

WEB SEMANTIQUE

COURS 4

SHINY, OH SO SHINY SPARQL

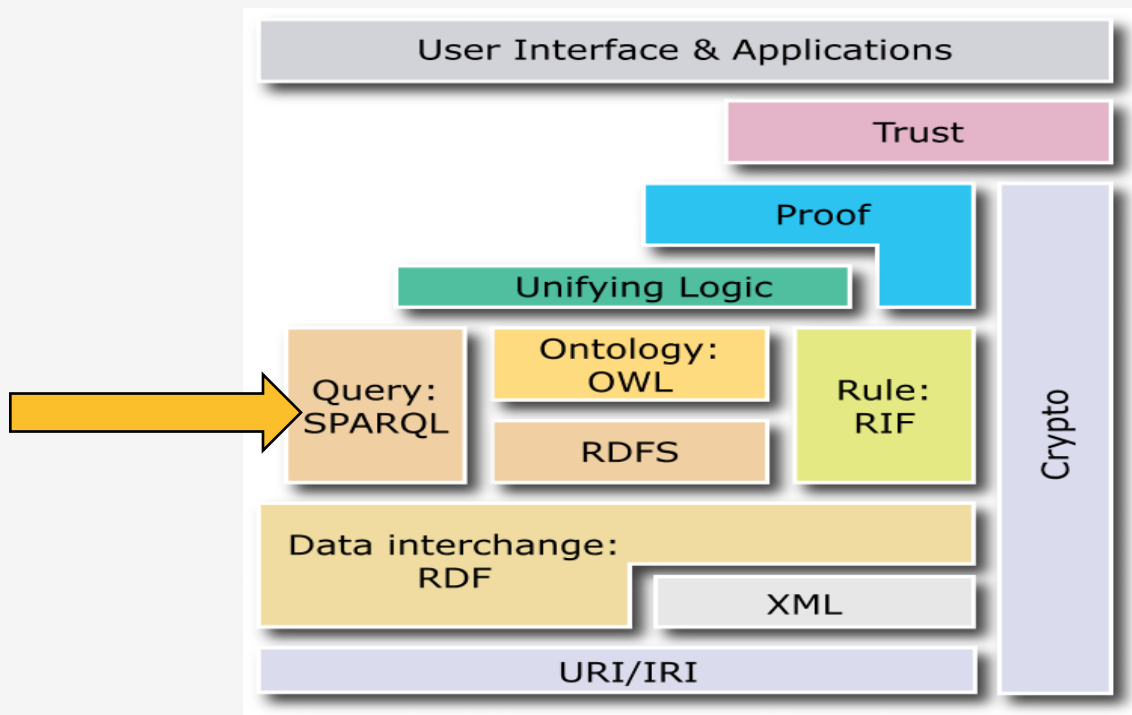




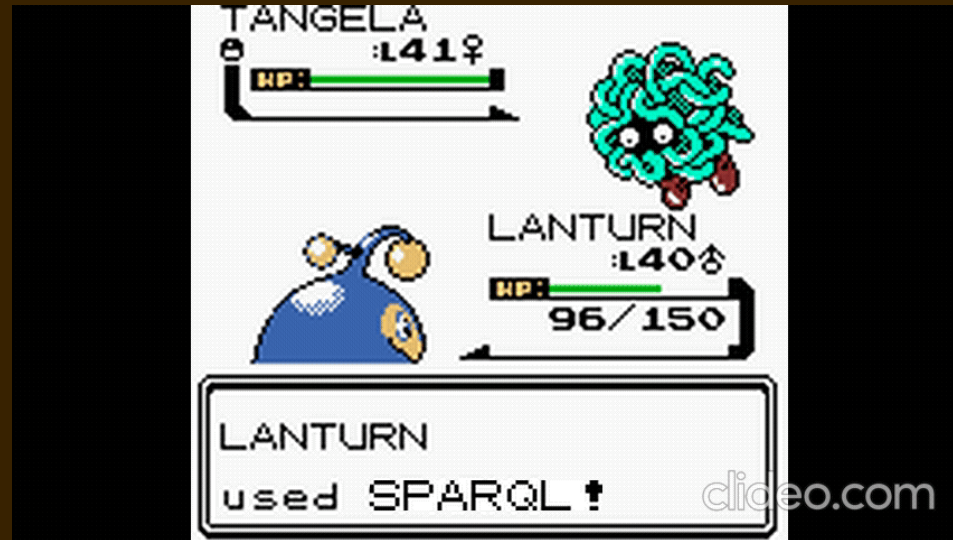
PRÉCÉDEMMENT...

- Créer des vocabulaires partagés dotés d'une sémantique (ontologie) pour rendre les données RDF interopérables
- Difficile d'atteindre un consensus...
- Mais admettons qu'on y arrive ! Comment exploiter les graphes de connaissances ainsi produits ?

LE MODÈLE EN COUCHES DU W3C



PARLONS DE SPARQL



SPÉCIFICATIONS SPARQL



- **SPARQL 1.0**

- SPARQL Query Language for RDF
- SPARQL Protocol for RDF
- SPARQL Query Results XML Format

- **SPARQL 1.1**

- SPARQL Federation Extension
- **SPARQL Update**
- SPARQL Service Description
- SPARQL Query Results JSON Format
- SPARQL Graph Store HTTP Protocol
- SPARQL Entailment Regimes
- SPARQL New Features and Rationale

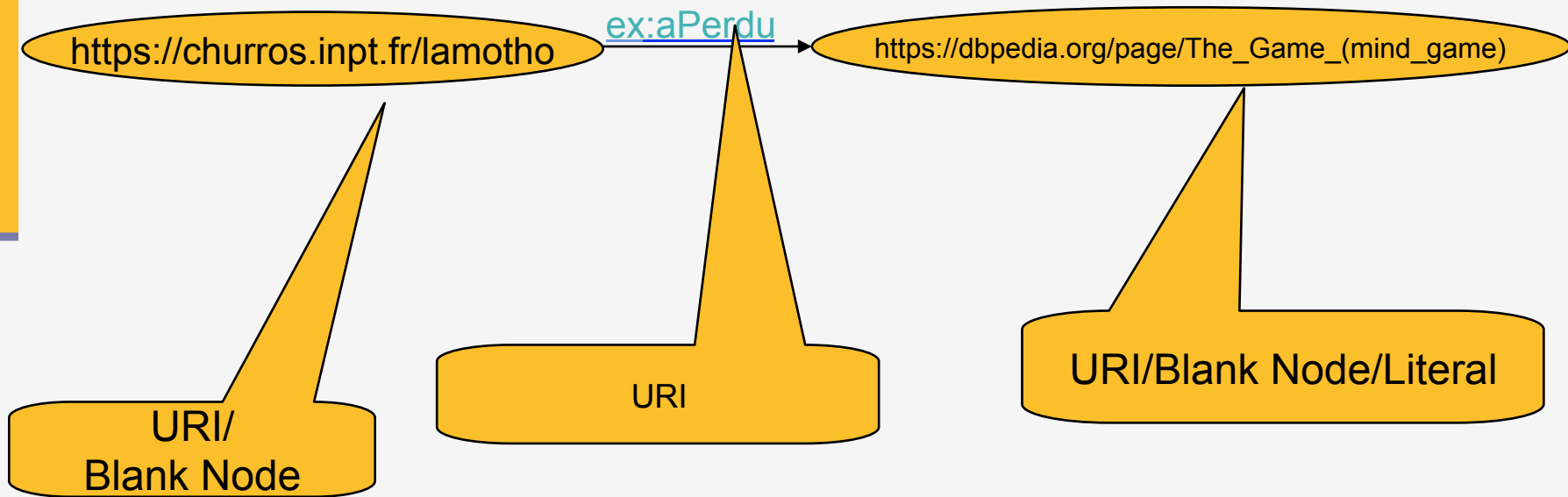
SPARQL

- Acronyme récursif : SPARQL Protocol and RDF Query Language
- **Protocole** d'échange de **requêtes** et de **résultats**
- Spécification du **format** des résultats
- Langage d'**interrogation** du web sémantique
- Envoie des requêtes à des dépôts SPARQL
- Avec SPARQL 1.1: langage de **mise à jour**
- **Documentation** sur le site du W3C <https://www.w3.org/TR/sparql11-query/>



RAPPEL RDF : LE TRIplet

- Olivier Lamothe a perdu le Jeu.



RAPPEL : NOTION DE NAMESPACE



- Un namespace est un raccourci pour éviter d'écrire toute l'URI à chaque fois.
- Exemples :
 - Namespace rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 - rdf:type signifie <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
 - Namespace dpb: <https://dbpedia.org/page/>



POURQUOI « # » ET POURQUOI « / » ?

URL = Locator `http://` `example.net` `/home/documents/toto.html` `# section1`
 protocole Nom de la machine Chemin du fichier Réf. interne

À la base, le Web c'est pour partager des documents depuis des machines : il faut les **localiser** pour les **récupérer**

<http://example.net/poulet#toto>

Je récupère le document « poulet » en entier mais je fais référence à la section « toto »

<http://example.net/poulet/toto>

Je récupère le document « toto » uniquement qui est dans le dossier « poulet »

SYNTAXE ET PRINCIPE



SYNTAXE TURTLE (TTL)



- Terse RDF Triple Language: format de **sérialisation** pour RDF
- Sous-ensemble de la notation N3
- Populaire car compacte et simple d'emploi
- Triplets factorisables avec **;** (sujet commun) ou **,** (sujet + prédicat communs)
- La déclaration de **préfixes** permet d'alléger ensuite l'écriture des triplets
- URI: **<uri/de/la/ressource>**
- URI préfixées: **prefix:suffix**
- Nœuds vides (blank nodes): **_:id**
- Littéraux typés: **"valeur"^^type "valeur"@en**
- Raccourci pour rdf:type **a**
- **Documentation** W3C: <https://www.w3.org/TR/2014/REC-turtle-20140225/>

EXEMPLE DE RDF EN TTL

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dc: <http://dublincore.org/documents/dcmi-namespace/>.
<http://www.irit.fr/~Marc.Pantel
>   dc:author <http://ns.irit.fr/Marc#me>;
     dc:publisher <http://www.irit.fr>;
     dc:title "Page professionnelle de Marc".
<http://www.irit.fr> dc:title "Institut de Recherche en Informatique de
Toulouse".
```



GRAPHES NOMMÉS

**SPOILER
ALERT!**



- Graphe = ensemble de triplets
- On peut donner des identifiants (URI) à des graphes Un graphe avec un identifiant == un graphe nommé
- **N-quads** : sujet prédicat objet graphe

```
@prefix : <http://data.musees.fr/> .  
@prefix musee: <http://musees.fr/ontology#>  
  
:Graphe1 {:MNHN musee:ville :Paris .}  
:Graphe2 {:Louvre musee:ville :Paris .}
```

STRUCTURE D'UNE REQUÊTE SPARQL



PREFIX ...

Liste des préfixes

SELECT ...

Sélection du format de résultat

FROM ...

Optionnel : pour choisir un seul graphe nommé

WHERE {

...

BGP : Basic Graph Pattern : **Motif de graphe**
Exprimé en syntaxe turtle

}

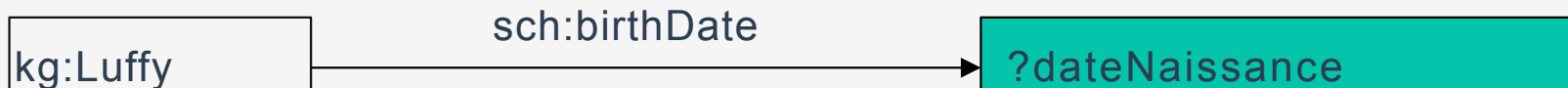
MOTIF DE GRAPHE

- Principe : écrire un graphe à « trous »
- Donner des « noms » à ces trous \Rightarrow **variables**
- Le moteur SPARQL va ensuite **appairer** le motif aux données : **isomorphisme de graphe**



MOTIF DE GRAPHE

- Principe : écrire un graphe à « trous »
- Donner des « noms » à ces trous \Rightarrow **variables**
- Le moteur SPARQL va ensuite **appariar** le motif aux données : **isomorphisme de graphe**
 - Quelle est la date de naissance de kg:Luffy ?



MOTIF DE GRAPHE

- Principe : écrire un graphe à « trous »
- Donner des « noms » à ces trous \Rightarrow **variables**
- Le moteur SPARQL va ensuite **appairer** le motif aux données : **isomorphisme de graphe**
 - Quelle est la date de naissance de kg:Luffy ?

```
PREFIX kg: <http://example.org/knowledgeGraph#>
PREFIX sch: <https://schema.org/>

SELECT ?dateNaissance WHERE {
  kg:Luffy sch:birthDate ?dateNaissance.
}
```



RÉSULTATS

JSON

```
{ "head": {  
  "vars": [ "dateNaissance" ]  
},  
"results": {  
  "bindings": [  
    {  
      "dateNaissance":  
        { "type": "literal",  
          "datatype": "http://www.w3.org/2001/XMLSchema#date",  
          "value": "1999/05/05" }  
    }  
  ]  
}
```

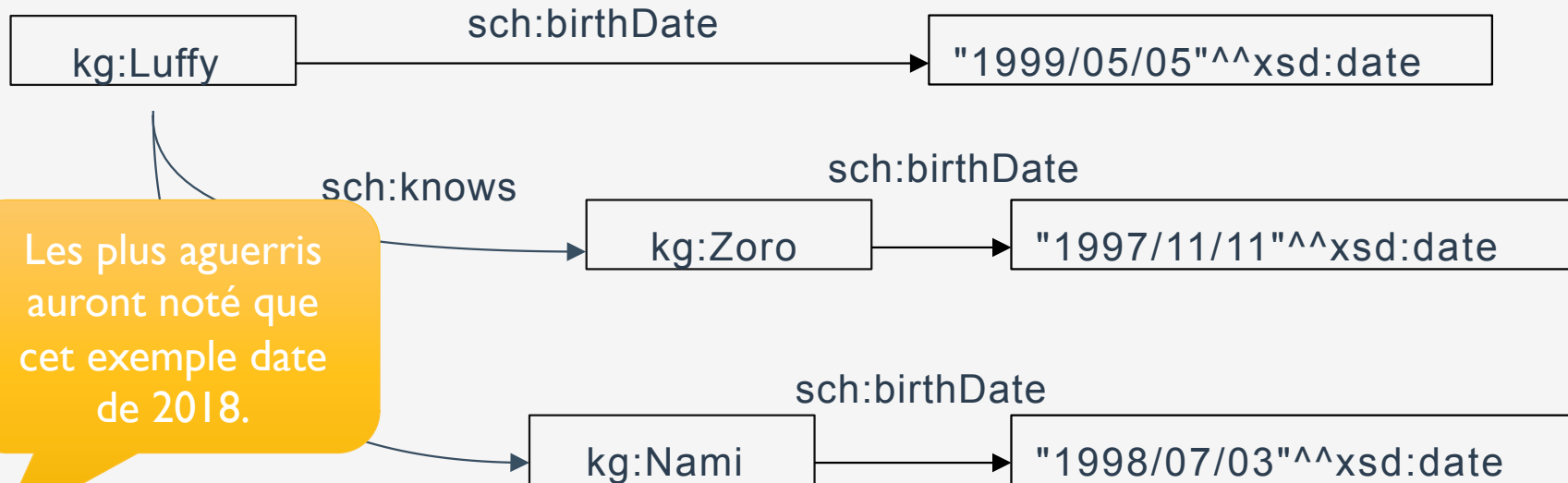
Tableau

dateNaissance

"1999/05/05"^^xsd:date



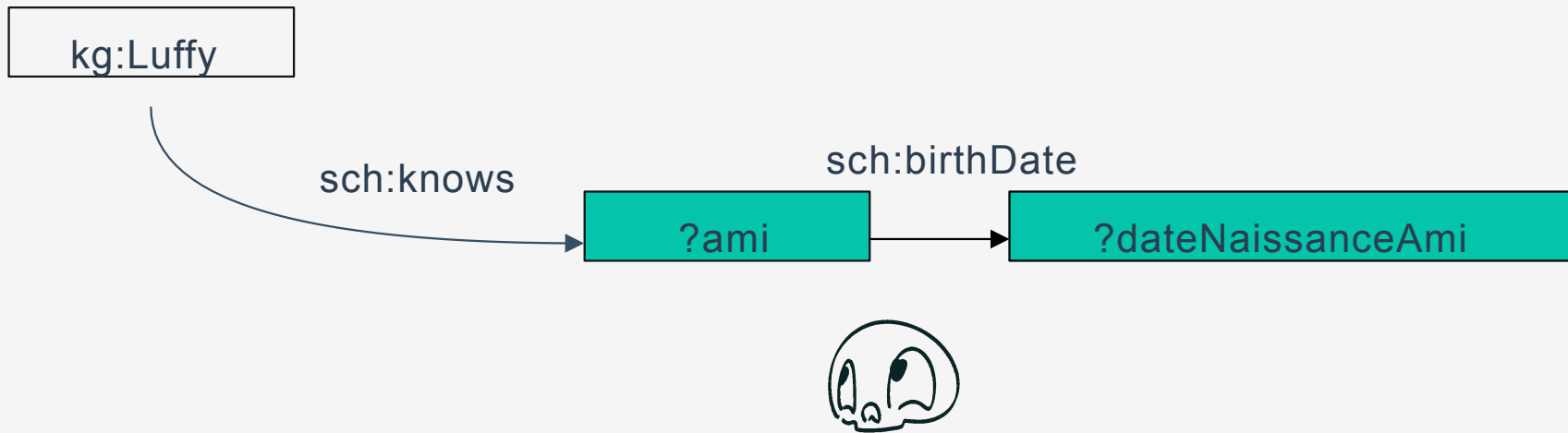
QUELLES SONT LES DATES DE NAISSANCES DES AMIS DE LUFFY ?



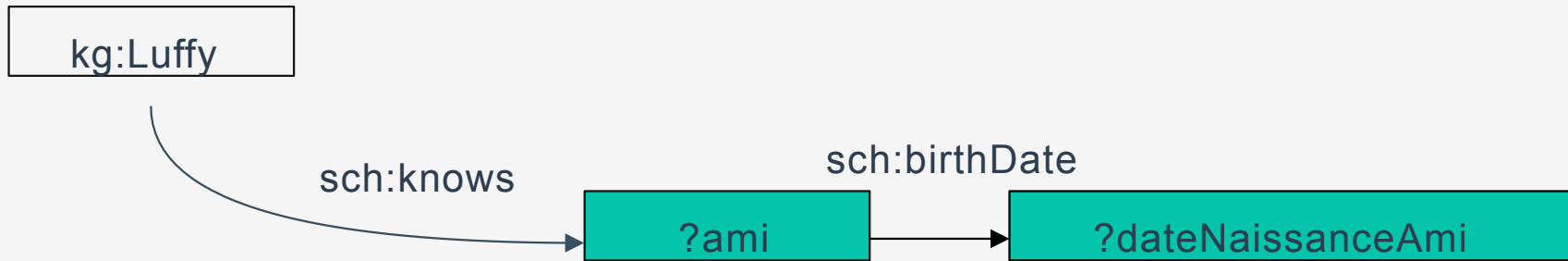
Les plus aguerris
auront noté que
cet exemple date
de 2018.



1. CRÉATION DU GRAPHE À TROUS



2. RÉDACTION DE LA REQUÊTE TRIPLET PAR TRIPLET

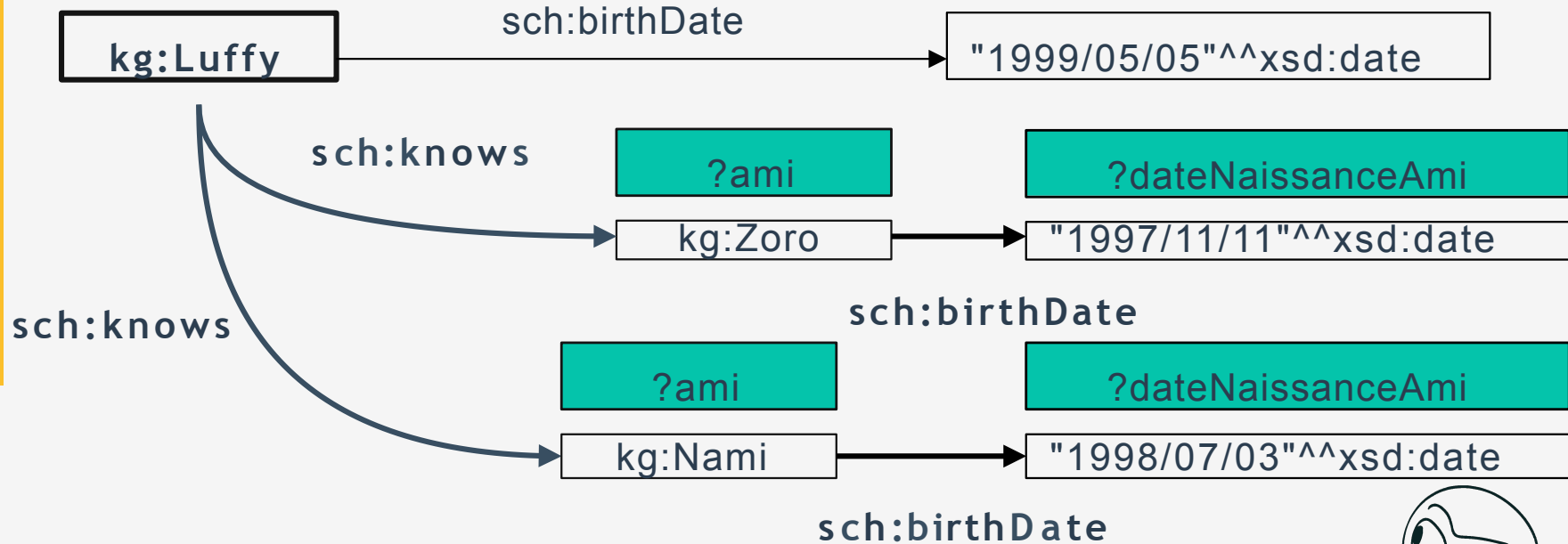


```
PREFIX kg: <http://example.org/knowledgeGraph#>
PREFIX sch: <https://schema.org/>

SELECT ?ami ?dateNaissanceAmi WHERE {
  kg:Luffy sch:knows ?ami.
  ?ami sch:birthDate ?dateNaissanceAmi.
}
```



3. APPARIEMENT PAR LE MOTEUR SPARQL



4. RETOUR DES RÉSULTATS

```
{ "head": {  
  "vars": [ "ami", "dateNaissanceAmi" ]  
},  
  "results": {  
    "bindings": [  
      {  
        "ami": { "type":  
          "uri",  
          "value": "http://example.org/knowledgeGraph#Zoro" }  
      },  
      {  
        "dateNaissanceAmi":  
        { "type": "literal",  
          "datatype": "http://www.w3.org/2001/XMLSchema#date",  
          "value": "1997/11/11" }  
      },  
      {  
        "ami": { "type":  
          "uri",  
          "value": "http://example.org/knowledgeGraph#Nami" }  
      },  
      {  
        "dateNaissanceAmi":  
        { "type": "literal",  
          "datatype": "http://www.w3.org/2001/XMLSchema#date",  
          "value": "1998/07/03" }  
      }  
    ]  
  }  
}
```

Tableau

ami	dateNaissanceAmi
kg:Zoro	1997/11/11 ^^xsd:date
kg:Nami	1998/07/03 ^^xsd:date



REQUÊTES UTILES – À VOUS !

- Tous les triplets
- Toutes les owl:Class
- Toutes les URI de classes instanciées



TOUS LES TRIPLETS

```
SELECT * WHERE {  
    ?sujet ?predicat ?objet  
}
```

* : toutes les variables
présentes dans le BGP

Tout peut être une variable,
Sujet, prédicat et/ou objet



TOUTES LES URI DE CLASSES INSTANCIÉES

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT * WHERE {
    ?x a owl:Class.
}
```

* : toutes les variables
présentes dans le BGP

Penser à mettre le préfixe owl

a : équivalent à `rdf:type` mais
pas besoin d'ajouter un
préfixe `owl`



TOUTES LES URI DE CLASSES INSTANCIÉES

```
SELECT DISTINCT ?y
WHERE {
    ?x a ?y.
}
```

DISTINCT : seulement des valeurs distinctes dans les résultats

Pourquoi utiliser DISTINCT ici ?

1. Appariement : récupération de **tous** les sous-graphes qui correspondent au BGP
(kg:Luffy, sch:Person) (kg:Zoro, sch:Person) (kg:Nami, sch:Person)
2. Récupération des valeurs prises par la variable ?y dans ces résultats
(sch:Person) (sch:Person) (sch:Person)
3. Si on ajoute DISTINCT : suppression des doublons dans les réponses
(sch:Person)



LITTÉRAUX ET FONCTIONS



LITTÉRAUX

- SPARQL supporte les types définis dans XML Schema:
 - xsd:integer
 - xsd:decimal
 - xsd:float
 - xsd:double
 - xsd:string
 - xsd:boolean
 - xsd:dateTime



TYPAGE DES LITTÉRAUX

- @prefix xsd: [<http://www.w3.org/2001/XMLSchema#>](http://www.w3.org/2001/XMLSchema#) .
- @prefix ex: [<http://example.org/>](http://example.org/) .
- ex:s1 ex:p "test" .
- ex:s2 ex:p "test"^^xsd:string .
- ex:s3 ex:p "test"@en .
- ex:s5 ex:p "test"^^http://example.org/datatype1 .
- ex:s4 ex:p "42"^^xsd:integer .

{ ?subject ex:p "test" . }



TYPAGE DES LITTÉRAUX

- @prefix xsd: [<http://www.w3.org/2001/XMLSchema#>](http://www.w3.org/2001/XMLSchema#).
- @prefix ex: [<http://example.org/>](http://example.org/).
- ex:s1 ex:p "test" .
- ex:s2 ex:p "test"^^xsd:string .
- ex:s3 ex:p "test"@en .
- ex:s5 ex:p "test"^^<<http://example.org/datatype1>> .
- ex:s4 ex:p "42"^^xsd:integer .

{ ?subject ex:p "test"@en . }



TYPAGE DES LITTÉRAUX

- @prefix xsd: [<http://www.w3.org/2001/XMLSchema#>](http://www.w3.org/2001/XMLSchema#).
- @prefix ex: [<http://example.org/>](http://example.org/).
- ex:s1 ex:p "test" .
- ex:s2 ex:p "test"^^xsd:string .
- ex:s3 ex:p "test"@en .
- ex:s5 ex:p "test"^^http://example.org/datatype1 .
- ex:s4 ex:p "42"^^xsd:integer .



```
{ ?subject ex:p 42 . }
```


TYPAGE DES LITTÉRAUX

- @prefix xsd: [<http://www.w3.org/2001/XMLSchema#>](http://www.w3.org/2001/XMLSchema#).
- @prefix ex: [<http://example.org/>](http://example.org/).
- ex:s1 ex:p "test" .
- ex:s2 ex:p "test"^^xsd:string .
- ex:s3 ex:p "test"@en .
- ex:s5 ex:p "test"^^[<http://example.org/datatype1>](http://example.org/datatype1).
- ex:s4 ex:p "42"^^xsd:integer .



```
{ ?subject ex:p "42" . }
```

FONCTIONS

- Utilisation d'opérateurs et de fonctions
- Dans **SELECT** avec **AS** == Projection
 - faire un calcul sur les résultats renvoyés
 - (/!\ opération faite avant le DISTINCT)
- Dans **FILTER**
 - Filtrer des résultats une fois que le graphe à trou a été apparié

```
SELECT ((?z*2) as ?e) WHERE {  
  ?x ?y ?z  
  FILTER(isNumeric(?z))  
}
```



OPÉRATEURS ET FONCTIONS

comparaison	=, <=, >=, !=
booléens	&&,
termes	isIRI(), isBlank(), isLiteral(), isNumeric(), str(), lang(), datatype(), strdt(), strlang()...
string	strlen(), substr(), ucase(), lcase(), strstarts(), strends(), contains(), regex(), encode_for_uri(), concat(), langMatches()...
nombres	abs, round, ceil, floor, rand, +, -, /, *
dates	year(), month(), day(), hours(), minutes(), seconds(), now(), timezone(), tz()
autres	bound(), if(), coalesce(), not exists(), sameTerm(), in()



Voir plus et usage sur

<https://www.w3.org/TR/2013/REC-sparql11-query-20130321/#expressions>

OPÉRATIONS SUR MOTIFS DE GRAPHE



MOTIFS DE GRAPHE

- Motifs de graphe élémentaires
 - Ensemble de triplets contenant ou non des variables
 - Délimités par des accolades { }
- Opérateur par défaut : **ET** logique

```
{  
    ?x sch:knows kg:elizabethII .  
    ?x sch:knows kg:Luffy .  
}
```

⇐ ?x connaît Luffy
ET ?x connaît Elisabeth II



UNION

- Mot-clé U
- OU logico

```
SELECT ?x  
{?x a ex:F  
UNION  
{?x a ex:l  
?x ex:lives  
ex:  
}
```

THE PATH FROM A TO B



ats de 2 BGP

ex:Person qui vivent
rStreet.

ge



OPTIONAL



- Mot-clé **OPTIONAL**: récupérer des valeurs si elles existent

```
SELECT ?x ?name WHERE {  
  ?x a ex:Person.  
  OPTIONAL {  
    ?x ex:name ?name.  
  }  
}
```

Les ex:Person et leur valeur de ex:name si cette valeur est renseignée.
Rien sinon.

x	name
kg:Zoro	Roronoa
kg:Nami	
kg:Luffy	Monkey D.



NÉGATION – MINUS

- 2 manières de faire la négation **MINUS** et **FILTER NOT EXISTS**
- **MINUS** Une fois l'appariement fait, il filtre les résultats pour enlever les occurrences du *sous-graphe apparié* pour filtrer ceux qui collent à la requête


```
SELECT ?x WHERE {  
  ?x a ex:Person.  
  MINUS {  
    ?x ex:name ?name.  
  }  
}
```

1. Récupère toutes les ex:Person
kg:Luffy kg:Zoro kg:Nami
2. Supprime des résultats ceux qui peuvent être appariés au BGP dans le MINUS
kg:Nami



NÉGATION – MINUS

- 2 manières de faire la négation **MINUS** et **FILTER NOT EXISTS**
- **MINUS** Une fois l'appariement fait, il filtre les résultats pour enlever les occurrences du *sous-graphe apparié* pour filtrer ceux qui collent à la requête



```
SELECT ?x WHERE {  
  ?x a ex:Person.  
  MINUS {  
    ?s ex:name ?name.  
  }  
}
```

1. Récupère toutes les ex:Person, binding pour ?x :

kg:Luffy kg:Zoro kg:Nami

2. Aucun binding pour ?s dans les sous-graphes appariés, aucun résultat n'est filtré

kg:Luffy kg:Zoro kg:Nami

NÉGATION – FILTER NOT EXISTS

- **FILTER NOT EXISTS** teste si un BGP peut être apparié au jeu de données, en se basant sur les bindings du BGP principal s'il le peut

```
SELECT ?x WHERE {  
  ?x a ex:Person.  
  FILTER NOT EXISTS {  
    ?x ex:name ?name.  
  }  
}
```

1. Récupère toutes les ex:Person, bindings pour ?x :

kg:Luffy kg:Zoro kg:Nami

2. Filtre des résultats si le triplet ?x ex:name existe dans le dataset en prenant les bindings précédents

reste *kg:Nami*



NÉGATION – FILTER NOT EXISTS

- **FILTER NOT EXISTS** teste si un BGP peut être apparié au *jeu de données*, en se basant sur les bindings du BGP principal s'il le peut

```
SELECT ?x WHERE {  
  ?x a ex:Person.  
  FILTER NOT EXISTS {  
    ?s ex:name ?name.  
  }  
}
```

1. Récupère toutes les ex:Person, bindings pour ?x :

kg:Luffy kg:Zoro kg:Nami

2. Filtre des résultats si le triplet ?s ex:name existe, pas de binding sur ?s. Le triplet ?s ex:name ?name existe dans le dataset, on supprime tout.

reste *rien*



SOUS-REQUÊTES

- Une requête = un BGP
- Possibilité d'imbriquer des requêtes

```
SELECT ?x ?cat WHERE {  
  ?x a ex:Person.  
  {  
    SELECT ?x ?cat WHERE {  
      ?x ex:pets ?cat.  
    }  
  }  
}
```



GROUPES ET AGRÉGATS



GROUPES ET AGRÉGATS

- Faire un **groupe** ou **agrégat** c'est regrouper les résultats sur une ou plusieurs variables
 - Le nombre d'amis PAR personne
 - La note maximale PAR matière et PAR année
- **GROUP BY** permet de regrouper des résultats sur une ou plusieurs variables présentes dans le **SELECT**
- **HAVING** permet de filtrer les groupes



OPÉRATEURS D'AGRÉGAT

COUNT(?v)	permet d'obtenir le nombre fois que v est affectée dans l'ensemble
SUM(?v)	somme des valeurs prises par v
MIN(?v)	plus petite des valeurs prises par v
MAX(?v)	plus grande des valeurs prises par v
AVG(?v)	moyenne des valeurs prises par v
SAMPLE(?v)	retourne au hasard une des valeurs prises par la variable v
GROUP_CONCAT(?v; separator = " ")	Concaténation des string des valeurs prises par v



EXEMPLES DE GROUP BY

Le nombre d'amis pour chaque personne

PREFIX sch: <<https://schema.org/>>

```
SELECT ?person COUNT(?ami)
WHERE {
  ?person sch:knows ?ami.
}
GROUP BY ?person
```



EXEMPLES DE GROUP BY

- ⚠ Toute variable dans le SELECT doit soit :
- apparaître dans le GROUP BY
 - être dans une fonction groupante

PREFIX sch: <https://schema.org/>

```
SELECT ?person ?name COUNT(?  
ami) WHERE {  
  ?person sch:knows ?ami.  
  ?person sch:name ?name.  
}  
GROUP BY ?person ?name
```

PREFIX sch: <https://schema.org/>

```
SELECT ?person GROUP_CONCAT(?  
name) COUNT(?ami)  
WHERE {  
  ?person sch:knows ?ami.  
  ?person sch:name ?name.  
}  
GROUP BY ?person
```



EXEMPLES DE GROUP BY



- ⚠ Toute variable dans le SELECT qui n'est pas groupée doit soit :
- apparaître dans le GROUP BY
 - être dans une fonction groupante

PREFIX sch: <https://schema.org/>

```
SELECT ?person ?name COUNT(?  
ami) WHERE {  
  ?person sch:knows ?ami.  
  ?person sch:name ?  
}          name.  
GROUP BY ?person ?  
          name
```

person	name	countAmi
kg:Luffy	Monkey D.	2
kg:Luffy	Luffy	2

PREFIX sch: <https://schema.org/>

```
SELECT ?person GROUP_CONCAT(?  
name) COUNT(?ami)  
WHERE {  
  ?person sch:knows ?ami.  
  ?person sch:name ?name.  
}  
GROUP BY ?person
```

person	groupName	countAmi
kg:Luffy	Monkey D. Luffy	2

HAVING



- Comme **FILTER** mais pour des valeurs groupées
- S'utilise hors du BGP

PREFIX sch: <<https://schema.org/>>

```
SELECT ?person
  WHERE {
    ?person sch:knows ?ami.
  }
GROUP BY ?person
HAVING COUNT(?ami) > 2
```

MODIFICATEURS ET FORMES DE RÉSULTATS



MODIFICATEURS

- ORDER BY + DESC
- LIMIT
- OFFSET
- DISTINCT

PREFIX sch: <<https://schema.org/>>

```
SELECT DISTINCT ?name
WHERE {
  ?person sch:name ?name.
}
ORDER BY DESC(?
name) LIMIT 1
OFFSET 1
```



FORMATS DE RÉSULTATS

- **SELECT**
 - Récupération de bindings variable / valeur
- **ASK**
 - Vrai/Faux
- **CONSTRUCT**
 - Créer des triplets RDF à partir des bindings du BGP
- **DESCRIBE <uri>**
 - Récupérer les triplets dont la <uri> est le sujet (et/ou objet)



ASK

Réponse : booléen

- **VRAI** si le BGP apparaît au moins 1 fois dans le dataset
- **FAUX** sinon

PREFIX sch: <<https://schema.org/>>

```
ASK{  
  kg:Luffy sch:name ?name.  
}
```



CONSTRUCT



Réponse : Triplets en RDF

- Créés à partir des bindings du BGP
- Résultat sérialisé dans un format de sérialisation du RDF (JSON-LD, TTL, N3, etc.)

PREFIX sch: <https://schema.org/>

```
CONSTRUCT{  
  ?x sch:knows ?y.  
}  
WHERE {  
  ?y a ex:Person.  
  ?x sch:name "Mister Universe".  
}
```

```
@prefix ex: <http://example.org/> .  
@prefix sch: <https://schema.org/> .
```

```
ex:Mr_Universe sch:knows ex:elizabethII.  
ex:Mr_Universe sch:knows ex:Luffy.
```

...

SERVICE



ONE RING TO RULE
THEM ALL

DÉCENTRALISATION !

PREFIX sch: <<https://schema.org/>>

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

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

SELECT ?x ?nameJap **WHERE** {

 ?x a sch:Person.

 ?x owl:sameAs ?xwiki.

SERVICE <<https://query.wikidata.org/sparql>> {

 ?xwiki rdfs:label ?nameJap.

 FILTER(langMatches(lang(?nameJap), "ja"))

}



SPARQL UPDATE



GRAPH BEFORE

GRAPH AFTER



SPARQL UPDATE

MISE À JOUR DE DONNÉES EN SPARQL

- **INSERT DATA { ... } ou DELETE DATA { ... }**
 - Pas de variables dans le BGP, que des données complètes
- **INSERT { ... } WHERE { ... } ou DELETE { ... } WHERE { ... }**
 - Ajout/suppression conditionnel de données basé sur le BGP
 - Utilisation de variables, binding issu du BGP
 - Comme CONSTRUCT
- **DELETE { ... } INSERT { ... } WHERE { ... }**
 - Comme au-dessus mais fait suppression et ajout en 1 fois



EXAMPLES



```
PREFIX ex: <http://example.org/> .  
PREFIX sch: <https://schema.org/> .
```

```
INSERT DATA {  
  ex:Mr_Universe sch:knows ex:elizabethII.  
  ex:Mr_Universe sch:knows ex:Luffy.  
}
```

```
PREFIX ex: <http://example.org/> .  
PREFIX sch: <https://schema.org/> .
```

```
DELETE DATA {  
  ex:Mr_Universe sch:children ex:elizabethII.  
}
```

EXAMPLES



PREFIX ex: <<http://example.org/>> .
PREFIX sch: <<https://schema.org/>> .

```
INSERT {  
  ?x a ex:FrenchPerson.  
}  
WHERE {  
  ?x sch:nationality ex:FrenchNationality.  
}
```

```
DELETE {  
  ?x ?y ?z.  
}  
WHERE {  
  ?x ?y ?z.  
}
```

EXAMPLES



PREFIX sch: <<https://schema.org/>>

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

```
DELETE{
  ?x sch:knows ?y
}
INSERT{
  ?x foaf:knows ?y
}
WHERE {
  ?x sch:knows ?y
}
```

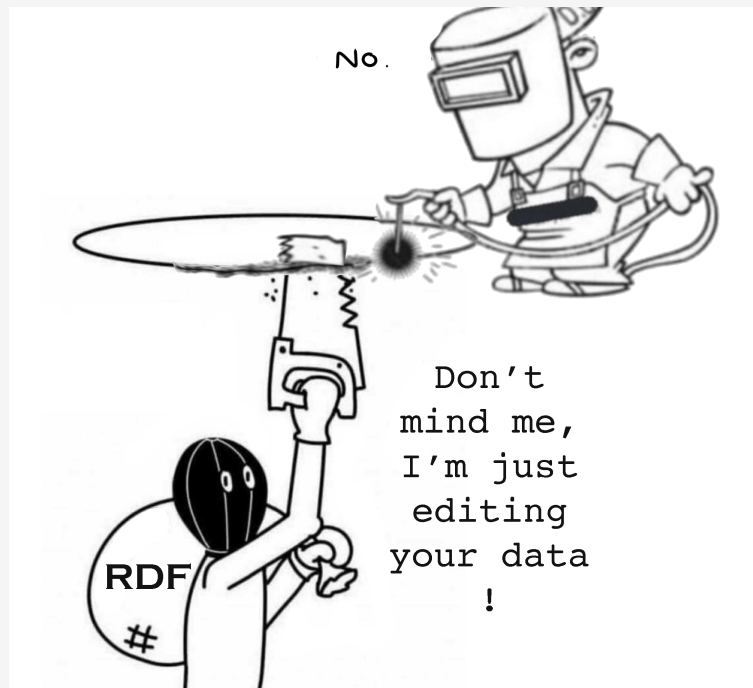
PROBLÈME !



Il ne faudrait pas permettre aux utilisateurs du dépôt SPARQL d'éditer nos données !

En pratique, pour un même entrepôt SPARQL, on définit deux accès

- Query
- Update





EN RÉSUMÉ

- SPARQL est un langage permettant d'interroger des graphes de connaissances afin d'en extraire les informations pertinentes
- Il repose sur un principe de graphes à trous exprimés sous forme de triplets Turtle (ou similaires)
- Les mots clés ressemblent à ceux de SQL, mais les modalités de matching sont tout autres
- Avec SPARQL Update : possibilité de mettre à jour massivement le graphe de connaissance.