



# "Big-Data computing technologies" course

## Who am I?

Stéphane Derrode, [stephane.derrode@ec-lyon.fr](mailto:stephane.derrode@ec-lyon.fr)

## Where to find information?

<http://perso.ec-lyon.fr/derrode.stephane/Teaching/BSC/readme/>

(course organisation, access to chapter details, assessments...)

## Course organisation : 3 chapters

- Chapter 1. Big Data Technologies (25h, L. Derrode & S. Derrode)
- Chapter 2. NoSql with MongoDB (8h, E. Egyed-Zsigmond). *Already done!*
- Chapter 3. Cloud computing and AWS certifications (20h, Y. Fornier)

# Chapter 1 outline

- **Lecturers**

Lamia Derrode & Stéphane Derrode (Centrale Lyon).

- **Time allocation** 25h,

including 1h for mid-term exam and 2h for project restitution.

- **Organisation**

- Introduction to big data
- Part 1. Linked Open Data (LOD) technology (8h) and project (7h).
- Part 2. Hadoop framework and MRJob library(10h).

- **Assessment**

- Lab report (Part 2): 20% of the final grade.
- LOD project (Part 1): 40% of the final grade: 20% for the report and 20% for the project defense.
- Exam: Mid-term exam (Part 1 and Part 2) will account for 20% of the final grade. (February 18th, 2025)

- **Detailed content :** [here](#)

# CHAPTER 1

## INTRODUCTION TO BIG-DATA

---

# Short video introduction

Let's have a look at the data generated per minute on the internet

Platform	Data Generated per Minute
WhatsApp	2.1 Million
Google	3.8 Million
Facebook	1.0 Million
YouTube	4.5 Million
Total	188 Million

That's a lot of data

0:55 / 5:12  
©Simplilearn. All rights reserved. simplilearn

# Data deluge



- Emails:** 241.2 Million emails sent
- Texts:** 18.8 Million texts sent
- YouTube:** 694,000 video hours viewed
- Zoom:** 6.3 million Zoom meeting minutes
- Snapchat:** 3.47 million snaps created
- Instagram:** 10.4 million Instagram viewing minutes
- Slack:** 34,247 Slack messages sent
- Google:** 2.4 million searches
- Emoji:** 6.94 million emoji sent
- Photos Taken with Smartphones:** 3.02 million
- App Downloads:** 271,309 downloaded
- ChatGPT:** 22,831 visits

# Data deluge

A large waterfall cascading over a cliff, with a person standing on a walkway in the foreground. The waterfall is the central focus, with water falling in multiple stages. The surrounding area is lush with green trees and grass. In the background, there are some industrial structures and a misty atmosphere. A person is visible in the bottom left corner, looking towards the waterfall.

**Big data** refers to large, complex datasets that are too vast to be processed and analyzed using traditional data processing tools. These datasets often come from a variety of sources, such as social media, sensors, transactions, and other digital activities, and can be analyzed to uncover patterns, trends, and insights that drive decision-making.

**Big data** : A generation of technologies and architectures designed to extract economic or scientific value from a wide variety of data by enabling their high-speed capture, discovery, and/or analysis.

# Big data – 3V+ framework (Gartner firm, 2001)

Big Data can be defined using the **3Vs** rule, which represents the three key characteristics that describe the nature of large datasets:

- 1. Volume:** This refers to the massive amount of data generated every second, from various sources like social media, sensors, transaction records, and more. The volume of data is so large that traditional data management tools cannot handle it effectively.
- 2. Velocity:** This pertains to the speed at which data is generated, processed, and analyzed. In the era of Big Data, information flows at an unprecedented rate, and real-time or near-real-time analysis becomes crucial for timely decision-making.

# Big data – 3V+ framework (Gartner firm, 2001)

- 3. Variety:** This refers to the different types of data, including structured, semi-structured, and unstructured data. Data comes in various formats such as text, images, videos, logs, and sensor data, making it challenging to process and analyze without the proper tools.

Together, the **3Vs** —Volume, Velocity, and Variety— capture the complexity and scale of Big Data and highlight the challenges of managing, processing, and extracting meaningful insights from it.

- + **Veracity** which refers to the trustworthiness, accuracy, and quality of the data (uncertainty, data integrity, bias and error).
- + **Value** which refers to the usefulness and insights that can be extracted from large datasets to drive decision-making, innovation, and business outcomes.

# Stockage : Data-centers



*société QTS à  
Atlanta*



The storage of these massive data is done in data warehouses, or data centers, which contain tens of petabytes (PB). These true factories face industrial challenges (energy distribution, heat dissipation) and connectivity constraints, consuming 1% to 2% of the global electricity consumption (International Energy Agency (IEA) estimation).

# Processing : supercomputers



TESLA : supercomputers Dojo, Texas  
Copilot project

# CHAPTER 1

## PART 1 - LINKED OPEN DATA

---

1. Open Data
2. Linked Open Data and "semantic" WEB
3. Resource Description framework (RDF), with examples
4. SparQL (query language)
5. LOD practical work and LOD project explanations

# 1. Big data: public data versus personal data

**Personnal Data:** individual behavior, private data

- Traces from mobile phone or connection
- Internet browsing, social networks, messaging
- Purchases (credit cards), individual or collective movements (badges, cards)
- Individual consumption : water, electricity...
- IoT

**Public Data:** public services, general interest, open

- Staff: municipal employees, students
- Timetables: real-time bus and tram schedules
- Heritage: trees, sports facilities, streetlights, drinking fountains, etc.
- Operational elements: books borrowed from libraries, budget and financial reports, public procurement
- Description of the territory: aerial photos, altimetry, street descriptions
- Reflection of local life: marriages, births, names, elections, agendas, etc.

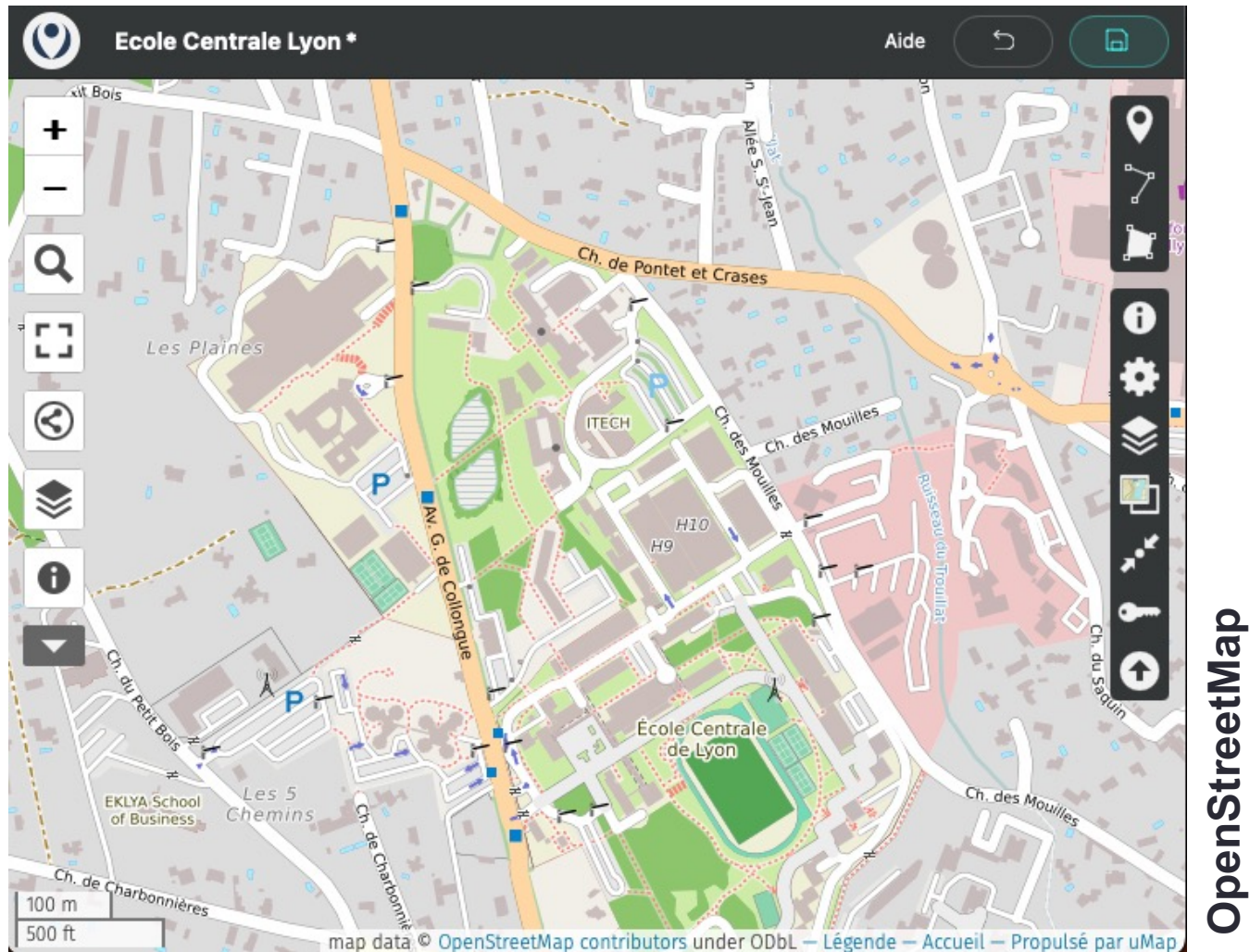
[Example: NYC open data](#)

# 1. Open Data

- 1. Public Accessibility:** Open data refers to data that is made available to the public, with no restrictions on access or use. Anyone can access, use, modify, and share the data.
- 2. Machine-readable Format:** It is typically provided in formats that can be easily processed by machines, such as CSV, JSON, or XML, to facilitate.
- 3. Free of Charge:** Open data is available for free, enabling a wide range of individuals, businesses, and organizations to leverage it without incurring costs.
- 4. Government and Institutional Data:** Many open data initiatives are led by governments, research institutions, or international organizations, with the goal of promoting transparency, innovation, and public engagement.  
[OpenDataBarometer](#), [Open Knowledge Foundation](#).
- 5. Encourages Innovation:** Open data can be used by developers, researchers, and businesses to create new services, applications, or insights, leading to economic, social, and technological advancements.

**Open Data** is publicly accessible, free of charge, machine-readable (big-) data that promotes transparency and innovation.

# 1. Open Data application: cartography



→ [Agence de Mobilité \(Lyon\)](#), [citymapper](#), [openWeatherMap](#)

# 1. Open Data: Worst parking spot in NY!



# 2. LOD : Five-star Open Data

According to **Tim Berners-Lee**, the inventor of the Web and Linked Data initiator.

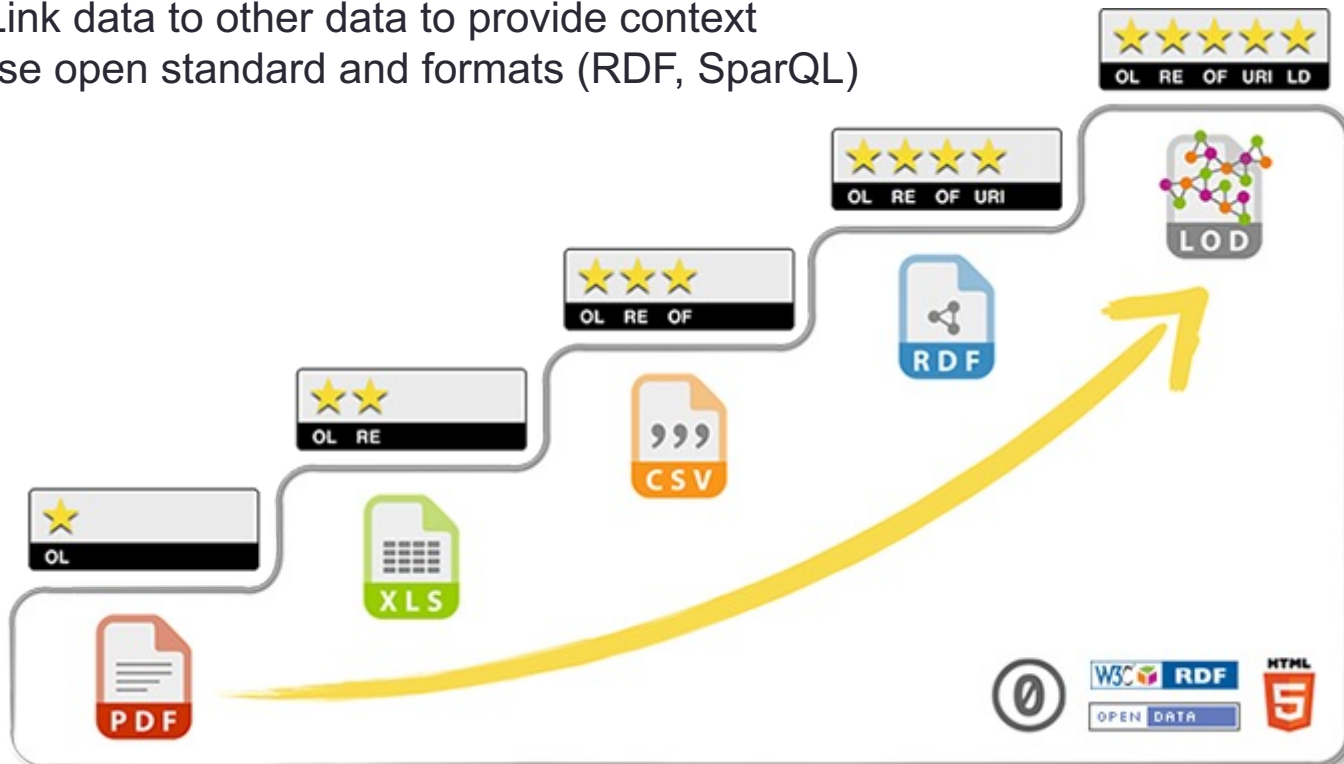
**OL:** Data available on the Web

**RE:** Structured data

**OF:** Data in open format

**URI:** Link data to other data to provide context

**LD:** Use open standard and formats (RDF, SparQL)



## 2. Open Data : Semantic Web

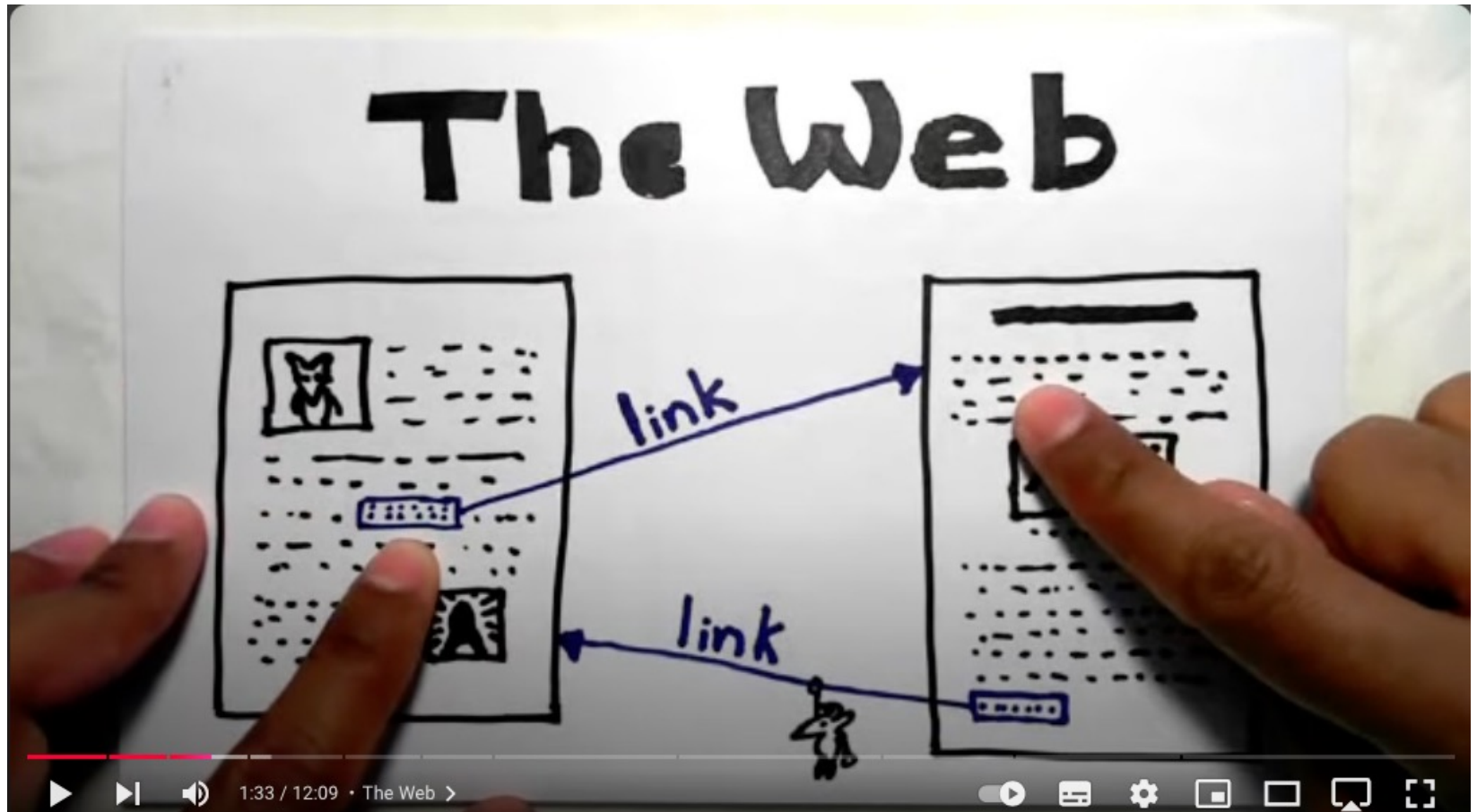
In the current web, most of the information is stored in **documents** (HTML pages) designed for human readers.

The **Semantic Web** refers to an extension of the current web that aims to make data on the internet **more machine-readable** and intelligible by adding meaning (or semantics) to the information.

One of the main principles of the Semantic Web is **linked data**, which involves connecting datasets across the web by using **URIs (Uniform Resource Identifiers)**. This means that data from different sources can be linked together, creating a **web of data**.



# 3. RDF - What is Linked Open Data?



# 3. RDF/LOD (1/3)

**RDF (Resource Description Framework)** is a standard developed by the **W3C** (World Wide Web Consortium) for representing structured information about resources on the web in a machine-readable way.

## 1. Keys features: Triple-based model

RDF uses a **triple** to represent data, where each triple consists of three parts:

- **Subject:** The resource being described (e.g., a person, an article).
- **Predicate:** The property or relationship between the subject and the object (e.g., "hasName", "isLocatedIn").
- **Object:** The value or resource related to the subject (e.g., "John Doe", "New York").

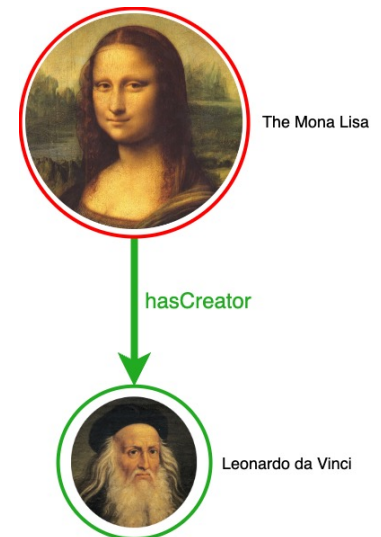
The RDF triple can be seen as a sentence :

**Subject** → **Predicate** → **Object**

**Example:**

"The Mona Lisa" → "hasCreator" → "Leonard Da Vinci"

This means " The Mona Lisa has creator Leonard Da Vinci ."



# 3. RDF/LOD (2/3)

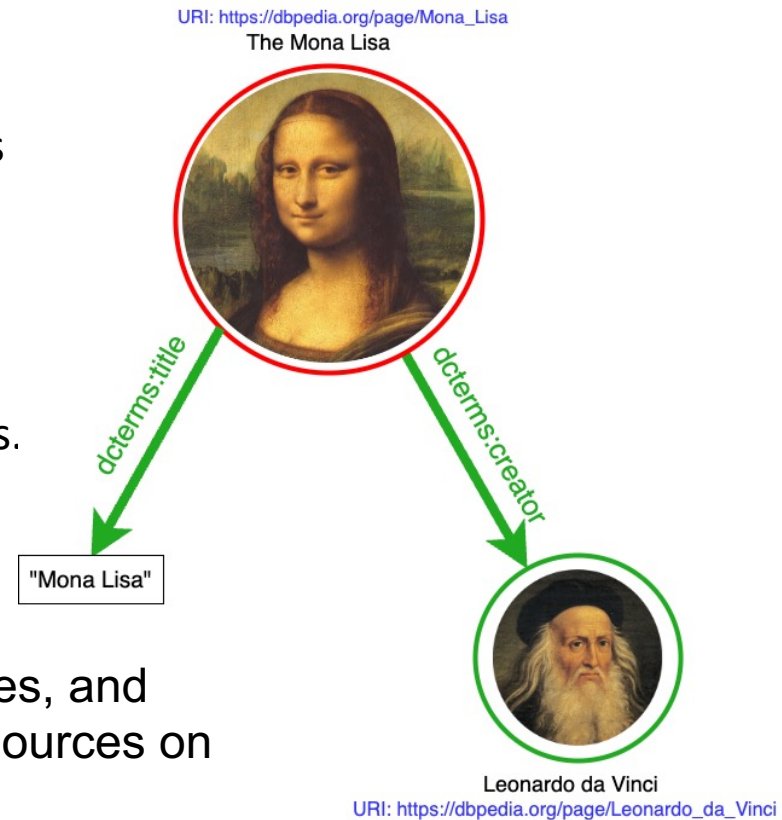
## 2. Keys features: Uniform Resource Identifier (URI)

In RDF, resources are typically identified by **URIs**, which provide a global, unambiguous identifier. This allows resources to be globally referenced across the web.

**Objects** can also be plain literals, such as numbers or strings, or can be identified by URIs.

**dcterms** is a vocabulary used in libraries, archives, and other institutions to describe a wide range of resources on the WEB (such as document, painting...).

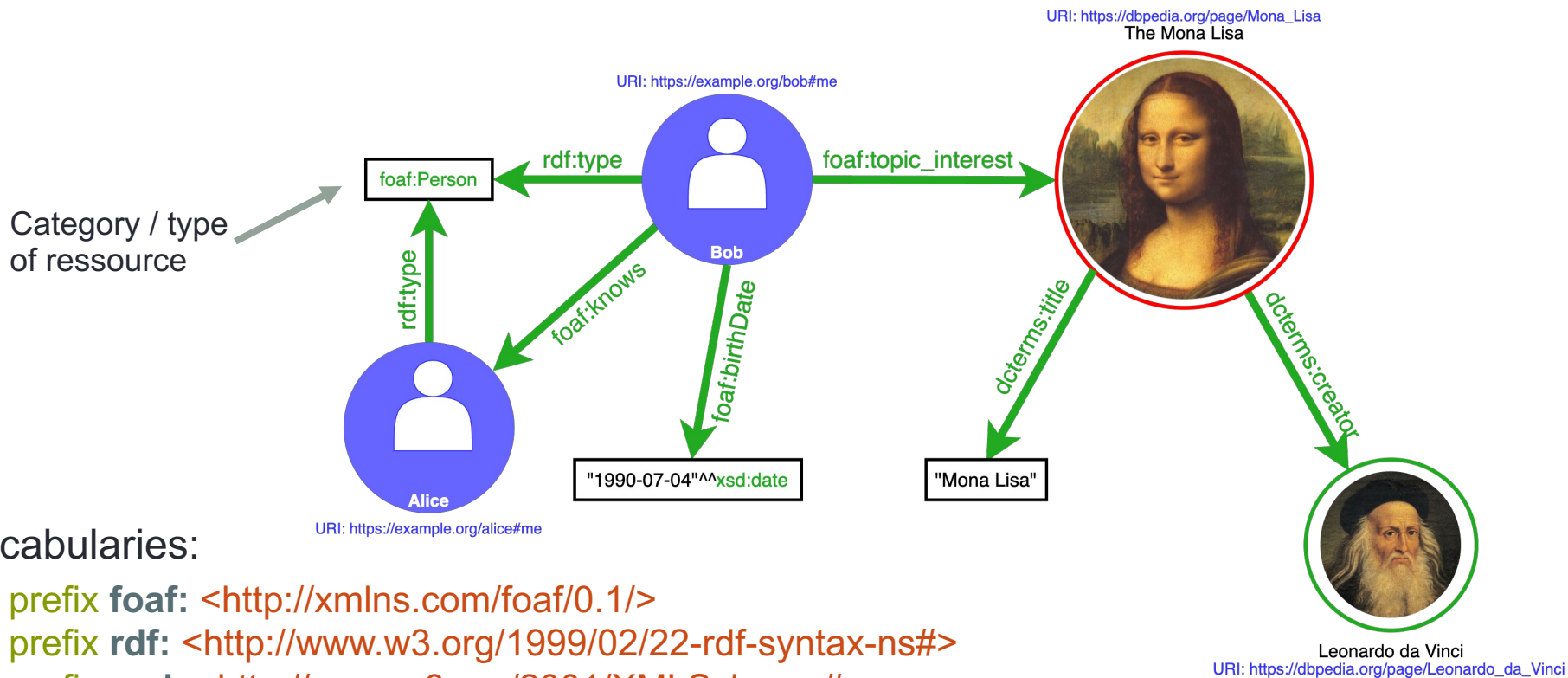
It is a prefix that points to this URI: <http://purl.org/dc/terms/>



# 3. RDF/LOD (3/3)

## 3. Keys features: Graph-based structure

RDF organizes data in a way that closely resembles a **graph** structure, where data is represented as **nodes** (entities) and **edges** (relationships). This graph-based model is powerful because it enables easy visualization, querying, and linking of data.



### Vocabularies:

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

prefix rdf: <<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>

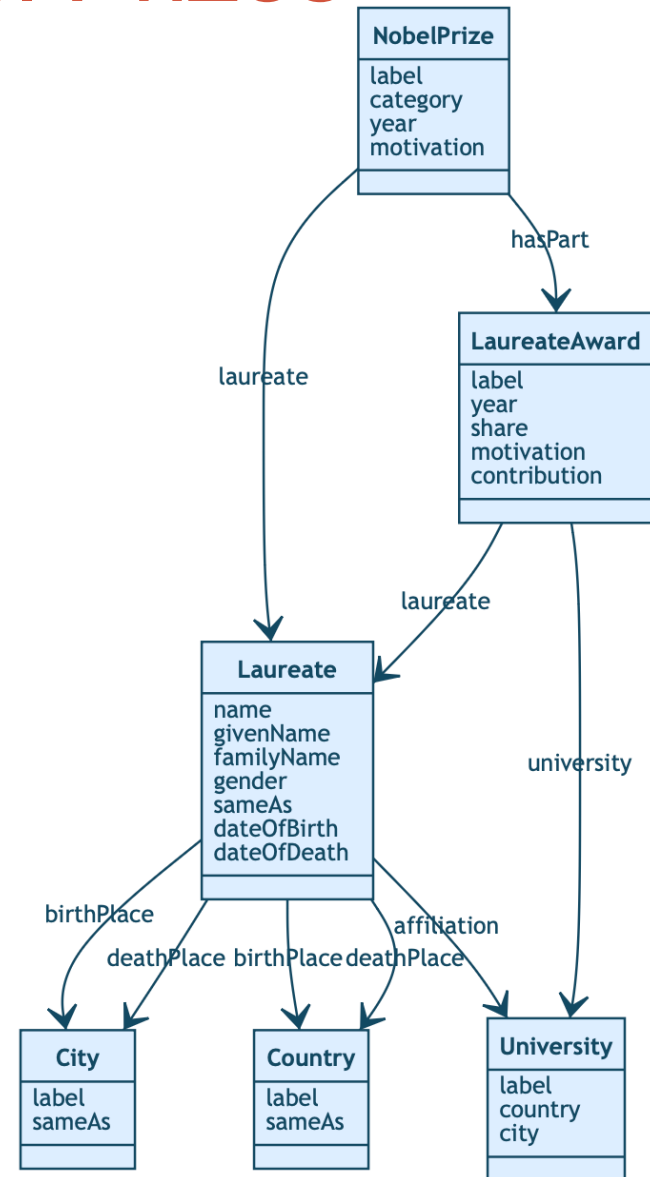
prefix xsd: <<http://www.w3.org/2001/XMLSchema#>>

prefix dcterms: <<http://purl.org/dc/terms/>>

# 3. LOD example: The Nobel Prizes

The **Nobel Prize Linked Data dataset** contains the information about Nobel Prizes and Laureates since 1901. This document specifies new classes and properties and reuse classes and properties from existing vocabularies.

An important characteristic of the Nobel Prizes are that they can be shared between up to three persons, in addition, the same person can receive multiple Nobel Prizes. To capture this the Nobel Prize expression in RDF makes use of the three classes NobelPrize, LaureateAward, and Laureate. Every NobelPrize contains between one and three LaureateAwards that among other things contains a motivation.



# 3. LOD example: The Nobel Prizes

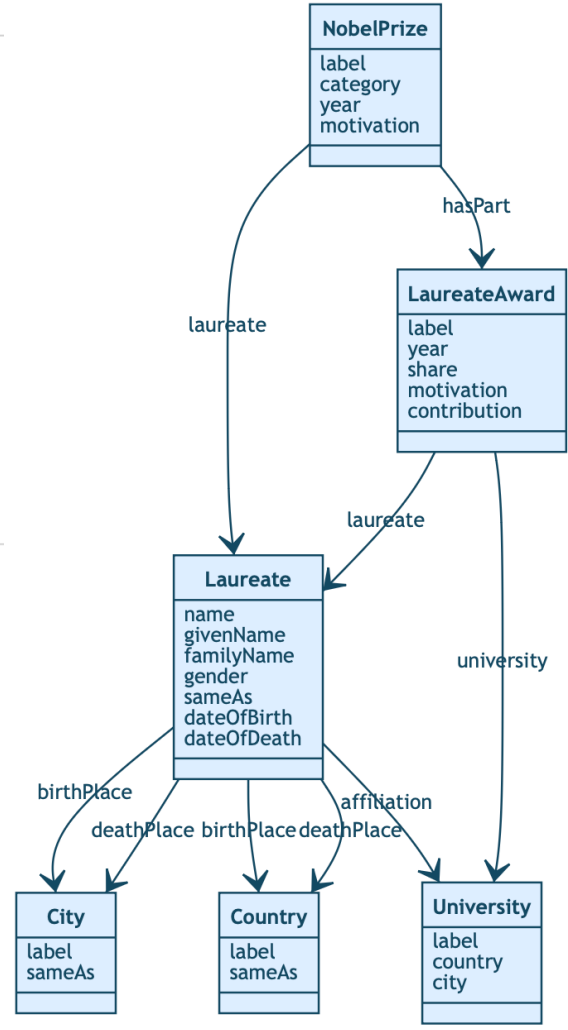
End point (web address to query the database) : <https://data.nobelprize.org/sparql>

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX nobel: <http://data.nobelprize.org/terms/>
4
5 SELECT DISTINCT ?name ?prizeName WHERE {
6   ?person rdf:type nobel:Laureate;
7   rdfs:label ?name;
8   nobel:nobelPrize ?prize.
9   ?prize rdfs:label ?prizeName;
10  nobel:year ?year.
11 FILTER (lang(?prizeName) = 'en' && ?year = 1901)
12 }
```

Table Response 6 results in 0.273 seconds

Simple view  Ellipse  Filter query results Page size: 50

name	prizeName
Wilhelm Conrad Röntgen	"The Nobel Prize in Physics 1901"@en
Sully Prudhomme	"The Nobel Prize in Literature 1901"@en
Jean Henry Dunant	"The Nobel Peace Prize 1901"@en
Frédéric Passy	"The Nobel Peace Prize 1901"@en
Emil Adolf von Behring	"The Nobel Prize in Physiology or Medicine 1901"@en
Jacobus Henricus van 't Hoff	"The Nobel Prize in Chemistry 1901"@en



# 3. LOD example: DBPedia

**DBpedia** is a **community-driven project** that extracts structured information from Wikipedia and makes it available on the web in RDF. It is one of the most prominent datasets in the **Linked Open Data (LOD)** cloud.

DBpedia's dataset is a **knowledge graph** that contains millions of **triples**. These triples describe entities and the relationships between them. E.g. :

*"Lyon" is a **city**  
with actual **mayor** "Grégory Doucet"  
and **population** of "522,250".*

Querying DBPedia with SparQL ([endpoint](#))

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

```
PREFIX dbo: <http://dbpedia.org/ontology/>
```

```
SELECT ?author
```

```
WHERE {
```

```
  ?book dbo:author ?author .
```


```
  ?book rdfs:label "The Catcher in the Rye"@en .
```

```
}
```

→  Result = J.D. Salinger

Wikipedia infobox for Lyon city



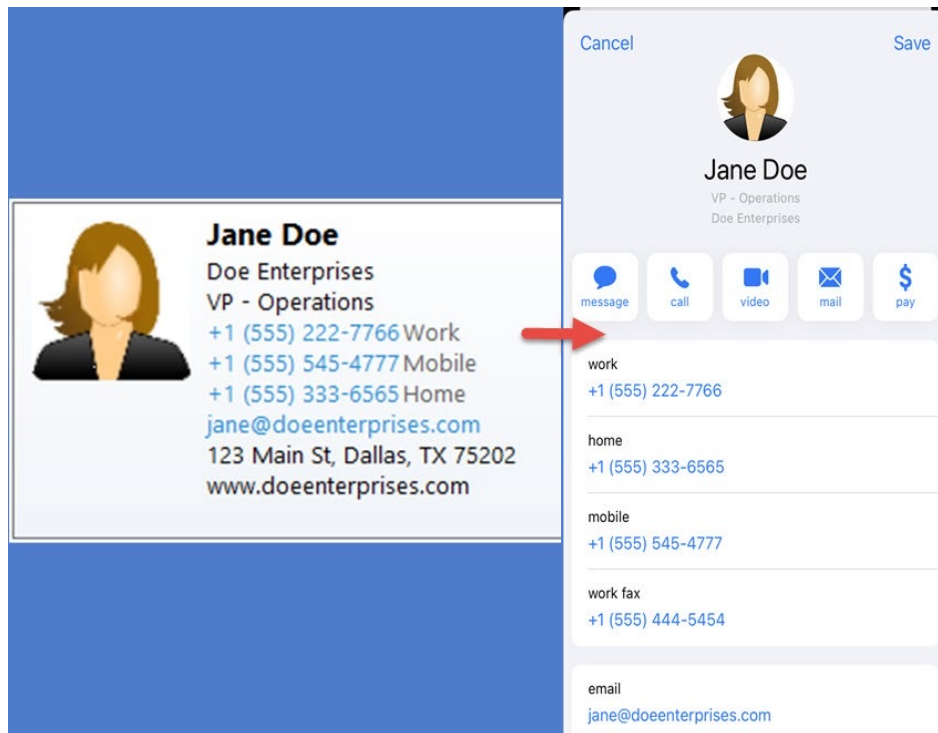
Show map of France  
 Show map of Auvergne-Rhône-Alpes  
 Show all  
Coordinates:  45°46'N 4°50'E

<b>Country</b>	France
<b>Region</b>	Auvergne-Rhône-Alpes
<b>Metropolis</b>	Lyon Metropolis
<b>Arrondissement</b>	Lyon
<b>Subdivisions</b>	9 arrondissements
<b>Government</b>	
<span> </span> • Mayor (2020–2026)	Grégory Doucet <sup>[2]</sup> (EELV)
<b>Area</b> <sup>1</sup>	47.87 km <sup>2</sup> (18.48 sq mi)
<span> </span> • Urban (2020 <sup>[3]</sup> )	1,141.4 km <sup>2</sup> (440.7 sq mi)
<span> </span> • Metro (2020 <sup>[4]</sup> )	4,605.8 km <sup>2</sup> (1,778.3 sq mi)
<b>Population</b> (2021) <sup>[5]</sup>	522,250
<span> </span> • Rank	3rd in France
<span> </span> • Density	11,000/km <sup>2</sup> (28,000/sq mi)
<span> </span> • Urban (Jan. 2021) <sup>[6]</sup>	1,702,921
<span> </span> • Urban density	1,500/km <sup>2</sup> (3,900/sq mi)
<span> </span> • Metro (Jan. 2021) <sup>[7]</sup>	2,308,818
<span> </span> • Metro density	500/km <sup>2</sup> (1,300/sq mi)



# 3. LOD: SparQL tutorial with vCard

**vCard (short for Virtual Card)** is a file format standard for electronic business cards that allows the sharing and storage of contact information in a standardized, machine-readable format. vCards are widely used for exchanging personal contact details, and they can store a variety of information such as names, addresses, phone numbers, email addresses, websites, and more.



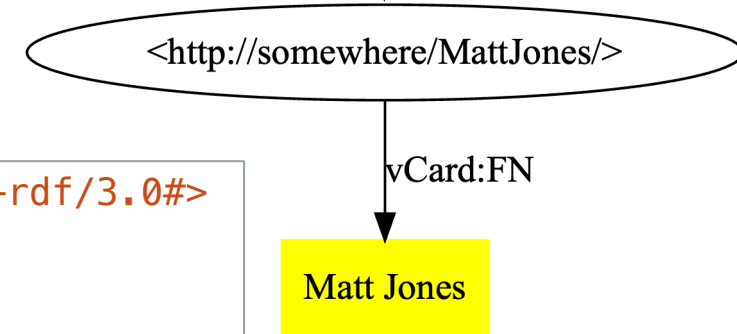
[vCard Vocabulary](#) - for describing People and Organizations

# 3. LOD : SparQL tutorial with vCard (1)

Database db1.ttl (*turtle* language)

```
@prefix vCard: <http://www.w3.org/2001/vcard-rdf/3.0#> .  
<http://somewhere/MattJones/> vCard:FN "Matt Jones" .
```

using [this tool](#)



Query rq1.rq (*SparQL* language)

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>  
  
SELECT ?x  
WHERE {  
  ?x vCard:FN "Matt Jones".  
}
```

Running the query against the database  
using [this tool](#) ([permanent link to the query](#))

X↑

[<http://somewhere/MattJones/>](#)

# 3. LOD : SparQL tutorial with vCard (2)

Database db2.ttl

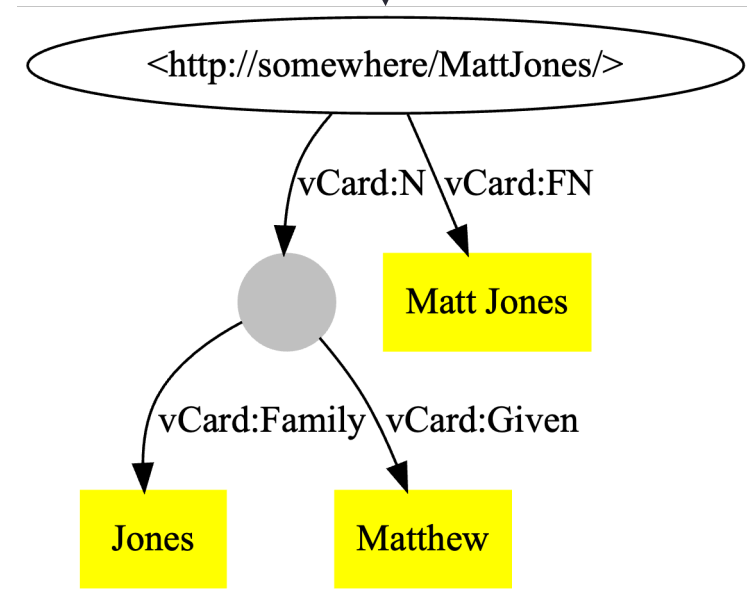
```
@prefix vCard: <http://www.w3.org/2001/vcard-rdf/3.0#> .  
  
<http://somewhere/MattJones/> vCard:FN "Matt Jones" .  
<http://somewhere/MattJones/> vCard:hasName _:b0 .  
_:b0 vCard:Family "Jones" .  
_:b0 vCard:Given "Matthew" .
```

[Permanent link](#)

Tip: called "Blank node"

Query rq2.rq

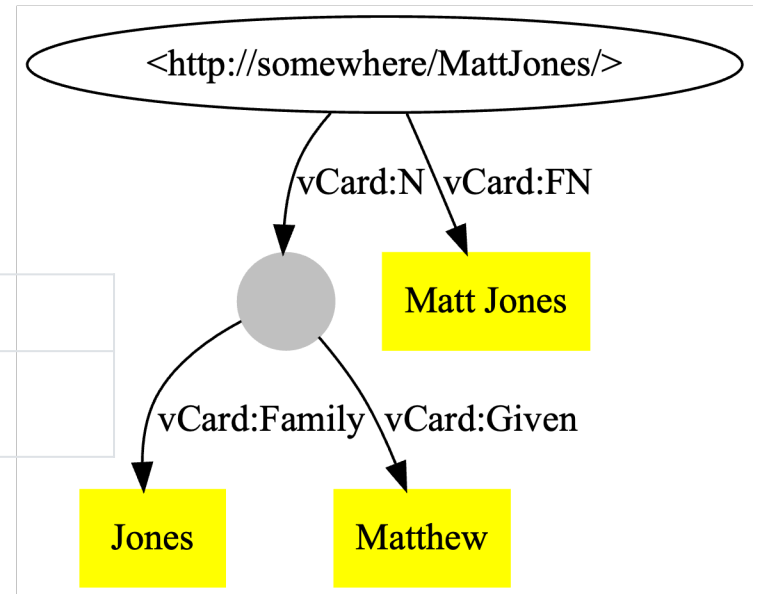
```
prefix vCard:  
<http://www.w3.org/2001/vcard-rdf/3.0#>  
  
SELECT ?x ?gn  
WHERE {  
  ?x vCard:FN "Matt Jones" ;  
    vCard:hasName ?z.  
  ?z vCard:Given ?gn.  
}
```



# 3. LOD : SparQL tutorial with vCard (2 con't)

Running the query against the database  
[Permanent link for the query](#)

X↑	Gn↑
<a href="http://somewhere/MattJones/">&lt;http://somewhere/MattJones/&gt;</a>	Matthew



# 3. LOD : SparQL tutorial with vCard (3)

```
@prefix vCard: http://www.w3.org/2001/vcard-rdf/3.0# .

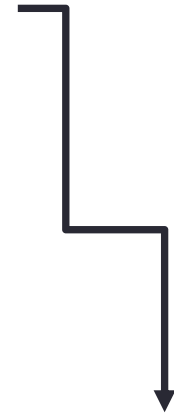
<http://somewhere/MattJones/> vCard:FN "Matt Jones" .
<http://somewhere/MattJones/> vCard:N _:b0 .
_:b0 vCard:Family "Jones" ;
    vCard:Given "Matthew" .

<http://somewhere/RebeccaSmith/> vCard:FN "Becky Smith" .
<http://somewhere/RebeccaSmith/> vCard:N _:b1 .
_:b1 vCard:Family "Smith" ;
    vCard:Given "Rebecca" .

<http://somewhere/JohnSmith/> vCard:FN "John Smith" .
<http://somewhere/JohnSmith/> vCard:N _:b2 .
_:b2 vCard:Family "Smith" ;
    vCard:Given "John" .

<http://somewhere/SarahJones/> vCard:FN "Sarah Jones" .
<http://somewhere/SarahJones/> vCard:N _:b3 .
_:b3 vCard:Family "Jones" ;
    vCard:Given "Sarah" .
```

Database db3.ttl

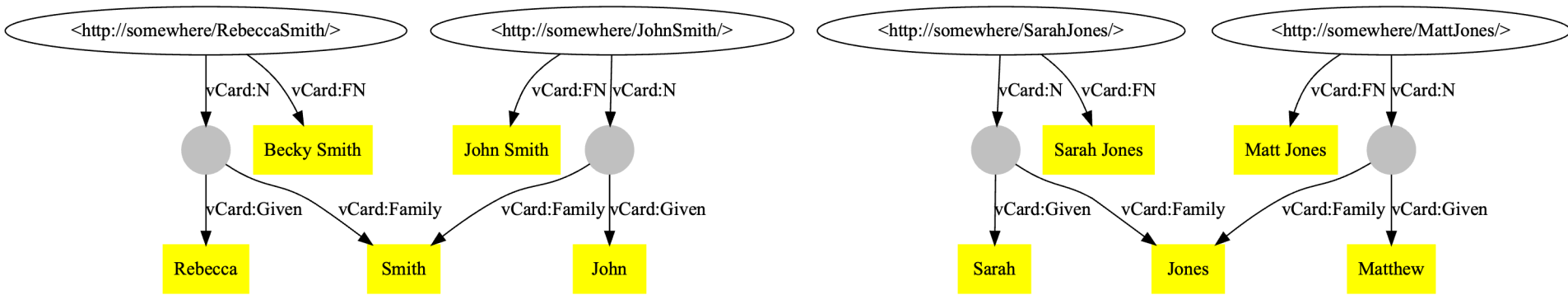


We talk about  
"triplestore" or  
"RDF graph".

It is important to realize that these are the same RDF graph and that the triples in the graph are in no particular order. They are just written in related groups above for the human reader - the machine does not care.

# 3. LOD : SparQL tutorial with vCard (3 con't)

Database db3.ttl



Query rq31.rq

```
prefix vCard: <http://.../3.0#>

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

Result

GivenName↑
John
Rebecca

# 4. SparQL : references

Let assume that we want to query an existing database of triples (such as "DBPedia ressource", or "Nobel Prize").

## Anatomy of a query

Declare prefix  
shortcut (optional)

```
PREFIX FOO: <...>  
PREFIX BAR: <...>
```

```
SELECT ...  
WHERE {  
  ...  
}
```

Query result clause

Query pattern

Query modifiers  
(optional)

```
GROUP BY ...  
HAVING ...  
ORDER BY ...  
LIMIT ...  
OFFSET ...
```

To illustrates concpets, next slides will query the "DBPedia ressource" database (accessible from this [endpoint](#) for example).

# 4. SparQL: optional

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?writer ?photo
WHERE {
  ?writer a dbo:Writer.
  OPTIONAL{
    ?writer foaf:depiction ?photo.
  }
}
```

In the result, the variables of **optional clauses** may therefore receive no value (null).

(show a demo with and without optional,  
using the this [endpoint](#))

# 4. SparQL: filter

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>

SELECT ?actor ?nat
WHERE {
  ?actor a                dbo:Actor;
         dbp:nationality ?nat.

  FILTER (CONTAINS(STR(?nat), "Japanese"))
}
LIMIT 5
```

← Query modifier

actor	nat
<a href="http://dbpedia.org/resource/Satoru_Matsuo">http://dbpedia.org/resource/Satoru_Matsuo</a>	Japanese
<a href="http://dbpedia.org/resource/Sayuri_Hara">http://dbpedia.org/resource/Sayuri_Hara</a>	Japanese
<a href="http://dbpedia.org/resource/Jukichi_Uno">http://dbpedia.org/resource/Jukichi_Uno</a>	Japanese
<a href="http://dbpedia.org/resource/Rei_Dan">http://dbpedia.org/resource/Rei_Dan</a>	Japanese
<a href="http://dbpedia.org/resource/Reisen_Ri">http://dbpedia.org/resource/Reisen_Ri</a>	Japanese

Total: 5, Shown: 5

# 4. SparQL: Informational and testing functions

- [CONTAINS](#): Evaluates whether the specified string contains the given pattern.
- [ISBLANK](#): Evaluates whether the given RDF term is a blank node.
- [ISIRI](#): Evaluates whether the given RDF term is an IRI.
- [ISLITERAL](#): Evaluates whether the given RDF term is a literal value.
- [ISNUMERIC](#): Evaluates whether the given RDF term is a numeric literal value.
- [ISURI](#): Evaluates whether the given RDF term is a URI.
- [LANG](#): Returns any language tags that are included with strings.
- [LANGMATCHES](#): Evaluates whether a string includes a language tag that matches the specified language range.

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>

SELECT ?actor ?nat
WHERE {
    ?actor a                dbo:Actor;
           dbp:nationality ?nat;
           dbp:name         ?name.
    filter( isLiteral(?nat) && LANGMATCHES(LANG(?name), "en"))
}
```

Functions described in [docs.cambridgesemantics.com](http://docs.cambridgesemantics.com)

# 4. SparQL : Logical functions

- [AND](#): Evaluates two logical expressions and returns true if both expressions are true.
- [BOUND](#): Evaluates whether an RDF term type is bound.
- [CASE](#): Evaluates a series of conditions and returns the matching result.
- [COALESCE](#): Evaluates a number of expressions and returns the results for the first expression that is bound and does not raise an error.
- [EXISTS](#): Evaluates whether the specified pattern exists.
- [IF](#): Evaluates a condition and returns the specified result depending on the outcome of the test.
- [IN](#): Evaluates whether the specified RDF term is found in any of the given test values.
- [NOT](#): Evaluates whether the specified logical expression is not true.
- [OR](#): Evaluates two logical expressions and returns true if at least one of the expressions is true.

Functions described in [docs.cambridgesemantics.com](http://docs.cambridgesemantics.com)

# 4. SparQL : Concat (query result)

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

SELECT ?p (CONCAT(?gn, " ", ?fn) AS ?name)
WHERE {
    ?p a foaf:Person;
       foaf:givenName ?gn;
       foaf:familyName ?fn.
}
LIMIT 5
```

p	name
<a href="http://dbpedia.org/resource/Sami_Kelopuro">http://dbpedia.org/resource/Sami_Kelopuro</a>	Sami Kelopuro
<a href="http://dbpedia.org/resource/Ben_Lamb_(poker_player)">http://dbpedia.org/resource/Ben_Lamb_(poker_player)</a>	Ben Lamb
<a href="http://dbpedia.org/resource/Juha_Helppi">http://dbpedia.org/resource/Juha_Helppi</a>	Juha Helppi
<a href="http://dbpedia.org/resource/Patrik_Antonius">http://dbpedia.org/resource/Patrik_Antonius</a>	Patrik Antonius
<a href="http://dbpedia.org/resource/Peter_Jetten">http://dbpedia.org/resource/Peter_Jetten</a>	Peter Jetten

Total: 5, Shown: 5

# 4. SparQL : Order By (query modifier)

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

SELECT ?p (CONCAT(?gn, " ", ?fn) AS ?name)
WHERE {
    ?p a foaf:Person;
        foaf:givenName ?gn;
        foaf:familyName ?fn.
}
ORDER BY ASC(?name)
LIMIT 5
```

p	name
<a href="http://dbpedia.org/resource/Ben_Lamb_(poker_player)">http://dbpedia.org/resource/Ben_Lamb_(poker_player)</a>	Ben Lamb
<a href="http://dbpedia.org/resource/Chris_Moorman">http://dbpedia.org/resource/Chris_Moorman</a>	Chris Moorman
<a href="http://dbpedia.org/resource/Christina_Pie">http://dbpedia.org/resource/Christina_Pie</a>	Christina Pie
<a href="http://dbpedia.org/resource/Eric_Haber">http://dbpedia.org/resource/Eric_Haber</a>	Eric Haber
<a href="http://dbpedia.org/resource/Ilari_Sahamies">http://dbpedia.org/resource/Ilari_Sahamies</a>	Ilari Sahamies

Total: 5, Shown: 5

# 4. SparQL : Group By (query modifier)

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

SELECT ?bpl (COUNT(?p) AS ?cp)
WHERE {
    ?p a foaf:Person;
        foaf:name ?n;
        dbo:birthPlace ?bp.
    ?bp dbp:name ?bpl.
}
GROUP BY ?bpl
LIMIT 5
```

More like  
this below

bpl	cp
Campos dos Goytacazes	74
Queensland	1335
San Diego	476
Berkeley, California	335
Breda	158

Total: 5, Shown: 5

# 4. SparQL : Group By and Having

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

SELECT ?bpl (COUNT(?p) as ?cp)
WHERE {
    ?p a foaf:Person;
        foaf:name ?n;
        dbo:birthPlace ?bp.
    ?bp dbp:name ?bpl.
}
GROUP BY ?bpl
HAVING(?cp <5)
LIMIT 7
```

bpl	cp
Cantrall	1
Casacalenda	2
Amsterdam-Noord	1
Province of Benevento	3
Ruffec	3
Berlin Township, Michigan	1
Bersillies	1

Total: 7, Shown: 7

# 4. SparQL : Group By and Group\_Concat

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

SELECT ?bpl (COUNT(?n) AS ?cn) (GROUP_CONCAT(?n, " | ") AS ?names)
WHERE {
    ?p a foaf:Person;
        foaf:name ?n;
        dbo:birthPlace ?bp.
    ?bp dbp:name ?bpl.
}
GROUP BY ?bpl
HAVING(?cn<5)
LIMIT 7
```

bpl	cn	names
Cantrall	1	Carl Vandagriff
Casacalenda	2	Carlo Montuori   Aldo Masciotta
Amsterdam-Noord	1	Calvin Twigt
Province of Benevento	2	Carlo Zotti   Pio of Pietrelcina
Ruffec	3	Carl Tourenne   Anne Charrier   Jean Jacques Marie Ferdinand de Béhagle
Berlin Township, Michigan	1	Carl D. Thompson
Bersillies	1	Camille Lou

Total: 7, Shown: 7

# 4. SparQL : Aggregate functions

- [AVG](#): Calculates the average (arithmetic mean) value for a group of numbers.
- [CHOOSE BY MAX](#): Returns the value from a group that corresponds to the maximum value from another group.
- [CHOOSE BY MIN](#): Returns the value from a group that corresponds to the minimum value from another group.
- [COUNT](#): Counts the number of values that exist for a group.
- [GROUP CONCAT](#): Concatenates a group of strings into a single string.
- [MAX](#): Returns the maximum value from a group of values.
- [MEDIAN](#): Returns the median number out of a group of numbers.
- [MIN](#): Returns the minimum value from a group of values.
- [MODE](#): Returns the mode (the value that occurs most frequently) from a group of values.
- [MODE PERCENT](#): Calculates the percentage of values in a group that belong to the mode.
- [SAMPLE](#): Returns an arbitrary value from the specified group of values.
- [SUM](#): Calculates the sum of the numbers within a group.
- [VAR](#): Calculates the unbiased (sample) variance of a group of numbers.
- [VARP](#): Calculates the biased (population) variance of a group of numbers.

# 4. SparQL : Union (query pattern)

Get all distinct Portuguese poets/philosophers

```
PREFIX dbc: <http://dbpedia.org/resource/Category:>
PREFIX dct: <http://purl.org/dc/terms/>

SELECT ?Xname
WHERE {
    { ?X dct:subject dbc:Portuguese_poets }
  UNION
    { ?X dct:subject dbc:Portuguese_philosophers}
  ?X foaf:name ?Xname.
}
ORDER BY DESC(?Xname)
LIMIT 5
```

Xname
Teófilo Braga
Texeira de Pascoaes
Texeira de Pascoaes
Teodoro de Almeida
Luis Filipe B. Teixeira
Total: 5, Shown: 5

# 4. SparQL : Exists, Not Exists !

NOT EXISTS offer flexible ways to check for the absence of a given pattern.

```
# Names of people where it is stated that they know at least one other person.
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
{
    ?x foaf:givenName ?name .
    FILTER EXISTS { ?x foaf:knows ?who . FILTER(?who != ?x) }
}
```

```
# Names of people who have not stated that they know anyone
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
{
    ?x foaf:givenName ?name .
    FILTER NOT EXISTS { ?x foaf:knows ?who }
}
```

# 4. SparQL : Minus, Not In

MINUS offer flexible ways to exclude possible solutions from the result set.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
{
  ?x foaf:givenName ?name .
  ?x foaf:knows ?y .
  MINUS { ?x foaf:knows <http://example.org/A> }
}
```

NOT IN: simpler form of negation for when you simply need to restrict a variable to not being in a given set of values

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
{
  ?x foaf:givenName ?name .
  ?x foaf:knows ?y .
  FILTER(?y NOT IN (<http://example.org/A>, <http://example.org/B>))
}
```

# 4. SparQL : Distinct (query clause)

Get all distinct Portuguese poets/philosophers

```
PREFIX dbc: <http://dbpedia.org/resource/Category:>
PREFIX dct: <http://purl.org/dc/terms/>

SELECT DISTINCT(?Xname) (COUNT(?X) AS ?cX)
WHERE {
    { ?X dct:subject dbc:Portuguese_poets }
    UNION
    { ?X dct:subject dbc:Portuguese_philosophers}
    ?X foaf:name ?Xname.
}
ORDER BY DESC(?cX)
LIMIT 5
```

Xname	cX
Texeira de Pascoaes	2
Jorge de Sena	1
Damião de Góis	1
Teodoro de Almeida	1
Desidério Murcho	1

Total: 5, Shown: 5

**Remark :** the same result can be obtained using "GROUP BY ?Xname" instead of "DISTINCT (?Xname)" !

# 4. SparQL : Bind (query pattern) - 1/2

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

SELECT ?p ?x
WHERE {
  ?p a foaf:Person.
  BIND (REPLACE(STR(?p), "Regex rule
    ^.*\/([\^\/]*)$", "$1") AS ?x)
}
LIMIT 4
```

p	x
<a href="http://dbpedia.org/resource/CaMia_Hopson">http://dbpedia.org/resource/CaMia_Hopson</a>	CaMia_Hopson
<a href="http://dbpedia.org/resource/Cab_Calloway">http://dbpedia.org/resource/Cab_Calloway</a>	Cab_Calloway
<a href="http://dbpedia.org/resource/Cab_Kaye">http://dbpedia.org/resource/Cab_Kaye</a>	Cab_Kaye
<a href="http://dbpedia.org/resource/Cabbrini_Foncette">http://dbpedia.org/resource/Cabbrini_Foncette</a>	Cabbrini_Foncette

Total: 4, Shown: 4

There exists numerous string functions:

LCASE, UCASE, STRLEN, SUBSTR, CONCAT, CONTAINS, ...

see [docs.cambridgesemantics.com](http://docs.cambridgesemantics.com)

# 4. SparQL : Bind (query pattern) – 2/2

```
SELECT ?name ?currentDateTime (CONCAT(?d," ",?m," ",?y) AS ?date_french)
WHERE {
  BIND(NOW () AS ?currentDateTime)
  BIND(YEAR (?currentDateTime) AS ?y)
  BIND(MONTH(?currentDateTime) AS ?m)
  BIND(DAY (?currentDateTime) AS ?d)
}
```

name	currentDateTime	date_french
	2024-11-01T09:38:08.998043	1 11 2024

Total: 1, Shown: 1

There exists numerous date functions:

PARSEDATE, MINUTES, TIMEZONE, YEARDAY, ...

see [docs.cambridgesemantics.com](https://docs.cambridgesemantics.com)

## 4. SparQL : IF/BOUND (query pattern) - 1/2

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbp: <http://dbpedia.org/property/>
```

```
SELECT ?p ?bb ?dd (BOUND(?bd) AS ?bbd) ?age
WHERE {
```

```
  ?p a foaf:Person .
```

```
  OPTIONAL{
```

```
    ?p dbp:birthDate ?bdate .
```

```
    BIND(xsd:date(?bdate) AS ?bb)
```

```
  }
```

```
  OPTIONAL{
```

```
    ?p dbp:deathDate ?ddate
```

```
    BIND(xsd:date(?ddate) AS ?dd)
```

```
  }
```

```
  BIND(
```

```
    IF(BOUND(?bb) && BOUND(?dd), YEAR(?dd)-YEAR(?bb), null)
```

```
    AS ?age)
```

```
}
```

```
OFFSET 18
```

```
LIMIT 6
```

To test wrong-shaped dates

Return True if the variable has a value

## 4. SparQL : IF/BOUND (query pattern) - 2/2

	p	bb	dd	bbd	age
<a href="http://dbpedia.org/resource/Cadalack_Ron">http://dbpedia.org/resource/Cadalack_Ron</a>	1981-04-28	2016-01-22		0	35
<a href="http://dbpedia.org/resource/Cadet_(rapper)">http://dbpedia.org/resource/Cadet_(rapper)</a>	1990-03-02	2019-02-09		0	29
<a href="http://dbpedia.org/resource/Cadmus_M._Wilcox">http://dbpedia.org/resource/Cadmus_M._Wilcox</a>	1824-05-20	1890-12-02		0	66
<a href="http://dbpedia.org/resource/Cadoc">http://dbpedia.org/resource/Cadoc</a>		0001-09-21		0	
<a href="http://dbpedia.org/resource/Cadwalader_Evans">http://dbpedia.org/resource/Cadwalader_Evans</a>		1841-10-26		0	
<a href="http://dbpedia.org/resource/Cadwalader_Ringgold">http://dbpedia.org/resource/Cadwalader_Ringgold</a>	1802-08-20	1867-04-29		0	65

Total: 6, Shown: 6

# 4. SparQL : nested query

To show the name and age of most aged people ...

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

SELECT ?name ?maxage
WHERE {
  {
    SELECT (MAX(?age) AS ?maxage)
    WHERE {
      ?person foaf:age ?age
    }
  }
  ?senior foaf:age ?maxage .
  ?senior foaf:name ?name
}
```

# 4. SparQL : Literals (1/3)

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?p ?name
WHERE {
  ?p a foaf:Person;
     foaf:name ?name;
     dbp:occupation "Actor"@en ;
     dbp:occupation "author"@en .
}
LIMIT 6
```

p	name
<a href="http://dbpedia.org/resource/Cady_McClain">http://dbpedia.org/resource/Cady_McClain</a>	Cady McClain
<a href="http://dbpedia.org/resource/Carl_Reiner">http://dbpedia.org/resource/Carl_Reiner</a>	Carl Reiner
<a href="http://dbpedia.org/resource/Carrie_Hope_Fletcher">http://dbpedia.org/resource/Carrie_Hope_Fletcher</a>	Carrie Hope Fletcher
<a href="http://dbpedia.org/resource/Rorke_Denver">http://dbpedia.org/resource/Rorke_Denver</a>	Rorke Denver
<a href="http://dbpedia.org/resource/Ross_Kemp">http://dbpedia.org/resource/Ross_Kemp</a>	Ross Kemp
<a href="http://dbpedia.org/resource/Roy_Hudd">http://dbpedia.org/resource/Roy_Hudd</a>	Roy Hudd

Total: 6, Shown: 6

# 4. SparQL : Literals (2/3)

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbp: <http://dbpedia.org/property/>

SELECT ?p ?name
WHERE {
  ?p a foaf:Person ;
  foaf:name ?name ;
  dbp:occupation "Actor"@en ;
  dbp:occupation "author"@en
  VALUES ?name { "Carl Reiner"@en "Ross Kemp"@en }
}
```

Give some results !

p	name
<a href="http://dbpedia.org/resource/Carl_Reiner">http://dbpedia.org/resource/Carl_Reiner</a>	Carl Reiner
<a href="http://dbpedia.org/resource/Ross_Kemp">http://dbpedia.org/resource/Ross_Kemp</a>	Ross Kemp

Total: 2, Shown: 2

# 4. SparQL : Literals (2/3)

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbp: <http://dbpedia.org/property/>

SELECT ?p ?name
WHERE {
    ?p a foaf:Person ;
        foaf:name ?name ;
        dbp:occupation "Actor"@en ;
        dbp:occupation "author"@en
    FILTER ( ( ?name="Carl Reiner"@en ) || ( ?name="Ross Kemp"@en ) )
}
```

Equivalent solution with a filter

p	name
<a href="http://dbpedia.org/resource/Carl_Reiner">http://dbpedia.org/resource/Carl_Reiner</a>	Carl Reiner
<a href="http://dbpedia.org/resource/Ross_Kemp">http://dbpedia.org/resource/Ross_Kemp</a>	Ross Kemp

Total: 2, Shown: 2

# 4. SparQL : Math functions

- [ABS](#): Calculates the absolute value of the specified number.
- [ADD](#): Adds two numeric values.
- [AVG](#): Calculates the average (arithmetic mean) value for a group of numbers.
- [CEIL](#): Rounds up a numeric value to the nearest integer.
- [COS](#): Calculates the cosine of an angle.
- [EXP](#): Raises e to the specified power.
- [FACT](#): Calculates the factorial of the specified number.
- [FLOOR](#): Rounds down a numeric value to the nearest integer.
- [HAMMING DIST](#): Calculates the hamming distance between two values.
- [HAVERSINE DIST](#): Computes the haversine distance between two latitude and longitude values.
- [LN](#): Calculates the natural logarithm of a double value.
- [MOD](#): Calculates the modulo of the division between two numbers.
- [PI](#): Returns the value for Pi.
- [POWER](#): Raises the specified number to the specified power.
- [RADIANS](#): Converts to radians an angle value that is in degrees.
- [RAND](#): Returns a random double value between 0 and 1. [ROUND](#): Rounds a numeric value to the nearest integer.
- [SQRT](#): Calculates the square root of a number.

Functions described in [docs.cambridgesemantics.com](https://docs.cambridgesemantics.com)

# 5. LOD practical work and project

Main site for the "Big Data computing technologies" course: [here](#)

Practical work for LOD technology: [here](#)

LOD project expectations: [here](#)