The background features a complex network graph with numerous nodes and edges, rendered in a light gray color. A vertical bar with a color gradient from blue at the top to red at the bottom is positioned on the left side of the image. The text is white and set against a solid black background on the right.

Knowledge Graph Deconstruction

- Suryalaxmi Ravianandan

Outline

- Introduction
 - Knowledge graph
 - Problem statement
 - RML Mapping and RDF data
- Workflow
 - Fundamental Aspect
 - Technical aspect
- Demo
- Challenges
- Further enhancements

Introduction – Knowledge Graph

Knowledge Graphs (KGs) are abilities for representing and linking data in a way that captures the relationships and context, making them ideal for applications like search, recommendation systems, and data integration.

➤ Structure:

- ✓ Nodes (Entities): Represent real-world concepts.
- ✓ Edges (Relationships): Define connections between entities.
- ✓ Graph-Based Format: Uses RDF (Resource Description Framework) with subject-predicate-object triples.

➤ Key Features:

- Data Integration from Multiple Sources
- Semantic Relationships for Better Understanding

Introduction - Problem Statement

Challenge:

RDF data is not easily accessible or usable for data analysts who typically work with tabular formats like CSV or JSON.

➤ Key Issues:

- ✓ Lack of Compatibility – Traditional data analysis tools (spreadsheets, visualization software) do not support RDF natively.
- ✓ Limited Accessibility – Analysts struggle to work with RDF due to its complex structure.
- ✓ Gap in Workflows – Disconnect between the semantic web and conventional data analysis methods.

Impact of this gap:

The adoption of KGs is hindered because they require specialized knowledge to use effectively.

Introduction – Knowledge Graph Deconstruction

Knowledge Graph Deconstruction is the process of converting RDF (Resource Description Framework) data back into tabular formats like CSV.

How It Works:

- ✓ Converts **subject-predicate-object** RDF triples into structured rows and columns.
- ✓ “Flattens” the graph structure while preserving semantic relationships.

Goal :

Develop an interpreter that Converts RDF data back into CSV format.

Reversing the Knowledge Graph construction process helps bridge the gap between the **semantic web** and **traditional data analysis workflows..**

Understanding RML Mapping Document

RML Mapping:

- ✓ RML is a declarative language used to map heterogeneous data sources, such as CSV, JSON, or XML, into RDF format.
- ✓ RML simplifies the process of transforming raw data into RDF triples, enabling seamless integration into Knowledge Graphs.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

```
@prefix rml: <http://w3id.org/rml/> .
```

```
<http://example.com/base/TriplesMap1> a  
rml:TriplesMap;
```

```
  rml:logicalSource [ a rml:LogicalSource;  
    rml:referenceFormulation rml:CSV;  
    rml:source [ a rml:RelativePathSource;  
      rml:root rml:MappingDirectory;  
      rml:path "student.csv"  
    ]  
  ];
```

```
  rml:predicateObjectMap [  
    rml:objectMap [  
      rml:reference "Name"  
    ];  
    rml:predicate foaf:name  
  ];
```

```
  rml:subjectMap [  
    rml:template "http://example.com/{Name}"  
  ] .
```

Understanding RDF Dataset

RDF Data:

- ✓ RDF represents data as triples: <subject><predicate><object> .
- ✓ These triples can be stored in formats like N-Triple, N-Quads, Turtle, etc.,
- ✓ SPARQL is the standard query language used to retrieve and manipulate data stored in RDF format.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
@prefix rml: <http://w3id.org/rml/> .
```

```
<http://example.com/base/TriplesMap1> a  
rml:TriplesMap;
```

```
  rml:logicalSource [ a rml:LogicalSource;  
    rml:referenceFormulation rml:CSV;  
    rml:source [ a rml:RelativePathSource;  
      rml:root rml:MappingDirectory;  
      rml:path "student.csv"  
    ]  
  ]  
];
```

```
  rml:predicateObjectMap [  
    rml:objectMap [  
      rml:reference "Name"  
    ]  
    ;  
    rml:predicate foaf:name  
  ]  
  rml:subjectMap [  
    rml:template "http://example.com/{Name}"  
  ] .
```

If student.csv contains:

Name
Alice
Bob

RDF Data: <http://example.com/Alice> foaf:name "Alice" .
 <http://example.com/Bob> foaf:name "Bob" .

Process for Transforming RDF Data into CSV

➤ Input:

RDF file (data.nq).

RML mapping file (mapping.ttl).

➤ Data Retrieval:

1. With RDF (rdf.nq) + RML (rml.ttl):

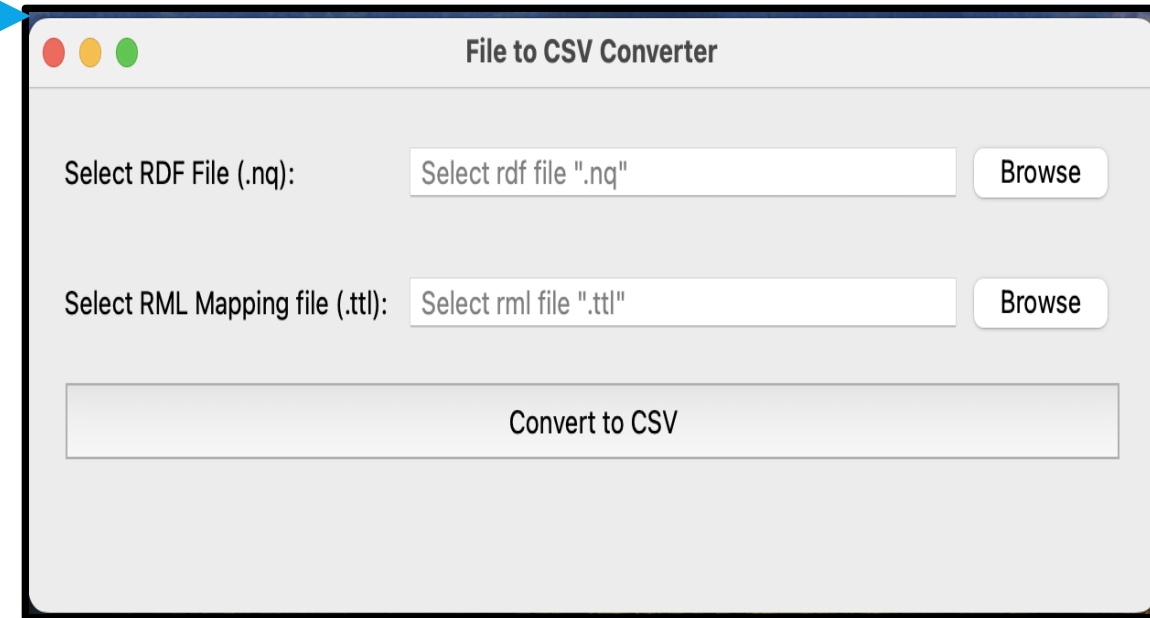
The RML mapping provides the logic to transform RDF triples into CSV rows and columns.

2. With SPARQL:

Use a SPARQL query to retrieve the specific data needed and then format it into CSV.

➤ Output:

A structured CSV file (output.csv).



Process for Transforming RDF Data into CSV

➤ Input:

RDF file (data.nq).

RML mapping file (mapping.ttl).

➤ Data Retrieval:



1. With RDF (rdf.nq) + RML (rml.ttl):

The RML mapping provides the logic to transform RDF triples into CSV rows and columns.

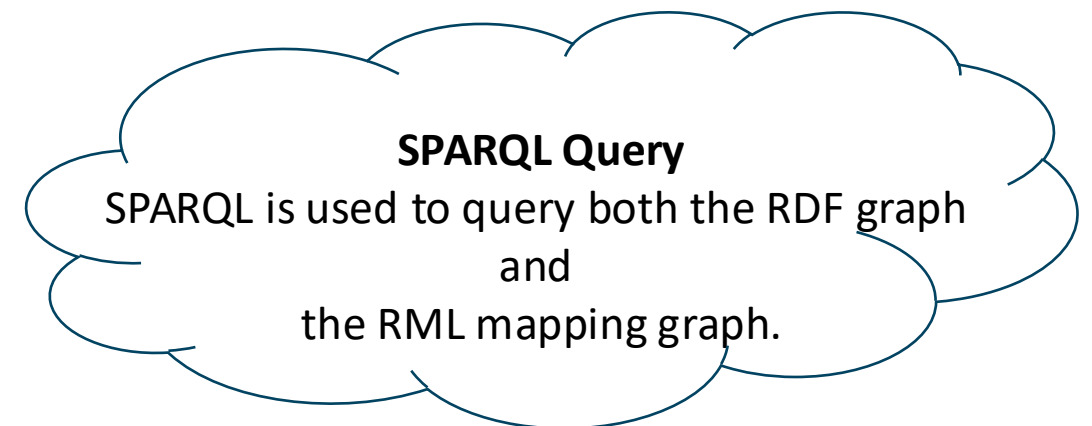
2. With SPARQL:

Use a SPARQL query to retrieve the specific data needed and then format it into CSV.

➤ Output:

A structured CSV file (output.csv).

1. Load RDF and RML mapping files into memory.
2. Extract the subject template from the RML mapping.
3. Extract column names from both RDF data and the RML mapping.
4. Extract the data by iterating through RDF triples
5. Write the structured data to a CSV file.



Process for Transforming RDF Data into CSV

➤ Input:

RDF file (data.nq).

RML mapping file (mapping.ttl).

➤ Data Retrieval:

1. With RDF (rdf.nq) + RML (rml.ttl):

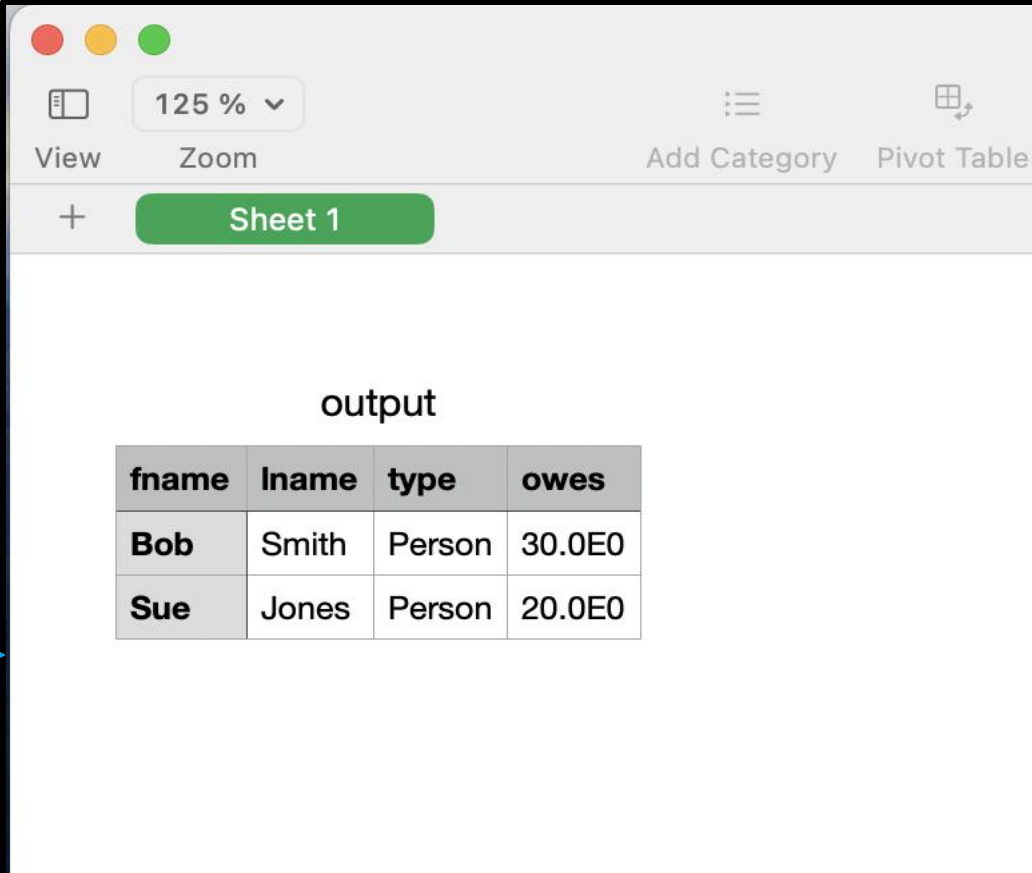
The RML mapping provides the logic to transform RDF triples into CSV rows and columns.

2. With SPARQL:

Use a SPARQL query to retrieve the specific data needed and then format it into CSV.

➤ Output:

A structured CSV file (output.csv).



The screenshot shows a window of a spreadsheet application. At the top, there are window control buttons (red, yellow, green) and a zoom dropdown set to 125%. Below the zoom are buttons for 'View', 'Zoom', 'Add Category', and 'Pivot Table'. A green tab labeled 'Sheet 1' is visible. The main area of the spreadsheet contains a table titled 'output'.

fname	lname	type	owes
Bob	Smith	Person	30.0E0
Sue	Jones	Person	20.0E0

Proposed Approach

```
rml:predicateObjectMap [  
  rml:objectMap [  
    rml:reference "Name"  
  ];  
  rml:predicate ex:name  
];  
rml:subjectMap [  
  rml:template "http://example.com/{Country Code}"  
] .
```

```
<http://example.com/1> <http://example.com/name> "Bolivia, Plurinational State of" .  
<http://example.com/2> <http://example.com/name> "Ireland" .  
<http://example.com/3> <http://example.com/name> "Saint Martin (French part)" .
```

Country_Code	name
3	Saint Martin (French part)
1	Bolivia, Plurinational State of
2	Ireland

1. Template Processing:

Extracts and maps values from subject templates and object templates.

- ✓ These templates define how the RDF triples were originally constructed, specifying the structure of subjects and objects using placeholders.
- ✓ By analyzing these templates, the placeholders and their corresponding values in the RDF triples are determined.

The interpreter

- parses the RDF triples,
- extracts the placeholder values
- maps them back to structured data fields, such as columns in a CSV file.

Proposed Approach

Code	label_es	label_en
BO	Estado Plurinacional de Bolivia	Bolivia, Plurinational State of
IE	Irlanda	Ireland

2. Language-Specific Columns: Handles multilingual data by creating separate columns for each language.

✓ RDF objects can have language tags (e.g., "Name"@en for English or "Name"@fr for French).

Interpreter identifies these language tags and creates separate columns in the CSV.

3. Datatype Mapping: Maps predicates to their corresponding datatypes and includes them in the CSV.

✓ RDF predicates can have associated datatypes

The interpreter extracts the datatype (e.g., xsd:int, xsd:string, xsd:date) and includes it as an additional column in the CSV.

Student	Details_datatype	Details
Name	string	Venus
DoB	date	1981-12-30
Age	int	43

Technical Aspect

1. Load RDF and RML mapping files into memory.

- Parses the RDF file (rdf.nq) using [rdflib.Dataset\(\)](#).
- Parses the RML mapping file (rml.ttl) using [rdflib.Graph\(\)](#).

2. Extract the subject template from the RML mapping.

- **SPARQL Query:** Looks for rml:subjectMap with an rml:template to determine how subjects in RDF are constructed.

3. Extract column names from both RDF and RML

- For every predicate in the RDF data,
 - ✓ The last part of the URI (after / or #) is extracted to create column names.
- Extracts Columns from RML
 - ✓ **SPARQL Query:** Fetches all predicates defined in the RML mapping to ensure columns are included

1. Load RDF and RML mapping files into memory.
2. Extract the subject template from the RML mapping.
3. Extract column names from both RDF data and the RML mapping.
4. Extract the data by iterating through RDF triples
5. Write the structured data to a CSV file.

Technical Aspect

4. Extract and structure RDF data according to columns.

- Iterates through RDF triples and organizes them into rows based on the subject.
- Object:
 - ✓ Converts blank nodes and URIRefs into readable strings.
 - ✓ Directly uses literal values for CSV.
- Maps Data to Columns:
 - ✓ For each subject, it initializes a row with all columns and fills in values based on the predicates (leaving empty cells for missing data).

5. Write the structured data to a CSV file.

- Converts the data dictionary into a **Pandas DataFrame**.
- Writes the DataFrame to a CSV file.

1. Load RDF and RML mapping files into memory.
2. Extract the subject template from the RML mapping.
3. Extract column names from both RDF data and the RML mapping.
4. Extract the data by Iterating through RDF triples
5. Write the structured data to a CSV file.

Demo

Challenges

Many-to-Many Relationships

`Student_sport.csv` file defines a many-to-many relationship between students and sports.

Eg., Student 11 (Fernando) plays both Football and Formula1.

Fails to generate separate rows for each sport played by the same student.

ID_Student	description	id	lastName	firstName	plays	ID_Sport
	Tennis	110				
	Formula1	112				
	Football	111				
11			Alonso	Fernando	111	111
12			Villa	David	111	111
10			Williams	Venus	110	110

`sport.csv`

ID	Description
110	Tennis
111	Football
112	Formula1

`Student_sport.csv`

ID_Student	ID_Sport
10	110
11	111
11	112
12	111

`student.csv`

ID	FirstName	LastName
10	Venus	Williams
11	Fernando	Alfonsa
12	David	Villa

Challenges

Many-to-Many Relationships

`Student_sport.csv` file defines a many-to-many relationship between students and sports.

Eg., Student 11 (Fernando) plays both Football and Formula1.

Fails to generate separate rows for each sport played by the same student.

Interpreter may be **overwriting or skipping** data during the transformation process.

`sport.csv`

ID	Description
110	Tennis
111	Football
112	Formula1

`Student_sport.csv`

ID_Student	ID_Sport
10	110
11	111
11	112
12	111

`student.csv`

ID	FirstName	LastName
10	Venus	Williams
11	Fernando	Alfonsa
12	David	Villa

ID_Student	description	id	lastName	firstName	plays	ID_Sport
	Tennis	110				
	Formula1	112				
	Football	111				
11			Alonso			
12			Villa	David		
10			Williams	Venus	110	110

Challenges

sport.csv

ID	Name
100	Tennis

student.csv

ID	Sport	Name
10	100	Venus Williams
20		Demi Moore

Missing Join Resolution

```
rml:joinCondition [  
  rml:child "Sport";  
  rml:parent "ID"  
];
```

This join condition is supposed to link the Sport column in student.csv to the ID column in sport.csv

ID	type	label	name	practises
20	Student		Demi Moore	
	Sport	Tennis		
10	Student		Venus Williams	sport_100

Challenges

sport.csv

ID	Name
100	Tennis

student.csv

ID	Sport	Name
10	100	Venus Williams
20		Demi Moore

Missing Join Resolution

```
rml:joinCondition [  
  rml:child "Sport";  
  rml:parent "ID"  
];
```

This join condition is supposed to link the Sport column in student.csv to the ID column in sport.csv

Join is not being resolved, and the raw resource identifier (sport_100) is being used.

ID	type	label	name	practises
20	Student		Demi Moore	
	Sport	Tennis		
10	Student		Venus Williams	sport_100



Further Enhancements

Blank Node Handling

Case 1 :: Subject Template: {Name}

Blank Node in RDF: _:Venus

It Works: The subject template {Name} directly matches the blank node identifier _:Venus. Interpreter can easily extract the Name value Venus because the blank node identifier is already in the expected format.

Case 2 :: Subject Template: students{ID}

Blank Node in RDF: _:students10

The ID value is not extracted because of the subject template students{ID}

Interpreter expects the blank node identifier to follow the format students10 (without the _:prefix).



Further Enhancements

Template Processing:

```
rml:predicateObjectMap [  
  rml:objectMap [  
    rml:template "\\{\\{\\{ {ISO 3166} \\}\\}\\}";  
    rml:termType rml:Literal  
  ];  
  rml:predicate ex:code  
];  
rml:subjectMap [  
  rml:template "http://example.com/{Country Code}/{Name}"  
] .
```

The rml:template includes unnecessary escaped curly braces (\\{\\{\\{ {ISO 3166} \\}\\}\\}).

Country_Code	Name	code	\\{ {ISO 3166}
1	Bolivia, Plurinational State of	{{{ BO }}	{{{
2	Ireland	{{{ IE }}	{{{
3	Saint Martin (French part)	{{{ MF }}	{{{

✓ Preprocess the data to remove unnecessary curly braces and standardise values.



Project Achievements

❖ **Template Parsing:**

The subject templates are parsed to extract placeholders, which are then used to reconstruct the corresponding columns in the CSV. This ensures that the structure of the original data source is recreated.

❖ **Support for Data Types and Language Tags:**

The approach includes support for handling data types and language tags. This ensures that the reconstructed CSV retains important metadata from the RDF dataset.

❖ **Generalised Framework:**

The approach provides a generalised framework for reversing RML mappings, making it applicable to a wide range of use cases and data sources.



Thank You for Your Attention!

“Open to Questions”



Additional slides

Technical Aspect

SPARQL Query:

- ✓ **SPARQL is used to query the RML mapping graph** (self.mapping_graph) to extract templates, term types, and mappings that define how RDF data should be processed.
- ✓ **SPARQL is also used to query the RDF graph** (self.graph) to extract the actual data (subjects, predicates, and objects) based on the mappings.

Query	Purpose
Extracting Subject Template	Retrieves the subject template and term type.
Extracting IRIs for a Specific Type	Retrieves all IRIs of subjects of a specific type (e.g., foaf:Person).
Extracting Object Maps with Datatypes	Retrieves object maps and their associated datatypes.
Extracting Object Templates for Predicates	Retrieves object templates for a specific predicate.

Technical Aspect

Convert the Data Dictionary into a Pandas DataFrame

The structured data is stored in a **dictionary**, where:

- Each **key** represents a subject (e.g., a URI or blank node).
 - Each **value** is another dictionary containing column names (predicates) as keys and their corresponding values (objects) as values.
- ```
data = { "http://example.com/John": {"name": "John", "age": "30"},
 "http://example.com/Jane": {"name": "Jane", "age": "25"}, }
```

This dictionary is converted into a **Pandas DataFrame**

```
pd.DataFrame.from_dict(data, orient="index")
```