Linked Open Data & SPARQL

Mini-course — TP1 Introduction

Stéphane Derrode & Lamia Derrode · Département MI · Centrale Lyon · 2025–2026

Today's Session

Part 1 — Linked Open Data (~7 min)

- What is Linked Data?
- The RDF triple model
- URIs and prefixes
- Graph-based structure
- Real-world examples: Nobel Prizes & DBpedia

Part 2 — SPARQL (~8 min)

- Query anatomy
- SELECT, WHERE, FILTER, OPTIONAL
- ORDER BY, GROUP BY
- Advanced: UNION, EXISTS, BIND, nested queries

Then: 4h hands-on lab — 14 guided queries on DBpedia + 1 original query on a LOD dataset of your choice

Part 1

Linked Open Data

▶ Shown in class

What is Linked Data?

Source: Manu Sporny · YouTube
Duration: ~3 min

Watch for:

- Why adding links between data matters
- What makes data 'open' vs simply 'published'
- The difference between a document and a data resource

"Linked Data is about using the Web to connect related data that wasn't previously linked."

From Data to Linked Data — 3 Steps

1. Classic table

Name | City | Born
Ada | London | 1815
Charles | London | 1791

Machine-readable but not linked

2. As a graph

```
Ada → bornIn → London
Ada → bornYear → 1815
Charles → bornIn →
London
```

Relationships explicit

3. With IRIs

```
dbr:Ada_Lovelace
  → dbo:birthPlace
  → dbr:London
```

Globally addressable & linked

RDF — The Triple Model

RDF (Resource Description Framework) is a W3C standard for representing structured information about resources in a machine-readable way.

Subject

The resource being described

e.g. The Mona Lisa

Predicate

The property or relationship

e.g. hasCreator

Object

The value or related resource

e.g. Leonardo da Vinci

Read as a sentence: Subject → Predicate → Object — 'The Mona Lisa hasCreator Leonardo da Vinci'

RDF — URIs and Prefixes

URIs provide global, unambiguous identifiers for resources.

Example:
`http://dbpedia.org/resource/Lyon`

→ *uniquely identifies the city of Lyon, worldwide*

Objects can also be literals: numbers, strings, dates.

Prefixes — shorthand for long URIs

`foaf:`	→	`http://xmlns.com/foaf/0.1/`
`dbo:`	→	`http://dbpedia.org/ontology/`
`dbr:`	→	`http://dbpedia.org/resource/`
`xsd:`	→	`http://www.w3.org/2001/XMLSchema#`

To find any prefix: prefix.cc

RDF — Graph-Based Structure

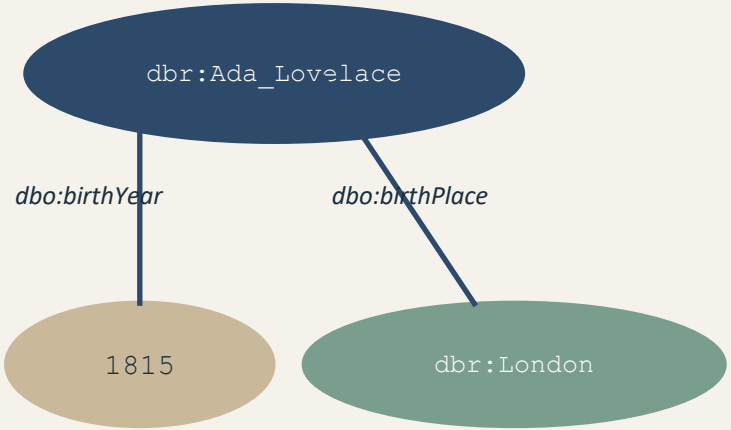
RDF organizes data as a directed graph:

- Nodes = resources (subjects and objects)
- Edges = properties (predicates)

This graph model enables:

- Easy visualization
- Flexible querying (SPARQL)
- Linking across datasets

All RDF triples together form a triplestore.



LOD Example: The Nobel Prize Dataset

- Contains all Nobel Prize data since 1901
- Three main classes: NobelPrize, LaureateAward, Laureate
- A prize can be shared between up to 3 laureates
- The same person can receive multiple prizes

`Endpoint: data.nobelprize.org/sparql`

`Documentation: data.nobelprize.org/specification`

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX nobel:
<http://data.nobelprize.org/terms/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
schema#>

SELECT DISTINCT ?name ?year WHERE {
  ?person a nobel:Laureate;
    rdfs:label ?name;
    nobel:laureateAward ?award.
  ?award nobel:year ?year.
}
ORDER BY DESC(?year)
LIMIT 10
```

LOD Example: DBpedia

- Community-driven project extracting structured data from Wikipedia
- One of the most prominent datasets in the LOD cloud
- Millions of triples describing entities and relationships

Example entity:

```
'Lyon' → mayor → 'Grégory Doucet'
'Lyon' → population → 522,250
'Lyon' → country → France
```

Endpoint: dbpedia.org/sparql

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

SELECT ?author
WHERE {
  ?book dbo:author ?author ;
        rdfs:label
          "The Catcher in the Rye"@en .
}
```

→ *Result: J.D. Salinger*

The LOD Cloud

- Maps all published LOD datasets and their links
- First version: 2007 — 12 datasets
- Today: 1,300+ datasets, billions of triples
- Domains: government, life sciences, media, geography...

→ lod-cloud.net (live, interactive)

Key datasets in the cloud:

- DBpedia — encyclopedic knowledge (hub)
- Wikidata — structured facts
- GeoNames — 11M geographic features
- Nobel Prizes, BNF, Europeana...

Principle 4 of Linked Data: include links to other IRIs — so data can be discovered automatically

Part 2

SPARQL

Querying Linked Data

Getting Started — SPARQL Endpoint & Yasgui

What is a SPARQL endpoint?

A SPARQL endpoint is a web service that accepts SPARQL queries and returns results.

Examples:

- DBpedia: dbpedia.org/sparql
- Nobel: data.nobelprize.org/sparql
- Wikidata: query.wikidata.org

Yasgui — your tool for this lab

1. Open yasgui.triply.cc
2. Default endpoint = DBpedia
3. Click endpoint URL to change it
4. Type your query → Run (Ctrl+Enter)
5. Results appear as a table below

💡 **Always add LIMIT 100 to exploratory queries — avoid timeouts!**

SPARQL Query Anatomy

```
PREFIX FOO: <...>          # Declare prefix shortcuts (optional)
PREFIX BAR: <...>

SELECT  ...                # Query result clause: what to return
WHERE {
    ...                    # Query pattern: conditions to match
}
GROUP BY ...               # Query modifiers (all optional):
HAVING   ...               #   filter on aggregates
ORDER BY ...               #   sort results
LIMIT    ...               #   cap number of results
OFFSET   ...               #   skip first N results
```

SELECT returns a table of results. Other clauses: ASK (yes/no), CONSTRUCT (new graph), DESCRIBE (resource info)

This course focuses on SELECT queries — the most common form used in the lab.

SPARQL Tutorial (1/3) — Simple SELECT + WHERE

Database (Turtle format):

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

<http://somewhere/MattJones/>
    vCard:FN "Matt Jones" .
```

Query:

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

```
Result: ?x = <http://somewhere/MattJones/>
```

?x is a variable — SPARQL finds all resources matching the pattern

SPARQL Tutorial (2/3) — Blank Nodes

Richer database:

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

_:b0 is a blank node — an anonymous resource with no URI

Query:

```
PREFIX vCard: <...>
SELECT ?x ?gn
WHERE {
  ?x vCard:FN       "Matt Jones" ;
     vCard:hasName ?z .
  ?z vCard:Given    ?gn .
}
```

```
Result: ?x = MattJones/ · ?gn = 'Matthew'
```

SPARQL Tutorial (3/3) — Multiple Persons & FILTER

Database with 4 persons:

```
MattJones     Family:"Jones"  Given:"Matthew"
RebeccaSmith  Family:"Smith"  Given:"Rebecca"
JohnSmith     Family:"Smith"  Given:"John"
SarahJones    Family:"Jones"  Given:"Sarah"
```

All triples in the graph have no intrinsic order.

Query: all family 'Smith'

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

Result: Rebecca, John

Query: names containing 'r' (case-insensitive)

```
SELECT ?g WHERE {
  ?y vCard:Given ?g .
  FILTER regex(?g, "r", "i")
}
```

Result: Matthew, Rebecca, Sarah

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 .
  }
}
```

OPTIONAL includes results even when the optional pattern does not match.

Without OPTIONAL → only writers WITH a photo
With OPTIONAL → all writers; photo = null if missing

Used in the lab for: birthDate + optional deathDate

Rule: variables inside OPTIONAL{} may be null — always check before filtering on them

SPARQL — FILTER

String filter (CONTAINS)

```
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
```

Language filter (LANGMATCHES)

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

Common FILTER functions: CONTAINS · regex() · LANGMATCHES · year() · isLiteral() · isIRI() · STRLEN()

SPARQL — ORDER BY

```
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
```

CONCAT() builds a new string from parts — here, combining given name + space + family name into a full name.

ORDER BY ASC(?x) → ascending / DESC(?x) → descending.
Multiple columns: ORDER BY ASC(?a) DESC(?b)

Part 3

SPARQL Advanced

SPARQL — GROUP BY & HAVING

GROUP BY — count persons by birthplace

```
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
```

HAVING — filter on aggregate result

```
GROUP BY ?bpl
HAVING(?cp < 5)    -- keep only birthplaces with fewer than 5 persons
LIMIT 7
```

HAVING is to GROUP BY what FILTER is to WHERE — it filters on aggregated values.

SPARQL — 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
```

GROUP_CONCAT joins all values of a variable in a group into a single string, separated by a delimiter.

Result example: ?bpl='Lyon' | ?cn=4 | ?names='Jean Moulin | Antoine de Saint-Exupéry | ...'

SPARQL — UNION & DISTINCT

UNION — combine results from two patterns

```
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
```

DISTINCT — remove duplicates

```
SELECT DISTINCT(?Xname) (COUNT(?X) AS ?cX) WHERE { ... }
ORDER BY DESC(?cX) LIMIT 5
```

DISTINCT and GROUP BY often give the same result — choose based on whether you need aggregation.

SPARQL — EXISTS, NOT EXISTS & MINUS

NOT EXISTS — people who know nobody

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name WHERE {
  ?x foaf:givenName ?name .
  FILTER NOT EXISTS { ?x foaf:knows ?who }
}
```

EXISTS — people who know someone other than themselves

```
SELECT ?name WHERE {
  ?x foaf:givenName ?name .
  FILTER EXISTS { ?x foaf:knows ?who .
                  FILTER(?who != ?x) }
}
```

MINUS — exclude solutions

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

SPARQL — BIND (Strings & Dates)

String functions with BIND

```
SELECT ?name ?upperName WHERE {
  ?p foaf:name ?name .
  BIND(UCASE(?name) AS ?upperName)
}
```

Common string functions:
LCASE, UCASE, STRLEN, SUBSTR,
CONCAT, CONTAINS, STR, LANG

Date functions with BIND

```
SELECT ?now ?y ?m ?d WHERE {
  BIND(NOW()       AS ?now)
  BIND(YEAR(?now)  AS ?y)
  BIND(MONTH(?now) AS ?m)
  BIND(DAY(?now)   AS ?d)
}
```

Common date functions:
NOW, YEAR, MONTH, DAY,
HOURS, MINUTES, TIMEZONE

BIND creates a new variable by applying a function — like a computed column in SQL.

SPARQL — Nested Queries

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

-- Find the name and age of the oldest person(s) --
SELECT ?name ?maxage
WHERE {
  {
    SELECT (MAX(?age) AS ?maxage)
    WHERE {
      ?person foaf:age ?age
    }
  }
  ?senior foaf:age  ?maxage ;
          foaf:name ?name .
}
```

The inner SELECT computes MAX(age) first — then the outer WHERE matches all persons with that age.

Nested queries are useful when you need to compute an aggregate and then filter on it — impossible with a simple HAVING.

SPARQL — Literals & VALUES

Typed literals — language tags

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

@en specifies the language of the literal — DBpedia stores labels in many languages.

VALUES — restrict results to specific values

```
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 }
}
```

VALUES is equivalent to: FILTER(?name = 'Carl Reiner'@en || ...) but more readable.

SPARQL Quick Reference

Clauses

```
SELECT · WHERE
GROUP BY · HAVING
ORDER BY ASC/DESC
LIMIT · OFFSET
DISTINCT · VALUES
```

Patterns

```
OPTIONAL { }
FILTER
UNION { }
MINUS { }
BIND(f AS ?v)
EXISTS / NOT EXISTS
```

String fns

```
CONTAINS · regex()
CONCAT · STR
STRLEN · SUBSTR
LCASE · UCASE
LANG · LANGMATCHES
```

Aggregate fns

```
COUNT · SUM
AVG · MIN · MAX
GROUP_CONCAT
SAMPLE
```

Full reference: docs.cambridgesemantics.com · W3C SPARQL 1.1: w3.org/TR/sparql11-query/

You're Ready — Let's Query!

Tools:

- Yasgui: yasgui.triply.cc
  (endpoint DBpedia pre-loaded)
- Prefix finder: prefix.cc
- Query validator: sparql.org/query-validator

Lab structure:

- Part 1: 14 guided DBpedia queries
- Part 2: 1 original query — your choice of LOD dataset

Key tips:

- Always add LIMIT 100 first
- Check the endpoint is alive before investing time
- Use prefix.cc for unknown namespaces
- Test with DESCRIBE <IRI> to explore a resource
- The lab instructions detail all available LOD datasets

Questions? → lamia.derrode@ec-lyon.fr · stephane.derrode@ec-lyon.fr