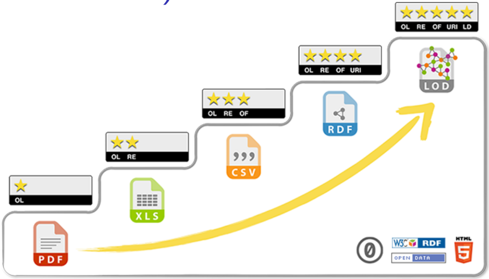
From CSV to RDF

Here be acronyms...

Julian Padget

Acknowledgements:
Grigoris Antoniou and Frank van Harmelen
"A Semantic Web Primer"

5 Star Info (Berners-Lee)



⋆ make your stuff available on the Web (whatever format) under an open license

⋆⋆ make it available as structured data (e.g., Excel instead of image scan of a table)

⋆⋆⋆ make it available in a non-proprietary open format (e.g., CSV as well as of Excel)

⋆⋆⋆⋆ use URIs to denote things, so that people can point at your stuff

⋆⋆⋆⋆⋆ link your data to other data to provide context

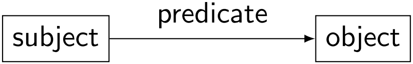
An illustration: DBpedia

- structured, semantically annotated content from Wikipedia
- semantic querying of DBpedia *and other sources*

```
import rdflib
g=rdflib.Graph()
g.load(’http://dbpedia.org/resource/Semantic_Web’)
semweb=rdflib.URIRef(’http://dbpedia.org/resource/Semantic_Web’)
dbpedia=rdflib.Namespace(’http://dbpedia.org/ontology/’)
abstracts=list(x for x in g.objects(semweb, dbpedia[’abstract’])
   if x.language==’en’)
print abstracts[0].value
abstracts=list(x for x in g.objects(semweb, dbpedia[’abstract’])
   if x.language==’ar’)
print abstracts[0].value
```

Graphs and Triples

- Basic building block: subject-predicate-object triple



- Many syntaxes, including XML
- The fundamental concepts of RDF are:
  - resources: the boxes (more later)
  - properties: the labels on arrows (more later)
  - statements: the s-p-o combination
- Graph is a set of triples
- Relations can be: 1-1, 1-many, many-1, many-many
- Query: triples that satisfy conditions, like SQL
- Resource description with user vocabularies: you define subject, predicate and object

Resources

- A resource as an object, a “thing” we want to talk about:
  - For example: house, insulation, heating system, ...
- Building block comes from web technology
- Every resource has a URI, a Universal Resource Identifier
- A URI can be:
  - A URL (Web address) or
  - some other kind of unique identifier

Properties

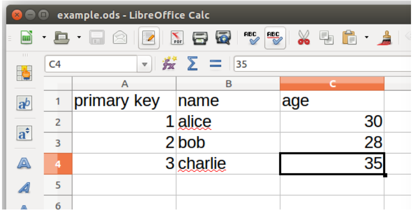
- Properties are a special kind of resources
- Used to describe relations between resources
  - For example: “type”, “construction date, “storeys”, etc.
- Properties also identified by URIs
- Advantages of using URIs:
  - Global, worldwide, unique naming scheme
  - Limits impact of homonym problem

Statements

- Statements assert the properties of resources
- A statement is an subject-predicate-object triple comprising:
  - a resource
  - a property, and
  - a value
- Literals are atomic values (strings)
- Values can be resources or literals

Converting CSV

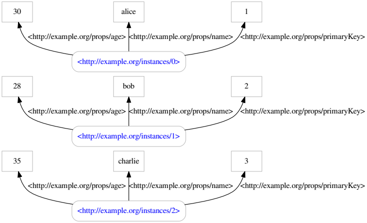
```
http://example.org/instances/0
http://example.org/props/primaryKey
1
http://example.org/instances/0
http://example.org/props/name
alice
http://example.org/instances/0
http://example.org/props/age
30
...
```



example.ods - LibreOffice Calc

C4 | 35

	A	B	C
1	primary key	name	age
2	1	alice	30
3	2	bob	28
4	3	charlie	35
5			
6			



30
alice
1
<http://example.org/props/age>
<http://example.org/props/name>
<http://example.org/props/primaryKey>
<http://example.org/instances/0>
28
bob
2
<http://example.org/props/age>
<http://example.org/props/name>
<http://example.org/props/primaryKey>
<http://example.org/instances/1>
35
charlie
3
<http://example.org/props/age>
<http://example.org/props/name>
<http://example.org/props/primaryKey>
<http://example.org/instances/2>

or as XML

- Convert csv to rdf

```
csv2rdf example.csv > example.rdf
```

- Convert rdf/n3 to rdf/xml

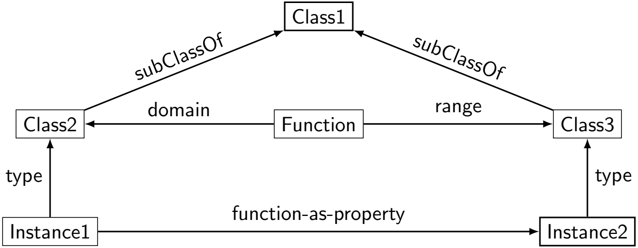
```
g=rdflib.Graph()
g.load('xls/example.rdf',format='n3')
print g.serialize()
```

- Giving

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
   xmlns:ns1="http://example.org/props/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <rdf:Description rdf:about="http://example.org/instances/0">
    <ns1:name>alice</ns1:name>
    <ns1:age>30</ns1:age>
    <ns1:primaryKey>1</ns1:primaryKey>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/instances/2">
    <ns1:name>charlie</ns1:name>
    <ns1:age>35</ns1:age>
    <ns1:primaryKey>3</ns1:primaryKey>
```

RDF Schema

- No assumed or defined domain semantics
- User-defined by Schema for:
  - Classes and Properties
  - Class Hierarchies and Inheritance
  - Property Hierarchies
- In practice:
  - Class, subClassOf, type
  - Property, subPropertyOf
  - domain, range



- Makes simple inference possible...

RDF(S) Semantics

- The (obvious) inference rules:

$$
\begin{aligned}
X\ R\ Y \wedge \mathrm{dom}(R) = T &\Rightarrow X \text{ IsOfType } T \\
X\ R\ Y \wedge \mathrm{range}(R) = T &\Rightarrow Y \text{ IsOfType } T \\
T1 \text{ SubClassOf } T2 + T2 \text{ SubClassOf } T3 &\Rightarrow T1 \text{ SubClassOf } T3 \\
X \text{ IsOfType } T1 + T1 \text{ SubClassOf } T2 &\Rightarrow X \text{ IsOfType } T2
\end{aligned}
$$

- Given a set of triples can infer other statements
  - Aspirin isOfType Painkiller + Painkiller subClassOf Drug ⇒ Aspirin isOfType Drug
  - Aspirin alleviates Headache + alleviates range Symptom ⇒ Headache isOfType Symptom
- Some triple stores do this automatically, others do not

Querying a set of triples with SPARQL

```
import rdflib
g=rdflib.Graph()
g.load(’xls/example.rdf’,format=’n3’)
for row in g.query(
        ’select␣?s␣?p␣?o␣where␣{␣?s␣?p␣?o␣.}’):
    print row.s
    print row.p
    print row.o
```

$\Longrightarrow$

```
http://example.org/instances/2
http://example.org/props/primaryKey
3
http://example.org/instances/1
http://example.org/props/name
bob
http://example.org/instances/1
http://example.org/props/age
28
http://example.org/instances/2
http://example.org/props/name
charlie
```

Querying a set of triples with SPARQL

```
import rdflib
g=rdflib.Graph()
g.load('xls/example.rdf',format='n3')
ns=dict(props=rdflib.Namespace('http://example.org/props/'))
for row in g.query(
        'select␣?s␣?o␣where␣{␣?s␣props:name␣?o␣.}',initNs=ns):
    print row.o
```

⟹

```
charlie
alice
bob
```

Triple stores: databases for triples

- A purpose-built database for the storage and retrieval of Resource Description Framework (RDF) metadata
- Implementation: special-purpose (e.g. Jena, JRDF, 4store) or on top of conventional SQL databases
- Advantage: unstructured data—no need to design table structure in advance, so can handle whatever relations are asserted
- Disadvantage: unstructured data—serious impact on performance due to absence of regularity

Ontology: DIY or re-use?

- DIY example: csv2rdf invented URIs from CSV data
  - `http://example.org/instances/`
  - `http://example.org/props/primaryKey`
  - `http://example.org/props/name`
  - `http://example.org/props/age`
- not very useful... just text labels
- FOAF = Friend-of-a-Friend, W3C-defined ontology
  - Classes: Agent, Organization, Person, ...
  - Properties: account, age, birthday, currentProject, familyName, homepage, ...
- publish ontology... re-use labels
- Writing ontologies: OWL (Web Ontology Language)
  - High-level language for classes and properties
  - Translates to RDF
  - full OWL, OWL-DL and OWL-Lite: computational complexity of reasoning
- Alignment: is my X the same as your X?

Summary

- URIs are names for resources/properties
- S-P-O triples connect resources with a property
- CSV translates easily into RDF
- RDF semantics provides simple reasoning
- Query set of triples like a database
- Triples stores for lots of triples
- Ontologies for sharing and re-using names (URIs)