RDF(S) - SPARQL - Stockage

E.Coquery

emmanuel.coquery@univ-lyon1.fr

http://liris.cnrs.fr/~ecoquery
→ Enseignement → MIF04 GDW

Ontologies, RDF, SPARQL, RDF-S

- Ontologie
  - ensemble de connaissances
  - formalisées dans un ou plusieurs langages
    - RDF(S), OWL, ...
- [RDF] : format de données
  - graphes
  - annotés par des IRI et des valeurs
- [SPARQL] : langage d'interrogation pour RDF
- RDF *store* : BD native RDF
- [RDF-S] : schéma non restrictif
  - Permet d'enrichir des graphes RDF

Web sémantique et Linked Open Data

[Web sémantique]

- extension du Web
- liens sémantiques entre les ressources
  - relation ayant un sens défini
  - permettant un compréhension par un humain
  - et un traitement par une machine

[Linked Open Data] : partie du Web sémantique

- Ensemble de ressources Web
- librement accessibles
- ayant des URI *déréférençables*
- contenant des liens vers d'autres ressources

[LOD Cloud view]

Linking Open Data cloud diagram 2014, by Max Schmachtenberg, Christian Bizer, Anja Jentzsch and Richard Cyganiak. `http://lod-cloud.net/`

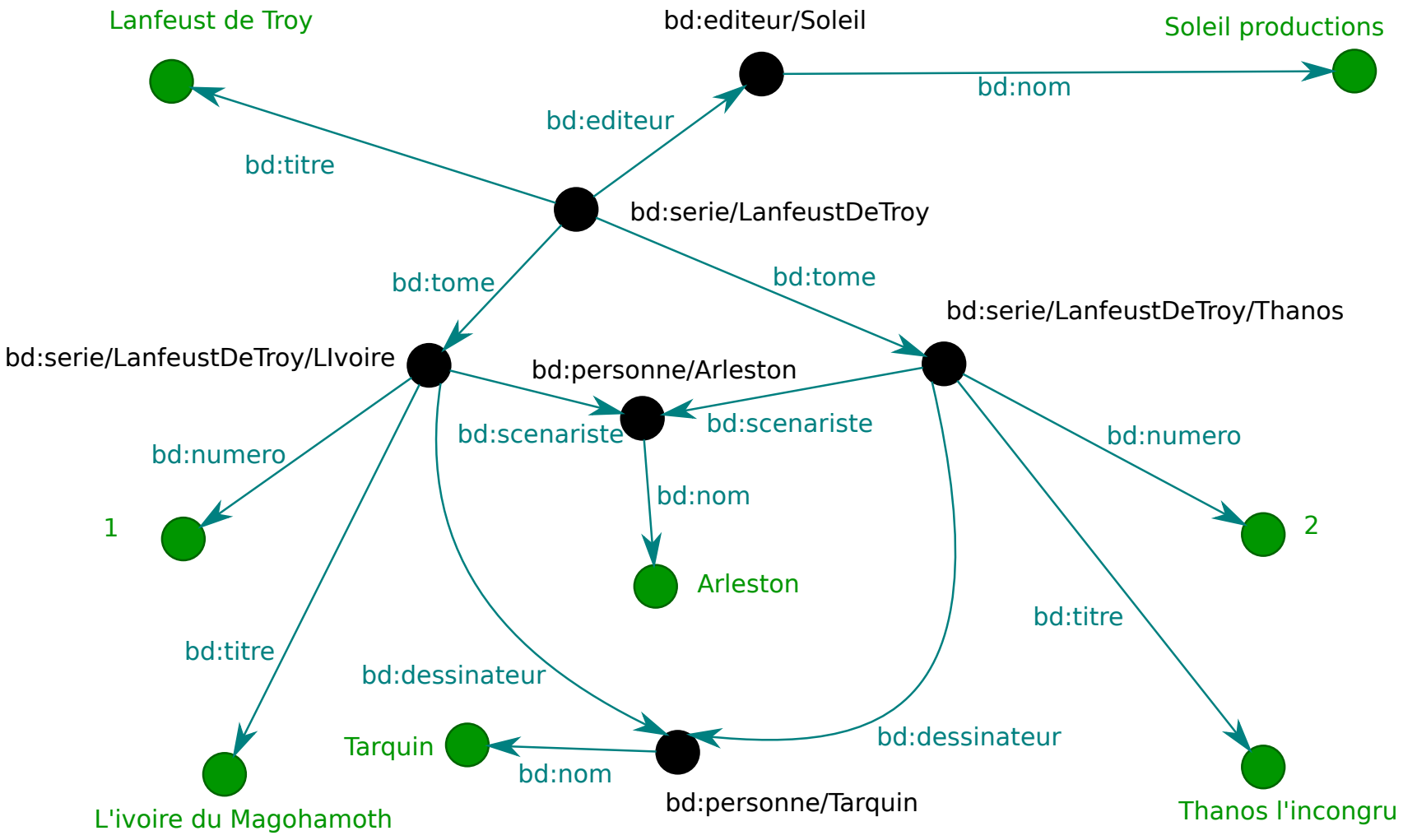
Graphes étiquetés

- Graphes orientés
  - Les sommets et les arêtes sont étiquetés
- Constitue un modèle de données alternatifs aux modèles :
  - Relationnel, semi-structuré, objet
- Permet de représenter aisément des liens entre des choses référencées par un identifiant :
  - Sommet : *chose*
  - Arête : relation entre deux *choses*

RDF : graphes pour le Web sémantique

- Standard du W3C
- Graphes RDF
  - Étiquetés (arêtes et sommets)
    - par des [IRIs] (ressources)
    - par des littéraux (valeurs), uniquement pour les sommets
    - au plus 1 sommet / étiquette
    - pas de max pour le nombre d'arêtes / étiquette
  - l'IRI est symbolique
    - IRIs déréférençables dans le cadre du *Linked Data*

RDF : exemple



bd :↔http://www.collection.com/bd

Triplet RDF

Description de graphe par des triplets représentant les arêtes

- Sujet
  - Étiquette du sommet de départ
- Prédicat (ou property)
  - Étiquette de l'arête
- Objet
  - Étiquette du sommet d'arrivée

Exemple :

`(bd:serie/LanfeustDeTroy, bd:editeur, bd:editeur/Soleil)`

Sérialisation : XML

Syntaxe pour représenter des triplets

Élément `rdf:Description`

- Déclaration de triplets ayant pour sujet l'IRI indiquée par l'attribut `rdf:about`
- Attributs/éléments :
  - Espaces de nommage + nom local = IRI du prédicat
- Valeur/attribut `rdf:resource`
  - Objet
  - Littéral/IRI
- Pour les littéraux : `rdf:datatype`

Exemple

```
<?xml version='1.0' encoding='UTF-8'?>
<rdf:RDF
     xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
     xmlns:bd='http://www.collection.com/bd/'>

  <rdf:Description rdf:about='http://www.collection.com/bd/serie/LanfeustDeTroy'>
    <bd:tome rdf:resource='http://www.collection.com/bd/serie/LaufeustDeTroy/LIvoire'/>
    <bd:tome rdf:resource='http://www.collection.com/bd/serie/LaufeustDeTroy/Thanos'/>
    <rdf:type rdf:resource='http://www.collection.com/bd/serie'/>
  </rdf:Description>

  <rdf:Description rdf:about='http://www.collection.com/bd/editeur/Soleil'>
    <bd:nom>Soleil Productions</bd:nom>
  </rdf:Description>

  <rdf:Description
       rdf:about='http://www.collection.com/bd/serie/LaufeustDeTroy/LIvoire'>
    <bd:numero rdf:datatype='http://www.w3.org/2001/XMLSchema#int'>1</bd:numero>
  </rdf:Description>
...
</rdf:RDF>
```

Sérialisation : TURTLE

Syntaxe alternative pour RDF

- IRI :
  - `<http://www.collection.com/bd/serie>`
  - bd :serie
    - `PREFIX bd: <http://www.collection.com/bd/>`
- Valeur :
  - `'Arleston'`
  - `"2.5"^^xsd:float`
- Triplet :
  - sujet predicat objet .
  - sujet predicat objet ; predicat objet .
  - sujet predicat objet, objet ; predicat objet, objet.

Exemple

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bd: <http://www.collection.com/bd/>

<http://www.collection.com/bd/serie/LanfeustDeTroy> rdf:type
      <http://www.collection.com/bd/serie> .
<http://www.collection.com/bd/serie/LanfeustDeTroy> bd:editeur
      <http://www.collection.com/bd/editeur/Soleil Productions> .

<http://www.collection.com/bd/editeur/Soleil Productions> bd:nom "Soleil Productions .

<http://www.collection.com/bd/serie/LaufeustDeTroy/LIvoire>
    bd:numero "1"^^xsd:integer ;
    bd:titre "L'ivoire du Magohamoth" ;
    bd:dessinateur <http://www.collection.com/personne/Tarquin> ;
    bd:scenariste <http://www.collection.com/personne/Arleston"> .


<http://www.collection.com/bd/serie/LanfeustDeTroy> bd:tome
      <http://www.collection.com/bd/serie/LaufeustDeTroy/LIvoire> ,
      <http://www.collection.com/bd/serie/LaufeustDeTroy/Thanos 1 incongru> .
```

Sérialisation : JSON-LD

[JSON-LD] : Représentation d'un graphe RDF en JSON

- JSON Object ↔ nœud du graphe RDF
- clés = IRI de prédicat
  - + des clés spéciales (e.g. `@id`, `@type`, `@context`)
- valeurs : objets (= nœuds IRI) ou valeur (= littéraux)

Exemple

```
{
  "@context": {
    "numero": {
      "@id":"http://www.collection.com/bd/numero",
      "@type": "http://www.w3.org/2001/XMLSchema/integer"
    }
  },
  "@id":"http://www.collection.com/bd/serie/LaufeustDeTroy/LIvoire",
  "numero":"1",
  "http://www.collection.com/bd/titre":"L'ivoire du Magohamoth",
  "http://www.collection.com/bd/dessinateur": {
    "@id":"http://www.collection.com/personne/Tarquin"
  },
  "http://www.collection.com/bd/scenariste": {
    "@id":"http://www.collection.com/personne/Arleston"
  }
}
```

Nœuds anonymes (blank nodes)

- Pas des IRIs, ni des littéraux
- Peuvent être utilisés comme des nœuds IRI
- *Intuitivement*, deux nœuds anonymes peuvent être replacés par un même noeud

Types de données

- Un type T :
  - Espace lexical EL : ensemble de chaînes de caractères
    - c.f. types simples XML
  - Espace (ensemble) de valeurs EV
  - Fonction $L2V(T) : EL \rightarrow EV$

- Exemple : T = Entiers XML Schema
  - EL : chaînes reconnues par `-?[0-9]+`
  - EV : $\mathcal{Z}$
  - $L2V(T)$: parsing des entiers

Requêtes

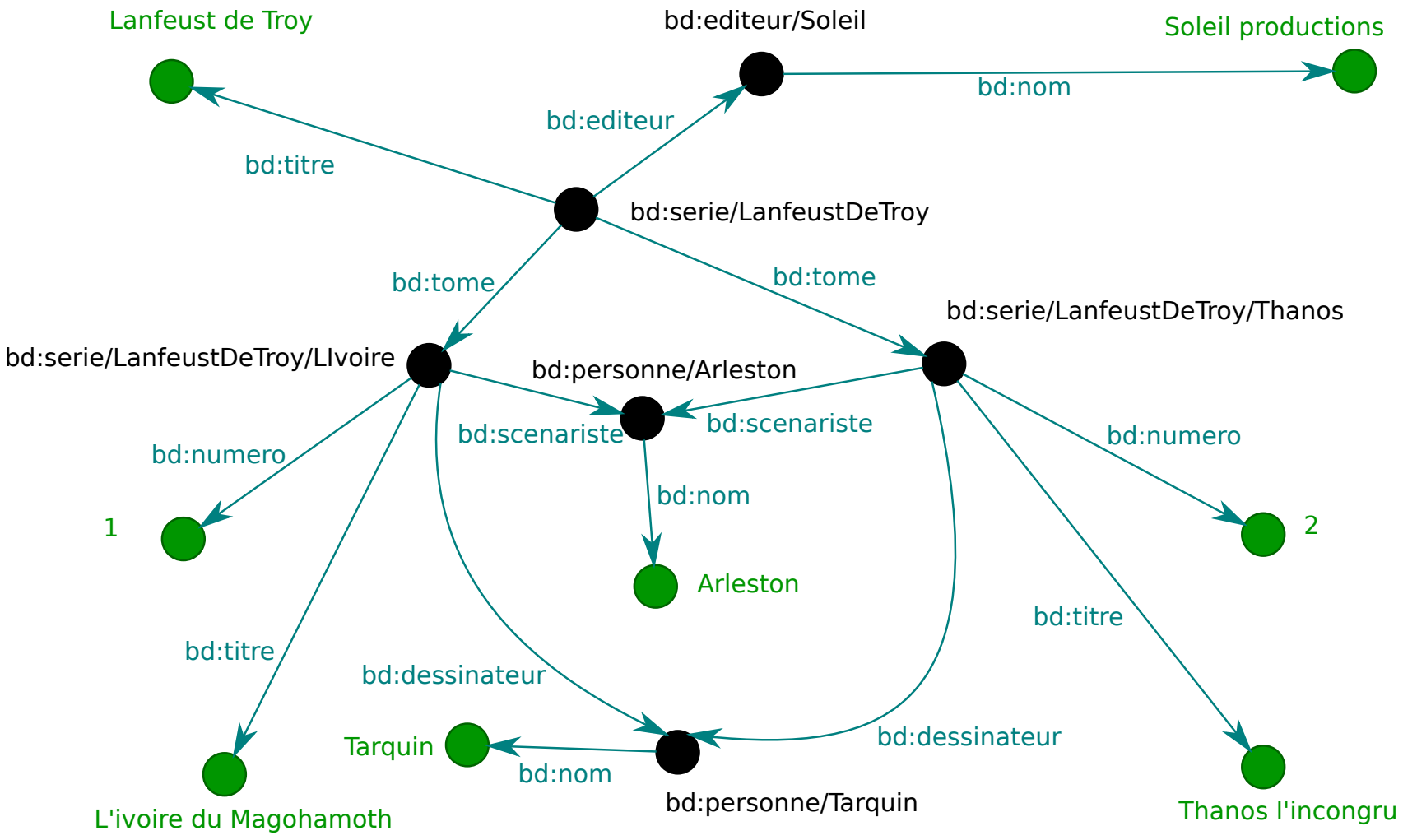
Comment requêter un graphe :

- comment identifier les endroits cherchés ?
- quelles sont les parties intéressantes ?
- comment agréger de l'information ?
- comment faire des recoupements ?

Matching de sous-graphe

- Spécifier des contraintes sur les parties du graphe à récupérer
  - *pattern matching* pour les graphes
- Pattern =
  - Un graphe exemple
  - Avec des variables
- Réponses
  - Sous-graphes du graphe de départ
  - Correspondant au pattern
    - En instanciant les variables
- Une réponse explique comment le graphe requêté implique le graphe de départ en supposant que les trous sont des nœuds anonymes

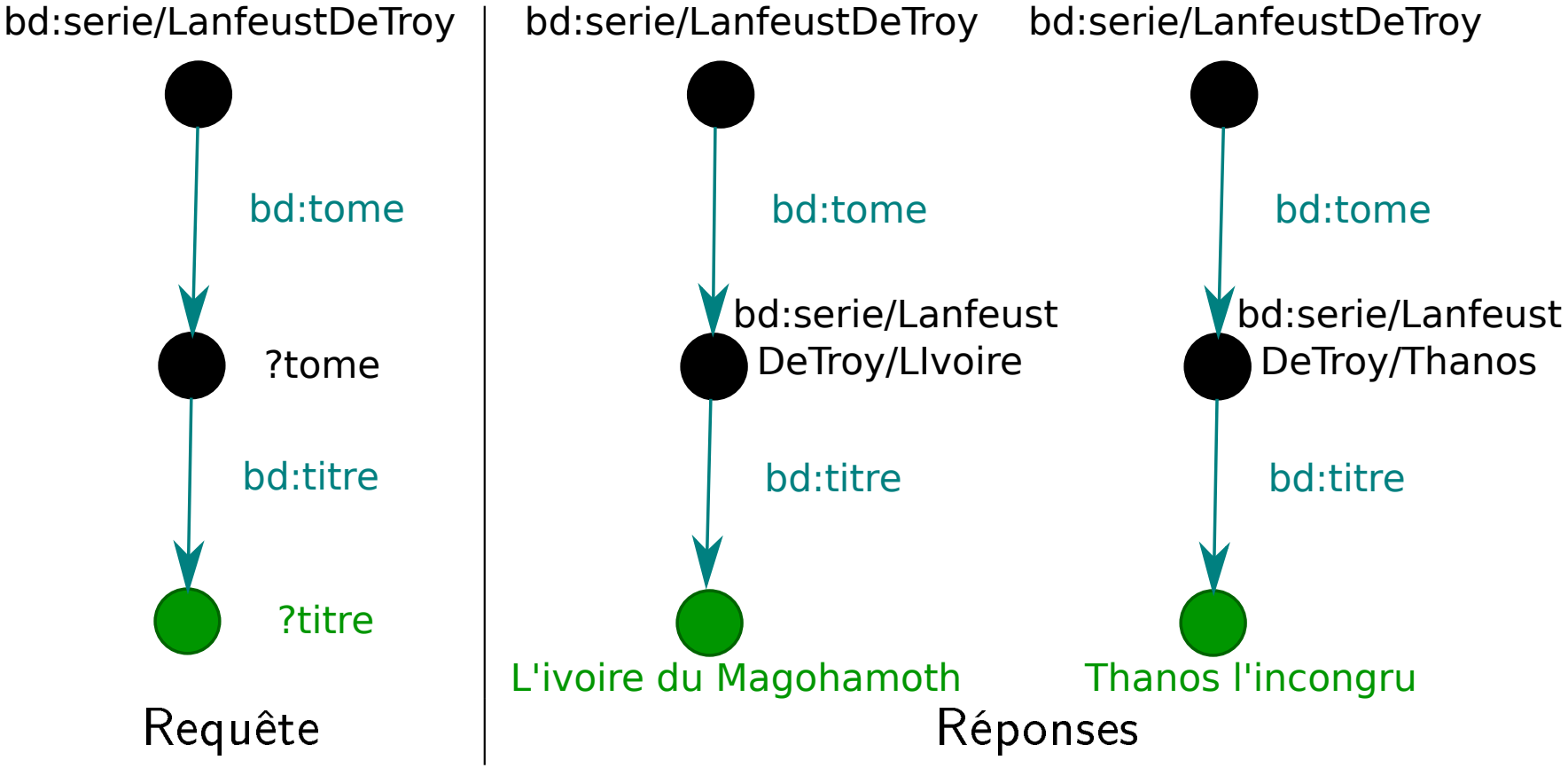
Exemple : graphe



bd :↔http://www.collection.com/bd

Exemple : requête et réponses

Quels sont les tomes de la série `bd:serie/LanfeustDeTroy`, avec leur titre ?



SPARQL

- Langage de requête pour les graphes RDF
  - Basé sur le matching de sous-graphe
  - Standard W3C
- Syntaxe des triplets basée sur TURTLE
  - Variables : `?nomVar`
  - Toute IRI/Valeur peut être remplacée par une variable
    - y compris les prédicats
- Réponse sous forme
  - D'affectation des variables du pattern (`SELECT`)
  - De graphe (`CONSTRUCT`)

Exemple

```
PREFIX bd: <http://www.collection.com/bd#>

SELECT * WHERE {
  ?t bd:dessinateur ?p .
}
```

Donner les ?t et ?p tels que ?p est le dessinateur de ?t.

Résultat (sérialisé en XML)

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="t"/>
    <variable name="p"/>
  </head>
  <results>
    <result>
      <binding name="t">
        <uri>http://www.collection.com/bd/serie/LaufeustDeTroy/Thanos</uri>
      </binding>
      <binding name="p">
        <uri>http://www.collection.com/personne/Tarquin</uri></binding>
    </result>
    <result>
      <binding name="t">
        <uri>http://www.collection.com/bd/LaufeustDeTroy/LIvoire</uri></binding
      <binding name="p"><uri>http://www.collection.com/personne/Tarquin</uri>
      </binding>
    </result>
  </results>
</sparql>
```

Exemple avec une variable sur un prédicat

```
PREFIX bd: <http://www.collection.com/bd#>

SELECT * WHERE {
   <http://www.collection.com/bd/Lanfeust de Troy> ?p ?v .
}
```

Nœuds anonymes

- Nœud non étiqueté
  - Dans le graphe requêté
- Dont l'étiquette n'est pas importante
  - dans le motif
- En TURTLE : []

Les ?t ayant un dessinateur :

```
PREFIX bd: <http://www.collection.com/bd#>

SELECT * WHERE {
  ?t bd:dessinateur []
}
```

Patterns plus complexes : et / ou

Opérateur ET : implicite

```
PREFIX bd: <http://www.collection.com/bd#>

SELECT * WHERE {
  ?t bd:titre ?ti .
  ?t bd:dessinateur [] .
}
```

Opérateur OU : UNION

```
PREFIX bd: <http://www.collection.com/bd#>

SELECT * WHERE {
  { ?t bd:scenariste ?p .}
UNION
  { ?t bd:dessinateur ?p .}
}
```

Filtres

Complémentaire au WHERE

- pas directement du matching
- !, &&, ||, =, !=, >, +, -, etc

```
PREFIX bd: <http://www.collection.com/bd#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT * WHERE {
   ?t bd:numero ?n .
   FILTER(?n > "1"^^xsd:integer).
}
```

⚠ syntaxe des littéraux typés ⚠

Requêter plusieurs graphes

Combiner les information provenant de plusieurs graphes FROM permet de spécifier le graphe requêté

- IRI simple
- `NAMED` : permet de faire référence dans le WHERE au graphe indiqué
- `GRAPH` : permet de spécifier un pattern à chercher dans un autre graphe
  - Le graphe peut être identifié par une variable

Exemple

```
PREFIX bd: <http://www.collection.com/bd#>

SELECT *
FROM <http://www.collection.com/bd#g1>
FROM NAMED bd:g2
WHERE {
  ?t bd:titre ?ti .
  GRAPH bd:g2 {?t bd:titre ?ti2 .}
}
```

Projection, calcul de valeur

Fonctionnement du SELECT similaire à celui de SQL :

- Possibilités de renommage
- Calculs de nouvelles valeurs

```
PREFIX bd: <http://www.collection.com/bd#>
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>

SELECT (fn:concat(?ti," par ",?scn," et ",?den) as ?album)
 WHERE {
    ?t bd:titre ?ti .
    ?t bd:scenariste ?sc .
    ?sc bd:nom ?scn .
    ?t bd:dessinateur ?de .
    ?de bd:nom ?den .
}
```

Agrégation

GROUP BY + fonctions d'agrégation

- COUNT, SUM, MAX, etc

```
PREFIX bd: <http://www.collection.com/bd#>

SELECT ?serie (COUNT(?to) as ?nbTomes) WHERE {
    ?serie bd:tome ?to .
}
GROUP BY ?serie
```

Correspondance partielle

Rendre certaines parties du motif non obligatoires : `OPTIONAL`

```
PREFIX bd: <http://www.collection.com/bd#>

SELECT * WHERE {
    ?serie bd:tome ?to .
    OPTIONAL {?serie bd:editor ?ed .}
}
```

Négation

Principe : pouvoir qu'il n'existe pas de correspondance pour un certain motif

- `OPTIONAL` + `FILTER(!bound(`*?var*`))` (SPARQL 1.0)
- `FILTER(NOT EXISTS {` *pattern* `})` (SPARQL 1.1)
- `{` *pattern* `} MINUS {` *pattern* `}` (SPARQL 1.1)
  - Sémantique peu intuitive : à éviter

Sur le fond : problématique car monde ouvert

Exemple !bound(...)

Tomes de séries dont on n'a pas l'éditeur

```
PREFIX bd: <http://www.collection.com/bd#>

SELECT ?to WHERE {
    ?serie bd:tome ?to .
    OPTIONAL {?serie bd:editor ?ed}
    FILTER(!bound(?ed)).
}
```

Exemple NOT EXISTS

Tomes de séries dont on n'a pas l'éditeur

```
PREFIX bd: <http://www.collection.com/bd#>

SELECT ?to WHERE {
    ?serie bd:tome ?to .
    FILTER(NOT EXISTS {?serie bd:editor []}).
}
```

Créer des graphes résultats

CONSTRUCT à la place de SELECT

```
PREFIX bd: <http://www.collection.com/bd#>
CONSTRUCT {
  ?serie bd:album ?ti .
  ?serie bd:numero ?n .
} WHERE {
  ?serie bd:tome ?to .
  ?to bd:numero ?n .
  ?to bd:titre ?ti .
}
```

Autres mots clés

- DISTINCT
  - Comme en SQL
- LIMIT, OFFSET
  - Nb réponses, quelles réponses
- ASK (à la place de SELECT)
  - true/false

Mise à jour

- `INSERT DATA { triples }`
- `INSERT { template } WHERE { pattern }`
- `DELETE DATA { triples }`
- `DELETE { template } WHERE { pattern }`
- `LOAD uri`
- `LOAD uri INTO GRAPH uri`
- `CLEAR GRAPH uri`
- `CREATE GRAPH uri`
- `DROP GRAPH uri`

Vocabulaires RDF

- Ensemble d'IRI
- Pouvant être utilisés par une application
- Ayant souvent une définition à minima informelle
- Souvent rattaché à un espace de nommage (en quelque sorte)

Exemple [FOAF] :

- `foaf:firstName` (prénom)
- `foaf:knows` (connaît)

Un vocabulaire particulier : RDF

Préfixe `rdf` : `http://www.w3.org/1999/02/22-rdf-syntax-ns#`

- Typage :
  `rdf:type rdf:langString rdf:Property`
- Réification :
  `rdf:subject rdf:predicate rdf:object`
- Listes :
  `rdf:first rdf:rest rdf:value rdf:nil rdf:List`
- Conteneurs : `rdf:_1 rdf:_2`

RDFS, OWL, ...

Règles/axiomes logiques permettant :

- De déduire des triplets additionnels
- D'ajouter des contraintes d'intégrité
  - Seulement sur les types de données en RDFS

Exemple : tous les tomes de série sont des livres

RDFS, OWL : vus comme des schémas

RDF Schema

- Classification des ressources
- Contraintes d'intégrité simples
  - Sur les types primitifs

OWL

- Logique plus riche
- Contraintes d'intégrité plus complexes

RDF Schema (RDFS)

- Système de classes de ressources
  - Avec système de sous-classes
- Description des prédicats
  - Quel sujet, quel objet ?
- Interprétation spécifiques → système d'inférence
  - Déduction de nouveaux faits
    Comment les prendre en compte ?

`rdfs:` → `http://www.w3.org/2000/01/rdf-schema#`
`rdf:` → `http://www.w3.org/1999/02/22-rdf-syntax-ns#`

Vocabulaire RDFS

- Typage des propriétés :
  - `rdfs:domain rdfs:range`
- Types de base
  - `rdfs:Resource rdfs:Literal rdfs:Datatype rdfs:Class`
- Relations entre types et propriétés
  - `rdfs:subClassOf rdfs:subPropertyOf`
- Conteneurs
  - `rdfs:member rdfs:Container rdfs:ContainerMembershipProperty`
- Divers
  - `rdfs:comment rdfs:seeAlso rdfs:isDefinedBy rdfs:label`

Inférence RDFS

Certains triplets peuvent être considérés comme existants *implicitement* :

- si A est sous-classe de B
- et B est sous-classe de C
- on considère *implicitement* que A est sous-classe de C

Inférence : rendre explicites les triplets implicites

Graphe **saturé** : contient toutes les inférences possibles

Classes et instances

- Classe ↔ ensemble de ressources
- Ressource $R \in$ classe C :
  `R rdf:type C`
- Une ressource peut appartenir à plusieurs classes

- Tout ce qui contient une ressource est une classe
  on a la règle d'inférence suivante :

$$\frac{R \texttt{ rdf:type } C}{C \texttt{ rdf:type rdfs:Class}}$$

Type de données des littéraux

- Les types de littéraux L sont de type `rdfs:Datatype`
- Les types de littéraux L sont des sous-classes de `rdfs:Literal`
- Idéalement, les types des littéraux doivent être respectés par leur interprétation
  - Seule vérification de type de RDFS
  - Une date ne peut pas être un entier
  - Norme un peu plus subtile

rdfs:domain

- Domaine au sens domaine d'une fonction
- Fixe le type T des sujets d'un prédicat P
  P `rdfs:domain` T
- Inférence

$$\frac{P \texttt{ rdfs:domain } T \quad \text{et} \quad S\ P\ O}{S \texttt{ rdf:type } T}$$

rdfs:range

- Dual de `rdfs:domain`
- Fixe le type T des objets pour une propriété P : P `rdfs:range` T
- Inférence

$$\frac{P \texttt{ rdfs:range } T \quad \text{et} \quad S \; P \; O}{O \texttt{ rdf:type } T}$$

Sous-classes

- Classe $C \subseteq$ une classe D :
  C `rdfs:subClassOf` D
- Exemple :
  `univ:admin rdfs:subClassOf univ:user`

- Inférence :

$$\frac{C \texttt{ rdfs:subClassOf } D \quad \text{et} \quad E \texttt{ rdf:type } C}{E \texttt{ rdf:type } D}$$

Sous-prédicats

- Prédicats P cas particulier d'un prédicat Q :
  P `rdfs:subPropertyOf` Q
- Exemple :
  ```
  univ:enseigne rdfs:subPropertyOf univ:participeA
  univ:inscritA rdfs:subPropertyOf univ:participeA
  ```
- Inférence :

$$\frac{P \texttt{ rdfs:subPropertyOf } Q \quad \text{et} \quad S\ P\ O}{S\ Q\ O}$$

Transitivité de subClassOf et subPropertyOf

Pour rdfs:subClassOf :

$$\frac{C \ \texttt{rdfs:subClassOf} \ D \quad \text{et} \quad D \ \texttt{rdfs:subClassOf} \ E}{C \ \texttt{rdfs:subClassOf} \ E}$$

Pour rdfs:subPropertyOf :

$$\frac{P \ \texttt{rdfs:subPropertyOf} \ Q \quad \text{et} \quad Q \ \texttt{rdfs:subPropertyOf} \ R}{P \ \texttt{rdfs:subPropertyOf} \ R}$$

Transitivité de subClassOf et subPropertyOf

Pour rdfs:subClassOf :

$$\frac{C \ \texttt{rdfs:subClassOf} \ D \quad \text{et} \quad D \ \texttt{rdfs:subClassOf} \ E}{C \ \texttt{rdfs:subClassOf} \ E}$$

Pour rdfs:subPropertyOf :

$$\frac{P \ \texttt{rdfs:subPropertyOf} \ Q \quad \text{et} \quad Q \ \texttt{rdfs:subPropertyOf} \ R}{P \ \texttt{rdfs:subPropertyOf} \ R}$$

Autres règles

Il existe d'autres règles RDFS, voir :

`http://www.w3.org/TR/rdf11-mt/#entailment-rules-informative`

Par ailleurs, rien n'empêche d'ajouter d'autre règles sur le même principe, plus orientées métier, par exemple :

$$\frac{S_1 \ \texttt{metro:suivante} \ S_2}{S_1 \ \texttt{metro:connecte} \ S_2}$$

$$\frac{S_1 \ \texttt{metro:connecte} \ S_2 \quad \text{et} \quad S_2 \ \texttt{metro:connecte} \ S_3}{S_1 \ \texttt{metro:connecte} \ S_3}$$

Rmq : OWL2 RL peut être pris traité avec de telles règles

Prise en compte des règles de déduction

Deux type de techniques

- Chaînage avant (*forward chaining*)
  - Faire toutes le déductions possibles et ajouter les triplets dans le store
- Chaînage arrière (*backward chaining*)
  - Lors du traitement d'une requête interroger les triplets qui correspondent à la requête + ceux qui permettent de générer des triplets correspondant à la requête

Chaînage avant

Pour chaque règle :

- chercher des triplet correspondant aux prémisses
  - ≡ requête SPARQL
- Pour chaque correspondance
  - ajouter le triplet conclusion correspondant
    - seulement s'il n'existe pas déjà

Il faut itérer la construction

- les nouveaux triplets peuvent à leur tour correspondre à des prémisses

Exemple

```
:toto :inscrit :gdw
:inscrit rdfs:domain :etu
:etu rdfs:subClassOf :mem
:mem rdfs:subClassOf :pers
```

→

Exemple

```
:toto :inscrit :gdw
:inscrit rdfs:domain :etu
:etu rdfs:subClassOf :mem
:mem rdfs:subClassOf :pers
```

→ `:toto rdf:type :etu`

$$\frac{P \texttt{ rdfs:domain } T \quad \text{et} \quad S\ P\ O}{S \texttt{ rdf:type } T}$$

Exemple

```
:toto :inscrit :gdw
:inscrit rdfs:domain :etu
:etu rdfs:subClassOf :mem
:mem rdfs:subClassOf :pers
```

→

```
:toto rdf:type :etu
:toto rdf:type :mem
```

$$\frac{C \texttt{ rdfs:subClassOf } D \quad \text{et} \quad E \texttt{ rdf:type } C}{E \texttt{ rdf:type } D}$$

Exemple

```
:toto :inscrit :gdw
:inscrit rdfs:domain :etu
:etu rdfs:subClassOf :mem
:mem rdfs:subClassOf :pers
```

→

```
:toto rdf:type :etu
:toto rdf:type :mem
:etu rdfs:subClassOf :pers
```

$$\frac{C \texttt{ rdfs:subClassOf } D \quad \text{et} \quad D \texttt{ rdfs:subClassOf } E}{C \texttt{ rdfs:subClassOf } E}$$

Exemple

```
:toto :inscrit :gdw                    :toto rdf:type :etu
:inscrit rdfs:domain :etu              :toto rdf:type :mem
:etu rdfs:subClassOf :mem        →     :etu rdfs:subClassOf :pers
:mem rdfs:subClassOf :pers             :toto rdf:type :pers
```

$$\frac{C \texttt{ rdfs:subClassOf } D \quad \text{et} \quad E \texttt{ rdf:type } C}{E \texttt{ rdf:type } D}$$

Chaînage arrière

2 approches :

- moteur de déduction entre le stockage et moteur de requête :
  - pour chaque motif de triplet de la requête
    - remonter les règles de déduction
    - pour chercher des triplets ayant pu permettre de déduire le triplet cherché
  - fonctionnement similaire à Prolog
- réécriture de requête :
  - "déplier" les règles dans la requête
  - génère beaucoup d'UNION

Exemple avec moteur de déduction

```
:titi rdf:type :etu          :inscrit rdfs:domain :etu
:toto :inscrit :gdw          :etu rdfs:subClassOf :mem
```

Motifs cherchés : {A `rdf:type :mem`}

Réponses : {}

Exemple avec moteur de déduction

```
:titi rdf:type :etu          :inscrit rdfs:domain :etu
:toto :inscrit :gdw          :etu rdfs:subClassOf :mem
```

Motifs cherchés : {A `rdf:type` `:mem`}
{A $rdf : type$ T. T `rdf:subClassOf` `:mem`.}

Réponses : $\{E \mapsto$ `:titi`$\}$

$$\frac{C \texttt{ rdfs:subClassOf } D \quad \text{et} \quad E \texttt{ rdf:type } C}{E \texttt{ rdf:type } D}$$

Exemple avec moteur de déduction

```
:titi rdf:type :etu          :inscrit rdfs:domain :etu
:toto :inscrit :gdw          :etu rdfs:subClassOf :mem
```

Motifs cherchés : {*A* `rdf:type` `:mem`}
{*A* *rdf:type* *T*. *T* `rdf:subClassOf` `:mem`.}
{*P* *rdfs:domain* *T*. *A* *P* *O*. *T* `rdf:subClassOf` `:mem`.}

Réponses : $\{E \mapsto$ `:titi`$, E \mapsto$ `:toto`$\}$

$$\frac{P \texttt{ rdfs:domain } T \quad \text{et} \quad S \; P \; O}{S \texttt{ rdf:type } T}$$

Exemple avec réécriture

Requête :

```
SELECT ?a WHERE {
    ?a rdf:type :mem.  }
```

Exemple avec réécriture

Requête :

```
SELECT ?a WHERE {
  { ?a rdf:type :mem. }
  UNION { ?t rdfs:subClassOf :mem.
          ?a rdf:type ?t.  } } }
```

$$\frac{C \ \texttt{rdfs:subClassOf} \ D \quad \text{et} \quad E \ \texttt{rdf:type} \ C}{E \ \texttt{rdf:type} \ D}$$

Exemple avec réécriture

Requête :

```
SELECT ?a WHERE {
  { ?a rdf:type :mem. }
  UNION { ?t rdfs:subClassOf :mem.
          { { ?a rdf:type ?t. }
            UNION { ?p rdfs:domain ?t.
                    ?a ?p [] } } } }
```

$$\frac{P \texttt{ rdfs:domain } T \quad \text{et} \quad S\ P\ O}{S \texttt{ rdf:type } T}$$

Exemple avec réécriture

Requête :

```
SELECT ?a WHERE {
  { ?a rdf:type :mem. }
  UNION { ?t rdfs:subClassOf :mem.
          { { ?a rdf:type ?t. }
            UNION { ?p rdfs:domain ?t.
                    ?a ?p [] } } } }
```

⚠ règles récursives (*e.g.* transitivité de `rdfs:subClassOf`)

Avantages et inconvénients des 2 approches

Chaînage avant :

- Pas de surcoût direct à l'exécution des requêtes
- Espace de stockage supplémentaire nécessaire
- Mise à jour plus coûteuse :
  - Ajouter les triplets déduit à chaque insersion
  - Supprimer les triplets qui ne sont plus déductibles à chaque suppression d'un triplet d'origine

Chaînage arrière :

- Pas d'espace de stockage supplémentaire
- Transparent dans les mises à jour
- Surcoût au requêtage

Stockage

Que stocker ?

- Relations entre nœuds
- URIs et littéraux

A savoir :

- Peu/pas de contrainte de schéma
- Évaluation SPARQL : pas si différent de l'algèbre relationnelle

Type de magasins (*store*) RDF

- Natifs
  Stockage dédié → refaire le SGBD
- Basés sur des bases relationnelles
  - Une relation ternaire `triplets`
  - Une relation binaire par étiquette d'arête
  - RDF *wrapper* basés sur une correspondance Relationnel ↔ RDF

Stockage natif : basé sur l'indexation

Stockage dans des structures d'index, nommés selon ce qui est indexé par quoi :

- Sujet (S), Prédicat (P), Objet (O)
- un index XYZ, donne en fonction de XY les valeurs de Z
  - *e.g.* PSO : en fonction du prédicat (P) et du sujet (S) donner les objets (O)
- RDF-3X : on utilise les 6 indexes possibles :
  SPO, SOP, PSO, POS, OPS, OSP
- Remarque : l'index seul suffit, pas besoin de stocker les triplets en dehors

Représentation des données

Pas de manipulation de chaîne de caractère

- dictionnaire chaîne ↔ `int` 64 bits
- on utilise que des `int` en interne
- optimise les jointures
- plus coûteux pour les FILTER sur les valeurs
  - nécessite un appel au dictionnaire
  - pari d'une opération moins fréquente

Stockage en relationnel : table `triplet`

Sans codage en entiers :

- `triplet`(sujet, predicat, objet)

Avec codage en entiers :

- `triplet`(sujet, predicat, objet, sorte)
  - sorte : entité ou littéral
- `uris`(id,uri)
- `literals`(id,valeur,type)
  - type : au sens RDF

Il faut ensuite traduire les requêtes SPARQL en SQL

Exemple

```
SELECT ?r WHERE {
  ?r agenda:invite ?p.
  ?p agenda:nom ?n.
  FILTER(?n = "Coquery").
}
```

devient

```
SELECT u.uri
FROM triplet invite, triplet nom, literals l, uris u
WHERE invite.obj = nom.suj
  AND nom.obj = l.id
  AND l.lit = 'Coquery'
  AND invite.suj = u.id
```

Stockage en relationnel : un attribut / prédicat

Une grande table :

- une ligne / sujet
- une colonne / prédicat
  + une colonne pour stocker l'ID du sujet
- valeur : objet
- on conserve les tables `uris` et `literals`

⚠ en cas d'arêtes multiples ayant un même prédicat

Découpage en plusieurs tables en fonction des attributs

- nécessite de connaître à l'avance les cardinalités possibles de chaque prédicat

Stockage en relationnel : une table / prédicat

Chaque prédicat a sa propre table

- on conserve les tables `uris` et `literals`
- traduction vers SQL assez simple
- sauf pour pour les variables sur le prédicat

Cas extrême de l'approche précédente

Wrapper autour du modèle relationnel

Variation sur le modèle "une table / sujet"

Mapping : schéma relationnel → modèle de graphe

- pour chaque table représentant une entité :
  - clé primaire → URI sujet
  - colonne → URI prédicat
    - conversion de type
    - gestion de clé étrangère
- gestion des URIs
  - pour chaque clé primaire de chaque table : fonction de construction d'URIs
- associations n-n
  - table → prédicat

⚠ lecture seule

Stockage hybride

Mélange de stockage relationnel et de stockage natif

- relationnel : plutôt *wrapper*
- utile pour intégrer des données des 2 modèles
- interrogation en SPARQL
- nécessite un moteur SPARQL hybride qui convertira une partie de la requête en SQL

Références

- http://www.w3schools.com/rdf/rdf_intro.asp
- http://www.cambridgesemantics.com/2008/09/sparql-by-example/
- https://www.w3.org/TR/sparql11-query/
- http://www.w3.org/TR/rdf-schema/
- http://www.w3.org/TR/rdf-mt
- http://www.foaf-project.org/
- https://users.dcc.uchile.cl/~cgutierr/papers/
- http://pubman.mpdl.mpg.de/pubman/item/escidoc:1324253/component/escidoc:1324252/rdf3x.pdf