

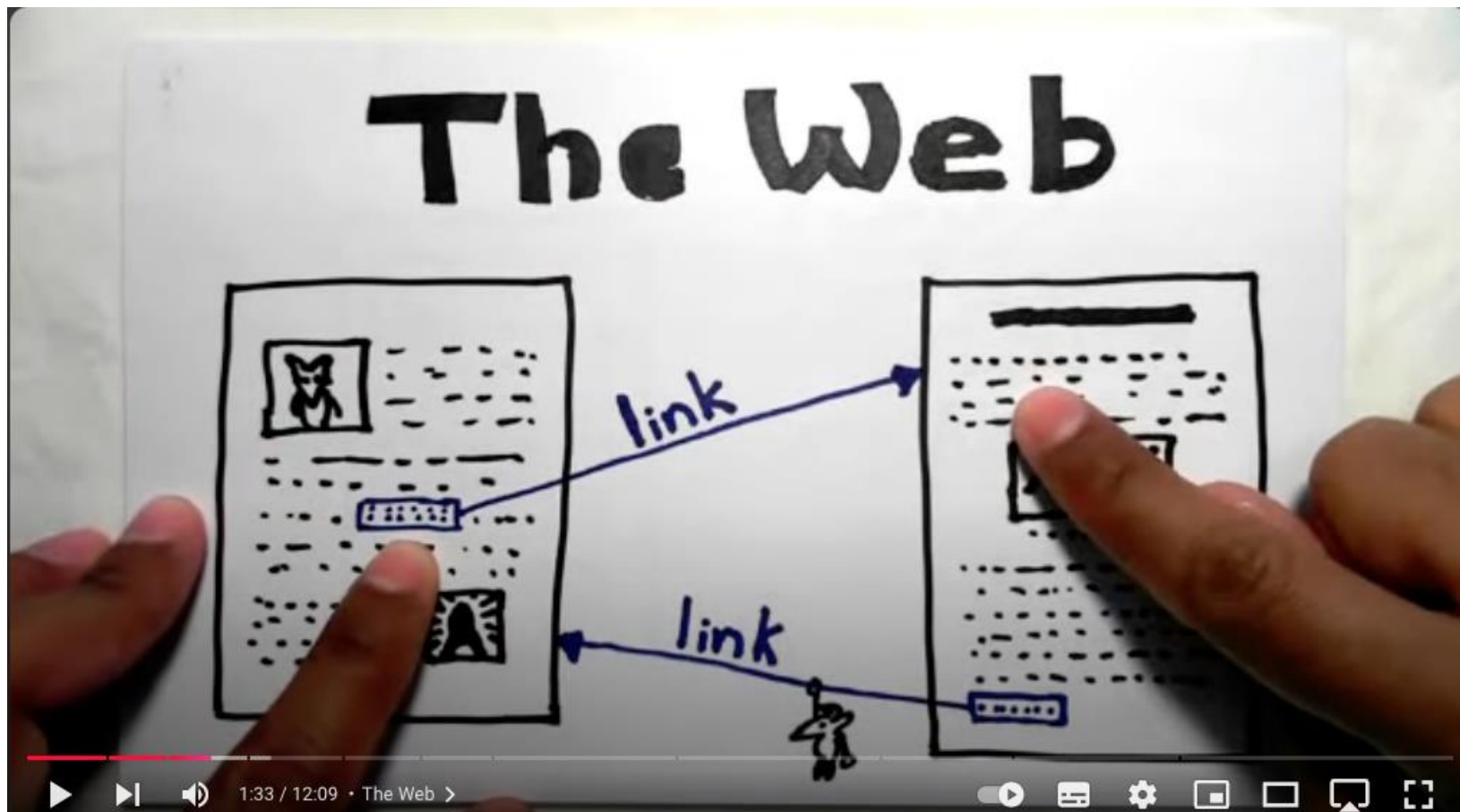


# LINKED OPEN DATA

---

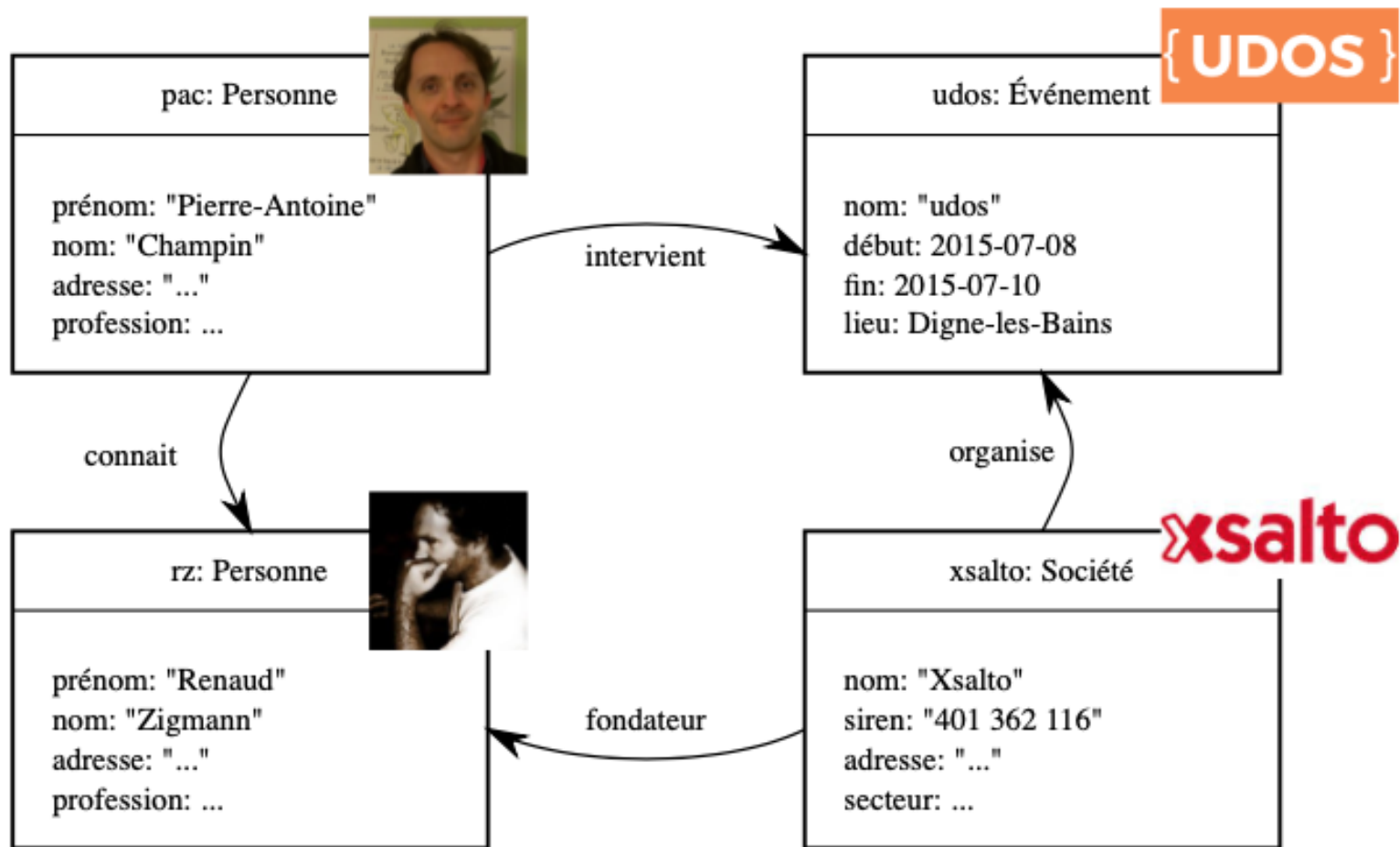
1. *Web des données, Linked Data*
2. *Resource Description Framework (RDF)*
3. SparQL

## 2. RDF : What is Linked Data ?



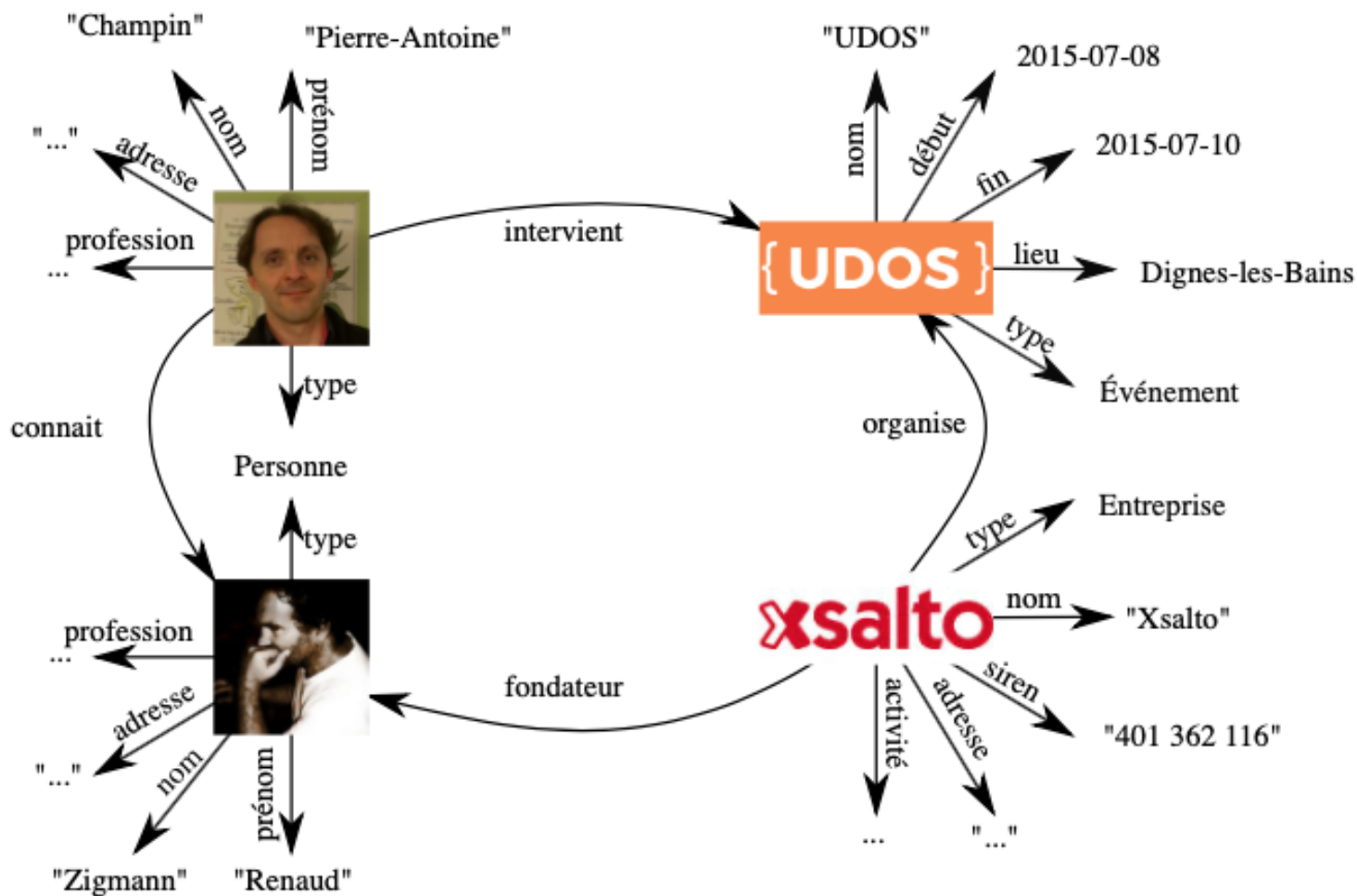
What is linked data?  
from Manu Sportny

## 2. RDF : des données aux données liées



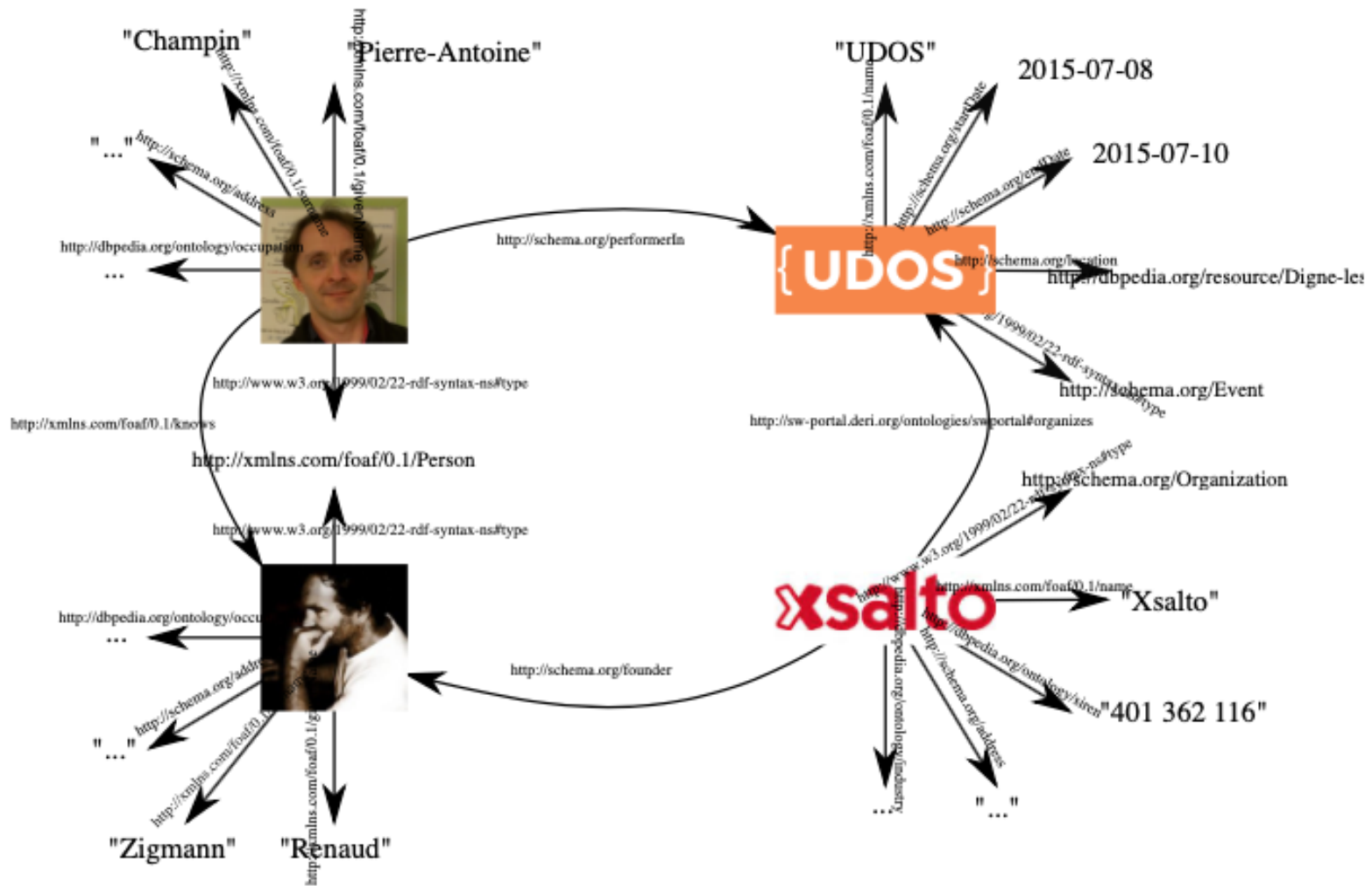
Données classiques

## 2. RDF : des données aux données liées



Données vues comme un graphe

# 2. RDF : Des données aux données liées



Données vues comme un graphe avec des IRIs

## 2. RDF : syntaxe abstraite

Toute information en RDF est représentée par un *triplet*, signifiant qu'une *chose* est en *relation* avec une autre.

Exemple :

Le laboratoire LIRIS (**Subject**)

a pour membre (**Predicate**)

Pierre-Antoine Champin (**Object**)

## 2. RDF : syntaxe abstraite

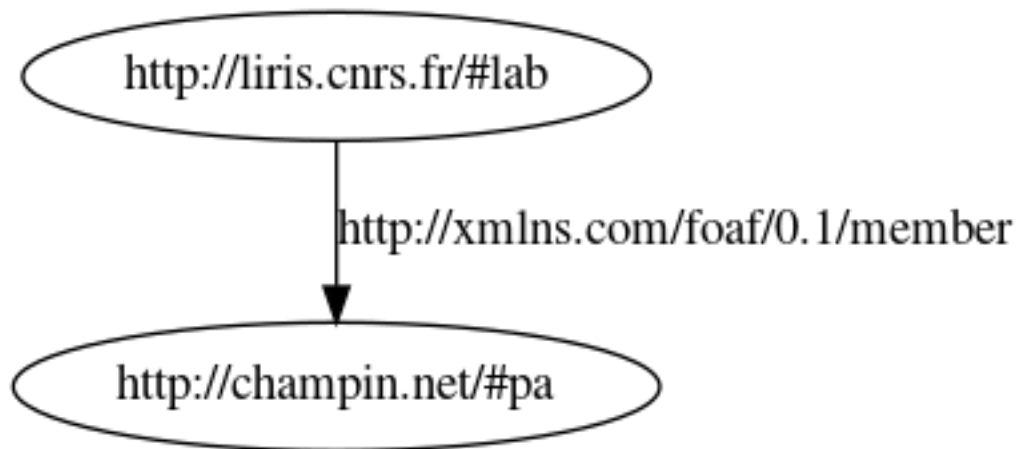
Les choses sont nommées par des IRIs :

<http://liris.cnrs.fr/#lab>

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

<http://champin.net/#pa>

On peut représenter ceci graphiquement :





## 2. RDF : préfixes

Pour simplifier les **notations**, on définit des préfixes courts correspondant à des préfixes d'IRI :

liris: → <http://liris.cnrs.fr/#>

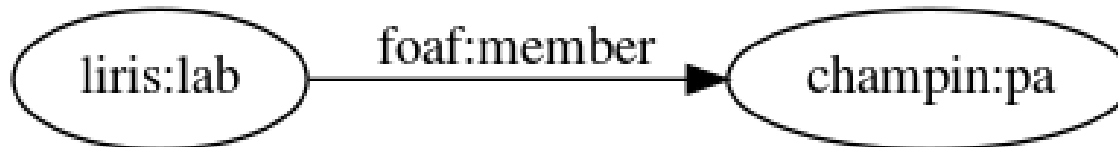
foaf: → <http://xmlns.com/foaf/0.1/>

champin: → <http://champin.net/#>

On utilise ensuite des *noms préfixés* :

`liris:lab foaf:member champin:pa`

et également sous forme graphique :

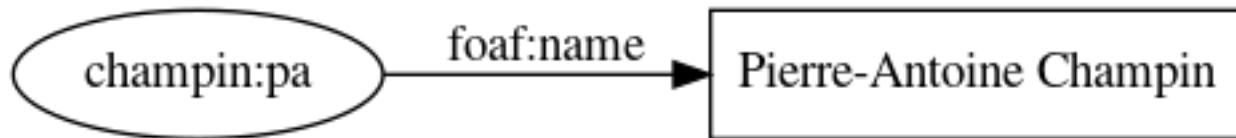


## 2. RDF : les littéraux

On peut également lier une ressource à une *donnée typée* (chaîne de caractères, entier, réel...), nommée un littéral.

```
champin:pa foaf:name "Pierre-Antoine Champin"
```

Traditionnellement, on représente les littéraux par des nœuds rectangulaires :

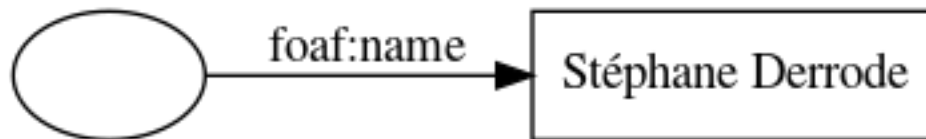


## 2. RDF : les nœuds muets

Enfin, RDF permet de parler d'une ressource sans connaître son IRI:

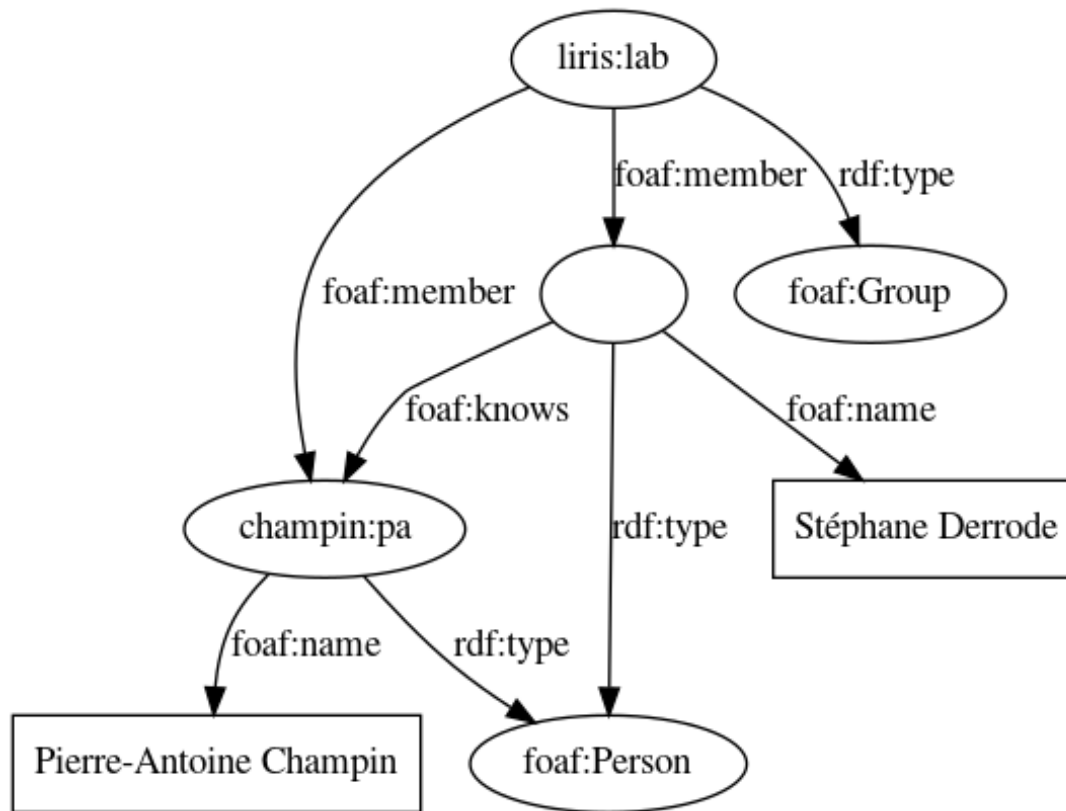
```
(quelque chose) foaf:name "Stéphane Derrode"
```

On parle alors de nœud *muet* (par analogie aux variables muettes). Graphiquement, on représente cette ressource par un nœud vierge (*blank node*).



## 2. RDF : exemple de graphe

Un ensemble de triplets forme un graphe orienté étiqueté.



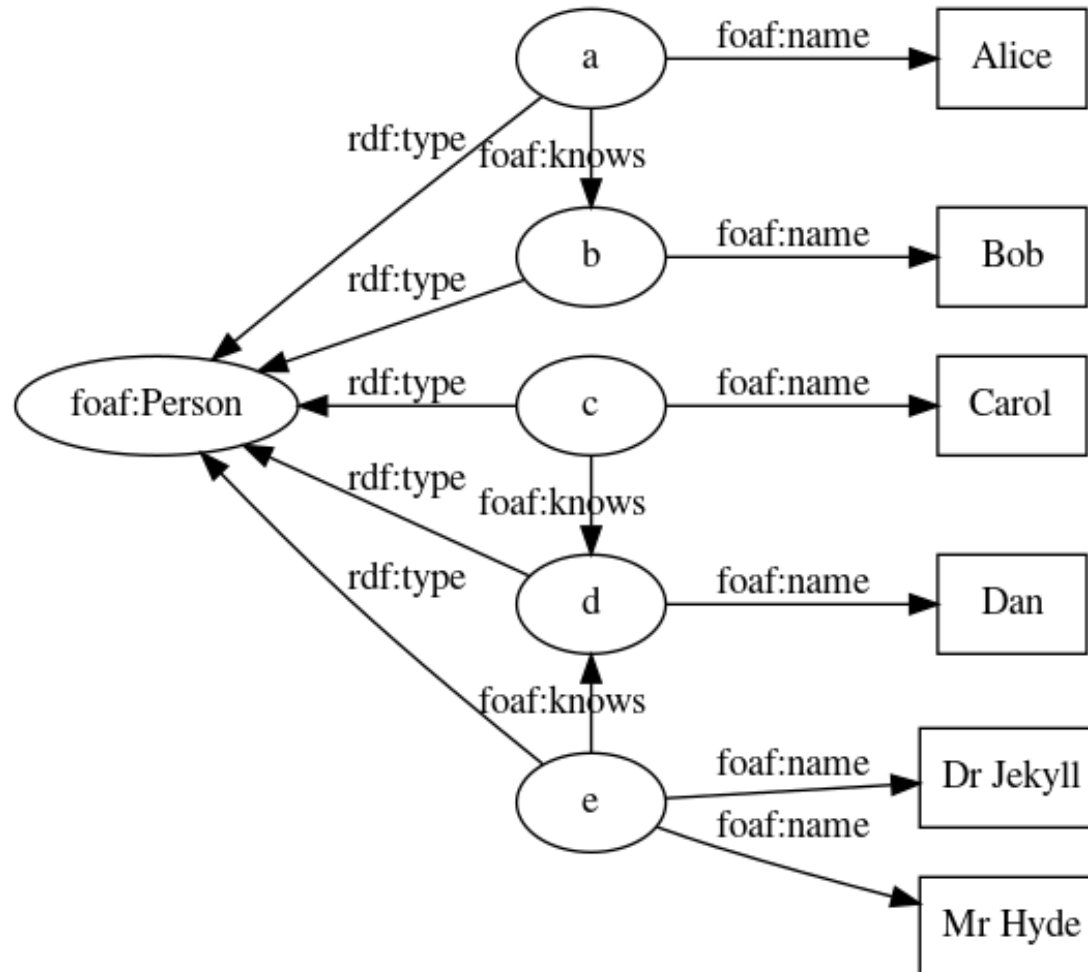
# 3. SparQL : objectifs

- Vous donner des bases pour écrire des requêtes SparQL.
- Bonus: lire/écrire du *Turtle* (très proche de SparQL).
- Ce n'est qu'une introduction ; pour en savoir plus :

<http://www.w3.org/TR/sparql11-overview/>

# 3. SparQL : basic query (1/3)

- Considérons les données suivantes



# 3. SparQL : basic query (2/3)

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

```
SELECT ?n1 ?n2
```

```
WHERE {
```

```
  ?p1 a foaf:Person;
```

```
  foaf:name ?n1;
```

```
  foaf:knows ?p2.
```

```
  ?p2 a foaf:Person;
```

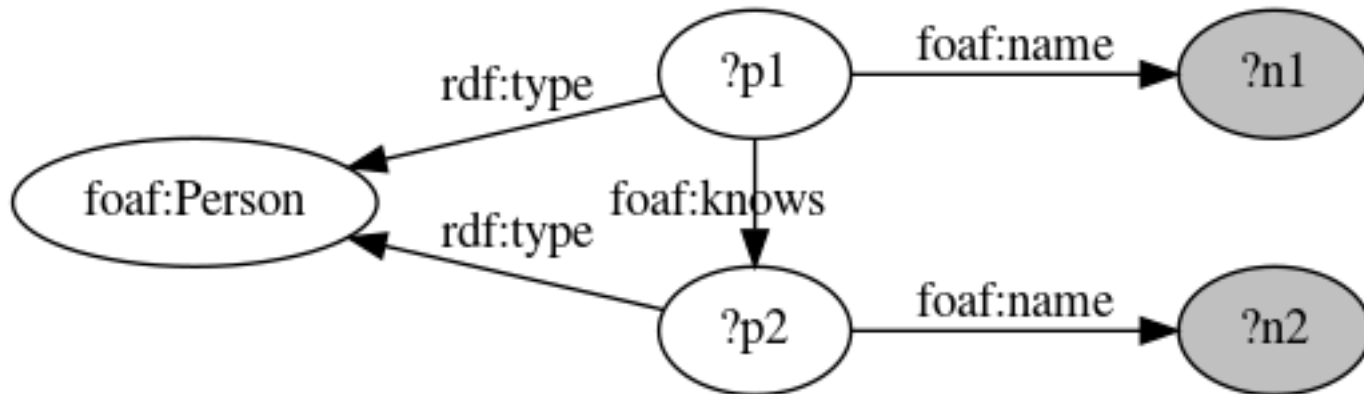
```
  foaf:name ?n2.
```

```
}
```

Une classe

Une variable

Une propriété



# 3. SparQL : basic query (3/3)

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

```
SELECT ?n1 ?n2
```

```
WHERE {
```

```
  ?p1 a    foaf:Person;
```

```
      foaf:name ?n1;
```

```
      foaf:knows ?p2.
```

```
  ?p2 a    foaf:Person;
```

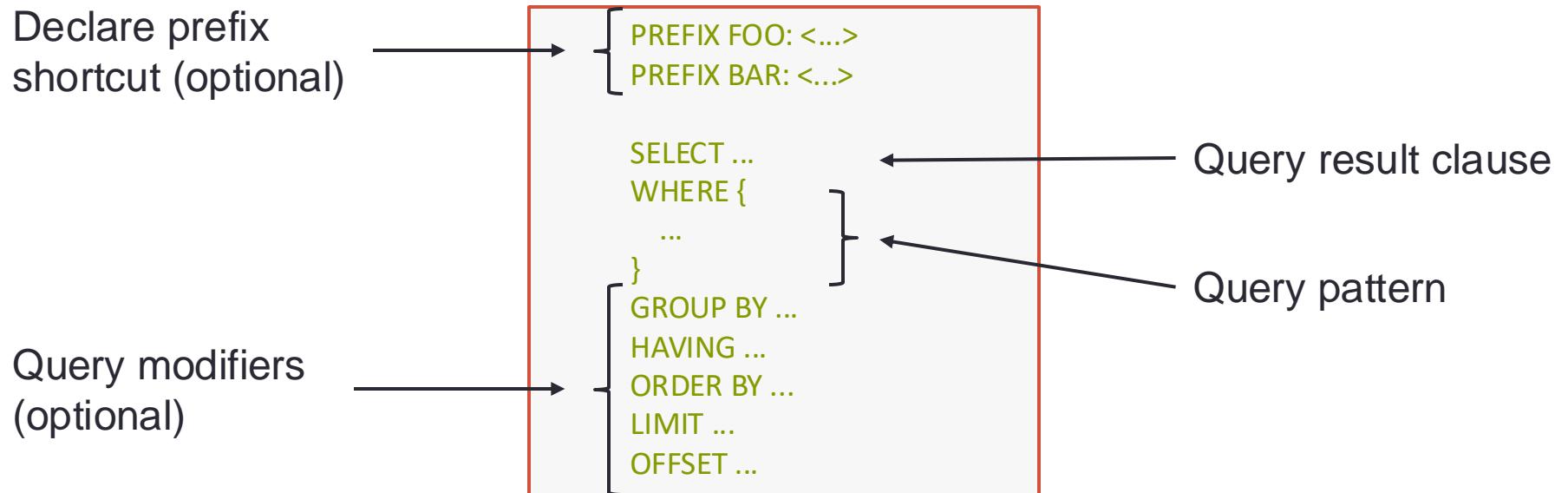
```
      foaf:name ?n2;
```

```
}
```

n1	n2
Alice	Bob
Carol	Dan
Dr Jekyll	Dan
Mr Hide	Dan



# 3. SparQL : anatomy of a query



# 3. SparQL : Optional (query pattern)

Sous-graphe optionnel :

```
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.
  }
}
```

Dans le résultat, les variables des clauses optionnelles peuvent donc ne recevoir aucune valeur (null).

# 3. SparQL : Filter (query pattern)

```
SELECT ?p
WHERE {
  ?p a    foaf:Person;
      foaf:age ?age.
  FILTER(?age > 18 && ?age < 30)
}
```

```
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), "British"))
}
OFFSET 6
LIMIT 4
```

Two query modifiers

actor	nat
<a href="http://dbpedia.org/resource/David_Dixon">http://dbpedia.org/resource/David_Dixon</a>	British
<a href="http://dbpedia.org/resource/David_Sterne">http://dbpedia.org/resource/David_Sterne</a>	British
<a href="http://dbpedia.org/resource/David_Westhead">http://dbpedia.org/resource/David_Westhead</a>	<a href="http://dbpedia.org/resource/British_people">http://dbpedia.org/resource/British_people</a>
<a href="http://dbpedia.org/resource/Delaval_Astley,_23rd_Baron_Hastings">http://dbpedia.org/resource/Delaval_Astley,_23rd_Baron_Hastings</a>	British

Total: 4, Shown: 4

# 3. 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 foaf: <http://xmlns.com/foaf/0.1/>
```

```
SELECT * ← Display all the variables
```

```
WHERE {
```

```
  ?x foaf:name ?name ;
```

```
  foaf:age ?age .
```

```
  FILTER(
```

```
    IF (LANGMATCHES(LANG(?name), "FR"), ?age>=18, ?age>=21)
```

```
  )
```

```
}
```

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

# 3. SparQL : Filter (query pattern)

Ce n'est pas une IRI !

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

SELECT ?actor ?nat
WHERE {
    ?actor a      dbo:Actor;
           dbp:nationality ?nat.
    FILTER (! ISIRI(?nat))
    FILTER (contains(?nat, "British"))
}
OFFSET 6
LIMIT 4
```

	actor	nat
	<a href="http://dbpedia.org/resource/David_Dixon">http://dbpedia.org/resource/David_Dixon</a>	British
	<a href="http://dbpedia.org/resource/David_Sterne">http://dbpedia.org/resource/David_Sterne</a>	British
	<a href="http://dbpedia.org/resource/Delaval_Astley,_23rd_Baron_Hastings">http://dbpedia.org/resource/Delaval_Astley,_23rd_Baron_Hastings</a>	British
	<a href="http://dbpedia.org/resource/Alfred_Herbert_Richardson">http://dbpedia.org/resource/Alfred_Herbert_Richardson</a>	British

Total: 4, Shown: 4

# 3. 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](https://docs.cambridgesemantics.com)

# 3. 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

# 3. 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



# 3. 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		

# 3. 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(COUNT(?p)<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		

# 3. 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(COUNT(?p)<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

# 3. 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.

# 3. 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

# 3. 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 }
}
```

# 3. 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>))
}
```

# 3. 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)" !



# 3. 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), "^.*/(^[^/]*)$", "$1") AS ?x)
```

```
}
```

```
LIMIT 4
```

"Regex rule"



	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)

# 3. 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)

### 3. 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)
```

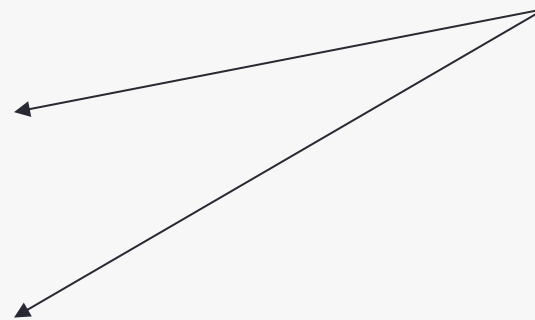
```
    AS ?age)
```

```
}
```

```
OFFSET 18
```

```
LIMIT 6
```

To test wrong-shaped dates



Return True if the variable has a value

### 3. 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

# 3. 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
}
```

# 3. 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

# 3. 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

# 3. 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		



# 3. 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)

# 4. TP : requêtes sur DBPédia

Liens vers le sujet du TP :

<http://perso.ec-lyon.fr/derrode.stephane/Teaching/ECCBigData/TP1/readme>

Compte-rendu de TP **individuel**, à déposer avant la fin de séance sur :  
pedagogie3.ec-lyon.fr

Les consignes sont détaillées dans le sujet.

## Tips:

- Pour trouver les préfixes, visitez ce site : <http://prefix.cc>
- SparQL query-validator : [SparQL.org](http://SparQL.org) (bonus : il ré-indenté et améliore la lisibilité)