UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

SEMI-AUTOMATIC EXTRACTION OF RDF TRIPLES FROM WIKIPEDIA TABLES

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

ADRIANA CECILIA CONCHA SEPÚLVEDA

PROFESOR GUÍA:
AIDAN HOGAN

MIEMBROS DE LA COMISIÓN:
MARIA CECILIA BASTARRICA PIÑEYRO
JOSÉ SAAVEDRA RONDO
DOMAGOJ VRGOC

Este trabajo ha sido parcialmente financiado por ANID – Millennium Science Initiative Program – Code ICN17_002 y FONDECYT Regular N° 1181896

SANTIAGO DE CHILE
2025

Extracción semi-automática de triples RDF de tablas de Wikipedia

Wikipedia es una enciclopedia en línea, multilingüe y gratuita disponible en 339 ediciones. La edición de Wikipedia en inglés actualmente cuenta con más de 6,7 millones de artículos y 46 millones de usuarios registrados. Una gran cantidad de información en Wikipedia se encuentra contenida en tablas. El problema radica en el gran número de tablas en Wikipedia, así como en la gran cantidad de tipos de esquemas de tablas en un formato semi-estructurado. En consecuencia, extraer manualmente información estructurada de estas tablas se vuelve impráctico, y automatizar este proceso se vuelve muy complejo.

Para abordar este problema, proponemos una solución que involucra el uso de clusters de tablas de Wikipedia agrupadas por encabezados similares y la selección de un lenguaje de mapeo adecuado para transformar la información de esas tablas en información estructurada.

En este estudio, realizamos un análisis comparativo de varios lenguajes de mapeo y sus respectivos procesadores usados para la extracción de relaciones semánticas de las tablas de Wikipedia. El objetivo principal es extraer información semántica de las tablas de Wikipedia con alta precisión a gran escala, con el objetivo de integrar el conocimiento resultante de manera más efectiva en bases de conocimiento existentes, como Wikidata.

Si bien estudios previos han comparado algunos lenguajes de mapeo y procesadores, ninguno ha abordado este corpus en particular. Nuestra comparación de diferentes procesadores aplicados a nuestro data corpus destaca a Tarql como el procesador más productivo, generando 984,260 triples, de los cuales 791,021 son nuevos en Wikidata. Además, al examinar 500 relaciones aleatorias extraídas por Tarql de los 10 clusters más grandes, obtuvimos como resultado una precisión promedio del 84.6%. Sin embargo, como la imprecisión está principalmente asociada con clusters específicos de tablas, excluir estos clusters podría mejorar significativamente la precisión lograda. Métodos automatizados previos lograron una precisión del 81.5% y del 70%, lo que indica una mejora en la precisión con nuestro enfoque.

Abstract

Wikipedia is a multilingual online free encyclopedia available in 339 language editions. English Wikipedia currently has more than 6.7 million articles and 46 million registered users. A huge amount of information in Wikipedia is contained in Wikipedia tables. The problem lies in the large number of tables on Wikipedia, as well as the presence of multiple types of table schemas in a semi-structured format. Consequently, manually extracting structured information from these tables becomes impractical, and automating this process becomes very complex.

To address this issue, we propose a solution that involves using clusters of Wikipedia tables grouped by similar headers based on previous research and selecting a suitable mapping language to transform information from those tables into structured information.

In this study, we conduct a comparative analysis of various mapping languages and processors suitable for semantic relation extraction from Wikipedia tables. The main goal is to extract semantic information from Wikipedia tables with high precision at large scale, aiming to integrate the resulting knowledge more effectively into existing knowledge bases, such as Wikidata.

While previous studies have compared some mapping languages and processors, none have addressed this particular data corpus. Our comparison of different tools applied to our data corpus highlights Tarql as the most productive processor, generating 984,260 triples, with 791,021 being novel in Wikidata. Furthermore, sampling 50 random relations extracted by Tarql from each of the top 10 largest clusters, assessing 500 relations in total, yielded an average precision of 84.6%. However, as the imprecision is primarily associated with specific clusters, excluding these clusters could significantly improve the precision achieved. Previous automated methods achieved a precision of 81.5% and 70%, indicating an improvement in precision with our approach.

Table of Content

List of Tables

List of Figures

Chapter 1

Introduction

Wikipedia is a multilingual online free encyclopedia. Volunteer editors from all around the world can collaboratively write and edit articles in a wiki-based system. According to the Top Websites Ranking published by Similarweb, as of 2024, Wikipedia is the seventh most visited website in the world.[1]

Wikipedia is available in 339 language editions. The English Wikipedia currently has more than 6.7 million articles and 46 million registered users.[2]

Despite the huge amount of information enclosed in Wikipedia, there are several issues that compromise the reliability and accessibility of Wikipedia data. The content on Wikipedia articles is meant for human consumption, so while the content is understandable and visually appealing for the human eye, the lack of structure hinders the automatic extraction of data.

The information in Wikipedia is manually maintained by its editors and each article, regardless of the language edition, is independent from the others. This leads to redundant edits of the same article across different languages, or even multiple edits within the same language on different articles that mention the same fact. For example, each new earthquake in Chile should be added to the article List of earthquakes in Chile, to the article List of earthquakes (on different tables, categorized by year, region, largest magnitude, etc.) and sometimes, if the magnitude is big enough, to the main article of Chile. All of these edits are made individually on each language version of Wikipedia, which leads to inconsistencies of information.

In order to overcome these issues several initiatives have made use of Wikipedia data, two of which are DBpedia and Wikidata.

[1] `https://www.similarweb.com/es/top-websites/`

[2] `https://en.wikipedia.org/wiki/Wikipedia:Statistics`

1.1 Initiatives for structuring Wikipedia information

DBpedia [22] is a community project that aims to extract structured information from Wikipedia. As of 2023, DBpedia contains more than 228 million entities. The extraction of information from Wikipedia articles has been mainly focused on the extraction of semantic triples from the Infoboxes of Wikipedia. Infoboxes are information boxes that appear on the top right of Wikipedia pages and provide a summary of key facts about a particular topic.

DBpedia has also made use of the categories assigned to Wikipedia articles. Categories are used on Wikipedia to group related pages together, based on their subject matter. In this manner, DBpedia can extract structured data about the topics covered by a Wikipedia article.

Similar to the Wikipedia language versions, DBpedia has 20 "chapters", each one of them in different languages independent from each other.

Another notable knowledge base is Wikidata [31]. Wikidata is a free and open knowledge base that was launched by the Wikimedia Foundation in 2012. The aim of Wikidata is to create a free, collaborative, multilingual, and open database of structured data that can be used by humans and machines. It aims to provide a common source of structured data that can be used by Wikipedia and different Wikimedia projects, as well as other websites and applications.

While several initiatives have been structuring data from Wikipedia articles, the information in knowledge bases like DBpedia or Wikidata remains incomplete. There is still plenty of information in Wikipedia articles, especially that contained in the tables of Wikipedia, that currently cannot be used in a meaningful way other than to be read by humans.

These initiatives use semantic triples to organize their information. Semantic triples provide a structured way of organizing data meaningfully. Represented in the format $(subject, predicate, object)$, semantic triples represent the relationship between a subject and an object through a specific property. The Resource Description Framework (RDF)[3], a World Wide Web Consortium (W3C) standard data model, is based on semantic triples. In essence, semantic triples form a cohesive framework that facilitates the meaningful organization and representation of data.

1.2 About the tables

As of 2019, English Wikipedia contained over 5.5 million articles, which collectively featured around 17 million tables. These tables were classified by Luzuriaga [25] into three categories:

[3] `https://www.w3.org/TR/rdf-primer/`

Infoboxes, Layout tables and Tables with useful content. The latter category, comprising about 22.5% of all tables, underwent a normalization process in the work of Luzuriaga that involved splitting merged cells and rearranging inner tables. Tables without headers were excluded, resulting in a final dataset of 3,631,228 tables, which were grouped into clusters of tables by similar headers.

The tables have a wide range of schema designs, intended for human consumption. However, their formatting features make automated data extraction difficult. For instance, some tables have empty columns used for aesthetic purposes, contain images, or use color coding that requires an external legend for interpretation. Furthermore, some cells include footnotes, and certain columns have multiple entities per cell that may have unclear relationships between each other.

For example Figure 1.1 shows some studio albums and their peak chart positions with their corresponding certifications across different countries. This table has multiple headers on the first row, multiple entities and footnotes in their cells and nested attributes in cells.

Title	Album details	Peak chart positions										Sales	Certifications
		UK [7][5]	AUS [8]	AUT [9]	FRA [10]	GER [11]	NL [12]	NZ [13]	SWE [14]	SWI [15]	US [6]		
Kiss Me, Kiss Me, Kiss Me	• Released: 26 May 1987 • Label: Fiction, Elektra • Formats: CD, 2×LP, MC	6	9	4	2	4	3	14	13	3	35		• BPI: Gold[17] • RIAA: Platinum[18]
Disintegration	• Released: 2 May 1989 • Label: Fiction, Elektra • Formats: CD, 2×LP, MC	3	9	5	3	2	3	6	10	4	12		• BPI: Gold[17] • BVMI: Gold[19] • IFPI SWI: Gold[20] • RIAA: 2× Platinum[18] • RMNZ: Gold[21]
Wish	• Released: 21 April 1992 • Label: Fiction, Elektra • Formats: CD, 2×LP, MC	1	1	14	17	4	22	3	10	5	2		• BPI: Gold[17] • ARIA: Platinum[22] • IFPI SWI: Gold[20] • RIAA: Platinum[18] • RMNZ: Gold[23]

Figure 1.1: Example of a Wikipedia table[4]

[4]abridged from the Wikipedia article `https://es.wikipedia.org/wiki/Anexo:Discograf%C3%ADa_de_The_Cure`

1.3 Table extraction

Extracting and categorizing semantic triples from Wikipedia tables is a challenging task, as the tables may have different structures and formats. Previous research has attempted to address this problem using various automated techniques such as machine learning [23], probabilistic graphical models [27], knowledge-based approaches [28, 29], and table clustering [25]. Despite these efforts, the precision of the RDF triples extracted using these methods may not always be reliable, which could impact their usefulness for enriching existing knowledge bases. Therefore, further improvements are necessary to enhance the accuracy and reliability of these methods, and to increase the potential for successfully incorporating the extracted triples into existing knowledge bases.

The core idea explored by this thesis is that the reliability of the information extracted can be improved by defining custom mappings from Wikipedia tables to RDF triples. Given the vast number of tables available on Wikipedia, creating custom mappings for each table is not a feasible task. However, by leveraging the clustering of tables based on similar headers, as proposed by Luzuriaga [25], it is possible to create a single mapping that can be applied to all tables within a given cluster.

A system that provides easy access to table clusters, allowing expert users to create custom mappings in a suitable language, would be highly beneficial for extracting new information from Wikipedia tables with high precision. With such a system in place, expert users could quickly and easily create mappings for multiple tables within a cluster, thereby increasing the accuracy and reliability of the extracted information. This approach would save time and effort, while enabling the extraction of valuable knowledge from Wikipedia tables.

Selecting an appropriate language is a complex task, given the variety of RDF mapping languages and processors developed in recent years. Therefore, it is necessary to conduct an exploratory analysis using Wikipedia table clusters to compare the expressiveness, scalability, and performance of these RDF mapping language processors over this particular data corpus.

1.4 Problem statement

Wikipedia tables are in a semi-structured format (HTML), which makes it difficult for machines to extract information. Extracting information from these tables would make it possible to link information on the Web, to carry out more complex queries on this data and to unify information within the different language versions of Wikipedia.

The problem is that there are a large number of tables in Wikipedia, as well as multiple types of table schemas, so manual extraction of these relationships becomes impractical and

automatic extraction becomes very complex.

In order to address this issue, we propose a solution that involves using clusters of Wikipedia tables grouped by similar headers [25] and choosing a suitable mapping language to transform information from those tables into RDF format. Through the comparison of various RDF mapping language processors and the identification of a suitable one, the extraction of RDF triples from Wikipedia tables can be made more accurate and reliable, and the resulting knowledge can be integrated more effectively into existing knowledge bases.

1.5 Hypothesis

It is feasible to extract a substantial volume of novel triples with high precision from Wikipedia tables by clustering the tables and utilizing mapping languages to RDF over these clusters for populating knowledge graphs, such as Wikidata.

1.6 Research questions

Here we formulate the key research questions that underlie the aforementioned hypothesis:

- How much information is in Wikipedia's tables?
- What are the largest table clusters?
- How many tables are in each table cluster?
- What mapping languages can be used?
- What is the expressiveness of the chosen mapping languages?
- What processors of those mapping languages can be used?
- What is the performance of the chosen mapping languages processors?
- What is the scalability of the chosen mapping languages processors?
- How many facts can be extracted per cluster?
- What percentage of such facts would be novel in knowledge-bases such as Wikidata?
- Are the extracted facts correct?

1.7 Objectives

1.7.1 General objective

The main goal of this thesis is to extract RDF triples from Wikipedia tables with high precision at large scale. To achieve this goal, an exploratory analysis is conducted to compare various mapping languages to RDF, facilitating the definition of mappings to RDF that will be applied to each table in the cluster. In this thesis, we will specifically explore the extraction of RDF triples using the Wikidata vocabulary.

We propose a novel method to extract RDF triples from Wikipedia tables by using mapping languages to transform semi-structured data into RDF. By defining mappings for clusters of tables rather than individual ones, we enable the extraction of triples at large scale. We define custom mappings to be applied to clusters of tables to further improve the precision and number of triples extracted. We evaluate this novel method by comparing different mapping languages and their processors.

1.7.2 Specific objectives

- Analyze the structure of clusters of Wikipedia tables, proposed by Luzuriaga [25], sharing the same schema. These schemas are defined by a subset of attributes, ensuring that all tables within a cluster adhere to this common schema.
- Compare the performance, expressiveness and scalability of multiple RDF mapping languages over different table structures.
- Extract relationships from a sample of clusters.
- Validate the quality of the solution by measuring precision[5], the number of RDF triples extracted and the number of RDF triples that are new in Wikidata.

1.8 Contributions

This thesis presents the following key contributions:

- A novel semi-automatic method for extracting triples from Web tables, such as those found on Wikipedia, at a large scale with high precision.

[5]Precision is calculated by considering the correct triples extracted as true positives and the retrieved triples as the sum of true positives and false positives, following the definition of Precision by D. Manning et al.(2008) [26]

- A benchmark consisting of 12 clusters of Wikipedia tables, including a collection of tables for comparing the expressiveness and performance of various processors that implement mapping languages from semi-structured data to RDF.
- A comparison of the features and expressiveness of different mapping languages when applied to a specific corpus of Wikipedia tables.
- A performance evaluation of various mapping language processors, using groups of tables and applying mappings with different levels of complexity.
- The generation of a large number of novel Wikidata triples with a precision of 84.6%.

1.9 Thesis structure

- Chapter 2: State of the Art
- Chapter 3: Data Corpus
- Chapter 4: Features of Mapping Languages
- Chapter 5: Experimental Results and Discussion
- Chapter 6: Conclusions and Future Work

Chapter 2

State of the Art

In this chapter, we present some of the languages, tools, techniques and initiatives that form the background of this thesis. Specifically, we discuss concepts relating to the Semantic Web, Web tables, and Mapping languages.

2.1 Semantic Web

The World Wide Web is a global information system invented in 1989 by Tim Berners-Lee. Since the beginning of the Web, the idea was to be able to share and link information displayed in documents in an easy automated way. These documents, encoded in HTML (HyperText Markup Language), are meant to be read by humans. This is one of the main reasons why computers don't have a reliable way to process the semantics behind the information extracted from the Web.

The Semantic Web is considered an extension of the World Wide Web, whose objective is to provide structure to the Web, in order to facilitate the integration of data from different sources on the Web and their accessibility by machines [9].

The components of the Semantic Web were described by Berners-Lee et al. (2006) [7] through the Semantic Web Stack (Figure 2.1).

The bottom of the stack represents the syntax components, where the Universal Resource Identifier (URI) standard allows for identifying resources on the Web, while Unicode provides the characters needed to describe data globally.

Next up, the Extensible Markup Language (XML) defines the structure of a document through user defined tags that encapsulate different elements inside the document, providing machine-readable information.

Above XML are located the semantic components. The Resource Description Framework (RDF) gives information about relationships between elements. Meanwhile, the RDF Schema (RDF-S) and the OWL ontology language define common vocabularies to give semantics to RDF Data. Once RDF data is modeled, access to the information is provided by the SPARQL query language, a W3C recommendation for querying and manipulating RDF graphs [15].

The top layers: Unifying logic, Proof and Trust combine Queries, Ontologies and Rules while providing explanation of the results obtained from the automated agents and assuring the trustworthiness of the sources. Finally, the results are given to the User Interface & applications layer.

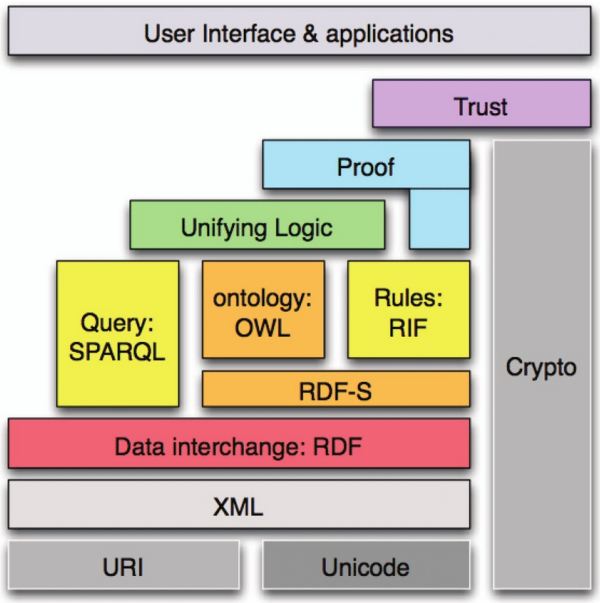

Figure 2.1: The Semantic Web Stack c.2006. [8]

In terms of the present day, many of the core elements of this stack have been defined and are in broad use. However, the top layers of Unifying Logic, Proof and Trust remain underdeveloped. Another important change has been to reduce the emphasis on XML, where native syntaxes for RDF [6, 10] and syntaxes based on JSON [19] have become more popular in recent years. On the other hand, new components have been added, such as a Validation component for checking RDF data with respect to certain constraints. The principal standard for Validation on the Semantic Web is the Shapes Constraint Language (SHACL) [20].

In the following, we focus on describing the RDF and SHACL standards along with the SPARQL query language as they are the most relevant for this work.

2.1.1 Resource Description Framework (RDF)

The current Web shares information through hyperlinks between documents. In order to make the data machine readable, it became necessary to identify and establish relationships between the things inside those documents. Thus, the World Wide Web Consortium (W3C) proposed the RDF standard.

The Resource Description Framework (RDF) is a standard data model for representing information about resources on the World Wide Web.[1] It is based on semantic triples of the form (s, p, o), where the subject s is related to an object o through a property p. The idea is to be able to identify things and the relationships between them using Web identifiers known as Uniform Resource Identifiers (URIs) or the generalized version of them known as Internationalized Resource Identifiers (IRIs).

An RDF graph is a set of RDF triples, where the subjects and objects are the nodes and the properties are the labels of the directed edges.

The resources in the RDF triple can be categorized into three different RDF terms: Literals, Blank nodes and IRIs.

- **Literals:** Literals are a lexical representation of constant values, such as numbers, dates or booleans. There are two types of literals: plain or typed. A plain literal is just a string, for example `"petrichor"`. An optional language tag may be appended to the string, such as `"petrichor"@en`, which indicates that this string is in English, following the ISO 639 standard for language codes.[2] A typed literal is a string with a datatype, for example `"10"^^xsd:int`, which indicates that the string is an integer. In Listing 2.1 `"2018"^^xsd:gYear` represents the year 2018.
  A literal may only be used in the object position of an RDF triple.

- **IRIs:** Internationalized Resource Identifiers (IRIs) are a generalization of URIs that allow Unicode characters. IRIs have a global scope, allowing the consistent interpretation of the resource in different contexts across the Web.

  IRIs can be abbreviated with a prefix. So for example the IRI

  `http://www.example.org/entity/human` can be addressed as `ex:human`, where the prefix `ex:` can be defined as standing for `http://www.example.org/entity/`.

  An IRI may appear as subject, predicate or object of an RDF triple.

- **Blank nodes:** A blank node represents the existence of a resource that is not a Literal or an IRI reference. A blank node identifier, with a local scope, may be used so several statements can reference the same blank node. For example, in Listing 2.1 `bn` and `bn1`

[1] `https://www.w3.org/TR/rdf-primer/#rdfmodel`
[2] `https://www.loc.gov/standards/iso639-2/php/English_list.php`

are identifiers for blank nodes representing awards received in specific years. Blank nodes may only be used in the subject or object positions of an RDF triple.

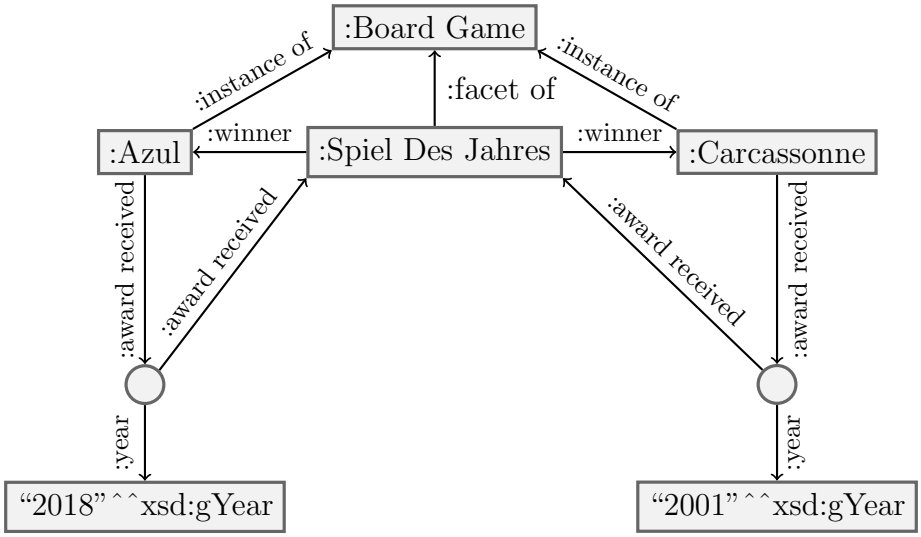

Figure 2.2: Example of an RDF graph.

```
@prefix : <http://example.org/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:Azul :instanceOf :BoardGame;
      :awardReceived _:bn.
_:bn  :awardReceived :SpielDesJahres;
      :year "2018"^^xsd:gYear.
:Carcassonne  :instanceOf :BoardGame;
              :awardReceived _:bn1.
_:bn1    :awardReceived :SpielDesJahres;
         :year "2001"^^xsd:gYear.
:SpielDesJahres :winner :Azul;
                :winner :Carcassonne;
                :facetOf :BoardGame.
```

Listing 2.1: Example of an RDF graph on Turtle syntax

There are several file formats to store RDF graphs, such as N-triples [10], JSON-LD [19], RDF/XML [5] and the Terse RDF Triple Language (Turtle) [6]. In Listing 2.1 a Turtle representation of the graph from Figure 2.2 is given. The Turtle syntax provides a concise way of writing RDF graphs by abbreviating IRIs through the use of prefixes, and the use of lists for nested blank nodes and RDF collections.

The advantages of using Turtle documents for storing and parsing RDF data is the conciseness and readability of the syntax; it is also beneficial for querying RDF graphs using the SPARQL query language, due to the similarities between both syntaxes.

2.1.2 SPARQL

Just as the Structured Query Language (SQL) is a language for querying relational databases, SPARQL is a language for querying RDF data. In 2008, SPARQL 1.0 was released as a W3C Recommendation [30] being followed by SPARQL 1.1 in 2013 [15].

SPARQL uses a triple pattern syntax compatible with the Turtle syntax for RDF graphs. Consequently the triples can be shortened by using prefixes, aiming to be easy to write and read by humans. The variables in a SPARQL query are represented by adding the symbol `?` or `$` before the name of the variable, like `?x` or `?y`.

A SPARQL query consists of five parts:

1. **Prefix declarations:** The Prefix section is used to establish abbreviations for the IRIs to be used afterwards in the query.
2. **Dataset clause:** The Dataset clause specifies the RDF graph being queried.
3. **Result type:** The result will depend on the type of SPARQL query. The result can be a table, an RDF graph, or a boolean.
4. **Query pattern:** Here is defined the pattern of the graph to be matched against the data in the RDF dataset.
5. **Solution modifiers:** The results can be ordered, sliced, limited to a number of solutions, among others.

As mentioned earlier, the result of the query will depend on the type of SPARQL query. SPARQL allows four types of queries.

- **Select query:** The `SELECT` clause selects the variables to be retrieved from the graph. This query type returns a table. For example, following Figure 2.2, the query in Listing 2.2 will retrieve board games that received the Spiel des Jahres award and their respectives years of winning (Table 2.1).

```
#Prefix declarations
@prefix : <http://example.org/>.
#Result type
SELECT ?game ?year
#Query pattern
WHERE {
    ?game :instanceOf :BoardGame;
          :awardReceived ?node.
    ?node :awardReceived :SpielDesJahres;
          :year ?year.
}
#Solution modifiers
ORDER BY  ?year
```

Listing 2.2: Select query example

Table 2.1: Result table for Listing 2.2

?game	?year
:Carcassonne	"2001"^^xsd:gYear
:Azul	"2018"^^xsd:gYear

- **Construct query:** Returns an RDF graph specified by a graph template. For example, the query in Listing 2.3 will return an RDF graph (Figure 2.3) from the input table of Table 2.2 showing actors and their roles on different TV shows, as could be returned from a SPARQL query pattern.

Table 2.2: Input table for Construct query on Listing 2.3

?actor	?title	?role
:Pedro Pascal	:The Mandalorian	:Din Djarin
:Pedro Pascal	:The Last of Us	:Joel Miller

```
#Prefix declarations
@prefix : <http://example.org/>.
#Result type
CONSTRUCT {
    ?title :characters ?role;
           :credit _:cast.
    _:cast :castMember ?actor;
          :characterRole ?role.
    ?actor :occupation :actor .
}
```

Listing 2.3: Construct query example

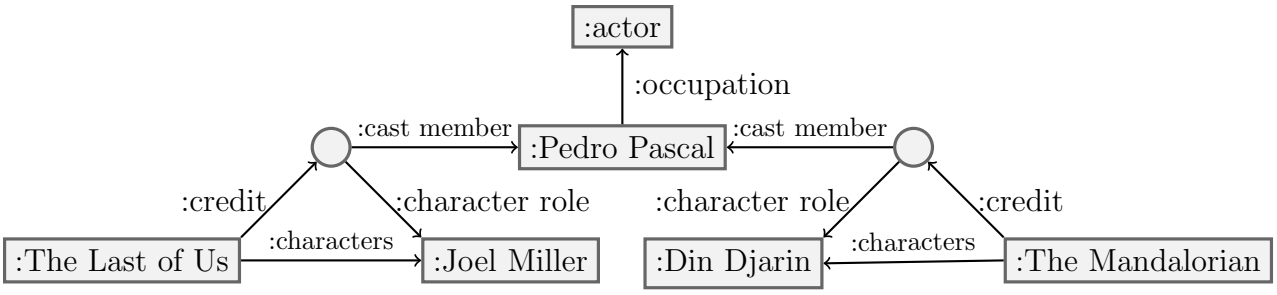

Figure 2.3: Output RDF graph for Construct query on Listing 2.3

- **Ask query:** Returns true if the query pattern has a match, false if not.
- **Describe query:** Returns an RDF graph describing resources.

2.1.3 SHACL

The Shapes Constraint Language (SHACL) [20] is a language for validating RDF graphs against a set of conditions.

SHACL provides a way to specify constraints and rules for describing the expected structure of RDF data. This includes specifying the shape of the data, the types of resources and properties that should be present, and the values that properties should have. SHACL can be used to validate that RDF data conform to these constraints, and can also be used to generate reports or modify data based on the constraints.

For example if the RDF graph on Figure 2.2 is validated against the shape defined in Listing 2.4, no errors will be found because each object value for the property `:year` satisfies the rule of having `xsd:gYear` as datatype.

```
@prefix : <http://example.org/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix s: <http://ex.org/shapes/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .

s:YearValue a sh:NodeShape ;
sh:targetObjectsOf :year ;
sh:datatype xsd:gYear .
```

Listing 2.4: SHACL example

2.1.4 DBpedia

Within the projects that involve the Semantic Web, there is DBpedia. This is a community project that extracts structured data from Wikipedia [22].

DBpedia has different language editions that extract information from their corresponding editions of Wikipedia. Each resource in DBpedia has a URI with the prefix `dbr: http://dbpedia.org/resource/` followed by the name of the resource, for example `dbr:Human`. In the Spanish edition of DBpedia, the analogous resource would be `dbr:Humano`, with the prefix `dbr:http://es.dbpedia.org/resource/`. Note that due to the language dependency between those two resources, even if they have the same meaning, they are not the same.

Regarding the information contained in tables, DBpedia has focused mainly on extracting information from Wikipedia's Infoboxes (Figure 2.4). Infoboxes are tables located in the upper right corner of a Wikipedia article, which show the most relevant information of the article in key-value format [22]. To extract information from Infoboxes in the form of semantic triples, DBpedia uses two types of extraction:[3]

- **Direct Extraction:** A URI (Uniform Resource Identifier) derived from the Wikipedia article title is used as the subject. Then, each Infobox attribute is parsed as a property of an RDF triple, whose object corresponds to the value of the attribute.
- **Mapping-based extraction:** To solve Wikipedia's problem of having different templates that describe the same kind of thing, or having different synonyms that refer to the same property, Wikipedia template mappings are manually written to an ontology from DBpedia. An ontology is a collection of terms (referring to concepts or relationships) that describe an area of interest.

2.1.5 Wikidata

While Wikipedia faces issues of inconsistency and duplicity of information due to their independent language editions, and lack of accessibility to the information by machines, there is Wikidata [31], a structured version of Wikipedia created in 2012 by the Wikimedia Foundation.

Wikidata is a free, collaborative, multilingual and easily accessible knowledge graph. It can be read and edited by both humans and machines. Wikidata has become the most edited Wikimedia project, having 150–500 edits per minute. About 90% of these edits are made by

[3] `https://wiki.dbpedia.org/services-resources/datasets/dbpedia-datasets#h434-10`

[4] abridged from the Wikipedia article `https://en.wikipedia.org/wiki/Irvine_Welsh`

```
{{Infobox writer
| name = Irvine Welsh
| birth_date = {{b-da|27 September 1958}}
| birth_place = [[Leith]], [[Edinburgh]],
   Scotland
| occupation = Writer
| alma_mater = [[Heriot-Watt University]] (MBA)
| genre = Novel, play, short story
| movement = [[Modernism]], [[post-modernism]]
| notableworks = ''[[Trainspotting (novel)|
   Trainspotting]]'' (1993)<br/> ''[[The Acid
   House]]'' (1994)<br>''[[Filth (novel)|Filth
   ]]'' (1998)''<br/>[[Porno (novel)|Porno]]''
   (2002)
| website = {{URL|http://www.irvinewelsh.net}}
}}
```

Irvine Welsh

Born	27 September 1958 (age 64) Leith, Edinburgh, Scotland
Occupation	Writer
Alma mater	Heriot-Watt University (MBA)
Genre	Novel, play, short story
Literary movement	Modernism, post-modernism
Notable works	*Trainspotting* (1993) *The Acid House* (1994) *Filth* (1998) *Porno* (2002)

Website

www.irvinewelsh.net

Figure 2.4: Infobox example with its respective Wikipedia template[4]

bots created by contributing users to automate tasks, yet one million edits per month are still made by humans [31]. Currently Wikidata contains more than 100 million items edited by 24.7 thousand active users.

A Wikidata item is a representation of anything in human knowledge. Each item has a corresponding URI with the prefix `wd:<http://www.wikidata.org/entity/>` followed by a unique identifier `Qx`, where `x` is an integer number, solving the issue of having different URIs for the same resource in different languages.

For example, `wd:Q752436` is the URI for the entity `common wombat`. On Figure 2.5 the Wikidata page for item `Q752436` is shown, displaying information about the item through a short description and statements associating values to the item through different properties.

The information in Wikidata can be queried through the Wikidata Query Service[6] that provides a SPARQL endpoint.

Wikidata operates under the open world assumption, which means that it considers any missing information as unknown, rather than false. This approach allows Wikidata to be flexible and adaptable, and new information can be incorporated as it becomes available without needing to modify or delete existing data.

Despite the vast amount of information available on Wikidata, it is still incomplete. While Wikidata is constantly evolving and being updated by volunteers, there are still topics and

[5]inspired by diagram of a Wikidata item in `https://www.wikidata.org/wiki/Wikidata:Introduction` using abridged version of `https://www.wikidata.org/wiki/Q752436`

[6]`https://query.wikidata.org/`

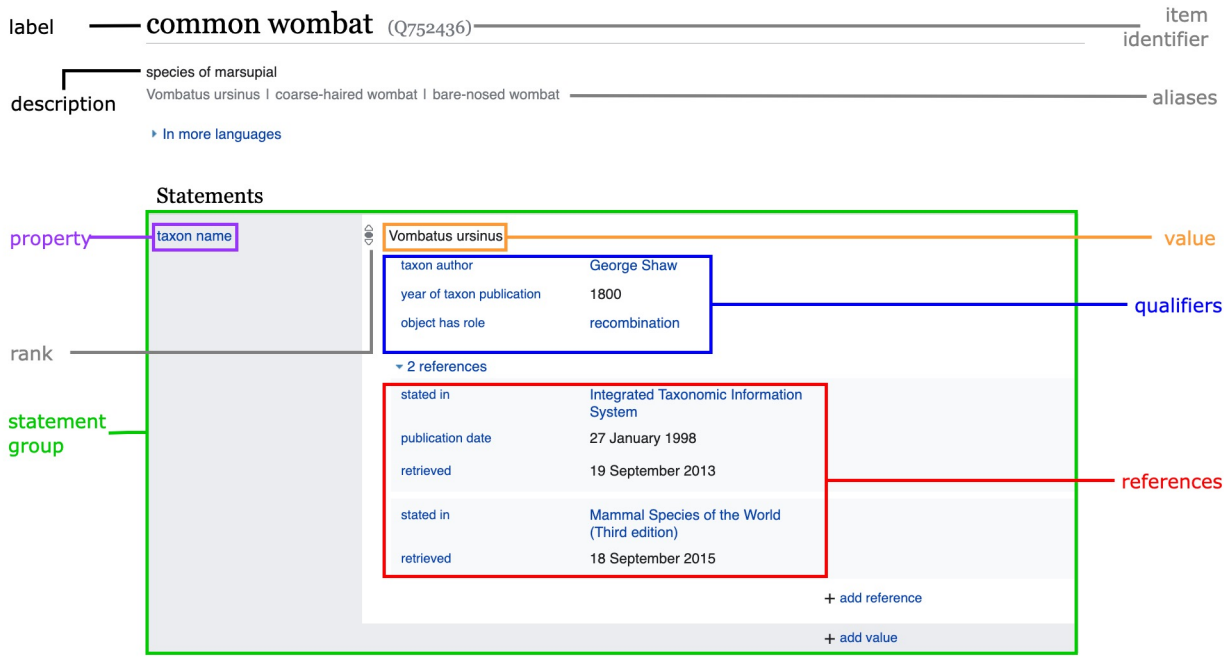

Figure 2.5: Wikidata entity model example[5]

areas with incomplete or missing information. Furthermore, there is plenty of information available on Wikipedia, particularly in tables, that could be incorporated into Wikidata, but that currently is not being used in a meaningful way.

2.2 Web tables

Extracting semantic information from Web tables, such as found on Wikipedia, can be summarized into two main tasks: Table detection, to identify and classify tables; and Table interpretation, to detect entities and relationships between the tables [25].

2.2.1 Table detection and categorization

In a first task, tables are identified and classified into various groups depending on their taxonomy. Liu et al. (2023) [24] present a novel classification of web tables based on existing research (Figure 2.6).

First, the authors classify tables into two primary categories: Layout tables and Genuine tables. Layout tables are used for design purposes, such as structuring content on web pages, and they lack semantic information within the table. For instance, a table might be used

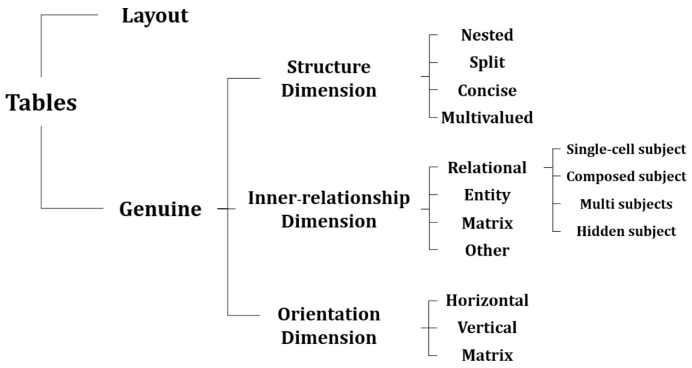

Figure 2.6: Classification of table types by Liu et al. (2023) [24]

for navigating between different contents (Figure 2.7). Genuine tables, in contrast, exhibit meaningful semantic relationships within their rows and columns.

v · t · e	Palos in flamenco	[hide]
Cantes a palo seco	debla · carceleras · trilla · saetas · martinetes · tonás	
Cantes related to soleá	soleá · soleá por bulerías · cantiñas (including alegrías, romeras, caracoles, mirabrás and other cantiñas) · bulerías · caña · polo	
Cantes related to seguiriya	seguiriya · cabales · serrana · livianas · toná liviana	
Cantes derived from fandangos	verdiales · jaberas · rondeñas · fandangos de lucena · malagueñas · tarantas · cartageneras · cantes de madrugá · minera · murciana · levantica · granaína	
Cantes related to tangos	tangos · tientos · **farruca** · garrotín · rumba · tanguillos · marianas	
Cantes de ida y vuelta	guajiras · vidalitas · milongas · and colombianas	
Other palos	sevillanas · nanas ("lullabies") · bambera · zambras · zorongo · campanilleros · peteneras	

Figure 2.7: Example of Layout table[7]

Since Layout tables aren't suitable for knowledge extraction, the authors delve into a more granular categorization of Genuine tables. The authors introduce a classification based on three non-mutually exclusive dimensions: structure, inner relationship, and orientation. The resulting table types emerge from a combination of these dimensions.

Structure dimension

This dimension focuses on the composition of table elements. Here, the authors identify four distinct subcategories.

- **Nested tables**
  These tables contain one or more tables in one or more of their cells. In Figure 2.8, the "Historical era" is presented as a nested table.

[7]from the Wikipedia article `https://en.wikipedia.org/wiki/Category:Flamenco_styles`
[8]abridged from the Wikipedia article `https://en.wikipedia.org/wiki/Byzantine_Empire`

Byzantine Empire Βασιλεία 'Ρωμαίων (Ancient Greek)[a] *Imperium Romanum* (Latin)	
Status	Empire
Capital	Constantinople (modern-day Istanbul)
Official languages	Greek
Historical era	Late Antiquity to Late Middle Ages
• First East–West division of the Roman Empire	1 April 286
• Inauguration of Constantinople	11 May 330
• Final East–West division after the death of Theodosius I	17 January 395
• Fall of the West; deposition of Romulus	4 September 476
• Sack of Constantinople by Catholic crusaders	12 April 1204
• Reconquest of Constantinople	25 July 1261
• Fall of Constantinople	29 May 1453
• Fall of Morea	29 May 1460
• Fall of Trebizond	15 August 1461

Figure 2.8: Nested Table[8]

- **Split tables**
  This are tables that can be split into sub-tables. For example, the table in Figure 2.9 can be split into various tables, such as a "Government" and "Area" tables. All of these subtables are independent of each other, but each one of them describes concepts related to the main entity of the article, in this case, the city of Talcahuano.

- **Concise tables**
  These tables incorporate merged cells to prevent redundancy, specifically avoiding repeated cells that reference the same content in both rows and/or columns. For example, Figure 2.10 presents merged cells in the columns "Year" and "Work".

- **Multivalued tables**
  These tables contain multiple values within a single cell. For instance, in Figure 2.11, multiple values are presented within the cells of the "Title" column.

Inner relationship dimension

This dimension focuses on the topological aspects of the semantic connections between cells.

[9] from the Wikipedia article `https://en.wikipedia.org/wiki/Talcahuano`

[10] abridged from the Wikipedia article `https://en.wikipedia.org/wiki/System_of_a_Down`

[11] from the Wikipedia article `https://en.wikipedia.org/wiki/B-Sides_%26_Rarities_(Deftones_album)`

Country	Chile
Region	Biobío
Province	Concepción
Founded	1764
Government[1][2]	
• Type	Municipality
• Alcalde	Gastón Saavedra Chandía (Ind.)
Area[3]	
• City, Port and Commune	92.3 km^2 (35.6 sq mi)
Elevation	1 m (3 ft)
Population (2012 Census)[3]	
• City, Port and Commune	150,499
• Density	1,600/km^2 (4,200/sq mi)
• Metro	250,348
• Urban	248,964
• Rural	1,384
Demonym	Talcahuian
Sex[3]	
• Men	121,778
• Women	128,570
Time zone	UTC−4 (CLT)
• Summer (DST)	UTC−3 (CLST)
Area code	56 + 41
Website	www.talcahuano.cl (in Spanish)

Figure 2.9: Split Table[9]

- **Relational tables**
  These tables describe relationships between entities and attributes. These tables can be oriented either vertically or horizontally, depending on the arrangement of the entities and their attributes within the table.

  Liu et al. characterized relational tables into further subcategories depending on the characteristics of their subjects. They defined four subtypes of relational tables: **Single-cell subject**, where each row or column (depending on whether it is a horizontal or vertical table) is associated with a single subject; **Composed subject**, where cells are combined to form a subject; **Multi subjects tables**, containing cells that refer to different subjects; and **Hidden subject tables**, which do not explicitly mention the subject of each row.

- **Entity tables**
  Entity tables, also known as attribute-value tables, are used to describe a distinct entity. These tables systematically list the attributes associated with the entity. Examples of entity tables include infoboxes found on Wikipedia (Figure 2.4).

- **Matrix tables**
  A matrix table is characterized by having the same value type in each cell where a row and column intersect. Values that are not in the first row/column cells are associated with both a row header and a column header (Figure 2.13).

Year	Awards	Category	Work	Result
1999	Kerrang! Awards[127]	Best International Live Act	—	Won
2002	Grammy Awards[128]	Best Metal Performance	"Chop Suey!"	Nominated
	MTV Video Music Awards[129]	Best Rock Video		Nominated
		Best Editing		Nominated
	Billboard Music Awards	Modern Rock Artist of the Year	—	Nominated
	MTV Video Music Awards Latinoamérica	Best International Rock Artist		Nominated
		Best International New Artist		Nominated

Figure 2.10: Concise Table[10]

No.	Title	Length
1.	"Savory" (Jawbox cover) (featuring Jonah Matranga, Shaun Lopez and Chris Robyn of Far)	4:37
2.	"Wax and Wane" (Cocteau Twins cover)	4:09
3.	"Change (In the House of Flies) (Acoustic)"	5:16
4.	"Simple Man" (Lynyrd Skynyrd cover)	6:20
5.	"Sinatra" (2005 remix of Helmet cover)	4:43
6.	"No Ordinary Love" (Sade cover, featuring Jonah Matranga of Far)	5:34
7.	"Teenager (Idiot Version)" (featuring Michael Harris and Daniel Anderson of Idiot Pilot)	3:45
8.	"Crenshaw Punch/I'll Throw Rocks at You"	4:49
9.	"Black Moon" (featuring B-Real of Cypress Hill)	3:18
10.	"If Only Tonight We Could Sleep" (The Cure cover, live in 2004 at *MTV Icon: The Cure*)	5:05
11.	"Please, Please, Please Let Me Get What I Want" (The Smiths cover)	2:04
12.	"Digital Bath (Acoustic)" (live in 2000 at KXTE, Las Vegas, Nevada)	4:48
13.	"The Chauffeur" (Duran Duran cover)	5:21
14.	"Be Quiet and Drive (Far Away) (Acoustic)" (additional vocals by Jonah Matranga of Far)	4:32

Figure 2.11: Multivalued Table[11]

- **Other tables**
  Liu et al. define other genuine tables as tables containing semantic information that doesn't fit into the other subcategories. Examples of such tables include enumeration and calendar tables.

Orientation dimension

This dimension considers the direction of relationships within a table.

- **Horizontal tables:** Each row in these tables describes a different subject (Figure 2.12).

[12]from the Wikipedia article `https://en.wikipedia.org/wiki/Origin_of_Symmetry`

Review scores	
Source	Rating
AllMusic	★★★★★[18]
Drowned in Sound	10/10[19]
Hot Press	8/12[20]
The Independent	★★★★★[21]
The List	4/5[22]
NME	9/10[23]
Pitchfork	8.3/10[17]
Q	★★★★★[24]
Stylus Magazine	C[25]
Sunday Herald	★★★★★[26]

Figure 2.12: Horizontal Relational Table[12]

- **Vertical tables:** Each column in these tables describes a different subject.
- **Matrix tables:** These tables are interpreted cell by cell (Figure 2.13).

Climate data for Concepción, Chile (Carriel Sur International Airport) 1981–2010, extremes 1966–present [hide]													
Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Year
Record high °C (°F)	34.1 (93.4)	34.4 (93.9)	30.9 (87.6)	27.2 (81.0)	25.8 (78.4)	22.0 (71.6)	21.7 (71.1)	25.0 (77.0)	28.8 (83.8)	27.6 (81.7)	32.5 (90.5)	33.4 (92.1)	34.4 (93.9)
Average high °C (°F)	22.8 (73.0)	22.5 (72.5)	21.0 (69.8)	18.3 (64.9)	15.4 (59.7)	13.7 (56.7)	13.2 (55.8)	14.0 (57.2)	15.5 (59.9)	17.2 (63.0)	19.4 (66.9)	21.4 (70.5)	17.9 (64.2)
Daily mean °C (°F)	16.5 (61.7)	15.9 (60.6)	14.5 (58.1)	12.3 (54.1)	10.7 (51.3)	9.5 (49.1)	8.8 (47.8)	9.3 (48.7)	10.2 (50.4)	11.9 (53.4)	13.8 (56.8)	15.6 (60.1)	12.4 (54.3)
Average low °C (°F)	10.9 (51.6)	10.6 (51.1)	9.8 (49.6)	8.1 (46.6)	7.5 (45.5)	6.6 (43.9)	5.8 (42.4)	6.0 (42.8)	6.1 (43.0)	7.4 (45.3)	8.7 (47.7)	10.1 (50.2)	8.1 (46.6)
Record low °C (°F)	0.9 (33.6)	3.6 (38.5)	1.6 (34.9)	−1.0 (30.2)	−2.1 (28.2)	−2.2 (28.0)	−3.8 (25.2)	−2.5 (27.5)	−1.4 (29.5)	−0.8 (30.6)	1.0 (33.8)	3.4 (38.1)	−3.8 (25.2)
Average precipitation mm (inches)	15.7 (0.62)	15.4 (0.61)	25.4 (1.00)	72.4 (2.85)	182.7 (7.19)	230.7 (9.08)	198.4 (7.81)	153.6 (6.05)	84.1 (3.31)	57.2 (2.25)	33.4 (1.31)	21.3 (0.84)	1,090.3 (42.93)
Average precipitation days (≥ 1.0 mm)	2.3	2.3	4.1	7.7	13.5	16.1	15.3	14.0	9.9	8.1	5.4	3.5	102.2
Average relative humidity (%)	74.6	76.3	80.0	82.7	86.0	86.9	85.1	83.9	81.5	80.4	77.2	75.5	80.8
Mean monthly sunshine hours	341	278	255	191	133	114	129	162	197	254	296	331	2,681
Source 1: Dirección Meteorológica de Chile[24][25][26]													
Source 2: World Meteorological Organization (precipitation days and humidity 1981–2010),[27] Ogimet (sun 1981–2010)[28][29]													

Figure 2.13: Matrix relational table[13]

2.2.2 Table interpretation

The goal of this step is to identify the entities and relationships present within the tables. This involves parsing and normalizing the tables. Next, the entities and attributes within the table cells are identified, and relations between previously identified entities and attributes are extracted. Works typically focus on interpreting horizontal relational tables only.

[13]from the Wikipedia article `https://en.wikipedia.org/wiki/Concepci%C3%B3n,_Chile`

2.2.3 Related Work

Detecting and interpreting web tables can be a challenging task due to the varied structures of tables found on the web and the absence of contextual information in some cases.

Limaye et al. (2010) [23] propose a method that applies machine learning techniques to annotate Web tables in the YAGO knowledge base. To do this, they annotate the entities contained in each table cell, associate each table column with a type, and extract relationships between pairs of columns. The method proposed by Limaye et al. (2010), for a dataset of Wikipedia tables (excluding Infoboxes), reaches an accuracy of 83% for entity annotation, 56% for type annotation and 68% for relation annotation.

Another method that infers types and relationships between pairs of columns is proposed by Mulwad et al. (2013) [27], through a probabilistic graphical model. Their approach outperformed the method proposed by Limaye et al. (2010) in terms of accuracy, achieving a relation annotation F-score of 97% compared to the 68% achieved by Limaye et al. However, it should be noted that the experiment was conducted on a limited dataset of only 36 non-Infobox tables extracted from Wikipedia.

Muñoz et al. (2013) [28] propose extracting relations between entities in different columns from DBpedia and thus, from existing relations between entities in the same row, propose new relations in other rows of the table. With this method, 24.4 million raw triples were extracted from Wikipedia's tables with an estimated precision of 52%.

Muñoz et al. (2014) [29] continues with the previous method, testing machine learning methods for classifying correct/incorrect triples extracting 7.9 million novel RDF triples over one million Wikipedia tables with an estimated precision of 81.5% and a F-score of 79.4%.

Luzuriaga (2019) [25] continues with the work of Muñoz et al. (2014), proposing clustering of tables according to their content and structure, with the aim of increasing the quantity and precision of the relations extracted. This method extracted 7.5 million novel triples over a more up-to-date collection of 3.6 million Wikipedia tables, reaching a precision of 70%.

2.3 Mapping languages to RDF

There are different languages for mapping tables, typically those tables stored in Relational Databases (RDBs), to RDF triples. First, we describe the Direct Mapping [1], a W3C recommendation for transforming relational data to RDF. Then, we discuss different mapping languages and their associated processors based on the selection of the current declarative mapping languages used in the comparison framework (Figure 2.14) presented by Iglesias-Molina et al. (2022) [17].

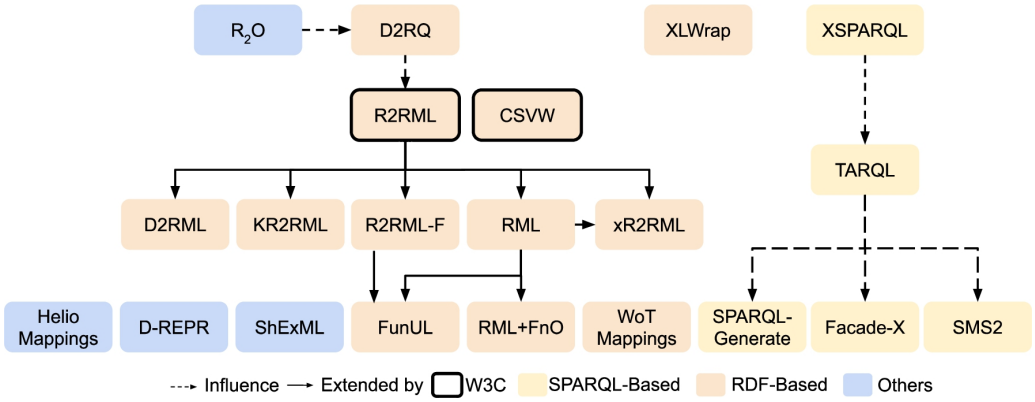

Figure 2.14: Existing mapping languages and their relationships, Iglesias-Molina et al. (2022) [17]

2.3.1 Direct Mapping

This approach takes a Relational Database as input and generates an RDF graph. In this method, each table in the RDB is mapped to an RDF class, each row in the table is mapped to an RDF instance, and each column in the table is mapped to an RDF property.

The Direct Mapping [1] preserves the structure of the RDB in its representation in the RDF graph and requires no explicit mapping.

Another feature is that in an RDF graph, when the URI of a resource is not available, a blank node is generated. For tables that do not have a primary key, a new blank node is generated for each tuple. This enables the preservation of duplicate rows.

An example of Direct Mapping using Listing 2.5 is presented, with Table 2.3 as input, where the primary key of the table is underlined.

Table 2.3: Animals

ID	Name	Description
1	Capybara	rodent
2	Wombat	marsupial

```
<Animals/ID=1> rdf:type <Animals> .
<Animals/ID=1> <Animals#ID> 1.
<Animals/ID=1> <Animals#Name> "Capybara".
<Animals/ID=1> <Animals#Description> "rodent".

<Animals/ID=2> rdf:type <Animals> .
<Animals/ID=2> <Animals#ID> 2.
<Animals/ID=2> <Animals#Name> "Wombat".
<Animals/ID=2> <Animals#Description> "marsupial".
```

Listing 2.5: Direct Mapping example

2.3.2 Declarative Mapping languages to RDF

Iglesias-Molina et al. (2022) categorize RDF mapping languages into three groups based on syntax: RDF-based, SPARQL-based, and others. Our focus will be on the first two categories.

RDF-based mapping languages

These mapping languages are defined as ontologies.

R2RML [13] Is a more flexible variant of the Direct Mapping. This mapping language allows the creation of custom mappings from relational data to RDF. The mappings generated with this language allow viewing relational data in an RDF model, with a structure and lexicon customized by the author of the mapping.

R2RML receives a Relational Database as input and generates an RDF Dataset as output, which represents a collection of RDF graphs. As for the blank nodes, R2RML allows their reuse, avoiding the existence of duplicate rows.

RML (RDF Mapping Language) [14] Is a language used to define mappings between non-RDF sources and RDF inspired by R2RML. RML is more general than R2RML, and allows for defining mappings from several semi-structured forms (JSON, XML, CSV) to RDF, rather than only from RDBs.

There are several tools that implement the RML specification. The RML Implementation report[14] evaluates some of these implementations, testing their coverage of RML's features.

[14] `https://rml.io/implementation-report/`

The tools tested are RMLMapper, CARML, RocketRML, SDM-RDFizer, RMLStreamer, Chimera, and Morph-KGC. Table 2.4 summarizes the results obtained by these tools on the tests with CSV as input data.

Table 2.4: RML Implementation report: Tests cases results with CSV as input data[15]

RML engine	Passed	Failed	Inapplicable
RMLMapper	39	0	0
CARML	35	4	0
RocketRML	25	14	0
SDM-RDFizer	39	0	0
RMLStreamer	39	0	0
Chimera	36	3	0
Morph-KGC	31	8	0

- **RMLMapper**[16] is a Java library utilized for processing RML rules to create RDF data. RMLMapper employs an in-memory approach for loading all data. This processor also offers the capability to remove duplicates from the output and supports the integration of functions within RML mappings[17].
- **RocketRML**[18] is a JavaScript implementation of RMLMapper, created to address specific scenarios involving large XML and JSON files. While RocketRML covers most of the capabilities of RMLMapper, it fails in some of the RML test cases such as generating blank nodes.
- **CARML**[19] (Customized Approach for RML) is a tool for generating RML mappings from templates. CARML allows users to define RML templates using a YAML-based syntax, which can then be compiled into RML mappings. This can be useful for generating mappings for data sources that have a consistent structure. It's worth mentioning that while CARML performs well in most cases, there are a few minor challenges have been observed in certain test cases, such as the generation of triples in the presence of data errors[20].
- **SDM-RDFizer**[21] is a Python library that optimizes the processing of RML Rules using novel physical data structures and operators, proving particularly valuable when dealing with extensive datasets characterized by a high duplication rate.

[15]Results aggregated from `https://rml.io/implementation-report/`
[16]`https://github.com/RMLio/rmlmapper-java`
[17]`https://fno.io/rml/`
[18]`https://github.com/semantifyit/RocketRML`
[19]`https://github.com/carml/carml`
[20]`https://github.com/carml/carml#carml-in-rml-test-cases`
[21]`https://github.com/SDM-TIB/SDM-RDFizer`

- **RMLStreamer**[22] is a Scala-based RML processor built on top of Apache Flink[23] that specializes in managing continuous data streams. It optimizes data ingestion and mapping tasks by enabling parallel processing, allowing the handling of high data volumes.

- **Chimera**[24] is a framework that extends Apache Camel[25] and consists of building blocks for semantic data transformation pipelines. These blocks include graph construction, transformation, validation, and exploitation.

- **Morph-KGC** [4] is an RML engine implemented as a Python library. Morph-KGC aims to reduce the amount of memory and execution time required to process the data into RDF by using mapping partitions. Arenas-Guerrero et al. (2022) [4] define mapping partitions as groups of mapping rules that generate disjoint subsets of the knowledge graph. This engine fails some of the RML test cases, specifically when processing columns in CSV files with special characters and generating triples using IRI values within columns.

  Arenas-Guerrero et al. (2023) [3] implement RML Views for tabular data sources within Morph-KGC, enabling the execution of SQL queries to define computations over the input data.

Given the CSV file "Books.csv" shown in Table 2.5 as input, an example mapping using RML (Listing 2.6) is applied. The result is shown in Listing 2.7.

Table 2.5: Books.csv

Title	Author	Genre	Year
Animal Farm	George Orwell	allegory	1945
Tender is the flesh	Agustina Bazterrica	dystopian novel	2017
Hurricane Season	Fernanda Melchor	literary fiction	2017

[22] `https://github.com/RMLio/RMLStreamer`
[23] `https://flink.apache.org/`
[24] `https://github.com/cefriel/chimera`
[25] `https://camel.apache.org/`

```
<#BooksMapping> a rr:TriplesMap;
  rml:logicalSource [
    rml:source ".../books.csv";
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
    rr:template "http://www.example.org/books/{Title}";
  ];
  rr:predicateObjectMap [
    rr:predicate :title;
    rr:objectMap [
      rml:reference "Title"
    ]
  ];
  rr:predicateObjectMap [
    rr:predicate :year;
    rr:objectMap [
      rml:reference "Year";
      rr:datatype xsd:int
    ]
  ];
  rr:predicateObjectMap [
    rr:predicate :genre;
    rr:objectMap [
       rml:reference "Genre"]
    ];
 rr:predicateObjectMap [
    rr:predicate :writtenBy;
    rr:objectMap [
       rr:template "http://www.example.org/author/{Author}"]
    ].
<#AuthorMapping> a rr:TriplesMap;
  rml:logicalSource [
    rml:source ".../books.csv";
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
    rr:template "http://www.example.org/author/{Author}"
  ];
  rr:predicateObjectMap [
    rr:predicate :name;
    rr:objectMap [
       rml:reference "Author"
    ]
   ].
```

Listing 2.6: RML mapping example

```
<http://www.example.org/books/Animal%20Farm>
  :title "Animal Farm" ;
  :year "1945"^^xsd:int ;
  :genre "allegory" ;
  :writtenBy <http://www.example.org/author/George%20Orwell> .

<http://www.example.org/books/Tender%20is%20the%20flesh>
  :title "Tender is the flesh" ;
  :year "2017"^^xsd:int ;
  :genre "dystopian novel" ;
  :writtenBy <http://www.example.org/author/Agustina%20Bazterrica> .

<http://www.example.org/books/Hurricane%20Season>
  :title "Hurricane Season" ;
  :year "2017"^^xsd:int ;
  :genre "literary fiction" ;
  :writtenBy <http://www.example.org/author/Fernanda%20Melchor> .

<http://www.example.org/author/George%20Orwell> :name "George Orwell" .
<http://www.example.org/author/Agustina%20Bazterrica> :name "Agustina Bazterrica" .
<http://www.example.org/author/Fernanda%20Melchor> :name "Fernanda Melchor" .
```

Listing 2.7: RML mapping example output

FunUL [18] This method aims to extend RML by integrating functions, with a specific emphasis on CSV data as the input. The implementation[26] is currently unavailable.

D2RML [11] Is an extension of R2RML that allows mapping data from heterogeneous sources (such as XML, JSON, and CSV) to RDF through a REST API[27].

SPARQL-based mapping languages

Several mapping languages are based on the SPARQL query language. Some of them are Tarql, SPARQL-Generate, SPARQL Anything, XSPARQL and SMS2.

Tarql[28] Is a command-line tool and a Java library for converting CSV files to RDF using SPARQL 1.1 syntax.

Tarql allows users to specify a SPARQL query that maps columns in a CSV file to

[26] https://github.com/CNGL-repo/RMLProcessor
[27] https://apps.islab.ntua.gr/d2rml/swagger/
[28] https://tarql.github.io/

RDF properties and resources. The resulting RDF data can be output in various formats, including Turtle, N-Triples, and RDF/XML.

To use Tarql, users need to provide a mapping file that contains a SPARQL query specifying the mapping between the CSV columns and the RDF properties and resources. Tarql can be executed via the command line, passing the mapping file and the input CSV file as arguments. The resulting RDF data can then be output to a file or printed to the console.

Using Table 2.5 as input, an example mapping is performed using Tarql (Listing 2.8). The outcome of this operation is displayed in Listing 2.11.

SPARQL-Generate [21] Is a Java implementation on top of Apache Jena[29]. SPARQL-Generate extends SPARQL 1.1 to generate RDF data from various data sources, including SQL, XML, JSON, CSV and HTML. It introduces three additional clauses to SPARQL 1.1: the `GENERATE` clause, which extends and replaces the `CONSTRUCT` clause; the `ITERATOR` clause, designed for extracting and binding keys from the input data; and the `SOURCE` clause, which binds a variable to a document.

SPARQL-Generate can be used on the web playground[30] as well as through an executable JAR[31], which takes a querying file as input.

To illustrate, consider the example mapping presented in Listing 2.10. In this case, the `GENERATE` clause maintains a structure similar to the `CONSTRUCT` clause in Listing 2.8. Additionally, the `ITERATOR` clause binds the columns of the CSV input data with SPARQL variables.

SPARQL Anything[32] It allows querying files in formats such as JSON, HTML, CSV, XML, and more using SPARQL 1.1. It overloads the SPARQL `SERVICE` operator with their virtual endpoint. Within the query file, the service is specified with the syntax:

```
SERVICE <x-sparql-anything:>.
```

An illustration of this concept is presented in Listing 2.9.

The resulting mapping is similar to the one in Tarql (as seen in Listing 2.8) differing primarily in the `WHERE` clause due to the `SERVICE` overloading. In this example, a basic graph pattern for the triplification of the input data is also provided within the `SERVICE` clause. Triples of the form `(fx:properties, fx:[OPTION-NAME], literal or variable value)` can also be defined here to handle the input data. For instance, in this case,

[29] `https://jena.apache.org/`
[30] `https://ci.mines-stetienne.fr/sparql-generate/playground.html`
[31] `https://ci.mines-stetienne.fr/sparql-generate/language-cli.html`
[32] `https://github.com/SPARQL-Anything/sparql.anything`

`fx:csv.null-string` instructs the CSV triplifier not to generate triples with an empty value string in the object positions of the triple.

XSPARQL[33] Is a query language that transforms XML data into RDF data, combining XQuery and SPARQL. It does not support other data formats as input.

Mapping Syntax 2 (SMS2)[34] Is a proprietary software designed to process a variety of inputs, including semi-structured data (such as JSON, CSV, MongoDB, and Elasticsearch) as well as structured data (SQL RDBMS) and transforms it into RDF data. Is based on the SPARQL `CONSTRUCT` query.

```
CONSTRUCT {
  ?book :title ?title ;
        :genre ?genre ;
        :year ?year ;
        :writtenBy ?author .
   ?author :name ?authorName .
}
FROM <file:.../books.csv>
WHERE {
  BIND(URI(CONCAT("http://example.org/book/", MD5(CONCAT(?title, ?author)))) AS ?book)
  BIND(xsd:string(?title) AS ?title)
  BIND(xsd:string(?genre) AS ?genre)
  BIND(xsd:integer(?year) AS ?year)
  BIND(URI(CONCAT("http://example.org/author/", MD5(?author))) AS ?author)
  BIND(xsd:string(?Author) AS ?authorName)
}
```

Listing 2.8: Tarql mapping example

[33] https://www.w3.org/Submission/xsparql-language-specification/

[34] https://docs.stardog.com/virtual-graphs/mapping-data-sources#sms2-stardog-mapping-syntax-2

```
CONSTRUCT {
    ?book :title ?title ;
          :genre ?genre ;
          :year ?year ;
          :writtenBy ?author .
    ?author :name ?authorName .
}
WHERE {
    SERVICE <x-sparql-anything:location=.../books.csv,csv.headers=true> {
        fx:properties  fx:csv.null-string  "" .
        ?c  xyz:Title ?Title;
            xyz:Author ?Author;
            xyz:Genre ?Genre;
            xyz:Year ?Year
    }
    BIND(URI(CONCAT("http://example.org/book/", MD5(CONCAT(?Title, ?Author)))) AS ?book)
    BIND(xsd:string(?Title) AS ?title)
    BIND(xsd:string(?Genre) AS ?genre)
    BIND(xsd:integer(?Year) AS ?year)
    BIND(URI(CONCAT("http://example.org/author/", MD5(?Author))) AS ?author)
    BIND(xsd:string(?Author) AS ?authorName)
}
```

Listing 2.9: SPARQL Anything mapping example

```
GENERATE {
    ?book :title ?title;
          :genre ?genre;
          :year ?year;
          :writtenBy ?author .
    ?author :name ?authorName .
}
ITERATOR iter:CSV(<.../books.csv>, "Title", "Author","Genre","Year") AS ?Title ?Author ?
    Genre ?Year
WHERE {
    BIND(URI(CONCAT("http://example.org/book/", MD5(CONCAT(?Title, ?Author)))) AS ?book)
    BIND(xsd:string(?Title) AS ?title)
    BIND(xsd:string(?Genre) AS ?genre)
    BIND(xsd:integer(?Year) AS ?year)
    BIND(URI(CONCAT("http://example.org/author/", MD5(?Author))) AS ?author)
    BIND(xsd:string(?Author) AS ?authorName)
}
```

Listing 2.10: SPARQL-Generate mapping example

```
<http://example.org/book/b598a5eccc04ce766a986dfd4d149076>
        :title      "Animal Farm" ;
        :genre      "allegory" ;
        :year       1945 ;
        :writtenBy  <http://example.org/author/8e6d6235e87369c7bfe1c7bc238fd7e7> .

<http://example.org/author/8e6d6235e87369c7bfe1c7bc238fd7e7>
        :name   "George Orwell" .

<http://example.org/book/414bc4b5fdaa000150d2e68e5c1447bc>
        :title      "Tender is the flesh" ;
        :genre      "dystopian novel" ;
        :year       2017 ;
        :writtenBy  <http://example.org/author/77f78ff81dcb92578f598a87d9798bbe> .

<http://example.org/author/77f78ff81dcb92578f598a87d9798bbe>
        :name   "Agustina Bazterrica" .

<http://example.org/book/eb86cb6327e7aa45b242e06d20665602>
        :title      "Hurricane Season" ;
        :genre      "literary fiction" ;
        :year       2017 ;
        :writtenBy  <http://example.org/author/83599716447b997712ac2845a390d40a> .

<http://example.org/author/83599716447b997712ac2845a390d40a>
        :name   "Fernanda Melchor" .
```

Listing 2.11: Tarql /SPARQL Anything / SPARQL-Generate mapping example output

2.4 Novelty

The interpretation of web tables is a challenging task that has been the focus of several previous studies. These investigations have primarily focused on devising automated techniques for interpreting web tables, with particular emphasis on studying infoboxes from Wikipedia or using a limited number of Wikipedia tables that are not infoboxes. However, the outcomes of these methods have often faced a trade-off between precision and recall of the relationships extracted.

Manual methods, on the other hand, are more accurate than automated methods, but they require a significant amount of human effort to map tables individually. This process can be tedious and time-consuming, especially when dealing with a large number of tables.

We propose a semi-automated approach using Luzuriaga's clustering as a promising alternative that could achieve high precision with minimal human effort. This approach involves clustering similar tables and applying a single mapping to the group, which enables the extraction of a high number of triples from various tables. The resulting data may be

imported into knowledge bases, where high precision is essential for successful importing.

The selection of an appropriate mapping language is crucial for extracting information from Wikipedia tables to enhance the precision and number of RDF triples extracted. Given the variety of mapping languages and processors available, it is essential to compare them within the context of this specific data corpus. While some comparisons of selected mapping language processors have been made, they use other benchmarks such as GTFS-Madrid-Bench[35] [3, 4], LUBM4OBDA[36] [3], COSMIC dataset[37] [16] and generated CSV documents[38] [21].

Overall, the semi-automated approach, utilizing Luzuriaga's clustering along with a suitable mapping language processor, has the potential to significantly reduce the manual effort required to map web tables and achieve high precision, making it a possible solution for the interpretation of web tables and the integration of data into knowledge bases.

[35] https://github.com/oeg-upm/gtfs-bench
[36] https://github.com/oeg-upm/lubm4obda
[37] https://cancer.sanger.ac.uk/cosmic
[38] http://GenerateData.com

Chapter 3

Data Corpus

The data corpus consists of tables extracted from English Wikipedia in 2019. Luzuriaga [25] preprocessed the dataset by categorizing, parsing, and normalizing it, including filtering out layout tables, infoboxes, and tables without headers, as well as splitting merged cells. The resulting dataset contains 3,631,228 Wikitables from 997,842 articles. Wikitables are relational tables found within Wikipedia articles.

The tables in the dataset are stored as JSON documents and include information about the Wikipedia article, the table title, and a matrix in HTML format that contains the attributes and values in the table cells. To minimize ambiguity, Luzuriaga concatenated multiple header rows after identifying the headers from the HTML tables. To identify similar headers across tables, the header text was stemmed. The clustering process involved merging tables by similar structure and then grouping them in clusters by similar headers.

Additionally, Luzuriaga offers an auxiliary dataset that comprises a subset of 3,598,232 tables containing Wikidata entities that were recognized using the hyperlinks to Wikipedia resources embedded in the cells of the tables.

In this chapter, we analyze the corpus, providing key statistics first with respect to individual tables, and later with respect to table clusters.

3.1 Tables analysis

We begin by analyzing individual tables extracted from Wikipedia articles. Among the articles that include tables, the article[1] characterized by a higher number of tables comprises a total of 676 tables, whereas the majority of articles, amounting to 51.6%, solely exhibit a

[1] `https://en.wikipedia.org/wiki/Monotype_typefaces`

single table (Figure 3.2).

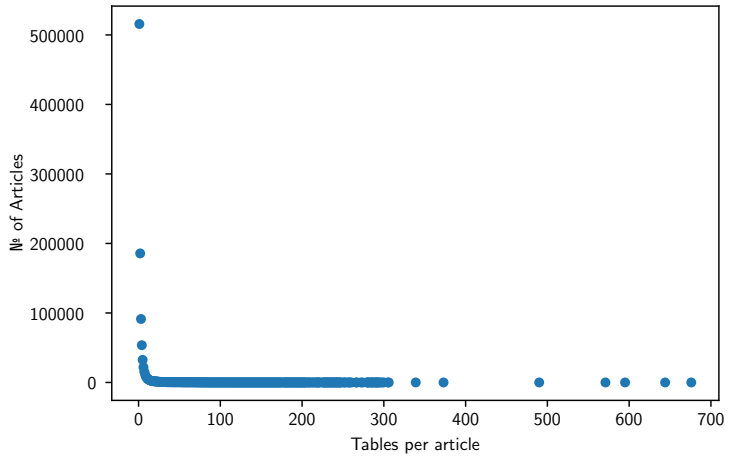

Figure 3.1: Tables per article

	№ of tables
mean	3.6
std	8.9
min	1.0
25%	1.0
50%	1.0
75%	3.0
max	676.0

Figure 3.2: Statistics about tables per article

Out of the aforementioned dataset of 3,598,232 tables that contain Wikidata entities, comprising 36,301,243 rows (excluding header rows), a total of 3,514,373 tables (35,428,465 rows) have an associated Wikidata entity for the article from which the table was extracted. The presence of these entities is of significance, as they can serve as a protagonist column in the tables and facilitate the extraction of triples between the protagonist column as the subject of the triple and other columns in the table, as detailed by Crestan and Pantel [12]. In Table 3.1, the article title is included in the table as a protagonist column, providing contextual information about the music album for the listed songs in the original table. Where available, we show the Wikidata Q identifier for entities in the table, which is identified based on links to Wikipedia articles in the table cells that can then be mapped to Wikidata.

Table 3.1: Example table with the article title as the protagonist column

Article title	№	Title	Length
Origin of Symmetry[Q210910]	1.	New Born[Q929307]	6:03
Origin of Symmetry[Q210910]	2.	Bliss[Q1997359]	4:12
Origin of Symmetry[Q210910]	3.	Space Dementia	6:20
Origin of Symmetry[Q210910]	6.	Citizen Erased	7:19

Table 3.2 presents statistics related to the body section of the tables, including the number of columns, rows, cells, and entities per table. It's important to note that the body section excludes the protagonist column.

Table 3.2: Statistics about the body section of the tables, per table

	№ of columns	№ of rows	№ of cells	№ of entities
mean	5.6	10.1	61.7	17.4
std	3.6	25.3	190.7	56.8
min	2.0	0.0	0.0	0.0
25%	3.0	1.0	6.0	1.0
50%	5.0	5.0	24.0	5.0
75%	6.0	12.0	60.0	15.0
max	372.0	6,195.0	47,634.0	15,027.0

The following observations were made regarding the dataset:

- There are 578,575 tables, consisting of 3,079,519 rows, that contain no entities with Wikidata identifiers other than the article entity.
- Among the tables that contain the article entity, 2,935,798 tables, containing 32,348,946 rows, have at least one other entity.
- A total of 25,673,105 rows have multiple entities, which may or may not include the article entity.
- One table has 372 columns (table: "Calendars with leap week at the end" in the article "Leap week calendar"[2])
- One table has 6,195 body rows (table: "Airline codes" in the article "List of airline codes"[3])
- One table has 47,634 cells (table in the article "List of townlands of County Kerry"[4])
- One table has 15,027 entities, contained in the body section (table in the article "List of members of the Lok Sabha (1952–present)"[5])

In the second-last case, the table contains 2,802 rows, most of which have 5 columns; however, one row has 17 columns, and during normalization, the entire table is expanded to fit this row, generating a high number of empty cells. In the last case, hidden row spans with high values cause entities to be duplicated many times during normalization.

[2] https://en.wikipedia.org/wiki/Leap_week_calendar
[3] https://en.wikipedia.org/wiki/List_of_airline_codes
[4] https://en.wikipedia.org/wiki/List_of_townlands_of_County_Kerry
[5] https://en.wikipedia.org/wiki/List_of_members_of_the_1st_Lok_Sabha

3.2 Cluster analysis

The dataset consists of 1,169,682 clusters of tables having the same header, and the most extensive cluster includes 81,277 tables (Table 3.3). Despite the fact that the majority of the clusters (82.7%) contain only one table, it is noteworthy that many of the most extensive clusters comprise a considerable number of tables (Table 3.4).

Table 3.4 provides insight into the top 10 clusters by the number of tables, offering information about the columns and the number of rows with multiple entities across the columns.

Table 3.6 also provides details on the number of rows with multiple entities within a cell and the number of rows per column containing entities within the cell.

Table 3.3: Statistics about tables per cluster

	№ of tables
mean	3.1
std	143.4
min	1.0
25%	1.0
50%	1.0
75%	1.0
max	81,277.0

Regarding the third cluster in Table 3.4, as a normalization step, Luzuriaga combined the names of columns across multiple header rows to prevent any potential ambiguity, as shown in the Table 3.5.

Figure 3.3 displays example tables from the top 10 largest clusters. Most of the external links displayed in blue within the tables already have a Wikidata identifier as part of the data corpus.

Table 3.4: Top 10 clusters by № of tables

Cluster ID	№ of Tables	№ of Columns	№ of Rows	№ of Rows with multiple entities	Columns
1	81,277	6	1,202,635	526,939	Party, Party, Candidate, Votes, %, ±
2	57,878	5	650,861	283,968	Party, Party, Candidate, Votes, %
3	57,427	2	187,707	106,544	Review scores**Source, Review scores**Rating
4	56,332	3	450,119	44,906	No.,Title, Length
5	51,229	4	496,619	495,033	No., spancol, Position, Player
6	25,449	6	50,898	340	spancol, 1, 2, 3, 4, Total
7	23,472	4	459,839	73,910	Notes, Year, Title, Role
8	20,772	3	65,300	65,226	Ship, Country, Description
9	17,204	4	138,504	36,345	No., Title, Writer(s), Length
10	15,596	2	128,267	120,208	Position, Player

Table 3.5: Example of header join

(a) Original header

Review scores	
Source	Rating
...	...

(b) Joined header

Review scores**Source	Review scores**Rating
...	...

Table 3.6: Top 10 clusters by № of tables

Cluster ID	№ of Tables	№ of Rows	№ of Rows with multiple entities within a cell	Columns	№ of cells with entities
1	81,277	1,202,635	18,843	ArticleEntity	1,202,044
				Party	103,257
				Party	526,277
				Candidate	275,771
				Votes	94,025
				%	18,118
				±	17,616
2	57,878	650,861	11,465	ArticleEntity	650,738
				Party	39,951
				Party	282,992
				Candidate	117,359
				Votes	39,790
				%	12,973
3	57,427	187,707	17	ArticleEntity	183,427
				Review scores**Source	166,352
				Review scores**Rating	56,963
4	56,332	450,119	7,262	ArticleEntity	407,824
				No.	3
				Title	46,098
				Length	8
5	51,229	496,619	202,589	ArticleEntity	494,008
				No.	5
				spancol	492,311
				Position	492,129
				Player	393,229
6	25,449	50,898	0	ArticleEntity	50,126
				spancol	340
				1	0
				2	0
				3	0
				4	0
				Total	0
7	23,472	459,839	4,379	ArticleEntity	458,668
				Notes	24,639
				Year	2540
				Title	51,363
				Role	2,479
8	20,772	65,300	49,045	ArticleEntity	65,300
				Ship	2,106
				Country	64,609
				Description	59,016
9	17,204	138,504	7,893	ArticleEntity	132,698
				No.	0
				Title	19,915
				Writer(s)	23,230
				Length	2
10	15,596	128,267	560	ArticleEntity	122,922
				Position	124,414
				Player	1,695

1996 Victorian state election: Springvale

Party	Candidate	Votes	%	±%
Labor	Eddie Micallef	17,076	55.0	+3.3
Liberal	Miriam Hillenga	11,850	38.2	-1.3
Independent	Andrew Hooper-Nguyen	2,110	6.8	+6.8
	Total formal votes	31,036	96.7	+2.5
	Informal votes	1,054	3.3	-2.5
	Turnout	32,090	94.1	
Two-party-preferred result				
Labor	Eddie Micallef	17,941	57.9	-0.1
Liberal	Miriam Hillenga	13,066	42.1	+0.1
Labor **hold**		**Swing**	-0.1	

(a) Cluster 1

California gubernatorial election, 2006[5][3]

Party	Candidate	Votes	%
Republican	**Arnold Schwarzenegger (incumbent)**	**4,850,157**	**55.9**
Democratic	Phil Angelides	3,376,732	39.0
Green	Peter Camejo	205,995	2.31
Libertarian	Art Olivier	114,329	1.28
Peace and Freedom	Janice Jordan	69,934	0.79
American Independent	Edward Noonan	61,901	0.70
Republican	Robert Newman (write-in)	219	0.00
Independent	James Harris (write-in)	46	0.00
Independent	Donald Etkes (write-in)	43	0.00
Independent	Elisha Shapiro (write-in)	43	0.00
Independent	Vibert Greene (write-in)	18	0.00
Independent	Dealphria Tarver (write-in)	6	0.00
	Invalid or blank votes	219,643	2.47
	Total votes	**8,899,059**	**100.00**
	Turnout		32.77
Republican **hold**			

(b) Cluster 2

Professional ratings

Review scores	
Source	**Rating**
AllMusic	★★★★½[2]
Christgau's Record Guide	B[1]
The Rolling Stone Album Guide	★★★½☆[19]
Spin Alternative Record Guide	7/10[20]

(c) Cluster 3

Original release

No.	Title	Length
1.	"New Born"	6:03
2.	"Bliss"	4:11
3.	"Space Dementia"	6:20
4.	"Hyper Music"	3:21
5.	"Plug In Baby"	3:38
6.	"Citizen Erased"	7:21
7.	"Micro Cuts"	3:38
8.	"Screenager"	4:20
9.	"Darkshines"	4:46
10.	"Feeling Good"	3:18
11.	"Megalomania"	4:39

(d) Cluster 4

No.	Pos.	Nation	Player
--	DF	ARG	Federico Pereyra *(from Karpaty Lviv)*
--	FW	PAN	Gabriel Torres *(from Lausanne Sport)*

No.	Pos.	Nation	Player
--	FW	VEN	José Caraballo *(from Deportivo Lara)*

(e) Cluster 5

Figure 3.3: Table examples from top 10 clusters
3.3a from the Wikipedia article https://en.wikipedia.org/wiki/Electoral_results_for_the_district_of_Springvale
3.3b from the Wikipedia article https://en.wikipedia.org/wiki/2006_California_elections
3.3c from the Wikipedia article https://en.wikipedia.org/wiki/Suffer_(album)
3.3d from the Wikipedia article https://en.wikipedia.org/wiki/Origin_of_Symmetry
3.3e from the Wikipedia article https://en.wikipedia.org/w/index.php?title=C.D._Huachipato&oldid=823964217

Period	1	2	3	4	Total
Australia	16	0	0	0	16
Brazil	0	0	0	8	8

(f) Cluster 6

Film [edit]

Year	Title	Role	Notes
2008	*Pageant*	Herself	Documentary
2016	*Hurricane Bianca*	Ambrosia Salad	Comedy
2018	*Hurricane Bianca 2*	Ambrosia Salad	Comedy

(g) Cluster 7

16 March

Ship	Country	Description
HMT *Maida*	Royal Navy	World War II: The naval trawler struck a mine and sank in the North Sea off Margate east of North Foreland, Kent, with the loss of six of her 12 crew. The survivors were rescued by HMT *Mare* (Royal Navy).[59][61][63]
Osman	Sweden	The cargo ship ran aground in the Skaggerak off Risør, Norway and was wrecked. Ten crew were killed.[64]
Slava	Yugoslavia	World War II: The cargo ship struck a mine and sank in the Bristol Channel south of Nash Point, Glamorgan, United Kingdom (51°19′45″N 3°38′45″W) with the loss of one crewmember.[61][65]

(h) Cluster 8

Side one

No.	Title	Writer(s)	Length
1.	"Nine Lives"	Steven Tyler, Joe Perry, Marti Frederiksen	4:01
2.	"Falling in Love (Is Hard on the Knees)"	Tyler, Perry, Glen Ballard	3:26
3.	"Hole in My Soul"	Tyler, Perry, Desmond Child	6:10
4.	"Taste of India"	Tyler, Perry, Ballard	5:53

(i) Cluster 9

Position	Player
GK	Arindam Bhattacharya[5]
DF	Gouramangi Singh[5]
DF	Dharmaraj Ravanan[5]
MF	Lenny Rodrigues[5]
MF	Eugeneson Lyngdoh[5]

(j) Cluster 10

Figure 3.3: Table examples from top 10 clusters (cont.)
3.3f from the Wikipedia article `https://en.wikipedia.org/wiki/2015_IFAF_World_Championship`
3.3g from the Wikipedia article `https://en.wikipedia.org/wiki/Alyssa_Edwards`
3.3h from the Wikipedia article `https://en.wikipedia.org/wiki/List_of_shipwrecks_in_March_1940`
3.3i from the Wikipedia article `https://en.wikipedia.org/wiki/Nine_Lives_(Aerosmith_album)`
3.3j from the Wikipedia article `https://en.wikipedia.org/wiki/2016_FC_Pune_City_season`

Table 3.7 shows the Wikidata entities that are most frequently mentioned within the tables of Cluster 1 and Cluster 2. This table provides an overview of the topics represented in these clusters and helps identify similarities between the two largest clusters, providing insights for defining mappings within those clusters.

Table 3.7: Top 15 entities mentioned on Cluster 1 and 2

(a) Cluster 1

Label	Entity ID	Count
voter turnout	Q1339847	230787
Two-party-preferred vote	Q7858737	97908
Conservative Party	Q9626	94256
Labour Party	Q9630	81068
Australian Labor Party	Q216082	60988
swing	Q7658517	57630
Liberal Democrats	Q9624	40032
Liberal Party of Australia	Q241149	37806
independent politician	Q327591	30106
voter registration	Q1980404	28767
Liberal Party	Q622441	21558
swing	Q7658512	18372
National Party of Australia	Q946040	14956
Green Party of England and Wales	Q9669	11864
Democratic Party	Q29552	9099

(b) Cluster 2

Label	Entity ID	Count
voter turnout	Q1339847	109992
Conservative Party	Q9626	51565
Labour Party	Q9630	47966
Republican Party	Q29468	39260
Democratic Party	Q29552	38828
swing	Q7658517	27277
Liberal Democrats	Q9624	22146
independent politician	Q327591	16778
voter registration	Q1980404	9741
Liberal Party	Q622441	7052
Green Party of England and Wales	Q9669	6056
incumbent	Q42841	5782
California Democratic Party	Q5020399	4509
Liberal Party of the Philippines	Q1476937	4048
UK Independence Party	Q10647	3992

3.3 Discussion

The analysis reveals the presence of substantial clusters in which a large amount of data can be extracted through a single mapping, despite the significant variation among the tables. Specifically, the largest cluster contains 81,277 tables encompassing 1,202,635 rows, with 536,939 rows exhibiting multi-column entity relationships suitable for relation extraction. Moreover, the columns that feature literal values can serve as objects in newly extracted relationships. As such, it would be feasible to extract at least half a million triples with a single mapping.

Upon closer inspection, certain clusters exhibit shared headers, such as "Party", "Candidate", "Votes" and "%" in Clusters 1 and 2 (Table 3.4). In such cases, the application of analogous mappings may serve to reduce the level of human effort required to define the mappings, while concurrently increasing the volume of extracted relations.

The potential success of this approach will be evaluated in the following chapters.

Chapter 4

Features of Mapping Languages

In this chapter, our aim is to study the suitability of various mapping languages and their features for transforming data from Wikipedia tables to RDF. This involves a comparison of different language processors, with a specific focus on addressing cases involving multivalued tables, managing n-ary relations, among others. To achieve this, the chapter explores mappings for tables sampled from the largest clusters (for details on these clusters, see Section 3.2), identifying features that would be commonly required for such mappings. The prefix declarations of the mappings are omitted; for a list of prefixes used in these mappings, please refer to `https://prefix.cc/`.

4.1 Selection of mapping language processors

Based on the mapping languages discussed in Section 2.3.2, we selected mapping languages with their respective processors that meet specific criteria. The chosen mapping language must accept CSV as a data input, aligning with the handling of the data corpus in CSV format. Additionally, the mapping language must have a readily available implementation and an open-source processor (Table 4.1).

As shown in Table 4.1, neither R2RML nor XSPARQL reads CSV as input. SMS2 is proprietary software, and while D2RML is available for free through the D2RML Processor Service[1], the code is not open-source. As for FunUL, the implementation is currently not available.

In the case of RML processors, we focus on those with better results in the RML Implementation report[2] (Table 2.4). This consideration includes tests related to blank node

[1] `https://apps.islab.ntua.gr/d2rml/swagger/`
[2] `https://rml.io/implementation-report/`

Table 4.1: Mapping Languages selection criteria

Mapping Language	CSV input	Open Source	Implementation
R2RML	✗	✓	✓
RML	✓	✓	✓
FunUL	✓	✓	✗
D2RML	✓	✗	✓
Tarql	✓	✓	✓
SPARQL-Generate	✓	✓	✓
SPARQL Anything	✓	✓	✓
XSPARQL	✗	✓	✓
SMS2	✓	✗	✓

generation since we require a reliable tool for extracting n-ary relations from our data corpus (Table 4.2).

Although RMLStreamer successfully passed the selected tests on the RML Implementation report, while testing the tools we encountered some errors related to a limit on the CSV input length. Additionally, it failed when attempting to use the bulk option to write all triples generated from one input record at once.

Table 4.2: RML processors selection criteria

RML engine	Blank nodes test passed	CSV input no length limit
RMLMapper	✓	✓
CARML	✓	✓
RocketRML	✗	✓
SDM-RDFizer	✓	✓
RMLStreamer	✓	✗
Chimera	✓	✓
Morph-KGC	✓	✓

Given the outlined selection criteria, we have opted for the processors for Tarql, SPARQL-Generate and SPARQL Anything. Regarding the RML mapping language, our selection includes the processors CARML, RMLMapper, SDM-RDFizer, Chimera and Morph-KGC.

4.2 Methodology

The experiments were conducted on an Apple M1 chip 8-core machine with 8GB of RAM and a 256GB disk running macOS Sonoma version 14.1. The processors employed are detailed in Table 4.3. The time of execution was measured using the `time` command in bash to measure microprocessor use. We calculate the sum of `user+sys` time, because it reflects the total direct microprocessor cost of executing the program [3]. The timeout for the experiments was set in one hour of `real` time. This can result in execution time greater than the timeout, as the sum of `user+sys` may be larger than `real` time in multi-core processing.

Extracted triples were categorized using the RDFLib library in Python, and queries were remotely executed using SPARQLWrapper over the Wikidata Query Service.

Table 4.3: Processor and Version Information

Processor	Version
Tarql	1.2
SPARQL-Generate	2.0.12
SPARQL Anything	0.8.2
CARML	0.4.0
RMLMapper	6.2.1
SDM-RDFizer	4.7.2.7
Chimera	2.2
Morph-KGC	2.6.4

4.3 Simple mapping over Horizontal Relational table example

Figure 4.1a displays the horizontal relational table called "Australian federal election, 1901: Senate, Queensland" extracted from Cluster 1. This table is a component of the article "Results of the Australian federal election, 1901 (Senate)" (Q4823761). It provides information regarding the candidates, their respective political parties, the number of votes they received, and various statistics pertaining to the electoral process.

In Figure 4.1b, the extracted Wikidata entities from Figure 4.1a are presented, with the addition of the article entity as a prepended protagonist column (the leftmost column). The Wikidata entities are extracted based on the Wikipedia links present in the corresponding cells of the original table. Sometimes cells may contain more than one link, and thus more

[3] `https://www.ibm.com/docs/en/aix/7.3?topic=performance-using-time-command-measure-microprocessor-use`

than one entity; we will discuss this case in Section 4.4. It is noteworthy that the "Party" column in Figure 4.1a is split into two columns in Figure 4.1b due to the presence of an extra colored column in the original table. Furthermore, there is a replicated entity observed across the columns labeled "1_Party", "2_Party" and "Candidate" which corresponds to the entity Turnout[Q1339847] as depicted in the original table.

Australian federal election, 1901: Senate, Queensland[5]

Party	Candidate	Votes	%	±%
Labour	William Higgs **(elected 1)**	29,452	62.1	+62.1
Labour	Anderson Dawson **(elected 2)**	29,350	61.9	+61.9
Protectionist	James Drake **(elected 3)**	26,552	56.0	+56.0
Labour	James Stewart **(elected 4)**	23,736	50.0	+50.0
Free Trade	John Ferguson **(elected 5)**	23,276	49.1	+49.1
Protectionist	Thomas Glassey **(elected 6)**	22,670	47.8	+47.8
Protectionist	Andrew Thynne	22,001	46.4	+46.4
Protectionist	John Bartholomew	20,624	43.5	+43.5
Protectionist	John Hamilton	18,680	39.4	+39.4
Protectionist	Alfred Cowley	18,265	38.5	+38.5
Protectionist	Edmund Plant	17,028	35.9	+35.9
Protectionist	Thomas Murray-Prior	13,236	27.9	+27.9
Independent	John Hoolan	7,382	15.6	+15.6
Protectionist	David Seymour	4,969	10.5	+10.5
Free Trade	Joseph Ahearne	4,516	9.5	+9.5
Protectionist	Charles Buzacott	2,918	6.2	+6.2
	Total formal votes	284,655 ~47,443 ballots		
	Informal votes	unknown		
	Turnout	unknown		
Party total votes				
Protectionist		166,943	58.6	+58.6
Labour		82,538	29.0	+29.0
Free Trade		27,792	9.8	+9.8
Independent		7,382	2.6	+2.6

(a) Original table[4]

ArticleEntity	1_Party	2_Party	Candidate	Votes	%	±
Q4823761		Q216082	Q8012328			
Q4823761		Q216082	Q118051			
Q4823761		Q1324190	Q6132824			
Q4823761		Q216082	Q6143638			
Q4823761		Q1453449	Q6232973			
Q4823761		Q1324190	Q7790057			
Q4823761		Q1324190	Q4757548			
Q4823761		Q1324190	Q6220926			
Q4823761		Q1324190	Q13520204			
Q4823761		Q1324190	Q4722493			
Q4823761		Q1324190	Q5339744			
Q4823761		Q1324190	Q16030649			
Q4823761		Q327591	Q6239766			
Q4823761		Q1324190	Q5240388			
Q4823761		Q1453449	Q6280927			
Q4823761		Q1324190	Q5078761			
Q4823761	Q1339847	Q1339847	Q1339847			
Q4823761		Q1324190				
Q4823761		Q216082				
Q4823761		Q1453449				
Q4823761		Q327591				

(b) 21108_4.csv

Figure 4.1: Horizontal relational Wikipedia table and the corresponding Wikidata entities

In this example we will apply mappings to extract RDF triples from the table shown in Figure 4.1b. Our focus will be on extracting relations characterized by a single predicate type (member of political party[P102]), with an expected triple pattern as illustrated in Figure 4.2.

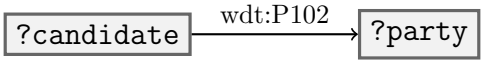

Figure 4.2: Graph pattern of expected triple for Simple mapping example

The RML mapping in Listing 4.1 is defined as follows. First, the CSV file from Figure 4.1b serves as the logical source from which the triples will be mapped. As the subject of the triples we defined the IRIs as the prefix of a Wikidata entity (`http://www.wikidata.org/entity/`) appended with the values found on the "Candidate" column. Similarly, the object map is

[4]from the Wikipedia article `https://en.wikipedia.org/wiki/1901_Australian_Senate_election`

constructed by extracting entities from the "2_Party" column. Then, each subject corresponding to a "Candidate" is linked to the corresponding object representing a "Party" through the Wikidata property member of political party[P102]. We note that this property can be used for historical data, combined in Wikidata with appropriate qualifiers to indicate a start and end date.

Analogous to the previous RML mapping, the Tarql (Listing 4.2), SPARQL Anything (Listing 4.3), and SPARQL-Generate (Listing 4.4) mappings bind the Wikidata prefix for entities with the values in the "Candidate" and "2_Party" columns. As a result, the construct query generates triples in the format `?candidate wdt:P102 ?party` as shown in Figure 4.2.

The output of all mappings is the same, with 17 triples extracted as presented in Listing 4.5. We obtain the triple Turnout[Q1339847] member of political party[P102] Turnout[Q1339847] due to the duplication of that entity across the columns. To prevent this, we can apply the `FILTER (?candidate != ?party)` clause within all SPARQL-based mapping languages or use RMl Views in Morph-KGC with the `WHERE candidate != party` clause embedded in the SQL query. However, achieving the same result in standard RML is not straightforward. A potential approach involves filtering these rows during data preprocessing or introducing a custom function when loading data into RML processors that support function extensions. The triples extracted as well as the time of execution for each processor is presented in Table 4.4.

```
<#Cluster1Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "21108_4.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{Candidate}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P102;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{2_Party}"
        ]
    ].
```

Listing 4.1: RML simple mapping example

```
CONSTRUCT {
    ?candidate wdt:P102 ?party.
}
FROM <file:21108_4.csv>
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Candidate)) AS ?candidate)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?2_Party)) AS ?party)
}
```

Listing 4.2: Tarql simple mapping example

```
CONSTRUCT {
    ?candidate wdt:P102 ?party.
}
WHERE {
    SERVICE <x-sparql-anything:location=21108_4.csv,csv.headers=true> {
        fx:properties  fx:csv.null-string  "" .
        ?c xyz:Candidate ?Candidate;
           xyz:2_Party ?2_Party
    }
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Candidate)) AS ?candidate)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?2_Party)) AS ?party)
}
```

Listing 4.3: SPARQL Anything simple mapping example

```
GENERATE {
    ?candidate wdt:P102 ?party .
}
ITERATOR iter:CSV(<21108_4.csv>, "Candidate", "2_Party") AS ?Candidate ?2_Party
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Candidate)) AS ?candidate)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?2_Party)) AS ?party)
}
```

Listing 4.4: SPARQL-Generate simple mapping example

```
wd:Q8012328 wdt:P102 wd:Q216082 .
wd:Q118051 wdt:P102 wd:Q216082 .
wd:Q6132824 wdt:P102 wd:Q1324190 .
wd:Q6143638 wdt:P102 wd:Q216082 .
wd:Q6232973 wdt:P102 wd:Q1453449 .
wd:Q7790057 wdt:P102 wd:Q1324190 .
wd:Q4757548 wdt:P102 wd:Q1324190 .
wd:Q6220926 wdt:P102 wd:Q1324190 .
wd:Q13520204 wdt:P102 wd:Q1324190 .
wd:Q4722493 wdt:P102 wd:Q1324190 .
wd:Q5339744 wdt:P102 wd:Q1324190 .
wd:Q16030649 wdt:P102 wd:Q1324190 .
wd:Q6239766 wdt:P102 wd:Q327591 .
wd:Q5240388 wdt:P102 wd:Q1324190 .
wd:Q6280927 wdt:P102 wd:Q1453449 .
wd:Q5078761 wdt:P102 wd:Q1324190 .
wd:Q1339847 wdt:P102 wd:Q1339847 .
```

Listing 4.5: Simple mapping example output

Table 4.4: Simple mapping example output comparison

	№ of Triples	Execution time [s]
Tarql	17	2.81
SPARQL-Generate	17	3.11
SPARQL Anything	17	2.57
CARML	17	1.67
RMLMapper	17	5.55
SDM-RDFizer	17	3.00
Chimera	17	4.94
Morph-KGC	17	8.45

4.4 Multivalued table mapping example

In certain scenarios, tables may contain multiple entities within a single cell of a row, known as a multivalued table. In this section, we present various examples and mapping techniques for extracting relations from these entities.

4.4.1 Extraction of relations for all entities within a cell

An example of a multivalued table is presented in Figure 4.3. Here, candidates are grouped by their respective political parties in a concise table and sorted by their position in the election. For this example we will apply the mappings presented in Section 4.3 with a slight modification to facilitate the extraction of triples in the form of Figure 4.2 for each candidate.

Party	Candidate	Votes	%	±%
	Quota	264,381		
Labor	1. Margaret Reynolds **(elected 1)** 2. Mal Colston **(elected 4)** 3. John Bird 4. Ian McLean	729,265	39.4	+0.3
Liberal	1. David MacGibbon **(elected 2)** 2. Warwick Parer **(elected 5)** 3. Ross Cunningham 4. Ann Buchanan 5. Henry Bird 6. Owen Davies	582,766	31.5	+2.2
National	1. Bill O'Chee **(elected 3)** 2. De-Anne Kelly 3. Teresa Cobb	268,809	14.5	+0.9
Democrats	1. John Woodley **(elected 6)** 2. Jonathan Cornish 3. Gayle Woodrow 4. Tony Walters	130,405	7.1	-5.4
Greens	1. Drew Hutton 2. Colin Hunt 3. Naomi Spencer	59,303	3.2	+3.2

(a) Original table[5]

ArticleEntity	1_Party	2_Party	Candidate	Votes	%	±
Q30600597	Q1260232	Q1260232	Q1260232			
Q30600597		Q216082	"Q6759833, Q15506241"			
Q30600597		Q241149	"Q5236984, Q6035315"			
Q30600597		Q946040	"Q4910376, Q5243805"			
Q30600597		Q781392	Q6264818			
Q30600597		Q781486	Q5307201			

(b) 71134_7.csv

Figure 4.3: Multivalued Wikipedia table and the corresponding Wikidata entities

Given that Tarql, SPARQL-Generate, and SPARQL Anything are built upon the Apache Jena framework, we can use the function `apf:strSplit`[6] on the Candidate column. This function splits the string and then binds each result to a variable. In Table 4.5 we can see how this function works when applied to the Candidate column in Figure 4.3b. It is noteworthy that each row with multiple entities is replicated as many times as there are entities in the cell. This replication occurs in order to capture each individual entity resulting from the string split process as a different row. The mappings are presented in Listings 4.6, 4.7 and 4.8.

The output contains 9 triples of the form `?candidate wdt:P102 ?party`, one for each entity present in the Candidate column of the table (Listing 4.11). Table 4.6 displays the № of extracted triples and the corresponding execution times for each processor.

[5]abridged from the Wikipedia article `https://en.wikipedia.org/wiki/1993_Australian_Senate_election`

[6]`https://jena.apache.org/documentation/query/library-propfunc.html`

Table 4.5: Split example

ArticleEntity	1_Party	2_Party	Candidate	CandidateSplit
Q30600597	Q1260232	Q1260232	Q1260232	Q1260232
Q30600597		Q216082	"Q6759833, Q15506241"	Q6759833
Q30600597		Q216082	"Q6759833, Q15506241"	Q15506241
Q30600597		Q241149	"Q5236984, Q6035315"	Q5236984
Q30600597		Q241149	"Q5236984, Q6035315"	Q6035315
Q30600597		Q946040	"Q4910376, Q5243805"	Q4910376
Q30600597		Q946040	"Q4910376, Q5243805"	Q5243805
Q30600597		Q781392	Q6264818	Q6264818
Q30600597		Q781486	Q5307201	Q5307201

In the context of RML, RMLMapper provides the default function `grel:string_split`[7] that can be used to split the cell and extract triples for individual entities. However, this function is not working on the latest RMLMapper version (RMLMapper v6.2.1).

In the case of Morph-KGC, it includes the default function:

`morph-kgc:string_split_explode`.[8]

This function provides the same output as `apf:strSplit`. The issue comes while applying the URL template to the Candidate split values. When employing the mapping outlined in Listing 4.9, the template is solely applied to the first candidate in the cell. To address this challenge, we lookup for a default function to transform values into URLs. The only available option was the function `to_upper_case_url`[9], which can prepend `http://` to the input value. However, our requirement is to prepend `http://www.wikidata.org/entity/` to each Candidate entity. While this function is beneficial, it lacks the necessary customization for our specific use case.

To resolve the issue, we opted for the default function `grel:string_replace` [10]. With this function we first replace each comma in the Candidate cell with:

`",http://www.wikidata.org/entity/"`

then we apply the `morph-kgc:string_split_explode` function. The corresponding mapping is illustrated in Listing 4.10, while the output is presented in Listing 4.11.

[7] `https://users.ugent.be/~bjdmeest/function/grel.ttl#`

[8] `https://github.com/morph-kgc/morph-kgc/blob/main/src/morph_kgc/fnml/built_in_functions.py`

[9] `https://github.com/morph-kgc/morph-kgc/blob/main/src/morph_kgc/fnml/built_in_functions.py`

[10] `https://users.ugent.be/~bjdmeest/function/grel.ttl#`

It's important to highlight that neither CARML, SDM-RDFizer, nor Chimera include a similar function as part of their default functionality.

```
CONSTRUCT {
    ?candidate wdt:P102 ?party.
}
FROM <file:71134_7.csv>
WHERE {
    OPTIONAL {?CandidateSplit apf:strSplit (?Candidate ",") }
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?CandidateSplit)) AS ?candidate)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?2_Party)) AS ?party)
}
```

Listing 4.6: Tarql mapping for multivalued table

```
GENERATE {
    ?candidate wdt:P102 ?party .
}
ITERATOR iter:CSV(<71134_7.csv>, "Candidate", "2_Party") AS ?Candidate ?2_Party
WHERE {
    OPTIONAL {?CandidateSplit apf:strSplit (?Candidate ",")}
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?CandidateSplit)) AS ?candidate)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?2_Party)) AS ?party)
}
```

Listing 4.7: SPARQL-Generate mapping for multivalued table

```
CONSTRUCT {
    ?candidate wdt:P102 ?party.
}
WHERE {
    SERVICE <x-sparql-anything:location=71134_7.csv,csv.headers=true> {
        fx:properties  fx:csv.null-string  "" .
        ?c  xyz:Candidate ?Candidate;
            xyz:2_Party ?2_Party
    }
    OPTIONAL {?CandidateSplit apf:strSplit (?Candidate ",") }
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?CandidateSplit)) AS ?candidate)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?2_Party)) AS ?party)
}
```

Listing 4.8: SPARQL Anything mapping for multivalued table

```
<MultivaluedTableMapping>
    rml:logicalSource [
        rml:source "71134_7.csv";
        rml:referenceFormulation rml:CSV
    ];
    rml:subjectMap [
        rml:functionExecution <#Execution> ;
        rml:termType rml:IRI
    ];
    rml:predicateObjectMap [
        rml:predicate wdt:P102;
        rml:objectMap [
            rml:template "http://www.wikidata.org/entity/{2_Party}"
            ]
    ] .

<#Execution>
    rml:function morph-kgc:string_split_explode ;
    rml:input [
        rml:parameter grel:valueParam ;
        rml:inputValueMap [
            rml:template "http://www.wikidata.org/entity/{Candidate}"
        ]
    ],
    [
        rml:parameter grel:param_string_sep ;
        rml:inputValue ","
    ] .
```

Listing 4.9: Morph-KGC initial mapping for multivalued table

```
<MultivaluedTableMapping>
    rml:logicalSource [
        rml:source "71134_7.csv";
        rml:referenceFormulation rml:CSV
    ];
    rml:subjectMap [
        rml:functionExecution <#Execution> ;
        rml:termType rml:IRI
    ];
    rml:predicateObjectMap [
        rml:predicate wdt:P102;
        rml:objectMap [
            rml:template "http://www.wikidata.org/entity/{2_Party}"
            ]
    ] .

<#Execution>
    rml:function morph-kgc:string_split_explode ;
    rml:input [
        rml:parameter grel:valueParam ;
        rml:inputValueMap [
            rml:functionExecution <#Execution2> ;
            rml:return grel:stringOut
        ]
    ],
    [
        rml:parameter grel:param_string_sep ;
        rml:inputValue ","
    ] .

<#Execution2> a rml:Execution ;
    rml:function grel:string_replace ;
    rml:input [
        a rml:Input ;
        rml:parameter grel:valueParam ;
        rml:inputValueMap [
            rml:template "http://www.wikidata.org/entity/{Candidate}"
        ]
    ],
    [
        a rml:Input ;
        rml:parameter grel:param_find ;
        rml:inputValue ","
    ] ,
    [
        a rml:Input ;
        rml:parameter grel:param_replace  ;
        rml:inputValue ",http://www.wikidata.org/entity/"
    ] .
```

Listing 4.10: Morph-KGC mapping for multivalued table

```
wd:Q1260232 wdt:P102 wd:Q1260232 .
wd:Q6759833 wdt:P102 wd:Q216082 .
wd:Q15506241 wdt:P102 wd:Q216082 .
wd:Q5236984 wdt:P102 wd:Q241149 .
wd:Q6035315 wdt:P102 wd:Q241149 .
wd:Q4910376 wdt:P102 wd:Q946040 .
wd:Q5243805 wdt:P102 wd:Q946040 .
wd:Q6264818 wdt:P102 wd:Q781392 .
wd:Q5307201 wdt:P102 wd:Q781486 .
```

Listing 4.11: Tarql, SPARQL-Generate, SPARQL Anything and Morph-KGC results for multivalued table output

Table 4.6: Multivalued table mapping example output comparison

	№ of Triples	Execution time [s]
Tarql	9	3.25
SPARQL-Generate	9	3.30
SPARQL Anything	9	2.56
CARML	-	-
RMLMapper	-	-
SDM-RDFizer	-	-
Chimera	-	-
Morph-KGC	9	8.45

4.4.2 Extraction of relations for the n^{th} entity within a cell

If our objective is to extract triples considering only the first or last entity within a cell containing multiple entities, we can use the SPARQL `REPLACE` function. For the example in Figure 4.3, we can apply `BIND(REPLACE(?Candidate,",Q[0-9]*","") AS ?first)` to extract the first entity within the cells of the Candidate column. This is particularly relevant when our objective is to extract the initial candidate in the cell, corresponding to the leading one for each party. Alternatively, `BIND(REPLACE(?Candidate,"Q[0-9]*,","") AS ?last)` allows us to extract the last entity within the cells of the Candidate column. These clauses will be inside the `WHERE` clause of the SPARQL-based mapping languages. These operators bind the first or last entity of `?Candidate` to the `?first` or `?last` variable, respectively.

Regarding RML, Morph-KGC allows us to extract the first or last entity within a cell using RML Views. Inside the SQL query, we can use the SQL function `split_part` by changing the index of the entity we want to extract. In this way, the functions used will be `split_part(Candidate, ',' , 1) AS first` to extract the first entity within a cell or `split_part(Candidate, ',' , -1) AS last` to extract the last entity.

Listing 4.12 illustrates the Tarql mapping designed for extracting the last entity within the cells of the Candidate column, with the mappings for SPARQL-Generate and SPARQL Anything following a similar structure. Listing 4.13 presents the equivalent Morph-KGC mapping.

The extracted triples from Figure 4.3 considering the last `?candidate` are presented in Listing 4.15. Equally, the mappings for capturing the first entity within the cells of the Candidate column exhibit an analogous structure, requiring only a modification in the previously mentioned `REPLACE` clause or changing the index in the function `split_part` in the case of Morph-KGC. The extracted triples from Figure 4.3 considering the first `?candidate` are presented in Listing 4.14.

We will explore another instance in Section 5.1.4, focusing on extracting only the first entity within a cell. This example pertains to information about songs in their respective albums. Our objective will be to extract the initial entity within the Title column, assuming it corresponds to the name of the song entity, as depicted in Table 4.7. As observed, the Title column contains five entities; however, it is noteworthy that only the first entity aligns with the actual title of the song. The mappings applied and the triples extracted from Table 4.7 are discussed in Section 5.1.4.

Table 4.7: Extraction of the first entity example [11]

No	Title	Length
1.	"Savory" [Q7428417] (Jawbox[Q1549376] cover) (featuring Jonah Matranga[Q1644952], Shaun Lopez[Q7490901] and Chris Robyn of Far[Q455060])	4:37

```
CONSTRUCT {
    ?candidate wdt:P102 ?party.
}
FROM <file:71134_7.csv>
WHERE {
    BIND(REPLACE(?Candidate,"Q[0-9]*,","") AS ?last)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?last)) AS ?candidate)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?2_Party)) AS ?party)
}
```

Listing 4.12: Tarql mapping for extracting the last entity of a cell

[11] Abridged from the Wikipedia article `https://en.wikipedia.org/wiki/B-Sides_%26_Rarities_(Deftones_album)`

```
<LastEntityMapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
        SELECT split_part(Candidate, ',' , -1) as last, "2_Party" as Party
        FROM '71134_7.csv'
        """ ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{last}"
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P102;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Party}"
        ]
    ].
```

Listing 4.13: Morph-KGC mapping for extracting the last entity of a cell

```
wd:Q5236984  wdt:P102  wd:Q241149 .
wd:Q1260232  wdt:P102  wd:Q1260232 .
wd:Q6759833  wdt:P102  wd:Q216082 .
wd:Q5307201  wdt:P102  wd:Q781486 .
wd:Q6264818  wdt:P102  wd:Q781392 .
wd:Q4910376  wdt:P102  wd:Q946040 .
```

Listing 4.14: Tarql, SPARQL-Generate, SPARQL Anything and Morph-KGC results for extracting the first entity of a cell output

```
wd:Q15506241  wdt:P102  wd:Q216082 .
wd:Q6035315  wdt:P102  wd:Q241149 .
wd:Q1260232  wdt:P102  wd:Q1260232 .
wd:Q5243805  wdt:P102  wd:Q946040 .
wd:Q5307201  wdt:P102  wd:Q781486 .
wd:Q6264818  wdt:P102  wd:Q781392 .
```

Listing 4.15: Tarql, SPARQL-Generate, SPARQL Anything and Morph-KGC results for extracting the last entity of a cell output

To extract the n^{th} entity from a cell containing multiple entities, we can apply the SPARQL `REPLACE` function, as before, manipulating the `?Candidate` variable using the `STRAFTER` function. To obtain the $(n+1)^{\text{th}}$ entity, we apply the `STRAFTER` function n times consecutively to the `?Candidate` variable. For instance, to retrieve the second entity in `?Candidate`, we can employ the following SPARQL expression, as exemplified in Listing 4.16, which presents a Tarql mapping

```
REPLACE(STRAFTER(?Candidate,","),",Q[0-9]*","").
```

Similarly, to extract the third entity, we can use:

```
REPLACE(STRAFTER(STRAFTER(?Candidate,","),","),",Q[0-9]*","").
```

The mappings for SPARQL-Generate and SPARQL Anything are analogous.

In the case of Morph-KGC, we can apply the SQL function `split_part` as before, changing the index to n:

```
split_part(Candidate, ',' , n)
```

Listing 4.17 presents the Morph-KGC mapping for extracting the second entity.

Table 4.8 provides the № of triples extracted from Figure 4.3b, taking into account the mapping of either the 1st, the 2nd, or last entity within the Candidate column. It also includes the corresponding execution times for each processor. The output triples are displayed in Listing 4.18.

```
CONSTRUCT {
    ?candidate wdt:P102 ?party.
}
FROM <file:71134_7.csv>
WHERE {
    BIND(REPLACE(STRAFTER(?Candidate,","),",Q[0-9]*","") AS ?second)
    FILTER(BOUND(?second)  && STRLEN(?second)>0)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?second)) AS ?candidate)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?2_Party)) AS ?party)
}
```

Listing 4.16: Tarql mapping for extracting the second entity of a cell

```
<NthEntityMapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
        SELECT split_part(Candidate, ',' , 2) as second, "2_Party" as Party
        FROM '71134_7.csv'
        """ ];
        rr:subjectMap [
            rr:template "http://www.wikidata.org/entity/{second}"
        ];
        rr:predicateObjectMap [
            rr:predicate wdt:P102;
            rr:objectMap [
                rr:template "http://www.wikidata.org/entity/{Party}"
            ]
        ].
```

Listing 4.17: Morph-KGC mapping for extracting the second entity of a cell

```
wd:Q15506241  wdt:P102  wd:Q216082 .
wd:Q5243805  wdt:P102  wd:Q946040 .
wd:Q6035315  wdt:P102  wd:Q241149 .
```

Listing 4.18: Tarql, SPARQL-Generate, SPARQL Anything and Morph-KGC results for extracting the second entity of a cell output

Table 4.8: Mapping the nth entity within a cell example output comparison

	1st entity		2nd entity		Last entity	
	№ of Triples	time [s]	№ of Triples	time [s]	№ of Triples	time [s]
Tarql	6	2.83	3	2.75	6	2.82
SPARQL-Generate	6	3.09	3	3.17	6	3.11
SPARQL Anything	6	2.47	3	2.50	6	2.45
CARML	-	-	-	-	-	-
RMLMapper	-	-	-	-	-	-
SDM-RDFizer	-	-	-	-	-	-
Chimera	-	-	-	-	-	-
Morph-KGC	6	7.60	3	7.55	6	7.82

Currently, there is no direct method for mapping the n^{th} entity within a cell in CARML, RMLMapper, SDM-RDFizer and Chimera processors. One potential approach is to employ custom pre-mapping functions to preprocess the input data.

4.4.3 Extraction of relations between entities within a cell

Figure 4.4 displays a multivalued table, presenting a list of sports events along with their respective gold, silver, and bronze medalists. In the columns for gold, silver, and bronze, there's an entity associated with a flag followed by the name of the athlete. In such instances, our objective is to extract relations between the entities within a cell. In this example specifically, we aim to extract triples following the graph pattern outlined in Figure 4.5.

For extracting relations between entities within a cell, we will follow a similar approach to that explained in Subsection 4.4.2. In this example, we will extract a relation between the first and the second entity within the cells in the "Gold", "Silver", and "Bronze" columns.

Listing 4.19 shows the Tarql mapping designed for extracting relations between the entities within the cells of the "Gold", "Silver", and "Bronze" columns, with the mappings for SPARQL-Generate and SPARQL Anything following a similar structure. Listing 4.20 presents the equivalent Morph-KGC mapping.

[12]from the Wikipedia article `https://en.wikipedia.org/wiki/Squash_at_the_2009_World_Games`

Event	Gold	Silver	Bronze
Men's singles[2] *details*	Nick Matthew	James Willstrop	Mohd Azlan Iskandar
Women's singles[3] *details*	Nicol David	Natalie Grinham	Omneya Abdel Kawy

(a) Original table[12]

ArticleEntity	Event	Gold	Silver	Bronze
Q3080197	Q7582145	"Q145, Q571095"	"Q145, Q525287"	"Q833, Q1726993"
Q3080197	Q7582146	"Q833, Q867189"	"Q55, Q2247692"	"Q79, Q3660925"

(b) 801513_2.csv

Figure 4.4: Multivalued Wikipedia table and the corresponding Wikidata entities

Figure 4.5: Graph pattern of expected triple for Figure 4.4

As observed in Listing 4.20, extracting the selected entities within a cell using the `split_part` function with RML Views is straightforward, unlike the process involving the use of `REPLACE` and `STRAFTER` in SPARQL-based mappings. With `split_part`, we simply adjust the index parameter. However, the mapping becomes lengthy as we need to define a triplesmap for each subject of the triples.

Table 4.9 provides the № of triples extracted from Figure 4.4b. It also includes the corresponding execution times for each processor. The output triples are displayed in Listing 4.21.

```
CONSTRUCT {
    ?athleteGold wdt:P27 ?countryGold.
    ?athleteSilver wdt:P27 ?countrySilver.
    ?athleteBronze wdt:P27 ?countryBronze.
}
FROM <file:../Input/801513_2.csv>
WHERE {
    #Gold
    BIND(REPLACE(STRAFTER(?Gold,","),",Q[0-9]*","") AS ?secondEntityGold)
    BIND ( IF (BOUND(?secondEntityGold) && strlen(?secondEntityGold)>0, URI(CONCAT('http:
    //www.wikidata.org/entity/', ?secondEntityGold)),?undef) AS ?athleteGold )
    BIND(REPLACE(?Gold,",Q[0-9]*","") AS ?firstEntityGold)
    BIND ( IF (BOUND(?firstEntityGold) && strlen(?firstEntityGold)>0, URI(CONCAT('http://
    www.wikidata.org/entity/', ?firstEntityGold)),?undef) AS ?countryGold )
    #Siver
    BIND(REPLACE(STRAFTER(?Silver,","),",Q[0-9]*","") AS ?secondEntitySilver)
    BIND ( IF (BOUND(?secondEntitySilver) && strlen(?secondEntitySilver)>0, URI(CONCAT('
    http://www.wikidata.org/entity/', ?secondEntitySilver)),?undef) AS ?athleteSilver )
    BIND(REPLACE(?Silver,",Q[0-9]*","") AS ?firstEntitySilver)
    BIND ( IF (BOUND(?firstEntitySilver) && strlen(?firstEntitySilver)>0, URI(CONCAT('
    http://www.wikidata.org/entity/', ?firstEntitySilver)),?undef) AS ?countrySilver )
    #Bronze
    BIND(REPLACE(STRAFTER(?Bronze,","),",Q[0-9]*","") AS ?secondEntityBronze)
    BIND ( IF (BOUND(?secondEntityBronze) && strlen(?secondEntityBronze)>0, URI(CONCAT('
    http://www.wikidata.org/entity/', ?secondEntityBronze)),?undef) AS ?athleteBronze )
    BIND(REPLACE(?Bronze,",Q[0-9]*","") AS ?firstEntityBronze)
    BIND ( IF (BOUND(?firstEntityBronze) && strlen(?firstEntityBronze)>0, URI(CONCAT('
    http://www.wikidata.org/entity/', ?firstEntityBronze)),?undef) AS ?countryBronze )
}
```

Listing 4.19: Tarql mapping for extracting relations between entities within a cell

```
<RelationsBetweenEntitiesMapping-Gold> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
            SELECT split_part(Gold, ',' , 2) as athleteGold, split_part(Gold, ',' , 1) as
     countryGold
            FROM '../Input/801513_2.csv'
        """ ];
        rr:subjectMap [
            rr:template "http://www.wikidata.org/entity/{athleteGold}"
        ];
        rr:predicateObjectMap [
            rr:predicate wdt:P27;
            rr:objectMap [
                rr:template "http://www.wikidata.org/entity/{countryGold}"
            ]
        ].
<RelationsBetweenEntitiesMapping-Silver> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
            SELECT split_part(Silver, ',' , 2) as athleteSilver, split_part(Silver, ',' ,
     1) as countrySilver
            FROM '../Input/801513_2.csv'
        """ ];
        rr:subjectMap [
            rr:template "http://www.wikidata.org/entity/{athleteSilver}"
        ];
        rr:predicateObjectMap [
            rr:predicate wdt:P27;
            rr:objectMap [
                rr:template "http://www.wikidata.org/entity/{countrySilver}"
            ]
        ].
<RelationsBetweenEntitiesMapping-Bronze> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
            SELECT split_part(Bronze, ',' , 2) as athleteBronze, split_part(Bronze, ',' ,
     1) as countryBronze
            FROM '../Input/801513_2.csv'
        """ ];
        rr:subjectMap [
            rr:template "http://www.wikidata.org/entity/{athleteBronze}"
        ];
        rr:predicateObjectMap [
            rr:predicate wdt:P27;
            rr:objectMap [
                rr:template "http://www.wikidata.org/entity/{countryBronze}"
            ]
        ].
```

Listing 4.20: Morph-KGC mapping for extracting relations between entities within a cell

```
wd:Q571095 wdt:P27 wd:Q145 .
wd:Q525287 wdt:P27 wd:Q145 .
wd:Q1726993 wdt:P27 wd:Q833 .
wd:Q867189 wdt:P27 wd:Q833 .
wd:Q2247692 wdt:P27 wd:Q55 .
wd:Q3660925 wdt:P27 wd:Q79 .
```

Listing 4.21: Tarql, SPARQL-Generate, SPARQL Anything and Morph-KGC results for extracting relations between entities within a cell output

Table 4.9: Extraction of relations between entities within a cell example output comparison

	№ of Triples	Execution time [s]
Tarql	6	1.12
SPARQL-Generate	6	1.33
SPARQL Anything	6	1.05
CARML	-	-
RMLMapper	-	-
SDM-RDFizer	-	-
Chimera	-	-
Morph-KGC	6	7.97

4.5 Extraction of n-ary relations example

Figure 4.6a illustrates a table from the Wikipedia article named "Amanda Hale". This table provides information about Amanda Hale's film credits, including details such as the film title, her role in each production, and additional notes. Figure 4.6b shows the entities for some of the values in Figure 4.6a. The identifier for the entity Amanda Hale (Q453921) is prepended as a protagonist column. Our objective with this table is to extract triples following the graph pattern outlined in Figure 4.7.

To represent the relationship between an actor, their role, and a specific film title, we employ an n-ary relation in the form of an RDF blank node.

In the RML mapping, we specify the blank node as the subject of the triples map `<#Cast>`. To utilize this blank node in the object position of the triple `?Title p:P161 _:bn` we use `<#Cast>` as the parent triples map and perform a join operation (`rr:joinCondition`) with `<#BnodesMapping>`, linking both triples maps by `?Title`.

In RMLMapper and Chimera, the blank node identifier is automatically generated (Listing 4.25), while in CARML, SDM-RDFizer and Morph-KGC, we must specify a template to assign a unique identifier to each node (Listing 4.26). We've defined this identifier as a concatenation of the variables ArticleEntity, Title, and Role, ensuring a unique key for each distinct n-ary relation of the input data. However, since the input data is incomplete, some of these variables may be missing. In such cases, the blank node isn't generated, resulting in fewer output triples for CARML, SDM-RDFizer and Morph-KGC when compared to other tools (Table 4.10).

In the context of the SPARQL-based mapping languages, the graph pattern is specified within the `CONSTRUCT` or `GENERATE` clause, where the blank node is explicitly identified as `_:cast`. In the case of SPARQL Anything, within the `SERVICE` clause, the variables `?Role` and `?Title` are defined as `OPTIONAL` (Listing 4.23). Consequently, even if `?Role` is missing, the process still generates triples, with the available variables linked to the blank node. This is necessary because if not stated, the variables are set as mandatory within the `SERVICE` clause. This clause is not necessary in Tarql (Listing 4.22) and SPARQL-Generate (Listing 4.24).

The triples extracted as well as the time of execution for each processor is presented in Table 4.10.

[13] abridged from the Wikipedia article `https://en.wikipedia.org/wiki/Amanda_Hale`

Year	Title	Role	Notes
2013	*The White Queen*	Margaret Beaufort	BBC One
	Being Human	Lady Mary	BBC Three
	Dates	Helen	Episode #1.8
	The Invisible Woman	Fanny Ternan	
	Sense and Sensibility	Elinor Dashwood	BBC Radio 4

(a) Original table[13]

ArticleEntity	Year	Title	Role	Notes
Q453921		Q9087190	Q229202	Q191472
Q453921		Q2031748	Q6563552	Q687427
Q453921		Q13427026		
Q453921		Q7742499		
Q453921		Q274744	Q385299	Q795598

(b) 1017281_1.csv

Figure 4.6: Wikipedia table and the corresponding Wikidata entities

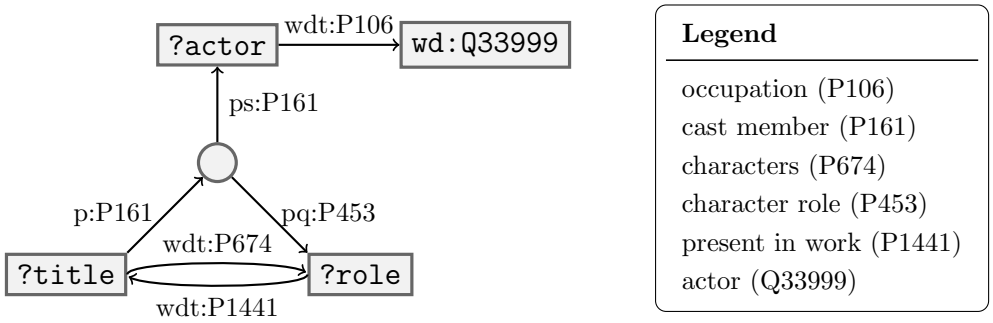

Figure 4.7: Graph pattern of expected triples, including n-ary relations

```
CONSTRUCT {
    ?title wdt:P674 ?role.
    ?role  wdt:P1441 ?title.
    ?title p:P161 _:cast.
    _:cast ps:P161 ?actor;
           pq:P453 ?role.
    ?actor wdt:P106 wd:Q33999.
}
FROM <file:../Input/bnodesMapping.csv>
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Title)) AS ?title)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Role)) AS ?role)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?ArticleEntity)) AS ?actor)
}
```

Listing 4.22: Tarql mapping for the extraction of n-ary relations example

```
CONSTRUCT {
    ?title wdt:P674 ?role.
    ?role  wdt:P1441 ?title.
    ?title p:P161 _:cast.
    _:cast ps:P161 ?actor;
           pq:P453 ?role.
    ?actor wdt:P106 wd:Q33999.
}
WHERE {
    SERVICE <x-sparql-anything:location=../Input/bnodesMapping.csv,csv.headers=true> {
        fx:properties  fx:csv.null-string  "" .
        ?c        xyz:ArticleEntity ?ArticleEntity.
        OPTIONAL{?c xyz:Title ?Title}.
        OPTIONAL{?c xyz:Role ?Role}.
    }
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Title)) AS ?title)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Role)) AS ?role)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?ArticleEntity)) AS ?actor)
}
```

Listing 4.23: SPARQL Anything mapping for the extraction of n-ary relations example

```
GENERATE {
    ?title wdt:P674 ?role.
    ?role  wdt:P1441 ?title.
    ?title p:P161 _:cast.
    _:cast ps:P161 ?actor;
           pq:P453 ?role.
    ?actor wdt:P106 wd:Q33999.
}
ITERATOR iter:CSV(<../Input/bnodesMapping.csv>, "ArticleEntity", "Title","Role") AS ?
    ArticleEntity ?Title ?Role
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Title)) AS ?title)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Role)) AS ?role)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?ArticleEntity)) AS ?actor)
}
```

Listing 4.24: SPARQL-Generate mapping for the extraction of n-ary relations example

```
<#BnodesMapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source  "../Input/bnodesMapping.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{Title}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P674;
        rr:objectMap [
```

```
            rr:template "http://www.wikidata.org/entity/{Role}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate p:P161;
        rr:objectMap [
            rr:parentTriplesMap <#Cast>;
            rr:joinCondition [
                rr:child "Title";
                rr:parent "Title";
            ];
        ]
    ].
<#Cast> a rr:TriplesMap;
    rml:logicalSource [
        rml:source  "../Input/bnodesMapping.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode;
    ];
    rr:predicateObjectMap [
        rr:predicate ps:P161;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{ArticleEntity}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate  pq:P453;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Role}"
        ]
    ].
<#Role> a rr:TriplesMap;
    rml:logicalSource[
        rml:source  "../Input/bnodesMapping.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{Role}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P1441;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Title}"
        ]
    ].
<#Actor> a rr:TriplesMap;
    rml:logicalSource [
        rml:source  "../Input/bnodesMapping.csv";
        rml:referenceFormulation ql:CSV
```

```
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{ArticleEntity}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P106;
        rr:objectMap [
            rr:constant wd:Q33999
        ]
    ].
```

Listing 4.25: RML mapping for the extraction of n-ary relations example

```
<#BnodesMapping> a rr:TriplesMap;
...
<#Cast> a rr:TriplesMap;
    rml:logicalSource [...];
    rr:subjectMap [
        rr:template "{ArticleEntity}_{Title}_{Role}";
        rr:termType rr:BlankNode
    ];
    rr:predicateObjectMap [
        rr:predicate ps:P161;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{ArticleEntity}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate  pq:P453;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Role}"
        ]
    ].
<#Role> a rr:TriplesMap;
...
<#Actor> a rr:TriplesMap;
...
```

Listing 4.26: CARML, SDM-RDFizer and Morph-KGC mapping for the extraction of n-ary relations example with `#BnodesMapping`, `#Role` and `#Actor` unchanged from Listing 4.25.

```
wd:Q9087190
  wdt:P674 wd:Q229202  ;
  p:P161 [
    ps:P161 wd:Q453921  ;
    pq:P453 wd:Q229202
  ] .

wd:Q229202  wdt:P1441 wd:Q9087190  .
wd:Q453921  wdt:P106 wd:Q33999  .
wd:Q2031748 wdt:P674 wd:Q6563552  ;
            p:P161 [
                ps:P161 wd:Q453921  ;
                pq:P453 wd:Q6563552
            ] .

wd:Q6563552 wdt:P1441 wd:Q2031748  .
wd:Q274744  wdt:P674 wd:Q385299  ;
            p:P161 [
                ps:P161 wd:Q453921  ;
                pq:P453 wd:Q385299
            ] .
wd:Q385299  wdt:P1441 wd:Q274744  .

#Missing triples in CARML, SDM-RDFizer and Morph-KGC output
wd:Q13427026  p:P161 [ ps:P161 wd:Q453921  ] .
wd:Q7742499  p:P161 [ ps:P161 wd:Q453921  ] .
```

Listing 4.27: Output for the extraction of n-ary relations example

Table 4.10: n-ary example output comparison

	№ of Triples	Execution time [s]
Tarql	20	2.70
SPARQL-Generate	20	3.22
SPARQL Anything	20	2.64
CARML	16	1.93
RMLMapper	20	5.66
SDM-RDFizer	16	3.14
Chimera	20	5.19
Morph-KGC	16	17.71

4.6 Relations across rows example

Figure 4.8a displays a horizontal table from the article "List of Olympic Medalists in Cross-Country Skiing" (Q1067245) with their corresponding Wikidata entities shown in Figure

4.8b. This table presents consecutive games and their respective medalists. In cases like this, we aim to extract relations between the subjects of consecutive rows. For this particular instance, we would like to extract relations between a game and its subsequent game in the next row, following the graph pattern displayed in Figure 4.9. For example, we anticipate the triple 1992 Albertville[Q1042417] followed by[P156] 1994 Lillehammer[Q602473].

Games	Gold	Silver	Bronze
1992 Albertville *details*	Vegard Ulvang Norway	Marco Albarello Italy	Christer Majbäck Sweden
1994 Lillehammer *details*	Bjørn Dæhlie Norway	Vladimir Smirnov Kazakhstan	Marco Albarello Italy
1998 Nagano *details*	Bjørn Dæhlie **(2)** Norway	Markus Gandler Austria	Mika Myllylä Finland

(a) Original table[14]

ArticleEntity	Games	Gold	Silver	Bronze
Q1067245	Q1042417	"Q370499, Q1092273"	"Q958134, Q1091872"	"Q1078525, Q1091882"
Q1067245	Q602473	"Q217505, Q1190505"	"Q560383, Q603465"	"Q958134, Q1077174"
Q1067245	Q1042412	"Q217505, Q970927"	"Q698381, Q781796"	"Q449544, Q1001549"

(b) 47539_25.csv

Figure 4.8: Wikipedia table with relations across rows and the corresponding Wikidata entities

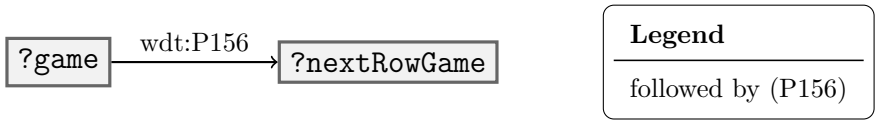

Figure 4.9: Graph pattern of expected triple for Figure 4.8

The requirements necessary for extracting triples between different rows are as follows:

- **Join operation:** We aim to conduct a Join operation within the table, aligning it with itself to extract the subject from one row and the object of the triple from the next row.
- **Row identifier:** We require the identification number for each row to determine the order between them.
- **Arithmetic operations:** These are necessary to filter and ensure that the number of the next row corresponds to the number of the current row plus one.

[14]from the Wikipedia article `https://en.wikipedia.org/wiki/List_of_Olympic_medalists_in_cross-country_skiing`

Table 4.11 presents the tested processors and their capabilities in conforming to the listed requirements, excluding the use of custom functions (as in RML + FnO). As we can see, Tarql is the only SPARQL-based mapping language that provides a built-in variable called `?ROWNUM` for accessing the number of the row within the input CSV file. However, Tarql doesn't allow join operations.

Regarding RML, the only processor that fulfills all the requirements is Morph-KGC, through the use of RML Views, which allows the execution of SQL queries against tabular input data, as seen in Listing 4.28. Using SQL inside the mapping, we can perform Join operations and utilize the SQL function `row_number`. The output of the mapping is presented in Listing 4.31 while the number of triples extracted and the execution time per processor is presented in Listing 4.13.

Table 4.11: Requirements for extracting triples across rows

Processor	Join	RowID	Arithmetic op
Tarql	✗	✓	✓
SPARQL-Generate	✓	✗	✓
SPARQL Anything	✓	✗	✓
CARML	✓	✗	✗
RMLMapper	✓	✗	✗
SDM-RDFizer	✓	✗	✗
Chimera	✓	✗	✗
Morph-KGC	✓	✓	✓

```
<RARMapping>
    rml:logicalSource [
        rml:query """
            SELECT a.game AS current, b.nextGame AS follower
            FROM (
                SELECT column0 AS game, row_number() over () AS "ID"
                FROM '../Input/RelationsAcrossRows.csv'
                ORDER BY "ID"
                OFFSET 1 ROW
            ) a
            JOIN (
            SELECT c.number AS "nextGame", row_number() over () AS ID2
            FROM (
                SELECT column0 AS number, row_number() over () AS ID
                FROM '../Input/RelationsAcrossRows.csv'
                ORDER BY ID
                OFFSET 1 ROW
            ) c
            ORDER BY ID2
            OFFSET 1 ROW
            ) b
            ON (a.ID = b.ID2)
        """ ];
    rr:subjectMap [
        rr:template "http://example.com/{current}"
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P156;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{follower}"
        ]
    ].
```

Listing 4.28: Morph-KGC mapping for the extraction of relations across rows example using RML Views.

An alternative for extracting relations between entities in the same column and in consecutive rows is by preprocessing the input data. This involves concatenating the following row to the first row and then applying a mapping for extracting the relation on the same row, as shown in Section 4.3. The result of this preprocessing is presented in Table 4.12 for Figure 4.8b. We present the columns from the first row with an appended "1" (as in "ArticleEntity1" and "Games1") and the following row with column names with an appended "2" (as in "ArticleEntity2" and "Games2"). The remaining columns follow the same rule.

The Tarql mapping is presented in Listing 4.29, the mappings for SPARQL-Generate and SPARQL Anything are analogous. The RML mapping is presented in Listing 4.30. The comparison of the number of triples extracted and the time of execution per processor is presented in Table 4.14. The triples extracted are the same as those extracted with Morph-

Table 4.12: Wikidata entities preprocessed

ArticleEntity1	Games1	...	ArticleEntity2	Games2	...
Q1067245	Q1042417	...	Q1067245	Q602473	...
Q1067245	Q602473	...	Q1067245	Q1042412	...
Q1067245	Q1042412	...			...

KGC using RML Views (Listing 4.31).

```
CONSTRUCT {
    ?games wdt:P156 ?games_next.
}
FROM <file:../Input/RelationsAcrossRows-preprocessed.csv>
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Games1)) AS ?games)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Games2)) AS ?games_next)
}
```

Listing 4.29: Tarql mapping for the extraction of relations across rows example with preprocessing.

```
<#RARmapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "../Input/RelationsAcrossRows-preprocessed.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{Games1}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P156;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Games2}"
        ]
    ].
```

Listing 4.30: RML mapping for the extraction of relations across rows example with preprocessing.

```
wd:Q1042417 wdt:P156 wd:Q602473 .
wd:Q602473 wdt:P156 wd:Q1042412 .
```

Listing 4.31: Output for the extraction of relations across rows example

Table 4.13: Relations across rows example output comparison

	№ of Triples	Execution time [s]
Tarql	-	-
SPARQL-Generate	-	-
SPARQL Anything	-	-
CARML	-	-
RMLMapper	-	-
SDM-RDFizer	-	-
Chimera	-	-
Morph-KGC	2	8.28

Table 4.14: Relations across rows example with preprocessing output comparison

	№ of Triples	Execution time [s]
Tarql	2	1.13
SPARQL-Generate	2	1.32
SPARQL Anything	2	1.02
CARML	2	0.07
RMLMapper	2	1.93
SDM-RDFizer	2	2.98
Chimera	2	5.48
Morph-KGC	2	8.55

4.7 Matrix table mapping example

Figure 4.10 displays a matrix table from Cluster 12, the largest cluster by number of tables that includes matrix tables. This cluster contains information about climate in different locations.

Wikidata manages climate information using the property weather history[P4150], which stores the data in tabular form. Although the conversion to RDF in this instance does not align with Wikidata's standard procedure, which links to a table online rather than modelling the data of the table as a graph, we will use table in Figure 4.10 as an illustrative example for handling matrix tables.

Each cell in Figure 4.10a can be interpreted by reading both the column header and the row header. Selecting the column header is a standard procedure in all mapping language processors, as we have seen in previous examples, where each column can be processed as

[15]from the Wikipedia article `https://en.wikipedia.org/wiki/Mountain_City,_Nevada`

Climate data for Mountain City, Nevada (1955-1999) [hide]													
Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Year
Record high °F (°C)	58 (14)	70 (21)	71 (22)	79 (26)	89 (32)	95 (35)	105 (41)	99 (37)	95 (35)	85 (29)	75 (24)	64 (18)	105 (41)
Mean daily maximum °F (°C)	38.0 (3.3)	42.3 (5.7)	47.1 (8.4)	56.2 (13.4)	65.0 (18.3)	74.6 (23.7)	84.7 (29.3)	84.4 (29.1)	75.1 (23.9)	63.5 (17.5)	47.1 (8.4)	39.3 (4.1)	59.8 (15.4)
Mean daily minimum °F (°C)	9.4 (−12.6)	13.2 (−10.4)	18.5 (−7.5)	24.3 (−4.3)	30.9 (−0.6)	36.4 (2.4)	39.7 (4.3)	37.7 (3.2)	29.8 (−1.2)	22.1 (−5.5)	17.1 (−8.3)	9.9 (−12.3)	24.1 (−4.4)
Record low °F (°C)	−48 (−44)	−35 (−37)	−23 (−31)	2 (−17)	7 (−14)	14 (−10)	20 (−7)	16 (−9)	8 (−13)	−10 (−23)	−22 (−30)	−46 (−43)	−48 (−44)
Average precipitation inches (mm)	1.30 (33)	1.03 (26)	1.12 (28)	1.06 (27)	1.66 (42)	1.21 (31)	0.50 (13)	0.55 (14)	0.75 (19)	0.92 (23)	1.38 (35)	1.35 (34)	12.83 (325)
Average snowfall inches (cm)	8.4 (21)	4.0 (10)	4.4 (11)	1.4 (3.6)	0.6 (1.5)	0 (0)	0 (0)	0 (0)	0 (0)	0.9 (2.3)	1.9 (4.8)	6.7 (17)	28.3 (71.2)
Source: [11]													

(a) Original table[15]

ArticleEntity	Month
Q6418841	
Q6418841	
Q6418841	
Q6418841	
Q6418841	Q25257
Q6418841	

(b) 975024_1.csv

Figure 4.10: Matrix Wikipedia table and the corresponding Wikidata entities

a variable, and the mapping processing is applied row by row. On the other hand, reading the horizontal header is not as direct. For instance, if we want to apply a mapping rule to the cell in the "Year" column and the Average precipitation inches (cm)[Q25257] row, we would need to apply a conditional function first in the `WHERE` clause of the SPARQL-based mapping languages:

```
BIND(IF(?Month = "Q25257",?Year,?undef) AS ?precipitationYear)
VALUES ?undef {UNDEF}
```

In Morph-KGC, using RML Views, we can add the following SQL query to the logical source to achieve the same result:

```
SELECT Year AS 'precipitationYear'
FROM '975024_1.csv'
WHERE Month = 'Q25257';
```

In Figure 4.10, `precipitationYear` would be associated with the value "12.83(325)" indicating the average precipitation in a year. We can then define mapping rules using this variable.

This process can be replicated for each cell from which we aim to extract information.

Listings 4.32 and 4.33 present mappings using Tarql and Morph-KGC, respectively, using the variable `precipitationYear`. The mappings in SPARQL Anything and SPARQL-Generate are analogous to the one in Listing 4.32. These mappings follow the graph pattern displayed in Figure 4.11. Since a Wikidata property for average precipitation doesn't exist, we define the property average precipitation[x] based on the existing Wikidata property precipitation height[P3036], which is used for maximum precipitation height. The start and end time qualifiers are extracted from the title of the table using SPARQL or SQL functions, depending on the processor, for string manipulation.

Table 4.15 shows the results of applying the mappings in the processors that support conditional variables. The relations extracted for Figure 4.10 are presented in Listing 4.34.

Figure 4.11: Graph pattern of expected triples for Figure 4.10

```
CONSTRUCT {
    ?location p:avgPrecipitation ?precipitation.
    ?precipitation ps:avgPrecipitation ?precipitationYear;
                   pq:P580 ?start;
                   pq:P582 ?end  .
}
FROM <file:../Input/975024_1.csv>
WHERE {
    BIND ( IF (BOUND(?ArticleEntity) && strlen(?ArticleEntity)>0, URI(CONCAT('http://www.
    wikidata.org/entity/', ?ArticleEntity)),?undef) AS ?location)
    # The following BINDs are for extracting the period of time of the precipitation
    height
    BIND ( IF (BOUND(?Title) && strlen(?Title)>0, REPLACE(REPLACE(?Title,"[^0-9-]",""),
    REPLACE(REPLACE(?Title,"[^0-9-]",""),"^([0-9]{4}-[0-9]{4}?)",""),"")  ,?undef) AS ?
    period )
    BIND ( STRBEFORE(?period,"-") as ?start )
    BIND ( STRAFTER(?period,"-") as ?end  )
    # The variable 'precipitationYear' is only defined if we are on the precipitation row
    BIND ( IF (BOUND(?Month) && strlen(?Month)>0 &&  ?Month = "Q25257", ?Year  ,?undef)
    AS ?precipitationYear )
    BIND ( IF (BOUND(?precipitationYear), BNODE(),?undef) AS ?precipitation )
    VALUES ?undef {UNDEF}
}
```

Listing 4.32: Tarql mapping for the extraction of relations in matrix table example

```
<#MatrixMapping> a rr:TriplesMap;
rml:logicalSource [
    rml:query """
    SELECT SPLIT_PART(Period, '-' ,1) AS 'start', SPLIT_PART(Period, '-' ,2) AS 'end',
    ArticleEntity AS 'location', precipitationYear
    FROM (
        SELECT ArticleEntity, REGEXP_REPLACE(p, REGEXP_REPLACE(p, '^([0-9]{4}-[0-9]{4}?)'
    , '', 'ig'), '', 'ig') AS 'Period',precipitationYear
            FROM (
                SELECT Year AS 'precipitationYear', ArticleEntity,  REGEXP_REPLACE(Title,
     '[^0-9-]', '', 'ig') AS 'p'
                FROM '../Input/975024_1.csv'
                WHERE Month = 'Q25257'
    )
    )
    """
];
rr:subjectMap [
    rr:template "http://www.wikidata.org/entity/{location}";
];
rr:predicateObjectMap [
    rr:predicate p:avgPrecipitation;
    rr:objectMap [
        rr:termType rr:BlankNode ;
        rr:template "_:source{location}{start}{end}{precipitationYear}"
    ]
```

```
].
<#Precipitation> a rr:TriplesMap;
rml:logicalSource [
    rml:query """
    SELECT SPLIT_PART(Period, '-' ,1) AS 'start', SPLIT_PART(Period, '-' ,2) AS 'end',
    ArticleEntity AS 'location', precipitationYear
    FROM (
        SELECT ArticleEntity, REGEXP_REPLACE(p, REGEXP_REPLACE(p, '^([0-9]{4}-[0-9]{4}?)'
    , '', 'ig'), '', 'ig') AS 'Period',precipitationYear
            FROM (
                SELECT Year AS 'precipitationYear', ArticleEntity,  REGEXP_REPLACE(Title,
     '[^0-9-]', '', 'ig') AS 'p'
                FROM '../Input/975024_1.csv'
                WHERE Month = 'Q25257'
    )
    )
    """
];
rr:subjectMap [
    rr:termType rr:BlankNode;
    rr:template "_:source{location}{start}{end}{precipitationYear}"
];
rr:predicateObjectMap [
    rr:predicate ps:avgPrecipitation;
    rr:objectMap [
        rml:reference "precipitationYear"
    ]
];
rr:predicateObjectMap [
    rr:predicate pq:P580;
    rr:objectMap [
        rml:reference "start"
    ]
];
rr:predicateObjectMap [
    rr:predicate pq:P582;
    rr:objectMap [
        rml:reference "end"
    ]
].
```

Listing 4.33: Morph-KGC mapping for the extraction of relations in matrix table example

```
wd:Q6418841  p:avgPrecipitation  [ pq:P580             "1955" ;
                                   pq:P582             "1999" ;
                                   ps:avgPrecipitation  "[' 12.83', ' (325.9)']"
                                 ] .
```

Listing 4.34: Output for the extraction of relations in matrix table example

Table 4.15: Relations across rows example with preprocessing output comparison

	№ of Triples	Execution time [s]
Tarql	4	1.17
SPARQL-Generate	4	1.38
SPARQL Anything	4	1.12
CARML	-	-
RMLMapper	-	-
SDM-RDFizer	-	-
Chimera	-	-
Morph-KGC	4	13.59

4.8 Querying Wikidata for existing triples

Significant amounts of valuable information in the form of triples can be extracted from Wikipedia tables. These triples can be used to enrich the information contained in Wikidata. For these purposes, it is useful to discern which triples already exist in Wikidata and which do not. As we will explore, this process becomes more complicated in the presence of n-ary relations. The process is applied after the mappings produce triples and is independent of a particular mapping language or processor.

We executed the following queries over the Wikidata Query Service to determine the number of extracted triples already present in Wikidata.

We applied different queries depending on whether the triples are connected to blank nodes or not. In the latter case, for triples with truthy predicates representing binary relations (usually prefixed by `wdt: http://www.wikidata.org/prop/direct/`), we used the query in Listing 4.35. This returns the triples that already exist in Wikidata, where the `VALUES` clause allows for checking a batch of triples in one request.

```
SELECT ?s ?p ?o
WHERE {
    ?s ?p ?o .
    VALUES (?s ?p ?o) {
        #Here the triples extracted
    }
}
```

Listing 4.35: Wikidata query simple predicates

In the case involving blank nodes, the process becomes more intricate. Applying the same mapping as illustrated in Listing 4.35 is not feasible due to the presence of n-ary relations featuring multiple optional properties, each linked to different objects. To effectively

partition the triples into values within the query, it becomes essential to discern which subjects are associated with distinct relations through blank nodes. To address this, we categorized the triples into three groups, for which we applied different custom queries:

- Subjects connected to statements with property statements and property qualifiers (Listing 4.36).
- Subjects connected to statements with property statements but no property qualifiers (Listing 4.37).
- Subjects connected to statements with property qualifiers but no property statements (Listing 4.38).

```
#Query when there are values connected to the blank node through property statement and
    property qualifier.
SELECT *
WHERE {
    ?s  p:...  [
        ps:...  ?o1 ;
        pq:...  ?o2
    ] .
    VALUES (?s ?o1 ?o2) {
        #Here the triples extracted
    }
}
```

Listing 4.36: Wikidata query predicates in statements

```
#Query when there are values connected to the blank node through property statement but
    no property qualifier.
SELECT ?s ?o1
WHERE {
    ?s  p:...  [
        ps:...  ?o1
    ] .
    VALUES (?s  ?o1) {
        #Here the triples extracted
    }
}
```

Listing 4.37: Wikidata query predicates in statements

```
#Query when there are values connected to the blank node through property qualifier but
    no property statement.
SELECT ?s ?o2
WHERE {
    ?s  p:...  [
        pq:...  ?o2
    ] .
    VALUES (?s  ?o2) {
        #Here the triples extracted
    }
}
```

Listing 4.38: Wikidata query predicates in statements

4.9 Discussion of expressiveness

In this chapter, we reviewed various features of different mapping language processors, applying mappings to RDF on samples of tables extracted from the largest clusters of the data corpus. Our observations revealed similar results in extracting simple triples across RML processors and SPARQL-based processors. However, when it comes to processing data within the mappings, SPARQL-based mapping languages and the use of SQL in RML Views exhibit several advantages.

In SPARQL-based mapping languages and RML Views, the ability to filter data within the mapping by applying conditional requirements to variables and defining optional variables is a notable strength, especially when handling matrix tables as input data. Given that Tarql, SPARQL-Generate, and SPARQL Anything are built upon Apache Jena, utilizing functions provided by the framework becomes feasible, for example, to split a CSV string in a table cell and bind each element to its own row. Morph-KGC also offers built-in functions, some of which were useful when dealing with multivalued tables. On the other hand, while RMLMapper offers some built-in functions, those required for our data were not functional.

Additionally, Morph-KGC includes other valuable built-in functions for managing strings, although they were not utilized in the provided examples. These functions includes string array slicing, converting a string to uppercase or lowercase, and casting a string to a date type.

For tasks such as extracting the n^{th} entity within a cell in a multivalued table, SPARQL functions were highly useful in SPARQL-based mapping languages, similar to the use of SQL functions with RML Views in Morph-KGC. We found the process of extracting the n^{th} entity within a cell more straightforward using RML Views, but the resulting mapping was longer than those in SPARQL-based mapping languages.

In our analysis of extracting relations between consecutive rows, we found that the only capable processor of fulfilling this task without requiring data preprocessing was Morph-KGC using RML Views. This is because it allows for join operations as well as row identification and arithmetic operations. While we obtained the expected results with SPARQL-based mapping languages, they required preprocessing of the data specifically for the case of extracting relations between consecutive rows of the table.

Additionally, for handling noise within the cells, we efficiently applied the `REPLACE` clause in SPARQL with `REGEX` conditions. While Morph-KGC offers a built-in `string_replace` fuction, this function only receives strings and not `REGEX` conditions as input.

Concerning n-ary relations, we found that the creation of blank nodes was more complex in RML using the join operation. Issues also arose with the RML processors CARML, SDM-RDFizer and Morph-KGC due to their requirement of establishing a template for blank nodes. While we acknowledge that these tools adhere to the RML specification, which stipulates that a term map (constant, column/reference or a template) must be provided while generating blank nodes, this constraint doesn't align with our use case. A more detailed discussion on allowing the identification of blank nodes without a template can be found in this RML-core issue[16]. In contrast, the creation of blank nodes proved to be straightforward in SPARQL-based mapping languages.

[16] `https://github.com/kg-construct/rml-core/issues/52`

Chapter 5

Experimental Results and Discussion

In this chapter, we will extract triples from the top ten largest clusters according to table count, as well as from additional selected clusters using the different techniques discussed in Chapter 4. The same processors from Chapter 4 will be employed. We will compare the number of extracted triples, execution time, and the pre-existing triples and n-ary relations in Wikidata for each cluster. Our goal is to answer questions such as how many triples we can extract, which mapping systems are more efficient in terms of runtimes for large clusters, and whether or not the expressiveness of the mapping languages is sufficient for this use-case.

5.1 Top 10 largest clusters

This section analyzes RDF triples extraction from the top 10 largest clusters by number of tables.

5.1.1 Cluster 1

Cluster description This cluster groups tables containing data related to electoral processes. Table 5.1 displays a sample from Cluster 1, while an HTML example from the same cluster is illustrated in Figure 3.3a.

As shown in Table 3.4, this cluster encompasses a total of 81,277 tables with 1,202,635 rows and 6 columns. In this experiment, our focus will be on entities within the "2_Party" and "Candidate" columns, along with the "ArticleEntity" appended as a protagonist column.

Table 5.1: Sample from Cluster 1

ArticleEntity	2_Party	Candidate
Results of the 1947 New South Wales state election [Q25246103]	National Party of Australia [Q946040]	David Drummond [Q5233082]
Results of the 1925 New South Wales state election [Q28428421]	Progressive Party [Q7248776]	David Drummond [Q5233082]
Sheffield Heeley [Q1031720]	Labour Party [Q9630]	Meg Munn [Q578990]
Sheffield Heeley [Q1031720]	Labour Co-operative [Q6467393]	Meg Munn [Q578990]
Sunderland [Q7639684]	Liberal Party [Q622441]	Ian Hannah [Q13529925]

Mappings applied We have applied the same mappings as detailed in Section 4.3 to extract triples following the pattern illustrated in Figure 4.2 over tables from Cluster 1. The mappings applied are presented in Section A.1.

As explained in Section 4.3, the application of mappings results in some incorrect triples, wherein the same entity appears in both the subject and object positions. This issue can be addressed using the `FILTER (?candidate != ?party)` clause, which is applicable across all SPARQL-based mapping languages. Since the cluster contains 18,843 rows with cells containing multiple entities (see Table 3.6), we could also apply the features discussed in Section 4.4. For comparison purposes, since these features only work in SPARQL-based mapping languages, we will not include them in this experiment.

Results The results are presented in Table 5.2. Triples with the same entity in the subject and object position of the triple are considered incorrect. To compare our extracted triples with those already existing on Wikidata, we executed the query in Figure 4.35 over the Wikidata Query Service to determine the number of extracted triples already present in Wikidata. Then, we present the amount of new triples extracted.

The triples extracted from the sample in Table 5.1 are shown in Listing 5.1.

```
wd:Q5233082 wdt:P102 wd:Q946040.
wd:Q5233082 wdt:P102 wd:Q7248776.
wd:Q578990 wdt:P102 wd:Q9630.
wd:Q578990 wdt:P102 wd:Q6467393.
wd:Q13529925 wdt:P102 wd:Q622441.
```

Listing 5.1: Sample from Cluster 1 output

Summary We extracted 23,026 triples from a total of 1,202,635 rows. This corresponds to the unique number of entity pairs between the "2_Party" and "Candidate" columns. The

Table 5.2: Cluster 1 output comparison

	№ of Triples	Execution time [s]	Existing triples on Wikidata	Incorrect triples	New triples
Tarql	23,026	9.67	14,137	289	8,600
SPARQL-Generate	23,026	17.83	14,137	289	8,600
SPARQL Anything	23,026	19.90	14,137	289	8,600
CARML	23,026	10.08	14,137	289	8,600
RMLMapper	23,026	53.11	14,137	289	8,600
SDM-RDFizer	23,026	32.28	14,137	289	8,600
Chimera	23,026	44.69	14,137	289	8,600
Morph-KGC	23,026	10.48	14,137	289	8,600

number of triples extracted is low compared to the number of rows since in most of the rows, either one of the entities was missing (no link was given in the table, or no corresponding Wikipedia article is available), or the pair was present in another row of the table. Among these 23,026 triples, 14,137 are already present in Wikidata, accounting for 61.4%. After excluding 289 triples identified as incorrect, we found a total of 8,600 new triples.

5.1.2 Cluster 2

Cluster description This cluster groups tables containing data about electoral processes, similar to Cluster 1. Table 5.3 displays a sample from Cluster 2, while an HTML example from the same cluster is illustrated in Figure 3.3b. As we can see from the HTML example, the only difference compared to Cluster 1 is the absence of the "±" column.

As shown in Table 3.4, this cluster encompasses a total of 57,878 tables with 650,861 rows and 5 columns. In this experiment, our focus will be on entities within the "2_Party" and "Candidate" columns, along with the "ArticleEntity" appended as a protagonist column.

Mappings applied We have applied the same mappings as detailed in Section 4.3 to extract triples, following the pattern illustrated in Figure 4.2, over tables from Cluster 2. The mappings applied, analogous to those in Cluster 1, are presented in Section A.2.

Results The results are presented in Table 5.4. Triples with the same entity in the subject and object position of the triple are considered incorrect. To compare our extracted triples with those already existing on Wikidata, we executed the query in Figure 4.35 over the Wikidata Query Service to determine the number of extracted triples already present in Wikidata. Then, we present the amount of new triples extracted.

Table 5.3: Sample from Cluster 2

ArticleEntity	2_Party	Candidate
Constituency election results in the United Kingdom general election, 1929 [Q25044294]	Labour Party [Q9630]	Charles Duncan [Q5076996]
Constituency election results in the United Kingdom general election, 1929 [Q25044294]	Conservative Party [Q9626]	Abraham Montagu Lyons [Q4669093]
Constituency election results in the United Kingdom general election, 1929 [Q25044294]	Conservative Party [Q9626]	William Brass, 1st Baron Chattisham [Q8005868]
Constituency election results in the United Kingdom general election, 1929 [Q25044294]	Labour Party [Q9630]	Philip Snowden, 1st Viscount Snowden [Q337483]
Constituency election results in the United Kingdom general election, 1929 [Q25044294]	Labour Party [Q9630]	Philip Noel-Baker, Baron Noel-Baker [Q211856]

The triples extracted from the sample in Table 5.3 are shown in Listing 5.2.

```
wd:Q5076996 wdt:P102 wd:Q9630.
wd:Q4669093 wdt:P102 wd:Q9626.
wd:Q8005868 wdt:P102 wd:Q9626.
wd:Q337483 wdt:P102 wd:Q9630.
wd:Q211856 wdt:P102 wd:Q9630.
```

Listing 5.2: Sample from Cluster 2 output

Table 5.4: Cluster 2 output comparison

	№ of Triples	Execution time [s]	Existing triples on Wikidata	Incorrect triples	New triples
Tarql	15,721	6.54	10,551	295	4,875
SPARQL-Generate	15,721	18.17	10,551	295	4,875
SPARQL Anything	15,721	11.87	10,551	295	4,875
CARML	15,721	6.89	10,551	295	4,875
RMLMapper	15,721	29.51	10,551	295	4,875
SDM-RDFizer	15,721	17.73	10,551	295	4,875
Chimera	15,721	20.74	10,551	295	4,875
Morph-KGC	15,721	9.17	10,551	295	4,875

Summary We extracted 15,721 triples from a total of 650,861 rows, corresponding to the number of distinct rows based on pairs of values in the "2_Party" and "Candidate" columns. Among these 15,721 triples, 10,551 are already present in Wikidata, accounting for 67.1% of the extracted triples. After excluding 295 triples identified as incorrect, we found a total of 4,875 new triples.

5.1.3 Cluster 3

Cluster description This cluster groups tables with information about ratings given by different sources to music albums. Table 5.5 displays a sample from Cluster 3, while an HTML example from the same cluster is illustrated in Figure 3.3c.

Table 5.5: Sample from Cluster 3

ArticleEntity	Review_score_Source_3	Review_score__rate_3
Strays[Q593929]	Entertainment Weekly[Q275033]	B+
Strays[Q593929]	Kludge[Q6421478]	(8/10)
Strays[Q593929]	NME[Q192632]	(8/10)

As shown in Table 3.4, this cluster encompasses a total of 57,427 tables with 187,707 rows and 2 columns. Out of the total number of rows, 106,544 rows present multi-column entity relationships suitable for relation extraction.

Mappings applied From this cluster, we aim to extract triples that conform to the graph pattern depicted in Figure 5.1. The mappings applied are presented in Section A.3, and they follow a similar structure to those introduced in Section 4.5, utilizing blank nodes. In these mappings, we will use entities present in the "ArticleEntity" column, for which we will assume albums of the songs, and entities present in the "source" column. As for the "rating", we will consider the string value present in this column as a literal value for the triple.

Figure 5.1: Graph pattern of expected triples for Cluster 3

In the SPARQL-based mapping languages, we incorporate the following clause:

```
FILTER(BOUND(?source)|| BOUND(?Review_score__rate_3))
```

This ensures that if both variables, `?source` and `?Review_score__rate_3`, are unbound, the blank node will not be created. However, it's worth noting that in RMLMapper, the blank node in the object position is generated regardless.

As discussed in Section 4.5, RMLMapper and Chimera automatically generate a blank node identifier, whereas CARML, SDM-RDFizer and Morph-KGC necessitate the specification of a template to assign unique identifiers to each node. Our identifier is formed by concatenating the variables ArticleEntity, ArticleTitle, Review_score__rate_3, and Review_score__Source_3, ensuring a distinct key for each distinct n-ary relation in the input data. However, due to the incompleteness of the input data, some of these variables may be missing. In such cases, the blank node is not generated. Thus, if one or both variables, `?source` and `?Review_score__rate_3`, are unbound, the blank node is not created in the CARML, SDM-RDFizer and Morph-KGC processors .

As for the handling of literal values, SPARQL-Generate automatically filters empty strings of the form `""`, omitting those triples, and automatically trims the string literal values. While Tarql also automatically filters empty strings of the form `""`, it requires an additional clause in the `WHERE` clause using the SPARQL `REPLACE` function to trim string values. On the other hand, in SPARQL Anything, we need to add the following functions in the `SERVICE` clause to filter strings of the form `""` and to trim strings. In that way, empty strings containing spaces, for instance `" "`, will also be filtered:

```
fx:properties fx:trim-strings  true ; fx:null-string "".
```

In the case of RML, all tested processors filter empty strings. CARML and SDM-RDFizer automatically trim strings, while Morph-KGC, RMLMapper, and Chimera preserve spaces at the beginning and end of the strings. In the case of Morph-KGC, we can apply the `grel:string_trim`[1] default function to trim the rating. Another interesting configuration of Morph-KGC is that, in the configuration file, different values can be interpreted as NULL and then be filtered in the results. The mapping with the added trim function is presented in Listing A.10, and the configuration file is shown in Listing A.11.

Results The results are presented in Table 5.6. As we use templates for the generation of blank nodes in SDM-RDFizer and CARML (Listing A.8), the number of triples extracted is lower than that extracted with the other processors. As for RMLMapper and Chimera, both processes reach the timeout limit (the timeout was set to 1 hour). The SPARQL-based

[1] `https://github.com/morph-kgc/morph-kgc/blob/main/src/morph_kgc/fnml/built_in_functions.py`

languages have similar results in extracted triples, with Tarql being the fastest in processing time.

To see if our extracted relations already exist on Wikidata, we categorized the output triples into three cases:

- № of Bnodes P444 P447: `?album` connected to blank nodes connected to `?rate` and `?source`
- № of Bnodes P444: `?album` connected to blank nodes connected to `?rate` but no `?source`.
- № of Bnodes P447: `?album` connected to blank nodes connected to `?source` but no `?rate`.

We executed the query in Listing 4.36 over the Wikidata Query Service to determine the number of extracted triples already present in Wikidata. In the first case, involving blank nodes with `?rate` and `?source`, where `?rate` is a string, we checked for triples with the same `?album`, `?source`, and any `?rate`, just in case the string in `?rate` was different. Even with this relaxed rule, the number of existing relations was less than 0.05% (Figure 5.2a). The results for each category are presented in Figure 5.2.

The output triples of SDM-RDFizer and Morph-KGC couldn't be split because the blank nodes generated with the template create blank nodes with illegal characters that cannot be processed by RDFLib, the tool we are using to categorize the triples. This isn't an issue in CARML because these characters are omitted. This problem can be addressed by preprocessing the data or by adding a custom function during data processing to avoid these characters.

Table 5.6: Cluster 3 output comparison

	№ of Triples	Execution time [s]
Tarql	244,285	6.83
SPARQL-Generate	244,285	10.50
SPARQL Anything	241,412	14.65
CARML	57,104	14.67
RMLMapper	Timeout	Timeout
SDM-RDFizer	57,108	21.68
Chimera	Timeout	Timeout
Morph-KGC	77,806	14.88

	№ of Bnodes P444 P447	Existing relations	% of existing relations [%]
Tarql	19,036	8	0.04
SPARQL-Generate	19,036	8	0.04
SPARQL Anything	19,035	8	0.04
CARML	19,031	8	0.04
RMLMapper	-	-	-
SDM-RDFizer	-	-	-
Chimera	-	-	-
Morph-KGC	-	-	-

(a) Relations with subject `?album` connected to blank nodes connected to `?rate` and `?source`

	№ of Bnodes P444	Existing relations	% of existing relations [%]
Tarql	4,611	3	0.07
SPARQL-Generate	4,611	3	0.07
SPARQL Anything	4,611	3	0.07
CARML	0	0	0
RMLMapper	-	-	-
SDM-RDFizer	-	-	-
Chimera	-	-	-
Morph-KGC	-	-	-

(b) Relations with subject `?album` connected to blank nodes connected to `?rate` but no `?source`

	№ of Bnodes P447	Existing relations	% of existing relations [%]
Tarql	87,017	815	0.94
SPARQL-Generate	87,017	815	0.94
SPARQL Anything	87,017	815	0.94
CARML	0	0	0
RMLMapper	-	-	-
SDM-RDFizer	-	-	-
Chimera	-	-	-
Morph-KGC	-	-	-

(c) Relations with subject `?album` connected to blank nodes connected to `?source` but no `?rate`.

Figure 5.2: Cluster 3 output comparison - existing triples and n-ary relations in Wikidata

The triples extracted from the sample in Table 5.5 are shown in Listing 5.3.

```
wd:Q593929  p:P444  [ pq:P447  wd:Q192632 ; ps:P444  "(8/10)];
            p:P444  [ pq:P447  wd:Q275033 ; ps:P444  "B+  "];
            p:P444  [ pq:P447  wd:Q6421478 ; ps:P444  "(8/10) "].
```

Listing 5.3: Sample from Cluster 3 output

Summary As shown in Figure 5.2a, we extracted 19,036 blank nodes connected to both `?rate` and `?source`, of which less than 0.05% were present in Wikidata. These relations contain rich information that could potentially be integrated into Wikidata.

Regarding the relations in Figure 5.2b, we extracted 4,611 blank nodes connected to `?rate` but not `?source`, of which 0.07% were present in Wikidata. This could potentially serve as a source of knowledge, as similar relations already exist in Wikidata. However, it may fail to meet the citation needed constraint[2], as statements for review score[3] should have at least one reference.

In Figure 5.2c, we see that we extract 87,017 blank nodes connected to `?source` but not `?rate`, of which 0.94% were present in Wikidata. Information about the ratings associated with those sources is not contained within the data corpus. This could be addressed by extracting some ratings from the metadata of the Wikipedia tables in an updated extraction of the tables.

5.1.4 Cluster 4

Cluster description This cluster groups tables with information about songs, including their length and number. We have assumed that the Article entity corresponds to the album of the records.

A sample from the cluster is provided in Table 5.7, focusing on the columns that will be used in the mappings. An HTML example from the same cluster is illustrated in Figure 3.3d.

As shown in Table 3.4, this cluster contains a total of 56,332 tables with 450,119 rows and 3 columns. Notably, 44,906 rows feature entities in more than one column.

Mappings applied In this cluster, our objective is to extract triples adhering to the graph pattern illustrated in Figure 5.3. Conditional relations, represented by dashed lines,

[2] `https://www.wikidata.org/wiki/Q54554025`
[3] `https://www.wikidata.org/wiki/Property:P444`

Table 5.7: Sample from Cluster 4

ArticleEntity	TitleID	Length	No
Mixed Up[Q1931761]	Lullaby[Q3047725]	7:43	['1.']
City to City[Q1094222]	Right Down the Line[Q28444210]	4:28, 5	['3.']
B-Sides & Rarities[Q2527719]	"Savory" [Q7428417](Jawbox[Q1549376] cover) (featuring Jonah Matranga[Q1644952], Shaun Lopez[Q7490901] and Chris Robyn of Far[Q455060])	4:37	['1.']

are included in the extraction only when specific variables are bound.

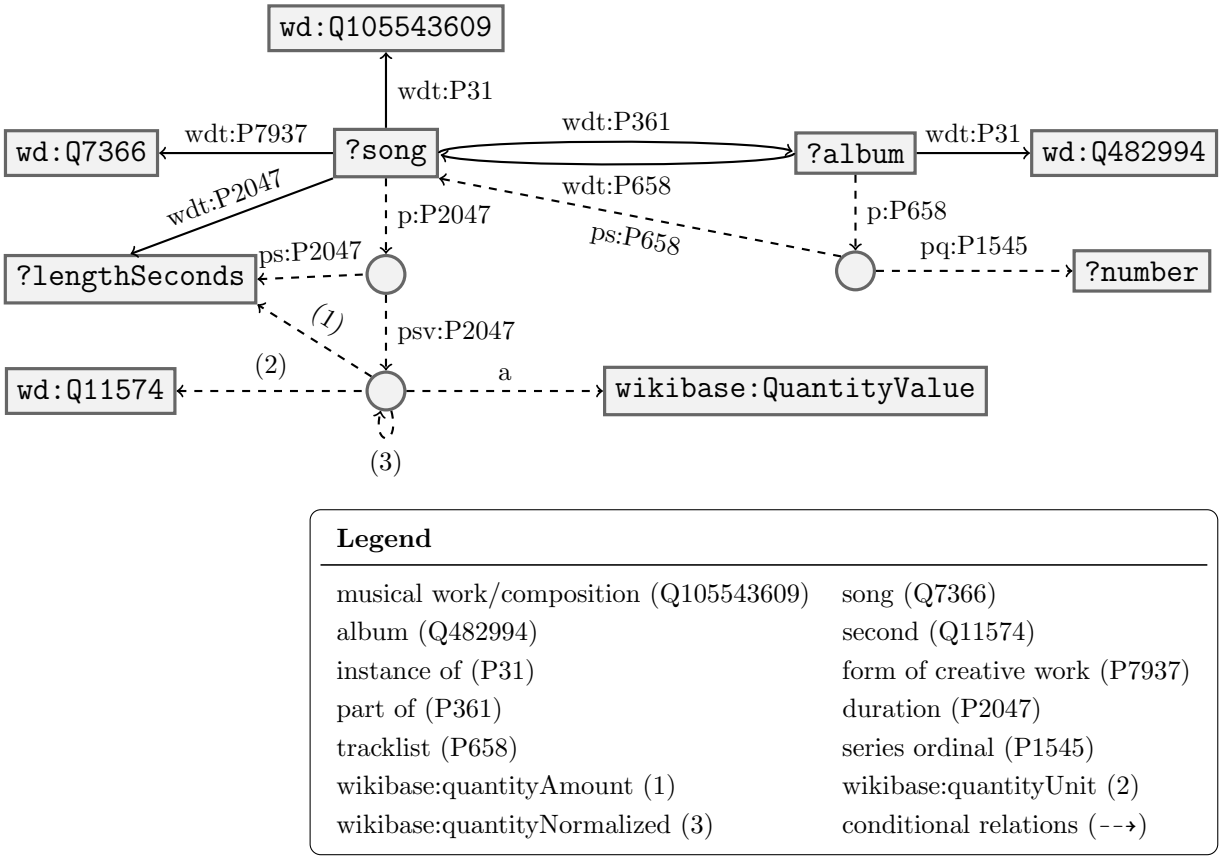

Figure 5.3: Graph pattern of expected triples for Cluster 4

Mappings in Listings A.12, A.13, and A.14 employ SPARQL to filter and process the input data within their `WHERE` clauses. To address the complexity arising from the "TitleID" column containing multiple entities within a cell (as discussed in Section 4.4), we opt to extract the first entity as `?song`, as subsequent entities typically denote other artists featured in the song.

For the `?length` variable, we apply the `REPLACE SPARQL` function with `regex` conditions to eliminate noise within the cell. Then we applied functions to convert the length to seconds, and typecast the variable as `xsd:decimal`. Finally, conditional bounding is applied to generate blank nodes in the graph only when specific variables are bound.

It's worth mentioning that due to a limitation in SPARQL, where the `IF` function does not accept `UNDEF` as a parameter [4], we define the `?undef` variable as `UNDEF` to overcome this constraint.

Regarding RML, Listing A.17 presents a Morph-KGC mapping using RML Views to generate the entire expected graph. As seen in Section 4.4, we applied the SQL function `split_part` to extract the first entity in the "TitleID" column. Analogous to what we have done in SPARQL-based mapping languages, we apply the SQL function `regexp_replace` using `regex` conditions to eliminate noise within the columns "no" and "length". In this case, we could not apply the `regex` condition `(?<=[0-9]+:[0-9]{2}).*` to clean the length variable as Morph-KGC uses DuckDB[5] to handle the SQL query, which uses the regular expression library RE2[6], which does not support `regex` lookbehind. To handle this, we apply a nested `regexp_replace` function to obtain the same results.

As for the other RML processors tested, a partial RML mapping is presented in Listings A.15 and A.16. These mappings do not cover the entire expected graph, as we are unable to perform operations on the length variable without preprocessing the data. Regarding cells with multiple entities, these mappings generate incorrect triples for such cells since they concatenate all the cell content to the `wd:` prefix, given the inability to select the first entity as in the SPARQL-based mappings.

Results The results of executing these mappings are detailed in Table 5.8. It is noteworthy that the CARML generated triples includes inaccuracies, corresponding to rows in the input data with multiple entities in their cells. These discrepancies can be addressed by applying a custom function to preprocess the data, eliminating these rows before applying the mapping, or post-processing the extracted triples.

We omit the results of RMLMapper, as it couldn't read the input data due to noise, and the results of SDM-RDFizer as it generates ill-formed blank nodes. Regarding Chimera, the processing reached the timeout set in one hour real time.

Figure 5.4 categorizes the extracted triples into two groups:

- Simple predicates (Figure 5.4a): These correspond to triples that don't contain blank nodes and are connected through direct Wikidata properties (prefixed with `wdt:`).

[4] `https://github.com/w3c/sparql-dev/issues/62`
[5] `https://duckdb.org/`
[6] `https://github.com/google/re2`

- № of Bnodes P658 P1545 (Figure 5.4b): `?album` connected to blank nodes connected to `?song` and `?number`.

We don't categorize the remaining triples, as they only complement the information of the already categorized triples. For example, if `?album` is connected to a blank node linked to `?song` and no `?number`, it reflects the same information as the simple predicate triple: `?album wdt:P658 ?song`, already covered in the Simple Predicates category.

Finally, we executed the queries in Listing 4.35 and 4.36 over the Wikidata Query Service to determine the number of extracted triples and n-ary relations already present in Wikidata. The results are presented in Figure 5.4.

Table 5.8: Cluster 4 output comparison

	№ of Triples	Execution time [s]
Tarql	666,048	31.65
SPARQL-Generate	665,891	35.64
SPARQL Anything	666,048	65.05
CARML	308,889	33.54
RMLMapper	-	-
SDM-RDFizer	-	-
Chimera	Timeout	Timeout
Morph-KGC	646,955	289.47

The triples extracted from the sample in Table 5.7 are shown in Listing 5.4.

	Simple predicates	Existing triples	% of existing triples [%]
Tarql	212,000	41,994	19.81
SPARQL-Generate	211,983	42,095	19.86
SPARQL Anything	212,000	41,994	19.81
CARML	-	-	-
RMLMapper	-	-	-
SDM-RDFizer	-	-	-
Chimera	-	-	-
Morph-KGC	211,998	42,096	19.86

(a) Triples that don't contain blank nodes and are connected through direct Wikidata properties

	№ of Bnodes P658 P1545	Existing relations	% of existing relations [%]
Tarql	43,992	431	0.98
SPARQL-Generate	43,992	431	0.98
SPARQL Anything	43,992	431	0.98
CARML	-	-	-
RMLMapper	-	-	-
SDM-RDFizer	-	-	-
Chimera	-	-	-
Morph-KGC	43,992	431	0.98

(b) Relations with subject `?album` connected to blank nodes connected to `?song` and `?number`

Figure 5.4: Cluster 4 output comparison - existing triples and n-ary relations in Wikidata

```
wd:Q3047725 wdt:P31 wd:Q105543609;
            wdt:P361 wd:Q1931761;
            wdt:P7937 wd:Q7366;
            wdt:P2047 "463"^^xsd:decimal;
            p:P2047  [ ps:P2047   "463"^^xsd:decimal; psv:P2047  _:b1 ].
 _:b1 a wikibase:QuantityValue;
            wikibase:quantityAmount "463"^^xsd:decimal;
            wikibase:quantityUnit wd:Q11574;
            wikibase:quantityNormalized  _:b1.

wd:Q1931761 wdt:P31 wd:Q482994;
            p:P658 [ps:P658 wd:Q3047725; pq:P1545 "1"].


wd:Q28444210 wdt:P31 wd:Q105543609;
            wdt:P361 wd:Q1094222;
            wdt:P7937 wd:Q7366;
            wdt:P2047 "268"^^xsd:decimal;
            p:P2047  [ ps:P2047   "268"^^xsd:decimal; psv:P2047  _:b2 ].

 _:b2 a wikibase:QuantityValue;
            wikibase:quantityAmount "268"^^xsd:decimal;
            wikibase:quantityUnit wd:Q11574;
            wikibase:quantityNormalized  _:b2.

wd:Q1094222 wdt:P31 wd:Q482994;
            p:P658 [ps:P658 wd:Q28444210; pq:P1545 "3"].

wd:Q7428417 wdt:P31 wd:Q105543609;
            wdt:P361 wd:Q2527719;
            wdt:P7937 wd:Q7366;
            wdt:P2047 "277"^^xsd:decimal;
            p:P2047  [ ps:P2047   "277"^^xsd:decimal; psv:P2047  _:b1 ].
 _:b1 a wikibase:QuantityValue;
            wikibase:quantityAmount "277"^^xsd:decimal;
            wikibase:quantityUnit wd:Q11574;
            wikibase:quantityNormalized  _:b1.

wd:Q2527719 wdt:P31 wd:Q482994;
            p:P658 [ps:P658 wd:Q7428417; pq:P1545 "1"].
```

Listing 5.4: Sample from Cluster 4 output

Summary As seen in Figure 5.4b, we extract 43,992 n-ary relations corresponding to rows with entities in the columns ArticleEntity (album of the song), Title (title of the song), and a string value in No. (number of the song within the album), of which 0.98% were already present in Wikidata.

As for simple predicates in Figure 5.4a, we extract over 212,000 triples from 450,119 rows,

of which 19.81% were already present in Wikidata.

5.1.5 Cluster 5

Cluster description This cluster presents information about players and their positions in different sports. A sample of the cluster is provided in Table 5.9, while an HTML example from the same cluster can be found in Figure 3.3e. As observed in Table 5.9, Wikipedia articles within this cluster may cover different sports topics; in the sample, we have a list of transfers, a soccer championship, and a football club. The "spancol" column within the cluster similarly varies; in the sample, it refers to countries and islands.

Table 5.9: Sample from Cluster 5

ArticleEntity	spancol	Position	Player
list of Chinese football transfers winter 2018 [Q48839324]	People's Republic of China [Q148]	midfielder [Q193592]	Nizamdin Ependi [Q50444719]
2008 CIS Men's Soccer Championship [Q4610671]	Canada [Q16]	goalkeeper [Q201330]	Srdjan Djekanović [Q562366]
Tan Holdings FC [Q16953034]	Northern Mariana Islands [Q16644]	forward [Q280658]	Joe Wang Miller [Q6212890]

As shown in Table 3.4, this cluster contains a total of 51,229 tables with 496,619 rows and 4 columns. Notably, 495,033 rows feature entities in more than one column. As shown in Table 3.6, there are 202,589 rows presenting multiple entities within a cell, specifically in the "Player" column. However, we notice that the entities in this cluster don't follow the order of appearance in the row, so we cannot establish which entity corresponds to the player. For that reason, those rows were omitted, resulting in 294,030 rows.

Mappings applied In this cluster, our objective is to extract triples adhering to the graph pattern illustrated in Figure 5.5.

Mappings are presented in Listing A.18, A.19, A.20 and A.21. These mappings follow the structure of mappings presented in Section 4.3.

Given that entities in "ArticleEntity" and "spancol" cover diverse topics, and the properties to be used are unclear, we choose to exclude these attributes from the mappings.

Results Table 5.10 displays the number of triples extracted per processor, their execution time, as well as the triples already present in Wikidata. The performance is consistent across all processors, with Tarql being the fastest.

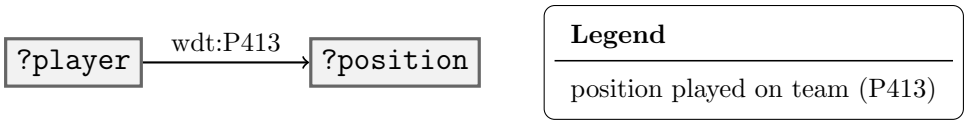

Figure 5.5: Graph pattern of expected triple for Cluster 5

Table 5.10: Cluster 5 output comparison

	№ of Triples	Execution time [s]	Existing triples on Wikidata	New triples
Tarql	73,985	10.81	51,526	22,459
SPARQL-Generate	73,985	18.02	51,526	22,459
SPARQL Anything	73,985	24.79	51,526	22,459
CARML	73,985	16.88	51,526	22,459
RMLMapper	73,985	12.13	51,526	22,459
SDM-RDFizer	73,985	13.62	51,526	22,459
Chimera	73,985	19.29	51,526	22,459
Morph-KGC	73,985	9.39	51,526	22,459

The output triples from the sample in Table 5.9 are presented in Listing 5.5.

```
wd:Q50444719 wdt:P413 wd:Q193592.
wd:Q562366 wdt:P413 wd:Q201330.
wd:Q6212890 wdt:P413 wd:Q280658.
```

Listing 5.5: Sample from Cluster 5 output

Summary We extracted 73,985 triples from a total of 294,030 rows, corresponding to the number of distinct rows based on pairs of values in the "Position" and "Player" columns. Among these 73,985 triples, 51,526 are already present in Wikidata, accounting for 69.6%. We found a total of 22,459 new triples.

5.1.6 Cluster 6

Cluster description This cluster presents information about sports teams and the sporting events in which they participated. A sample of the cluster is provided in Table 5.11, while an HTML example from the same cluster can be found in Figure 3.3f.

As depicted in Table 3.4, this cluster includes a total of 25,449 tables with 50,898 rows and 6 columns. Notably, only 340 rows present multi-column entity relationships suitable for relation extraction. This is due to the fact that while the "ArticleEntity" column contains

entities in 50,126 rows, entities in the "spancol" column are present in only 340 rows (refer to Table 3.6).

Table 5.11: Sample from Cluster 6

ArticleEntity	spancol
2015 IFAF World Championship[Q4630300]	Australia national American football team[Q3873624]
2015 IFAF World Championship[Q4630300]	Brazil national American football team[Q585174]

Mappings applied From this cluster, we aim to extract triples that conform to the graph pattern depicted in Figure 5.6. The mappings applied are presented in Section A.6. These mappings follow the structure presented in Section 4.3.

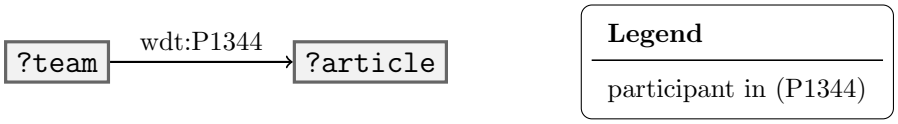

Figure 5.6: Graph pattern of expected triple for Cluster 6

Results Table 5.12 displays the number of triples extracted per processor and their execution time. After searching for the extracted triples in the Wikidata Query Service, we found that all of them are new triples. Regarding execution time, SDM-RDFizer and SPARQL Anything are the fastest.

The output triples from the sample in Table 5.11 are presented in Listing 5.6.

Table 5.12: Cluster 6 output comparison

	№ of Triples	Execution time [s]	Existing triples on Wikidata	New triples
Tarql	175	6.31	0	175
SPARQL-Generate	175	6.70	0	175
SPARQL Anything	175	4.84	0	175
CARML	175	5.34	0	175
RMLMapper	175	7.90	0	175
SDM-RDFizer	175	4.44	0	175
Chimera	175	6.34	0	175
Morph-KGC	175	7.99	0	175

```
wd:Q3873624 wdt:P1344 wd:Q4630300.
wd:Q585174 wdt:P1344 wd:Q463030.
```

Listing 5.6: Sample from Cluster 6 output

Summary We extracted 175 triples from a total of 50,898 rows, corresponding to the number of distinct rows based on pairs of values in the "ArticleEntity" and "spancol" columns. None of these triples are already present in Wikidata.

5.1.7 Cluster 7

Cluster description This cluster provides information about actors and their film credits, including details such as the film title, their role in each production, and additional notes.

A sample from Cluster 7 is presented in Table 5.13, while an HTML example from the same cluster can be found in Figure 3.3g.

Table 5.13: Sample from Cluster 7

ArticleEntity	Title	Role
Derek Jacobi[Q256164]	Othello[Q2634673]	Michael Cassio[Q6829164]
Derek Jacobi[Q256164]	Hamlet[Q898721]	King Claudius[Q1811294]

As depicted in Table 3.4, this cluster includes a total of 23,472 tables with 459,839 rows and 4 columns. Notably, 73,910 rows present multi-column entity relationships suitable for relation extraction.

Mappings applied We have applied the same mappings as detailed in Section 4.5 to extract triples following the pattern illustrated in Figure 4.7 over tables from Cluster 7. The mappings applied, are presented in Section A.7.

Results The results are presented in Table 5.14. As explained in Section 4.5, while generating the blank nodes in CARML, SDM-RDFizer and Morph-KGC, some variables of the templates are missing, resulting in fewer extracted triples. As for RMLMapper, there are no results because the processing reached the timeout set in one hour. As for Chimera, the process took 3338.94 seconds in real time, not reaching the timeout of one hour. However, as explained in Section 4.2, the Execution time was calculated considering `user+sys` time, which, when executed in multi-core, can be larger than the `real` time.

We categorized the extracted triples into four categories:

- Simple predicates: These correspond to triples that don't contain blank nodes and are connected through direct Wikidata properties (prefixed with `wdt:`).
- № of Bnodes P161 P453: `?title` connected to blank nodes connected to `?actor` and `?role`.
- № of Bnodes P161: `?title` connected to blank nodes connected to `?actor` but no `?role`.
- № of Bnodes P453: `?title` connected to blank nodes connected to `?role` but no `?actor`.

Then we apply queries for each category to check for existing triples and n-ary relations in Wikidata. The results for each category are presented in Figure 5.7.

Table 5.14: Cluster 7 output comparison

	№ of Triples	Execution time [s]
Tarql	526,121	16.19
SPARQL-Generate	526,121	34.90
SPARQL Anything	526,005	46.05
CARML	24,546	21.88
RMLMapper	Timeout	Timeout
SDM-RDFizer	24,546	56.48
Chimera	526,121	13271.96
Morph-KGC	24,546	24.56

The output triples from the sample in Table 5.13 are shown in Listing 5.7.

```
wd:Q2634673 p:P161  [ pq:P453  wd:Q6829164 ; ps:P161  wd:Q256164];
            wdt:P674  wd:Q6829164 .
wd:Q6829164  wdt:P1441  wd:Q2634673 .
wd:Q256164  wdt:P106  wd:Q33999 .

wd:Q898721  p:P161  [ pq:P453  wd:Q1811294 ; ps:P161  wd:Q256164] ;
            wdt:P674  wd:Q1811294 .
wd:Q1811294  wdt:P1441  wd:Q898721 .
```

Listing 5.7: Sample from Cluster 7 output

	Simple predicates	Existing triples	% of existing triples [%]
Tarql	18,837	12,834	68.13
SPARQL-Generate	18,837	12,834	68.13
SPARQL Anything	18,831	12,833	68.15
CARML	18,837	12,834	68.13
RMLMapper	-	-	-
SDM-RDFizer	18,837	12,834	68.13
Chimera	18,837	12,834	68.13
Morph-KGC	18,837	12,834	68.13

(a) Triples that don't contain blank nodes and are connected through direct Wikidata properties

	№ of Bnodes P161 P453	Existing relations	% of existing relations [%]
Tarql	1,903	454	23.86
SPARQL-Generate	1,903	454	23.86
SPARQL Anything	1,903	454	23.86
CARML	1,903	454	23.86
RMLMapper	-	-	-
SDM-RDFizer	1,903	454	23.86
Chimera	1,903	454	23.86
Morph-KGC	1,903	454	23.86

(b) Relations with subject `?title` connected to blank nodes connected to `?actor` and `?role`

	№ of Bnodes P161	Existing relations	% of existing relations [%]
Tarql	47,658	15,070	31.62
SPARQL-Generate	47,658	15,070	31.62
SPARQL Anything	47,658	15,070	31.62
CARML	-	-	-
RMLMapper	-	-	-
SDM-RDFizer	-	-	-
Chimera	47,658	15,070	31.62
Morph-KGC	-	-	-

(c) Relations with subject `?title` connected to blank nodes connected to `?actor` but no `?role`

	№ of Bnodes P453	Existing relations	% of existing relations [%]
Tarql	3	1	33.33
SPARQL-Generate	3	1	33.33
SPARQL Anything	-	-	-
CARML	-	-	-
RMLMapper	-	-	-
SDM-RDFizer	-	-	-
Chimera	3	1	33.33
Morph-KGC	-	-	-

(d) Relations with subject `?title` connected to blank nodes connected to `?role` but no `?actor`

Figure 5.7: Cluster 7 output comparison - existing triples and n-ary relations in Wikidata

Summary As depicted in Figure 5.7a, we extracted over 18,837 simple predicates, of which 68.13% were already present in Wikidata.

In Figure 5.7b, we extracted 1,903 n-ary relations connecting `?title` to `?actor` and `?role`, with 23.86% already present in Wikidata.

Concerning the relations in Figure 5.7c, we extracted 47,658 n-ary relations connecting `?title` to `?actor` but not `?role`, with 31.62% already present in Wikidata.

Lastly, in Figure 5.7d, we extracted 3 n-ary relations connecting `?title` to `?role` but not `?actor`, of which 1 relation is novel.

5.1.8 Cluster 8

Cluster description This cluster groups tables with information about shipwrecks. Table 5.15 displays a sample from Cluster 8, while an HTML example from the same cluster can be found in Figure 3.3h.

As depicted in Table 3.4, this cluster includes a total of 20,772 tables with 65,300 rows and 3 columns. Notably, 65,226 rows exhibit multi-column entity relationships suitable for relation extraction. This prevalence is attributed to a substantial number of rows containing entities in columns "ArticleEntity" and "Country", as shown in Table 3.6. In contrast, the "Ship" column has entities in only 2,106 rows.

Table 5.15: Sample from Cluster 8

ArticleEntity	Ship	Country	Description
list of shipwrecks[Q6639048] in 1958	SS Üsküdar[Q7394589]	Turkey[Q43]	Lodos[Q17050959]
list of shipwrecks[Q6639048] in 1958	MS Skaubryn[Q6718204]	Norway[Q20]	Indian Ocean[Q1239]
list of shipwrecks[Q6639048] in 1958	SS San Flaviano[Q7394327]	United Kingdom[Q145]	

Mappings applied From this cluster, we aim to extract triples that conform to the graph pattern depicted in Figure 5.8. The mappings applied are presented in Section A.8, and they follow a similar structure to those introduced in Section 4.3.

As observed, our focus centers on the "Ship" and "Country" columns. Despite the "ArticleEntity" column containing a substantial number of rows with entities, we couldn't

establish a clear relation with the other columns. This is because the articles typically represent lists of shipwrecks.

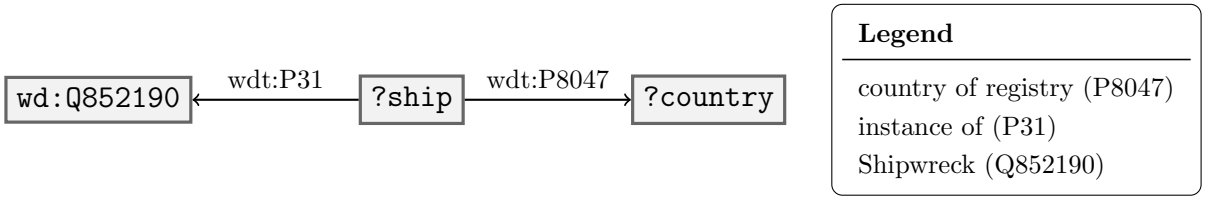

Figure 5.8: Graph pattern of expected triple for Cluster 8

Results Table 5.16 shows the number of triples extracted per processor and their execution time. As seen, CARML and SDM-RDFizer are the fastest. In the same table, we present the already existing triples in Wikidata.

Table 5.16: Cluster 8 output comparison

	№ of Triples	Execution time [s]	Existing triples on Wikidata	New triples
Tarql	789	4.84	94	695
SPARQL-Generate	789	5.53	94	695
SPARQL Anything	787	4.22	94	693
CARML	789	3.92	94	695
RMLMapper	789	7.74	94	695
SDM-RDFizer	789	3.91	94	695
Chimera	789	5.8	94	695
Morph-KGC	789	9.40	94	695

The output triples from the sample in Table 5.15 are shown in Listing 5.8.

```
wd:Q7394589 wdt:P8047 wd:Q43;
            wdt:P31 wd:Q852190.
wd:Q6718204 wdt:P8047 wd:Q20;
            wdt:P31 wd:Q852190.
wd:Q7394327 wdt:P8047 wd:Q145;
            wdt:P31 wd:Q852190.
```

Listing 5.8: Sample from Cluster 8 output

Summary As shown in Figure 5.16, we extracted over 789 triples, with 11.9% of them already present in Wikidata. As mentioned earlier, the number of extracted triples was restricted to the quantity of entities found in the "Ship" column.

5.1.9 Cluster 9

Cluster description This cluster contains similar information to Section 5.1.4, presenting details about songs and their albums. Table 5.17 showcases a sample from Cluster 9, while an HTML example from the same cluster can be found in Figure 3.3i.

Table 5.17: Sample from Cluster 9

No	Length	ArticleEntity	Title	Writer
14.	5:54	True Colors[Q19816324]	"Clarity[Q6614153], fox[Q8331], Tiësto[Q183508]"	"Skylar Grey[Q233750], Porter Robinson[Q21084]"

As shown in Table 3.4, this cluster contains a total of 17,204 tables with 138,504 rows and 4 columns. Notably, 36,345 rows feature entities in more than one column.

Mappings applied Figure 5.9 illustrates the expected graph pattern, which mirrors Section 5.1.4's graph pattern (Figure 5.3) but includes two additional triples related to the songwriters.

Listings A.35, A.36, A.37 and A.39 present mappings that are equivalent to those introduced in Section 5.1.4, now incorporating the new triples. As the Writer column contains rows with multiple entities per cell, we apply the functions outlined in Section 4.4. This ensures that all entities within a Writer cell are bound to the variable `?writer`. In the case of Morph-KGC, we also had to remove the spaces within the cell using the SQL `regexp_replace` function to generate well-formed URLs for each writer.

Results Table 5.18 displays the number of extracted triples per processor. As per Section 5.1.4, it is not feasible in CARML, RMLMapper, SDM-RDFizer, and Chimera to process the data and extract relations for the n^{th} entity within a cell without defining custom functions.

Similar to Section 5.1.4, we categorize the extracted triples into two groups. The number of triples per category and the existing triples and n-ary relations in Wikidata are presented in Figure 5.10.

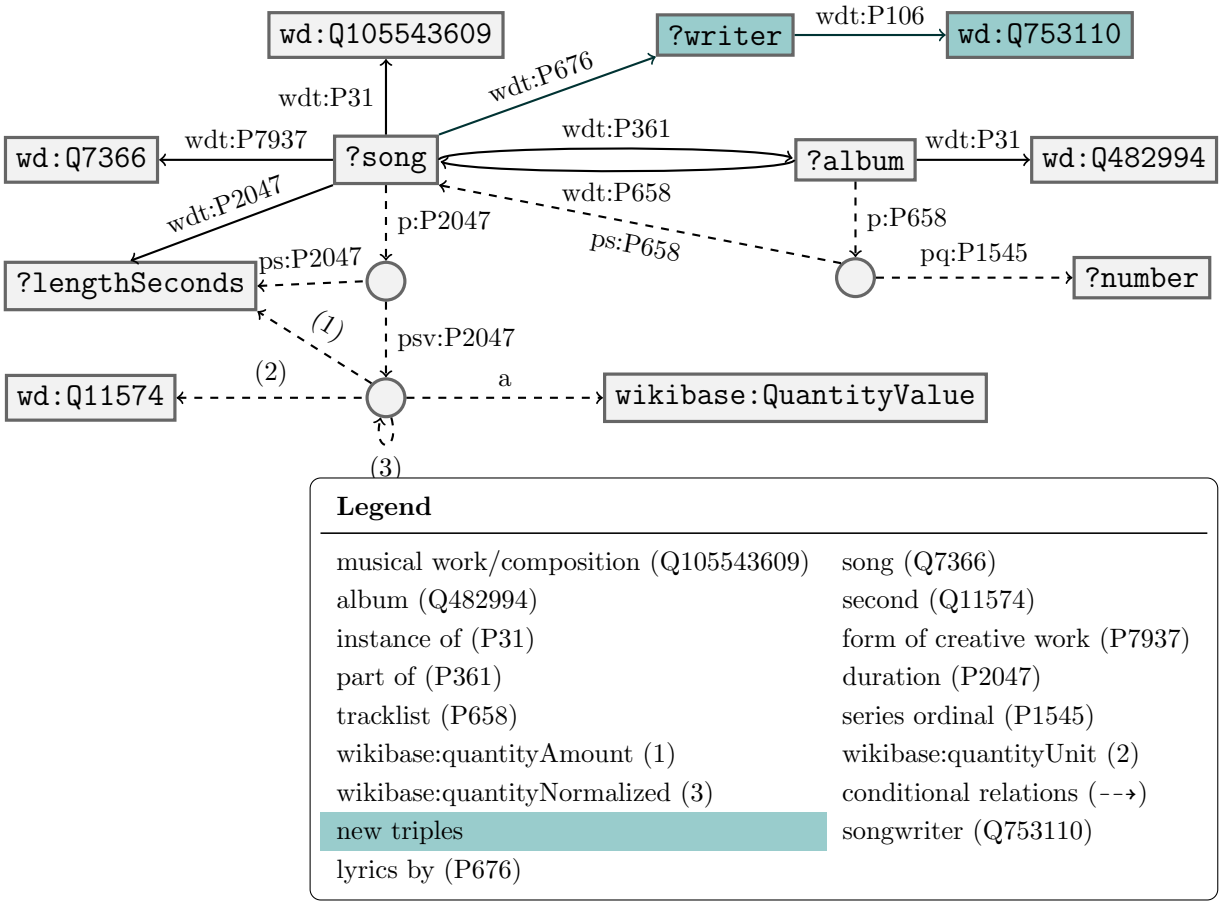

Figure 5.9: Graph pattern of expected triples for Cluster 9

Table 5.18: Cluster 9 output comparison

	№ of Triples	Execution time [s]
Tarql	305,977	15.38
SPARQL-Generate	305,908	18.86
SPARQL Anything	305,977	25.46
CARML	-	-
RMLMapper	-	-
SDM-RDFizer	-	-
Chimera	-	-
Morph-KGC	296,111	121.29

The triples extracted from the sample in Table 5.17 are shown in Listing 5.9.

```
wd:Q6614153 p:P2047  [ ps:P2047   354.0 ; psv:P2047  _:b10122] ;
            p:P2047    [ ps:P2047   354.0 ; psv:P2047  _:b8772] ;
            wdt:P2047  354.0 ;
            wdt:P31    wd:Q105543609 ;
            wdt:P361   wd:Q19816324 ;
            wdt:P676   wd:Q21084 , wd:Q233750 ;
            wdt:P7937  wd:Q7366 .

_:b10122    a   wikibase:QuantityValue ;
            wikibase:quantityAmount      354.0 ;
            wikibase:quantityNormalized  _:b10122 ;
            wikibase:quantityUnit        wd:Q11574 .

wd:Q19816324    p:P658    [ pq:P1545  "14" ; ps:P658   wd:Q6614153] ;
                wdt:P31   wd:Q482994 ;
                wdt:P658  wd:Q6614153 .

wd:Q21084  wdt:P106  wd:Q753110 .
wd:Q233750 wdt:P106   wd:Q753110 .
```

Listing 5.9: Sample from Cluster 9 output

Summary As seen in Figure 5.10b, we extract 18,883 n-ary relations connecting `?album` to `?song` and `?number`, with 1.4% already present in Wikidata.

As for simple predicates in Figure 5.10a, we extract over 109,447 triples from 38,504 rows, of which 23.9% were already present in Wikidata.

	Simple predicates	Existing triples	% of existing triples [%]
Tarql	109,483	26,122	23.86
SPARQL-Generate	109,447	26,203	23.94
SPARQL Anything	109,483	26,122	23.86
CARML	-	-	-
RMLMapper	-	-	-
SDM-RDFizer	-	-	-
Chimera	-	-	-
Morph-KGC	109,479	26,203	23.93

(a) Triples that don't contain blank nodes and are connected through direct Wikidata properties

	№ of Bnodes P658 P1545	Existing relations	% of existing relations [%]
Tarql	18,883	266	1.41
SPARQL-Generate	18,883	266	1.41
SPARQL Anything	18,883	266	1.41
CARML	-	-	-
RMLMapper	-	-	-
SDM-RDFizer	-	-	-
Chimera	-	-	-
Morph-KGC	18,883	266	1.41

(b) Relations with subject `?album` connected to blank nodes connected to `?song` and `?number`

Figure 5.10: Cluster 9 output comparison - existing triples and n-ary relations in Wikidata

5.1.10 Cluster 10

Cluster description This cluster provides information about players and their respective positions. A sample of the cluster is detailed in Table 5.19, while an HTML example from the same cluster can be found in Figure 3.3j.

Table 5.19: Sample from Cluster 10

ArticleEntity	Position	Player
Chacarita Juniors[Q853053]	goalkeeper[Q201330]	Cristian Daniel Campestrini[Q3697368]
Chacarita Juniors[Q853053]	goalkeeper[Q201330]	Facundo Ferrero[Q30111798]
Chacarita Juniors[Q853053]	goalkeeper[Q201330]	Lucas Álvarez[Q47093375]

As shown in Table 3.4, this cluster contains a total of 15,596 tables with 128,267 rows

and 2 columns. Notably, 120,208 rows feature entities in more than one column.

Mappings applied The expected graph is identical to that of Section 5.1.5, as illustrated in Figure 5.5.

Mappings are presented in Section A.10. These mappings, analogous to those in Section 5.1.5, follow the structure of mappings presented in Section 4.3.

Results A sample of the cluster is detailed in Table 5.19, with the output triples from this sample presented in Listing 5.10.

Table 5.20 displays the number of triples extracted per processor, along with their execution times, and indicates the triples already present in Wikidata. Notably, performance remains consistent across all processors, with CARML being the fastest.

Table 5.20: Cluster 10 output comparison

	№ of Triples	Execution time [s]	Existing triples on Wikidata	New triples
Tarql	224	8.24	142	82
SPARQL-Generate	224	8.57	142	82
SPARQL Anything	224	8.04	142	82
CARML	224	6.02	142	82
RMLMapper	224	9.87	142	82
SDM-RDFizer	224	6.44	142	82
Chimera	224	7.83	142	82
Morph-KGC	224	8.57	142	82

The triples extracted from the sample in Table 5.19 are shown in Listing 5.10.

```
wd:Q3697368 wdt:P413 wd:Q201330;
            wdt:P54 wd:Q853053.
wd:Q30111798 wdt:P413 wd:Q201330;
            wdt:P54 wd:Q853053.
wd:Q47093375 wdt:P413 wd:Q201330;
            wdt:P54 wd:Q853053.
```

Listing 5.10: Sample from Cluster 10 output

Summary We extracted 224 triples from a total of 128,267 rows. This corresponds to the unique number of entity pairs between the “Position” and “Player” columns. The number of triples extracted is low compared to the number of rows since in most of the rows, either one

of the entities was missing (no link was given in the table, or no corresponding Wikipedia article is available), or the pair was present in another row of the table. Among these 224 triples, 142 are already present in Wikidata, accounting for 63.4%. We found a total of 82 new triples.

5.2 Other clusters

This section analyzes RDF triple extraction from selected clusters of the data corpus, focusing on applying features of the mapping languages that weren't used in the top 10 largest clusters.

5.2.1 Cluster 22

Cluster description This cluster groups tables with information about sport events along with their respective gold, silver, and bronze medalists. An HTML sample from Cluster 22 is provided in Figure 5.11 with their respective entities listed in Table 5.21.

This cluster contains a total of 7,492 tables with 53,294 rows in total and 4 columns.

Event	Gold	Silver	Bronze
10 metre air pistol *details*	Lynda Kiejko Canada	Alejandra Zavala Mexico	Lilian Castro El Salvador
25 metre pistol *details*	Lynda Kiejko Canada	Sandra Uptagrafft United States	Mariana Nava Mexico
10 metre air rifle *details*	Goretti Zumaya Mexico	Fernanda Russo Argentina	Eglis de la Cruz Cuba
50 metre rifle three positions *details*	Eglis Yaima Cruz Cuba	Amelia Fournel Argentina	Yarimar Mercado Martinez Puerto Rico
Trap *details*	Amanda Chudoba Canada	Kayle Browning United States	Kimberley Bowers United States
Skeet *details*	Kim Rhode United States	Melisa Gil Argentina	Francisca Crovetto Chile

(a) Table 622922.48

Event	Gold	Silver	Bronze
Men's singles	István Jónyer	Antun Stipančić	Mitsuru Kohno Norio Takashima
Women's singles	Pak Yung-Sun	Zhang Li	Tatiana Ferdman Ge Xin'ai
Men's doubles	Gábor Gergely István Jónyer	Antun Stipančić Dragutin Šurbek	Jean-Denis Constant Jacques Secrétin Katsuyuki Abe Shigeo Itoh
Women's doubles	Maria Alexandru Shoko Takahashi	Zhu Xiangyun Lin Meiqun	Elmira Antonyan Tatiana Ferdman Yukie Ozeki Sachiko Yokota
Mixed doubles	Stanislav Gomozkov Tatiana Ferdman	Sarkis Sarchayan Elmira Antonyan	Liang Geliang Zhang Li Shigeo Itoh Yukie Ozeki

(b) Table 872092.2

Figure 5.11: HTML table examples from Cluster 22
5.11a from the Wikipedia article `https://en.wikipedia.org/wiki/List_of_2015_Pan_American_Games_medalists`
5.11b from the Wikipedia article `https://en.wikipedia.org/wiki/1975_World_Table_Tennis_Championships`

Discussion In this cluster, our objective is to extract triples adhering to the graph pattern illustrated in Figure 5.12. As shown in Figure 5.11, the columns "Gold", "Silver", and "Bronze" often contain multiple values. From these columns, we aim to extract relations between entities within the same cell, as explained in Subsection 4.4.3. In Subsection 4.4.3, we applied mappings over the table in Figure 4.4 extracted from Cluster 22 to extract the relation `?athlete wdt:P27 ?country`. In that example, we took the first entity within the

Table 5.21: Sample from Cluster 22

ArticleEntity	Event	Gold	Silver	Bronze	TableID
List of 2015 Pan American Games medal winners	10 metre air pistol details[Q20648058]	Lynda Kiejko [Q24262328] Canada[Q17513922]	Mexico[Q17511729] Alejandra Zavala [Q4714454]	El Salvador[Q17508394] Lilian Castro [Q20111550]	622922.48
List of 2015 Pan American Games medal winners	10 metre air rifle details[Q20648067]	Mexico[Q17511729] Goretti Zumaya [Q26225895]	Fernanda Russo [Q22280191] Argentina[Q17512684]	Eglys de la Cruz [Q290906] Cuba[Q17512137]	622922.48
List of 2015 Pan American Games medal winners	Skeet details[Q20648094]	United States[Q17513790] Kim Rhode [Q233759]	Argentina[Q17512684] Melisa Gil [Q10328405]	Francisca Crovetto [Q4414558] Chile[Q17509104]	622922.48
1975 World Table Tennis Championships [Q339390]	Men's singles [Q48860954]	István Jónyer[Q579551] _ [Q28]	_ [Q83286] Antun Stipančić [Q610070]	Mitsuru Kohno[Q1371156] _ [Q17]	872092.2
1975 World Table Tennis Championships [Q339390]	Men's singles [Q48860954]	István Jónyer[Q579551] _ [Q28]	_ [Q83286] Antun Stipančić [Q610070]	Norio Takashima[Q11669684] _ [Q17]	872092.2
1975 World Table Tennis Championships [Q339390]	Women's singles [Q48990338]	Pak Yung-Sun[Q513352] _ [Q423]	Zhang Li [Q197235] _ [Q148]	_ [Q15180] Tatiana Ferdman[Q2395657]	872092.2
1975 World Table Tennis Championships [Q339390]	Women's singles [Q48990338]	Pak Yung-Sun[Q513352] _ [Q423]	Zhang Li [Q197235] _ [Q148]	_ [Q148] Ge Xin'ai[Q438572]	872092.2
1975 World Table Tennis Championships [Q339390]	Men's doubles [Q55600930]	István Jónyer[Q579551] Gábor Gergely[Q673552] _ [Q28]	_ [Q83286] Dragutin Šurbek[Q1255276] Antun Stipančić[Q610070]	Jacques Secrétin[Q1563253] Jean-Denis Constant[Q16269365] _ [Q142]	872092.2
1975 World Table Tennis Championships [Q339390]	Men's doubles [Q55600930]	István Jónyer[Q579551] Gábor Gergely[Q673552] _ [Q28]	_ [Q83286] Dragutin Šurbek[Q1255276] Antun Stipančić[Q610070]	Katsuyuki Abe[Q11657772] Shigeo Itoh[Q2165647] _ [Q17]	872092.2
1975 World Table Tennis Championships [Q339390]	Women's doubles [Q55600934]	_ [Q218] Shoko Takahashi[Q50281830] Maria Alexandru[Q446797] _[Q17]	_ [Q148] Zhu Xiangyun[Q50771538] Lin Meiqun[Q50770702]	_ [Q15180] Tatiana Ferdman[Q2395657] Elmira Antonyan[Q693175]	872092.2
1975 World Table Tennis Championships [Q339390]	Women's doubles [Q55600934]	_ [Q218] Shoko Takahashi[Q50281830] Maria Alexandru[Q446797] _[Q17]	_ [Q148] Zhu Xiangyun[Q50771538] Lin Meiqun[Q50770702]	Sachiko Yokota[Q11543370] _[Q17] Yukie Ozeki[Q515682]	872092.2
1975 World Table Tennis Championships [Q339390]	Mixed doubles [Q55600933]	Stanislav Gomozkov[Q2331207] _[Q15180] Tatiana Ferdman[Q2395657]	Sarkis Sarchayan[Q50322034] _[Q15180] Elmira Antonyan[Q693175]	Zhang Li[Q197235] _[Q148] Liang Geliang[Q1822722]	872092.2
1975 World Table Tennis Championships [Q339390]	Mixed doubles [Q55600933]	Stanislav Gomozkov[Q2331207] _[Q15180] Tatiana Ferdman[Q2395657]	Sarkis Sarchayan[Q50322034] _[Q15180] Elmira Antonyan[Q693175]	Shigeo Itoh[Q2165647] _[Q17] Yukie Ozeki[Q515682]	872092.2

cell as the country variable and the second entity as the athlete variable, which worked well for that table.

However, the issue arises from the fact that, as depicted in the sample in Figure 5.11, the tables grouped in the cluster share the same headers but not necessarily the same order within the cells. For instance, in some tables, the country is presented at the beginning of the cell (Figure 5.11b), while in others, it's presented last (Figure 5.11a). Moreover, within the data corpus, the extraction of entities doesn't consistently follow the order of the original table. For example, in Table 5.21, in the first row of the "Gold" column, the entity for the country is last, whereas in the "Silver" column, the entity for the country is first, despite the observation in Figure 5.11a that in both cells, the country is situated last within the cell. It's important to note that this issue lies not with this work, but rather with the extraction framework. This could potentially be addressed by re-running the extraction of tables from Wikipedia.

Another challenge is that the entity representing the country may not always be the

identifier for the country itself. For instance, in Table 5.21, sometimes we have an entity like Japan[Q17], while in other cases, we have Canada at the 2015 Pan American Games[Q17513922]. Additionally, Table 5.21 shows that the number of athletes who won a certain medal may vary. This issue could be addressed in the future by post-processing the extracted triples, for example, by checking if the entities belong to a certain type, such as a country.

Due to these reasons, which are external to this work, we cannot extract the relations expected in Figure 5.12 without knowing the order of the entities and without discerning which entities correspond to athletes, countries, or participation of a country at a certain event.

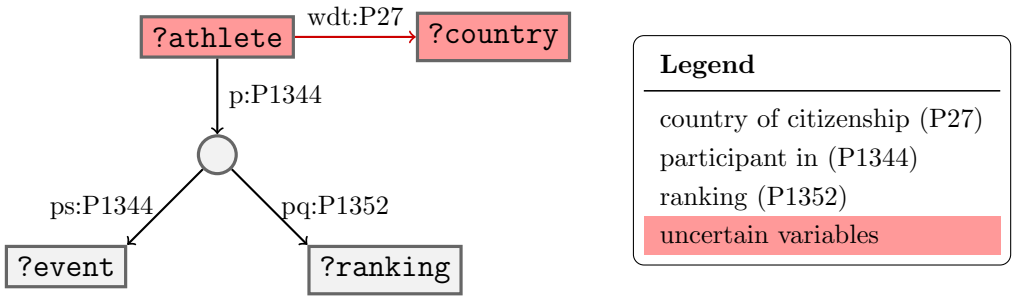

Figure 5.12: Graph pattern of expected triples for Cluster 22

5.2.2 Cluster 90

Cluster description This cluster groups tables containing information about sport games and their respective gold, silver and bronze medalists. The games listed in each table follow a consecutive order. Table 5.22 displays a sample of two tables from Cluster 90, showing only the columns that we will use for extracting relations, while their corresponding full HTML tables are presented in Figure 5.13. An additional column in Table 5.22 includes the identifiers of the tables.

This cluster contains a total of 2,177 tables with 20,847 rows in total and 4 columns.

Mappings applied We have applied similar mappings to those applied in Section 4.6 in order to extract triples following the pattern illustrated in Figure 4.9 over tables from Cluster 90. The mappings applied are presented in Section A.11.

As explained in Section 4.6, the only processor capable of extracting relations between a cell and the following cell within a column is Morph-KGC through the use of RML Views. Listing A.44 presents the mapping applied, which is similar to the mapping presented in Listing 4.28 with the following modifications. As seen in the example in Figure 5.13, the cells in games are often multivalued, following the pattern "name of the game, details". In

Table 5.22: Sample from Cluster 90

ArticleEntity	Game	TableID
Archery at the Youth Olympic Games [Q25220667]	2010 Singapore details [Q1086462] [Q1073794]	725259.2
Archery at the Youth Olympic Games [Q25220667]	2014 Nanjing details [Q15671385] [Q18281104]	725259.2
Archery at the Youth Olympic Games [Q25220667]	2018 Buenos Aires [Q870879]	725259.2
List of Olympic medalists in weightlifting [Q777748]	2000 Sydney details [Q602592] [Q7980006]	972218.12
List of Olympic medalists in weightlifting [Q777748]	2004 Athens details [Q748707] [Q1750587]	972218.12
List of Olympic medalists in weightlifting [Q777748]	2008 Beijing details [Q212887] [Q1095367]	972218.12
List of Olympic medalists in weightlifting [Q777748]	2012 London details [Q190233] [Q1962531]	972218.12
List of Olympic medalists in weightlifting [Q777748]	2016 Rio details [Q18564445] [Q26212190]	972218.12
List of Olympic medalists in weightlifting [Q777748]	2020 Tokyo details [Q39080756]	972218.12

Games	Gold	Silver	Bronze
2010 Singapore *details*	Kwak Ye-Ji South Korea	Tan Ya-Ting Chinese Taipei	Tatiana Segina Russia
2014 Nanjing *details*	Li Jiaman China	Melanie Gaubil France	Lee Eun-gyeong South Korea
2018 Buenos Aires			

(a) Table 725259.2

Games	Gold	Silver	Bronze
2000 Sydney *details*	Lin Weining China	Erzsébet Márkus Hungary	Karnam Malleswari India
2004 Athens *details*	Liu Chunhong China	Eszter Krutzler Hungary	Zarema Kasaeva Russia
2008 Beijing *details*	Oksana Slivenko Russia	Leydi Solís Colombia	Abeer Abdelrahman Egypt
2012 London *details*	Rim Jong-sim North Korea	Roxana Cocoș Romania	Anna Nurmukhambetova Kazakhstan
2016 Rio *details*	Xiang Yanmei China	Zhazira Zhapparkul Kazakhstan	Sara Ahmed Egypt
2020 Tokyo *details*			

(b) Table 972218.12

Figure 5.13: HTML table examples from Cluster 90
5.13a from the Wikipedia article `https://en.wikipedia.org/wiki/Archery_at_the_Summer_Youth_Olympics`
5.13b from the Wikipedia article `https://en.wikipedia.org/wiki/List_of_Olympic_medalists_in_weightlifting`

this case, we would like to extract the first entity within the cell corresponding to the game itself. For that, we use the SQL function `split_part`, as explained in Section 4.4. We also added the condition `TableID1 = TableID2` in the `JOIN` condition within the SQL query, which indicates that the current row and the following row belong to the same table. This ensures that when we process all the tables from the cluster, we will not get incorrect triples relating different types of games, for example, the triple: 2018 Buenos Aires [Q870879] followed by 2000 Sydney [Q602592] in the Figure 5.22 example.

Similarly to what is presented in Section 4.6, we perform preprocessing of the input data to concatenate each row with the following row. We also apply the condition that the current row and the following row belong to the same table during preprocessing. Then, the extraction of triples remains on the same row. As mentioned before, the cells in column "Games" are multivalued, and for that, we can apply the features described in Section 4.4 to extract the first entity within a row. These features are only present in the SPARQL-based mapping languages as well as Morph-KGC through the use of RML Views.

Results The extracted triples, execution time, and already existing triples in Wikidata without applying preprocessing of the data, are presented in Table 5.23. The results of preprocessing the data before applying the mappings are presented in Table 5.24.

Table 5.23: Cluster 90 output comparison

	№ of Triples	Execution time [s]	Existing triples on Wikidata	New triples
Tarql	-	-	-	-
SPARQL-Generate	-	-	-	-
SPARQL Anything	-	-	-	-
CARML	-	-	-	-
RMLMapper	-	-	-	-
SDM-RDFizer	-	-	-	-
Chimera	-	-	-	-
Morph-KGC	6,556	8.50	2,099	4,457

Table 5.24: Cluster 90 output comparison with preprocessing of input data

	№ of Triples	Execution time [s]	Existing triples on Wikidata	New triples
Tarql	6,556	3.41	2,099	4,457
SPARQL-Generate	6,556	3.52	2,099	4,457
SPARQL Anything	6,556	5.08	2,099	4,457
CARML	-	-	-	-
RMLMapper	-	-	-	-
SDM-RDFizer	-	-	-	-
Chimera	-	-	-	-
Morph-KGC	6,556	8.10	2,099	4,457

The triples extracted from the sample in Table 5.22 are shown in Listing 5.11.

```
wd:Q1086462 wdt:P156 wd:Q15671385.
wd:Q15671385 wdt:P156 wd:Q870879.
wd:Q602592 wdt:P156 wd:Q748707.
wd:Q748707 wdt:P156 wd:Q212887.
wd:Q212887 wdt:P156 wd:Q190233.
wd:Q190233 wdt:P156 wd:Q18564445.
wd:Q18564445 wdt:P156 wd:Q39080756.
```

Listing 5.11: Sample from Cluster 90 output

Summary We extracted 6,556 triples from a total of 20,847 rows. This corresponds to the unique number of entity pairs between the current cell in the "game" column and the following cell within the same column. Among these 6,556 triples, 2,099 are already present in Wikidata, accounting for 32.0%. We found a total of 4,457 new triples.

5.3 Precision of the extracted triples

We evaluated the precision of novel relations extracted from the top ten largest clusters by the number of tables. These novel relations are those that were not previously present in Wikidata. Within each of these clusters, we randomly sampled 50 relations from the output of Tarql, the mapping language with the most extracted triples, to assess their precision, thus assessing 500 relations in total. Precision is calculated as the ratio of correctly classified relations to the total number of relations sampled[7].

Table 5.25 displays the results obtained. We classify relations as either correct or incorrect. Although some relations may require additional qualifiers to fully convey the information, they are still deemed correct. This is due to the open world assumption in Wikidata, where missing information is treated as unknown rather than false. Consequently, such relations are considered valid, with the potential for further information to be added in the form of qualifiers in the future.

Some of the reasons for why the relations are considered as incorrect are as follows:

- **Incorrect Wikipedia link:** The link provided in the Wikipedia table (from which the Wikidata entity was extracted by Luzuriaga [25]) is incorrect. For example, the triple Gerardo Flores[] position played on team[P413] defender[Q336286] extracted from Cluster 5, is correct for the subject Gerardo Flores[Q5550269] the footballer, but the entity extracted was Gerardo Flores[Q5550262] the murderer.

[7]Precision is calculated by considering the correct triples extracted as true positives and the retrieved triples as the sum of true positives and false positives, following the definition of Precision by D. Manning et al.(2008) [26]

- **Subject or object doesn't have an entity:** The subject or object of the triple doesn't have a Wikidata entity, but there is a different entity in the same cell. For example, the song "Another Day" from Cluster 4 doesn't have a Wikidata entity. Therefore, the entity extracted in the "Title" column in Figure 5.14 for that cell was M. J. Cole[Q708129]. When we executed the mapping for this cluster we obtained the incorrect triple M. J. Cole[Q708129] form of creative work[P7937] song[Q7366].

No.	Title	Length
5.	"Another Day" (MJ Cole Remix)	5:42
8.	"Show Me a Sign" (Popeska Remix)	4:08

Figure 5.14: Subject doesn't have an entity[8]

- **Incorrect assumption in mapping:** When we defined the mappings, we made some assumptions about the articles from which the tables were extracted. For example, in Cluster 6, we assumed that the article corresponded to a sports event, and in Clusters 4 and 9, we assumed that the article corresponded to an album. While in most cases, these assumptions were accurate, in some cases they led to incorrect triples being extracted. For instance, for the article 'The Time of the Oath (song)' from Cluster 9, we obtained the incorrect triple The Time of the Oath[Q760753] instance of[P31] album[Q482994].

- **Columns with diverse entity types:** A column may contain subtly different types of entities. For example, in Cluster 8, some tables indicate a navy instead of a location in the "State" column. For example, from the table in Figure 5.15, we extracted the incorrect triple Rossia[Q690108] country of registry[P8047] Soviet Navy[Q796754]. We also included incorrect facts in this category.

Ship	State	Description
Rossia	Soviet Navy	The armored cruiser broke free from her tow in the Baltic Sea and stranded on the Dyvelseye Shoal.[253] She was refloated in July 1922.

Figure 5.15: Columns with diverse entity types[9]

For clusters 1, 3, and 7, all the sampled relations were correct, obtaining a precision of 100%. The average precision of the sampled relations of the top ten largest clusters is 84.6%.

[8] abridged from the Wikipedia article `https://en.wikipedia.org/wiki/Evolution_Theory_(Modestep_album)`

[9] abridged from the Wikipedia article `https://en.wikipedia.org/wiki/List_of_shipwrecks_in_1922`

Table 5.25: Evaluation of extracted triples from the Top 10 largest clusters

Cluster	Correct	Incorrect	Precision [%]	Reasons for Incorrectness	№ of incorrect relations per reason
1	50	0	100	–	
2	46	4	92	Incorrect Wikipedia link	4
3	50	0	100	–	
4	25	25	50	Subject or object doesn't have an entity Incorrect assumption in mapping Incorrect Wikipedia link	21 2 2
5	43	7	86	Subject or object doesn't have an entity Incorrect Wikipedia link	4 3
6	42	8	84	Incorrect assumption in mapping Incorrect Wikipedia link	7 1
7	50	0	100	–	
8	43	7	86	Columns with diverse entity types Incorrect Wikipedia link	6 1
9	40	10	80	Subject or object doesn't have an entity Incorrect assumption in mapping	8 2
10	34	16	68	Incorrect Wikipedia link Subject or object doesn't have an entity	9 7

5.4 Language & processor comparison

In this chapter, we applied various mappings using Tarql, SPARQL-Generate, SPARQL Anything, and RML (including CARML, RMLMapper, SDM-RDFizer, Chimera and Morph-KGC processors).

In certain cases, like in Section 5.1.4 and Section 5.1.9, a complete RML mapping using CARML, RMLMapper, SDM-RDFizer and Chimera wasn't defined due to the need to create custom functions. The functions needed were for processing the input data, extracting substrings from cells, retrieving specific entities from cells in multivalued tables, assigning data types to literal values, and applying conditional bindings for variables. These capabilities were achieved using RML Views in Morph-KGC. Additionally, in SPARQL-based mappings, similar functionalities were successfully achieved through the use of SPARQL features and Apache Jena features.

We encountered difficulties with noise and subheaders present in the input data. These issues were addressed using SPARQL functions in SPARQL-based mapping languages and SQL functions using RML Views. Additionally, several mappings required the handling of multivalued tables, selecting the n^{th} entity within a cell. This task was efficiently managed using SPARQL functions or SQL functions via RML Views.

As discussed in Section 4.9, while generating blank nodes the RML specification requires a template constraint that makes the definition of mappings of n-ary relations when defining

a template not straightforward.

Comparatively, SPARQL-based language mappings have demonstrated to be more concise and expressive than the mappings in RML. However, exceptions arose in cases where `JOIN` operations and row identification were necessary, such as in Subsection 5.2.2. In that particular use case, Morph-KGC showcased greater expressiveness, making it a viable alternative to SPARQL-based approaches.

Figure 5.16 presents a comparison in execution times for each top ten largest clusters and for each processor tested. Note that the timeout (TMO) of 3600 seconds is set lower in the figure for visualization purposes. We observe that Tarql and CARML exhibited the fastest execution times in most of the experiments. However, it's important to note that in some cases, CARML extracted fewer triples due to the necessity of establishing a template for blank nodes, a situation also encountered in SDM-RDFizer and Morph-KGC mappings, as we see in Figure 5.17. Another issue with blank nodes in SDM-RDFizer and Morph-KGC was the presence of special characters in the assigned template, which are illegal in the RDF syntax for blank nodes and thus were not supported by the RDF tool RDFLib, the tool we rely on to process the extracted triples. This highlights the necessity to preprocess the data to avoid those characters if working with these processors.

Figure 5.17 shows a comparison of the number of triples extracted per processor, aggregating all ten largest clusters within the data corpus. Each stacked bar also illustrates the number of new triples that are not present in Wikidata. We obtained the highest number of triples extracted in the SPARQL-based mapping languages, with the highest extracted by Tarql, totaling 984,260 triples, of which 791,021 are novel in Wikidata.

Figure 5.18 shows a comparison of the number of relations extracted per processor, aggregating all ten largest clusters within the data corpus. The number of relations is fewer than the number of triples extracted shown in Figure 5.17, as a relation may contain multiple triples.

While we didn't categorize all relations extracted in Section 5.1.4 and Section 5.1.9, as they only complemented the information of the already categorized triples, and furthermore, we didn't query Wikidata for all relations extracted in those sections, we can affirm that we obtained at least the number of new triples and new relations displayed in Figure 5.17 and Figure 5.18.

Regarding the number of triples extracted, they were generally similar, except in cases where blank nodes were involved. This discrepancy arose due to the need for a template definition in CARML, SDM-RDFizer and Morph-KGC for generating the blank nodes. In the case of RMLMapper and Chimera processors, this was not an issue as they do not follow this constraint.

As RMLMapper often reached a timeout, it resulted in fewer triples being extracted.

While comparing the SPARQL-based mapping languages processors, we found that we have to declare `OPTIONAL` variables in the `SERVICE` clause of SPARQL Anything if some of them have unbound values, while this is not necessary in Tarql nor SPARQL-Generate. Additionally, SPARQL Anything required extra functions while handling literal values, as it considered empty strings. In contrast, these triples were automatically omitted in TARQL and SPARQL-Generate.

In the comparison between Tarql and SPARQL-Generate, Tarql demonstrated similar results in number of triples extracted (Figure 5.17) but faster execution times (Figure 5.16).

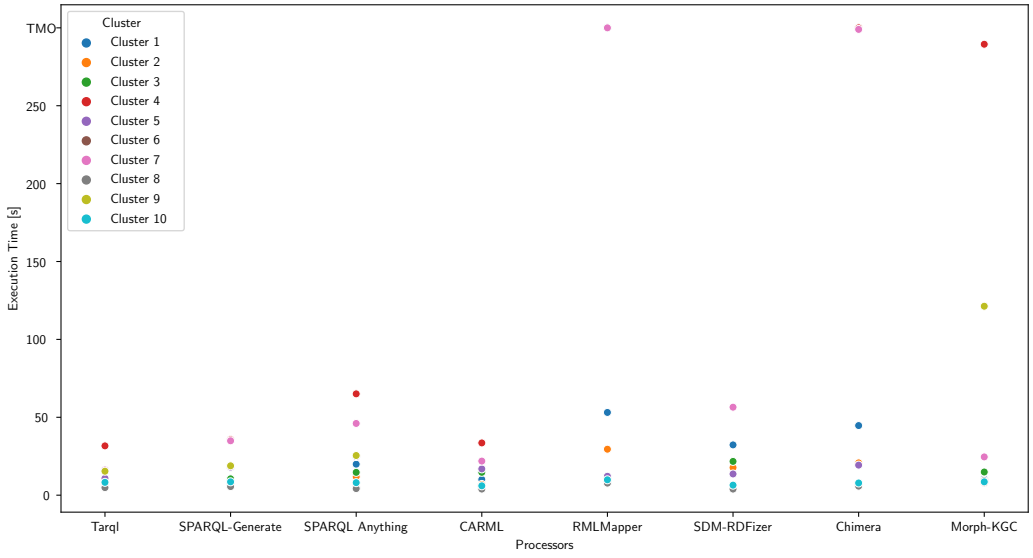

Figure 5.16: Execution Time for each Cluster and Processor

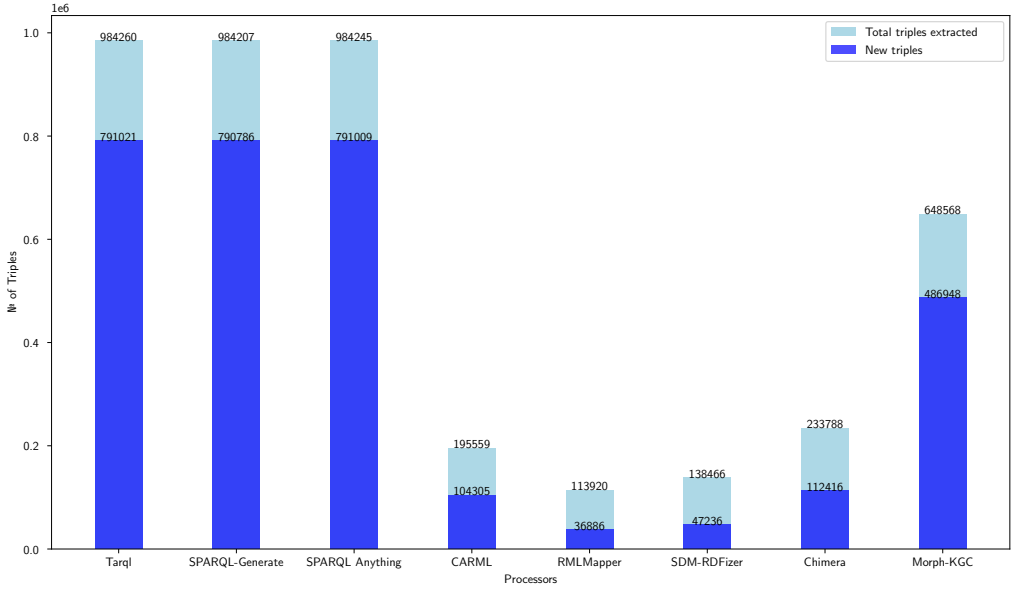

Figure 5.17: New triples over all triples extracted for each Processor - Top 10 clusters

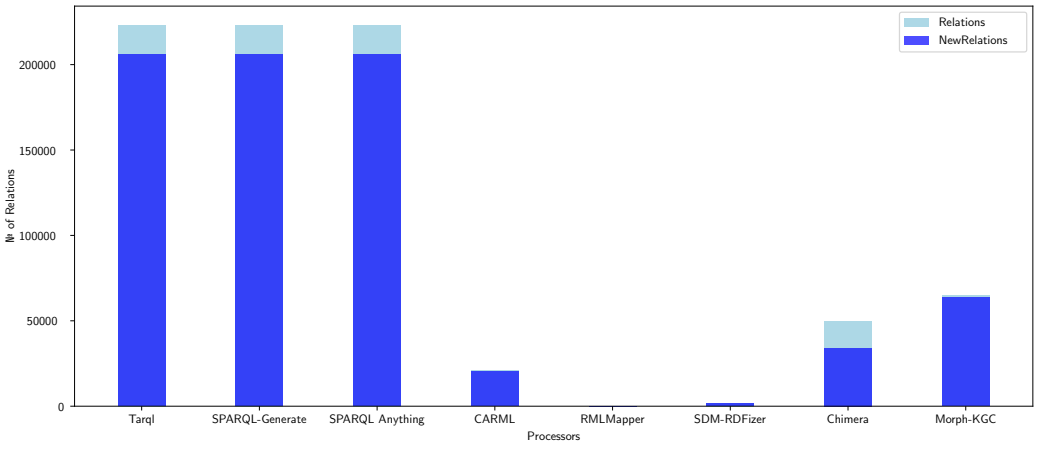

Figure 5.18: New relations over all relations extracted for each Processor - Top 10 clusters

Chapter 6

Conclusions and Future Work

This thesis presents a comparison analysis of RDF mapping languages and their corresponding processors, by studying their suitability for extracting RDF triples from Wikipedia tables using a previous work that merges Wikipedia tables having the same table headers.

Additionally, we review the current state of the art of the Semantic web; the extraction, categorization and interpretation of web tables, as well as the state of the art of the different mapping languages and processors currently available for the extraction of semantic triples from semi-structured data.

We addressed the problem that presents the extraction of RDF triples from the Wikipedia tables, given the large number of tables and the multiple types of table schemas that complicate the manual extraction of those triples. Even though in the state of the art there exist automatic techniques for extracting those triples, those methods are not always reliable and the precision needed to be improved to populate knowledge graphs such as Wikidata.

We provided an analysis of the Wikipedia table corpus, offering insights into individual tables and table clusters to understand the extent of information contained in Wikipedia tables. Notably, the largest cluster, consisting of 81,277 tables with 1,202,635 rows, features 536,939 rows suitable for relation extraction due to multi-column entity relationships.

We emphasize the importance of selecting an appropriate language and processor to extract relations from this specific data corpus. Using various examples with different table schemas, including horizontal relational, multivalued, and matrix tables, we assess the suitability of RDF mapping languages and processors. Specifically, we evaluate SPARQL- and RML-based mapping languages to RDF, examining their expressiveness and performance.

While the mapping languages studied showed similar results in extracting simple triples, SPARQL-based mapping languages and the use of SQL in RML Views offered significant advantages in data processing, filtering, and handling complex table schemas.

Experimental evaluations involve applying mappings to the top ten largest clusters by table count, measuring triple and relation extraction, processor performance, and the quantity of novel triples not present in Wikidata. Additionally, selected clusters demonstrate the suitability of mapping languages and processors for other cluster types.

While previous studies compare some mapping languages and processors, none address this particular data corpus. Our comparison of different tools applied to our data corpus highlights Tarql as the most productive processor with the best performance time for processing large clusters. Tarql generated 984,260 triples, with 791,021 being novel in Wikidata. Furthermore, sampling 50 random relations extracted by Tarql from each of the top 10 largest clusters yielded an average precision of 84.6%. However, as the imprecision is primarily associated with specific clusters, excluding these clusters could significantly improve the precision achieved. Previous automated methods achieved a precision of 81.5% [29] and 70% [25], indicating an improvement in precision with our approach.

These findings support our hypothesis that significant volumes of high-precision novel triples can be extracted from Wikipedia tables by clustering them and utilizing mapping languages to RDF over these clusters for populating knowledge graphs, such as Wikidata.

While this thesis has provided valuable insights into the extraction of RDF triples from Wikipedia tables using various mapping languages and processors, it is essential to acknowledge some limitations of this work and avenues for future work.

Firstly, the data corpus used in this study consists of Wikipedia tables extracted by Luzuriaga [25] in 2019. Although this dataset serves as a foundation for our analysis, the quality of extracted relations could potentially be enhanced by incorporating a more up-to-date data corpus. Updating the corpus would ensure the relevance and accuracy of the extracted information.

Moreover, there are opportunities for enhancing the granularity of information extracted. For instance, providing additional details such as the position of entities within individual cells in multivalued tables and associating entities with table titles could enrich the extracted data.

In terms of future work, one promising avenue is the development of a systematic approach for relation extraction from Wikipedia tables using selected mapping languages and processors based on the findings of this thesis. This system could empower expert users to apply mappings across clusters of tables, facilitating the extraction of significant volumes of information with minimal manual effort.

Furthermore, the novel relations extracted in this study present an opportunity for further enrichment of knowledge graphs such as Wikidata. Future endeavors could involve the integration of these extracted relations into Wikidata, thereby enabling Wikidata users to expand upon this information by adding additional details and context.

Another potential approach involves post-processing the extracted triples, such as verifying the entity types in Wikidata. This could be a quick and useful way to improve precision even further.

In summary, while this thesis has made significant strides in understanding the extraction of RDF triples from Wikipedia tables, there remains ample room for improvement and exploration in terms of dataset quality and system development for scalable relation extraction. These avenues for future work hold the potential to further advance the field of semantic data extraction from Wikipedia tables and enrich knowledge bases like Wikidata.

Bibliography

[1] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, and Juan Sequeda. A Direct Mapping of Relational Data to RDF. `https://www.w3.org/TR/rdb-direct-mapping/`, 2012.

[2] Julián Arenas-Guerrero, Mario Scrocca, Ana Iglesias Molina, Jhon Toledo, Luis Pozo-Gilo, Daniel Doña, Oscar Corcho, and David Chaves-Fraga. Knowledge Graph Construction with R2RML and RML: An ETL System-based Overview. *CEUR workshop proceedings.*, 2873, June 2021. Ontology Engineering Group OEG.

[3] Julián Arenas-Guerrero, Ahmad Alobaid, María Navas-Loro, María Pérez, and Oscar Corcho. Boosting Knowledge Graph Generation from Tabular Data with RML Views. In *Proceedings of the 20th Extended Semantic Web Conference*, volume 13870, pages 484–501. Springer Nature Switzerland, May 2023. Ontology Engineering Group OEG.

[4] Julián Arenas-Guerrero, David Chaves-Fraga, Jhon Toledo, María S. Pérez, and Oscar Corcho. Morph-KGC: Scalable knowledge graph materialization with mapping partitions. *Semantic Web*, 2022.

[5] Dave Beckett and Brian McBride. RDF/XML syntax specification (revised). *W3C recommendation*, 10(2.3), 2004.

[6] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. RDF 1.1 Turtle. *World Wide Web Consortium*, pages 18–31, 2014.

[7] Tim Berners-Lee, Wendy Hall, James Hendler, Kieron O'Hara, Nigel Shadbolt, and Daniel Weitzner. A Framework for Web Science. *Foundations and Trends® in Web Science*, 1, 01 2006.

[8] Tim Berners-Lee, Wendy Hall, James Hendler, Kieron O'Hara, Nigel Shadbolt, and Daniel Weitzner. The Semantic Web Stack. Reproduced as Figure 3.2 in [7], 2006.

[9] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[10] Gavin Carothers and Andy Seaborne. RDF 1.1 N-Triples: A line-based syntax for an RDF graph. *World Wide Web Consortium. http://www.w3.org/TR/n-triples/*, 24, 2014.

[11] Alexandros Chortaras and G. Stamou. D2RML: Integrating Heterogeneous Data and Web Services into Custom RDF Graphs. In *LDOW@WWW*, 2018.

[12] Eric Crestan and Patrick Pantel. Web-scale table census and classification. In Irwin King, Wolfgang Nejdl, and Hang Li, editors, *Web Search and Web Data Mining (WSDM)*, pages 545–554. ACM, 2011. DOI:10.1145/1935826.1935904.

[13] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. `https://www.w3.org/TR/r2rml/`, 2012.

[14] Anastasia Dimou, Miel Vander Sande, Ben De Meester, Pieter Heyvaert, and Thomas Delva. RDF Mapping Language (RML). `https://rml.io/specs/rml/`, 2022.

[15] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. SPARQL 1.1 Query Language. `https://www.w3.org/TR/sparql11-query/`, 2013.

[16] Enrique Iglesias, Samaneh Jozashoori, David Chaves-Fraga, Diego Collarana, and Maria-Esther Vidal. SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, page 3039–3046, New York, NY, USA, 2020. Association for Computing Machinery.

[17] Ana Iglesias-Molina, Andrea Cimmino Arriaga, Edna Ruckhaus, David Chaves-Fraga, Raúl García Castro, and Oscar Corcho. An Ontological Approach for Representing Declarative Mapping Languages. *Semantic Web*, 12 2022.

[18] Ademar Crotti Junior, Christophe Debruyne, Rob Brennan, and Declan O'Sullivan. Funul: a method to incorporate functions into uplift mapping languages. In *Proceedings of the 18th International Conference on Information Integration and Web-Based Applications and Services*, iiWAS '16, page 267–275, New York, NY, USA, 2016. Association for Computing Machinery.

[19] Gregg Kellogg, Pierre-Antoine Champin, and Dave Longley. JSON-LD 1.1 – A JSON-based Serialization for Linked Data. Technical report, W3C, December 2019.

[20] Holger Knublauch and Dimitris Kontokostas. Shapes Constraint Language (SHACL). `https://www.w3.org/TR/shacl/`, 2017.

[21] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. A SPARQL extension for generating RDF from heterogeneous formats. In Eva Blomqvist, Diana Maynard, Aldo Gangemi, Rinke Hoekstra, Pascal Hitzler, and Olaf Hartig, editors, *The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May*

28 - June 1, 2017, Proceedings, Part I, volume 10249 of *Lecture Notes in Computer Science*, pages 35–50, 2017.

[22] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[23] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and Searching Web Tables Using Entities, Types and Relationships. *PVLDB*, 3:1338–1347, 09 2010.

[24] Jixiong Liu, Yoan Chabot, Raphaël Troncy, Viet-Phi Huynh, Thomas Labbé, and Pierre Monnin. From tabular data to knowledge graphs: A survey of semantic table interpretation tasks and methods. *Journal of Web Semantics*, 76:100761, 2023.

[25] Jhomara Luzuriaga. Merging HTML Tables for Extracting Relations. Tesis de Magíster, Universidad de Chile, Santiago, Chile, 2019.

[26] Manning, Christopher D and Raghavan, Prabhakar and Schütze, Hinrich. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[27] Varish Mulwad, Tim Finin, and Anupam Joshi. Semantic Message Passing for Generating Linked Data from Tables. In *International Semantic Web Conference (ISWC)*, pages 363–378. Springer, 2013. DOI:10.1007/978-3-642-41335-3_23.

[28] Emir Muñoz, Aidan Hogan, and Alessandra Mileo. Triplifying Wikipedia's Tables. In *First International Conference on Linked Data for Information Extraction - Volume 1057, LD4IE'13, pages 26–37, Aachen, Germany, Germany*, 2013.

[29] Emir Muñoz, Aidan Hogan, and Alessandra Mileo. Using linked data to mine RDF from Wikipedia's tables. In *Web Search and Web Data Mining (WSDM)*, pages 533–542. ACM, 2014. DOI:10.1145/2556195.2556266.

[30] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. `https://www.w3.org/TR/rdf-sparql-query/`, 2008.

[31] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, 2014.

ANNEX A

Experiments per Cluster - mappings

A.1 Cluster 1

```
CONSTRUCT {
    ?candidate wdt:P102 ?party.
}
FROM <file:1.csv>
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Candidate)) AS ?candidate)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?2_Party)) AS ?party)
}
```

Listing A.1: Tarql mapping for Cluster 1

```
<#Cluster1Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "1.csv" ;
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{Candidate}"
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P102 ;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{2_Party}"
        ]
    ].
```

Listing A.2: RML mapping for Cluster 1

```
CONSTRUCT {
    ?candidate wdt:P102 ?party.
}
WHERE {
    SERVICE <x-sparql-anything:location=1.csv,csv.headers=true> {
        fx:properties  fx:csv.null-string  "" .
        ?c        xyz:Candidate ?Candidate;
                  xyz:2_Party ?2_Party
    }
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Candidate)) AS ?candidate)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?2_Party)) AS ?party)
}
```

Listing A.3: SPARQL Anything mapping for Cluster 1

```
GENERATE {
    ?candidate wdt:P102 ?party .
}
ITERATOR iter:CSV(<1.csv>, "Candidate", "2_Party") AS ?Candidate ?2_Party
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Candidate)) AS ?candidate)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?2_Party)) AS ?party)
}
```

Listing A.4: SPARQL-Generate mapping for Cluster 1

A.2 Cluster 2

We have applied the same mappings as in Cluster A.1 over tables from Cluster 2.

A.3 Cluster 3

```
CONSTRUCT {
    ?album p:P444 _:review.
    _:review ps:P444 ?rateTrim;
             pq:P447 ?source.

}
FROM <file:3.csv>
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?ArticleEntity)) AS ?album)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Review_score__Source_3)) AS ?
    source)
    BIND(replace(?Review_score__rate_3, " ", "") as ?rateTrim)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Review_score__Rating_1)) AS ?
    rating)
    FILTER(BOUND(?source) || BOUND(?Review_score__rate_3))
}
```

Listing A.5: Tarql mapping for Cluster 3

```
CONSTRUCT {
    ?album p:P444 _:review.
    _:review ps:P444 ?Review_score__rate_3;
             pq:P447 ?source.
}
WHERE {
    SERVICE <x-sparql-anything:location=3.csv,csv.headers=true> {
        fx:properties   fx:trim-strings  true ;
                        fx:null-string  "".
        ?c        xyz:ArticleEntity ?ArticleEntity;
                  xyz:Review_score__Source_3 ?Review_score__Source_3;
                  xyz:Review_score__rate_3 ?Review_score__rate_3
}
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?ArticleEntity)) AS ?album)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Review_score__Source_3)) AS ?
    source)
    FILTER(BOUND(?source) || BOUND(?Review_score__rate_3))
}
```

Listing A.6: SPARQL Anything mapping for Cluster 3

```
GENERATE {
    ?album p:P444 _:review.
    _:review ps:P444 ?Review_score__rate_3;
             pq:P447 ?source.
}
ITERATOR iter:CSV(<3.csv>, "ArticleEntity", "Review_score__Source_3", "
    Review_score__rate_3") AS ?ArticleEntity ?Review_score__Source_3
?Review_score__rate_3
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?ArticleEntity)) AS ?album)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Review_score__Source_3)) AS ?
    source)
    FILTER(BOUND(?source) || BOUND(?Review_score__rate_3))
}
```

Listing A.7: SPARQL-Generate mapping for Cluster 3

```
<#CLuster3Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "3.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{ArticleEntity}";
    ];
    rr:predicateObjectMap [
        rr:predicate p:P444;
        rr:objectMap [
            rr:termType rr:BlankNode ;
            rr:template "_:source{ArticleEntity}{ArticleTitle}{Review_score__rate_3}{
    Review_score__Source_3}"
        ]
    ].
<#Review> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "3.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode;
        rr:template "_:source{ArticleEntity}{ArticleTitle}{Review_score__rate_3}{
    Review_score__Source_3}"
    ];
    rr:predicateObjectMap [
        rr:predicate ps:P444;
        rr:objectMap [
            rr:reference "Review_score__rate_3"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate pq:P447;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Review_score__Source_3}"
        ]
    ].
```

Listing A.8: CARML and SDM-RDFizer mapping for Cluster 3

```
<#CLuster3Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "../Input/cluster3.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{ArticleEntity}";
    ];
    rr:predicateObjectMap [
        rr:predicate p:P444;
        rr:objectMap [
            rr:parentTriplesMap <#Review>;
            rr:joinCondition [
                rr:child "ArticleEntity";
                rr:parent "ArticleEntity";
            ];
  ]
    ].

<#Review> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "../Input/cluster3.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode
    ];
    rr:predicateObjectMap [
        rr:predicate ps:P444;
        rr:objectMap [
     rml:reference "Review_score__rate_3"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate pq:P447;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Review_score__Source_3}"
        ]
    ].
```

Listing A.9: RMLMapper and Chimera mapping for Cluster 3

```
<#CLuster3Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "3.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{ArticleEntity}";
    ];
    rr:predicateObjectMap [
        rr:predicate p:P444;
        rr:objectMap [
            rr:termType rr:BlankNode ;
            rr:template "_:source{ArticleEntity}{ArticleTitle}{Review_score__rate_3}{
    Review_score__Source_3}"
        ]
    ].
<#Review> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "3.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode;
        rr:template "_:source{ArticleEntity}{ArticleTitle}{Review_score__rate_3}{
    Review_score__Source_3}"
    ];
    rr:predicateObjectMap [
        rr:predicate ps:P444;
        rr:objectMap [
           rml:functionExecution <#Execution>
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate pq:P447;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Review_score__Source_3}"
        ]
    ].
<#Execution> a rml:Execution ;
    rml:function grel:string_trim ;
    rml:input [
        rml:parameter grel:valueParam ;
        rml:inputValueMap [
            rml:reference "Review_score__rate_3"
        ]
    ] .
```

Listing A.10: Morph-KGC mapping for Cluster 3

```
[DataSourceCSV]
mappings=Morph-KGC_mapping.ttl

[CONFIGURATION]
output_file: Morph-KGC_outputTriples.nt
na_values=,#N/A,N/A,#N/A N/A,n/a,NA,<NA>,#NA,NULL,null,NaN,nan,None,"","""","\""
na_filter= 'yes'
```

Listing A.11: Morph-KGC configuration file for Cluster 3

A.4 Cluster 4

```
CONSTRUCT {
    ?song wdt:P31 wd:Q105543609;
        wdt:P361 ?album;
        wdt:P7937 wd:Q7366;
        wdt:P2047 ?lengthSeconds;
        p:P2047 ?duration.

    ?duration ps:P2047 ?lengthSeconds;
        psv:P2047 ?b1.

    ?b1 a wikibase:QuantityValue;
        wikibase:quantityAmount ?lengthSeconds;
        wikibase:quantityUnit wd:Q11574;
        wikibase:quantityNormalized ?b1.

    ?album wdt:P31 wd:Q482994;
        wdt:P658 ?song;
        p:P658 ?tracklist.

    ?tracklist ps:P658 ?song;
        pq:P1545 ?number.

}
FROM <file:../Input/cluster4.csv>
WHERE {
    BIND ( IF (BOUND(?ArticleEntity) && strlen(?ArticleEntity)>0, URI(CONCAT('http://www.
    wikidata.org/entity/', ?ArticleEntity)),?undef) AS ?album )
    BIND ( IF (BOUND(?TitleID) && strlen(?TitleID)>0,  REPLACE(replace(?TitleID, " ", "")
    ,",Q[0-9]*",""),?undef) AS ?first )
    BIND ( IF (BOUND(?length) && strlen(?length)>0, REPLACE(REPLACE(?length,"[^0-9:]","")
    ,"(?<=[0-9]+:[0-9]{2}).*",""),?undef) AS ?l )
    BIND ( IF (BOUND(?no) && strlen(?no)>0, REPLACE(?no,"[^0-9+]",""),?undef) AS ?n )
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?first)) AS ?song)
    BIND (xsd:decimal(strbefore(?l,":"))*60 + xsd:decimal(strafter(?l,":"))  AS ?
    lengthSeconds)
    BIND ( IF (BOUND(?lengthSeconds) && BOUND(?song), BNODE(),?undef) AS ?duration )
    BIND ( IF (BOUND(?lengthSeconds) && BOUND(?song), BNODE(),?undef) AS ?b1 )
    BIND ( IF (BOUND(?song) && BOUND(?album), BNODE(),?undef) AS ?tracklist )
    BIND ( IF (BOUND(?n) && strlen(?n)>0, ?n ,?undef) AS ?number )
    VALUES ?undef {UNDEF}
}
```

Listing A.12: Tarql mapping for Cluster 4

```
CONSTRUCT {
    ?song wdt:P31 wd:Q105543609;
        wdt:P361 ?album;
        wdt:P7937 wd:Q7366;
        wdt:P2047 ?lengthSeconds;
        p:P2047 ?duration.

    ?duration ps:P2047 ?lengthSeconds;
        psv:P2047 ?b1.

    ?b1 a wikibase:QuantityValue;
        wikibase:quantityAmount ?lengthSeconds;
        wikibase:quantityUnit wd:Q11574;
        wikibase:quantityNormalized ?b1.

    ?album wdt:P31 wd:Q482994;
        p:P658 ?tracklist.

    ?tracklist ps:P658 ?song;
        wdt:P658 ?song;
        pq:P1545 ?number.

}
FROM <file:../Input/cluster4.csv>
WHERE {
SERVICE <x-sparql-anything:location=../Input/cluster4.csv,csv.headers=true> {
    OPTIONAL{ ?c xyz:ArticleEntity ?ArticleEntity .}
    OPTIONAL{ ?c xyz:no ?no .}
    OPTIONAL{ ?c xyz:TitleID ?TitleID .}
    OPTIONAL{ ?c xyz:length ?length .}
}
    BIND ( IF (BOUND(?ArticleEntity) && strlen(?ArticleEntity)>0, URI(CONCAT('http://www.
    wikidata.org/entity/', ?ArticleEntity)),?undef) AS ?album )
    BIND ( IF (BOUND(?TitleID) && strlen(?TitleID)>0,  REPLACE(replace(?TitleID, " ", "")
    ,",Q[0-9]*",""),?undef) AS ?first )
    BIND ( IF (BOUND(?length) && strlen(?length)>0, REPLACE(REPLACE(?length,"[^0-9:]","")
    ,"(?<=[0-9]+:[0-9]{2}).*",""),?undef) AS ?l )
    BIND ( IF (BOUND(?no) && strlen(?no)>0, REPLACE(?no,"[^0-9+]",""),?undef) AS ?n )
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?first)) AS ?song)
    BIND (xsd:decimal(strbefore(?l,":"))*60 + xsd:decimal(strafter(?l,":"))  AS ?
    lengthSeconds)
    BIND ( IF (BOUND(?lengthSeconds) && BOUND(?song), BNODE(),?undef) AS ?duration )
    BIND ( IF (BOUND(?lengthSeconds) && BOUND(?song), BNODE(),?undef) AS ?b1 )
    BIND ( IF (BOUND(?song) && BOUND(?album), BNODE(),?undef) AS ?tracklist )
    BIND ( IF (BOUND(?n) && strlen(?n)>0, ?n ,?undef) AS ?number )
    VALUES ?undef {UNDEF}
}
```

Listing A.13: SPARQL Anything mapping for Cluster 4

```
GENERATE {
    ?song wdt:P31 wd:Q105543609;
        wdt:P361 ?album;
        wdt:P7937 wd:Q7366;
        wdt:P2047 ?lengthSeconds;
        p:P2047 ?duration.

    ?duration ps:P2047 ?lengthSeconds;
        psv:P2047 ?b1.

    ?b1 a wikibase:QuantityValue;
        wikibase:quantityAmount ?lengthSeconds;
        wikibase:quantityUnit wd:Q11574;
        wikibase:quantityNormalized ?b1.

    ?album wdt:P31 wd:Q482994;
        wdt:P658 ?song;
        p:P658 ?tracklist.

    ?tracklist ps:P658 ?song;
        pq:P1545 ?number.
}
ITERATOR iter:CSV(<../Input/cluster4.csv>, "ArticleEntity", "TitleID", "length","no") AS
    ?ArticleEntity ?TitleID
?length ?no

WHERE {
    BIND ( IF (BOUND(?ArticleEntity) && strlen(?ArticleEntity)>0, URI(CONCAT('http://www.
    wikidata.org/entity/', ?ArticleEntity)),?undef) AS ?album )
    BIND ( IF (BOUND(?TitleID) && strlen(?TitleID)>0,  REPLACE(replace(?TitleID, " ", "")
    ,",Q[0-9]*",""),?undef) AS ?first )
    BIND ( IF (BOUND(?length) && strlen(?length)>0, REPLACE(REPLACE(?length,"[^0-9:]","")
    ,"(?<=[0-9]+:[0-9]{2}).*",""),?undef) AS ?l )
    BIND ( IF (BOUND(?no) && strlen(?no)>0, REPLACE(?no,"[^0-9+]",""),?undef) AS ?n )
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?first)) AS ?song)
    BIND (xsd:decimal(strbefore(?l,":"))*60 + xsd:decimal(strafter(?l,":"))  AS ?
    lengthSeconds)
    BIND ( IF (BOUND(?lengthSeconds) && BOUND(?song), BNODE(),?undef) AS ?duration )
    BIND ( IF (BOUND(?lengthSeconds) && BOUND(?song), BNODE(),?undef) AS ?b1 )
    BIND ( IF (BOUND(?song) && BOUND(?album), BNODE(),?undef) AS ?tracklist )
    BIND ( IF (BOUND(?n) && strlen(?n)>0, ?n ,?undef) AS ?number )
    VALUES ?undef {UNDEF}
}
```

Listing A.14: SPARQL-Generate mapping for Cluster 4

```
<#Cluster4Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "cluster4.csv";
        rml:referenceFormulation ql:CSV
    ];
```

```
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{TitleID}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P31;
        rr:objectMap [
     rr:constant wd:Q105543609
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P361;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{ArticleEntity}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P7937;
        rr:objectMap [
            rr:constant wd:Q7366
        ]
    ].
<#Album> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "cluster4.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{ArticleEntity}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P31;
        rr:objectMap [
     rr:constant wd:Q482994
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate  wdt:P658;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{TitleID}"
        ]
    ];
rr:predicateObjectMap [
    rr:predicate p:P658;
    rr:objectMap [
        rr:termType rr:BlankNode ;
  rr:template "_:source{ArticleEntity}{TitleID}{no}"
    ]
].
<#Tracklist> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "cluster4.csv";
```

```
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode;
        rr:template "_:source{ArticleEntity}{TitleID}{no}"
    ];
    rr:predicateObjectMap [
        rr:predicate ps:P658;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{TitleID}"
  ]
    ];
    rr:predicateObjectMap [
        rr:predicate pq:P1545;
        rr:objectMap [
            rml:reference "no"
  ]
    ].
```

Listing A.15: CARML / SDM-RDFizer mapping for Cluster 4

```
<#Cluster4Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "../Input/cluster4.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{TitleID}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P31;
        rr:objectMap [
            rr:constant wd:Q105543609
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P361;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{ArticleEntity}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P7937;
        rr:objectMap [
     rr:constant wd:Q7366
        ]
    ].
<#Album> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "../Input/cluster4.csv";
        rml:referenceFormulation ql:CSV
    ];
```

```
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{ArticleEntity}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P31;
        rr:objectMap [
     rr:constant wd:Q482994
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate  wdt:P658;
        rr:objectMap [
     rr:template "http://www.wikidata.org/entity/{TitleID}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate p:P658;
        rr:objectMap [
            rr:parentTriplesMap <#Tracklist>;
            rr:joinCondition [
                rr:child "ArticleEntity";
                rr:parent "ArticleEntity";
      ];
        ]
].
<#Tracklist> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "cluster4.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode;
        rr:template "_:source{ArticleEntity}{TitleID}{no}"
    ];
    rr:predicateObjectMap [
        rr:predicate ps:P658;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{TitleID}"
  ]
    ];
    rr:predicateObjectMap [
        rr:predicate pq:P1545;
        rr:objectMap [
            rml:reference "no"
  ]
    ].
```

Listing A.16: RMLMapper / Chimera mapping for Cluster 4

```
<#Cluster4Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
```

```
        SELECT
        CASE
        WHEN TRIM(l) IS NOT NULL
        THEN (TRY_CAST(SPLIT_PART(l, ':', 1) AS DECIMAL) * 60 + TRY_CAST(SPLIT_PART(l,
    ':', 2) AS DECIMAL))
        ELSE NULL
        END AS 'lengthSeconds', first, ArticleEntity
        FROM (
            SELECT  first, ArticleEntity, REGEXP_REPLACE(l1, REGEXP_REPLACE(l1, '[0-9]+
    :[0-9]{2}', '', 'ig') , '', 'ig') as 'l', l1
            FROM (
                SELECT split_part(TitleID, ',' , 1) as first, ArticleEntity,
    REGEXP_REPLACE(length, '[^0-9:]', '', 'ig') as 'l1'
      FROM '../Input/cluster4.csv'
            ) b
        ) a
        """
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{first}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P31;
        rr:objectMap [
            rr:constant wd:Q105543609
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P361;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{ArticleEntity}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P7937;
        rr:objectMap [
            rr:constant wd:Q7366
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P2047;
        rr:objectMap [
            rml:reference "lengthSeconds";
            rr:datatype xsd:decimal
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate p:P2047;
        rr:objectMap [
            rr:termType rr:BlankNode ;
            rr:template "_:source{ArticleEntity}{first}{lengthSeconds}"
```

```
        ]
    ].

<#Duration> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
        SELECT
        CASE
        WHEN TRIM(l) IS NOT NULL
        THEN (TRY_CAST(SPLIT_PART(l, ':', 1) AS DECIMAL) * 60 + TRY_CAST(SPLIT_PART(l,
    ':', 2) AS DECIMAL))
        ELSE NULL
        END AS 'lengthSeconds', first, ArticleEntity
        FROM (
            SELECT  first, ArticleEntity, REGEXP_REPLACE(l1, REGEXP_REPLACE(l1, '[0-9]+
    :[0-9]{2}', '', 'ig') , '', 'ig') as 'l', l1
            FROM (
                SELECT split_part(TitleID, ',' , 1) as first, ArticleEntity,
    REGEXP_REPLACE(length, '[^0-9:]', '', 'ig') as 'l1'
                FROM '../Input/cluster4.csv'
            ) b
        ) a
  """
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode;
        rr:template "_:source{ArticleEntity}{first}{lengthSeconds}"
    ];
    rr:predicateObjectMap [
        rr:predicate ps:P2047;
            rr:objectMap [
                rml:reference "lengthSeconds";
                rr:datatype xsd:decimal
            ]
    ];
    rr:predicateObjectMap [
        rr:predicate psv:P2047;
        rr:objectMap [
            rr:termType rr:BlankNode ;
            rr:template "_:source{ArticleEntity}{first}{lengthSeconds}b1"
        ]
    ].

<#b1> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
        SELECT
        CASE
        WHEN TRIM(l) IS NOT NULL
        THEN (TRY_CAST(SPLIT_PART(l, ':', 1) AS DECIMAL) * 60 + TRY_CAST(SPLIT_PART(l,
    ':', 2) AS DECIMAL))
```

```
        ELSE NULL
        END AS 'lengthSeconds', first, ArticleEntity
        FROM (
            SELECT  first, ArticleEntity, REGEXP_REPLACE(l1, REGEXP_REPLACE(l1, '[0-9]+:[0-9]{2}', '', 'ig') , '', 'ig') as 'l', l1
            FROM (
                SELECT split_part(TitleID, ',' , 1) as first, ArticleEntity, REGEXP_REPLACE(length, '[^0-9:]', '', 'ig') as 'l1'
                FROM '../Input/cluster4.csv'
            ) b
        ) a
 """
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode;
        rr:template "_:source{ArticleEntity}{first}{lengthSeconds}b1"
    ];
    rr:predicateObjectMap [
        rr:predicate rdf:type;
        rr:objectMap [
            rr:constant wikibase:QuantityValue
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wikibase:quantityAmount;
        rr:objectMap [
            rml:reference "lengthSeconds";
            rr:datatype xsd:decimal
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wikibase:quantityUnit;
        rr:objectMap [
            rr:constant wd:Q11574
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wikibase:quantityNormalized;
        rr:objectMap [
            rr:parentTriplesMap <#b1>
        ]
    ].

<#Album> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
        SELECT REGEXP_REPLACE(no, '[^0-9]', '', 'ig') AS 'number', split_part(TitleID, ',' , 1) AS first, ArticleEntity
        FROM '../Input/cluster4.csv'
        """
    ];
```

```
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{ArticleEntity}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P31;
        rr:objectMap [
            rr:constant wd:Q482994
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate  wdt:P658;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{first}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate p:P658;
        rr:objectMap [
            rr:termType rr:BlankNode ;
            rr:template "_:source{ArticleEntity}{first}{number}"
        ]
    ].

<#Tracklist> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
        SELECT REGEXP_REPLACE(no, '[^0-9]', '', 'ig') AS 'number', split_part(TitleID, ',
    ' , 1) AS first, ArticleEntity
        FROM '../Input/cluster4.csv'
  """
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode;
        rr:template "_:source{ArticleEntity}{first}{number}"
    ];
    rr:predicateObjectMap [
        rr:predicate ps:P658;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{first}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate pq:P1545;
        rr:objectMap [
            rml:reference "number"
        ]
    ].
```

Listing A.17: Morph-KGC mapping for Cluster 4 using RML Views

A.5 Cluster 5

```
CONSTRUCT {
    ?player wdt:P413 ?position.
}
FROM <file:../Input/cluster5.csv>
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Player_3)) AS ?player)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Position_3)) AS ?position)
}
```

Listing A.18: Tarql mapping for Cluster 5

```
CONSTRUCT {
    ?player wdt:P413 ?position.
}
WHERE {
    SERVICE <x-sparql-anything:location=../Input/cluster5.csv,csv.headers=true> {
        fx:properties  fx:csv.null-string  "" .
        ?c  xyz:Player_3 ?Player_3;
            xyz:Position_3 ?Position_3
    }
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Player_3)) AS ?player)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Position_3)) AS ?position)
}
```

Listing A.19: SPARQL Anything mapping for Cluster 5

```
GENERATE {
    ?player wdt:P413 ?position.
}
ITERATOR iter:CSV(<../Input/cluster5.csv>, "Player_3", "Position_3") AS  ?Player_3
?Position_3
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Player_3)) AS ?player)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Position_3)) AS ?position)
}
```

Listing A.20: SPARQL-Generate mapping for Cluster 5

```
<#Cluster5Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "../Input/cluster5.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{Player_3}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P413;
        rr:objectMap [
        rr:template "http://www.wikidata.org/entity/{Position_3}"
        ]
    ].
```

Listing A.21: RML mapping for Cluster 5

A.6 Cluster 6

```
CONSTRUCT {
    ?team wdt:P1344 ?article.
}
FROM <file:../Input/cluster6.csv>
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?spancol_3)) AS ?team)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?ArticleEntity)) AS ?article)
}
```

Listing A.22: Tarql mapping for Cluster 6

```
CONSTRUCT {
    ?team wdt:P1344 ?article.
}
WHERE {
    SERVICE <x-sparql-anything:location=../Input/cluster6.csv,csv.headers=true> {
        fx:properties  fx:csv.null-string  "" .
        ?c      xyz:spancol_3 ?spancol_3;
          xyz:ArticleEntity ?ArticleEntity
    }
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?spancol_3)) AS ?team)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?ArticleEntity)) AS ?article)
}
```

Listing A.23: SPARQL Anything mapping for Cluster 6

```
GENERATE {
    ?team wdt:P1344 ?article.
}
ITERATOR iter:CSV(<../Input/cluster6.csv>, "spancol_3", "ArticleEntity") AS  ?spancol_3
?ArticleEntity
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?spancol_3)) AS ?team)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?ArticleEntity)) AS ?article)
}
```

Listing A.24: SPARQL Generate mapping for Cluster 6

```
<#Cluster6Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "../Input/cluster6.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{spancol_3}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P1344;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{ArticleEntity}"
        ]
    ].
```

Listing A.25: RML mapping for Cluster 6

A.7 Cluster 7

We have applied the same mappings as detailed in Section 4.5 to extract triples following the pattern illustrated in Figure 4.7 over tables from Cluster 7. The results are presented in Table 5.14. As explained in Section 4.5, while generating the blank nodes in CARML and SDM-RDFizer, some variables of the templates are missing, resulting in fewer extracted triples. As for RMLMapper, there are no results because the processing reached the timeout set in one hour.

```
CONSTRUCT {
    ?title wdt:P674 ?role.
    ?role  wdt:P1441 ?title.
    ?title p:P161 _:cast.
    _:cast ps:P161 ?actor;
           pq:P453 ?role.
    ?actor wdt:P106 wd:Q33999.
}
FROM <file:../Input/cluster7.csv>
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Title_3)) AS ?title)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Role_3)) AS ?role)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?ArticleEntity)) AS ?actor)
}
```

Listing A.26: Tarql mapping for Cluster 7

```
CONSTRUCT {
    ?title wdt:P674 ?role.
    ?role  wdt:P1441 ?title.
    ?title p:P161 _:cast.
    _:cast ps:P161 ?actor;
           pq:P453 ?role.
    ?actor wdt:P106 wd:Q33999.
}
WHERE {
    SERVICE <x-sparql-anything:location=../Input/cluster7.csv,csv.headers=true> {
        fx:properties  fx:csv.null-string  "" .
        ?c        xyz:ArticleEntity ?ArticleEntity.
        OPTIONAL{?c xyz:Title_3 ?Title}.
        OPTIONAL{?c xyz:Role_3 ?Role}.
    }
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Title)) AS ?title)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Role)) AS ?role)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?ArticleEntity)) AS ?actor)
}
```

Listing A.27: SPARQL Anything mapping for Cluster 7

```
GENERATE {
    ?title wdt:P674 ?role.
    ?role  wdt:P1441 ?title.
    ?title p:P161 _:cast.
    _:cast ps:P161 ?actor;
           pq:P453 ?role.
    ?actor wdt:P106 wd:Q33999.
}
ITERATOR iter:CSV(<../Input/cluster7.csv>, "ArticleEntity", "Titl_3","Role_3") AS ?
    ArticleEntity ?Title ?Role
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Title)) AS ?title)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Role)) AS ?role)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?ArticleEntity)) AS ?actor)
}
```

Listing A.28: SPARQL-Generate mapping for Cluster 7

```
<#Cluster7Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source  "../Input/cluster7.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{Title}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P674;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Role}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate p:P161;
        rr:objectMap [
            rr:parentTriplesMap <#Cast>;
            rr:joinCondition [
                rr:child "Title";
                rr:parent "Title";
            ];
        ]
    ].
<#Cast> a rr:TriplesMap;
    rml:logicalSource [
        rml:source  "../Input/cluster7.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode;
    ];
    rr:predicateObjectMap [
        rr:predicate ps:P161;
```

```
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{ArticleEntity}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate  pq:P453;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Role}"
        ]
    ].
<#Role> a rr:TriplesMap;
    rml:logicalSource[
        rml:source  "../Input/cluster7.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{Role}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P1441;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Title}"
        ]
    ].
<#Actor> a rr:TriplesMap;
    rml:logicalSource [
        rml:source  "../Input/cluster7.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{ArticleEntity}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P106;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/Q33999"
        ]
    ].
```

Listing A.29: RML mapping for Cluster 7

```
<#Cluster7Mapping> a rr:TriplesMap;
...
<#Cast> a rr:TriplesMap;
  rml:logicalSource ... ;
    rr:subjectMap [
            rr:template "{ArticleEntity}_{Title}_{Role}";
            rr:termType rr:BlankNode
    ];
    rr:predicateObjectMap [
        rr:predicate ps:P161;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{ArticleEntity}"
        ]
  ];
    rr:predicateObjectMap [
        rr:predicate  pq:P453;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Role}"
        ]
  ].
<#Role> a rr:TriplesMap;
...
<#Actor> a rr:TriplesMap;
...
```

Listing A.30: CARML / SDM-RDFizer / Morph-KGC mapping for Cluster 7

A.8 Cluster 8

```
CONSTRUCT {
    ?ship wdt:P8047 ?country;
          wdt:P31 wd:Q852190.
}
FROM <file:../Input/cluster8.csv>
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Ship_3)) AS ?ship)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Country_3)) AS ?country)
}
```

Listing A.31: Tarql mapping for Cluster 8

```
CONSTRUCT {
    ?ship wdt:P8047 ?country;
          wdt:P31 wd:Q852190.
}
WHERE {
    SERVICE <x-sparql-anything:location=../Input/cluster8.csv,csv.headers=true> {
        fx:properties  fx:csv.null-string  "" .
        ?c  xyz:Ship_3 ?Ship_3;
            xyz:Country_3 ?Country_3
    }
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Ship_3)) AS ?ship)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Country_3)) AS ?country)
}
```

Listing A.32: SPARQL Anything mapping for Cluster 8

```
GENERATE {
    ?ship   wdt:P8047 ?country;
            wdt:P31 wd:Q852190.
}
ITERATOR iter:CSV(<../Input/cluster8.csv>, "Ship_3", "Country_3") AS ?Ship_3 ?Country_3

WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Ship_3)) AS ?ship)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Country_3)) AS ?country)
}
```

Listing A.33: SPARQL Generate mapping for Cluster 8

```
<#Cluster8Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "cluster8.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{Ship_3}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P8047;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Country_3}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P31;
        rr:objectMap [
            rr:constant wd:Q852190
        ]
    ].
```

Listing A.34: RML mapping for Cluster 8

A.9 Cluster 9

This cluster contains similar information to A.4, presenting details about songs and their albums.

```
CONSTRUCT {
    ?song   wdt:P676 ?writer;
            wdt:P31 wd:Q105543609;
            wdt:P361 ?album;
            wdt:P7937 wd:Q7366;
            wdt:P2047 ?lengthSeconds;
            p:P2047 ?duration.

    ?duration   ps:P2047 ?lengthSeconds;
                psv:P2047 ?b1.

    ?b1 a   wikibase:QuantityValue;
        wikibase:quantityAmount ?lengthSeconds;
        wikibase:quantityUnit wd:Q11574;
        wikibase:quantityNormalized ?b1.

    ?album  wdt:P31 wd:Q482994;
            wdt:P658 ?song;
            p:P658 ?tracklist.

    ?tracklist  ps:P658 ?song;
                pq:P1545 ?number.

    ?writer wdt:P106 wd:Q753110.
}
FROM <file:../Input/cluster9.csv>
WHERE {
    BIND ( IF (BOUND(?ArticleEntity) && strlen(?ArticleEntity)>0, URI(CONCAT('http://www.
    wikidata.org/entity/', ?ArticleEntity)),?undef) AS ?album )
    BIND ( IF (BOUND(?TitleID) && strlen(?TitleID)>0,  REPLACE(replace(?TitleID, " ", "")
    ,",Q[0-9]*",""),?undef) AS ?first )
    BIND ( IF (BOUND(?length) && strlen(?length)>0, REPLACE(REPLACE(?length,"[^0-9:]","")
    ,"(?<=[0-9]+:[0-9]{2}).*",""),?undef) AS ?l )
    BIND ( IF (BOUND(?no) && strlen(?no)>0, REPLACE(?no,"[^0-9+]",""),?undef) AS ?n )
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?first)) AS ?song)
    BIND (xsd:decimal(strbefore(?l,":"))*60 + xsd:decimal(strafter(?l,":"))  AS ?
    lengthSeconds)
    BIND ( IF (BOUND(?lengthSeconds) && BOUND(?song), BNODE(),?undef) AS ?duration )
    BIND ( IF (BOUND(?lengthSeconds) && BOUND(?song), BNODE(),?undef) AS ?b1 )
    BIND ( IF (BOUND(?song) && BOUND(?album), BNODE(),?undef) AS ?tracklist )
    BIND ( IF (BOUND(?n) && strlen(?n)>0, ?n ,?undef) AS ?number )
    OPTIONAL {?WriterSplit apf:strSplit (?WriterID ",") }
    BIND ( IF (BOUND(?WriterSplit) && strlen(?WriterSplit)>0, URI(CONCAT('http://www.
    wikidata.org/entity/', ?WriterSplit)),?undef) AS ?writer )
    VALUES ?undef {UNDEF}
}
```

Listing A.35: Tarql mapping for Cluster 9

```
CONSTRUCT {
    ?song   wdt:P676 ?writer;
            wdt:P31 wd:Q105543609;
            wdt:P361 ?album;
            wdt:P7937 wd:Q7366;
            wdt:P2047 ?lengthSeconds;
            p:P2047 ?duration.

    ?duration   ps:P2047 ?lengthSeconds;
                psv:P2047 ?b1.

    ?b1 a   wikibase:QuantityValue;
        wikibase:quantityAmount ?lengthSeconds;
        wikibase:quantityUnit wd:Q11574;
        wikibase:quantityNormalized ?b1.

    ?album  wdt:P31 wd:Q482994;
            wdt:P658 ?song;
            p:P658 ?tracklist.

    ?tracklist  ps:P658 ?song;
                pq:P1545 ?number.

    ?writer wdt:P106 wd:Q753110.
}
WHERE {
    SERVICE <x-sparql-anything:location=../Input/cluster9.csv,csv.headers=true> {
        OPTIONAL{ ?c xyz:ArticleEntity ?ArticleEntity .}
        OPTIONAL{ ?c xyz:no ?no .}
        OPTIONAL{ ?c xyz:TitleID ?TitleID .}
        OPTIONAL{ ?c xyz:length ?length .}
        OPTIONAL{ ?c  xyz:WriterID ?WriterID}.
    }
    BIND ( IF (BOUND(?ArticleEntity) && strlen(?ArticleEntity)>0, URI(CONCAT('http://www.
    wikidata.org/entity/', ?ArticleEntity)),?undef) AS ?album )
    BIND ( IF (BOUND(?TitleID) && strlen(?TitleID)>0,  REPLACE(replace(?TitleID, " ", "")
    ,",Q[0-9]*",""),?undef) AS ?first )
    BIND ( IF (BOUND(?length) && strlen(?length)>0, REPLACE(REPLACE(?length,"[^0-9:]","")
    ,"(?<=[0-9]+:[0-9]{2}).*",""),?undef) AS ?l )
    BIND ( IF (BOUND(?no) && strlen(?no)>0, REPLACE(?no,"[^0-9+]",""),?undef) AS ?n )
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?first)) AS ?song)
    BIND (xsd:decimal(strbefore(?l,":"))*60 + xsd:decimal(strafter(?l,":"))  AS ?
    lengthSeconds)
    BIND ( IF (BOUND(?lengthSeconds) && BOUND(?song), BNODE(),?undef) AS ?duration )
    BIND ( IF (BOUND(?lengthSeconds) && BOUND(?song), BNODE(),?undef) AS ?b1 )
    BIND ( IF (BOUND(?song) && BOUND(?album), BNODE(),?undef) AS ?tracklist )
    BIND ( IF (BOUND(?n) && strlen(?n)>0, ?n ,?undef) AS ?number )
    OPTIONAL {?WriterSplit apf:strSplit (?WriterID ",") }
    BIND ( IF (BOUND(?WriterSplit) && strlen(?WriterSplit)>0, URI(CONCAT('http://www.
    wikidata.org/entity/', ?WriterSplit)),?undef) AS ?writer )
    VALUES ?undef {UNDEF}
}
```

Listing A.36: SPARQL Anything mapping for Cluster 9

```
GENERATE {
    ?song   wdt:P676 ?writer;
            wdt:P31 wd:Q105543609;
            wdt:P361 ?album;
            wdt:P7937 wd:Q7366;
            wdt:P2047 ?lengthSeconds;
            p:P2047 ?duration.

    ?duration   ps:P2047 ?lengthSeconds;
                psv:P2047 ?b1.

    ?b1 a   wikibase:QuantityValue;
        wikibase:quantityAmount ?lengthSeconds;
        wikibase:quantityUnit wd:Q11574;
        wikibase:quantityNormalized ?b1.

    ?album  wdt:P31 wd:Q482994;
            wdt:P658 ?song;
            p:P658 ?tracklist.

    ?tracklist  ps:P658 ?song;
                pq:P1545 ?number.

    ?writer wdt:P106 wd:Q753110.
}
ITERATOR iter:CSV(<../Input/cluster9.csv>, "TitleID", "WriterID","ArticleEntity","length"
    ,"no") AS ?TitleID ?WriterID ?ArticleEntity ?length ?no
WHERE {
    BIND ( IF (BOUND(?ArticleEntity) && strlen(?ArticleEntity)>0, URI(CONCAT('http://www.
    wikidata.org/entity/', ?ArticleEntity)),?undef) AS ?album )
    BIND ( IF (BOUND(?TitleID) && strlen(?TitleID)>0,  REPLACE(replace(?TitleID, " ", "")
    ,",Q[0-9]*",""),?undef) AS ?first )
    BIND ( IF (BOUND(?length) && strlen(?length)>0, REPLACE(REPLACE(?length,"[^0-9:]","")
    ,"(?<=[0-9]+:[0-9]{2}).*",""),?undef) AS ?l )
    BIND ( IF (BOUND(?no) && strlen(?no)>0, REPLACE(?no,"[^0-9+]",""),?undef) AS ?n )
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?first)) AS ?song)
    BIND (xsd:decimal(strbefore(?l,":"))*60 + xsd:decimal(strafter(?l,":"))  AS ?
    lengthSeconds)
    BIND ( IF (BOUND(?lengthSeconds) && BOUND(?song), BNODE(),?undef) AS ?duration )
    BIND ( IF (BOUND(?lengthSeconds) && BOUND(?song), BNODE(),?undef) AS ?b1 )
    BIND ( IF (BOUND(?song) && BOUND(?album), BNODE(),?undef) AS ?tracklist )
    BIND ( IF (BOUND(?n) && strlen(?n)>0, ?n ,?undef) AS ?number )
    OPTIONAL {?WriterSplit apf:strSplit (?WriterID ",") }
    BIND ( IF (BOUND(?WriterSplit) && strlen(?WriterSplit)>0, URI(CONCAT('http://www.
    wikidata.org/entity/', ?WriterSplit)),?undef) AS ?writer )
    VALUES ?undef {UNDEF}
}
```

Listing A.37: SPARQL-Generate mapping for Cluster 9

```
<#Cluster9Mapping> a rr:TriplesMap;
  rml:logicalSource [
    rml:source "../Input/cluster9.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{TitleID}";
    ];

    rr:predicateObjectMap [
        rr:predicate wdt:P676;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{WriterID}"
        ]
      ];
    rr:predicateObjectMap [
        rr:predicate wdt:P31;
        rr:objectMap [
            rr:constant wd:Q105543609
        ]
      ];
    rr:predicateObjectMap [
        rr:predicate wdt:P361;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{ArticleEntity}"
        ]
      ];
    rr:predicateObjectMap [
        rr:predicate wdt:P7937;
        rr:objectMap [
            rr:constant wd:Q7366
        ]
      ].

<#Writer> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "../Input/cluster9.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{WriterID}";
    ];

    rr:predicateObjectMap [
        rr:predicate wdt:P106;
      rr:objectMap [
            rr:constant wd:Q753110
        ]
      ].
```

Listing A.38: RML mapping for Cluster 9

```
<#Cluster9Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
        SELECT
        CASE
        WHEN TRIM(l) IS NOT NULL
        THEN (TRY_CAST(SPLIT_PART(l, ':', 1) AS DECIMAL) * 60 + TRY_CAST(SPLIT_PART(l,
    ':', 2) AS DECIMAL))
        ELSE NULL
        END AS 'lengthSeconds', first, ArticleEntity, WriterIDtrim
        FROM (
            SELECT  WriterIDtrim, first, ArticleEntity, REGEXP_REPLACE(l1, REGEXP_REPLACE
    (l1, '[0-9]+:[0-9]{2}', '', 'ig') , '', 'ig') as 'l', l1
            FROM (
                SELECT split_part(TitleID, ',' , 1) as first, ArticleEntity,
    REGEXP_REPLACE(WriterID, '[[:space:]]*','', 'ig') AS WriterIDtrim, REGEXP_REPLACE(
    length, '[^0-9:]', '', 'ig') as 'l1'
      FROM '../Input/cluster9.csv'
            ) b
        ) a
        """
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{first}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P31;
        rr:objectMap [
            rr:constant wd:Q105543609
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P361;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{ArticleEntity}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P7937;
        rr:objectMap [
            rr:constant wd:Q7366
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P2047;
        rr:objectMap [
            rml:reference "lengthSeconds";
            rr:datatype xsd:decimal
        ]
    ];
    rr:predicateObjectMap [
```

```
        rr:predicate p:P2047;
        rr:objectMap [
            rr:termType rr:BlankNode ;
            rr:template "_:source{ArticleEntity}{first}{lengthSeconds}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P676;
        rr:objectMap [
            rml:functionExecution <#Execution> ;
            rml:termType rml:IRI
        ]
    ].

<#Duration> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
        SELECT
        CASE
        WHEN TRIM(l) IS NOT NULL
        THEN (TRY_CAST(SPLIT_PART(l, ':', 1) AS DECIMAL) * 60 + TRY_CAST(SPLIT_PART(l,
    ':', 2) AS DECIMAL))
        ELSE NULL
        END AS 'lengthSeconds', first, ArticleEntity
        FROM (
            SELECT  first, ArticleEntity, REGEXP_REPLACE(l1, REGEXP_REPLACE(l1, '[0-9]+
    :[0-9]{2}', '', 'ig') , '', 'ig') as 'l', l1
            FROM (
                SELECT split_part(TitleID, ',' , 1) as first, ArticleEntity,
     REGEXP_REPLACE(length, '[^0-9:]', '', 'ig') as 'l1'
                FROM '../Input/cluster9.csv'
            ) b
        ) a
  """
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode;
        rr:template "_:source{ArticleEntity}{first}{lengthSeconds}"
    ];
    rr:predicateObjectMap [
        rr:predicate ps:P2047;
            rr:objectMap [
                rml:reference "lengthSeconds";
                rr:datatype xsd:decimal
            ]
    ];
    rr:predicateObjectMap [
        rr:predicate psv:P2047;
        rr:objectMap [
            rr:termType rr:BlankNode ;
            rr:template "_:source{ArticleEntity}{first}{lengthSeconds}b1"
```

```
        ]
    ].

<#b1> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
        SELECT
        CASE
        WHEN TRIM(l) IS NOT NULL
        THEN (TRY_CAST(SPLIT_PART(l, ':', 1) AS DECIMAL) * 60 + TRY_CAST(SPLIT_PART(l,
    ':', 2) AS DECIMAL))
        ELSE NULL
        END AS 'lengthSeconds', first, ArticleEntity
        FROM (
            SELECT  first, ArticleEntity, REGEXP_REPLACE(l1, REGEXP_REPLACE(l1, '[0-9]+
    :[0-9]{2}', '', 'ig') , '', 'ig') as 'l', l1
            FROM (
                SELECT split_part(TitleID, ',' , 1) as first, ArticleEntity,
    REGEXP_REPLACE(length, '[^0-9:]', '', 'ig') as 'l1'
                FROM '../Input/cluster9.csv'
            ) b
        ) a
  """
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode;
        rr:template "_:source{ArticleEntity}{first}{lengthSeconds}b1"
    ];
    rr:predicateObjectMap [
        rr:predicate rdf:type;
        rr:objectMap [
            rr:constant wikibase:QuantityValue
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wikibase:quantityAmount;
        rr:objectMap [
            rml:reference "lengthSeconds";
            rr:datatype xsd:decimal
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wikibase:quantityUnit;
        rr:objectMap [
            rr:constant wd:Q11574
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate wikibase:quantityNormalized;
        rr:objectMap [
            rr:parentTriplesMap <#b1>
```

```
        ]
    ].

<#Album> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
        SELECT REGEXP_REPLACE(no, '[^0-9]', '', 'ig') AS 'number', split_part(TitleID, ',
    ' , 1) AS first, ArticleEntity
        FROM '../Input/cluster9.csv'
        """
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{ArticleEntity}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P31;
        rr:objectMap [
            rr:constant wd:Q482994
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate  wdt:P658;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{first}"
        ]
    ];
    rr:predicateObjectMap [
        rr:predicate p:P658;
        rr:objectMap [
            rr:termType rr:BlankNode ;
            rr:template "_:source{ArticleEntity}{first}{number}"
        ]
    ].

<#Tracklist> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
        SELECT REGEXP_REPLACE(no, '[^0-9]', '', 'ig') AS 'number', split_part(TitleID, ',
    ' , 1) AS first, ArticleEntity
        FROM '../Input/cluster9.csv'
  """
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode;
        rr:template "_:source{ArticleEntity}{first}{number}"
    ];
    rr:predicateObjectMap [
        rr:predicate ps:P658;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{first}"
        ]
```

```
    ];
    rr:predicateObjectMap [
        rr:predicate pq:P1545;
        rr:objectMap [
            rml:reference "number"
        ]
    ].

<Writer>
    rml:logicalSource [
        rml:query """
        SELECT  REGEXP_REPLACE(WriterID, '[[:space:]]*','', 'ig')  AS WriterIDtrim
        FROM '../Input/cluster9.csv'
        """
    ];
    rml:subjectMap [
        rml:functionExecution <#Execution> ;
        rml:termType rml:IRI
    ];
    rml:predicateObjectMap [
        rml:predicate wdt:P106;
        rml:objectMap [
            rr:constant wd:Q753110
        ]
    ] .

<#Execution>
    rml:function morph-kgc:string_split_explode ;
    rml:input [
        rml:parameter grel:valueParam ;
        rml:inputValueMap [
            rml:functionExecution <#Execution2> ;
            rml:return grel:stringOut
        ]
    ],
    [
        rml:parameter grel:param_string_sep ;
        rml:inputValue ","
    ] .


<#Execution2> a rml:Execution ;
    rml:function grel:string_replace ;
    rml:input [
        a rml:Input ;
        rml:parameter grel:valueParam ;
        rml:inputValueMap [
            rml:template "http://www.wikidata.org/entity/{WriterIDtrim}"
        ]
    ],
    [
```

```
        a rml:Input ;
        rml:parameter grel:param_find ;
        rml:inputValue ","
    ],
    [
        a rml:Input ;
        rml:parameter grel:param_replace  ;
        rml:inputValue ",http://www.wikidata.org/entity/"
    ].
```

Listing A.39: Morph-KGC mapping for Cluster 9 using RML Views

A.10 Cluster 10

```
CONSTRUCT {
    ?player wdt:P413 ?position.
}
FROM <file:../Input/cluster10.csv>
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Player_3)) AS ?player)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Position_3)) AS ?position)
}
```

Listing A.40: Tarql mapping for Cluster 10

```
CONSTRUCT {
    ?player wdt:P413 ?position.
}
WHERE {
    SERVICE <x-sparql-anything:location=../Input/cluster10.csv,csv.headers=true> {
        fx:properties  fx:csv.null-string  "" .
        ?c  xyz:Player_3 ?Player_3;
            xyz:Position_3 ?Position_3
    }
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Player_3)) AS ?player)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Position_3)) AS ?position)
}
```

Listing A.41: SPARQL Anything mapping for Cluster 10

```
GENERATE {
    ?player wdt:P413 ?position.
}
ITERATOR iter:CSV(<../Input/cluster10.csv>, "Player_3", "Position_3") AS ?Player_3 ?
    Position_3
WHERE {
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Player_3)) AS ?player)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?Position_3)) AS ?position)
}
```

Listing A.42: SPARQL Generate mapping for Cluster 10

```
<#CLuster10Mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "../Input/cluster10.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{Player_3}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P413;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{Position_3}"
        ]
    ].
```

Listing A.43: RML mapping for Cluster 10

A.11 Cluster 90

```
<Cluster90Mapping>
rml:logicalSource [
    rml:query """
        SELECT a.game AS current, b.nextGame AS follower
        FROM (
            SELECT split_part(gameID,'''',1) AS game, row_number() over () AS "ID",
    TableID
            FROM '../Input/cluster90.csv'
            ORDER BY "ID"
        ) a
        JOIN (
        SELECT c.number AS "nextGame", row_number() over () AS ID2, c.TableID AS TableID2
        FROM (
            SELECT split_part(gameID, '''' , 1) as number, row_number() over () AS ID,
    TableID
            FROM '../Input/cluster90.csv'
            ORDER BY ID
            OFFSET 1 ROW
        ) c
        ORDER BY ID2
        ) b
        ON ((a.ID = b.ID2) AND (a.TableID = b.TableID2))
    """ ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{current}" ];
        rr:predicateObjectMap [
            rr:predicate wdt:P156;
            rr:objectMap [
                rr:template "http://www.wikidata.org/entity/{follower}"
            ]
        ].
```

Listing A.44: Morph-KGC mapping for Cluster 90

```
CONSTRUCT {
    ?games wdt:P156 ?games_next.
}
FROM <file:../Input/cluster90preprocessed.csv>
WHERE {
    BIND(REPLACE(?gameID1, " ", "") as ?gameID1trim)
    BIND(REPLACE(?gameID2, " ", "") as ?gameID2trim)
    BIND(REPLACE(?gameID1trim,",Q[0-9]*","") AS ?firstGameID1)
    BIND(REPLACE(?gameID2trim,",Q[0-9]*","") AS ?firstGameID2)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?firstGameID1)) AS ?games)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?firstGameID2)) AS ?games_next)
}
```

Listing A.45: Tarql mapping for Cluster 90 with preprocess of input data

```
GENERATE {
    ?games wdt:P156 ?games_next.
}
ITERATOR iter:CSV(<../Input/cluster90preprocessed.csv>, "gameID1", "gameID2") AS ?gameID1
     ?gameID2
WHERE {
    BIND(REPLACE(?gameID1, " ", "") as ?gameID1trim)
    BIND(REPLACE(?gameID2, " ", "") as ?gameID2trim)
    BIND(REPLACE(?gameID1trim,",Q[0-9]*","") AS ?firstGameID1)
    BIND(REPLACE(?gameID2trim,",Q[0-9]*","") AS ?firstGameID2)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?firstGameID1)) AS ?games)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?firstGameID2)) AS ?games_next)
}
```

Listing A.46: SPARQL-Generate mapping for Cluster 90 with preprocess of input data

```
CONSTRUCT {
    ?games wdt:P156 ?games_next.
}
WHERE { SERVICE <x-sparql-anything:location=../Input/cluster90preprocessed.csv,csv.
    headers=true> {
    fx:properties  fx:csv.null-string  "" .
                    ?c        xyz:gameID1 ?gameID1;
                              xyz:gameID2 ?gameID2.
}
WHERE {
    BIND(REPLACE(?gameID1, " ", "") as ?gameID1trim)
    BIND(REPLACE(?gameID2, " ", "") as ?gameID2trim)
    BIND(REPLACE(?gameID1trim,",Q[0-9]*","") AS ?firstGameID1)
    BIND(REPLACE(?gameID2trim,",Q[0-9]*","") AS ?firstGameID2)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?firstGameID1)) AS ?games)
    BIND (URI(CONCAT('http://www.wikidata.org/entity/', ?firstGameID2)) AS ?games_next)
}
```

Listing A.47: SPARQL Anything mapping for Cluster 90 with preprocess of input data

```
<#Cluster90mappingPreprocess> a rr:TriplesMap;
    rml:logicalSource [
        rml:query """
        SELECT split_part(gameID1, ',' , 1) as current, split_part(gameID2, ',' , 1) as
    follower
        FROM '../Input/cluster90preprocessed.csv'
        """
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{current}" ];
        rr:predicateObjectMap [
            rr:predicate wdt:P156;
            rr:objectMap [
                rr:template "http://www.wikidata.org/entity/{follower}"
            ]
        ].
```

Listing A.48: Morph-KGC mapping for Cluster 90 with preprocess of input data

```
<#Cluster90mapping> a rr:TriplesMap;
    rml:logicalSource [
        rml:source "../Input/cluster90preprocessed.csv";
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://www.wikidata.org/entity/{GameID1}";
    ];
    rr:predicateObjectMap [
        rr:predicate wdt:P156;
        rr:objectMap [
            rr:template "http://www.wikidata.org/entity/{GameID2}"
        ]
    ].
```

Listing A.49: Partial RML mapping for Cluster 90 with preprocess of input data