RDF-based Knowledge Graphs

Ernesto Jiménez-Ruiz
Lecturer in Artificial Intelligence

Before we start...

Module Leader and Lecturer

- **Ernesto Jiménez-Ruiz**
- Research Centre for **Adaptive Computing Systems and Machine Learning**.
- Contact:
  - Moodle forum / Teams
  - ernesto.jimenez-ruiz@city.ac.uk
  - https://www.city.ac.uk/people/academics/ernesto-jimenez-ruiz

Lecture and Lab: IN3067/INM713

Lecture:

- Lecture Capture: Recorded and live-streamed
- **Thursday, 09:00-10:50**,
- C312 Tait Building

Laboratory:

- **R201 (Franklin Building)**. 48 seats/PCs. 10 min walk.
- **Session 1: Thursday, 11:00-11:50**.
- Session 2: Thursday, 12:00-12:50 (Weeks: 1-4, weeks 7, 9 and 11). C301 on weeks 5, 8 and 10.
- Programming languages: Python and/or Java

Drop-in hours

- **Term 2**
  - Tuesday 2-3pm (online)
  - Thursdays 2-4pm (on-campus or online). Not today!
- Additional (online) drop-ins can be arranged via email.

Coursework

(Automatic) transformation of tabular data into semantic data.

- Part 1 (20%): **creation of an ontology** that covers the knowledge of a given domain. **Deadline**: Sunday, 3 March 2024, 5:00 PM
- Part 2 (80%): Design and development of a **software component**. **Deadline**: Sunday, 12 May 2024, 5:00 PM
- Allowed to **work in pairs**, or individually. Please register.

Where are we? Module organization.

1. ✓ Introduction: Becoming a knowledge scientist.
2. **RDF-based knowledge graphs.** (Today)
3. OWL ontology language. Focus on modelling.
4. SPARQL 1.0 Query Language.
5. From tabular data to KG.
6. RDFS Semantics and OWL 2 profiles.
7. Ontology Alignment.
8. Ontology (KG) Embeddings and Machine Learning.
9. SPARQL 1.1 and Graph Database solutions.
10. (Large) Language Models and KGs. (Seminar)

Graph Data Models

Data Graphs

- **Foundation** of any Knowledge Graph
- **Intuitive abstractions** to capture entities and their relationships.
- **Do not require schema** to start adding data.
- **Flexibility** to assemble data from multiple sources.

Data Graphs

- **Foundation** of any Knowledge Graph
- **Intuitive abstractions** to capture entities and their relationships.
- **Do not require schema** to start adding data.
- **Flexibility** to assemble data from multiple sources.
- Enable **queries over paths** of arbitrary length.
- Allow the use of **graph analytics** techniques.
- (Potential) correspondence with **logical** unary and binary **predicates**.

Types of graph-structured data models (i)

- Directed edge-labelled (multi)graphs (multi-relational graphs)
- Property graphs
- Heterogeneous graphs (heterogeneous information within a node)

Types of graph-structured data models (i)

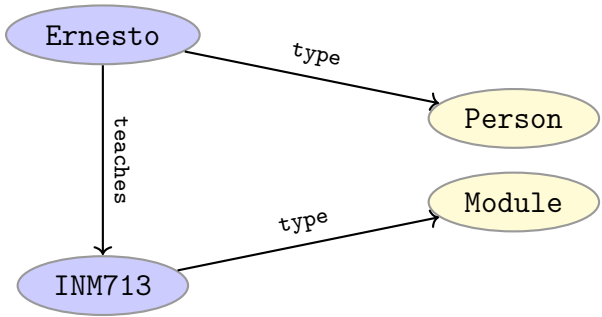
- Directed edge-labelled (multi)graphs (multi-relational graphs)
- Property graphs
- Heterogeneous graphs (heterogeneous information within a node)
- Hypergraphs (edges connecting set of nodes)
- Hypernodes (nested graphs in a node)

Types of graph-structured data models (i)

- **Directed edge-labelled (multi)graphs** (multi-relational graphs)
- **Property graphs**
- Heterogeneous graphs (heterogeneous information within a node)
- Hypergraphs (edges connecting set of nodes)
- Hypernodes (nested graphs in a node)

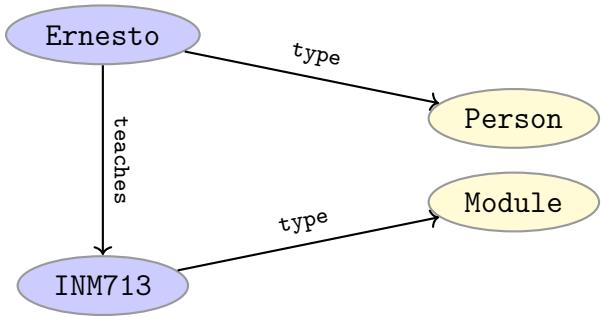
Types of graph-structured data models (ii)

Directed edge-labelled (multi)graphs.



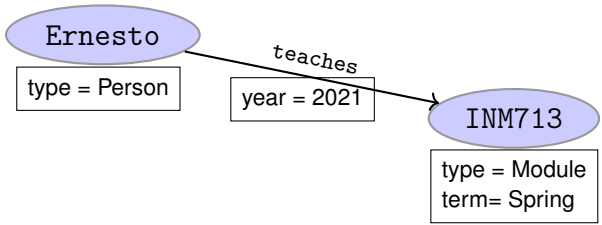
Types of graph-structured data models (ii)

Directed edge-labelled (multi)graphs. (In this module).



Types of graph-structured data models (iii)

Property graphs



How to store Graph models?

- Relational model
  - Single table with 3 columns (source, edge, target)
  - Property tables (one table per type of entity, *e.g.*, Person, Module)
  - Binary tables (one table per relationship, *e.g.*, source, target)

M. Wylot and others. RDF Data Storage and Query Processing Schemes. ACM Computing Surveys 2018

How to store Graph models?

- Relational model
  - Single table with 3 columns (source, edge, target)
  - Property tables (one table per type of entity, *e.g.*, Person, Module)
  - Binary tables (one table per relationship, *e.g.*, source, target)
- **Graph-based databases** (NoSQL)

M. Wylot and others. RDF Data Storage and Query Processing Schemes. ACM Computing Surveys 2018

NoSQL (Not only SQL) databases (i)

- **No strict definition** of NoSQL.
- They typically have a **denormalised** data model.
  - Data has some duplication.
  - Query performance is increased.
  - Writing and updates are less efficient.
- **No integrity constraints (*i.e.*, more flexibility).**
- Consistency is checked afterwards (“eventually consistent” models).

NoSQL (Not only SQL) databases (ii)

Types of NoSQL databases:

- Key-Value pair-based databases (*e.g.*, ArangoDB)
- Column-oriented databases (*e.g.*, MonetDB)
- Document-oriented databases (*e.g.*, MongoDB, ArangoDB)

(*) Object-oriented databases can also be seen as a type of NoSQL but they include data integrity.
(**) RDFLib and Apache Jena are libraries to deal with RDF graphs.

NoSQL (Not only SQL) databases (ii)

Types of NoSQL databases:

- Key-Value pair-based databases (*e.g.*, ArangoDB)
- Column-oriented databases (*e.g.*, MonetDB)
- Document-oriented databases (*e.g.*, MongoDB, ArangoDB)
- **Graph databases** (*e.g.*, Neo4j, Virtuoso, Amazon Neptune, Oracle)
  - **RDF Triplestores** (*e.g.*, RDFox, Apache Jena TBD, GraphDB)
  - Most Graph databases are also suitable as RDF triplestores.

(*) Object-oriented databases can also be seen as a type of NoSQL but they include data integrity.
(**) RDFLib and Apache Jena are libraries to deal with RDF graphs.

RDF: Resource Description Framework

RDF in a nutshell (i)

- Resource Description Framework (RDF)
- A **standardised data model** based on the directed edge-labelled graph model.
  - **Nodes**: Internationalised Resource Identifiers (IRIs), literals, and blank nodes (nodes without identifier).
  - **Edges**: IRIs
- **W3C recommendation**.
- Conceptual **modelling of resources**.

RDF in a nutshell (ii)

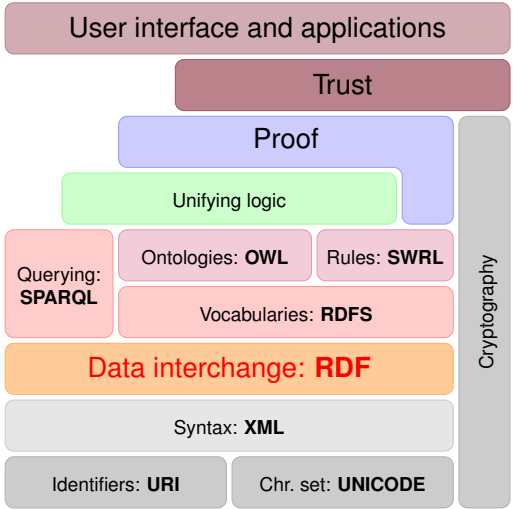
- RDF intended for annotation of Web-accessible resources (1999).
- Evolved into a general purpose language for describing structured information—**on the web or elsewhere**.
- The goal of RDF is to enable applications to **exchange data in a meaning-preserving way**.

RDF in a nutshell (ii)

- RDF intended for annotation of Web-accessible resources (1999).
- Evolved into a general purpose language for describing structured information—**on the web or elsewhere**.
- The goal of RDF is to enable applications to **exchange data in a meaning-preserving way**.
- No dependency on the database/storage solution.
- It is considered the **basic representation format** underlying the Semantic Web (and many KGs).
- In this module we will study **RDF-based Knowledge Graphs**.

Semantic Web Stack

– Central block in the SW stack.



RDF data model

All you need are triples

RDF Triples

- All information in RDF is expressed using a *triple* pattern.
- A triple consists of a subject, a predicate, and an object.

RDF Triples

- All information in RDF is expressed using a *triple* pattern.
- A triple consists of a subject, a predicate, and an object.
  Examples:

subject	predicate	object
England	has capital	London

RDF Triples

- All information in RDF is expressed using a *triple* pattern.
- A triple consists of a subject, a predicate, and an object.
  Examples:

subject	predicate	object
England	has capital	London
England	has king	Charles III

RDF Triples

- All information in RDF is expressed using a *triple* pattern.
- A triple consists of a subject, a predicate, and an object.
  Examples:

subject	predicate	object
England	has capital	London
England	has king	Charles III
Charles III	born year	1948

RDF Triples

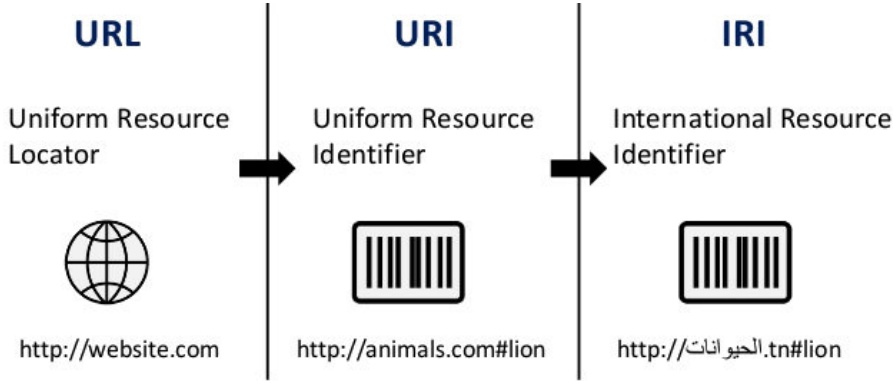
- All information in RDF is expressed using a *triple* pattern.
- A triple consists of a subject, a predicate, and an object.
  Examples:

subject	predicate	object
England	has capital	London
England	has king	Charles III
Charles III	born year	1948

- Another word for an RDF triple is a *statement* or *fact*.
- The elements of an RDF triple are either: *URI references*, *literals*, *blank nodes*.

All you need are triples... and URIs

Identifying resources



Identifying resources in RDF

- RDF talks about *resources*.
- Resources are identified by IRIs/URIs.
- E.g., in `dbpedia.org` KG graph:

Identifying resources in RDF

- RDF talks about *resources*.
- Resources are identified by IRIs/URIs.
- E.g., in `dbpedia.org` KG graph:

England:	`http://dbpedia.org/resource/England`
has capital:	`http://dbpedia.org/ontology/capital`
London:	`http://dbpedia.org/resource/London`
has king:	`http://dbpedia.org/ontology/monarch`
Charles III:	`http://dbpedia.org/resource/Charles,_Prince_of_Wales`

Identifying resources in RDF

- RDF talks about *resources*.
- Resources are identified by IRIs/URIs.
- E.g., in `dbpedia.org` KG graph:

England:	`http://dbpedia.org/resource/England`
has capital:	`http://dbpedia.org/ontology/capital`
London:	`http://dbpedia.org/resource/London`
has king:	`http://dbpedia.org/ontology/monarch`
Charles III:	`http://dbpedia.org/resource/Charles,_Prince_of_Wales`

- URIs/IRIs are not necessarily dereferenceable.

Building an URI

`scheme:[//authority]path[?query][#fragment]` (in brackets optional elements)

- scheme: `http`, `https`, `ftp`, `file`, etc.
- authority: typically the domain, `//www.city.ac.uk`
- path: `/sst/cs/academics`
- query: additional information for a Web service (not used in our URIs)
- fragment: to address a sub-part of a retrieved resource, `#ernesto`

e.g., `http://www.city.ac.uk/sst/cs/academics#ernesto`

Pascal Hitzler, Markus Krotzsch, Sebastian Rudolph. Foundations of Semantic Web Technologies. CRC Press 2009.

Identifying resources: URI $\not\subseteq$ URL

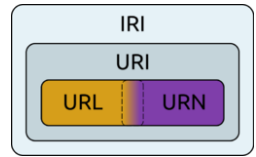
URLs are not the only URIs:

Identifying resources: URI $\not\subseteq$ URL

URLs are not the only URIs:

– ISBN:

`urn:isbn:978-1-4503-7615-0`



Identifying resources: URI $\not\subset$ URL

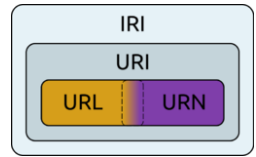
URLs are not the only URIs:

- ISBN:

  `urn:isbn:978-1-4503-7615-0`

- Geo:

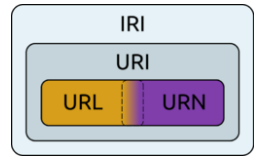
  `geo:51.527264,-0.10247`



Identifying resources: URI $\not\subseteq$ URL

URLs are not the only URIs:

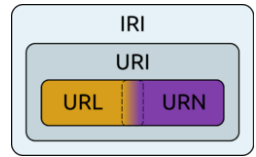
- ISBN:
  `urn:isbn:978-1-4503-7615-0`
- Geo:
  `geo:51.527264,-0.10247`
- Mail:
  `mailto:ernesto.jimenez-ruiz@city.ac.uk`



Identifying resources: URI $\not\subseteq$ URL

URLs are not the only URIs:

- ISBN:
  `urn:isbn:978-1-4503-7615-0`
- Geo:
  `geo:51.527264,-0.10247`
- Mail:
  `mailto:ernesto.jimenez-ruiz@city.ac.uk`
- and others . . .



Identifying resources: URIs and (local) CURIs

– **URIs are often long** and hard to read and write.

Identifying resources: URIs and (local) CURIs

- **URIs are often long** and hard to read and write.
- Most serialisations use an **abbreviation** mechanism.
  - Define “prefixes”, “namespaces”.
  - RDF/XML format: XML namespaces and entities.

Identifying resources: URIs and (local) CURIs

- **URIs are often long** and hard to read and write.
- Most serialisations use an **abbreviation** mechanism.
  - Define “prefixes”, “namespaces”.
  - RDF/XML format: XML namespaces and entities.
- E.g., in Turtle serialisation:

```
@prefix dbr:  <http://dbpedia.org/resource/> .
@prefix dbo:  <http://dbpedia.org/ontology/> .
```

- A *CURI* (Compact URI) or *QName* like ‘`dbr:London`’ stands for `http://dbpedia.org/resource/London`

URIs, QNames and data value

– We can then state that England's capital is London as:

```
<http://dbpedia.org/resource/England
<http://dbpedia.org/ontology/capital>
<http://dbpedia.org/resource/London> .
```

URIs, QNames and data value

– We can then state that England's capital is London as:

```
<http://dbpedia.org/resource/England
<http://dbpedia.org/ontology/capital>
<http://dbpedia.org/resource/London> .
```

– Or using prefixes:

```
dbr:England  dbo:capital  dbr:London .
```

URIs, QNames and data value

– We can then state that England's capital is London as:

```
<http://dbpedia.org/resource/England
<http://dbpedia.org/ontology/capital>
<http://dbpedia.org/resource/London> .
```

– Or using prefixes:

```
dbr:England  dbo:capital  dbr:London .
```

– But what if we want to state that London's population is **9,304,000**?

– We cannot have one URI for every integer, decimal number, string, etc.

All you need are triples... and URIs... and literals

Literals in RDF (i)

- Literals are used to represent **data values**.
- A literal in a RDF consist of these elements:
  - A lexical form: “London”, “9,304,000”, “Londres”
  - A (supported) datatype IRI (from OWL, RDF or XML Schema datatypes): *e.g.*, ^^xsd:integer
  - A language tag (*e.g.*, “en”, “es”)

Literals in RDF (i)

- Literals are used to represent **data values**.
- A literal in a RDF consist of these elements:
  - A lexical form: “London”, “9,304,000”, “Londres”
  - A (supported) datatype IRI (from OWL, RDF or XML Schema datatypes): *e.g.*, ^^xsd:integer
  - A language tag (*e.g.*, “en”, “es”)
- Examples:

```
dbr:London dbo:population "9,304,000"^^xsd:integer .
dbr:London rdfs:label "London"^^xsd:string .
dbr:London rdfs:label "Londres"@es .
```

Literals in RDF (ii)

– If the **literal is not typed** it is assumed to be a string:

```
dbr:London dbo:officialName "London" .
```

Literals in RDF (ii)

- If the **literal is not typed** it is assumed to be a string:

```
dbr:London dbo:officialName "London" .
```

- The **language defines metadata** about the string value, but **not correctness** of the language.

Literals in RDF (ii)

- If the **literal is not typed** it is assumed to be a string:

```
dbr:London dbo:officialName "London" .
```

- The **language defines metadata** about the string value, but **not correctness** of the language.
- **Ill-typed** literals:
  - Syntactically correct but semantically inconsistent.
  - The graph should still be constructed but with a warning.
  - *e.g.*, `dbr:London dbo:areaUrbanKm2 "1737.9"^^xsd:integer .`

Literals in RDF (iii)

- Equality:
  - "1572.0"^^xsd:float and "1572"^^xsd:float
  - "01"^^xsd:integer and "1"^^xsd:integer
  - "Argentina"@es and "Argentina"@en
  - "1"^^xsd:integer and "1.0"^^xsd:float
  - The values are (semantically) the same, but the RDF literal is not (syntactically) the same.

RDF Graph = set of triples with URIs and literals (and blank nodes)

RDF Graphs

– An *RDF graph* is a set of triples. E.g.,

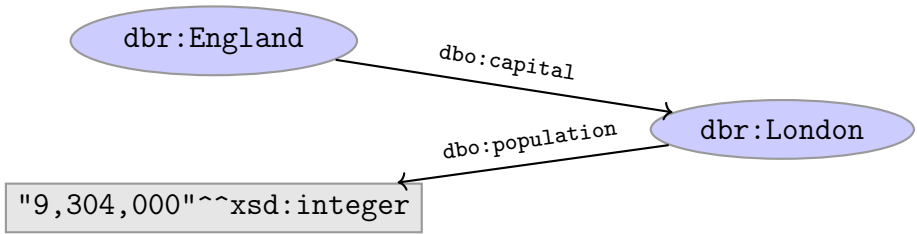
```
dbr:England  dbo:capital      dbr:London .
dbr:London    dbo:population  "9,304,000"^^xsd:integer .
```

RDF Graphs

– An *RDF graph* is a set of triples. E.g.,

```
dbr:England  dbo:capital      dbr:London .
dbr:London   dbo:population   "9,304,000"^^xsd:integer .
```

– RDF graphs are often represented as a directed labelled graph:

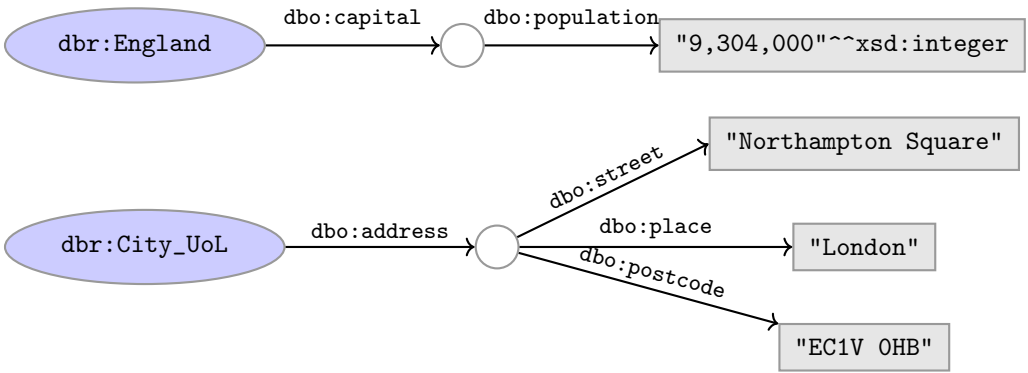


RDF Named Graphs

- We can **refer to a set of triples** with a name (*i.e.*, URI)
- Why?
  - **Independent** sources: 1 file, 1 graph.
  - Add specific **context** to a set of triples (*e.g.*, provenance).
  - Allowing to **query** a specific set of triples.
- Triples in a named graph sometimes referred as **quads**.

RDF Graphs: Blank nodes

- Blank nodes are like resources without a URI.
- Use when resource is unknown, or has no (natural) identifier. *e.g.*,:



RDF Triple Grammar

– Literals and blank nodes may not appear everywhere in triples:

RDF Triple Grammar

– Literals and blank nodes may not appear everywhere in triples:

	s	p	o
• URI references may occur in all positions	✔	✔	✔

RDF Triple Grammar

– Literals and blank nodes may not appear everywhere in triples:

	s	p	o
• URI references may occur in all positions	✔	✔	✔
• Literals may only occur in object position	✘	✘	✔

RDF Triple Grammar

– Literals and blank nodes may not appear everywhere in triples:

	s	p	o
• URI references may occur in all positions	✔	✔	✔
• Literals may only occur in object position	✘	✘	✔
• Blank nodes can not occur in predicate position	✔	✘	✔

RDF Triple Grammar

– Literals and blank nodes may not appear everywhere in triples:

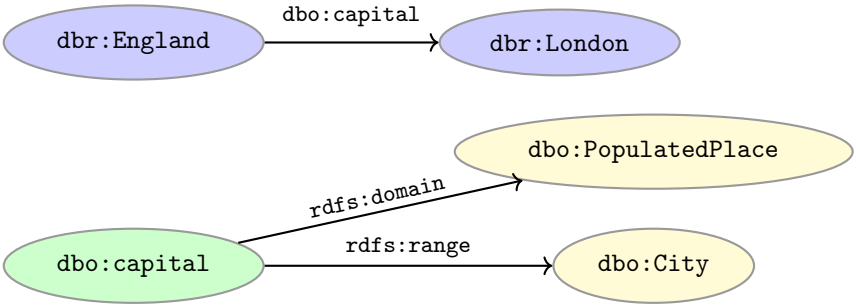
	s	p	o
• URI references may occur in all positions	✔	✔	✔
• Literals may only occur in object position	✘	✘	✔
• Blank nodes can not occur in predicate position	✔	✘	✔

– Why?
  – Literals are just values, no relationships from literals allowed.
  – Blank nodes in predicate position deemed “too meaningless” and confusing.

Is a RDF graph a graph?

Yes, although a resource can appear as both an edge and a node:

```
dbr:England   dbr:capital   dbr:London            .
dbr:capital   rdfs:domain   dbo:PopulatedPlace    .
dbr:capital   rdfs:range    dbr:City              .
```



RDF Data Model: Summary

Why URIs?

- URIs naturally have a **“global” scope**, unique throughout the web.
  - Contrasts to, e.g., keys in rel. DB which are unique within a table.
  - Helps to avoid name clashes.
  - Example: merging two product catalogues.

    `http://www.abc-company.com/category/item/123`

    `http://www.xyz-company.com/product/123`

Why URIs?

- URIs naturally have a **“global” scope**, unique throughout the web.
  - Contrasts to, e.g., keys in rel. DB which are unique within a table.
  - Helps to avoid name clashes.
  - Example: merging two product catalogues.

    `http://www.abc-company.com/category/item/123`

    `http://www.xyz-company.com/product/123`

- URLs are also **addresses**.
  - Exploit the well-functioning machinery of web browsing.
  - Find data by following data identifiers, i.e., URIs.
  - `https://dbpedia.org/resource/England`
  - *“A web of data.”*

Why triples?

- Simple unit of information
- Any information format can be transformed into triples.
  - Examples:

Tabular (spreadsheets, DBs):	row	column	cell
Trees (XML):	parent	path	child

Why triples?

- Simple unit of information
- Any information format can be transformed into triples.
  - Examples:

    Tabular (spreadsheets, DBs): row column cell
    Trees (XML): parent path child

- Relationships are made explicit and are first-class citizens:
  - The predicate is an element in the triple.
  - Can be described in RDF (*i.e.*, triples describing a predicate).

    dbo:capital rdfs:domain dbo:PopulatedPlace .

Why graphs?

- A single, but highly **versatile**, format.
  - Everything is on the same format: triples!
- Common way of **(visually) representing** entities and relationships.

Why graphs?

- A single, but highly **versatile**, format.
  - Everything is on the same format: triples!
- Common way of **(visually) representing** entities and relationships.
- **Graph analytics** can be performed over the RDF graph.

Why graphs?

- A single, but highly **versatile**, format.
  - Everything is on the same format: triples!
- Common way of **(visually) representing** entities and relationships.
- **Graph analytics** can be performed over the RDF graph.
- **Flexibility to merge** RDF graphs: Just take their union
  - With tabular data, table dimensions must match.
  - With trees, a node can only have one parent.

Why graphs?

- A single, but highly **versatile**, format.
  - Everything is on the same format: triples!
- Common way of **(visually) representing** entities and relationships.
- **Graph analytics** can be performed over the RDF graph.
- **Flexibility to merge** RDF graphs: Just take their union
  - With tabular data, table dimensions must match.
  - With trees, a node can only have one parent.
- **Flexibility to extend** the RDF graph? Just add more triples!
  - Need not redefine the database table, or
  - to restructure the XML schema.

Example RDF-based (Knowledge) Graph

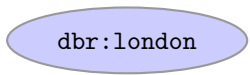
London is a city in England called Londres in Spanish

```
dbr:london rdf:type dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```

Example RDF-based (Knowledge) Graph

London is a city in England called Londres in Spanish

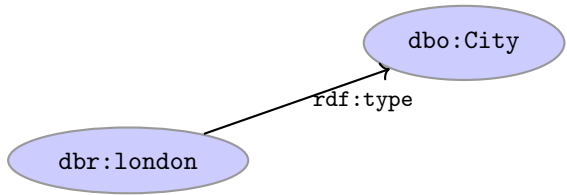
```
dbr:london rdf:type dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```



Example RDF-based (Knowledge) Graph

London is a city in England called Londres in Spanish

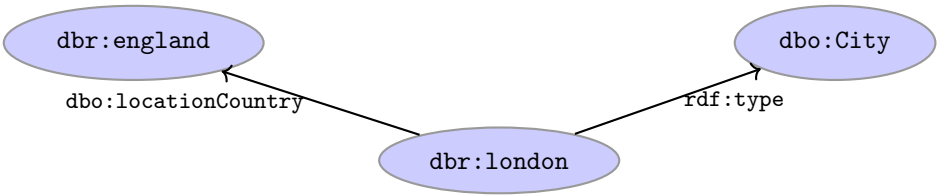
```
dbr:london rdf:type dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```



Example RDF-based (Knowledge) Graph

London is a city in England called Londres in Spanish

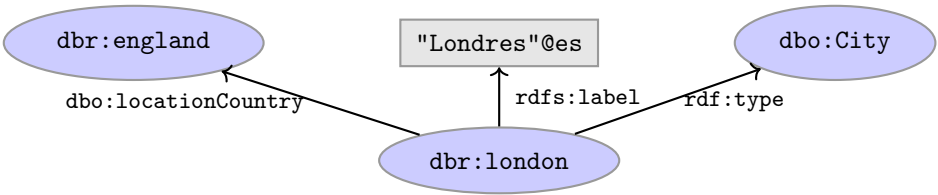
```
dbr:london rdf:type dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```



Example RDF-based (Knowledge) Graph

London is a city in England called Londres in Spanish

```
dbr:london rdf:type dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```



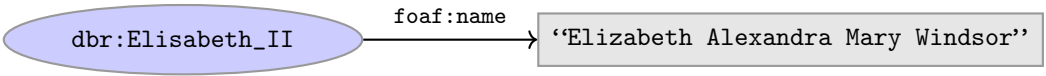
RDF serialisations

RDF Serialisations

There are many serialisations for the RDF data model:

- RDF/XML the W3C standard
- **Turtle**
- JSON-LD
- N3
- TriG

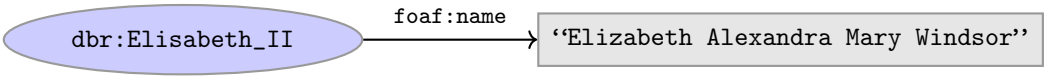
RDF/XML Serialisation



Machine readable:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:dbp="http://dbpedia.org/resource/"
         xmlns:foaf="http://xmlns.com/foaf/0.1/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://dbpedia.org/resource/Elisabeth_II">
    <foaf:name>Elizabeth Alexandra Mary Windsor</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

Turtle Serialisation



Human readable/writable

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

dbr:Elisabeth_II foaf:name "Elizabeth Alexandra Mary Windsor" .
```

Turtle: URI references and triples

Full URIs are surrounded by < and >:

```
<http://dbpedia.org/resource/London>
```

Turtle: URI references and triples

Full URIs are surrounded by < and >:

```
<http://dbpedia.org/resource/London>
```

Statements are triples terminated by a period '.':

```
<http://dbpedia.org/resource/London>
   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://dbpedia.org/ontology/City> .
```

Turtle: URI references and triples

Full URIs are surrounded by < and >:

```
<http://dbpedia.org/resource/London>
```

Statements are triples terminated by a period '.':

```
<http://dbpedia.org/resource/London>
   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://dbpedia.org/ontology/City> .
```

Use '`a`' to abbreviate `rdf:type`:

Turtle: Namespaces

QNames/CURIs are written without any special characters (*e.g.*, < and >).

Turtle: Namespaces

QNames/CURIs are written without any special characters (*e.g.*, < and >).

Namespace prefixes are declared with `@prefix`:

```
@prefix dbr: <http://dbpedia.org/resource/> .

dbr:London a <http://dbpedia.org/ontology/City> .
```

Turtle: Namespaces

QNames/CURIs are written without any special characters (*e.g.*, < and >).

Namespace prefixes are declared with `@prefix`:

```
@prefix dbr: <http://dbpedia.org/resource/> .

dbr:London a <http://dbpedia.org/ontology/City> .
```

A default namespace may be declared:

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix : <http://dbpedia.org/ontology/> .

dbr:London a :City .
```

Turtle: Literals

Literal values are enclosed in double quotes:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix : <http://dbpedia.org/ontology/> .

dbr:London rdfs:label "City of London" .
```

Turtle: Literals

Literal values are enclosed in double quotes:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix : <http://dbpedia.org/ontology/> .

dbr:London rdfs:label "City of London" .
```

Possibly with type or language information:

```
dbr:London rdfs:label "Londres"@es .
dbr:London :population "9,304,000"^^xsd:integer .
```

Turtle: Literals

Literal values are enclosed in double quotes:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix : <http://dbpedia.org/ontology/> .

dbr:London rdfs:label "City of London" .
```

Possibly with type or language information:

```
dbr:London rdfs:label "Londres"@es .
dbr:London :population "9,304,000"^^xsd:integer .
```

Numbers and booleans may be written without quotes:

```
dbr:London :population 9,304,000 .
dbr:London :isCapital true .
```

Turtle: Statements sharing elements (i)

Instead of:

```
dbr:London rdf:type dbo:City .
dbr:London rdfs:label "London" .
dbr:London :population 9,304,000 .
```

Turtle: Statements sharing elements (i)

Instead of:

```
dbr:London rdf:type dbo:City .
dbr:London rdfs:label "London" .
dbr:London :population 9,304,000 .
```

. . . statements may share a subject with ‘;’:

```
dbr:London rdf:type dbo:City ;
           rdfs:label "London" ;
           :population 9,304,000 .
```

Turtle: Statements sharing elements (ii)

Instead of:

```
dbr:London rdfs:label "London"@en .
dbr:London rdfs:label "Londres"@es .
dbr:London rdfs:label "Londra"@it .
```

Turtle: Statements sharing elements (ii)

Instead of:

```
dbr:London rdfs:label "London"@en .
dbr:London rdfs:label "Londres"@es .
dbr:London rdfs:label "Londra"@it .
```

. . . statements may share subject and predicate with ‘,’:

```
dbr:London rdfs:label "London"@en ,
                      "Londres"@es ,
                      "Londra"@it .
```

Turtle: Blank nodes

Blank nodes are designated with underscores or `[...]`.

England has a capital with population 9,304,000:

```
dbr:England :capital _:someplace .
_:someplace :population 9,304,000 .
```

Turtle: Blank nodes

Blank nodes are designated with underscores or `[...]`.

England has a capital with population 9,304,000:

```
dbr:England :capital _:someplace .
_:someplace :population 9,304,000 .
```

There is a city with official name London:

```
[] a :City ;
   :officialName "London" .
```

Turtle: Blank nodes

Blank nodes are designated with underscores or `[...]`.

England has a capital with population 9,304,000:

```
dbr:England :capital _:someplace .
_:someplace :population 9,304,000 .
```

There is a city with official name London:

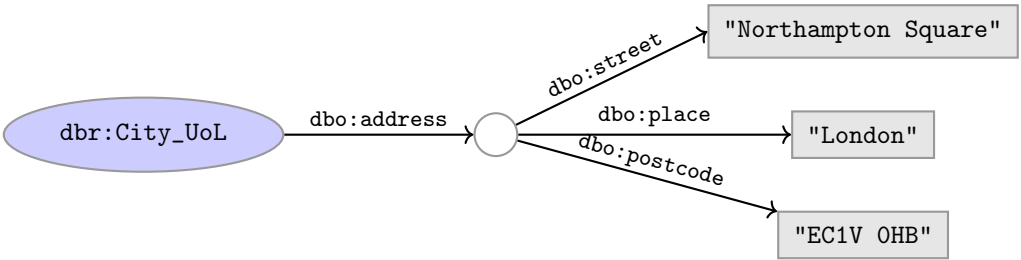
```
[] a :City ;
   :officialName "London" .
```

City has address Northampton Square, EC1V 0HB:

```
:City_UoL :address [ :street "Northampton Square" ;
                     :postcode "EC1V 0HB" ] .
```

Turtle: Blank nodes (Question)

The blank node here:

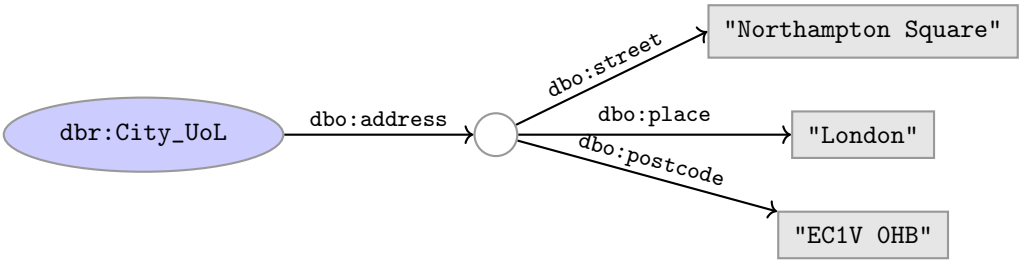


has no ‘name’ or ‘id’.

Why does Turtle use ‘blank node identifiers’ like `_:someplace`?

Turtle: Blank nodes (Question)

The blank node here:



has no ‘name’ or ‘id’.

Why does Turtle use ‘blank node identifiers’ like `_:someplace`?

Answer: makes it possible to use same node in several triples.

Turtle: Merging RDF files (i)

Merging two RDF files containing named blank nodes

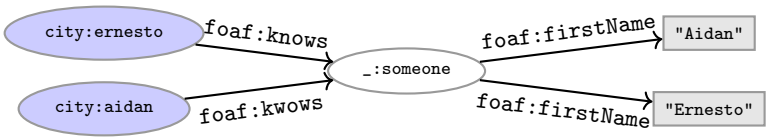
File 1

```
city:ernesto foaf:knows _:someone .
_:someone foaf:firstName  "Aidan" .
```

File 2

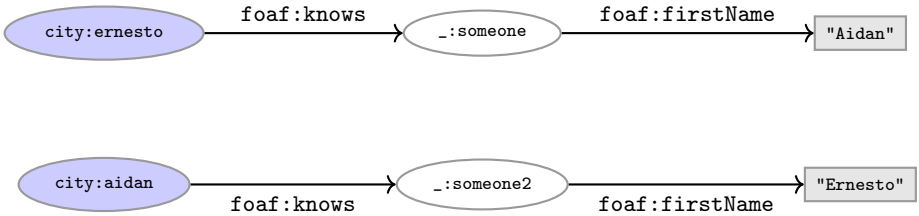
```
city:aidan foaf:knows _:someone .
_:someone foaf:firstName  "Ernesto" .
```

gives the RDF graph:



Turtle: Merging RDF files (i)

Solution: Renaming `_:someone` to `_:someone2` in File 2.



Turtle: complex statements

We can use triples to form complex statements, e.g.:

Turtle: complex statements

We can use triples to form complex statements, e.g.:

Data structures

```
:INM373 :hasLecturer
          [ rdf:first :epriego ;
            rdf:rest [ rdf:first :ernesto ;
                       rdf:rest [ rdf:first :carlos ;
                                  rdf:rest [ rdf:first :andy ;
                                             rdf:rest [ rdf:first :nikos ;
                                                        rdf:rest rdf:nil ]
                                  ] ] ] ] .
```

Turtle: complex statements

We can use triples to form complex statements, e.g.:

Data structures

```
:INM373 :hasLecturer
          [ rdf:first :epriego ;
            rdf:rest [ rdf:first :ernesto ;
                       rdf:rest [ rdf:first :carlos ;
                                  rdf:rest [ rdf:first :andy ;
                                             rdf:rest [ rdf:first :nikos ;
                                                        rdf:rest rdf:nil ]
                                           ] ] ] ] .
```

Turtle shorthand for lists

```
:INM373 :hasLecturer (:epriego :ernesto :carlos :andy :nikos ) .
```

Turtle: TriG to support named graphs

– TriG RDF syntax extends Turtle with support for named graphs.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <http://www.example.org/university/london/city#> .

:G1 {
    :INM373 rdf:type :Module .
    :INM373 :hasLecturer ( :epriego :ernesto :carlos :andy :nikos ) .
    :INM373 :module_type "Core" }
:G2 {
   :IN3067-INM713 rdf:type :Module .
   :IN3067-INM713 :hasLecturer ( :ernesto ) .
   :IN3067-INM713 :module_type "Elective" }
```

Turtle: Other things

Use ‘#’ to comment:

```
# This is a comment.
dbr:London a dbo:City . # This is another comment.
```

Turtle: Other things

Use ‘#’ to comment:

```
# This is a comment.
dbr:London a dbo:City . # This is another comment.
```

Use ‘\’ to escape special characters:

```
:someGuy :foaf:name "James \"Mr. Man\" Olson" .
```

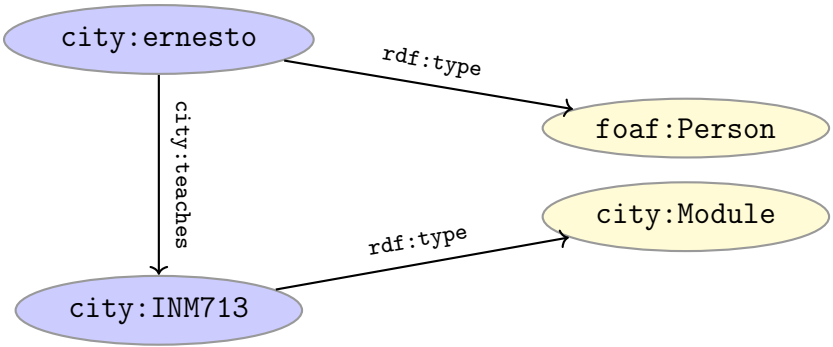
Turtle specification: `http://www.w3.org/TR/turtle/`.

RDF and Property Graphs

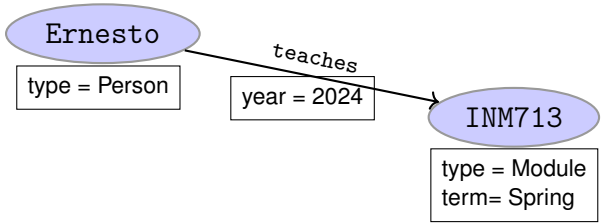
Annotating statements/triples in RDF (i)

How to encode that **“Ernesto teaches INM713 in 2024”**?

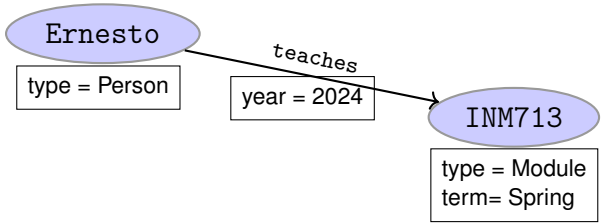
(*i.e.*, **Metadata of a triple**: provenance, beliefs, uncertainty, creator, time, etc.)



Using property Graphs

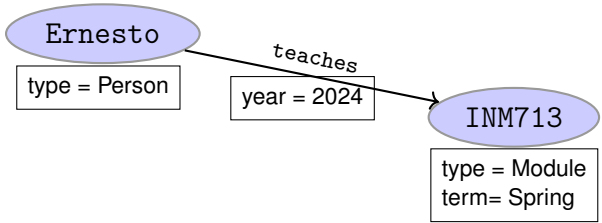


Using property Graphs



But can we do it in RDF?

Using property Graphs



But can we do it in RDF? Yes!

Annotating statements/triples in RDF (ii)

RDF Solution 1: **Named Graphs**

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dbo: <http://dbpedia.org/ontology/>
@prefix : <http://www.example.org/university/london/city#> .

:AcademicYear2024 {
   :ernesto :teaches :INM713 .
   :INM713 rdf:type :Module .
   :INM713 :module_type "Elective"
}
:AcademicYear2024 dbo:year "2024"^^xsd:gYear .
```

Annotating statements/triples in RDF (iii)

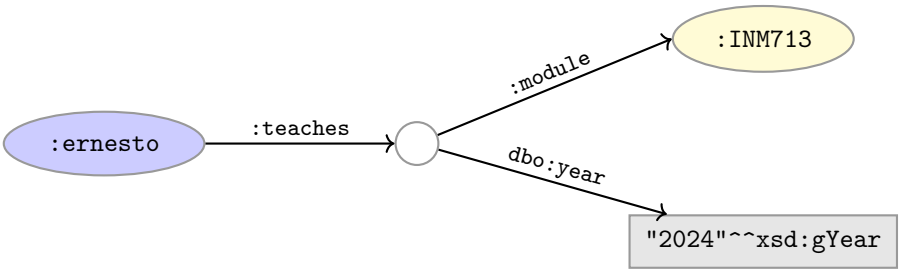
RDF Solution 2: **Singleton Property**

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dbo: <http://dbpedia.org/ontology/>
@prefix : <http://www.example.org/university/london/city#> .

:teaches#2024 rdf:singletonPropertyOf :teaches .
:teaches#2024 dbo:year "2024"^^xsd:gYear .
:ernesto :teaches#2024 :INM713 .
```

Annotating statements/triples in RDF (iv)

RDF Solution 3: **Blank Nodes**



Annotating statements/triples in RDF (v)

RDF Solution 4: **Reification**

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dbo: <http://dbpedia.org/ontology/>
@prefix : <http://www.example.org/university/london/city#> .

:ernesto :teaches :INM713 .
_:s rdf:type rdf:Statement
    rdf:subject :ernesto;
    rdf:predicate :teaches ;
    rdf:object :INM713 ;
    dbo:year "2024"^^xsd:gYear .
:aidan :likes _:s .
```

Annotating statements/triples in RDF (vi)

RDF Solution 5: **RDF*** or **RDF-Star** (soon a W3C recommendation)

- Uses triple operator « »
- Compact solution: less triples, less space, faster to load
- Closer to Property Graphs

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dbo: <http://dbpedia.org/ontology/>
@prefix : <http://www.example.org/university/london/city#> .

:ernesto :teaches :INM713 .
<<:ernesto :teaches :INM713>> dbo:year "2024"^^xsd:gYear .
```

O. Hartig and B. Thompson. Foundations of an Alternative Approach to Reification in RDF. https://arxiv.org/abs/1406.3399.
Latest information: https://w3c.github.io/rdf-star/

RDF and Property Graphs

– RDF workarounds/extensions to annotate triples.

– Is this a limitation?

RDF and Property Graphs

- RDF workarounds/extensions to annotate triples.
- Is this a limitation?
- More a **feature** due to its relation to **(logic-based) unary and binary predicates**.

RDF and Property Graphs

- RDF workarounds/extensions to annotate triples.
- Is this a limitation?
- More a **feature** due to its relation to **(logic-based) unary and binary predicates**.
- (OWL-layered) **RDF-based KGs have clear semantics**.
- Property graphs do not have the same semantic commitment.

RDF vocabularies

Vocabularies

- Families of related notions are grouped into *vocabularies*.
- Some important, well-known namespaces—and prefixes:

  Modelling vocabulary:

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

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

  `owl:` `<http://www.w3.org/2002/07/owl#>` – OWL

  Support vocabularies:

  `foaf:` `<http://xmlns.com/foaf/0.1/>` – Friend of a friend

  `dc:` `<http://purl.org/dc/terms/>` – Dublin Core

  `bfo:` `<http://purl.obolibrary.org/obo/bfo.owl#>` – Basic Formal Ontology

Vocabularies

- Families of related notions are grouped into *vocabularies*.
- Some important, well-known namespaces—and prefixes:

  Modelling vocabulary:

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

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

  `owl:` `<http://www.w3.org/2002/07/owl#>` – OWL

  Support vocabularies:

  `foaf:` `<http://xmlns.com/foaf/0.1/>` – Friend of a friend

  `dc:` `<http://purl.org/dc/terms/>` – Dublin Core

  `bfo:` `<http://purl.obolibrary.org/obo/bfo.owl#>` – Basic Formal Ontology

- Usually, a description is published at the namespace base URI.
- Note that the prefix is not standardised.

Example vocabularies: RDF, RDFS

RDF: describing RDF graphs.

- `rdf:Statement`
- `rdf:subject`, `rdf:predicate`, `rdf:object`
- `rdf:type`

RDFS: describing RDF vocabularies.

- `rdfs:Class`
- `rdfs:subClassOf`, `rdfs:subPropertyOf`
- `rdfs:domain`, `rdfs:range`
- `rdfs:label`

Examples:

```
dbr:London rdf:type dbo:City .
dbr:London rdfs:label "London"@en .
dbo:City rdfs:subClassOf dbo:Place .
```

Example vocabularies: OWL

OWL: describing ontologies

- `owl:inverseOf`
- `owl:equivalentClass`
- `owl:disjointWith`
- `owl:sameAs`

Examples:

```
dbr:London owl:sameAs ex:London .
dbo:location owl:inverseOf dbo:isLocatedIn .
dbo:City owl:disjointWith dbo:Person .
dbo:City owl:equivalentClass ex:City .
```

Example vocabularies: FOAF, Dublin Core

FOAF: person data and relations.

- `foaf:Person`
- `foaf:knows`
- `foaf:firstName`,
  `foaf:lastName`,
  `foaf:gender`

Example vocabularies: FOAF, Dublin Core

FOAF: person data and relations.

- `foaf:Person`
- `foaf:knows`
- `foaf:firstName`, `foaf:lastName`, `foaf:gender`

Dublin Core: library metadata.

- `dc:creator`, `dc:contributor`
- `dc:format`, `dc:language`, `dc:licence`

Example vocabularies: FOAF, Dublin Core

FOAF: person data and relations.

- `foaf:Person`
- `foaf:knows`
- `foaf:firstName`, `foaf:lastName`, `foaf:gender`

Dublin Core: library metadata.

- `dc:creator`, `dc:contributor`
- `dc:format`, `dc:language`, `dc:licence`

Examples:

```
city:ernesto rdf:type foaf:Person .
city:ernesto foaf:knows city:carlos .
city:IN3067-INM713 dc:creator city:ernesto .
```

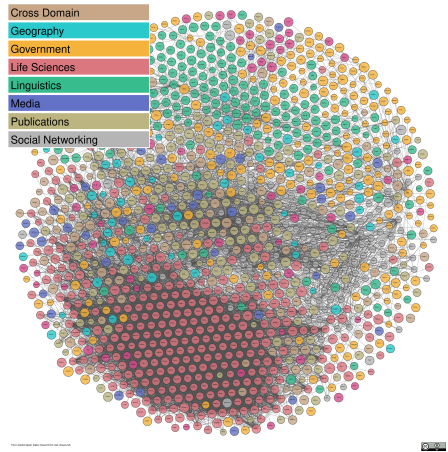
Example vocabularies: BFO

- Basic Formal Ontology:
  `http://www.obofoundry.org/ontology/bfo.html`
- It is an “upper level ontology”
- Lays the foundations of many ontologies in the biological domain.
- *e.g.*, `http://bioportal.bioontology.org/`

Knowledge Graphs on the Web and beyond

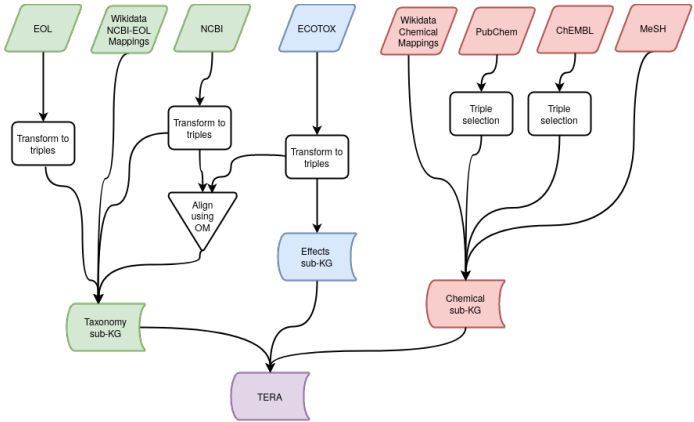
Open KGs

- Linked Open Datasets:
  `https://lod-cloud.net/`
- Diverse domains: media, government, geography, tourism, life sciences, ecotoxicology, etc.
- General purpose KGs: DBpedia, Freebase, Wikidata, YAGO
  - `https://www.wikidata.org/`
  - `https://dbpedia.org/`



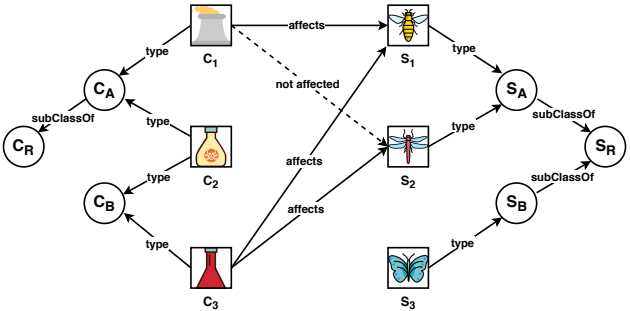
Domain specific KGs: Ecotoxicology (i)

– **Integrates** disparate sources about species, chemicals and effect data.



Domain specific KGs: Ecotoxicology (ii)

– Enhances **data access** and drives the **prediction** of effect data.



Resources and publications: `https://github.com/NIVA-Knowledge-Graph/`

Enterprise KGs (i)

- Websearch (*e.g.*, Bing, Google),
- Commerce (*e.g.*, Airbnb, Amazon, eBay, Uber),
- Social networks (*e.g.*, Facebook, LinkedIn),
- Finance (*e.g.*, Accenture, Banca d'Italia, Bloomberg, Capital One)

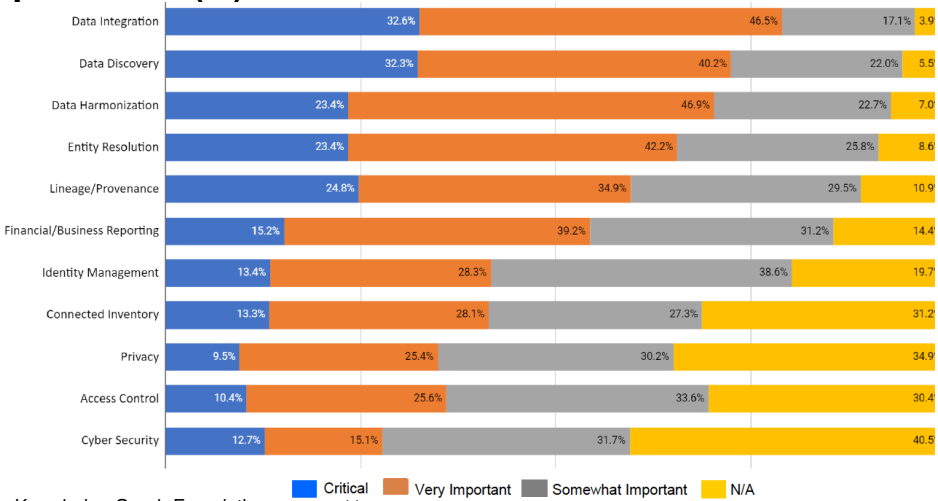
Aidan Hogan et al. Knowledge Graphs. Morgan & Claypool 2021.

Enterprise KGs (ii)

	Data model	Size of the graph	Development stage
Microsoft	The types of entities, relations, and attributes in the graph are defined in an ontology.	~2 billion primary entities, ~55 billion facts	Actively used in products
Google	Strongly typed entities, relations with domain and range inference	1 billion entities, 70 billion assertions	Actively used in products
Facebook	All of the attributes and relations are structured and strongly typed, and optionally indexed to enable efficient retrieval, search, and traversal.	~50 million primary entities, ~500 million assertions	Actively used in products
eBay	Entities and relation, well-structured and strongly typed	Expect around 100 million products, >1 billion triples	Early stages of development and deployment
IBM	Entities and relations with evidence information associated with them.	Various sizes. Proven on scales documents >100 million, relationships >5 billion, entities >100 million	Actively used in products and by clients

Natasha Noy et al. Industry-scale Knowledge Graphs: Lessons and Challenges. Communications of the ACM (2019).

Enterprise KGs (iii): Use cases



Enterprise Knowledge Graph Foundation: `https://www.ekgf.org`

How can we access the KGs?

- RDF dumps.
- From *SPARQL endpoints*:
  - Data kept in a *triple store*
  - Exposes data (in different formats)
    - with endpoint frontends, e.g.,
      `http://dbpedia.org/resource/London`,
      `https://dbpedia.org/page/City,_University_of_London` or
    - by direct SPARQL query endpoint: *e.g.*,
      `http://dbpedia.org/sparql`
      `https://query.wikidata.org/`
      `https://www.ebi.ac.uk/rdf/datasets/`

Ontology Modelling using RDF triples

Some Ontology Axioms as RDF triples (i)

A KG with a small Ontology:

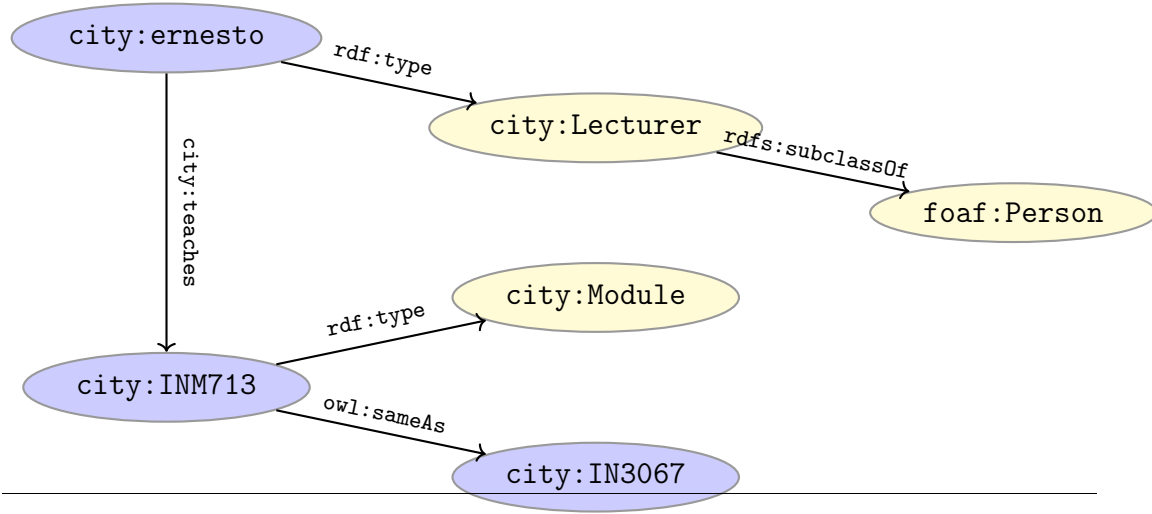
```
#Data
city:ernesto city:teaches city:INM713 .
city:IN3067 owl:sameAs city:INM713 .

#Linking data to ontology
city:INM713 rdf:type city:Module .
city:ernesto rdf:type city:Lecturer .

#Ontology
city:Lecturer rdfs:subClassOf foad:Person .
```

(*) Semantics will come on Weeks 3 and 7.

Some Ontology Axioms as RDF triples (ii)



Laboratory Session

Laboratory Session

- R201 (Franklin building) - 10 min. walk.
- This session is about creating small RDF-based KGs.

RDF in Python with RDFLib (i)

- We rely on RDFLib: `from rdflib import Graph`
- Creates empty graph: `g = Graph()`

RDF in Python with RDFLib (i)

- We rely on RDFLib: `from rdflib import Graph`
- Creates empty graph: `g = Graph()`
- Loads an RDF graph: `g.parse("beatles.ttl", format="ttl")`
- Saves and RDF graph:
  `g.serialize(destination='beatles.rdf', format='xml')`

RDF in Python with RDFLib (i)

- We rely on RDFLib: `from rdflib import Graph`
- Creates empty graph: `g = Graph()`
- Loads an RDF graph: `g.parse("beatles.ttl", format="ttl")`
- Saves and RDF graph:
  `g.serialize(destination='beatles.rdf', format='xml')`
- Iterates over a graph:

```
for s, p, o in g:
    print((s.n3(), p.n3(), o.n3()))
```

RDF in Python with RDFLib (ii)

- Basic triple elements: `from rdflib import URIRef, BNode, Literal`
- Creates an URI:
  `ernesto = URIRef("http://ex.org/univ/city#ernesto")`
- Creates a blank node: `bnode = BNode()`
- Creates a literal: `year = Literal('2021', datatype=XSD.gYear)`

RDF in Python with RDFLib (iii)

– Namespaces:

```
from rdflib import Namespace
```

RDF in Python with RDFLib (iii)

- Namespaces:
  `from rdflib import Namespace`
- Default namespaces and vocabulary: `from rdflib.namespace import OWL, RDF, RDFS, FOAF, XSD`
  - *e.g.*, RDF.type is equivalent to
    `URIRef("http://www.w3.org/1999/02/22-rdf-syntax-ns#type")`

RDF in Python with RDFLib (iii)

- Namespaces:
  `from rdflib import Namespace`
- Default namespaces and vocabulary: `from rdflib.namespace import OWL, RDF, RDFS, FOAF, XSD`
  - *e.g.*, RDF.type is equivalent to
    `URIRef("http://www.w3.org/1999/02/22-rdf-syntax-ns#type")`
- User defined:
  `city = Namespace("http://ex.org/univ/city#")`
  - *e.g.*, `city.ernesto` is equivalent to
    `URIRef("http://ex.org/univ/city#ernesto")`

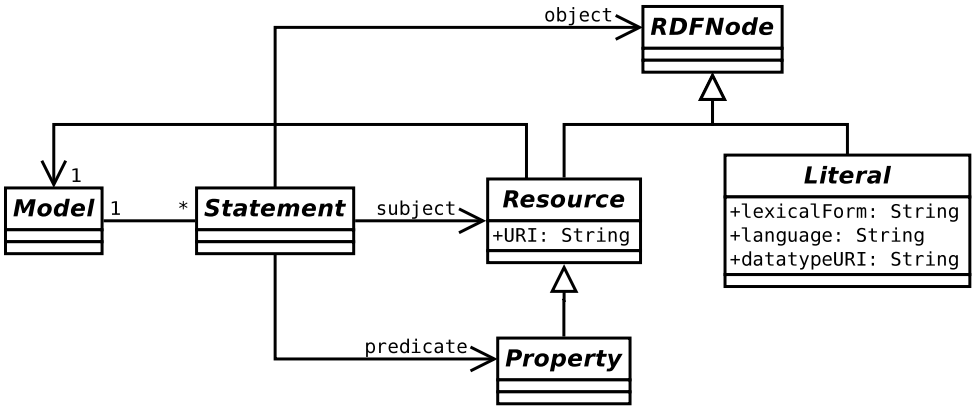
RDF in Python with RDFLib (iv)

– Adding triples:

```
g.add((city.ernesto, RDF.type, FOAF.Person))
g.add((city.ernesto, FOAF.name, name))
g.add((city.ernesto, city.teaches, city.inm713))
```

– Prefixes: `g.bind("city", city)`

RDF in Java with Jena API (i)



RDF in Java with Jena API (ii)

– Creates empty graph:

```
Model model = ModelFactory.createDefaultModel();
```

RDF in Java with Jena API (ii)

- Creates empty graph:

```
Model model = ModelFactory.createDefaultModel();
```

- Loads an RDF graph:

```
Dataset dataset = RDFDataMgr.loadDataset(file);
Model model = dataset.getDefaultModel();
```

- Saves and RDF graph:

```
OutputStream out = new FileOutputStream(output_file);
RDFDataMgr.write(out, model, RDFFormat.TURTLE);
```

RDF in Java with Jena API (ii)

- Creates empty graph:

```
Model model = ModelFactory.createDefaultModel();
```

- Loads an RDF graph:

```
Dataset dataset = RDFDataMgr.loadDataset(file);
Model model = dataset.getDefaultModel();
```

- Saves and RDF graph:

```
OutputStream out = new FileOutputStream(output_file);
RDFDataMgr.write(out, model, RDFFormat.TURTLE);
```

- Iterator over RDF statements:

```
StmtIterator iter = model.listStatements();
```

RDF in Java with Jena API (iii)

– Creates a Resource:

```
Resource ernesto_res =
model.createResource("http://ex.org/univ/city#ernesto");
```

RDF in Java with Jena API (iii)

- Creates a Resource:

```
Resource ernesto_res =
model.createResource("http://ex.org/univ/city#ernesto");
```

- Creates a blank node: `Resource blank = model.createResource();`

- Creates an Property:

```
Property teaches_prop =
model.createProperty("http://ex.org/univ/city#teaches");
```

- Creates a literal: `Literal year_lit =`
  `model.createTypedLiteral("2021", XSDDatatype.XSDgYear);`

RDF in Java with Jena API (iv)

- Default namespaces and vocabulary:

```
import org.apache.jena.datatypes.xsd.XSDDatatype;
import org.apache.jena.vocabulary.RDF;
```

  - *e.g.*, RDF.type is equivalent to

```
model.createProperty(
"http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
);
```

RDF in Java with Jena API (v)

– Adding triples:

```
model.add(ernesto_res, RDF.type, Person_res)
model.add(ernesto_res, name_prop, name_lit)
model.add(ernesto_res, teaches_prop, inm713_res)
```

– Prefixes: `model.setNsPrefix("city", "http://ex.org/univ/city#");`

Acknowledgements

Acknowledgements

- Prof. Martin Giese and others (University of Oslo)
- INF4580 – Semantic technologies
- `https://www.uio.no/studier/emner/matnat/ifi/INF4580/`