UNIVERSITY OF MANNHEIM

Web Data Integration

Data Exchange Formats

- Part 2 -

Outline

1. Data Exchange Formats - Part I
    1. Character Encoding
    2. Comma Seperated Values (CSV)
    3. Extensible Markup Language (XML)
2. Data Exchange Formats - Part II
    1. JavaScript Object Notation (JSON)
        1. Basic Syntax
        2. JSON in Java
    2. Resource Description Framework (RDF)
        1. RDF Data Model
        2. RDF Syntaxes
        3. RDF Schema
        4. SPARQL Query Language
        5. RDF in Java

2.1 JavaScript Object Notation (JSON)

- JavaScript
  - a popular programming language on the Web
  - understood by all Web browsers
  - originally:
    - used for simple interactions (e.g., change image on mouse over)
  - nowadays:
    - also used for complex applications, Ajax (Asynchronous JavaScript and XML)
    - for instance used to implement Google Docs
- JSON
  - is a lightweight data exchange format that uses JavaScript syntax
  - less verbose alternative to XML
  - widely adopted
    - by Web APIs as data exchange format
    - for embedding structured data in the HEAD section of HTML pages

JavaScript Object Notation (JSON)

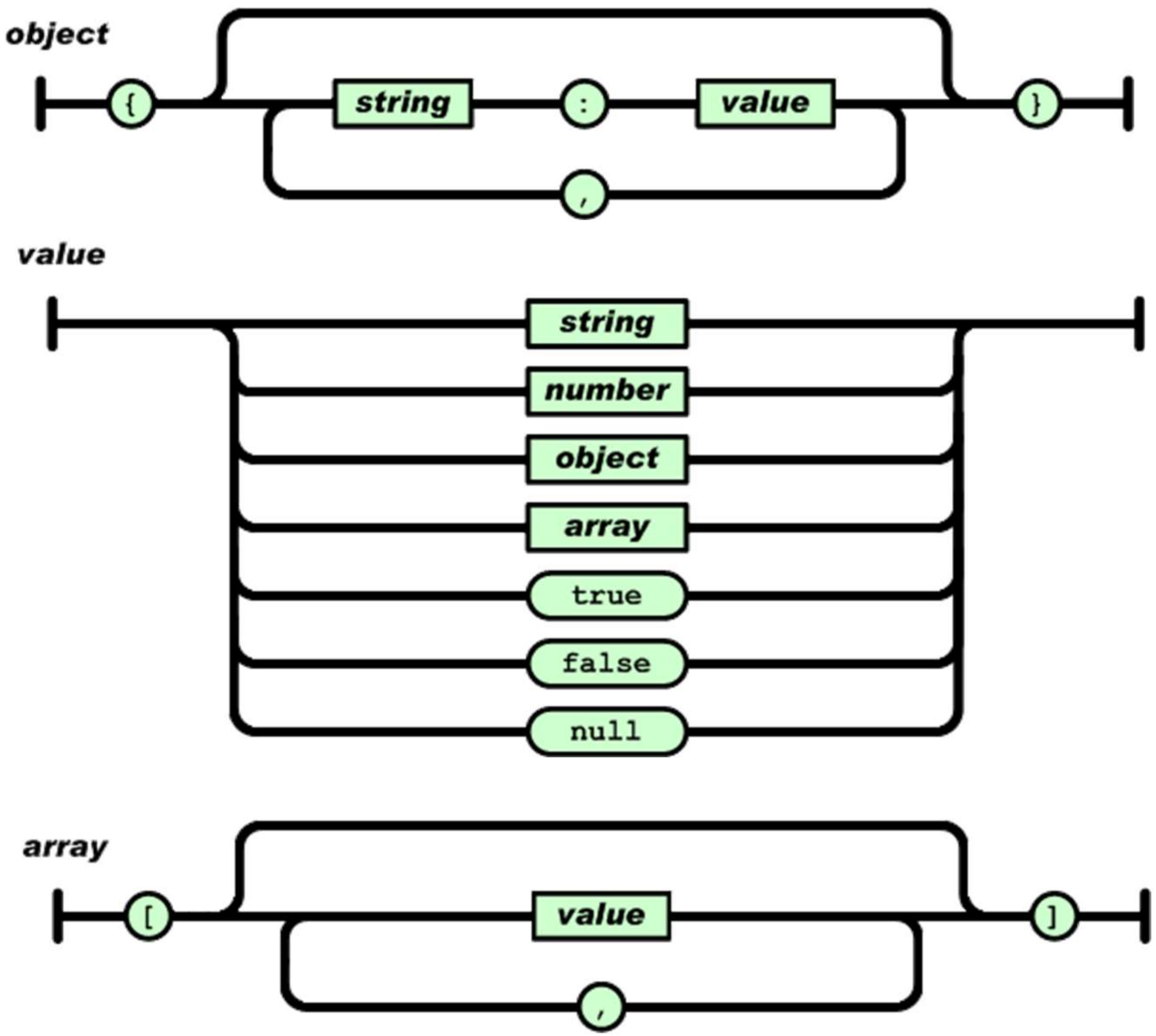
- Basics:
  - objects are noted as in JavaScript
  - objects are enclosed in curly brackets { … }
  - data is organized in key value pairs separated by colons { key : value }
- Example:

```
{ "firstname" : "John" ,
  "lastname" : "Smith" ,
  "age" : 46 }
```

- Simple processing with JavaScript:

```
var obj = JSON.parse(jsonString) ;
var name = obj.firstname + " " + obj.lastname ;
var backToString = JSON.stringify(obj)
```

The JSON Syntax



Source: json.org

Arrays in JSON

```
{ "id" : 1,
  "name" : “Good book",
  "tags" : [
     “Novel",
     “Fiction"
  ],
  "stock" : {
    "warehouse" : 300,
    "retail" : 20
  }
}
```

Nested Objects in JSON

JSON

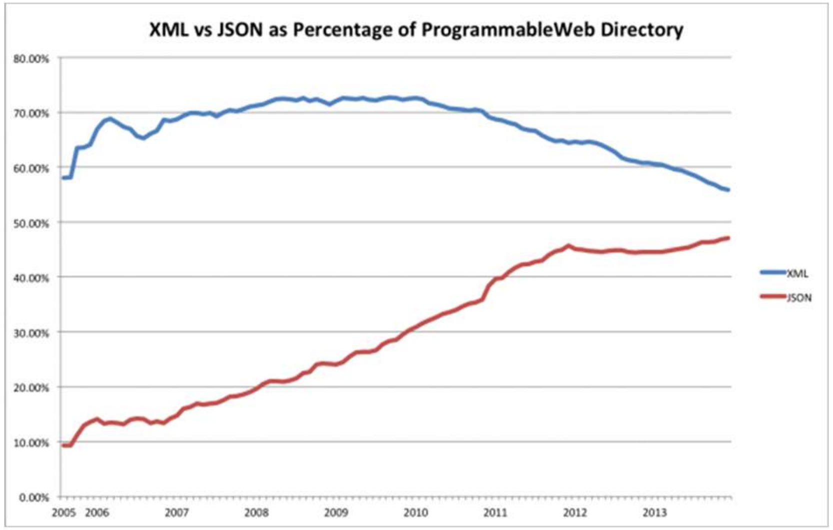
```
{ "firstname" : "John" ,
  "lastname" : "Smith" ,
  "age" : 46 ,
  "employer" : {
     "name" : "Tech Inc." ,
     "address" : {
        "street" : "Main St." ,
        "number" : 14,
        "city" : "Smalltown"
     }
  }
}
```

XML

```
<firstname>John</firstname>
<lastname>Smith</lastname>
<age>46</age>
<employer>
   <name>Tech Inc.</name>
   <address>
     <street>Main St.</street>
     <number>14</number>
     <city>Smalltown</city>
   </address>
</employer>
```

JSON versus XML

- JSON is a lot like XML
  - data model: tree
  - opening/closing tags/brackets
- Differences
  - more compact notation compared to XML
  - no id/idref – JSON data is *strictly* tree shaped
  - less data types (only string, number, and Boolean)
- Adoption
  - XML: Wider adoption in enterprise context
  - JSON: Wider adoption in Web context
  - Programmable Web 2019:
    - 2800 XML APIs vs. 5400 JSON APIS



Processing JSON with Java

- GSON
  - Library for parsing and serializing JSON in Java
  - https://github.com/google/gson
- Class Definition

```
public class Person {
  private String firstname;
  private String lastname;
  private int age;
}
```

```
{ "firstname" : "John" ,
  "lastname" : "Smith" ,
  "age" : 46
}
```

- Object Deserialization

```
Person obj = gson.fromJson(jsonString, Person.class);
```

- Object Serialization

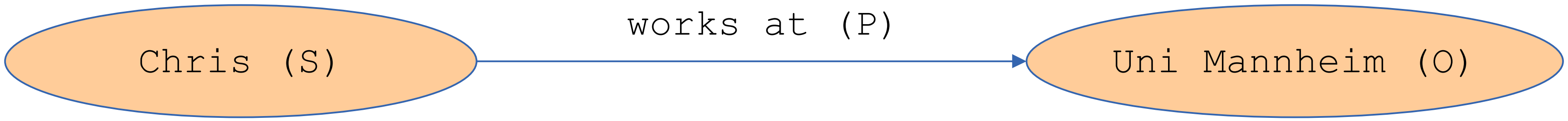
```
String json = gson.toJson(obj);
```

2.2 Resource Description Framework (RDF)

Graph data model designed for sharing data on the Web.

- Applications:
  - annotation of Web pages (RDFa, JSON-LD)
  - publication of data on the Web (Linked Data)
  - exchange of graph data between applications
- View 1: Sentences in form Subject-Predicate-Object (called Triples)

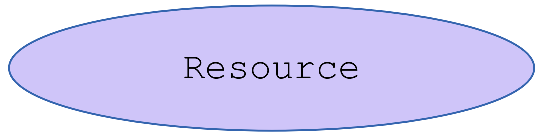
„Chris works at University of Mannheim."



- View 2: Labeled directed graph
  - A set of RDF triples forms a labeled directed graph

RDF Basic Concepts

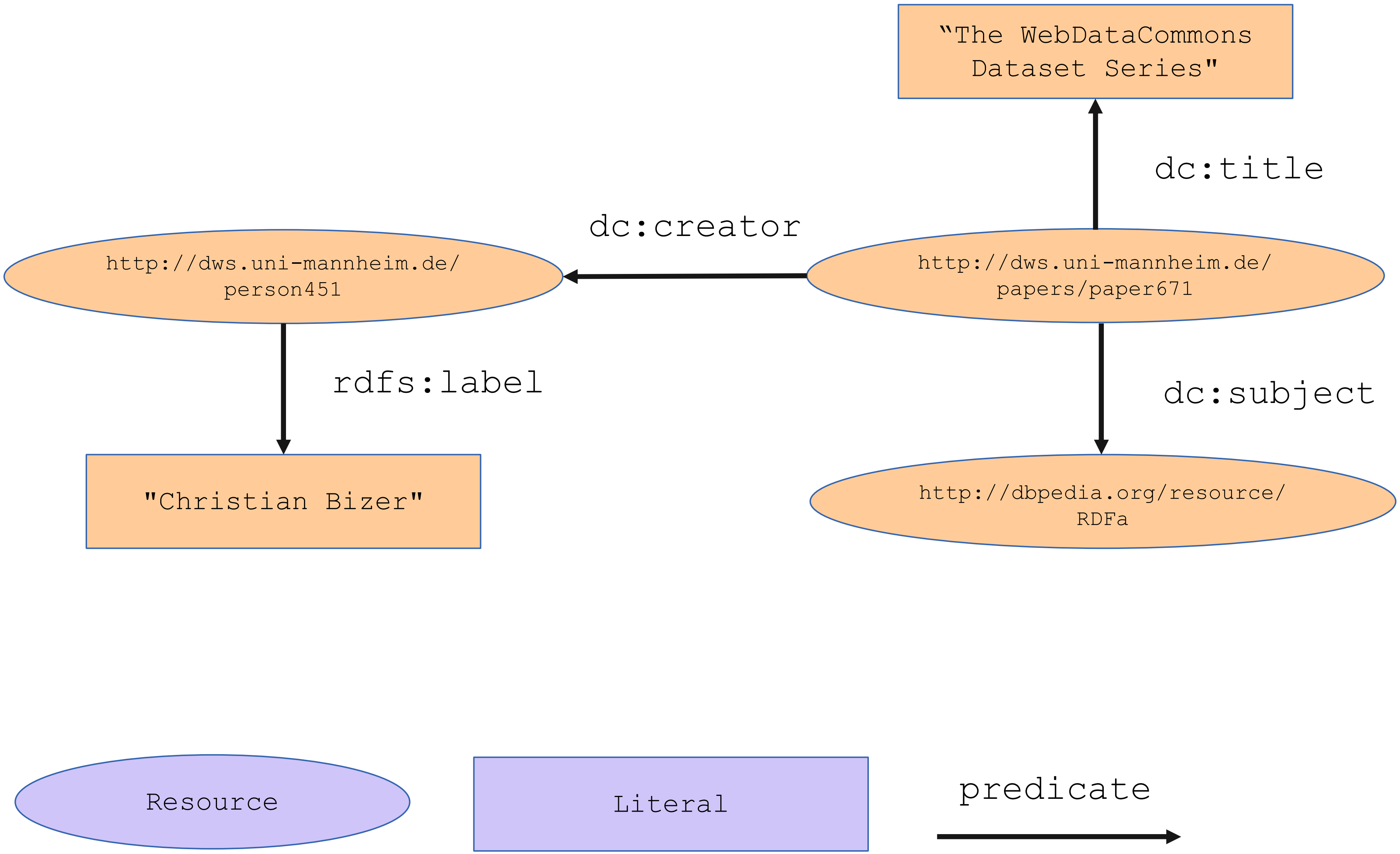
- Resources
  - everything (a person, a place, a web page, …) is a resource
  - are identified by URI references
  - may have one or more types (e.g. foaf:Person)



- Literals
  - are data values, e.g., strings or integers
  - may only be objects, not subjects of triples
  - may have a data type or a language tag

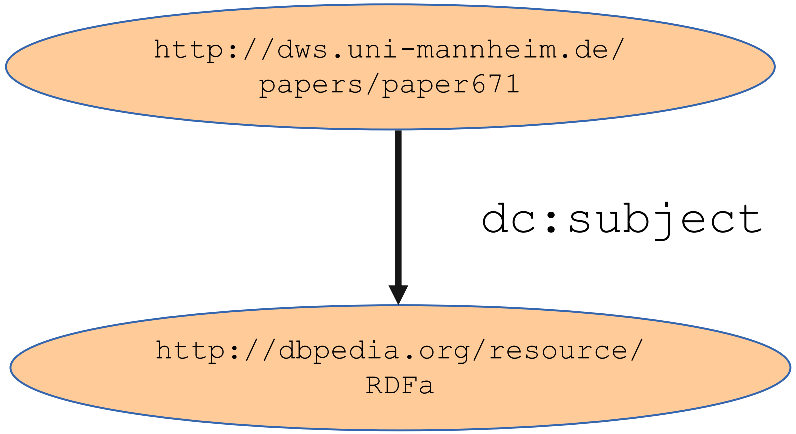
- Predicates (Properties)
  - connect resources to other resources
  - connect resources to literals

RDF as a Labeled Directed Graph



The Role of URIs in RDF

– In a typical database or XML document, identifiers are unique only with respect to the database or XML document.
  - they have no meaning outside the database/document
– RDF uses URI's as global identifiers for resources
  - hence, all data is connected to its origin
  - multiple data sets can refer to each other
  - lays the foundation for a global data space
– Advantage
  - global references between data items are possible (Linked Data)



– Disadvantage
  - RDF is rather verbose.
  - ➔ most syntaxes use QNames (e.g. dc:subject).

Language Tags and Data Types

- RDF literals may have language tags or data types (but not both)
- Examples:

Literal

ex:Muenchen ex:hasName "München"@de .
ex:Muenchen ex:hasName "Munich"@en .
ex:Muenchen ex:hasPopulation "1356594 "^^xsd:integer .
ex:Muenchen ex:hasFoundingYear "1158-01-01"^^xsd:date .

- RDF uses the XML Schema data types
- Be careful, the following three literals are different:
  - "München"
  - "München"@de
  - "München"^^xsd:string

RDF Syntaxes

There are various syntaxes for serializing RDF graphs.

1. N-Triples and Turtle: Plain text syntaxes
2. RDF/XML: RDF serialization in XML
3. RDFa: Syntax for embedding RDF into HTML pages
4. JSON-LD: RDF serialization in JSON

N-Triples and Turtle

- N-Triples is a line-based, plain text serialization format for RDF graphs

```
<http://www.dws.uni-mannheim.de/teaching/wdi>
<http://purl.org/dc/elements/1.1/subject>
<http://dbpedia.org/resource/RDFa> .

<http://www.dws.uni-mannheim.de/teaching/wdi>
<http://purl.org/dc/elements/1.1/title>
"Web Data Integration"@en .
```

Point marks end of triple

URIs are enclosed by <>

Literals enclosed by " "

- Turtle extends N-Triples with QNames

```
@BASE <http://www.dws.uni-mannheim.de/teaching/>
@PREFIX dc: <http://purl.org/dc/elements/1.1/>
@PREFIX dbpedia: <http://dbpedia.org/resource/>

:wdi dc:subject dbpedia:RDFa .

:wdi dc:title "Web Data Integration"@en .
```

Empty prefix refers to BASE namespace

RDF/XML

- XML-based serialization format for RDF

- Describing resources:

```
<rdf:Description rdf:about="http://www.dws.uni-mannheim.de/teaching/wdi">
    <dc:creator>Christian Bizer</dc:creator>
</rdf:Description>
```

- Resource with a type:

```
<rdf:Description rdf:about="http://www.dws.uni-mannheim.de/teaching/wdi">
    <rdf:type rdf:resource="http://www.dws.uni-mannheim.de/teaching/Course"/>
    <dc:creator>Christian Bizer</dc:creator>
</rdf:Description>
```

- Alternative notation:

```
<dws:Course rdf:about="http://www.dws.uni-mannheim.de/teaching/wdi" />
```

JSON-LD

- JSON syntax for RDF used for embedding RDF into the HEAD section of HTML pages

```
<script type="application/ld+json">
{
  "@context": "http://schema.org/",
  "@type": "Organization",
  "url": "http://www.example.com",
  "name": "Unlimited Ball Bearings Corp.",
  "contactPoint": {
    "@type": "ContactPoint",
    "telephone": "+1-401-555-1212",
    "contactType": "Customer service"
  }
}
</script>
```

https://json-ld.org/
https://developers.google.com/search/docs/guides/intro-structured-data

2.3 RDF Schema

Language for defining RDF vocabularies.

- RDF schema provides for defining:
  - classes (that are used as types) and
  - properties (that are used as predicates)

- Example of a RDF schema vocabulary definition:

```
dws:Teacher rdf:type rdfs:Class .
dws:Course rdf:type rdfs:Class .
dws:teaches rdf:type rdf:Property .
```

- RDF triples using the vocabulary:

```
dws:ChrisBizer rdf:type dws:Teacher .
dws:WebDataIntegration rdf:type dws:Course .
dws:ChrisBizer dws:teaches dws:WebDataIntegration .
```

Classes and Properties may form Hierarchies

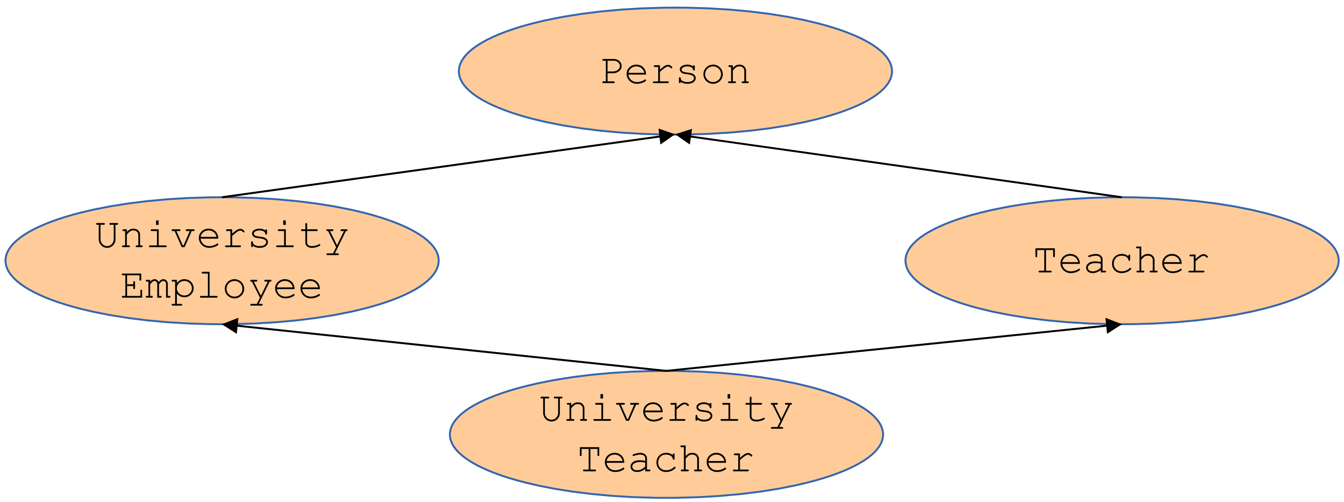
- Subclass Definition

```
dws:UniversityTeacher rdfs:subClassOf dws:Teacher .
```

- Subproperty Definition

```
dws:CourseName rdfs:subPropertyOf dc:title .
```

- Implication: All dws:UniversityTeachers are also dws:Teachers
- Multiple inheritance is allowed



Domain and Range Definitions

- RDF Schema provides for defining domains and ranges of properties:

```
dws:teaches rdf:type rdf:Property .
dws:teaches rdfs:domain dws:Teacher .
dws:teaches rdfs:range dws:Course .
```

- Implications:
  1. All resources that have a `dws:teaches` property are of `rdf:type dws:Teacher`.
  2. All objects of `dws:teaches` triples are of `rdf:type dws:Course`.

- Domains and ranges are inherited to subproperties

RDF Schema Reasoning

- Given the RDF schema

```
dws:Teacher rdfs:subClassOf foaf:Person .
dws:teaches rdfs:domain dws:Teacher .
dws:teaches rdfs:range dws:Course .
```

- and the single triple

```
dws:ChrisBizer dws:teaches dws:WebDataIntegration .
```

- A machine (reasoning engine) can infer (conclude) that

```
dws:ChrisBizer rdf:type dws:Teacher .
dws:ChrisBizer rdf:type foaf:Person .
dws:WebDataIntegration rdf:type dws:Course .
```

- OWL (Web Ontology Language)
  - provides for more expressive definitions and inferences
  - see course: IE650 Knowledge Graphs

Purpose of RDF Schema

- Recap: XML Schema defines *allowed* structures
- In contrast: RDF Schema does *not* constrain anything

- Purpose of XML Schema
  - validation of XML documents
- Purpose of RDF Schema
  - machine interpretability of RDF data
    - by inferring additional triples
    - by setting links (correspondences) between different RDF terms
      e.g. `dws:Teacher rdfs:subClassOf foaf:Person`
  - NOT validation
  - W3C SHACL Shapes Constraint Language provide for RDF validation
    https://www.w3.org/TR/shacl/

2.4 SPARQL

Language for querying RDF graphs.

- Queries are expressed in the form of triple patterns
- Query results are tabular and given as XML, JSON, or CSV
- The SPARQL Protocol is used to query remote endpoints
- Example query:

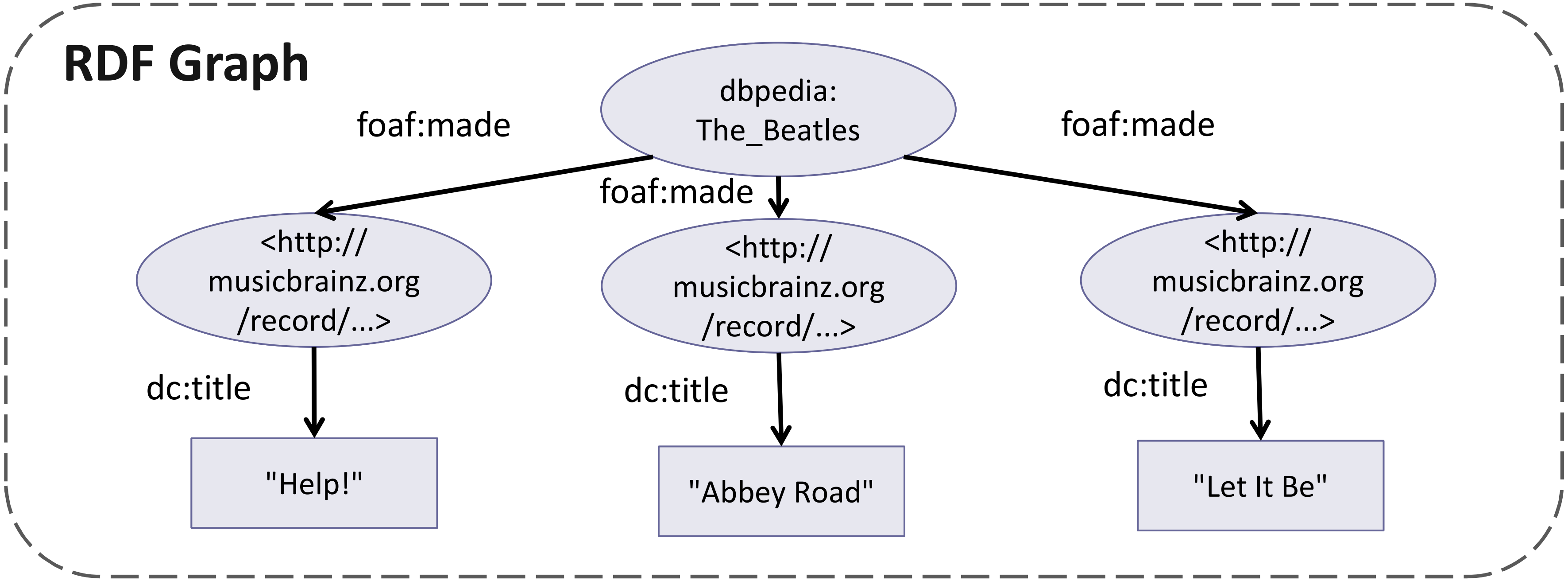
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person rdf:type foaf:Person .
  ?person foaf:name ?name .
  ?person foaf:mbox ?email .
}
```

Prefix definition

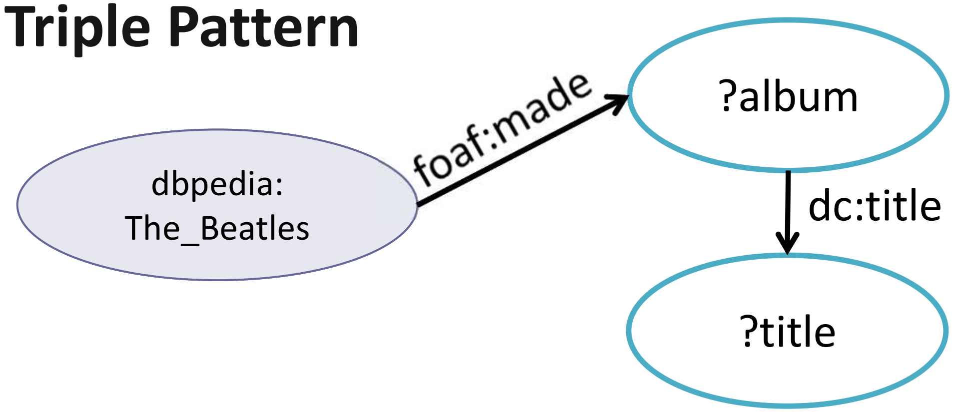
Result definition

Triple patterns
(?x = variables)

Triple Pattern Matching



Triple Pattern



Query Result

?album	?title
<http://...>	"Help!"
<http://...>	"Abbey Road"
<http://...>	"Let It Be"

Source: EUCLID - Querying Linked Data

Optional Triple Patterns

- Declaring triple patterns as OPTIONAL allows you to get query results even if only a subset of the patterns matches

```
WHERE { A OPTIONAL { B } }
```

- Keep all solutions from A whether or not there is a matching solution for B
- Important for querying endpoints with a lot of missing values
- Example:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?name ?birth ?death
WHERE {
     ?person foaf:name ?name .
     ?person dbo:birthDate ?birth .
     OPTIONAL { ?person dbo:deathDate ?death . }

}
```

FILTER Clauses

- FILTER clauses keep only solutions that fulfil a condition (expression must evaluate to true)
- Example

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?name ?birth
WHERE {
        ?person dbo:birthPlace :Berlin .
        ?person dbo:birthDate ?birth .
        ?person foaf:name ?name .
        FILTER (?birth < "1900-01-01"^^xsd:date) .
}
```

- Comparators: `=  !=  <  >  <=  >=`
- Logical Operators: `&&  ||  !`
- Functions: SUBSTR(), regex(), month(now()), isURI(), …
  - more functions: http://www.w3.org/TR/sparql11-query/#SparqlOps

Solution Modifiers

- Sort results

  `ORDER BY ?name`

- Restrict number of results

  `LIMIT 100`

- Page over result list

  ```
  LIMIT 100
  OFFSET 0
  ```

  ```
  LIMIT 100
  OFFSET 100
  ```

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?name ?birth
WHERE {
        ?person dbo:birthPlace :Berlin .
        ?person dbo:birthDate ?birth .
        ?person foaf:name ?name .
}
ORDER BY ?name
LIMIT 10
OFFSET 100
```

Exercise: Querying DBpedia 1

- Question 1: What is the population and the area code of Mannheim?
  - http://dbpedia.org/resource/Mannheim
- Query tool
  - http://dbpedia.org/snorql/

2.5 Processing RDF in Java: Jena

- Jena is a popular framework for processing RDF in Java
- Download: https://jena.apache.org/
- Capabilities
  - supports various RDF syntaxes
  - SPARQL query language
  - RDF Schema and OWL reasoning
  - various storage back ends
- Central concepts
  - model (i.e., RDF graphs): `class Model`
  - resource: `class Resource`

Processing RDF in Java: Jena

- Read a graph from a URL (or local file):

```
model.read("http://dbpedia.org/resource/Mannheim");
```

- Navigating through a model

```
Resource mannheim =
     model.getResource("http://dbpedia.org/resource/
                              Mannheim");

Literal areaCode =  mannheim.getProperty(
    "http://dbpedia.org/ontology/areaCode")
    .getLiteral();
```

Querying a Local Model with SPARQL

```
String queryString = "SELECT ?x ...";

Query query = QueryFactory.create(queryString);

QueryExecution qexec =
    QueryExecutionFactory.create(query, model);

ResultSet results = qexec.execSelect();

while(results.hasNext()) {

  QuerySolution sol = results.next();
    String s = sol.get("x").toString();
    ...

}
```

Querying a Public SPARQL Endpoint

- Many RDF data sources provide SPARQL endpoints
  - e.g. DBpedia, Linked GeoData, EU Open Data Portal, …
  - List of public endpoints: https://labs.mondeca.com/sparqlEndpointsStatus/index.html

- Access with Jena

```
String queryString = "SELECT ...";
String endpoint = "http://dbpedia.org/sparql";
Query query = QueryFactory.create(queryString);
QueryExecution qexec =
    QueryExecutionFactory.sparqlService(endpoint, query);
ResultSet RS = qexec.executeSelect();
```

Exercice on Data Exchange Formats online

– Task: Access and query data in
  - XML, XPath
  - JSON
  - RDF and SPARQL

  using the introduced Java libraries

https://www.uni-mannheim.de/dws/teaching/course-details/courses-for-master-candidates/ie-670-web-data-integration/#c149382

Wrap-up: Data Exchange Formats

- Data is provided on the Web using various exchange formats
  – CSV, XML, JSON, RDF, ….
- Exchange formats provide us with syntaxes for transferring data
- Exchange formats do not solve the actual data integration challenges:
  1. Do two records describe the same real-world entity?
  2. Which elements in different schemata have the same meaning?
  3. Which conflicting values from different sources should I trust?
- These challenges will be the topics of the upcoming lectures
- Still, which format should I use for my application?
  - Answer depends more on social than on technical factors:
    - What formats are already used by others in wider application domain?
    - What formats can the programming language of choice read and write out of the box?

3. References

– Standards and specifications
  - JSON: http://www.json.org/
  - RDF: http://www.w3.org/TR/rdf11-concepts/
  - RDF Schema: http://www.w3.org/TR/rdf-schema/
  - SPARQL: http://www.w3.org/TR/sparql11-overview/

– Tutorials
  - GSON: https://github.com/google/gson
  - RDF: https://www.w3.org/TR/rdf-primer/
  - JENA: http://jena.apache.org/documentation/
  - Euclid Curriculum covering SPARQL: http://www.euclid-project.eu/

– Course
  - IE 650 Knowledge Graphs