Aalto University
School of Science

UNIVERSITY OF HELSINKI

SeCo

HELDIG
Helsinki Centre for Digital Humanities

Creating LOD from databases

Annastiina Ahola, Heikki Rantala

Converting Data to RDF

- **Converting good, clean structured data is pretty easy, but data is rarely that good.**
- **What do you need to consider?**
  - Controlled vocabularies
  - URIs (identifiers)
  - Data models / ontologies
  - …

Tools for Converting Data to RDF

- **Programming libraries**
  - RDFLib (Python), Jena (Java)...
- **RDF Mapping Language (RML)**
- **OpenRefine**
- **Lots of other tools…**

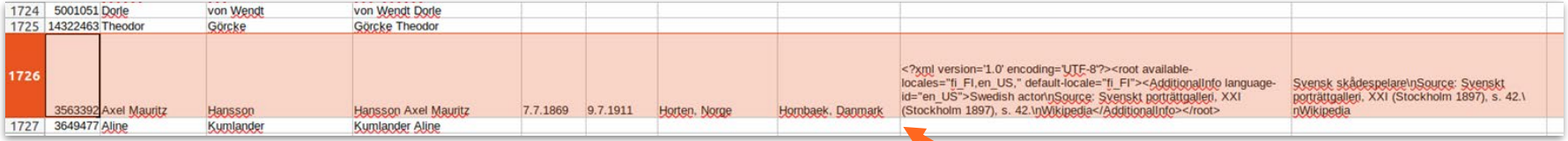
Example data: OperaSampo

- **Evening-specific data on opera and music theatre performances in Finland 1830–1960**
  - Information on compositions, performances, venues as well as people involved in the performances (opera singers, directors, etc.)
  - approx. 9,500 performances, 4,700 people, 750 compositions, 7,000 role characters and 130 performance venues
- **Originally part of a SQL-based Reprises database**
  - Running on legacy software and would likely soon cease to function
    → Linked Data service and portal as the new solution

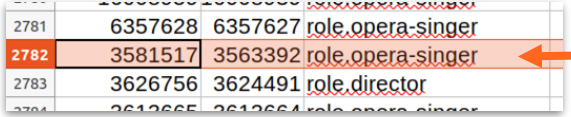
OperaSampo transformation process

- **Data from the SQL database as a dump of CSV files**
  - Each CSV file corresponding to one table in the original database
    - *18 CSV files (/tables) total → 16 relevant*
- **Data conversion from CSV files to RDF done using Python scripts**
- **The script iterates over each row in each of the relevant CSV files and extracts the values in the columns of each file**
  - URIs are formed based on the identifiers available in the original data
  - For XML formatted fields, the textual content and language information is extracted and stored as a string literal with the relevant language tag attached

CSV example: OperaSampo



1724	5001051	Dorle	von Wendt	von Wendt Dorle						
1725	14322463	Theodor	Görcke	Görcke Theodor						
1726	3563392	Axel Mauritz	Hansson	Hansson Axel Mauritz	7.7.1869	9.7.1911	Horten, Norge	Hornbaek, Danmark	<?xml version='1.0' encoding='UTF-8'?><root available-locales="fi_FI,en_US," default-locale="fi_FI"><AdditionalInfo language-id="en_US">Swedish actor\nSource: Svenskt porträttgalleri, XXI (Stockholm 1897), s. 42.\nWikipedia</AdditionalInfo></root>	Svensk skådespelare\nSource: Svenskt porträttgalleri, XXI (Stockholm 1897), s. 42.\nWikipedia
1727	3649477	Aline	Kumlander	Kumlander Aline						



2781	6357628	6357627	role.opera-singer
2782	3581517	3563392	role.opera-singer
2783	3626756	3624491	role.director

foc_PersonRole.csv

foc_Person.csv

```
personId|firstName|lastName|displayName|dateOfBirth|dateOfDeath|placeOfBirth|placeOfDeath|additionalInfo|editorNotes
...
3563392|Axel Mauritz|Hansson|Hansson Axel Mauritz|7.7.1869|9.7.1911|Horten, Norge|Hornbaek, Danmark|"<?xml version='1.0' encoding='UTF-8'?><root available-locales=""fi_FI,en_US,"" default-locale=""fi_FI""><AdditionalInfo language-id=""en_US"">Swedish actor\nSource: Svenskt porträttgalleri, XXI (Stockholm 1897), s. 42.\nWikipedia</AdditionalInfo></root>"|Svensk skådespelare\nSource: Svenskt porträttgalleri, XXI (Stockholm 1897), s. 42.\nWikipedia
...
```

foc_Person.csv

```
personRoleId|personId|role
...
3581517|3563392|role.opera-singer
...
```

foc_PersonRole.csv

```
ops:persons_3563392 a scop:Person ;
                    scop:additionalInfo "Swedish actor<br>Source: Svenskt porträttgalleri,
                                         XXI (Stockholm 1897), s. 42.<br>Wikipedia"@en ;
                    scop:dateOfBirth "7.7.1869" ;
                    scop:dateOfDeath "9.7.1911" ;
                    scop:editorNotes "Svensk skådespelare<br>Source: Svenskt porträttgalleri,
                                         XXI (Stockholm 1897), s. 42.<br>Wikipedia" ;
                    scop:placeOfBirth "Horten, Norge" ;
                    scop:placeOfDeath "Hornbaek, Danmark" ;
                    scop:role ops:occupation_roles_opera-singer ;
                    skos:prefLabel "Hansson Axel Mauritz"@fi ;
                    foaf:firstName "Axel Mauritz" ;
                    foaf:surname "Hansson" .
```

RDF

Tutorial example data

- **Small subset of data extracted from the original OperaSampo data for tutorial purposes**
  - Limited to the compositions of four different composers and their performers as well as related data (venues, producers, people)
    → approx. 40 performances, 90 people
- **Data and scripts as well as a portal setup available on GitHub: https://github.com/SemanticComputing/sampo-lod-tutorial**
  - Data and scipts in the ‘create-lod’ folder: https://github.com/SemanticComputing/sampo-lod-tutorial/tree/main/create-lod

Tutorial example data

- **Requirements for running the scripts and setting up a local Fuseki SPARQL server**
  - Python
    - *Libraries:*
      - RDFLib (creating RDF data)
      - pandas (reading CSV files, alternative library: csv)
      - untangle (parsing XML to Python objects)
      - BeautifulSoup (extracting formatted data from XML)
  - Docker (for running the Fuseki SPARQL server)

Example code: Imports

```
# import necessary libraries
import pandas as pd # for JSON data: import json
import untangle as ut
from bs4 import BeautifulSoup
from rdflib import Namespace, URIRef, Literal, Graph, RDF, RDFS, XSD, FOAF
# define namespaces for easier use (e.g., SCOP.Object = <http://ldf.fi/schema/operasampo/Object>)
SKOS = Namespace('http://www.w3.org/2004/02/skos/core#')
OPS = Namespace('http://ldf.fi/operasampo/')
SCOP = Namespace('http://ldf.fi/schema/operasampo/')
...
```

Example code: Graph creation

```
# create a new graph
graph = Graph()
# bind relevant namespaces
graph.bind('rdf', 'http://www.w3.org/1999/02/22-rdf-syntax-ns#')
graph.bind('skos', 'http://www.w3.org/2004/02/skos/core#')
graph.bind('ops', 'http://ldf.fi/operasampo/')
graph.bind('scop', 'http://ldf.fi/schema/operasampo/')
graph.bind('foaf',  'http://xmlns.com/foaf/0.1/')
...
```

Example code: Reading files

```
...
# open your source data file and load its data
performances_file =
pd.read_csv('../csv/foc_Performance.csv', sep=";",
dtype=object)
# iterate over the rows in the CSV file
for i, row in performances_file.iterrows():
        # extract data from the row ...
    handle_performance_row(graph, row)
...
```

JSON

```
with open('objects.json') as file:
    data = json.load(file)
# iterate over the objects
for item in data:
    # extract data from the object
    base = 'http://example.org/o'
    id = str(item['objectId'])
    art_object = URIRef(base + id)
```

Example code: Handling rows

```
def handle_performance_row(graph, row):
            # form URI for the performance using the value in the performanceId column
            performanceURI = URIRef('http://ldf.fi/operasampo/performances_' + str(row['performanceId']))
            graph.add((performanceURI, RDF.type, SCOP.Performance))

            # add reference to the original composition with the value in the compositionId column
            if not pd.isna(row['compositionId']):
                        # form URI for the composition
                        # (this should match the URI formulation in the composition conversion script)
                        compositionURI = URIRef('http://ldf.fi/operasampo/compositions_' +
str(row['compositionId']))
                        graph.add((performanceURI, SCOP.composition, compositionURI))

            # get the data from all the other columns ...
            ...
```

Example code: Serializing

```
...
# serialize the graph when you're done to save it (in this
# case a file named performances.ttl' to the folder ttl/)
g.serialize('ttl/performances.ttl', format='turtle')
```

Local Fuseki setup for querying data

- **You can use a Apache Jena Fuseki (https://jena.apache.org/documentation/fuseki2/) SPARQL server with Docker for serving your data locally**
  - This enables you to query your own created data locally with SPARQL similarly to an online endpoint and use it for analysis
    - *More on how to utilize SPARQL with LOD later in "Using the LOD service via SPARQL and Notebooks, data analysis, network analysis, and visualizations"*

Conclusions

- **Your data drives the process: What *is* in your data?**
  - Use actual example cases from your data to help you think about the data model and conversion process – your data doesn't have to be "beautiful" to be useful and you can always adjust things (e.g., different modeling decisions) later
- **Work in steps: Start small and iterate**
  - You can first include only some properties in your conversion
    - *Does the output look correct? Are there any issues with the data that might need some preprocessing, for example?*
  - Continue expanding your conversion process to include other properties when the previous ones are OK

Conclusions

- **Validating your data**
  - Setting up an user interface could help you see mistakes in data (e.g., missing values or something being overrepresented)
  - Formal validation with SHACL, ShEx
  - Domain-specific validation
    - *8-star model – is your data truthful?*
      - e.g., person participating in an event after their death or a start date being later in time than an end date