive C
- Aggregatio
- Subqueries

```
SELECT ?
e .
e ?age .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?person2
WHERE {
        ?person1 foaf:knows+ ?person2 .
}
```

Note: https://
query/#propertypath

```
SELECT ?group (COUNT(?person) AS ?count)
WHERE {
```

```
SELECT ?person
WHERE {
        ?person foaf:name ?name .
        ?person foaf:age ?age .
        FILTER (?age > (
                SELECT (AVG(?age) AS ?avg_age)
        WHERE {
                        ?person foaf:age ?age .
        }
        ))
}
```

Features in SPARQL 1.1 (2013)

- Negation:

```
SELECT ?person
WHERE {
        ?person foaf:name ?name .
        FILTER NOT EXISTS {
                ?person foaf:age ?age .
        }
}
```

Features in SPARQL 1.1 (2013)

- <u>Transitive Queries:</u>

```
PREFIX foaf: <http://xmlns.com/foa

SELECT ?person2
WHERE {
        ?person1 foaf:knows+ ?person2 .
}
```

“one or more”

<u>Note: https://www.w3.org/TR/sparql11-query/#propertypaths</u>

- Alice knows Bob
- Bob knows Charlie
- Charlie knows Dana

```
SELECT ?connection WHERE {
:Alice foaf:knows+ ?connection .
}
```

- Bob
- Charlie
- Dana

Features in SPARQL 1.1 (2013)

- Aggregation and Grouping

```
SELECT ?group (COUNT(?person) AS ?count)
WHERE {
        ?person foaf:member ?group .
}
GROUP BY ?group
```

Alice is a member of Chess Club
Bob is a member of Java Club
Charlie is a member of Java Club

Features in SPARQL 1.1 (2013)

- Subqueries

```
SELECT ?person
WHERE {
        ?person foaf:name ?name .
        ?person foaf:age ?age .
        FILTER (?age > ( SELECT (AVG(?age) AS ?avg_age)   WHERE {
                                ?person foaf:age ?age .    } ))
}
```

Basic graph pattern in SPARQL

- Most general graph pattern is a triple ending with a full stop

  <subject> <predicate> <object> .

- S,P,O can be:
  - Variable (starts with ? Or $):

    ?e prt:symbol “Au” .

  - Full IRI:

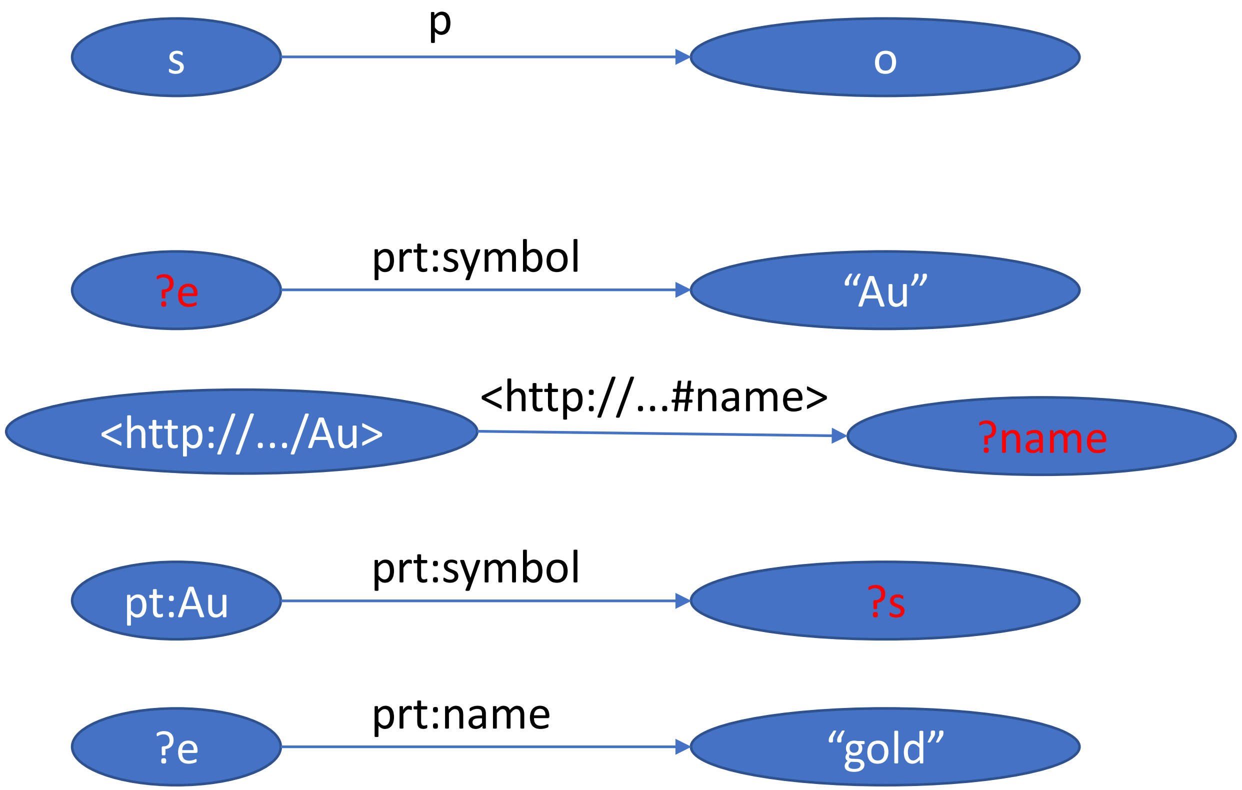
    `<http://.../Au> <http://...#name> ?name .`

  - Qname (qualified name) style:

    `pt:Au` prt:symbol **?s .**

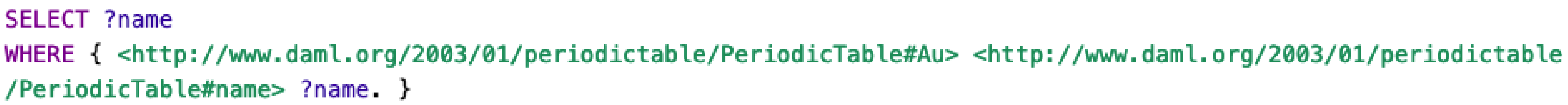
  - Literal string:

    $e prt:name “gold”^^xsd:string .

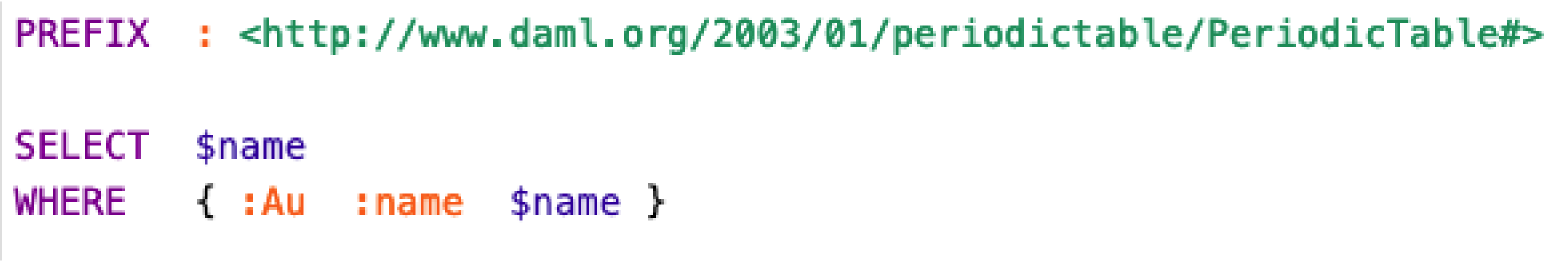


Example 1: these 4 queries are identical

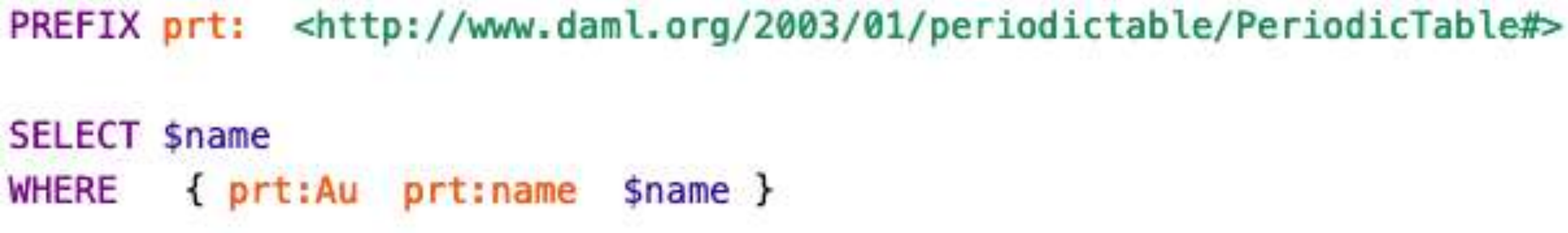
Selecting the name of element whose IRI is <Au>



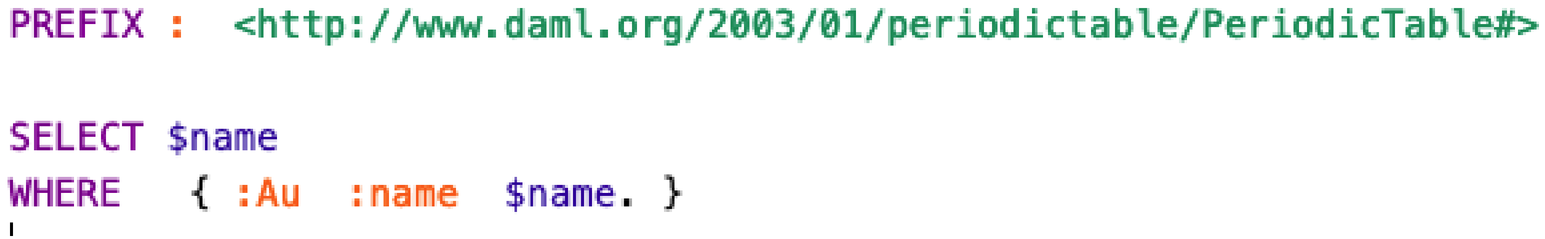
```
SELECT ?name
WHERE { <http://www.daml.org/2003/01/periodictable/PeriodicTable#Au> <http://www.daml.org/2003/01/periodictable
/PeriodicTable#name> ?name. }
```



```
PREFIX  : <http://www.daml.org/2003/01/periodictable/PeriodicTable#>

SELECT  $name
WHERE   { :Au  :name  $name }
```



```
PREFIX prt:  <http://www.daml.org/2003/01/periodictable/PeriodicTable#>

SELECT $name
WHERE   { prt:Au  prt:name  $name }
```

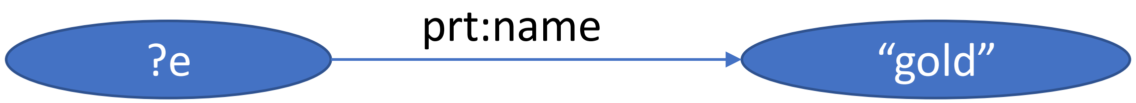


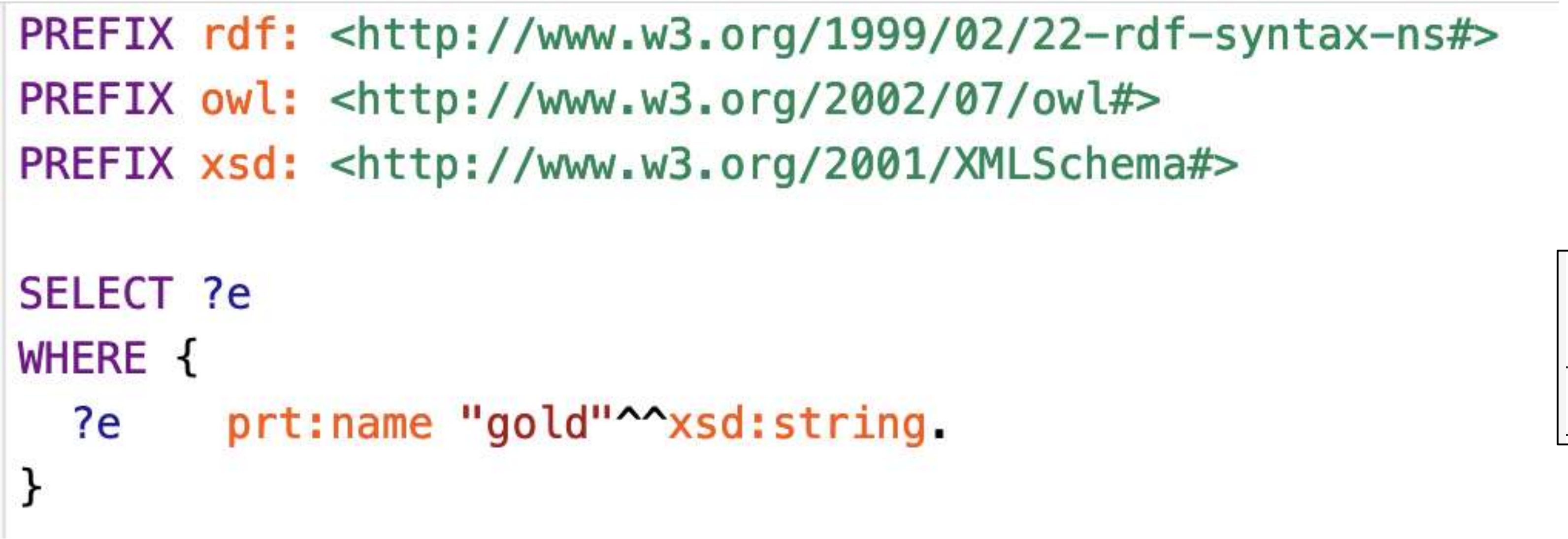
```
PREFIX :  <http://www.daml.org/2003/01/periodictable/PeriodicTable#>

SELECT $name
WHERE   { :Au  :name  $name. }
```

But 2nd and 4th look identical?!

Example 2: Simple pattern

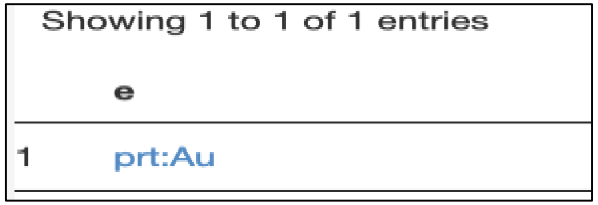
Select the IRI of the element named “gold”





```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?e
WHERE {
  ?e    prt:name "gold"^^xsd:string.
}
```

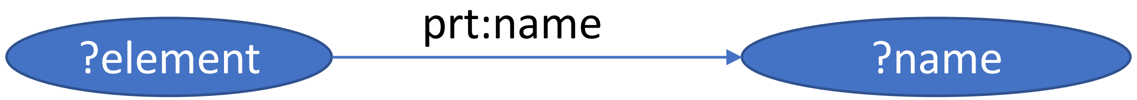
Result



Showing 1 to 1 of 1 entries

	e
1	prt:Au

Example 3: simple pattern



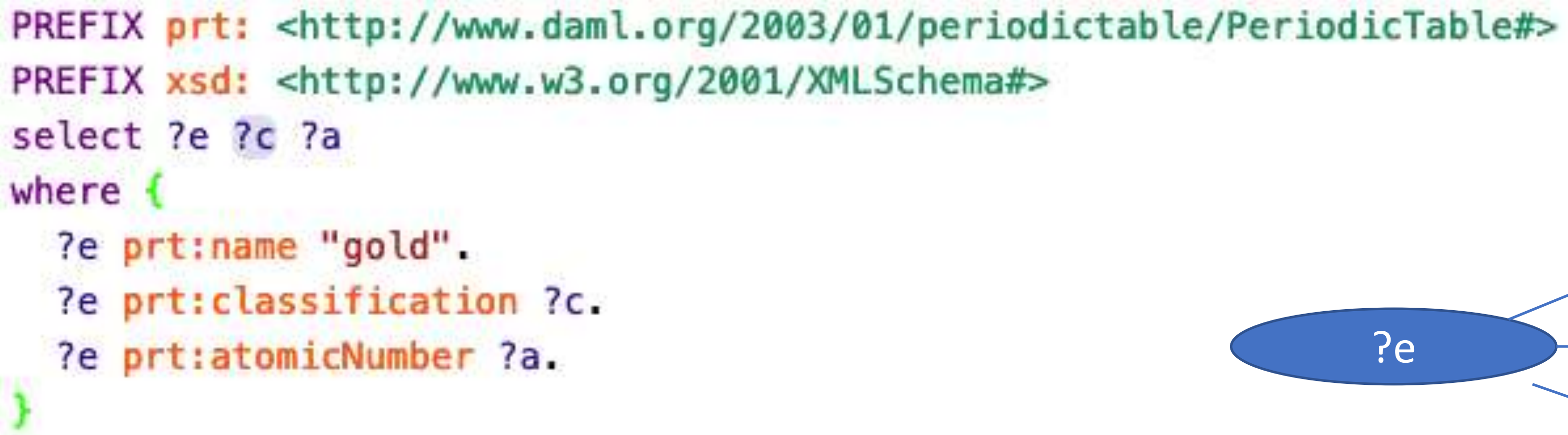
- Select all element names

```
PREFIX prt: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?name
WHERE { ?element prt:name ?name. }
```

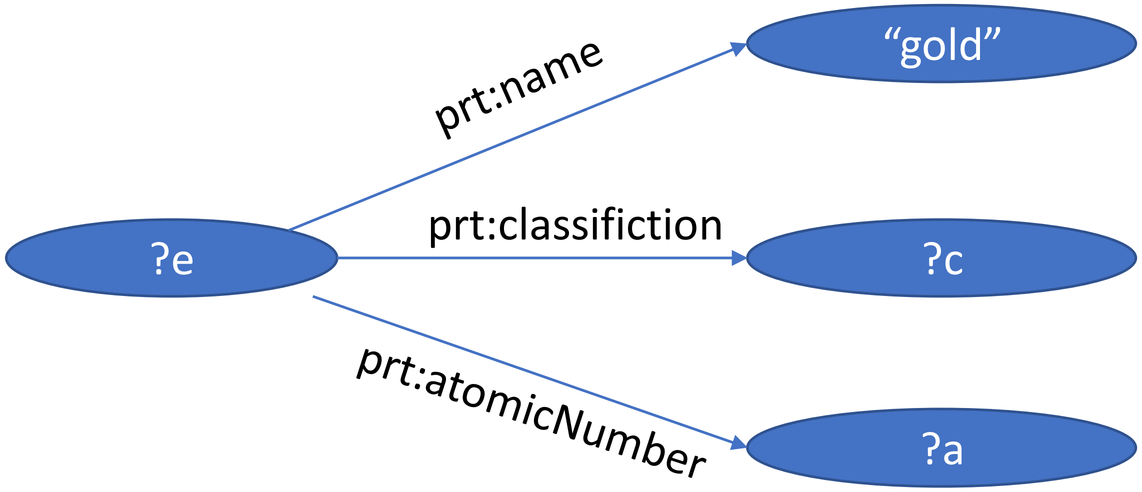
- Only the names are displayed

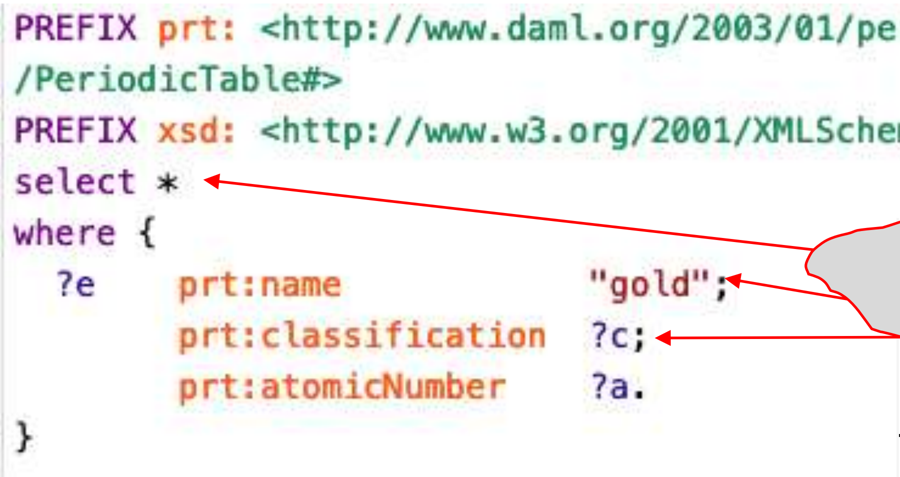
More general graph pattern

Select the *IRI, classification* and *atomic number* of the element named “gold”



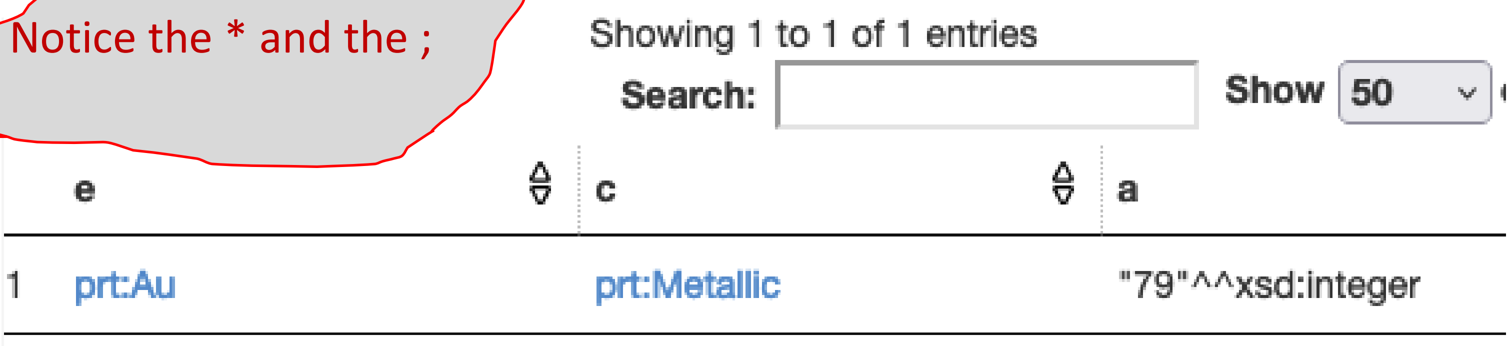
```
PREFIX prt: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
select ?e ?c ?a
where {
  ?e prt:name "gold".
  ?e prt:classification ?c.
  ?e prt:atomicNumber ?a.
}
```





```
PREFIX prt: <http://www.daml.org/2003/01/periodictable
/PeriodicTable#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
select *
where {
  ?e     prt:name             "gold";
         prt:classification   ?c;
         prt:atomicNumber     ?a.
}
```

Notice the * and the ;



Showing 1 to 1 of 1 entries

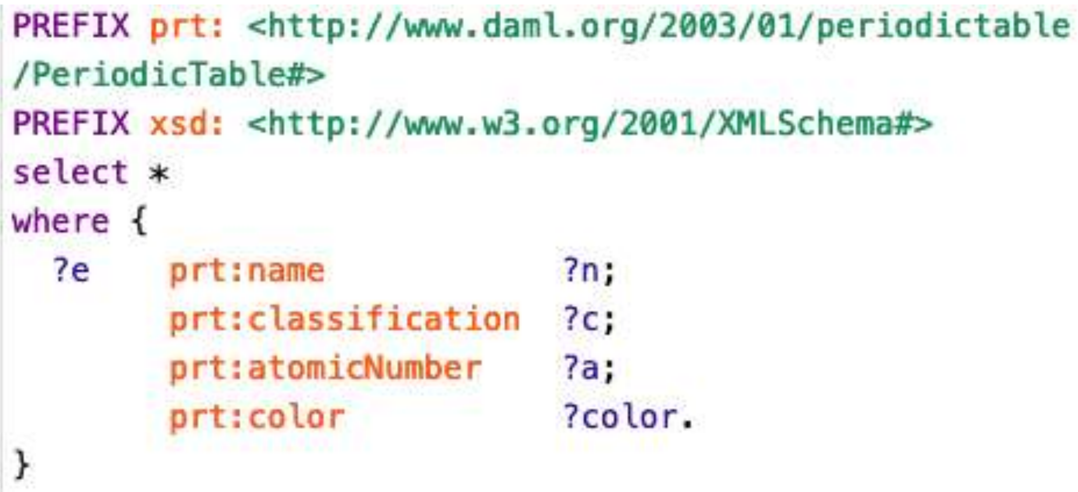
Search:

Show 50

	e	c	a
1	prt:Au	prt:Metallic	"79"^^xsd:integer

SPARQL OPTIONAL

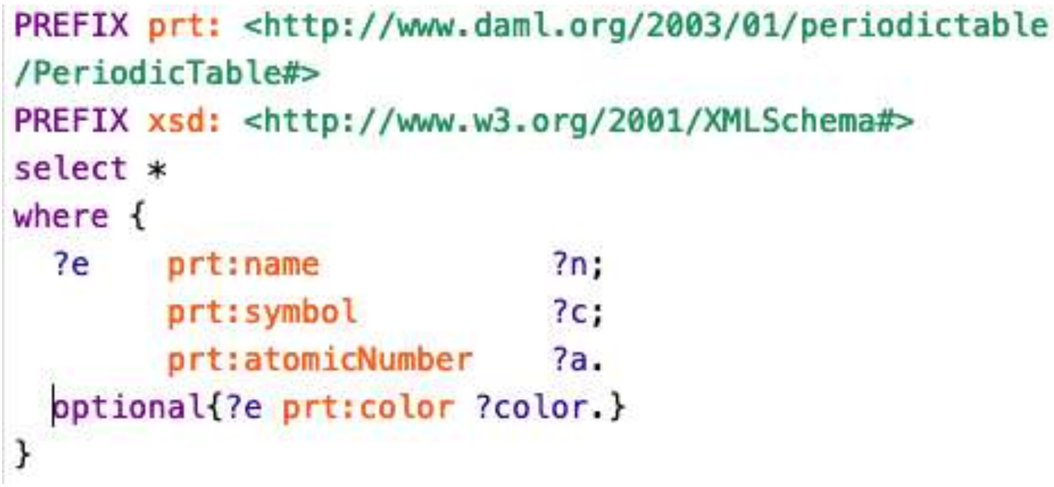
Select all elements that have a colour.



```
PREFIX prt: <http://www.daml.org/2003/01/periodictable
/PeriodicTable#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
select *
where {
  ?e    prt:name             ?n;
        prt:classification   ?c;
        prt:atomicNumber     ?a;
        prt:color            ?color.
}
```

Elements that have no colour are not matched:
Elements 113, 115, 117 are not retrieved
as they do not have colour…

Select all elements and their colours if they have one.



```
PREFIX prt: <http://www.daml.org/2003/01/periodictable
/PeriodicTable#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
select *
where {
  ?e    prt:name             ?n;
        prt:symbol           ?c;
        prt:atomicNumber     ?a.
  optional{?e prt:color ?color.}
}
```

Here all 118 elements are retrieved…
Matching the colour property has been made **optional**

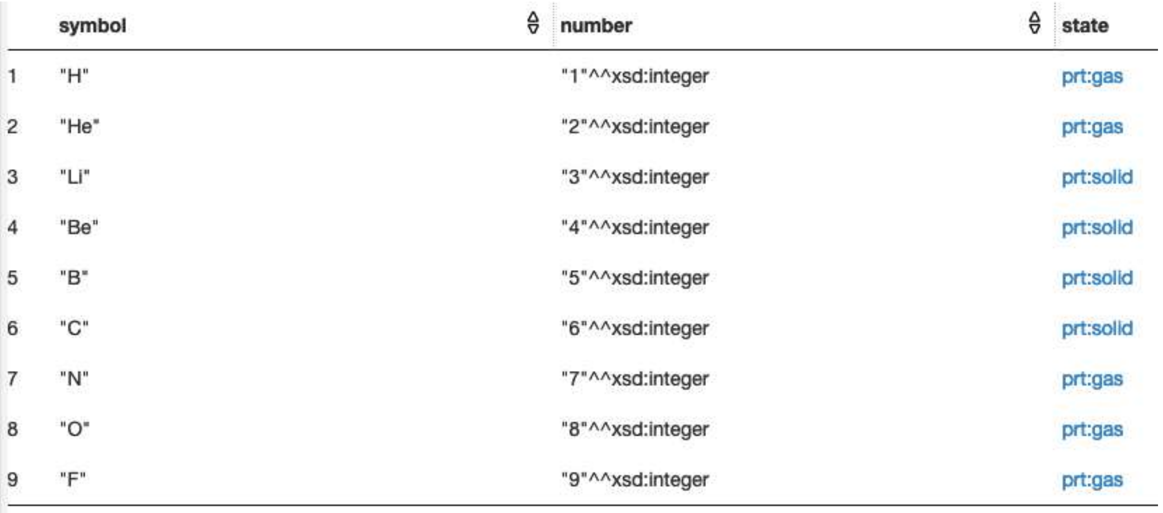
FILTER

- FILTER allows ***restricting the results*** by applying conditions.
- Conditions are Boolean expressions.
- Operators like >, <, >=, <= , &&, || can be used
- Can apply to:
    - Numbers:
    - Dates: same operators
    - Strings: regular expressions

```
^(?:\+60)?(\d{10,11})$
```

Filtering Numbers

- FILTER elements that have an atomic number lower than 10



	symbol	number	state
1	"H"	"1"^^xsd:integer	prt:gas
2	"He"	"2"^^xsd:integer	prt:gas
3	"Li"	"3"^^xsd:integer	prt:solid
4	"Be"	"4"^^xsd:integer	prt:solid
5	"B"	"5"^^xsd:integer	prt:solid
6	"C"	"6"^^xsd:integer	prt:solid
7	"N"	"7"^^xsd:integer	prt:gas
8	"O"	"8"^^xsd:integer	prt:gas
9	"F"	"9"^^xsd:integer	prt:gas

```
PREFIX prt: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?symbol ?number ?state
WHERE{
   ?element prt:symbol ?symbol;
   prt:atomicNumber ?number;
   prt:standardState ?state .
   FILTER(?number < 10)
}
```

Filtering Numbers

- FILTER individual persons that are born between 1962 and 2020.

```
PREFIX pep: <http://hw.ac.uk/#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT *
WHERE{
   ?sub pep:bornOn ?date .
   FILTER(?date < “2020” &&  ?date > “1962”)
}
```

Filtering Strings

- Regular expressions can be used to filter based on string values
- SPARQL uses Xpath regular expressions

https://www.w3.org/TR/xpath-functions/#regex-syntax

- Example:
  - ^ start of the string
  - $ end of the string
  - i case insensitive

Find elements where symbol starts with n or N

```
PREFIX prt:
<http://www.daml.org/2003/01/periodictable
/PeriodicTable#>
SELECT ?symbol ?number ?state
WHERE{
    ?element prt:symbol ?symbol;
    prt:atomicNumber ?number;
    prt:standardState ?state .
    FILTER REGEX(?symbol, '^n', 'i’)
}
```

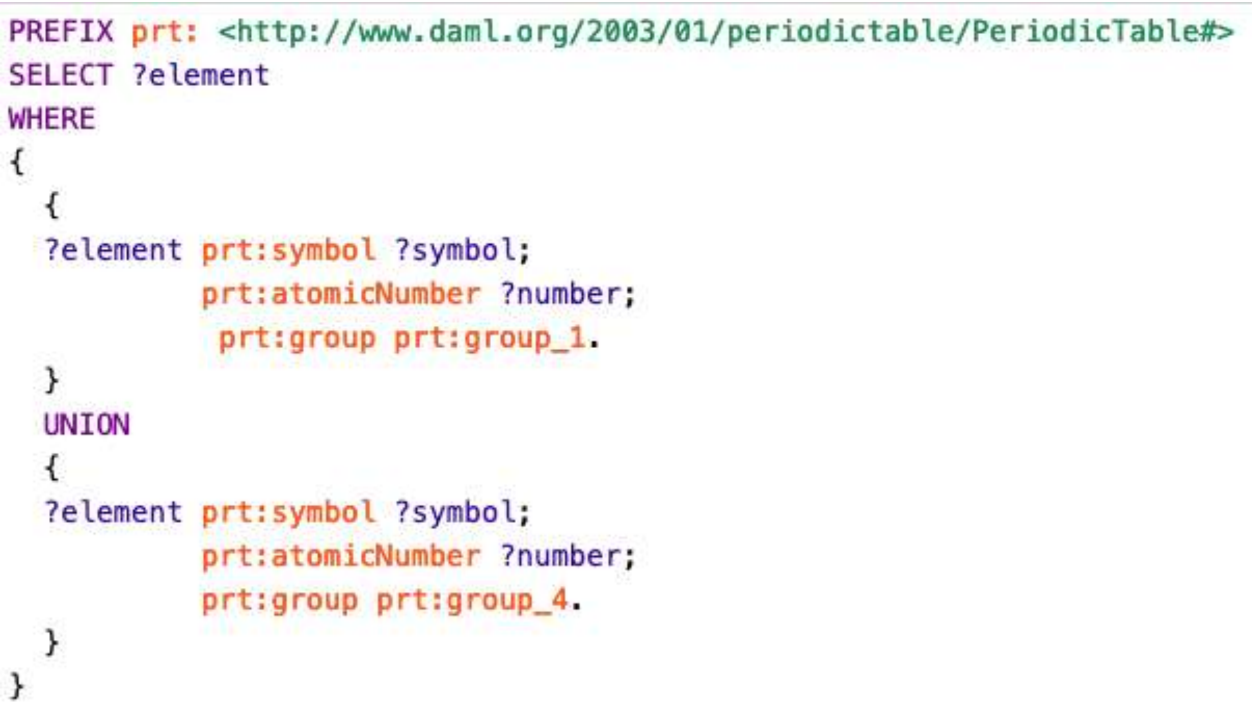
Filtering with existing or non-existing properties

```
PREFIX prt:
<http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT *
WHERE
{
            ?element prt:name ?name ;
                        prt:symbol ?symbol ;
                        prt:atomicNumber ?number .
            MINUS { ?element prt:color ?color . }
}
```

```
PREFIX prt:
<http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT *
WHERE
{
  ?element prt:name ?name;
   prt:symbol ?symbol;
   prt:atomicNumber ?number.
  FILTER NOT EXISTS { ?element prt:color ?color .}
}
```

SPARQL Union

- Union operator allows selecting with a conjunction of patterns.
- The resulting set is the union of the triples that match the first pattern and those that match the second.
- Any number of unions can be used.



```
PREFIX prt: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?element
WHERE
{
  {
  ?element prt:symbol ?symbol;
           prt:atomicNumber ?number;
            prt:group prt:group_1.
  }
  UNION
  {
  ?element prt:symbol ?symbol;
           prt:atomicNumber ?number;
           prt:group prt:group_4.
  }
}
```

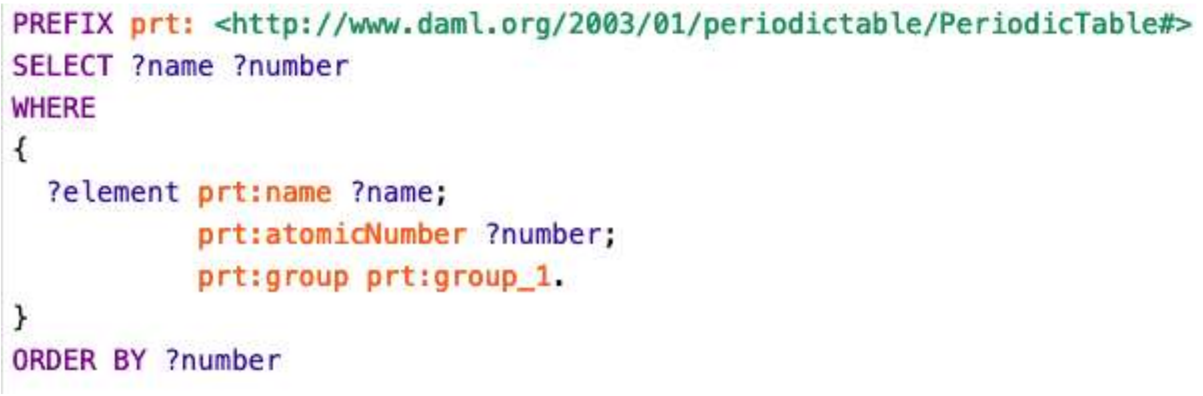
Query modifier

- SPARQL offers multiple query modifiers:
    - ORDER BY
    - LIMIT
    - OFFSET
    - GROUP BY
    - HAVING

SPARQL ORDER BY

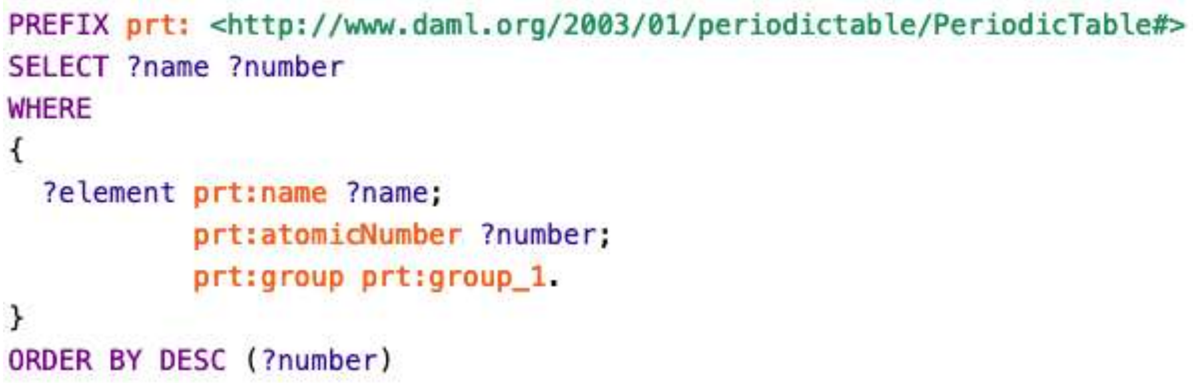
- Order results according to one variable.
- The ordering variable must be in the select clause.

Ascending order (by default)



```
PREFIX prt: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?name ?number
WHERE
{
  ?element prt:name ?name;
           prt:atomicNumber ?number;
           prt:group prt:group_1.
}
ORDER BY ?number
```

Descending order

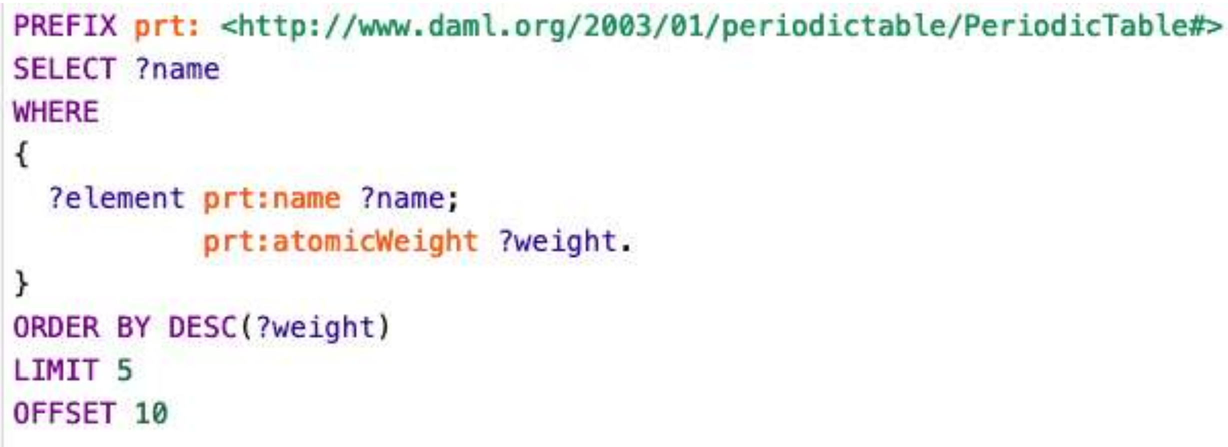


```
PREFIX prt: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?name ?number
WHERE
{
  ?element prt:name ?name;
           prt:atomicNumber ?number;
           prt:group prt:group_1.
}
ORDER BY DESC (?number)
```

SPARQL LIMIT and OFFSET

- We can limit the number of matches that are displayed using LIMIT.
- We can also start the display after a given number of matches (that will be left off) using OFFSET.
  - Offset 10 = skip 10; show 11th

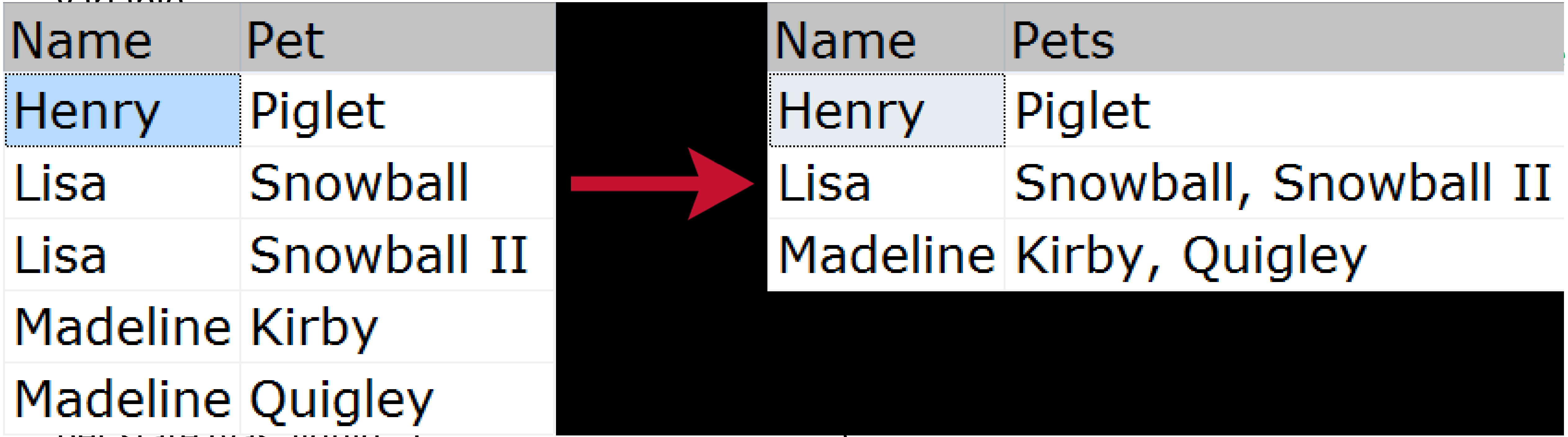
Limit to 5 matches and start from the 10th



```
PREFIX prt: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?name
WHERE
{
  ?element prt:name ?name;
           prt:atomicWeight ?weight.
}
ORDER BY DESC(?weight)
LIMIT 5
OFFSET 10
```

GROUP BY

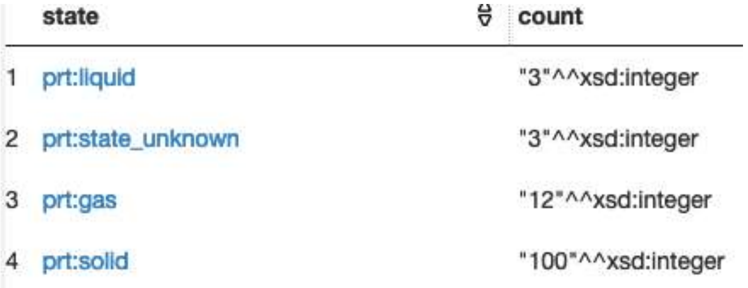
- Creates ***groups*** based on the value of a variable



Name	Pet
Henry	Piglet
Lisa	Snowball
Lisa	Snowball II
Madeline	Kirby
Madeline	Quigley

Name	Pets
Henry	Piglet
Lisa	Snowball, Snowball II
Madeline	Kirby, Quigley

per state (gas, liquid…)



	state	count
1	prt:liquid	"3"^^xsd:integer
2	prt:state_unknown	"3"^^xsd:integer
3	prt:gas	"12"^^xsd:integer
4	prt:solid	"100"^^xsd:integer

}

GROUP BY ?state
ORDER BY ?num

HAVING

- HAVING applies a condition ***after*** the groups have been created.
  - with **GROUP BY**
- Only groups that satisfy the condition are returned.
- Example: select only states that have a number of elements greater than 10.

```
PREFIX prt:
<http://www.daml.org/2003/01/periodictable/
PeriodicTable#>
SELECT ?state (COUNT(??symbol) as ?num
WHERE
{
 ?element prt:name ?symbol;
  prt:standardState ?state;
  prt:atomicNumber ?number.
 }
GROUP BY ?state
HAVING (?num > 10)
ORDER BY ?num
```

C

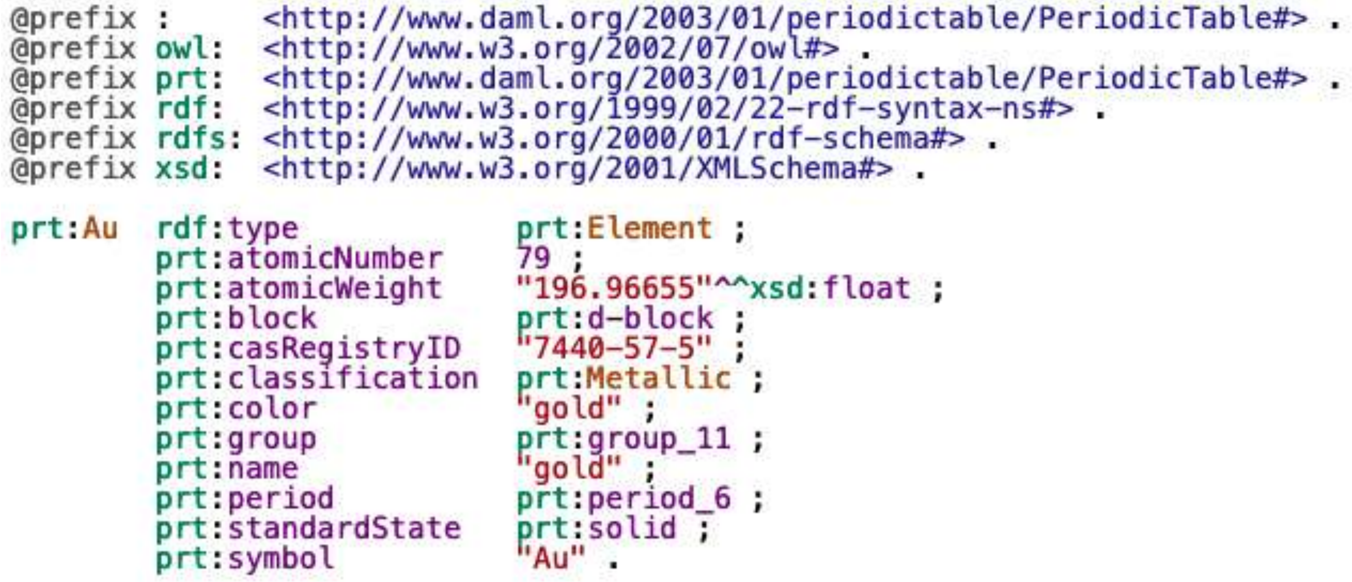
- CO pre ma e/ f-
- ***Ou***
- Exa
  -
- ***Ple an***
- The a gra

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix prt: <http://www.daml.org/2003/01/periodictable/PeriodicTable#> .

# Hydrogen (No color in data, so no rdfs:comment)
 prt:H rdfs:label "Hydrogen" .

# Helium (No color in data)
prt:He rdfs:label "Helium" .

# Chlorine (Has a color, so rdfs:comment is included)
prt:Cl rdfs:label "Chlorine" ;
        rdfs:comment "greenish yellow" .

# Bromine prt:Br rdfs:label "Bromine" ;
        rdfs:comment "reddish-brown" .
```

DESCRIBE

- DESCRIBE allows retrieving ***all properties*** of an entity given its IRI.
- It ***returns RDF triples***
- Example, describe the element Au

```
PREFIX prt: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>

DESCRIBE prt:Au
```



```
@prefix :     <http://www.daml.org/2003/01/periodictable/PeriodicTable#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix prt:  <http://www.daml.org/2003/01/periodictable/PeriodicTable#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .

prt:Au  rdf:type            prt:Element ;
        prt:atomicNumber    79 ;
        prt:atomicWeight    "196.96655"^^xsd:float ;
        prt:block           prt:d-block ;
        prt:casRegistryID   "7440-57-5" ;
        prt:classification  prt:Metallic ;
        prt:color           "gold" ;
        prt:group           prt:group_11 ;
        prt:name            "gold" ;
        prt:period          prt:period_6 ;
        prt:standardState   prt:solid ;
        prt:symbol          "Au" .
```

DESCRIBE - continued

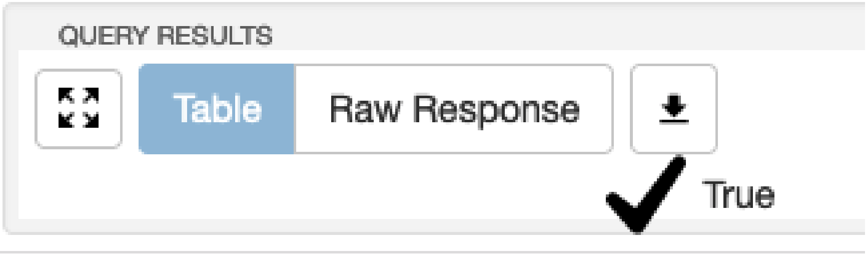
- DESCRIBE can also take a WHERE clause to select entities that match some patterns then return their properties as a list of triples

```
PREFIX prt:
<http://www.daml.org/2003/01/periodictable/PeriodicTable#>
DESCRIBE ?e
WHERE {
  ?e a prt:Element
}
```

ASK

- ASK checks whether a ***pattern*** matches the data.
- It ***returns true if some matches have been found.***
- And ***false*** if none have been found.
- In other words, it just checks that matches exist or not.

```
PREFIX prt:
<http://www.daml.org/2003/01/periodictable/PeriodicTable#>
ASK {
 ?e a prt:Element;
     prt:symbol ?s.
 FILTER REGEX (?s, 'A', 'i')
}
```



✓ True

SERVICE

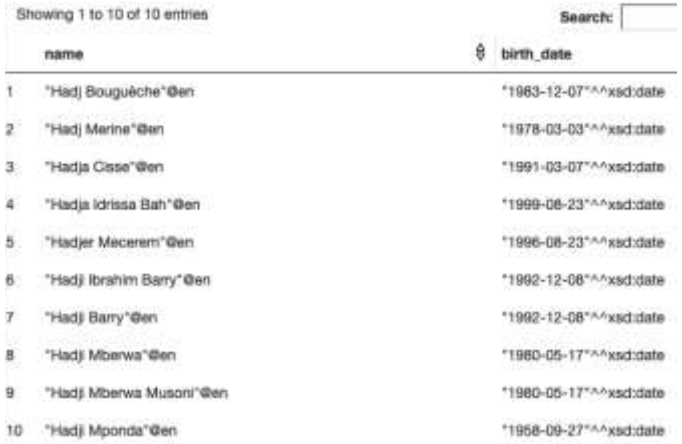
- SERVICE allows setting up a remote SPARQL endpoint for the query
- This ***allows the query to be sent to the endpoint***
- Example, querying DBPedia for people names and birthdate of people whose name starts with ‘Hadj’
- SERVICE is usually used to ***federate queries from different end points***

```
Select *
Where{
{  … put local here
}
Service …
{
}
}
```

- Multiple SERVICE clauses can be used

URL of the endpoint

```
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?birth_date
WHERE {
  SERVICE <http://dbpedia.org/sparql> {
   ?person a foaf:Person ;
   foaf:name ?name ;
   dbpedia:birthDate ?birth_date .
   FILTER REGEX ( ?name, ‘^Hadj', 'i')
 }
}
```



Showing 1 to 10 of 10 entries

Search:

	name	birth_date
1	"Hadj Bouguèche"@en	"1963-12-07"^^xsd:date
2	"Hadj Merine"@en	"1978-03-03"^^xsd:date
3	"Hadja Cisse"@en	"1991-03-07"^^xsd:date
4	"Hadja Idrissa Bah"@en	"1999-08-23"^^xsd:date
5	"Hadjer Mecerem"@en	"1996-08-23"^^xsd:date
6	"Hadji Ibrahim Barry"@en	"1992-12-08"^^xsd:date
7	"Hadji Barry"@en	"1992-12-08"^^xsd:date
8	"Hadji Mberwa"@en	"1980-05-17"^^xsd:date
9	"Hadji Mberwa Musoni"@en	"1980-05-17"^^xsd:date
10	"Hadji Mponda"@en	"1958-09-27"^^xsd:date

Reasoning within Triplestores

- Many triplestores include one or more reasoners.
- Some have native reasoners, such as GraphDB.
- Others use reasoners from frameworks like RDF4J or Jena.
- Types of Rules:
  - RDFS subClassOf/subPropertyOf
  - RDFS
  - OWL sameAs
  - OWL rules
    - *We want to infer that something is a "HealthyFood" if it is a "Fruit" or a "Vegetable".*
  - Custom rules
    - *give a DiscountedPrice to FrequentCustomers on weekdays.*

Forward and Backward Chaining

- To infer new knowledge from existing facts.
- Forward Chaining: working from facts to conclusions (data-driven approach)
  - Start from ***known facts***
  - Applies rules repeatedly (match facts to conditions of rules)
  - Until no rule can be applied
  - Adds all derives triples to the data

Reasoning is done **once**; may be slow if many facts.

- Backward Chaining: working from conclusions to facts (goal-driven approach)
  - Start from ***conclusions to be proven***
  - Repeat apply possible rules (match facts to conclusions of rules)
  - Creates new sub-goals when needed
  - Stops when a fact is found or no more rule to applies

Reasoning is done for each query. No upfront inferencing cost when first loading the data.

Example

Scenario

- **Rules**:
  - *Rule 1*: *If it is raining, then the ground is wet*.
  - *Rule 2*: *If the ground is wet, then the grass is wet*.
- **Fact**:
  - It is raining.

Forward Chaining

1. **Start with the fact**:
   - *It is raining*.
2. **Apply Rule 1**:
   - *Since it is raining, we can conclude that the ground is wet*.
3. **Apply Rule 2**:
   - *Since the ground is wet, we can conclude that the grass is wet*.

Backward Chaining

Assuming that we want to prove the conclusion of “***The grass is wet***”.

1. **Start with the goal**:
   - *The grass is wet*.
2. **Apply Rule 2**:
   - *To prove that “The grass is wet”, we need to prove that “The ground is wet”.*
3. **Apply Rule 1**:
   - *To prove that “The ground is wet”, we need to prove that “It is raining”*.
4. **Fact found**:
   - *It is raining.*