Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik

Normalization Techniques For Improving The Performance Of Knowledge Graph Creation Pipelines

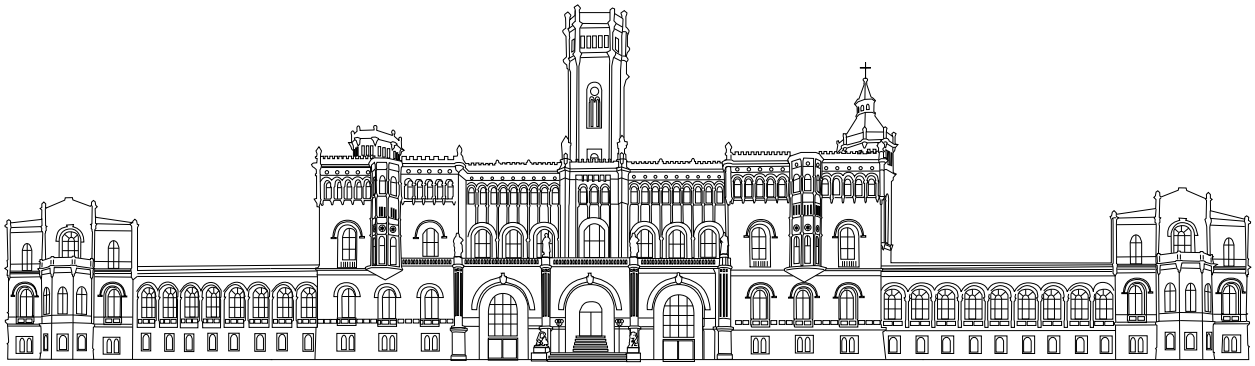
A thesis submitted in fulfillment of the requirements for the degree of
Master of Science in Internet Technologies and Information Systems (ITIS)

BY

Mohammad Torabinejad
Matriculation number: 3203980
E-mail: mohammad.torabinejad@stud.uni-hannover.de

First evaluator: Prof. Dr. Sören Auer
Second evaluator: Prof. (Univ. Simon Bolivar) Dr. Maria-Esther Vidal
First Supervisor: Prof. (Univ. Simon Bolivar) Dr. Maria-Esther Vidal
Second supervisor: Samaneh Jozashoori

September 8, 2020



Declaration of Authorship

I, Mohammad Torabinejad, declare that this thesis titled, 'Normalization techniques for Improving the performance of knowledge graph creation pipelines' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Mohammad Torabinejad

Signature: _____

Date: _____

You are a treasure, if the gems are your aim.
No more than a grain, if a loaf is your claim!
Recall this secret, when you play this game
Whatever you pursued- is what you became!

— Rumi

Acknowledgements

First of all, I would like to appreciate Prof. (Univ. Simón Bolívar) Dr. Maria-Esther Vidal for her wholeheartedly support and patient guidance. Without her enthusiastic encouragement and useful critiques during this research work, reaching my thesis's precious destination would not be possible. Besides, I would like to thank Prof. Dr. Maria-Esther Vidal for her extraordinary efforts during the tough time of the pandemic and for her invaluable assistance in keeping my progress on schedule during this time. I would like to be thankful to Prof. Dr. Auer for providing me the opportunity to work on my master thesis in the TIB research group and also for accepting the important role of hosting and evaluating my work.

I would also like to thank Samaneh Jozashoori for her help and support during my thesis and to offer me the opportunity to discuss hence reassessing my work. I would also like to extend my gratitude to the TIB and the research group of Scientific Data Management (SDM) to offer me the resources to develop and run different sections of this thesis. I want to thank every member of the SDM group for giving me the honour to be part of this family.

I want to be thankful to my friend Javad Kazemi for being patient and supportive during this challenging time. A special thanks to the lovely and graceful couple, Somayeh Zoghi and Mohammad Gholami, for their intimacy and companionship. I want to extend my special thanks to my friends Dr. Hadi Razaghi and Qazi Asim Ijaz Ahmad, for their pure motivation and support for academic suggestions and their invaluable fellowships during my whole study.

Finally, and most importantly, I would like to express my deep love and gratitude to my family, especially my parents, who have supported me. They are the most precious assets of my life for their absolute heartening and inspiring morning words, which made not only my day great but also my whole abroad life energetic. Therefore, I would like to dedicate my most valuable achievement in my life, especially for them.

Mohammad Torabinejad

Abstract

With the rapid growth of data within the web, demands on discovering information within data and consecutively exploiting knowledge graphs rise much more than we think it does. Data integration systems can be of great help to meet this precious demand in that they offer transformation of data from various sources and with different volumes. To this end, a data integration system takes advantage of utilizing mapping rules– specified in a language like RML – to integrate data collected from various data sources into a knowledge graph. However, large data sources may suffer from various data quality issues, being redundant one of them. Regarding this, the Semantic Web community contributes to Knowledge Engineering with techniques to create a knowledge graph efficiently. The thesis reported in this document tackles creating knowledge graphs in the presence of data sources with redundant data, and a novel normalization theory is proposed to solve this problem. This theory covers not only the characteristics of the data sources but also mapping rules used to integrate the data sources into a knowledge graph. Based on this, three normal forms are proposed and an algorithm for transforming mapping rules and data sources into these normal forms. The proposed approach's performance is evaluated in different testbeds composed of real-world data and synthetic data. The observed results suggest that the proposed techniques can dramatically reduce the execution time of knowledge graph creation. Therefore, this thesis's normalization theory contributes to the repertoire of tools that facilitate the creation of knowledge graphs at scale.

Keywords: Normalization . Mapping rules . Knowledge graph creation . Data integration system . Database

Indice

Elenco delle figure

Elenco delle tabelle

Listings

Acronimi

1NF first Normal Form
2NF second Normal Form
3NF third Normal Form

BCNF Boyce-Codd Normal Form

DBMS Database Management System

FD functional dependency

GAV global-as-view

LAV local-as-view

RDF Resource Description Framework
RML RDF Mapping Language

Capitolo 1

Introduction

The tremendous amount of web data and its exponential growth demands for scalable technologies to generate actionable knowledge. Despite knowledge graphs that can naturally represent heterogeneous data, techniques for scaling up to data variety, volume, and velocity are still required. Data integration systems have been responsible for transforming these data into knowledge graphs, e.g., RDF knowledge graph, with mapping languages[1]. Moreover, with the evolution of source-independent mapping languages such as RDF Mapping Language (RML), data integration has been facilitated to a high degree. Although such mapping languages offer the advantage of integrating heterogeneous data and language extensibility, large data volumes are still challenging when the transformation task of data comes to play. Albeit extensive size data, it will be shown that a considerable part of it contains redundant data. This chapter's content is devoted to an example which is the motivation for the work of this thesis. In addition, our contribution to the community of semantic web is presented in the last section.

1.1 Motivating Example

To motivate this thesis's work, consider a table with 1M tuples of genomic mutations described in terms of five attributes and having a low amount of distinct values concerning most of these attributes. This means that many of the data items within the data source are repeating for some small number of attributes, hence a significant number of redundancies. Now consider the data integration system, in the middle of a knowledge graph creation process, presented in figure 1.1, which transforms data within a data source with the help of mapping rules to an RDF graph[2] and later to a unified global schema. The data mentioned above, the table, and a mapping rule defined over that data are given input to this system. As shown in the figure, most of the low cardinality domains are involved in this mapping rule. Thus a significant portion of redundant data is subject to transformation. As an illustration, SDM-RDFizer[3], a mapping rule interpreter for RML[1, 4], is used to semantify the data according to the mapping rule in two rounds, one with duplicate elimination functionality and other without. As a result, there is almost 75% reduction in the size of the intermediate results and the size of the data source itself. Therefore, it is perceived that the RDF graph resulting from the integration process over the data above leads to duplicated RDF triples, as depicted in figure 1.1.

With further investigations, it can be realized that the existing redundancies are originating from the concept of functional dependencies in relational databases. For instance, it is clear in the data shown in 1.1 that $Mutation\_somatic\_status$ is functionally implied by $MUTATION\_ID$ hence a former's value can be repeated as many as all the latter's unique values. In contrast, it is

$$\{\text{MUTATION\_ID}\} \rightarrow \{\text{Gene\_CDS\_length}\}$$
$$\{\text{MUTATION\_ID}\} \rightarrow \{\text{Mutation\_Description}\}$$
$$\{\text{Mutation\_Description}\} \rightarrow \{\text{Mutation\_somatic\_status}\}$$

Listing 1.1 – List of functional dependencies holding in genomic data table depicted in data integration process of figure 1.1

not true in the opposite direction, and each $MUTATION\_ID$ is always about the same value, $Mutation\_somatic\_status$. However, these redundancies are escalated when redundant functional dependencies come to play. These redundant functional dependencies are implied logically from the current set of dependencies using Armstrong's axioms[5]. To exemplify, combination of $SAMPLE_ID$ and $MUTATION\_ID$ functionally determines $MUTATION\_ID$ and due to large number of unique values of $SAMPLE_ID$, $MUTATION\_ID$ repeats numerous times hence values of $Mutation\_somatic\_status$. Listing 1.1 depicts whole list of functional dependencies by which table in our example is characterized.

Removing redundancies from data sources may not only improve the velocity of the data integration process but also decreases the volume of intermediate data to be transferred and later reduce the time of query execution over the generated graphs. The normalization theory of relational databases [6, 7] is tightly connected to eliminating redundant functional dependencies hence removing redundancies in data. The decomposition of the original table needs to be lossless. However, it can not guarantee that the output RDF graph resulted from the data integration system is also lossless or even if it does not contain extraneous RDF triples. Albeit important role of decomposing relation databases in improving the knowledge graph creation

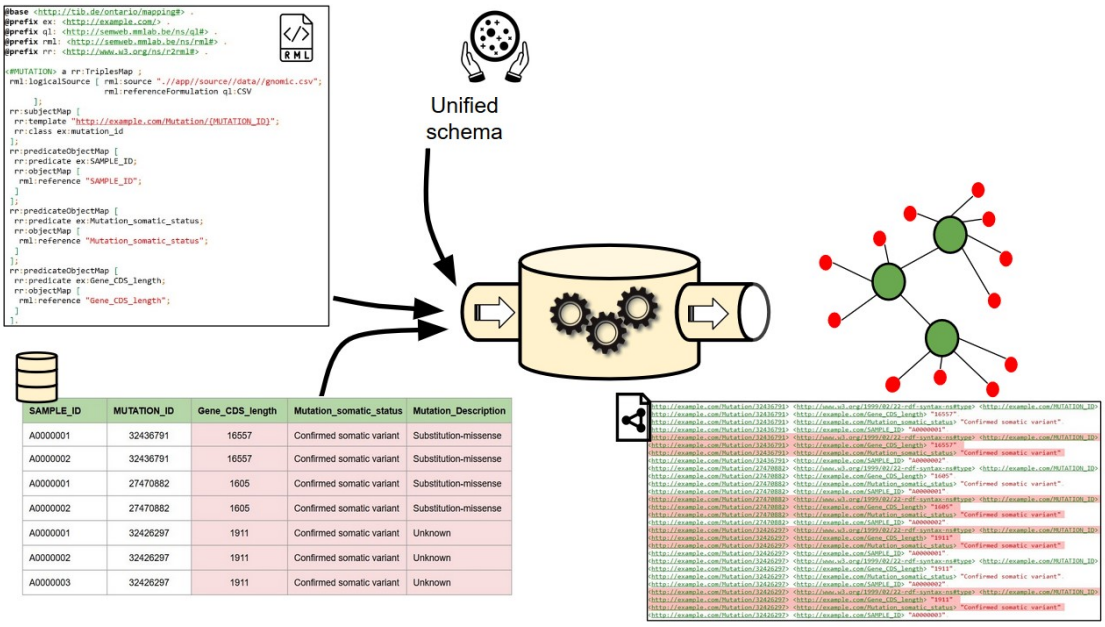


Figura 1.1 – **Motivating example** A data integration system receiving a unified schema, a data source, and an RML mapping rule defined over the data source as inputs and generates RDF graph accordingly. Input data source and resulted RDF graph contains redundant data.

process's performance, this does not suffice to solve the problem of redundant data within data integration systems, since both data sources and mapping rules are involved intensely with the task of data transformation in the process. Naturally, one can consider removing duplicate triples in a generated RDF graph after data integration to solve the recent problem. Regarding this, factorization and compression approaches can be employed to remove these redundancies within knowledge graphs [8, 9]. Nevertheless, eliminating redundancies after the semantification process of data costs expensive and does not offer the possibility to speed up the data integration process, i.e., knowledge graph creation pipelines. With these limitations in mind, our approach tackles the problem of redundant data in both data sources and output RDF triples and performance improvement of data integration systems.

1.2 Problem And Contributions

This work will present a normalization theory considering both mapping rules and their corresponding data sources. This, in turn, can lead to performance improvement of the data integration process and significant space savings in terms of data sources and knowledge graph. Although there is no doubt in the usefulness of normalization relational databases, a mapping rule [1, 4] violating normal forms may have redundancies too. Therefore, we find it necessary for both the source and the mapping rules to be normalized. Accordingly, it is essential to define a theory of normalization correctly matching the concept and structure of corresponding mapping rule language. Lastly, normalized rules and data sources must produce the same knowledge graph as the original ones. Based on the above, new normalization theory considering both mapping rules and data sources and related definitions are presented in this work. It is determined accordingly that the lossless join property holds for transformed mapping rules.

To eliminate the redundancies existing in data sources and those generated as output by mapping rules and improve the knowledge graph creation process's performance, both mapping rule and its corresponding data source need to be in certain normal forms. Normalization of data sources not only prevents anomalies concerning insertion, updating, and deletion of data but also remove existing data redundancies [6]. Normalization of a relational table may lead to a decomposition of that table into several new tables that might have a much smaller total size than the original table. These new tables do not present any more recent problems. Considering the original table as an input source to a mapping rule [1, 10], current work presents a new form of normalization which is applicable for mapping rules. In addition, this work also includes efforts to normalize a specific type of mapping rules, namely RML. As a result of this normalization, decrements in the size of intermediate sources, i.e., input data sources and generated RDF graph will be shown and proved. Furthermore, an algorithm will be proposed for transforming existing mapping rules alongside their corresponding input sources into normal forms. This algorithm offers the advantages of using a new normalization theory called Mapping Rule Normalization Theory. A Python implementation of this algorithm for RML mapping rules and CSV data sources is a part of this work to investigate the impacts of this theory on different dimensions and be helpful to the community of semantic web.

In addition to the efforts mentioned above, this work proves performance improvement when creating knowledge graphs based on Mapping Rule Normalization Theory. This may lead to decompositions of rules and data sources as well as using join operator between the rules in order to generate the same RDF graph. Although the join operator is the most expensive one, especially in relational algebra, we mostly show performance improvement where the original data source contains numerous redundancies. Concerning scientific experiments, there is a need for generating testbeds, which is part of the current work. To that end, a tool for generating

relational tables has been developed. This tool is generating data based on a specified functional dependency set. Thus, in the end, the data implies every functional dependency existing in the set. Furthermore, it is proven that the new normalization form fulfills lossless join property, therefore using joins does not result in losing data. All in all, the current thesis place emphasize on the following novel contributions:

- Normalization of input sources and mapping rules concurrently.
- The definition of normalization theory over mapping rules.
- An Algorithm for normalizing mapping rules
- RML-Normalizer - an implementation of the proposed algorithm.
- Lossless join decomposition of mapping rules based on the proposed algorithm.
- A tool for generating synthetic testbeds based on a desired set of functional dependencies.
- Experimental evaluations demonstrating the performance improvement in knowledge graph creation process as well as space savings.

The rest of this work is organized as follows. In Section 2, preliminaries and backgrounds related to the work of this thesis are described. The following section presents an analysis of the related work done in the area of data integration. In Section 4, the proposed normal forms are formalized, and the approach is explained. An implementation of the proposed algorithm is presented in Section 5. Section 6 shows different experiments accompanied by their results. Furthermore, to support the position of this work, Section 7 contains some discussions and analyses of experiments in the previous section. Finally, this section exhibits limitations of this work as well as an outlook for future works open in this area to the community.

1.3 Summary Of The Chapter

To conclude, this chapter introduces to the working area of this thesis as well as a motivating example to explain the existing problem. In addition, this chapter described the contribution of this thesis.

Capitolo 2

Background

Databases play a crucial role in organizing data produced by users and, more specifically, to make instances of the real world. Due to dramatic advances in database systems, they are used in different scientific areas and implemented other underlying platforms, particularly web-based platforms. There is no doubt that these data are of great importance to both users and machines. Considering these two consumers of data and significant growth of produced data within the web, web technologies like semantic web influences almost every aspect of our life much more than we think it does. In addition, semantic web technologies need to be fed with the data produced in different formats with relational tables the most popular of them [1, 11]. In this section, some basic concepts of semantic web and data integration systems are introduced. Finally, some fundamentals of relational databases are explained.

2.1 Semantic Web

Data existing in Web documents and utilizing them further in different types of applications require a framework available to other communities and organizations. The *semantic web* provides different communities, organizations, and applications with this opportunity. Thus data within web documents can be used to discover new information. In addition, the *semantic web* plays a vital role in connecting different objects, whether in the real-world or abstract. With this in mind, the existing data will be extended with new additional data, which they are connected with other types of data and so forth. To this end, the *semantic web* deals with large quantities of data stored in different applications and even with different formats [12]. According to Tim Berners-Lee, the inventor of the World Wide Web, ≪*The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation*≫. To have such a kind of framework, one needs to consider a data-model and a formal language. As a result, the Resource Description Framework (RDF) language has been adopted by *W3C* and applied in semantic web applications [13].

2.1.1 Resource Description Framework (RDF)

RDF provides different communities with a common framework for describing the information in the web. It is a vendor- and operating system-independent infrastructure which is accessible and extensible in different domains of knowledge. A semi-structured data model can be designed and built based on semantic of RDF. This, however, may need different communities to define their own new vocabulary in order to collect, process, and share the data. Of course, this new

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dbo: <http://dbpedia.org/ontology/>
@prefix dbp: <http://dbpedia.org/property/>
@prefix dbr: <http://dbpedia.org/resource/>

<dbr: Lebron James> <rdf:type> <dbo:BasketballPlayer>.
<dbr: Lebron James> <dbo: team> <dbr: cleveland cavaliers>.
<dbr: Lebron James> <dbo :position> <dbr: small forward>.
<dbr: Lebron James> <dbp: nationality> "American".
```

Figura 2.1 – **An example set of RDF triples;** RDF triples serialized in turtle format and describing an entity existing in DBpedia

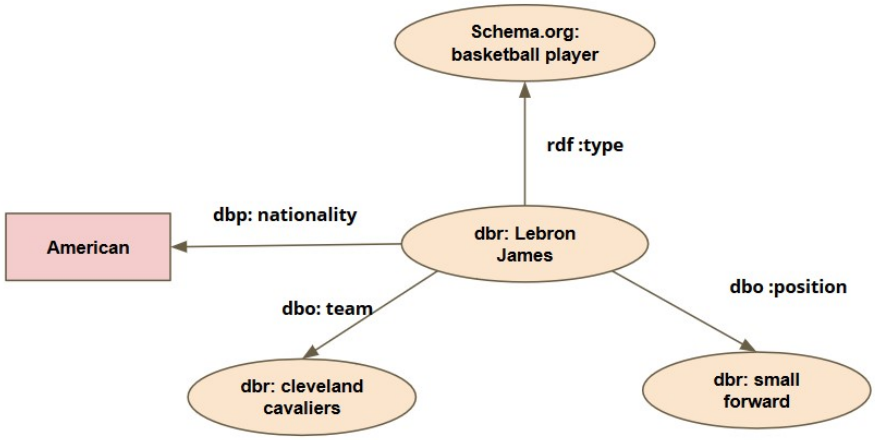


Figura 2.2 – **An example of an RDF graph;** Nodes represents resources and literals while pointed arcs are properties and show relations between nodes.

vocabulary is operational, as long as, an RDF vocabulary is in use. This means that a new vocabulary is always defined based on RDF vocabulary [2, 14]. The following definitions state the two fundamental concepts of RDF.

Definition 2.1 (**RDF triple**)**.** Given sets of URIs, blank nodes, and literals defined over U, B, and L respectively, an RDF triple is a tuple of the form (s, p, o) in which s, p, and o stand for Subject, Property, and Object. Note that each pair of these sets are disjoint. A sample of some RDF triples can be observed in figure 2.1

Definition 2.2 (**RDF graph**)**.** An RDF graph is a collection of RDF triples that may show different resources along with their properties and property values. Like any other graph, it is based on node and arcs in which arcs are directed, i.e., it is a directed graph. Nodes are representative of resources or literals, and a directed arc shows a property of that resource. Figure 2.2 shows a example of RDF graph.

```
@base <http://tib.de/ontario/mapping#> .
@prefix ex: <http://example.com/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .

<#NBA_P>
 rml:logicalSource [ rml:source "//app//nba_players.csv";
                     rml:referenceFormulation ql:CSV
      ];
 rr:subjectMap [
  rr:template "http://example.com/resource//{name}";
  rr:class ex:Basketballplayer
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Position;
  rr:objectMap [
   rml:reference "player_position";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Nationality;
  rr:objectMap [
   rml:reference "player_nationality";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:team;
  rr:objectMap [
   rr:parentTriplesMap <#NBA_team>;
        rr:joinCondition [
            rr:child "team_id";
            rr:parent "team_id";
        ];
  ]
 ]
 .

<#NBA_team>
 rml:logicalSource [ rml:source "//app//nba_teams.csv";
                     rml:referenceFormulation ql:CSV ];
 rr:subjectMap [
  rr:template "{team_name}"
  rr:class ex:Baskteballteam
 ];
 rr:predicateObjectMap [
  rr:predicate ex:league_champs;
  rr:objectMap [
   rml:reference "league_champs";
  ]
 ]
```

Figura 2.3 – **a sample RML mapping rule;** This rule contains a join between two triples maps, i.e., NBA_P (child triples map) and NBA_Team (parent triples map) each of which has references to different sources and different number of columns

2.2 RML

The RDF Mapping Language (RML) extends the W3C-standard mapping language R2RML with the possibility of supporting heterogeneous formats (e.g., CSV, Relational, JSON, and XML) [4, 15]. As the W3C-standard R2RML, TriplesMap corresponds to mapping rules where the resources (a.k.a. subjectMap) of an RDF class and their properties (a.k.a. predicateMap) are assigned to values (a.k.a. objectMap) based on logical data sources (a.k.a. logicalSource). An objectMap can be also defined as a reference or a join with the subjectMap in another TriplesMap (a.k.a. RefObjectMap and joinCondition, respectively). An example of an RML mapping rule, with two triples map joining to each other, is shown in the figure 2.3.

2.3 Data Integration

With the rapid increase of data produced within the web and the need to discover knowledge behind this data and making decisions and actions based on them, data integration has been playing a vital rule in the semantic web and specifically knowledge graph creation process. Data integration refers to collecting data from different sources, represented whether in heterogeneous or homogeneous formats and providing users with a unified structure for further uses [16].

2.4 Relational Databases

To begin with, a relational database is based on the concept of set theory in mathematics [6]. Having one or more domains, a subset of the Cartesian product of these domains provides us with the concept of relation in relational databases. For instance, a subset of the Cartesian product with regard to NBA player"s data can be observed in table 2.1. This subset and any other subset of that product is basically a relation. Each member of the above Cartesian product is called a tuple when we talk about relations. Each row of the table is the corresponding concept of a tuple. One can consider names for each component of the above relation. If it is so, these names are called attributes. With regard to this, a relation scheme is defined as a set of attributes assigned with each component of a relation. A relation scheme with **K** attributes, i.e., domains, is called of arity K. If a relation has the same arity and same domains as a relation scheme, then it is a possible relation for that scheme that can be an instance of the specified scheme. Thus, the above-mentioned relation can be seen as the table shown in the figure in which each row and each column are representative for a tuple and a component, respectively.

2.4.1 Database Design

A properly designed database offers advantages of competitive query execution times over it and efficient space usage. However, it is necessary to understand facts about poorly designed databases so perceiving well structures is much more easier. A low quality design of a database can hold the following drawbacks [6]:

- **Redundancy:** This happens when a column's value is repeating for more than one another column's value. One reason for that is the presence of redundant function dependencies within a data relation. In table 2.1 values of *Player_nationality* and *team_name* are two examples of redundancies.
- **Update anomalies:** As a result of the first drawback, modifying an existing value forces an update overall occurrences of the same value, otherwise the data will present inconsistencies.

Name	Player_nationality	team_id	team_name
Lebron James	USA	20	Cleveland cavaliers
Ron Harper	USA	20	Cleveland cavaliers
Mo Williams	USA	20	Cleveland cavaliers
Dirk Nowitzki	Germany	30	DalLas Mavericks
Dennis Rodman	USA	30	DalLas Mavericks

Tabella 2.1 – **A sample relation holding data about NBA basketball players;** This relation is a subset of the Cartesian product of four domains $Name$, $Player\_nationality$, $team\_id$, $team\_name$

In our current example, any modification to the value "Cleveland cavaliers" within the column $Team\_name$ needs to be applied to all three entries existing in the column.

- **Insertion anomalies:** This occurs when creating a new row is impossible since one of the columns, which is always optional in reality, must have a value. Based on our example, suppose we need to create a new row in the table, but there is not yet any assigned $Team\_name$ for the intended row. From a database point of view, we must have a value for this column. One can consider allowing null values in this column. Accordingly, we need to delete this row, the moment that the real value, is assigned. Moreover, when this issue is happening in one of several attributes in a compound key, it prevents the actual task of the primary index.
- **Deletion anomalies:** This issue is the one that occurs when for example we want to delete one specific value of $Team_name$. So then we need to delete every and each row containing the same value for $Team_name$. Consequently, we delete every other value too, and exactly at this moment, we lose the track of all other information in current relation for this specific row, e.g., the name of a player.

Through the normalization of tables, the above-mentioned problems of a poor design database can be solved[6]. However, due to the nature of join operators and the fact that they cost expensive for a database, there needs to be some trade-off between decomposing a table and using Join operations.

2.4.2 Functional Dependency

It is stated in [6, 7, 17] that functional dependencies, FD, are integrity constraints developed in the real-world through which legal possible relations for a relation scheme will be indicated. This means that not every possible relation can be an instance of a particular scheme unless all functional dependencies hold for that relation. A functional dependency f is a function $f_1 : X \to Y$ where each of X and Y is subsets of a set of attributes in a relation. To clarify this, given a specific value of X, Y has only one value associated with the value of X. It is not necessary for Y to be associated exactly with one value of X. If it is so, then the relationship between X and Y is a one to one relationship, otherwise it is a Many to one relationship. In fact, the former shows that there is another functional dependency $f_2 : Y \to X$. However, if there are no such kind of functional dependencies between X and Y, then it can be inferred that at least one value in X has one or more assigned values of Y and vice versa.

We say a relation r satisfies a functional dependency $f : X \to Y$ whenever for each tuple μ and ν in r one of the following holds:

- $\mu[X] = \nu[X]$ then $\mu[Y] = \nu[Y]$
- $\mu[X] \neq \nu[X]$

On contrary, if $\mu[X] = \nu[X]$ then $\mu[Y] \neq \nu[Y]$, we say the relation r violates functional dependency f. In order to investigate whether a functional dependency holds in an existing scheme, it is necessary to check all of the possible relations for the violation of that dependency. Considering the relation in the recent example, table 2.1, where $Name$ is the primary key of the relation. The dependency between $Name$ and all other attributes in the relation is one clear example of a functional dependencies. List of functional dependencies defined over this relation can be seen in listing 2.1. It is obvious in table 2.1 that each unique value of $Name$ is in relation with exactly one value from all other domains. Therefore the related functional dependency is holding in the relation.

2.4.3 Redundancy In Relational Databases

Redundancy and inconsistencies in databases are generated by the lack of satisfaction of integrity constraints. One typical example that generates usually redundancies is the lack of enforcement of functional dependencies. As an illustration, violating functional dependencies causes that specific columns to have repeated values. This will cause certain problems described in section 2.4.1. One example is when two functional dependencies $f_1 : X \to Y$ and $f_2 : Y \to Z$ hold in a relation concurrently. To put it simply, based on the nature of functional dependency f_1, values in Y can be repeated several times for each different value of X, yet each unique value of Y must not have more than one unique value. Now the values of Z are repeating per value in Y. Therefore for each repeating value of Y, the corresponding value of Z will be repeated extra. Principally, the presence of the redundant functional dependency $f_3 : X \to Z$, which can be inferred using Armstrong's axioms [5], results in redundancies in the relation. To help clarify the point, consider the relation in our current example. In table 2.1 and according to set of functional dependencies presented in listing 2.1, the functional dependency $Name \to team_name$ is implied logically over the relation using Armstrong's axioms. Them, it is called a redundant dependency and it needs to be removed to tackle the problem described in 2.4.1.

2.4.4 The Armstrong's Axioms

In the current section some of the inference rules defined on top of the Armstrong's axioms [5, 6] are described; they are used for discovering new functional dependencies as well as keys in a relation. Given a relation with a set of attributes X, a set $Y \subseteq X$, a set $Z \subseteq Y$, a set $W \subseteq X$ and a set of functional dependency F over its attributes the following rules are valid and sound:

- **Reflexivity:** $Y \to Z$ can be inferred logically from F and more specifically it is called a trivial dependency.
- **Augmentation** If $Y \to Z$ holds in X, then $Y \cup W \to Z \cup W$ can be implied logically from F.
- **Transitivity** if both $Y \to Z$ and $Z \to W$ holds in X, then based on this rule $Y \to W$ is a logical implication in F.

There are some other rules stated in [6, 7] which can be built based on the above-mentioned rules.

2.4.5 A Closure Set Of Functional Dependencies

The inference rules described in the last section can be used to discover and build a full set of functional dependencies upon attributes in a relation. This complete set, which contains all of the explicit and implicit dependencies, is called a closure set of functional dependencies. It is stated in [6] that the closure set of functional dependencies plays an essential role in database design. For one thing, it aims strongly in finding the violations of normal forms. Another thing

```
{Name} → {player_nationality}
{Name} → {team_id}
{team_id} → {team_name}
```

Listing 2.1 – List of functional dependencies holding in sample relation in 2.1

is, this set can be used to find the keys existing in a relation. Finally, it can be used to define different methods to bring a relation to a specific normal form, e.g., Third Normal Form. The next section aims to describe the basics of normal forms in relational databases.

2.4.6 Normalization Theory Of Relational Databases

In order to understand different normal forms, it is necessary to have some basic definitions first. Given a relation R with set of attributes X, the followings are definitions explained in [6, 7, 18].

Definition 2.3 (**Candidate key**)**.** A set of attributes $Y \subseteq X$ is a candidate key if it functionally determines all attributes within X, even its own elements, yet there is no other set $Z \subseteq Y$ which has the same property.

Definition 2.4 (**Primary key**)**.** One of the candidate keys within R can be chosen as the primary key of the relation. Primary is often and generally called just "Key".

Definition 2.5 (**Superkey**)**.** Any set of attributes $Y \subseteq X$ containing any candidate key is called a superkey. Therefore set X is a trivial superkey based on set theory.

Definition 2.6 (**Prime attribute**)**.** An attribute is called prime whenever it is an element of any key in a relation.

Definition 2.7 (**Partial dependency**)**.** Given a single attribute $A \in X$ and a set $Y \subseteq X$, where $Y = Y_1, Y_2, ..., Y_k, ..., Y_m$ and a functional dependency $f_1 : Y \to A$ in F, a partial dependency occurs whenever there exists a set $Z = Y_1, Y_2, ..., Y_k$ for which $f_2 : Z \to A$ holds also in X. Subsequently, it can be seen that attributes $Y_{k+1}, Y_{k+2}, ..., Y_m$ are extraneous.

Definition 2.8 (**Full dependency**)**.** Considering the definition in 2.7, if there is no such a set Z, then A is fully dependant on Y.

Definition 2.9 (**Transitive dependency**)**.** Given a single attribute $A \in X$, two sets $Y \subseteq X$ and $Z \subseteq X$, a transitive dependency occurs whenever there exists functional dependencies $f_1 : Y \to Z$ and $f_2 : Z \to A$ in F, yet $A \notin Y$ and $A \notin Z$ and $f_3 : Y \not\to X$ in F.

Definition 2.10 (**First Normal Form**)**.** The relation R is in First Normal Form (1NF) if all of its domains are atomic and any attributes belonging to R have only single values of its domain.

Definition 2.11 (**Second Normal Form**)**.** The relation R is in Second Normal Form (2NF) if it is in first normal form and for each attribute $A \in X$ and each key $Y \subseteq X$, either A is a prime attribute or it is fully dependant on Y.

Definition 2.12 (**Third Normal Form**)**.** A relation R is in Third Normal Form (3NF) if it is in Second Normal Form and for each nonprime attribute $A \in X$, there is no key $Y \subseteq X$ yet A is transitively dependant upon Y.

Definition 2.13 (**Lossless join decomposition into Third Normal Form**)**.** Decomposition of relations play an important role when there is a need to remove violations of normal forms. However, not every decomposition is possible and it is because of lossy property of some decompositions. It is claimed in [6], that a lossless join decomposition shows always the same results. A lossless decomposition of the relation R into relation $R_1, R_2, ..., R_n$ is basically defined as below:

- $R = \Pi_{R_1} R \bowtie \Pi_{R_2} R \bowtie ... \Pi_{R_n} R$

2.5 Data Integration Systems

A Data Integration System (DIS) defines the components that allow for the integration of heterogeneous data sources into a unified schema following a set of mapping rules. A seminal article by Lenzerini [16] presents an abstract structure of DISs based on the following definitions:

Definition 2.14 (Data Integration System [16])**.** A data integration system $\mathcal{I}$ is defined in terms of a triple $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$, where

- $\mathcal{O}$ is the global schema or ontology, expressed in a language $\mathcal{L}_\mathcal{O}$ over an alphabet $\mathcal{A}_\mathcal{O}$. The alphabet comprises a symbol for each element of $\mathcal{O}$ (i.e., relation if $\mathcal{O}$ is relational, class if $\mathcal{O}$ is an ontology, etc.).

- $\mathcal{S}$ is a set of data sources expressed in a language $\mathcal{L}_\mathcal{S}$ over an alphabet $\mathcal{A}_\mathcal{S}$. The alphabet $\mathcal{A}_\mathcal{S}$ includes a symbol for each element of the sources.

- $\mathcal{M}$ is a set of mappings $\mathcal{O}$ and $\mathcal{S}$ of the form:

$$q_\mathcal{S} \rightsquigarrow q_\mathcal{O},$$

$$q_\mathcal{O} \rightsquigarrow q_\mathcal{S}$$

  where $q_\mathcal{S}$ and $q_\mathcal{O}$ are two queries of the same arity over the source schema $\mathcal{S}$, and over the global schema/ontology. Queries $q_\mathcal{S}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ over the alphabet $\mathcal{A}_\mathcal{S}$, and queries $q_\mathcal{O}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{O}}$ over the alphabet $\mathcal{A}_\mathcal{O}$. Intuitively, an assertion $q_\mathcal{S} \rightsquigarrow q_\mathcal{O}$ specifies that the concept represented by the query $q_\mathcal{S}$ over the sources corresponds to the concept in the global schema represented by the query $q_\mathcal{O}$ (similarly for an assertion of type $q_\mathcal{O} \rightsquigarrow q_\mathcal{S}$).

Mappings among sources and the global schema can be specified in following different paradigms, e.g., *global-as-view* (GAV) [19], and *local-as-view* (LAV) [20].

Mappings established using the local-as-view (LAV) approach represents the data sources as views over the global schema. The mappings in $\mathcal{M}$ associate each element s in source schema $\mathcal{S}$ with a query $q_\mathcal{O}$ defined over the global schema. The mappings established by LAV comprise a set of assertions, one for each element s of source schema $\mathcal{S}$, given as:

$$s \rightsquigarrow q_\mathcal{O}$$

where $s \in \mathcal{S}$ and $q_\mathcal{O}$ is a query defined over the global schema $\mathcal{O}$

Global-as-view (GAV) approach represents the concepts in the global schema as a set of views over the data sources. The mappings in $\mathcal{M}$ associate each element g in the global schema with a query $q_\mathcal{S}$ over the data source. The mappings established by GAV encompass a set of assertions, one for each element g of $\mathcal{O}$, given as:

$$g \rightsquigarrow q_\mathcal{S}$$

where $g \in \mathcal{O}$ and $q_\mathcal{S}$ is query defined over the sources in $\mathcal{S}$.

The mapping language RML follows the Global-as-view (GAV) approach, i.e., it enables the definition of each entity of a class and their predicates in terms of the attributes of a set of data sources. Furthermore, RML allows for the definition of mapping rules among data sources in diverse formats, e.g., CSV, JSON, XML, and relational databases.

2.6 Summary Of The Chapter

This chapter presented all the concepts required to understand the problem addressed in this thesis, as well as the proposed solution and the empirical evaluation conducted for validating the efficiency of the proposed approach.

Capitolo 3

Related Work

With the growing popularity trend of knowledge graphs day by day and the development of novel technologies, the role of data, especially web data, is much more obvious to us than before. However, the presence of redundancies within the data can cause serious problems, being inefficient execution of knowledge graph creation processes one of them. To tackle the problem of redundant data, some efforts have been done in both communities of Semantic Web as well as Database. Although there are approaches which are concentrating on discovering unknown dependencies in data[21], the main parts of the works are dealing with Normalization of Relational Databases in order to eliminate redundancies in databases [22, 23]. However, the problem of redundancies in intermediate resources during the knowledge graph creation process as well as in existing knowledge graphs is of great value too. This is investigated and solved to some extent in existing works [24]. What is certain, the latter works are often working on the same problem as this thesis.

3.1 Normalization Of Relational Databases

. The facts demonstrate that the normalization of relational databases is still a controversial problem. To speak more specifically, bringing relational tables into Boyce-Codd Normal Form (BCNF) is one most challenging tasks and works in this area are highly appreciated. Papenbrock et al. [23] propose *NORMALIZE*, a semi-automatic data-driven algorithm to normalize datasets into BCNF. In addition to that, this work presents two algorithms to calculate closure set of functional dependencies which can be helpful not only in normalization but also for query optimization and data cleansing. Demba et al. [22] also propose an algorithm for normalizing relation databases but only up to third normal form. The proposed algorithm requires pre-processing to discover minimal cover of the functional dependency set. Although these works are carried out to normalize relational databases, they differ from the work of this thesis at some points. First of all, current work is focused on normalizing mapping rules and their corresponding data sources concurrently. Secondly, our approach considers the set of functional dependencies as input instead of discovering dependencies from within instances of data. Finally, since the input to our algorithm is already minimal, there is no need for our approach to finding the minimal cover of dependencies.

3.2 Mining Functional Dependencies

Metadata of relational databases, e.g., functional dependencies and candidate keys, are not always available and that relational schemes in some case are numerous, there has been a strong demand to mine data to find extra information. This in turn helps to reduce the amount of work done by users to specify those metadata manually. FD_Mine [21] aims at discovering functional dependencies from the extension of a relational scheme by finding equivalent sets of attributes within a dataset and ignoring logically implied functional dependencies in the hope of reducing the search space. To that purpose, 4 rules are explained in that work for pruning the search space, i.e., set of attributes and functional dependencies. Like FD_Mine, $FDTool$[25] is also tool for discovering minimal set of functional dependencies, candidate keys and equivalent attributes. It is re-implemented in hopes of performance and process improvement. Although these works have a novel contribution to the relational database community, the work of our thesis is not focused on the mining of functional dependencies and finding equivalences, since the set of functional dependencies is considered as an input to our approach. However, it should be noted that *FDtool* is utilized in testbed generations within our work and particularly for checking the satisfaction of functional dependencies in the generated relational tables.

3.3 Transforming Data Integration Systems

During recent years great efforts have been done in order to solve the problem of knowledge graph creation and in particular to speed up the process of big data integration. An ongoing issue in this area of knowledge is the scale of data which is the subject of transformation to a knowledge graph. *MapSDI* [24], a scaled-up Data Integration framework for knowledge graph creation, is relying on the transformation of a mapping rule's data source to solve the problem. Moreover, it is exploiting the semantic data within mapping rules to project out referred attributes from data sources. Subsequently, *MapSDI* eliminates redundancies within the recent set of attributes and select the relevant data. Despite the second to none contribution of *MapSDI*, it is focusing to solve the problem by eliminating redundancies in data, e.g., remained after attribute projection process or by merging different datasets. Our work is different in the sense that we are normalizing a mapping rule along with its corresponding data source and accordingly eliminating redundancies caused by specific functional dependencies. Nevertheless, *MapSDI* targets also the same problem as in this thesis. Needless to say, it can be used to preprocess the input data source to our approach in the hope of reducing normalization time and further performance improvement in the knowledge graph creation process.

3.4 Normalization In Graph Databases

Nowadays, scientists in the semantic web community inform us about data redundancies happening in existing knowledge graphs and discuss the advantages of normalizing these graphs in order to reduce those redundancies. In this respect, redundant data within the graphs increases their size without being necessary. Another thing is, the redundancies can be a serious problem when processing queries over knowledge graphs. Karim et al.[9] solve the problem of redundant data within knowledge graphs on the basis of factorization techniques. In this innovative work, the concept of *factorized RDF graphs* is formulated which is based on so-called *frequent star patterns*. The aim is to have a compressed knowledge graph whose size of redundant properties and objects are reduced. The amount of data that is streamed from sensor networks and is stored in the knowledge graph is unprecedentedly large and it grows rapidly without any doubt. Needless

to say, measurements by sensors observed often many times and this results in redundancies of data. A factorization approach is proposed also to eliminate the redundant entries within a knowledge graph consisting of sensor data [8]. Besides, an algorithm is presented to evaluate queries over the tabular representation of a factorized RDF graph. In spite of the fact that precious works have been done in this area, none of the above-mentioned works on removing redundancies prior to creating knowledge graphs. Our proposed approach presents normalization in the level of data integration and proves the lossless decomposition of mapping rules. Therefore this work can be also helpful to the above-mentioned works by reducing the number of redundancies within the RDF knowledge graph.

3.5 Summary Of The Chapter

In the current chapter reviews of some relevant works in different areas of knowledge were explained and accordingly they were compared to the work of this thesis.

Capitolo 4

A Normalization Theory for Mapping Rules

In the present thesis, we aim at explaining the problem of knowledge graph creation concerning the scale of data to be transformed. Accordingly, in current section we first attempt to formalize the problem of data integration systems. After that, a novel normalization theory for mapping rules, including different definitions, is introduced. On basis of this theory, redundancies caused by functional dependencies, holding over a set of attributes mentioned by mapping rules, are exemplified. Then, a solution to the problem of data integration system is proposed. This includes an algorithm to transform mapping rules and corresponding data sources into decompositions respecting newly defined normal forms. Lastly, lossless property of recent decompositions is proved theoretically.

4.1 Problem Statement

The problem of knowledge graph creation which is described in this thesis, is basically the one originating from two components of a data integration system, namely source schema and mapping. Given a data integration system $DIS_G = \langle U, D, M \rangle$ [16, 24], modeled as *Global as view (GAV)*, the components are defined as following:

- **U** is a unified global schema, i.e., ontology, which is a triple in form of $U = (C, P, A)$. C and P are referring to vocabulary of U, i.e., classes and properties, while A is a set containing axioms used for interpreting the content of vocabulary in a given domain of discourse.
- **D** is set of data sources, later utilized by mapping rules, which is defined over alphabet set $\langle D_1^{Attr_1}, D_2^{Attr_2}, ..., D_n^{Attr_n} \rangle$ with each D_n an element of D , e.g., relation in relational databases or object in object oriented data model etc. In addition, each $Attr_n$ specifies set of attributes corresponding to element D_n.
- **M** is set of mapping rules defined in a mapping language of choice which is defined as following:

$$r_k : \underbrace{c_j(X, \overline{X})}_{\text{head of the rule}} :- \underbrace{D_1(\overline{X_1}), D_2(\overline{X_2}), ..., D_m(\overline{X_m})}_{\text{body of the rule}} \quad (1)$$

  where c_j is a class in C and it is a conjunctive query on the sources and attributes in D. X is a variable in the above definition and $\overline{X}$ is a set of pairs $(P_{i,j}, X_{i,j})$ with $P_{i,j}$ a property

of type C and $X_{i,j}$ a variable. The above rule is safe since all of the variables are involved in the body of rule. In addition, $D_z(X_z)$ is a source in D with X_z as a set of pairs $(a_{i,z}, X_{i,z})$ where $X_{i,z}$ is a variable and $a_{i,z}$ is an attribute in D_z.

Let V be set of all variables in the rules existing in M and set I be union of all data items within D. Then the evaluation of each rule r_k in the above mentioned definition of data integration system leads to generation of an RDF knowledge graph G and it is defined as following.

$$\mu : V \rightarrow I. \tag{2}$$

Based on 1 and 2, evaluation $D_z(\overline{X_z})$ on μ, i.e., $eval(D_z(\overline{X_z}), \mu)$, corresponds to a set μ_{D_z} composed of pairs $(a_{i,z}, X_{i,z})$ in $\overline{X_z}$, each of which hold both of the followings:

- The pair $(X_{i,z}, \mu(X_{i,z}))$ belongs to μ_{D_z}
- If $\langle a_{1,z}, ..., a_{q,z} \rangle$ are the attributes of D_z in X_z, then the tuple $\langle (a_{1,z}, \mu(X_{1,z})), ..., (a_{q,z}, \mu(X_{q,z})) \rangle$ belongs to the data extension of D_z

Let t be an RDF triple of the form $t = (s\ p\ o)$. Then the evaluation of each rule r_k in (1) over μ defined as $Eval(r_k, \mu)$ results in a set of RDF triples t obtained as one of the following:

- $\boldsymbol{r_k : c_j(X, \overline{X}) :\text{–} D_1(\overline{X_1})}$

  If the pair $(X, \mu(X))$ exists in μ_{S_1}, then for each $(P_{i,1}, X_{i,1}) \in X$ and each $(X_{i,1}, \mu(X_{i,1})) \in \mu_{S_1}$, $t = (\mu(X)\ P_{i,1}\ \mu(X_{i,1}))$

- $\boldsymbol{r_k : c_j(X, \overline{X}) :\text{–} D_1(\overline{X_1}), D_2(\overline{X_2}), ..., D_m(\overline{X_m})}$

  If the pair $(X, \mu(X))$ exists in at least one μ_{D_z}, then for each $(P_{i,z}, X_{i,z}) \in \overline{X}$ and each $(X_{i,z}, \mu(X_{i,z})) \in \mu_{D_z}$, $t = (\mu(X)\ P_{i,z}\ \mu(X_{i,z}))$

Let $RDFize(.)$ be a function that maps $DIS_G\langle U, D, M \rangle$ with the resulted knowledge graph from evaluation of all rules r_k utilizing sources D, then result of this function can only be impacted by rules r_k or D_z. In fact, the execution time of $RDFize(.)$ plays an important role in this thesis which is strongly increased by redundancies resulted from functional dependencies. Based on this, we intend to rewrite a data integration system $DIS'_G\langle U, D', M' \rangle$ with the following properties:

- The execution time of $RDFize(DIS'_G\langle U, D', M' \rangle)$ is minimized.
- $RDFize(DIS_G\langle U, D, M \rangle)$ and $RDFize(DIS'_G\langle U, D', M' \rangle)$ produce the same results, i.e., $RDFize(DIS_G\langle U, D, M \rangle) = RDFize(DIS'_G\langle U, D', M' \rangle)$.

4.2 Proposed Solution

In this thesis, we propose a normalization theory to solve the above-mentioned problem. We hypothesize that data integration systems composed of mappings and data sources that are free of redundancies will enable the generation of knowledge graphs in less time. Thus, we consider as a solution to the problem a data integration system $DIS'_G\langle U, D', M' \rangle$ that meets the following conditions:

- All mapping rules in M' and their corresponding data sources in D' are normalized up to third normal form.

- The data sources in D' meet the lossless join property with respect to the data sources in D.
- The intermediate results of $RDFize(DIS'_G\langle U, D', M'\rangle)$ meet the lossless join property respecting to the intermediate results generated by the execution of $RDFize(DIS_G\langle U, D, M\rangle)$.

In this section, normal forms for mapping rules and data sources are defined.

4.2.1 A Mapping Rule Normalization Theory

With variety and velocity of data produced and used in the web nowadays, and considering the fact of huge redundancies existing in those data, we find it necessary for mapping rules to hold certain normal forms in order to avoid redundancies. In fact, when it comes to data integration systems and mapping rules, it does not suffice to normalize only the relations, i.e., data sources and normalization of mapping rules plays a vital role in knowledge graph creation due to several reasons. For one thing, execution of mapping rule, violating normal forms, generates intermediate data containing redundancies. As a consequence, execution of other related mapping rules will be influenced so the pace of knowledge graph creation process, specially when the data sources are huge. Another thing is execution of this mapping rule alone produces duplicate RDF triples. Naturally, this can be of great impact, when it comes to big data.

Suppose a mapping rule along with its corresponding data source is not normalized with respect to some set of functional dependencies. Accordingly, the mapping rule dominates which subsets of attributes should be utilized to transform the data within data sources into a knowledge graph. As a result, existence of redundant functional dependencies not only in data sources but also in mapping rules generate duplicated results, thus both of them must to be normalized. For one thing, decomposing data sources embodying redundant dependencies results in eliminating redundancies in data following by improving the performance of data integration process. Another thing is that normalization of mapping rules prevents the generation of redundant RDF triples prior to evaluation of a data integration system. This offers the advantage of eliminating redundancies with lower costs compared to other solutions such as removing duplication completely at data level. All in all, it is necessary for a data integration system that both of a data source and a mapping rule are normalized according to the set of functional dependencies.

4.2.2 Normal Forms For Mapping Rules

First, the proposed normal forms will be presented, and an algorithm for normalizing mapping rules will be shown in the following section. Lastly, it will be proved that the corresponding decomposition of a mapping rule, based on the algorithm, generates no different knowledge graph compared to original rule.

Mapping Rule First Normal Form

The main goal of First Normal Form of mapping rules is to have atomic data types as well as atomic data domains with respect to attributes mentioned in mapping rules.

Definition 4.1. Given a data integration system $DIS_G\langle U, D, M\rangle$, source $D_z \in D$, with sets of attributes $Attr_z$, and mapping rule r_k defined over D_z, with object maps defined over the attributes $T = \{T_1, T_2, ..., T_n\}$ such that $S \subseteq T$ as the subject map and $T \subseteq Attr_z$. Mapping rule r_k is in First Normal Form, whenever D_z is in First Normal Form and both of the followings holds:

```
{MUTATION_ID} → {Gene_CDS_length}
{MUTATION_ID} → {Mutation_Description}
{Mutation_Description} → {Mutation_somatic_status}
```

Listing 4.1 – List of functional dependencies holding in genomic datasets of 4.1

- For each object map within r_k, it is referring to exactly one attribute T_z, i.e., column.

A mapping rule is not in First Normal Form whenever its object maps are are composed of one attributes existing in the related data source. It is essentially suggested to check violations of 1NF of mapping rules with all object maps presented in that mapping rule, whether it contains column-valued or template-valued term.

Mapping Rule Second Normal Form

Like second normal form of relational databases [6, 7], this normal form is defined over mapping rules based on definitions of partial and full dependency.

Definition 4.2. Given a data integration system $DIS_G\langle U, D, M\rangle$ and sets of functional dependencies F, source $D_z \in D$, with sets of attributes $Attr_z$ characterized by set of functional dependencies F_z, and mapping rule r_k defined over D_z, with object maps defined over the attributes $T = \{T_1, T_2, ..., T_n\}$ such that $S \subseteq T$ as the subject map and $T \subseteq Attr_z$. Mapping rule r_k is in second normal form with respect to set of functional dependencies F_z, whenever both r_k and D_z are in first normal forms and both of the followings holds:

- $f : S \rightarrow T_i \in F_z$ holds in T, then either T_i is fully dependant upon S or $T_i \subseteq S$
- $Y \subseteq T$ and $f : Y \rightarrow S \in F_z$ holds in T, then S is fully dependant upon Y or $S \subseteq Y$

A mapping rule is not in second normal form, when one of the aforementioned conditions is not satisfied. Like relational databases [6], a violation of 2NF must not be looked for only in set of functional dependencies but also in closure set of functional dependencies. This violation in the mapping rules will be removed by placing the related attributes, which caused the partial dependency, in a different mapping rules. In other words, *Logical References* to these attributes in current mapping rule must be replaced with *Referencing Object Maps*.

Example 4.3. It can be seen in Figure 4.1 that the RML rule is violating the second normal form defined above. It is happening to be the case since the second condition mentioned in definition 4.2 is not satisfied. To make it clear, the rule is mentioning attributes *Mutation_somatic_status* and *Gene_CDS_length* which both are dependant upon *MUTATION_ID*, yet *MUTATION_ID* is part of another functional dependency, i.e., {*SAMPLE_ID,MUTATION_ID*}. It turns out that this violation leads to redundancies in both attributes *Mutation_somatic_status* and *Gene_CDS_length* which results also redundancy in intermediate data generated during the evaluation of the mapping rules. Consequently, this can increase the execution time of mapping rules in the process of knowledge graph creation. In addition, the redundancies caused by violation of second normal form generate duplicate RDF triples, thus, affects the further applications over resulted RDF graph, i.e., query execution. Figure 4.2 presents the RDF graphs resulted by executing original version of the mapping rule versus normalized version. The duplicates generated by the original version is clear in Figure 4.2.

Example 4.4. Unlike the mapping rule shown in figure 4.1a, the mapping rule in figure 4.1b is completely in second normal form with respect to its set of functional dependencies, thus there is no need to normalize this mapping rule. These two mapping rules differ in that the latter does not mention attribute *SAMPLE_ID*, thus all attributes within this mapping rule are fully dependant on the subject.

Mapping Rule Third Normal Form

Although a mapping rule is in second normal form following the definition 4.2, it may still need to deal with issues described in 4.2.1. In fact a mapping rule can still have redundancies, since it holds transitive functional dependencies with regard to set of functional dependencies defined over its set of attributes.

Definition 4.5. Given a data integration system $DIS_G\langle U, D, M\rangle$ and sets of functional dependencies F, source $D_z \in D$, with sets of attributes $Attr_z$ characterized by set of functional dependencies F_z, and mapping rule r_k defined over D_z, with object maps defined over the attributes $T = \{T_1, T_2, ..., T_n\}$ such that $S \subseteq T$ as the subject map and $T \subseteq Attr_z$. Mapping rule r_k is in third normal form with respect to set of functional dependencies F_z, whenever both r_k and D_z are in second normal forms and exactly one of the followings holds:

- functional dependency $f : S \rightarrow T_i \in F_z$ holds in T for every T_i
- for each T_i, a functional dependency $f : Y \rightarrow S \in F_z$ holds in T where $Y \subseteq T$ and $T_i \in Y$

Similar to *Mapping Rule Second Normal Form*, to examine and discover violations of third normal form in a mapping rule, it is necessary to consider always closure set of functional dependencies.

```
@base <http://tib.de/ontario/mapping#> .
@prefix ex: <http://example.com/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .

<#MUTATION> a rr:TriplesMap ;
 rml:logicalSource [ rml:source ".//app//source//data//gnomic.csv";
                     rml:referenceFormulation ql:CSV
      ];
 rr:subjectMap [
  rr:template "http://example.com/Mutation/{MUTATION_ID}";
  rr:class ex:mutation_id
 ];
 rr:predicateObjectMap [
  rr:predicate ex:SAMPLE_ID;
  rr:objectMap [
   rml:reference "SAMPLE_ID";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Mutation_somatic_status;
  rr:objectMap [
   rml:reference "Mutation_somatic_status";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Gene_CDS_length;
  rr:objectMap [
   rml:reference "Gene_CDS_length";
  ]
 ].
```

(a) RML mapping rule violating the 2NF

```
@base <http://tib.de/ontario/mapping#> .
@prefix ex: <http://example.com/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .

<#MUTATION> a rr:TriplesMap ;
 rml:logicalSource [ rml:source ".//app//source//data//gnomic.csv";
                     rml:referenceFormulation ql:CSV
      ];
 rr:subjectMap [
  rr:template "http://example.com/Mutation/{MUTATION_ID}";
  rr:class ex:mutation_id
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Mutation_somatic_status;
  rr:objectMap [
   rml:reference "Mutation_somatic_status";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Gene_CDS_length;
  rr:objectMap [
   rml:reference "Gene_CDS_length";
  ]
 ].
```

(b) RML mapping rule holding the 2NF

Figura 4.1 – **RML mapping rules defined over genomic data source with set of functional dependencies shown in 4.1 defined over its set of mentioned attributes;** (a): An RML mapping rule containing a set of attributes that violate 2NF (b): An RML mapping rule containing a set of attributes that respect 2NF

```
<http://example.com/Mutation/32436791> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32436791> <http://example.com/SAMPLE_ID> "A0000001".
<http://example.com/Mutation/32436791> <http://example.com/Gene_CDS_length> "16557".
<http://example.com/Mutation/32436791> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/32436791> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32436791> <http://example.com/SAMPLE_ID> "A0000002".
<http://example.com/Mutation/32436791> <http://example.com/Gene_CDS_length> "16557".
<http://example.com/Mutation/32436791> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/27470882> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/27470882> <http://example.com/SAMPLE_ID> "A0000001".
<http://example.com/Mutation/27470882> <http://example.com/Gene_CDS_length> "1605".
<http://example.com/Mutation/27470882> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/27470882> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/27470882> <http://example.com/SAMPLE_ID> "A0000002".
<http://example.com/Mutation/27470882> <http://example.com/Gene_CDS_length> "1605".
<http://example.com/Mutation/27470882> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/32426297> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32426297> <http://example.com/SAMPLE_ID> "A0000001".
<http://example.com/Mutation/32426297> <http://example.com/Gene_CDS_length> "1911".
<http://example.com/Mutation/32426297> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/32426297> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32426297> <http://example.com/SAMPLE_ID> "A0000002".
<http://example.com/Mutation/32426297> <http://example.com/Gene_CDS_length> "1911".
<http://example.com/Mutation/32426297> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/32426297> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32426297> <http://example.com/SAMPLE_ID> "A0000003".
<http://example.com/Mutation/32426297> <http://example.com/Gene_CDS_length> "1911".
<http://example.com/Mutation/32426297> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
```

(a) Output RDF graph with duplicates

```
<http://example.com/Mutation/32436791> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32436791> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/32436791> <http://example.com/Gene_CDS_length> "16557".
<http://example.com/Mutation/32436791> <http://example.com/SAMPLE_ID> <A0000001>.
<http://example.com/Mutation/32436791> <http://example.com/SAMPLE_ID> <A0000002>.
<http://example.com/Mutation/27470882> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/27470882> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/27470882> <http://example.com/Gene_CDS_length> "1605".
<http://example.com/Mutation/27470882> <http://example.com/SAMPLE_ID> <A0000001>.
<http://example.com/Mutation/27470882> <http://example.com/SAMPLE_ID> <A0000002>.
<http://example.com/Mutation/32426297> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32426297> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/32426297> <http://example.com/Gene_CDS_length> "1911".
<http://example.com/Mutation/32426297> <http://example.com/SAMPLE_ID> <A0000001>.
<http://example.com/Mutation/32426297> <http://example.com/SAMPLE_ID> <A0000002>.
<http://example.com/Mutation/32426297> <http://example.com/SAMPLE_ID> <A0000003>.
```

(b) Output RDF graph without duplicates

Figura 4.2 – **Output RDF graph resulted by executing** (a): RML rule described in example 4.3 and (b): normalized version of RML rule described in example 4.3

Example 4.6. Figure 4.3b depicts an RML mapping rule which is in second normal form, yet violating the third normal form. This rule is not in third normal form according to set of functional dependencies expressed in 4.1, since the transitive dependency between *MUTATION_ID* and *Mutation_somatic_status* through *Mutation_Description* is holding in the RML mapping rule. This dependency, which is a redundant one, is the main reason of redundancies appearing in the data. This leads later to redundancies in the output RDF graph which is shown in the figure 4.4a. Furthermore, this dependency creates redundant data in the intermediate data. As a matter of fact, performance of knowledge graph creation process will be strongly affected. Third normal form of mapping rules eliminate this transitive dependency and hence the redundancy in the data as well as output RDF graph. The latter is presented in figure 4.4b. By comparing RDF graph appeared in figure 4.4, one can see the difference between the original and normalized version of RML mapping rules in terms of output results.

```
@base <http://tib.de/ontario/mapping#> .
@prefix ex: <http://example.com/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .

<#MUTATION> a rr:TriplesMap ;
 rml:logicalSource [ rml:source ".//app//source//data//gnomic.csv";
                     rml:referenceFormulation ql:CSV
      ];
 rr:subjectMap [
  rr:template "http://example.com/Mutation/{MUTATION_ID}";
  rr:class ex:mutation_id
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Mutation_Description;
  rr:objectMap [
   rml:reference "Mutation_Description";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Mutation_somatic_status;
  rr:objectMap [
   rml:reference "Mutation_somatic_status";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Gene_CDS_length;
  rr:objectMap [
   rml:reference "Gene_CDS_length";
  ]
 ].
```

(a) SDM-RDFizer

```
@base <http://tib.de/ontario/mapping#> .
@prefix ex: <http://example.com/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .

<#MUTATION> a rr:TriplesMap ;
 rml:logicalSource [ rml:source ".//app//source//data//gnomic.csv";
                     rml:referenceFormulation ql:CSV
      ];
 rr:subjectMap [
  rr:template "http://example.com/Mutation/{MUTATION_ID}";
  rr:class ex:mutation_id
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Mutation_Description;
  rr:objectMap [
   rml:reference "Mutation_Description";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Gene_CDS_length;
  rr:objectMap [
   rml:reference "Gene_CDS_length";
  ]
 ].
```

(b) RMLMapper

Figura 4.3 – **RML mapping rules defined over genomic data source with set of functional dependencies shown in 4.1;** (b): An RML mapping rule containing a set of attributes which violate 3NF together (a): An RML mapping rule containing a set of attributes that respect 3NF

Example 4.7. An RML mapping rule which is in both second and third normal form is presented in figure 4.3a. It is obvious that there is no transitive functional dependency implied by any subset of attributes within the rule.

On the basis of the above, in the next section, we present an algorithm to bring mapping rules in third normal form and accordingly second normal form. This will include a decomposition of existing mapping rules.

4.2.3 An Algorithm For Transforming Mapping Rules

We rely on the definition in 4.2.1 and based on that, we propose a solution for the problem described in 4.1. To start with, the redundancies show negative effects on the performance of the knowledge graph creation. For instance, the interpreter of a mapping rule, i.e., *RDFizer*, needs to load a huge amount of data although full of redundancies. Additionally, these redundancies lead to having large intermediate data which can be be expensive for a data integration system. As an illustration, a join between two mapping rules lasts extremely long. Naturally, the possible solution is to eliminate the redundancies. It is already demonstrated that the problem of redundancies existing in data sources and further produced in a resulted RDF knowledge graph is mainly because of violation of some normal forms within a data source and a mapping rule over that. To tackle this problem, our approach offers a novel solution to transform mapping rules alongside data sources into normal forms.

Our approach transforms a data integration system $DIS_G = \langle U, D, M \rangle$ into an equivalent $DIS'_G = \langle U, D', M' \rangle$ such that the sources in D' and the mapping rules in M' up to third normal form. The algorithm 1 depicts the steps required to perform normalization over the above mentioned components of DIS_G. It receives a mapping rule as well as its data source and a set

```
<http://example.com/Mutation/27470882> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/27470882> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/27470882> <http://example.com/Mutation_Description> "Substitution-missense".
<http://example.com/Mutation/27470882> <http://example.com/Gene_CDS_length> "1605".
<http://example.com/Mutation/27470882> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/27470882> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/27470882> <http://example.com/Mutation_Description> "Substitution-missense".
<http://example.com/Mutation/27470882> <http://example.com/Gene_CDS_length> "1605".
<http://example.com/Mutation/32426297> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32426297> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/32426297> <http://example.com/Mutation_Description> "Unknown".
<http://example.com/Mutation/32426297> <http://example.com/Gene_CDS_length> "1911".
<http://example.com/Mutation/32426297> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32426297> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/32426297> <http://example.com/Mutation_Description> "Unknown".
<http://example.com/Mutation/32426297> <http://example.com/Gene_CDS_length> "1911".
<http://example.com/Mutation/32426297> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32426297> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/32426297> <http://example.com/Mutation_Description> "Unknown".
<http://example.com/Mutation/32426297> <http://example.com/Gene_CDS_length> "1911".
<http://example.com/Mutation/32436791> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32436791> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/32436791> <http://example.com/Mutation_Description> "Substitution-missense".
<http://example.com/Mutation/32436791> <http://example.com/Gene_CDS_length> "16557".
<http://example.com/Mutation/32436791> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32436791> <http://example.com/Mutation_somatic_status> "Confirmed somatic variant".
<http://example.com/Mutation/32436791> <http://example.com/Mutation_Description> "Substitution-missense".
<http://example.com/Mutation/32436791> <http://example.com/Gene_CDS_length> "16557".
```

(a) Output RDF graph with duplicates

```
<http://example.com/Mutation/32436791> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32436791> <http://example.com/Gene_CDS_length> "16557".
<http://example.com/Mutation/32436791> <http://example.com/Mutation_Description> "Substitution-missense".
<http://example.com/Mutation/32436791> <http://example.com/Mutation_somatic_status> <Confirmed somatic variant>.
<http://example.com/Mutation/32426297> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/32426297> <http://example.com/Gene_CDS_length> "1911".
<http://example.com/Mutation/32426297> <http://example.com/Mutation_Description> "Unknown".
<http://example.com/Mutation/32426297> <http://example.com/Mutation_somatic_status> <Confirmed somatic variant>.
<http://example.com/Mutation/27470882> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/MUTATION_ID>.
<http://example.com/Mutation/27470882> <http://example.com/Gene_CDS_length> "1605".
<http://example.com/Mutation/27470882> <http://example.com/Mutation_Description> "Substitution-missense".
<http://example.com/Mutation/27470882> <http://example.com/Mutation_somatic_status> <Confirmed somatic variant>.
```

(b) Output RDF graph without duplicates

Figura 4.4 – **Output RDF graph resulted by executing** (a): RML rule described in example 4.6 and (b): normalized version of RML rule described in example 4.6

of functional dependencies defined over its attributes as input. The idea behind the algorithm is to remove attributes from their original location, i.e., mapping rule or data source, where they cause violations of normal forms. For instance, line 31 shows an attribute T_i which is selected out from the original source into another source and later its corresponding reference in mapping rule, i.e., *Logical Reference*, will be removed to a new mapping rule. To be clear, the algorithm decomposes each input mapping rule and data source therefore two effective components of DIS'_G resulted from our approach, D' and M', will be in third normal form.

Normalizing above mentioned components of a data integration system is strongly connected with functional dependencies over set of attributes utilized in a mapping rule. To this end, these attributes need to be selected out from the original data source. As it can be seen can see in line 3 of algorithm 1, only mentioned attributes are moved to a new data source. After that, a redundant functional dependency holding within this set of attributes must be eliminated. In fact, these redundancies are the main reason of partial and transitive dependencies and specifically redundancies in data. Functional dependencies involving recent attributes will be used later as a basis for removing attributes and respectively eliminating redundancies. Overall, data sources are decomposed into third normal form based on left side and right of these dependencies.

On the basis of our problem defined in 4.1, both of components M and D play vital roles in solving the problem. With regard to this, our proposed decomposition consider the principles of mapping rules as the key point for normalization. To make it clear, the decomposition is carried out by taking subject map of a mapping rule into account. The reason is that a subject map generates URI of RDF triples produced by a mapping rule and basically it can not be removed from a mapping rule. Therefore for decomposing into certain normal forms, e.g., 2NF and 3NF, it is necessary to remove attributes which are not implied by the subject of a mapping rule. To that end, each such attribute will be selected out from the input source and moved to a new data source and based on that new source, a new mapping rule with the regarding attribute as Subject Map will be created.

Algorithm 1: for decomposing a mapping rule and its data source to 3NF

Input: A data source D_z with set of attributes $Attr_z$, a set of functional dependencies F_z over $Attr_z$ and a mapping rule r_k composed of a triples map m_k defined over the data source D_z, with object maps defined over the attributes $T = T_1, T_2, ..., T_n$ such that $S \subset T$ as the subject map and $T \subset Attr_z$

Output: Decomposed mapping rule r'_k and its related set of data sources D'_z holding corresponding third normal forms

r'_k=r_k
if $T \neq Attr_z$ **then**
  $D_S = \Pi_X D_z$
end
else
  $D_S = D_z$
end
for $Y \subset T$ **do**
  if $f : Y \rightarrow T \in F$ **then**
    $D'_z = D_S$
    return r'_k, D'_z
  end
end
foreach $T_i \notin S$ **do**
  if $\nexists Y \subset T$ *such that* $f : Y \rightarrow T_i \in F_z$ **then**
    $D'_{ST_i} = \Pi_{S \cup T_i} D_z$
    $D'_z = D'_z \cup D'_{ST_i}$
    Add a new triples map m_{T_i} to r'_k with T_i as Subject Map
    Transform Object Map T_i in original triples map m_k to a referencing object map with a join to m_{T_i} over S.
  end
  else if $\exists Y \subset T$ *such that* $Y \neq S$ *and* $f : Y \rightarrow T_i \in F_z$ **then**
    $D'_{YT_i} = \Pi_{Y \cup T_i} D_z$
    $D'_z = D'_z \cup D'_{YT_i}$
    Add a new triples map m_{T_i} to r'_k with T_i as Subject Map
    Transform Object Map T_i in original triples map m_k to a referencing object map with a join to m_{T_i} over Y.
  end
  $D'_S = D'_S - T_i$
end
foreach $T_i \in S$ **do**
  if $\exists\ Y \subset T$ *and* $W \subset T$ *such that* $Y \neq W$ *and* $S \nsubseteq W$ *and* $f_1 : Y \rightarrow W \in F_z$ *and* $f_2 : W \rightarrow T_i \in F_z$ **then**
    $D'_{WY} = \Pi_{W \cup Y} D_z$
    $D'_z = D'_z \cup D'_{WY}$
    foreach $T_i \in Y$ **do**
      Add a new triples map m_{T_i} to $r'_k$with T_i as Subject Map
      Transform Object Map T_i in original triples map m_k to a referencing object map with a join to m_{T_i} over W.
    end
  end
  $D'_S = D'_S - Y$
end

Algorithm 1: for decomposing a mapping rule and its data source to 3NF

```
40 foreach T_i ∉ S do
41     if ∃Y ⊂ T such that S ≠ Y and f_1 : Y → S ∈ F_z and f_2 : S → T_i ∈ F_z then
42         D'_SY = Π_{S∪Y} D_z
43         D'_z = D'_z ∪ D'_SY
44         foreach T_i ∈ Y do
45             Add a new triples map m_{T_i} to r'_k with T_i as Subject Map
46             Transform Object Map T_i in original triples map m_k to a referencing object map with a join to
                 m_{T_i} over S.
47         end
48     end
49     D'_S = D'_S − Y
50 end
51 D'_z = D'_z ∪ D'_S
52 return r'_k, D'_z
```

Given a data integration system $DIS_G\langle U, D, M\rangle$, source $D_z \in D$, with sets of attributes $Attr_z$, and mapping rule r_k defined over D_z, with object maps defined over the attributes $T = \{T_1, T_2, ..., T_n\}$ such that $S \subseteq T$ as the subject map and $T \subseteq Attr_z$, our proposed algorithm 1 decompose mapping rule r_k and data source D_z in four different steps. To begin with, it removes any attribute which is not functionally dependant on any other set of attributes (possibly singleton) in the set T, yet it is not part of S. This can be seen in line 15. In addition, each attribute that is partially or transitively dependant on S need to be removed. Line 21 of the algorithm guarantees this point. Moreover, if S itself is involved in the right side of a partial or transitive dependency, then all attributes in the right side must be eliminated. This is depicted in line 30. Finally, there may be a subset of attributes in T remaining which may imply transitively another subset of attributes. This is dealt with in line 41 and basically the set of attributes (possibly singleton) implying S will be removed. The original data source and mapping rule, as well as their decompositions are depicted in figures 4.5 and 4.6 respectively. Applying these four steps, the resulted mapping rule and the data source(s) are in third normal form.

According to the definition of a data integration system and our problem statement mentioned in 4.1, the evaluation of transformed mapping rules M and data sources D must remains exactly the same as before. Therefore, the decomposed mapping rules and data sources based on the above mentioned steps must lead to the same output RDF knowledge graph. First of all, the algorithm ensures that there will be no extra RDF triples produced. To that end, it takes into account that every and each decomposed parent triples map have exactly one term map and that is a subject map, thus the join between two triples map does not lead to any additional data. This guarantees that the parent triples map does not produce any predicate and object with regard to its specified resource. This fact is demonstrated in figure 4.6b. Secondly, joining of the mapping rules, built during the process of algorithm 1, must satisfy the lossless property. Based on the results generated by the algorithm , the property is determined to be assured and it can be seen in figure 4.4. However, we need to prove this theoretically. It is actually the main subject of next section and we will talk about it later. All in all, the RDF knowledge graph produced remains the same as original one yet the output size is reduced.

4.2.4 Lossless Join Property

In order to ensure the correctness of our approach, it is necessary to prove that applying normalization forms on each mapping rule and its corresponding source does not lead to any data loss, whether within data sources or in intermediate results produced by a data integration system. To that end, the proof of correctness of our proposed algorithm is first explained. Then, we show that the resulted mapping rules and data sources are guaranteed to be lossless.

SAMPLE_ID	MUTATION_ID	Gene_CDS_length	Mutation_somatic_status	Mutation_Description
A0000001	32436791	16557	Confirmed somatic variant	Substitution-missense
A0000002	32436791	16557	Confirmed somatic variant	Substitution-missense
A0000001	27470882	1605	Confirmed somatic variant	Substitution-missense
A0000002	27470882	1605	Confirmed somatic variant	Substitution-missense
A0000001	32426297	1911	Confirmed somatic variant	Unknown
A0000002	32426297	1911	Confirmed somatic variant	Unknown
A0000003	32426297	1911	Confirmed somatic variant	Unknown

(a) Original data source

SAMPLE_ID	MUTATION_ID
A0000001	32436791
A0000002	32436791
A0000001	27470882
A0000002	27470882
A0000001	32426297
A0000002	32426297
A0000001	29455299

MUTATION_ID	Gene_CDS_length	Mutation_somatic_status
32436791	16557	Confirmed somatic variant
27470882	1605	Confirmed somatic variant
32426297	1911	Confirmed somatic variant

(b) Normalized data source

Figura 4.5 – **Data source of motivating example decomposed into 3NF using algorithm 1** (a): Original data source as a flat table (b): Decomposition of original data source holding 3NF

```
@base <http://tib.de/ontario/mapping#> .
@prefix ex: <http://example.com/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .

<#MUTATION> a rr:TriplesMap ;
 rml:logicalSource [ rml:source ".//app//source//data//gnomic.csv";
                     rml:referenceFormulation ql:CSV
      ];
 rr:subjectMap [
  rr:template "http://example.com/Mutation/{MUTATION_ID}";
  rr:class ex:mutation_id
 ];
 rr:predicateObjectMap [
  rr:predicate ex:SAMPLE_ID;
  rr:objectMap [
   rml:reference "SAMPLE_ID";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Mutation_somatic_status;
  rr:objectMap [
   rml:reference "Mutation_somatic_status";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Gene_CDS_length;
  rr:objectMap [
   rml:reference "Gene_CDS_length";
  ]
 ].
```

(a) Original mapping rule

```
@base <http://tib.de/ontario/mapping#> .
@prefix ex: <http://example.com/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .

<#MUTATION> a rr:TriplesMap ;
 rml:logicalSource [ rml:source ".//app//source//data//mutation.csv";
                     rml:referenceFormulation ql:CSV
      ];
 rr:subjectMap [
  rr:template "http://example.com/Mutation/{MUTATION_ID}";
  rr:class ex:mutation_id
 ];
 rr:predicateObjectMap [
  rr:predicate ex:SAMPLE_ID;
  rr:objectMap [ rr:joinCondition [ rr:child "MUTATION_ID" ;
                                    rr:parent "MUTATION_ID" ] ;
                 rr:parentTriplesMap <#SAMPLE> ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Mutation_somatic_status;
  rr:objectMap [
   rml:reference "Mutation_somatic_status";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Gene_CDS_length;
  rr:objectMap [
   rml:reference "Gene_CDS_length";
  ]
 ].

<#SAMPLE> a rr:TriplesMap ;
 rml:logicalSource [ rml:referenceFormulation ql:CSV ;
                     rml:source ".//app//source//data//mutation_sample.csv" ] ;
 rr:subjectMap [ rr:template "SAMPLE_ID" ] .
```

(b) Normalized mapping rule

Figura 4.6 – **RML mapping of motivating example decomposed into 3NF using algorithm 1** (a): Original RML mapping rule containing one triples map (b): Decomposition of original RML mapping rule with two triples map holding 3NF and using data sources in 4.5b

3NF Decomposition Of Data Sources

We prove that the resulted data sources form our approach described in 1 are in third normal form. Given the data integration system $DIS_G\langle U, D, M\rangle$ defined in 4.1 and source $D_z \in D$, with sets of attributes $Attr_z$ characterized by set of functional dependencies $F_z \in F$, and mapping rule $r_k : c_j(X, \overline{X}) : -D_z(\overline{X_z})$ defined over D_z. In our proposed approach, two types of relations, i.e., data sources, can be decomposed from the original data source D_z due to normalization process. One is D'_{YW} or D'_{SY} or D'_S that is resulted from having the functional dependencies $Y \rightarrow W$ or $Y \rightarrow S$ and the other one is D'_{YT_i} that is the result of presence of functional dependency $Y \rightarrow T_i$. Next, we prove that the decomposed data sources in the former case[1] is in third normal form. To that end, a contradiction proof is employed.

[1]Proof of D'_{YT_i} being in third normal form can be done in the similar way

Suppose the data source D'_{YW} is not in third normal form, thus there must exist a functional dependency $X \to A$ that make the data source violate third normal form. According definition of normal form for relational databases, X is not a super key for D'_{YW}, A is not a prime attribute and $A \notin X$. Considering $A \in W$, then it must hold $X \subset Y \cup W - \{A\}$. According to our assumption $X \subset Y$ holds as well. Therefore, under all conditions, i.e., $X \subset Y$, $X \subseteq W$ and $X \subset Y \cup W - \{A\}$, it is shown that $Y \to X$. Since $X \to A$ holds according to our assumption, then $Y \to A$ is an extraneous functional dependency. This in turn contradicts to the concept of minimal cover in relational databases. Now by considering $A \in Y$, it holds that $X \subset Y - \{A\} \cup W$. Based on our assumption, Y is a super key since A is not a prime attribute. As a result, there must exist a $Z \subset Y$ for which the functional dependency $Z \to W$ holds. The presence of $Y \to W$ and $Z \to W$ as well as the fact that $Y \to Z$ lead to contradiction to the definition of minimal cover in relational databases. Accordingly, our contradiction assumption is proved to be wrong and as the result the relation, i.e., the data source is in third normal form.

Lossless Join Property Of Data Sources

The proof of the lossless property of decomposed data sources, is based on the demonstration presented by [20]; it inductively demonstrates that the algorithm produces in each iteration a decomposition that is lossless. One example may help clarify this point. Table 4.1 shows the result of application of this algorithm over the motivating example in this thesis. In this table *S_ID*,*M_ID*,*M_Som* and *G_Length* stands for the attributes $SAMPLE\_ID$, $MUTATION\_ID$, $Mutation\_somatic\_status$ and $Gene\_CDS\_length$, respectively, in the data source of genomic mutation. In addition, $MUTATION\_SOM\_GENE$ and $SAMPLE\_MUTATION$ are decompositions of the original data source produced by our proposed algorithm. The table 4.1 proves the lossless-join property in the case of our motivating example that reaches after two iterations. As it can be seen one row of the table, namely $SAMPLE\_MUTATION$'s row, contains only a_i's after second iteration. According to the definition of testing algorithm[20], the decomposition of the original source into these data sources does not lead to loss of any data.

	S_ID	M_ID	M_Som	G_Length
$MUTATION\_SOM\_GENE$	b_{11}	a_2	a_3	a_4
$SAMPLE\_MUTATION$	a_1	a_2	b_{23}	b_{24}

(a) Initial state of the table

	S_ID	M_ID	M_Som	G_Length
$MUTATION\_SOM\_GENE$	b_{11}	a_2	a_3	a_4
$SAMPLE\_MUTATION$	a_1	a_2	a_3	a_4

(b) Modified state of the table after one iteration

Tabella 4.1 – **Results generated after two iteration.** After the second iteration, it is shown that the decomposition is lossless.

The 3NF Decomposition Of RML Mapping Rules

Like the data sources, it is necessary to prove that our proposed algorithm decompose the mapping rules within a data integration system into the third normal form with respect to its defined set of functional dependencies. Given the data integration system $DIS_G\langle U, D, M\rangle$ defined in 4.1 and source $D_z \in D$, with sets of attributes $Attr_z$ characterized by set of functional

dependencies $F_z \in F$, and mapping rule $r_k : c_j(X, \overline{X}) : -D_z(\overline{X_z})$ defined over D_z. Our algorithm transforms a mapping rule r_k with one triples map into a mapping rule r'_k composed of one child triples map and several parent triples maps in order to achieve the goal of normalization up to 3NF. Therefore we need to prove the correctness of our algorithm in the recent two cases. In fact, generated triples maps m_{T_i} are always composed of exactly one term map and that is subject map. The subject map is always referring to one attribute, thus it can not indicate any violations of any forms. However, the proof must be done for the transformation of an existing triples map to a child triples map according to functional dependency $S \rightarrow O$, where $O \subseteq T$ and it refers to attributes used in object maps. We support this position by using contradiction. Consider that the child triples map is not in third normal form. Therefore, we suppose that there exist functional dependencies $Y \rightarrow T_i$ where $T_i \notin Y$ and Y is not the subject, i.e., S. The complete proof is concerned about two cases: $(1)T_i \in S$ $(2)T_i \in O$. In both cases, due to presence of $Y \rightarrow T_i$, it can be concluded that $S \rightarrow T_i$ is redundant. However, in the first case $S \rightarrow T_i$ is a trivial dependency. This contradicts to the fact that minimal cover does not include redundant dependencies. To conclude, all of the transformed triples maps are in third normal form.

Lossless Join Property Of Mapping Rules

In this section, it is proven that the decomposition of a mapping rule into several mapping rules with respect to set of functional dependencies over its used attributes ensures that the same knowledge graph is created as before. In order to prove that, we rely on the definition of lossless joins for relational databases [20] and we ground the lossless property of transformed rules.

Given the data integration system $DIS_G\langle U, D, M\rangle$ defined in 4.1 and source $D_z \in D$, with sets of attributes $Attr_z$ characterized by set of functional dependencies $F_z \in F$, and mapping rule $r_k : c_j(X, \overline{X}) : -D_z(\overline{X_z})$ defined over D_z. For a lossless decomposition $Attr'_z = \{Attr'_{z1}, Attr'_{z2}, ..., Attr'_{zn}\}$ resulted from normalization of D_z with respect to set of functional dependencies F_z defined over its set of attributes $Attr_z$ such that:

- $Attr'_{zg} \subset Attr_z, and\ 1 \leq g \leq n$
- $Attr_z = \cup_{g=1}^{n} Attr'_{zg}$
- $D_z \ = \bowtie_{g=1}^{n} \Pi_{Attr'_{zg}} D_z$
- $D'_z = \{D'^{Attr'_{zg}}_{zg} \mid D'^{Attr'_{zg}}_{zg} = \Pi_{Attr'_{zg}} D_z \ , and\ 1 \leq g \leq n\}$

our approach replaces the source D_z in D' with all D'_{zg}'s, $1 \leq g \leq n$. In addition, the rule r'_k in M' , the transformed of r_k, is defined as following:

$$r'_k : c_j(X, \overline{X}) : -D'_{z1}(\overline{X'_{z1}}), D'_{z2}(\overline{X'_{z2}}), ..., D'_{zn}(\overline{X'_{zn}}).$$

Now suppose $D'_{z1}(\overline{X'_{z1}})$ as the only source used in the only child triples map existing in r'_k after normalization and also set $\overline{X_H}$ as the set of all variables existing in the head of the rule, i.e., variables in $\overline{X}$ and X. Then the rule r'_k meets also the following condition for each $\overline{X'_{zg}}$, $1 \leq g \leq n$:

$$\overline{X'_{zg}} = \begin{cases} \{(a_{i,zg}, X_{i,zg}) \mid (a_{i,zg}, X_{i,zg}) \in \overline{X_H}\ and \\ \qquad a_{i,zg} \in Attr_{zg}\} & g = 1 \\ \{(a_{i,zg}, X_{i,zg}) \mid (a_{i,zg}, X_{i,zg}) \in \overline{X_H}\ and \\ \qquad a_{i,zg} \in Attr_{zg}\ and \\ \qquad a_{i,zg} \notin (Attr_{zg} \cap Attr_{z1})\} & 1 \leq g \end{cases}$$

and also for all attributes $a'_i \in (Attr_{zg} \cap Attr_{z1})$, our approach adds new variables that exist only in the body of the rule, i.e., for joining different data sources, as follows:

$$\overline{X'_{zg}} = \overline{X'_{zg}} \cup \{(a_{i,zg}, X_{i,zg}) \mid (a_{i,zg}, X_{i,zg}) \notin \overline{X_H} \; and \; a_{i,zg} \in (Attr_{zg} \cap Attr_{z1})\}$$

Accordingly, our approach guarantees that the generated RDF graph by executing $RDFize(DIS'_G \langle U, D', M' \rangle)$ remains the same as that of $RDFize(DIS_G \langle U, D, M \rangle)$, while in the former the execution time is improved and the redundancies are removed. We ground lossless property of approach with respect to the above mentioned conditions:

- According to the lossless property definition, the transformed rule r'_k contains the same attributes as in $Attr_z$ and the join of all the data sources D'_{zg} leads to the same tuples existing in D_z prior to decomposition.
- In order to join the sources D'_{zg}, new variables are added to the body of rule r'_k. These are not appearing in the head of the rule. Thus the rule can transform all values which were accessible prior to transformation, i.e., by employing rule r_k, and head of the rule remains the same as before too.
- The only variables added to r'_k are those for joining different sources. Therefore, our approach does not force any extra data in the output.

4.3 Summary Of The Chapter

The aim of this chapter was to explain the existing problem as well as new theories and definitions. In addition, an algorithm for tackling the defined problem in this section was presented. Finally, it has been proven that transformation of mapping rules and data sources into normal forms are lossless.

Capitolo 5

Implementation

In this chapter, an implementation of the algorithm, described in the last chapter, will be presented. This is done in Python[1] language version 3.7 and it is tailored for the RML[2] mapping language. In fact, it is called *RML-Normalizer*. In the rest of this chapter, first, the inputs' formats to the implementation and the utilized python libraries are presented and then RML-Normalizer and its implementation will be described.

5.1 Input Formats

On the basis of our approach, the implementation receives three inputs namely data source, mapping rule, and set of functional dependencies. Next, the specific formats of inputs to this version of implementation are explained.

5.1.1 Data Source

In the current implementation, only CSV files as data sources can be received. Since an RML mapping rule reads the data within a data source, the input data source is therefore extracted from inside an RML mapping rule. It is done using the *logicalSource* of a triples map inside the mapping rule.

5.1.2 Mapping Rule

As it is mentioned before, this implementation is specifically done for the purpose of transforming RML mapping rules. These mapping rules are given as input with the Turtle[3] serialization.

5.1.3 Set of functional dependencies as Input

Functional dependencies are the main part of our algorithm to decompose mapping rules and data sources. In order to have a unified format with most of the other resources within the data integration system as well as to keep track of the dependencies even in our knowledge graph, we decided to have the input set of functional dependencies in turtle format. Having different data sources with different sets of functional dependencies over them, this format can be very helpful

[1] `https://www.python.org/`
[2] `https://rml.io/`
[3] `https://www.w3.org/TR/turtle/`

```
@prefix fd: <http://example-fd-set.com/> .
<#Gnomic> fd:key [fd:column_name "{SAMPLE_ID,MUTATION_ID}";
                  fd:determine [fd:column_name "MUTATION_ID";
                                fd:dependant "{SAMPLE_ID,MUTATION_ID}";
                                ];
                  fd:determine [fd:column_name "SAMPLE_ID";
                                fd:dependant "{SAMPLE_ID,MUTATION_ID}";
                                ];
                  fd:determine [fd:column_name "Mutation_somatic_status";
                                fd:dependant "{Mutation_Description}";
                                ];
                  fd:determine [fd:column_name "Gene_CDS_length";
                                fd:dependant "{MUTATION_ID}"
                                ];
                  fd:determine [fd:column_name "Mutation_Description";
                                fd:dependant "{MUTATION_ID}"
                                ]
                  ].
```

Figura 5.1 – **An example of a FD set as input to RML-Noramlizer**

later to discover new patterns. This format can be introduced later as new vocabulary and it can extend the RML mapping language as a real-world example. This can be seen in figure 5.1.

5.2 Python Libraries

In order to implement the RML-Normalizer, two popular libraries are used, and mainly the function of RML-Normalizer is based on these two libraries. First of all, the RDFlib[4] library version 5.0.0 is used to load, parse, and edit an existing RML mapping rule. To illustrate it, this library is used in the first place along with a SPARQL query over the content of an RML mapping rule to read and parse the content. The result is then processed to remove the undesired attributes from the input mapping rule. After that, the library is used to add new triples maps into the mapping rule. In addition, the pandas [5] version 1.0.3 is used to read CSV data sources as well as to edit them. With regard to this, the RML-Normalizer takes the benefits of the Pandas library to project out the attributes and extensions of data sources into new data sources. Needless to say, the duplicate rows within the extensions will be removed by using pandas.

5.3 RML-Normalizer

RML-Normalizer is a specific implementation of our approach to deal with RML mapping rules and their input sources. However, our approach can be considered as a basis for processing other types of mapping rules. As a matter of fact, RML is a language for integrating heterogeneous data sources based on different formats, e.g., CSV, JSON, and XML. RML-Normalizer is publicly available as a resource in Github [6].

[4] https://rdflib.dev/
[5] https://pandas.pydata.org/
[6] https://github.com/SDM-TIB/rml-normalizer

```
new_rml_graph.add((new_subject,r2rml['template'],new_subject_term))
object_map_node=new_rml_graph.value(subject=None,predicate=rml['reference'],object=Literal(obj))
    new_rml_graph.add((object_map_node,r2rml['parentTriplesMap'],new_triplesmap_uri))
    for i in join_condition:
        new_rml_graph.add((object_map_node,r2rml['joinCondition'],new_join_condition))
        new_rml_graph.add((new_join_condition,r2rml['child'],Literal(i)))
        new_rml_graph.add((new_join_condition,r2rml['parent'],Literal(i)))
    new_rml_graph.remove((object_map_node,rml['reference'],Literal(obj)))
```

Figura 5.2 – **An example of RML-Noramlizer implementation for decomposing RML mapping rules** Parent triples maps are decomposed from original RML mapping rule and then join conditions are added

```
df_new_source=pandas.read_csv(triples_map.data_source,usecols=object_list_on_sub)
df_new_source.drop_duplicates(keep='first',inplace=True)
df_new_source=df_new_source[object_list_on_sub]
df_new_source.to_csv(v_main_dir+child_new_source_file,index=False)
```

Figura 5.3 – **An example of RML-Noramlizer implementation for decomposing data sources** Decomposition and duplicate eliminations are performed in this portion.

5.3.1 Implementation Of RML-Normalizer

As it is mentioned above, the current implementation of RML-Normalizer relies on RML mapping rules as input mapping rules and CSV files as input data sources. The former is read by the RDFlib library and it will be decomposed possibly into new triples maps. To this end, RML-Normalizer uses this library to parse the mapping rule into its elements, e.g., Subject Map, Object Map, Logical Source and etc. These are used in turn to find the attributes made violations of normal forms. In other words, the library is used to traverse the set of functional dependencies and to find the redundant dependencies, thus remove the relevant attributes from the mapping rule. Having the relevant attributes, RML-Normalizer takes advantage of editing the parsed graph, by RDFlib, to add new triples maps and to transform existing *Logical References* into *Referencing Object Maps*. Figure 5.2 show portion of the code meant to perform recent function. Normalizing of data sources via pandas is performed in three main steps and after reading the data source from within the content of input RML mapping rule. To start with, the undesired attributes will be removed from the original data source to a new data source holding normal forms, e.g., 2NF and 3NF. In addition to removing this attribute, RML-Normalizer attaches the join condition(s) to the set of attributes in a new data source. Another step is to project out attributes that are fully and directly dependant on the subject map. Last but not least, the duplicates in each of the decomposed data sources are dropped. This is clearly shown in the figure 5.3.

5.4 Summary Of The Chapter

An implementation of our approach tailored for RML mapping rules and CSV data sources is presented. We also described the inputs to this implementation as well as the programming libraries used in python language to achieve our goal.

Capitolo 6

Experimental Evaluation

To investigate the bright and dark side of our work, several experiments are specified and performed. These experiments aim at evaluating the performance of a knowledge graph creation pipeline and to compare data space usage of original resources versus the normalized ones based on our approach. To that end, the research questions to be answered in this work are formulated as following: **RQ1:** *What is the impact of the proposed approach on space savings?* **RQ2:** *What is the impact of the approach on the knowledge graph creation?* The rest of this work is organized as follows: First, the testbed generation and generated testbeds are described. After that, the experimental setups are explained. Finally, the results and their corresponding analysis will be depicted.

6.1 Testbed Generation

There are intense demands to discover the aspects in which each state-of-the-art tool and technology need to be developed. Proposing new tools and technologies requires these aspects to be precise. Therefore the existence of data is of great importance for studying the bright side and dark side of every existing technology and showing the great value of new works. Similarly, having testbeds for evaluating different tools in the area of data integration systems can be very helpful to *Semantic Web Community*. This thesis also includes implementing a tool for generating testbeds based on the configuration of functional dependencies set related to relational databases. This tool is developed to run different experiments in this work. However, this can be used and extended to generate testbeds for other related experiments and study the shortcomings of current works.

This tool is implemented during this thesis to provide us with data sources to prove our work's impacts on existing data integration systems. Our testbed tool is publicly available as a resource in Github [1]. This tool's basic idea is originating from the concept of functional dependencies between different attributes in a set. Each an attribute in a set can be functionally dependant upon another subset (possibly singleton) of the same set [6, 7]. According to this, one can explicitly determine whether there exists a relation between two single attributes or not. However, the same configuration can be applied implicitly to sets of attributes. By utilizing *FDTool* [25][2], one can ensure that desired functional dependencies holds within a set of attributes. This, in turn, confirms that our testbed generation tool works well according to the given configurations, and the data sources are holding certain types of functional dependencies.

[1] `https://github.com/SDM-TIB/rml-normalizer`
[2] `https://github.com/mburanosky17/FDTool/tree/v0.1.7`

```
{NBA_ID} → {Name}
{NBA_ID} → {player_position}
{NBA_ID} → {player_nationality}
{NBA_ID} → {team_id}
{team_id} → {team_name}
{team_id} → {league_champs}
```

Listing 6.1 – A sample set of functional dependencies for generating testbed

Our testbed generation tool is implemented in Python 3.7, and basically, its main point is a recursive function. To feed the tool, we use a configuration file, i.e., INI file, in which each level acts as one attribute of the output data source. Figure 6.1 depicts one example structure of this configuration file based on the functional dependencies in 6.1. The number of distinct attributes within the set of dependencies equals the number of levels in this configuration file. According to this structure, *level7* is representative for the attribute NBA_ID and the four preceding levels are presenting attributes which are directly dependant on that attribute, i.e., $Name$, *player_position*, *player_nationality* and *team_id* exactly in this order. In other words, *level7* is key to the data source to be generated. However, *level1* and *level2* are related to attributes *team_name* and *league_champs*. Our tool performs recursive calls based on these levels starting from *level*1 to create a tuple of N-arity, in this case, 7-arity, in each row of the output data source.

This configuration file is presenting much more detail than just a set of attributes. Each level of the configuration file has at least four properties that are used in order to reach the goal of having a data source holding certain functional dependencies. To start with, *domain_cardinality* shows the number of distinct values existing in each level, i.e., for each attribute. Therefore, the value of this property in the last level, in this case *level*7, shows the total number of tuples to be generated. Property *number_of_distinct_vals* is to specify the number of times each previously generated value(s), in an iteration, is duplicated. For instance, a value generated in one iteration in *level*1 is repeated 2 times with regard to *level*2 and respectively values, i.e., tuple, generated in the same iteration by *level*1 and *level*2 is repeated 5 times with regard to *level*3.

The property *is_parent* determines whether a specific attribute implies its preceding levels or not. In other words, it shows if in a new iteration the same values as the previous iteration should be used or not. This actually makes sure that there will be some certain functional dependencies between a level and its preceding levels. For instance, the value of corresponding property in *level*3, i.e., $True$, ensures existence of the last two functional dependencies mentioned in list6.1 within data. In a sense, this property also guarantees that there will be no undesired functional dependency between some levels. For example, every value in *level*1 will be duplicated for each value in the *level*2. Accordingly, there will be an N-to-N relationship between these two attributes hence no functional dependency between them, as it is in list 6.1. In a similar way, there will be no functional dependencies between *level*4, *level*5, and *level*6. Moreover, the property *total_vals* refers to the number of times that a value of an attribute B is repeated for a value of another attribute A, holding the functional dependency $B \rightarrow A$ in the minimal cover. The exception is the last level in which this property determines total number of tuples to be generated.

```
[level1]
is_parent = False
number_of_distinct_vals = 2
total_vals = 20
domain_cardinality = 2

[level2]
is_parent = False
number_of_distinct_vals = 2
total_vals = 10
domain_cardinality = 2

[level3]
is_parent = True
number_of_distinct_vals = 5
total_vals = 5
domain_cardinality = 20

[level4]
is_parent = False
number_of_distinct_vals = 2
total_vals = 400
domain_cardinality = 2

[level5]
is_parent = False
number_of_distinct_vals = 4
total_vals = 200
domain_cardinality = 4

[level6]
is_parent = False
number_of_distinct_vals = 25
total_vals = 50
domain_cardinality = 25

[level7]
is_parent = True
number_of_distinct_vals = 2
total_vals = 8000
domain_cardinality = 8000
```

Figura 6.1 – **An example configuration file to generate data according to set of functional dependencies stated in 6.1**

6.2 RML Interpreters

It is generally agreed today that evaluating results of proposed approaches as well as existing approaches via different tools not only exposes the pros and cons of an intended approach but also proves the presence of those pros and cons. We run three different RML interpreter engines over our proposed solution, i.e., the normalized RML mapping rules and their data sources. SDMrdfizer [3] [3], RMLMapper[4] and RocketRML [5] [26] are three interpreters used in these experiments. Finally, our evaluations for all of these tools are shown. However, the analogy of these tools is beyond the work of this thesis.

6.3 Experimental Configurations

A necessary set of experimental configurations needs to be set in a proper way to run the desired experiments. Some of these configurations include the environment, e.g., operating system, hardware, e.g., CPU and memory usage, input data, in our case, data source, RML mapping rules, and set of functional dependency. Next, the testbeds and their characteristics are described. After that, the input RML mapping rules are presented. Then, we will show which parameters

[3] https://github.com/SDM-TIB/SDM-RDFizer/tree/v3.2
[4] https://github.com/RMLio/rmlmapper-java
[5] https://github.com/semantifyit/RocketRML

play essential roles in our proposed solution and explain which of them are used, and we reveal which metrics are defined to evaluate the effects of these parameters. Finally, the evaluations, i.e., results of experiments, are demonstrated.

6.3.1 Datasets

To evaluate the results produced by our solution, two different types of testbeds are used. First of all, a real-world data source is used which is mentioned all over this thesis as a motivating or running example. Secondly, we take advantage of our novel testbed generation tool and produce synthetic data to investigate additionally those aspects which are not available in the real-world testbed.

Real-World Dataset

Like in any other area of knowledge, the importance of using real-world data in the semantic web is noticeable to almost everyone. In the first place, it provides us with real-world issues and complexities in data, thus we can discover the strengths and weaknesses of a work. In the second place, the abundance of these types of data will bring us diversity in structures and data source configurations. As a result, there will be no need to generate data synthetically hence there will be most probably a great time-saving in developing new ideas and state-of-the-art technologies.

In our work, the running example is used to presents the way our approach solves the problem of defining a new data integration system described in 4.1 and also to show the effectiveness of our approach. The dataset is a subset of somatic mutations data provided by *COSMIC*[6] . Subset of attributes used for our experiments includes $SAMPLE\_ID$, $MUTATION\_ID$, $Gene\_CDS\_length$, $Mutation\_somatic\_status$ and $Mutation\_Description$. This dataset is characterized with a set of functional dependency upon it as mentioned in 1.1 and it contains 50M records. However, according to parameters of our experiments and based on different experiments only some parts of the data is used. A combination of $SAMPLE\_ID$ and $MUTATION\_ID$ is the key of the dataset and every other data item in this dataset is repeating for values in $MUTATION\_ID$ with different rates hence for the key. Table 6.1 shows the total number of distinct values in each domain.

SAMPLE_ID	MUTATION_ID	Gene_CDS_length	Mutation_somatic_status	Mutation_Description
100000	500	415	15	8

Tabella 6.1 – Number of distinct values in each domain within dataset of genomic data

[6] `https://cancer.sanger.ac.uk/cosmic/download` according to version v91 released $07^{th} April 2020$

```
{NBA_Player_ID} → {Team_ID}
{Team_ID} → {Founded}
{Team_ID} → {Division}
{Team_ID} → {Champs_Of}
```

Listing 6.2 – List of functional dependencies holding in synthetic datasets

Synthetic Dataset

Synthetic methods are alternative options for empirical evaluations of state-of-the-art and newly proposed technologies, whenever differently structured data are not publicly available or the variety of data is probably not high enough. Therefore, the synthetic data play a vital role, whenever several different configurations are to be examined. However, synthetically generated data need to be checked to ensure if they are satisfying the desired configuration. Moreover, one needs to be sure that the same data are reproducible later. Otherwise, the results can not be reproduced and the evaluations are useless for a community. In addition, the synthetic data may need to follow some real-world restrictions. For example, a generated relational data source may need to hold a set of functional dependencies. All in all, synthetic data is of great importance, since it can clarify each and every strength point and shortcomings in a work.

We rely on our testbed generation tool and generate different datasets according to the aims of our experiments. The generated testbed includes three data sets with 4M, 8M, and 16M records each. These datasets hold the set of functional dependencies over five attributes shown in listing 6.2. Each of the attributes is given different domains and a different number of unique values similar to the real-world data. Since there are needs to perform experiments for parameters under study, it does not suffice only to consider different numbers of unique values for each experiment. To explain it, it is necessary to consider all numbers fixed and just change the diversity of unique values corresponding to attribute $Team\_ID$, since it is the joining condition for our mapping rules hence specifies join selectivity. Table 6.2 shows clearly the fact mentioned above.

NBA_Player_ID	Team_ID	Founded	Division	Champs_of
4M	40	5	2	2
	80			
	160			
	400			
	800			
	2K			
	4K			
	8K			
	16K			
	40K			
8M	80	5	2	2
16M	160	5	2	2

Tabella 6.2 – Number of distinct values in each domain within synthetic datasets

```
@base <http://tib.de/ontario/mapping#> .
@prefix ex: <http://example.com/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .

<#NBA_P>
 rml:logicalSource [ rml:source "//app//nba_players.csv";
                     rml:referenceFormulation ql:CSV
      ];
 rr:subjectMap [
  rr:template "http://example.com/resource//{NBA_Player_ID}";
  rr:class ex:Basketballplayer
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Team_ID;
  rr:objectMap [
   rml:reference "Team_ID";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Founded_in_year;
  rr:objectMap [
   rml:reference "Founded";
  ]
 ].
```

(a) Original mapping rule violating 3NF

```
@base <http://tib.de/ontario/mapping#> .
@prefix ex: <http://example.com/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .

<#NBA_P>
 rml:logicalSource [ rml:source "//app//nba_players.csv";
                     rml:referenceFormulation ql:CSV
      ];
 rr:subjectMap [
  rr:template "http://example.com/resource//{NBA_Player_ID}";
  rr:class ex:Basketballplayer
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Team_ID;
  rr:objectMap [
   rml:reference "Team_ID";
  ]
 ];
 rr:predicateObjectMap
  rr:predicate ex:Founded_in_year;
  rr:objectMap [
   rr:parentTriplesMap <#Team_Foundation>;
        rr:joinCondition [
            rr:child "team_id";
            rr:parent "team_id";
        ];
  ]
 .

<#Team_Foundation>
 rml:logicalSource [ rml:source "//app//nba_teams.csv";
                     rml:referenceFormulation ql:CSV ];
 rr:subjectMap [
  rr:template "{Founded}"
 ].
```

(b) Normalized mapping rule in 3NF

Figura 6.2 – **RML mapping rules over synthetic data characterized with set of functional dependencies described in listing 6.2** (a): Original RML mapping rule containing two object maps violating 3NF (b): Normalized RML mapping rule with one join condition holding 3NF

6.3.2 RML Mapping Rules

Mapping rules are part of the defined problem in 4.1 and they have a great effect on the final results. With regard to this, we need to take into account different mapping rules based on different sets of functional dependencies and it is essential that there exist different kinds of violations as to normal forms. Thus the evaluation results make sense regarding the proposed solution. In addition to the RML mapping rule available in our running example, which is violating 2NF, a mapping rule violating 3NF is applied over the testbeds. Figure 6.2 depicts recent RML mapping rule. In fact, this mapping rule is considering the key of the data sources as the subject and any other attributes act as object maps in this rule. To study different behaviors of our solution we need to consider a different number of joins as well. In other words, it can be of great importance to run experiments with different mapping rules in terms of the number of object maps.

6.3.3 Experimental Parameters

Without any doubt, it is necessary to indicate parameters that affect the behavior of a system or process which is under study. Chaves-Fraga et al. [27] explain in his work the parameters that can be effective during the knowledge graph creation process, from mapping to source to output and platform. Like any other novel works, we identify parameters that play important rules in the evaluation of a data integration system and knowledge graph creation process as

	Experimental Parameter
1	Data volume
2	Cardinality of attributes
3	Number of transitive dependencies

Tabella 6.3 – **Experimental parameters with regard to evaluation of RML-Normalizer**

well as future query processing over the resulted RDF knowledge graph. These parameters are in correlation with our solution based on the normalization of mapping rules and data sources.

The parameters in our work which mainly affect the evaluation of a data integration system defined in 4.1 include those originating from transforming mapping rules as well as their corresponding data source. These parameters can be seen in table 6.3. Firstly, the *data volume* refers to the number of tuples existing in a dataset utilized by a mapping rule during the process of data integration. Secondly, *cardinality of attributes* expresses the number of unique values in one domain attached to the number of unique values in another domain. This of course can be very informative within the process, since normalization leads to decomposing of data sources and mapping rules. Finally, the decomposition results in having different joins to different mapping rules in order to approve the completeness of the output RDF knowledge graph. This may in turn affects the execution time of a knowledge graph creation pipeline. For that purpose, we consider also *number of transitive dependencies*.

6.3.4 Metrics

As in any other research work in the community, to evaluate the goals of our work, there are needs to define metrics to observe the end results and analyze them. In our work we defined three metrics shown in table 6.4. Accordingly, definition of each metric are as follows:

- *Space:* Size of the results generated by using an approach including intermediate and end results.

- *Execution time:* The time needed for the construction of a complete knowledge graph by executing RML mapping rule(s) over its corresponding CSV data source(s). Needless to say, the normalization time is considered in execution times of experiments, when it comes to normalized versions of RML mapping rules and data sources.

- *Completeness:* Output RDF knowledge graph by the execution of an RML mapping rule over its corresponding CSV data source.

In fact, normalization may reduce the size of intermediate results leading to occupying fewer spaces. In addition, it causes the elimination of duplicated output RDF triples. As a result of these, the construction time of a knowledge graph can be influenced. Therefore the execution time of a knowledge graph creation process needs to be observed. However, normalization leads to the decomposition of rules and data sources, thus the execution of joins between rules and data sources impacts the execution time too. This can be another explanation for observing the execution time. Last but not least, these joins must present the same RDF graph as the original RDF graph. To that end, it is necessary to check the completeness of the RDF graph resulted from our solution.

	Experimental Metric
1	Execution time
2	Space saving
3	Completeness

Tabella 6.4 – **Experimental metrics to measure efficiency and correctness of RML-Normalizer**

6.3.5 Environment Settings

In this thesis, we utilize the first version of RML-Normalizer, implemented in Python 3.7, to normalize RML mapping rules and data sources. Intended experiments are carried out on a Ubuntu 18.04.3 LTS 64 bit machine with two Intel(R) Xeon(R) Platinum 8160 2.10 GHz CPUs and CPU cores sum up to 96 with 2 threads each. The memory settings installed on the machine are 755 GiB DDR4 RAM. Moreover, to run experiments we used Docker version 19.03.4 with python:3.7 as the template image for building a docker image. Additionally, the docker's resources are limited in terms of numbers of CPUs and used Memory to 20 and 50Gb respectively.

6.4 Evaluations

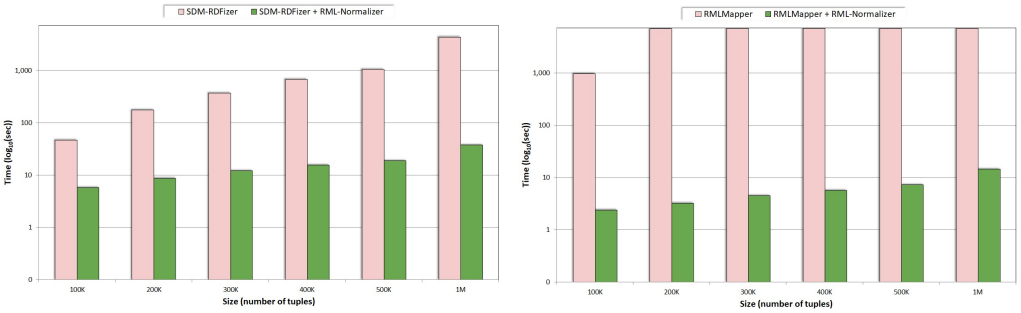
In this work and according to experimental parameters, three different experiments are executed. In the next sections, each of these experiments with the produced results as well as their impact on experimental metrics are explained.

6.4.1 Different size data sources

In this section, two experiments and their results are explained in which RML mapping rules shown in 4.6, i.e., running example, and 6.2 are executed using different RDFizer engines. The former is violating the second normal form of mapping rules and the latter is violating the third normal form. In addition, the structure of two mapping rules differs in that the former has part of the key from the data source as its subject map and part of it as its object map, while in the latter the key itself is the subject map in the mapping rule. As mentioned before these mapping rules are characterized by two lists of functional dependencies, namely listing 4.1 and 6.2. Moreover, the volume dimension of data sources plays a great role in these experiments. In the first experiments shown in figure 6.3a, interpreting of RML mapping rules are over several subsets of the whole dataset namely 100K, 200K, 300K, 400K, 500K, and 1M tuples. However, the second experiment includes three datasets of 4M, 8M, and 16M tuples. Last but not least, in both of the datasets cardinality as well as the number of joins remains unchanged all over a specific experiment. They both include 1 join, however, the former has a cardinality of 10% and the latter cardinality of 2.5%. In all the experiments, a timeout of 7,200,000 milliseconds is considered.

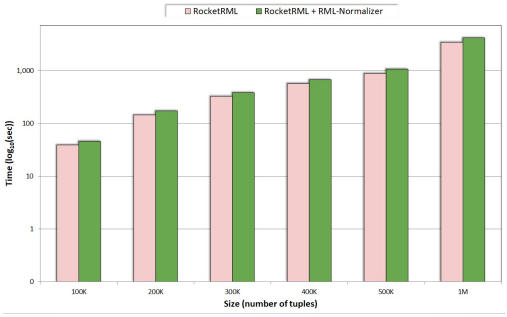
Execution Time

With regard to our metrics and specifically execution time, it is generally reduced, with some cases showing considerable effects of normalization. It is clear in figures 6.3a and 6.3b that the execution time of knowledge graph creation process is decreased significantly in all 6 executions albeit normalization process. The time is improved by more than 95% in all of the cases except



(a) SDM-RDFizer

(b) RMLMapper

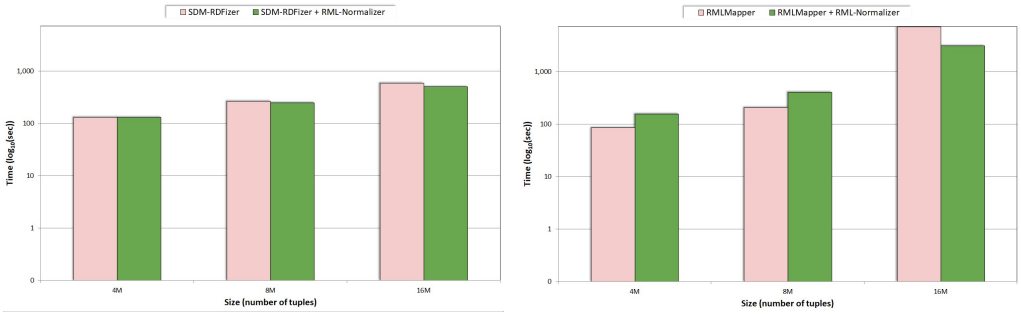


(c) RocketRML

Figura 6.3 – **Performance of different RDFizer engines with and without using RML-Normalizer over genomic mapping rules and data sources violating 2NF** Total execution times, (normalization time is included if any), needed by each RDFizer engine to finish the assigned tasks based on different data volumes.

for data source with 100K tuples integrated by SDM-rdfizer whose improvement is almost 87%. Although RMLMapper could not finish the tasks in most of the original cases and could not produce results, RML-Normalizer improved the execution time of RMLMapper nearly 99.9% in all cases. Considering RocketRML, there are different behaviors with respect to normalization. Figure 6.3c depicts that RocketRML offers no improvement in the case of normalized mapping rules and data sources, though the differences in times are not significant, with 300K data size the lowest execution time increment. Although RocketRML is finishing all of its assigned tasks in the specified time, its resulted RDF graphs are always incomplete and only contain triples generated from child triples map. According to the above-mentioned figures, the execution time of the knowledge graph creation process is increased slightly as the size of data source increases, with RocketRML with the highest time increment among all. While RocketRML presents unknown behaviors regarding different sizes of data, the above-mentioned analysis may explain different implementations of join operators within all different engines.

It can be seen in Figure 6.4a that RML-Normalizer can improve the processing time of SDM-RDFizer[3] always compared to using original RML mapping rules and data source. Although the improvement in the case of 4M tuples is relatively low and near to zero, in other cases up to 15% performance improvement can be seen. However, the performance of RMLMapper in the first two cases is highly reduced by applying RML-Noramlizer. This is shown in figure 6.4b. Actually, RMLMapper takes almost 60 and 180 seconds more to finish the whole tasks of 4M and 8M tuples respectively. Nevertheless, it can be seen that RML-Normalizer helps RMLMapper



(a) SDM-RDFizer (b) RMLMapper

Figura 6.4 – **Performance of RDFizer engines with and without using RML-Normalizer over synthetic mapping rules and data sources violating 3NF** Total execution times, (normalization time is included if any), needed by each RDFizer engine to finish the assigned tasks based on different data volumes.

to finish the given task of 16M at the specified time limit which is not the case for the original RML mapping rule and data source. It is observed in experiments that this recent execution needs nearly less than half of the time for the normalized version compared to the original one, which is incomplete in terms of execution.

It is observed in 6.4 that there are less performance improvements regarding the normalized version compared to the results shown in figure 6.3. Although the performance of SDM-RDFizer and RMLMapper is improved in general, RocketRML reached time out in almost all cases including original and normalized versions hence no RDF graph is produced in this case. This can be described by the large size of datasets. Compared to the results depicted in figure 6.3, improvements in times is almost lower for two reasons. For one thing, this is caused by different cardinalities between two-parent triples maps and child triples maps in genomic data versus synthetic data. In the former, the child triples map is much more selective than the parent triples map, while in the latter it is contrariwise. Another thing is the former offers a cardinality, i.e., join selectivity of 10% while the latter offers 2.5%. Overall, creating knowledge graphs from big data having large volumes performs much more faster in presence of RML-Normalizer.

Space Savings

Tables 6.5 and 6.6 show the amount of space saved by using RML-Normalizer divided into genomic data sources and synthetic data sources respectively. It is clear in table 6.5 that sizes

Data source volume	Original size (in MB)	Size via RML-Normalizer (in MB)
100K	5.9	1.8
200K	12	3.6
300K	18.2	5.7
400K	24.3	7.6
500K	30.4	9.5
1M	59.8	18

Tabella 6.5 – **Size of genomic data sources before and after normalizing via RML-Normalizer;** Total size of data source file(s) in MB.

Data source volume	Original size (in MB)	Size via RML-Normalizer (in MB)
4M	175.5	68.7
8M	351.0	137.3
16M	707.7	280.4

Tabella 6.6 – **Size of synthetic data sources before and after normalizing via RML-Normalizer;** Total size of data source file(s) in MB.

of data sources are reduced to almost one-quarter of the original size. Since the cardinality is the same over every genomic data set, it can be expected that the ratio of total normalized data sources to the original data source with 100K tuples is less than the same number in data source 1M tuples. Overall, there is an almost 70% reduction in sizes of data sources after using RML-Normalizer. Although synthetic data volumes demonstrated in Table 6.6 are much more larger than the genomic data sources, the sizes of data sources are decreased approximately 61% after applying RML-Normalizer which is slightly less than the same number for genomic data sources. This can be observed in table 6.6 clearly. This difference can be justified by different cardinalities between decomposed sources. For instance, within the data source of 4M cardinality between the sources is 1-100K while in the genomic data source of 100K it is 10-10k. Needless to say, the ratio of high cardinality column to the whole tuples is greater in genomic data source compared to one in the synthetic data source, with 10% and 2.5% respectively. In addition, the structure of RML mapping rules along with the configuration of functional dependencies play significant roles. Considering genomic data sources, most of the attributes appear in the same data source where the low cardinality column appears, whereas in synthetic data sources attributes are equally distributed.

It is obvious according to our problem that the redundancies not only involve the data source but also the resulted RDF graphs. With regard to this, Table 6.7 depicts a dramatic reduction in number of generated RDF triples. It can be observed that in all experiments number of output RDF triples is reduced to nearly 25% of the originally generated RDF triples. This is happening because $MUTATION\_ID$ is part of the key in the data source, thus it is repeating for each value of $SAMPLE\_ID$, which is the other attribute in the key. In addition, every dependency on $MUTATION\_ID$ is repeating for the whole data source due to partial dependency. By normalizing the mapping rules and data sources, unnecessary repeating of $MUTATION\_ID$ will be avoided. Therefore, all of its dependencies are repeating to the number of unique values in $MUTATION\_ID$. The normalization causes no loss of data and this will be shown empirically later. The numbers of RDF triples resulted from synthetic data and RML mapping rules over

Data source volume	#of original RDF triples	#of RDF triples via RML-Normalizer
100K	400,000	100,030
200K	800,000	200,030
300K	1,200,000	300,030
400K	1,600,000	400,030
500K	2,000,000	500,030
1M	4,000,000	1,000,030

Tabella 6.7 – **Size of RDF graphs generated from genomic data before and after normalizing via RML-Normalizer;** Total number of generated RDF triples divided by data volume. RML-Normalizer eliminate duplicates within data.

Data source volume	Original size (in MB)	Size via RML-Normalizer (in MB)
100K	41	9
200K	81	17
300K	121	25
400K	161	33
500K	201	42
1M	401	82

Tabella 6.8 – **Size of RDF graphs generated from genomic data sources before and after normalizing via RML-Normalizer;** Total size of RDF graphs in MB divided by data volume. RML-Normalizer eliminate portions of RDF triples containing duplicates.

them is however different from the ones over genomic data. These numbers remain exactly the same as originally generated and exhibit no savings. It is clear from the evidence that the construction of RML mapping rules, especially its Subject Map, plays a vital role in space savings. To clarify this, the subject in the synthetic data source is the key to the original data source which means it is unique in the whole data source as opposed to the genomic data source. Therefore, after normalization for each subject generated in child triples map exactly one value from the parent triples map is assigned. This is analogous to the original run over synthetic data, though the duplicated values in data sources are eliminated and both RML mapping rules and data sources are in normal forms.

According to table 6.8, size of RDF graphs are highly impacted by the process of RML-Normalizer. It can be seen that this size is reduced always by almost 80%. Data source with 100K tuples with 78% space reduction compared to the original RDF graph presents the lowest saving. However, considering RML mapping rules over synthetic data sources, there exist no space savings with regard to the output graph. This is explicitly correlated with the number of generated RDF triples described above. The numbers remain the same as before hence the size of the RDF graph. With regard to this, the nature of no saving in the RDF graph is originating from the fact of having the key of the original table as the Subject Map of RML mapping rules, whereas in genomic experiments this is not the case. Therefore it can be concluded that space savings in output RDF graphs are always a function of the structure of functional dependencies as well as RML mapping rules rather than the volume of data sources. Generally speaking, RML-Noramlizer takes advantage of those configurations and reduces the size of intermediate results as well as the final results, i.e., RDF graphs.

Completeness

It is necessary to ensure that the RDF graph generated by applying RML-Normalizer equals the RDF graph generated by the traditional process. To this end, investigating number of generated RDF triples in a graph play as important role as the content of RDF graph too, i.e., resulted resources and properties. In our work, it is taken into account in each RDFizer engine that the resulted graph from traditional process contains no duplicate RDF triples. In fact, each of the engines has a configuration flag to remove the duplicates and we considered these flags in each run of experiments. For instance, in SDM-RDFfizer one can remove the duplicates in RDF graph by setting flag *remove_duplicate* which is responsible for that action.

It is observed in experiments that generally generated graphs are the same as before. Results generated by most of the engines before normalizing from one side and those generated after applying normalization theory from another side can be proof of our study. However, in those cases that RMLMapper could not finish the task successfully, the number of output triples equals

zero. In fact, in all such cases, RMLMapper reached the time limit. In addition, RocketRML generates low numbers of triples while employing RML-Normalizer. The reason is this RDFizer engine is not able to operate joins generated via RML-Normalizer over decomposed RML mapping rules, though it finished the tasks in the specified time limit. Apart from that, RocketRML could finish none of the tasks assigned to it for integrating original synthetic data sources in the defined time limit, hence no output was generated. Similarly, RMLMapper could not transform 16M data source into RDF triples via the traditional approach. Therefore the number of output RDF triples in all the later cases equal to zero. With regard to the experiment performed after employing RML-Normalizer, none of RocketRML's try was actually successful. Altogether, large data sources signify to be challenging for RocketRML engine. In opposite, RMLMapper exploits the possibility offered by RML-Normalizer to finish its assigned tasks. Overall, depicted results evidence that RML-Normalizer leads to the same number of RDF triples as expected, though changes in data volume.

Normalization Execution Time

It is necessary to have the information about the time our approach takes to normalize mapping rules as well as data sources. Considering this time, one can study effectiveness of employing the approach in terms of time. Table 6.9 depicts this time in seconds for each of the experiments based on varying the data volume. As it can be seen the execution time of RML-Normalizer is tightly coupled with the size of data source. Although in big size data source like 16M the execution time is considerable, this is still small part of the total execution time of a knowledge graph creation process. In general, in none of the cases RML-Normalizer is the cause of the longer execution time compared to the traditional approach.

Data source volume	Total execution time (in sec)
100K	0.341
200K	0.667
300K	0.861
400K	1.116
500K	1.407
1M	2.443

(a) Genomic mapping rules and dataset

Data source volume	Total execution time (in sec)
4M	10.471
8M	22.106
16M	43.996

(b) Synthetic mapping rules and dataset

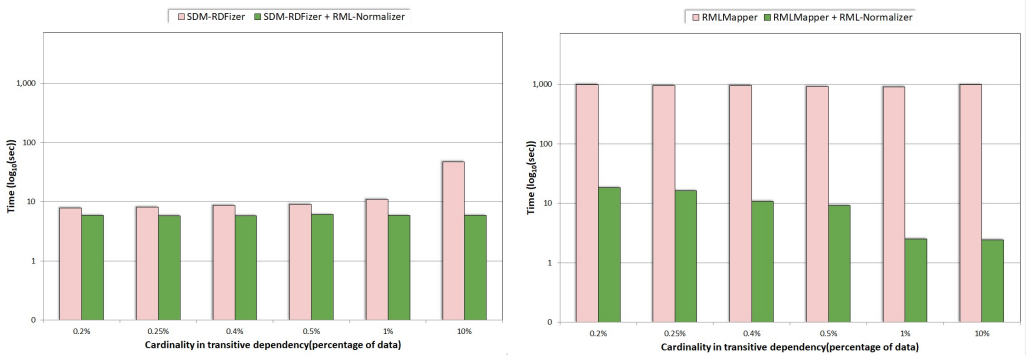
Tabella 6.9 – **Execution time needed by RML-Normalizer to decompose genomic mapping rules and data sources into 3NF while varying data volume;** Total execution times in second with regard to (a) genomic mapping rules and data sources (b) synthetic mapping rules and data sources.

6.4.2 Different Cardinalities

In addition to the experiments based on different data volumes, two experiments with respect to different cardinalities between subsets of attributes are considered. Like the first type of experiment, this is involved with two same RML mapping rules as before. However, in the case of genomic data, we are considering a different subset of data source than the one before, in order to provide each run of experiments with different cardinalities. In fact, 6 subsets of the whole genomic data source are used as data source each of them containing 100K tuples. 10%, 1%, 0.5%, 0.4%, 0.25% and 0.2% are different cardinalities which are taken into consideration for each run of this experiment. Furthermore, for generating synthetic data the same configuration of functional dependencies as before is considered, yet the number of distinct values in $Team\_ID$ are different, as it is shown in 6.2. Accordingly, cardinalities start from 0.0025% up to 2.5% to study different impacts of RML-Normalizer on each system while changing cardinalities, i.e., the join selectivities in case of normalized resources. Needless to say, every other attribute in this experiment owns the same number of distinct values as before hence the same domain. To put it simply, the only used data volume is 4M tuples in this case. In order to observe the effects of different cardinalities over the data sources, other experimental parameters, i.e., the number of joins and data volume are fixed during the whole experiments.

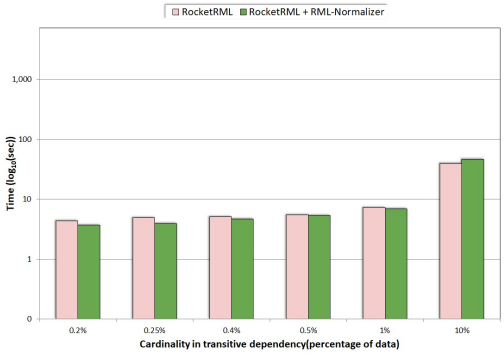
Execution time

With regard to execution times in these experiments, observations indicate often that changes in cardinality between two sets of attributes do not lead to great changes in the execution time of normalized resources. This can be seen in figure 6.5. It can be seen in this figure that the executions of normalized RML mapping rules along with their normalized data sources over genomic data via SDM-RDFizer engine are done almost at the same amount of time. Generally speaking, the performance improvement of SDM-RDFizer in high numbers of cardinality is much more clear than the low ones. To clarify this, the time needed by SDM-RDFizer to perform integration over original inputs fell down dramatically when we change cardinalities from 10% to 1%. Although RMLMapper shows this dramatic change as well, in the case of normalized mapping rules and data sources it needs more time as the cardinality decreases. It can be noted that performance of RMLMapper while using RML-Normalizer is not affected strongly by high cardinalities and it remains always around 98%. Unlike SDM-RDFizer and RMLMapper, RocketRML exposes different behavior with regard to the use of RML-Normalizer and it needs less time as the cardinality fells down. However, due to incomplete output RDF graph resulted from this engine for the case of the normalized version, this result is less reliable. This is again caused by the incapability of RocketRML to perform joins provided by RML-Normalizer. In addition to the above-mentioned dataset, we studied another experiment over synthetic data sources with different cardinalities between two specific attributes. Figure 6.6 determines that RML-Normalizer does not change the execution time of SDM-RDFizer so severely and in some cases, this time remains even almost unchanged. For example, the corresponding execution time for cardinalities of 2.5%, 2.5%, and 0.0025% is almost around 11 seconds. Likewise, the execution time of original resources by this engine remains steady. In opposite, extreme changes of cardinalities, i.e., 0.0025%, 0.00625%, and 0.0125% results in time out of RMLMapper when applying RML-Normalizer. It is observed in experiments that normalizing in the presence of low cardinality between attributes leads to dramatic increments in the execution time of this engine. For instance, the execution time of the engine over normalized mapping rules and data sources with 0.05% is 5 times more than the one done with 0.125%. Nevertheless, RMLMapper performs always faster in original tasks and finish them. However, the huge difference between the traditional approach and approach based on normalization theory can be described by different



(a) SDM-RDFizer

(b) RMLMapper



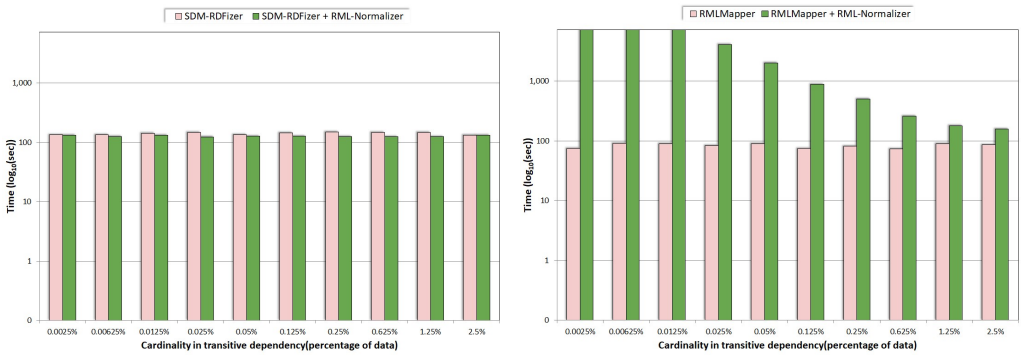
(c) RocketRML

Figura 6.5 – **Performance of different RDFizer engines with and without using RML-Normalizer over genomic mapping rules and data sources violating 2NF** Total execution times, (normalization time is included if any), needed by each RDFizer engine to finish the assigned tasks based on different cardinalities in transitive functional dependencies.

implementations of join operators in different RDFizer engines. Moreover, it should be stated that RocketRML was not successful to finish the tasks due to time out. In general, RML-Normalizer offers slight performance improvements with regard to the presence of different cardinalities, especially when it comes to data with large sizes.

Space Savings

It is clear in table 6.10 and 6.11 that change of cardinality does not affect space savings of RML-Normalizer significantly. However, it can be seen that the ratio of normalized data sources to original data sources is increased whenever cardinality is reduced. Altogether, RML-Normalizer reduces the size of genomic data and synthetic data to approximately 30% and 40% of the original data source respectively. Comparing two similar cardinalities of 0.25% in both data sources, we observe a difference in space savings. Actually, the structure of RML mapping rules in these two data sources are different. RML mapping rules of genomic data result in transforming the data source such that most of the attributes appear in the same data source where the attribute with low cardinality domain set is, whereas in synthetic data attributes are divided equally in the decomposed source. Therefore, the saving in the former is greater than the latter indeed.



(a) SDM-RDFizer (b) RMLMapper

Figura 6.6 – **Performance of different RDFizer engines with and without using RML-Normalizer over synthetic mapping rules and data sources violating 3NF** Total execution times, (normalization time is included if any), needed by each RDFizer engine to finish the assigned tasks based on different cardinalities within transitive functional dependencies.

Cardinality	Original size (in MB)	Size via RML-Normalizer (in MB)
10%	5.9	1.8
1%	6.4	2.0
0.5%	6.3	2.01
0.4%	6.3	2.01
0.25%	6.3	2.01
0.2%	6.3	2.02

Tabella 6.10 – **Size of genomic data sources before and after normalizing via RML-Normalizer;** Total size of data source file(s) in MB divided based on different cardinalities in transitive functional dependencies.

Number and size of generated RDF triples shown in 6.12 provide us with extra information about normalized RML mapping rules and data sources. As can be observed, the size of the RDF graphs generated after the application of RML-Normalizer is much less than the original size. In fact, in every case of genomic data, RDF graphs in terms of numbers of triples and also sizes are reduced by more than 75%, though no loss of information. However, by reducing cardinality of attributes this size starts increasing. In other words, the pace of increment in both measures is a function of the number of attributes in the low cardinality table as well as the cardinality itself. Considering synthetic data source, none of those measures is changed from original to normalized version. To clarify this, a synthetic experiment is based on an RML mapping rule whose Subject Map is the same as the key in the original table. According to the definition of the key in relational databases, there should exist no redundant value for the key hence no difference in the number of output triples and their size. All in all, changes in the sizes of data sources depend on cardinalities between attributes, whereas changes in the sizes of output RDF graphs are dependent on the structure of RML mapping rules in addition to cardinalities.

Cardinality	Original size (in MB)	Size via RML-Normalizer (in MB)
2.5%	175.48	68.67
1.25%	175.48	68.67
0.625%	175.48	68.67
0.25%	175.48	68.67
0.125%	175.48	68.68
0.05%	175.48	68.70
0.025%	175.48	68.73
0.0125%	175.48	68.80
0.00625%	175.48	68.94
0.0025%	175.48	69.35

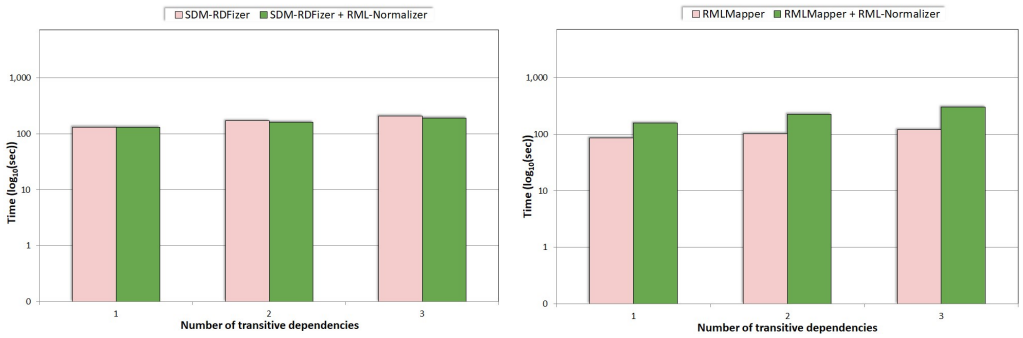
Tabella 6.11 – **Size of synthetic data sources before and after normalizing via RML-Normalizer;** Total size of data source file(s) in MB divided based on different cardinalities in transitive functional dependencies.

Cardinality	Original		RML-Normalizer	
	#of RDF triples	Size (in MB)	#of RDF triples	Size (in MB)
10%	400,000	41.0	100,030	9.0
1%	400,000	39.4	100,300	8.3
0.5%	400,000	39.3	100,600	8.3
0.4%	400,000	39.3	100,750	8.3
0.25%	400,000	39.4	101,200	8.4
0.2%	400,000	39.4	101,500	8.4

Tabella 6.12 – **Size of RDF graphs generated from genomic data sources before and after normalizing via RML-Normalizer;** Total size of RDF graphs in terms of number of RDF triples along with size in MB divided by different cardinalities in transitive functional dependencies. RML-Normalizer eliminate portions of RDF triples containing duplicates.

Completeness

In our experiments, we always consider that the number of generated RDF triples equals to the number of traditionally generated RDF triples after eliminating duplications. In other words, it is essential to observe the same number of unique RDF triples resulted from both original and normalized RML mapping rules. To achieve this goal, each engine is configured for removing duplicates when running on a traditional basis. In fact, the number of unique RDF triples produced as the result of normalized RML mapping rules and data sources is equal to the original generated ones in most cases. However, RocketRML shows a low number of RDF triples as a result of applying RML-Normalizer and this is justified with failure to make joins between decomposed mapping rules. In other words, in recent cases, Rocket RML is able only to transform data from child triples maps and fails to create joins to parent triples map to collect data from the decomposed data source(s). In addition, the number of generated RDF triples by RML mapping rules over synthetic data source is almost always the same with two exceptions. In fact, RMLMapper can not finish the tasks for the data sources with three lowest cardinalities as it was described before in execution times, thus the numbers equal to zero. Analogously, all the numbers for RocketRML in both cases of original and normalized are equal to zero. In general, RDF graphs resulted after applying RML-Normalizer contains the same number of triples as in



(a) SDM-RDFizer (b) RMLMapper

Figura 6.7 – **Performance of different RDFizer engines with and without using RML-Normalizer over synthetic mapping rules and data sources violating 3NF** Total execution times, (normalization time is included if any), needed by each RDFizer engine to finish the assigned tasks based on different numbers of transitive functional dependencies.

the original graph regardless of changing cardinalities.

Normalization Execution Time

With regard to the execution time needed by our approach for normalizing mapping rules and data sources, we measured this time while running every experiment. According to the observed execution times, the normalization process is affected very low by the different cardinalities in our experiments so that the differences are negligible. This can be explained by the fact that the number of redundant data items can not affect the performance of RML-Normalizer.

6.4.3 Different Number Of Transitive Dependencies

In this experiment, different numbers of transitive dependencies and their effects on the result of our work is investigated. This parameter plays a vital role since different numbers of transitive dependencies lead to different numbers of join conditions between decomposed RML mapping rules as well as data sources. In this section, it is satisfying to run only one experiment and only over synthetic data, due to two reasons. For one thing, in genomic data and considering the same structure of RML mapping rules as in figure 4.6 there are only 2 transitive dependencies,i.e., possibility to have two join conditions, yet they do not show same cardinalities hence different join selectivities. This contradicts our aim of changing only one parameter at once. Another thing is discovering the same cardinality for both transitive dependencies within that structure is not plausible because of increment in the size. Like the last reason, this will oppose the purpose of experiments in this work but this time it is because of the variable number of the dependencies and size simultaneously. Therefore, it is chosen to run only one experiment and only over the synthetic data which has a different structure of RML mapping rules. According to this experiment, we consider a synthetic data source of 4M tuples and cardinality of 2.5%, between transitive dependencies, upon which three different RML mapping rules running. These rules are described in 6.2, A.1 and A.2 with one, two and three transitive dependencies respectively.

Execution Time

It can be seen in figure 6.7 that the performance of the knowledge graph creation process supplied with RML-Normalizer is affected differently regarding the number of existing transitive dependencies, i.e., the number of joins. Although SDM-RDFizer exposes negligible improvement in the case of using RML-Normalizer for 4M tuples, it takes advantage of normalization theory and performs more than 13% faster in other cases compared to the traditional approach. In opposite, RML-Normalizer can improve the performance of RMLMapper in none of the studied experiments. In fact, in the worst case, i.e., 16M tuples RMLMapper needs almost 160 more seconds to finish the assigned task. Furthermore, due to time out RocketRML can transform none of the data sources into RDF triples. Overall, the performance of the knowledge graph creation process while applying RML-Normalizer depends on the number of joins and the implementations of joins simultaneously.

Space Savings

With regard to space occupied by data sources, the numbers show literally no further improvements than those in the last experiments. RML-Normalizer offers more than almost 61% in all cases which is quite a great number. In addition, the number of generated RDF triples after applying RML-Normalizer is the same compared to the traditional approach. This leads to the fact that there are also no changes in the size of the generated RDF graph when RML-Normalizer is employed. In general, if RML mapping rules mention attributes with no duplicates, e.g., the key of a data source, as their Subject Map, RML-Normalizer offers no improvement in the size of output RDF graph but it makes considerable changes in the size of data sources regardless.

Completeness

It is clearly observed in our experiments that without any doubt application of RML-Normalizer leads to the same number of output RDF triples. In this case, the numbers of generated RDF triples in both approaches are the same hence a guarantee for having the same graph as before. In fact, in both cases, numbers of RDF triples equal to 12,000,000, 16,000,000 and 20,000,000 with respect to 1, 2 and 3 joins in data source. The differences between numbers of triples in each specific case can be justified by having the same numbers of tuples yet different numbers of joins. All in all, while using RML-Normalizer, generated RDF graphs remain the same as before in terms of the number of RDF triples.

Normalization Execution Times

The execution of the RML-Normalizer plays a vital role in improving the performance knowledge graph creation process. However, the time needed by RML-Normalizer should be observed and we measured that while running our experiments. Table 6.13 demonstrate the total time in seconds taken by our approach to normalize RML mapping rules and corresponding data sources in case of synthetic data. It can be observed that, number of dependencies affects the execution times of RML-Normalizer. Increasing the number of transitive dependencies, i.e., joins, can lead to rise of normalization time. This can be justified with the fact of having more attributes involved in the process. In fact, more attributes result in generating more decomposed data sources as well as more time consumption by elimination of redundancies. Albeit increased execution time of the normalization process, none of the experiments are affected so that the execution time of the knowledge graph creation process exceeds the one in the traditional approach.

Number of transitive dependencies	Total execution time (in sec)
1	10.471
2	12.722
3	14.668

Tabella 6.13 – **Execution time needed by RML-Normalizer to decompose synthetic mapping rules and data sources into 3NF while varying number of dependencies;** Total execution times in second with regard to synthetic mapping rules and data sources

6.5 Summary Of The Chapter

In this chapter, testbeds and their origins as well as experimental configurations, parameters, and metrics were described. In addition, we evaluated the performance of improvements offered by our approach in different cases and explained this empirically along with the results shown in graphs and tables. Our goal in this chapter was to answer the research questions raised at the beginning of this chapter.

Capitolo 7

Conclusion and Future work

In this thesis, we addressed the problem of data integration systems with regard to redundant data within data sources as well as in the final RDF graph. Accordingly, a new normalization theory along with proper definitions for mapping rules and data sources is proposed. On the basis of this theory, an algorithm is presented in which not only every violating mapping rule but also every violating data source is decomposed into a new form. It is demonstrated that the new forms of a mapping rule and a data source are all normalized up to 3NF with respect to a defined set of functional dependencies. Moreover, a specific implementation of this approach is introduced in which RML mapping rules and CSV data sources are considered as input. An application of our algorithm over the motivating example is described in order to approve the applicability of our work on real data examples. In addition, a tool for creating synthetic testbeds in our work is implemented and introduced which can be reused in wide ranges within the community. Like any other scientific work, in this work, several experiments with different configurations are carried out and related results are studied. Each experimental parameter is analyzed in turn and its impact on different metrics is discussed. The content of this chapter is devoted to the discussions about experiments performed in the last chapter and to explain answers to research questions defined at the beginning of the last chapter. Obviously, limitations of our work whether in terms of the defined normalization theory or the implementation is explained. Accordingly, future works are described in order to provide the community with problems that are still open.

7.1 Discussions

Different experiments are performed to evaluate the performance of RML-Normalizer; the results are shown in the last chapter. With the aim of our experiments, it is specified that the size of input data sources reduced dramatically and at almost the same rate in all cases of experiments. However, in some cases, these savings are considerably great because of the structure of the mapping rule. Similarly, the number of generated RDF triples and the size of RDF graphs are affected. An exception, in this case, is whenever mapping rules are not in proper structure thus no savings in terms of RDF can be observed. In general answer to **RQ1** can be described as following: *RML-Normalizer considerably reduces the size of intermediate results in presence of a large number of instances violating functional dependencies.*.

The results of the experimental study suggest that the RML-Normalizer improves the performance of a knowledge graph creation process in presence of large datasets. However, the lower the join selectivity between mapping rules the more time the process takes for generating RDF graphs and creating a knowledge graph. Nevertheless, the behavior of the RML-Normalizer is

highly affected by the efficiency of the implementation of the join operators in an RML interpreter engine. This holds also for having different numbers of joins included by normalization. Different RDFizer engines execute mapping rules with joins in different manners. It is the case that in one engine RML-Normalizer improves performance but in one another engine, it reduces performance. Therefore, answer to **RQ2** is: *RML-Normalizer improves the performance of the knowledge graph creation process strongly in terms of total time whenever join operators are implemented properly*

7.2 Limitations

We have seen in our experiments that there are some situations that normalization of mapping rules and data sources does not speed up the process of knowledge graph creation. One of those cases is when there are small size data sources to be transformed. Regarding this, the rate of performance improvement in terms of time is reduced with some special cases even failed to improve. However, issues regarding these special cases are almost about the structure of mapping rules as well as the implementation of join operators. For one thing, with improper customization of mapping rules, one can not observe great time improvements through big data. Another thing is, implementing join operators in an optimal manner plays a vital role in most cases, with varying volume one of those cases. In fact, the bad implementation of join operators can result even in improvement failure. In addition, in some rare cases, it can lead to having incomplete results albeit finishing the tasks within specified times.

Moreover, experimental results indicate that the join operators between mappings include during the normalization process may be impacted by their selectivity. To illustrate it, inefficient implemented joins cannot only reduce the performance of a knowledge graph creation process, but also in some cases, they hinder the process from finishing the task. Apart from that, the total execution time for creating a knowledge graph is affected by the cardinality of attributes to really low extents. Moreover, inefficiently implemented joins can result also in time rise when the number of transitive dependencies, i.e., the number of joins are increased. Excluding this, the number of joins does not impact execution time negatively. Generally speaking, changes in the size of intermediate results depends on mapping rules' structures. To clarify the point, normalization of mapping rules can dramatically reduce the size of intermediate results as long as the rules contain all attributes with redundant data items. Aside from that, one can observe space savings up to almost 70%.

7.3 Future Works

In our work and especially in the normalization algorithm, we supposed that set of functional dependencies is an input to the algorithm. However, due to the fact that functional dependencies defined over a set of attributes are not always given and they must be mined, one can consider empowering the approach with mining of functional dependencies. This can be even of great importance when big data is involved. One possible case is to have a universal table with numerous attributes upon which functional dependencies are not simply available by humans. Therefore, machines may be efficient mining these dependencies.

In addition, the given input data sources to RML-Normalizer are of data tables in CSV format. Although it was a simple assumption, it was strongly helpful due to the availability of a huge amount of data in CSV in our community. Nevertheless, having a relational database as input to the approach is an essential need for two reasons. First of all, the presence of Database Management System (DBMS) over tremendous data can empower the effectiveness of

the normalization process due to great performance in reading data into the data integration system. Furthermore, a relational database offers the great advantage of using queries, i.e. database views, on top to provide mapping rules with the desired portion of data. Accordingly the performance of the normalization can be exposed to a higher degree.

7.4 Summary Of The Chapter

This chapter discussed the experimental results of our works and the cases in which the limitation of our work is presented. In addition, future works that can be meaningful for different related communities were suggested.

Appendice A

RML mapping rules

In our experiments we considered RML mapping rules with different characteristics. For instance, to study the impact of numbers of joins on execution time of a knowledge graph creation pipeline, different mapping rules has been taken into consideration. Regarding to number of transitive dependencies, i.e., number of joins, three RML mapping rules with different number of object maps are considered. Also the normalized version of these rules hold different number of joins. In chapter 6, one of these mapping rules is shown, namely mapping rule with one join. Figures A.1 and A.2 depict two other mapping rules with 2 and 3 joins respectively.

```
@base <http://tib.de/ontario/mapping#> .
@prefix ex: <http://example.com/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .

<#NBA_P>
 rml:logicalSource [ rml:source "//app//nba_players.csv";
                     rml:referenceFormulation ql:CSV
      ];
 rr:subjectMap [
  rr:template "http://example.com/resource//{NBA_Player_ID}";
  rr:class ex:Basketballplayer
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Team_ID;
  rr:objectMap [
   rml:reference "Team_ID";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Founded_in_year;
  rr:objectMap [
   rml:reference "Founded";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Champs_of;
  rr:objectMap [
   rml:reference "Champs_of";
  ]
 ]
.
```

(a) Original mapping rule violating 3NF

```
@base <http://tib.de/ontario/mapping#> .
@prefix ex: <http://example.com/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .

<#NBA_P>
 rml:logicalSource [ rml:source "//app//nba_players.csv";
                     rml:referenceFormulation ql:CSV
      ];
 rr:subjectMap [
  rr:template "http://example.com/resource//{NBA_Player_ID}";
  rr:class ex:Basketballplayer
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Team_ID;
  rr:objectMap [
   rml:reference "Team_ID";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Founded_in_year;
  rr:objectMap [
   rr:parentTriplesMap <#Team_Foundation>;
        rr:joinCondition [
            rr:child "team_id";
            rr:parent "team_id";
        ];
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Champs_of;
  rr:objectMap [
   rr:parentTriplesMap <#Team_Champs>;
        rr:joinCondition [
            rr:child "team_id";
            rr:parent "team_id";
        ];
  ]
 ]
.

<#Team_Foundation>
 rml:logicalSource [ rml:source "//app//nba_teams.csv";
                     rml:referenceFormulation ql:CSV ];
 rr:subjectMap [
  rr:template "{Founded}"
 ].
<#Team_Champs>
 rml:logicalSource [ rml:source "//app//nba_teams.csv";
                     rml:referenceFormulation ql:CSV ];
 rr:subjectMap [
  rr:template "{Champs_of}"
 ].
```

(b) Normalized mapping rule in 3NF

Figura A.1 – **RML mapping rules over synthetic data described in chapter 6;** (a) shows the original mapping composed of three object maps while (b) shows the normalized mapping rule after normalization that is composed of two joins

```
@base <http://tib.de/ontario/mapping#> .
@prefix ex: <http://example.com/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .

<#NBA_P>
 rml:logicalSource [ rml:source "//app//nba_players.csv";
                     rml:referenceFormulation ql:CSV
       ];
 rr:subjectMap [
  rr:template "http://example.com/resource//{NBA_Player_ID}";
  rr:class ex:Basketballplayer
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Team_ID;
  rr:objectMap [
   rml:reference "Team_ID";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Founded_in_year;
  rr:objectMap [
   rml:reference "Founded";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Champs_of;
  rr:objectMap [
   rml:reference "Champs_of";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Division;
  rr:objectMap [
   rml:reference "Division";
  ]
 ]
.
```

(a) Original mapping rule violating 3NF

```
@base <http://tib.de/ontario/mapping#> .
@prefix ex: <http://example.com/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .

<#NBA_P>
 rml:logicalSource [ rml:source "//app//nba_players.csv";
                     rml:referenceFormulation ql:CSV
       ];
 rr:subjectMap [
  rr:template "http://example.com/resource//{NBA_Player_ID}";
  rr:class ex:Basketballplayer
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Team_ID;
  rr:objectMap [
   rml:reference "Team_ID";
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Founded_in_year;
  rr:objectMap [
   rr:parentTriplesMap <#Team_Foundation>;
        rr:joinCondition [
            rr:child "team_id";
            rr:parent "team_id";
        ];
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Champs_of;
  rr:objectMap [
   rr:parentTriplesMap <#Team_Champs>;
        rr:joinCondition [
            rr:child "team_id";
            rr:parent "team_id";
        ];
  ]
 ];
 rr:predicateObjectMap [
  rr:predicate ex:Division;
  rr:objectMap [
   rr:parentTriplesMap <#Team_Divisions>;
        rr:joinCondition [
            rr:child "team_id";
            rr:parent "team_id";
        ];
  ]
 ]
.

<#Team_Foundation>
 rml:logicalSource [ rml:source "//app//nba_teams.csv";
                     rml:referenceFormulation ql:CSV ];
 rr:subjectMap [
  rr:template "{Founded}"
 ].
<#Team_Champs>
 rml:logicalSource [ rml:source "//app//nba_teams.csv";
                     rml:referenceFormulation ql:CSV ];
 rr:subjectMap [
  rr:template "{Champs_of}"
 ].
<#Team_Divisions>
 rml:logicalSource [ rml:source "//app//nba_teams.csv";
                     rml:referenceFormulation ql:CSV ];
 rr:subjectMap [
  rr:template "{Division}"
 ].
```

(b) Normalized mapping rule in 3NF

Figura A.2 – **RML mapping rules over synthetic data described in chapter 6;** (a) shows the original mapping composed of four object maps while (b) shows the normalized mapping rule after normalization that is composed of three joins

Bibliografia

[1] Satya S Sahoo et al. ≪A survey of current approaches for mapping of relational databases to RDF≫. In: *W3C RDB2RDF Incubator Group Report* 1 (2009), pp. 113–130.

[2] Eric Miller. ≪An introduction to the resource description framework≫. In: *Bulletin of the American Society for Information Science and Technology* 25.1 (1998), pp. 15–19.

[3] Enrique Iglesias et al. *SDM-TIB/SDM-RDFizer: v3.2*. Ver. v3.2. Giu. 2020. DOI: 10.5281/zenodo.3872104. URL: https://doi.org/10.5281/zenodo.3872104.

[4] Anastasia Dimou et al. ≪RML: a generic language for integrated RDF mappings of heterogeneous data≫. In: (2014).

[5] William Ward Armstrong. ≪Dependency structures of database relationship≫. In: *Information processing* (1974), pp. 580–583.

[6] JD Ullman. *Principles of Database and Knowledge-Base Systems, volume Volume I-Fundamental Concepts*. 1988.

[7] Philip A Bernstein. ≪Synthesizing third normal form relations from functional dependencies≫. In: *ACM Transactions on Database Systems (TODS)* 1.4 (1976), pp. 277–298.

[8] Farah Karim et al. ≪Large-scale storage and query processing for semantic sensor data≫. In: *Proceedings of the 7th international conference on web intelligence, mining and semantics*. 2017, pp. 1–12.

[9] Farah Karim, Maria-Esther Vidal e Sören Auer. ≪Compacting frequent star patterns in RDF graphs≫. In: *Journal of Intelligent Information Systems* (2020), pp. 1–25.

[10] Juan F Sequeda. ≪Integrating relational databases with the semantic web: A reflection≫. In: *Reasoning Web International Summer School*. Springer. 2017, pp. 68–120.

[11] Tim Berners Lee. ≪Relational databases on the semantic web≫. In: *Design Issues (published on the Web)* (1998).

[12] *W3C Semantic Web Activity*. https://www.w3.org/2001/sw/.

[13] Claudio Gutierrez et al. ≪Foundations of semantic web databases≫. In: *Journal of Computer and System Sciences* 77.3 (2011), pp. 520–541.

[14] Ora Lassila, Ralph R Swick et al. ≪Resource description framework (RDF) model and syntax specification≫. In: (1998).

[15] Anastasia Dimou et al. ≪Extending R2RML to a Source-independent Mapping Language for RDF.≫ In: *International Semantic Web Conference (Posters & Demos)*. Vol. 1035. 2013, pp. 237–240.

[16] Maurizio Lenzerini. ≪Data integration: A theoretical perspective≫. In: *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM. 2002, pp. 233–246.

[17] Thomas M Connolly e Carolyn E Begg. *Database systems: a practical approach to design, implementation, and management*. Pearson Education, 2005.

[18] Chris J Date. *SQL and relational theory: how to write accurate SQL code*. " O'Reilly Media, Inc.", 2011.

[19] Alon Y Halevy. ≪Answering queries using views: A survey≫. In: *The VLDB Journal* 10.4 (2001), pp. 270–294.

[20] Jeffrey D Ullman. ≪Information integration using logical views≫. In: *International Conference on Database Theory*. Springer. 1997, pp. 19–40.

[21] Hong Yao e Howard J Hamilton. ≪Mining functional dependencies from data≫. In: *Data Mining and Knowledge Discovery* 16.2 (2008), pp. 197–219.

[22] Moussa Demba. ≪Algorithm for relational database Normalization up to 3NF≫. In: *International Journal of Database Management Systems* 5.3 (2013), p. 39.

[23] Thorsten Papenbrock e Felix Naumann. ≪Data-driven Schema Normalization.≫ In: *EDBT*. Vol. 17. 2017, pp. 342–353.

[24] Samaneh Jozashoori e Maria-Esther Vidal. ≪MapSDI: A Scaled-Up Semantic Data Integration Framework for Knowledge Graph Creation≫. In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer. 2019, pp. 58–75.

[25] Matt Buranosky et al. ≪FDTool: a Python application to mine for functional dependencies and candidate keys in tabular data≫. In: *F1000Research* 7 (2018).

[26] Umutcan Simsek, Elias Kärle e Dieter Fensel. ≪RocketRML-A NodeJS implementation of a use-case specific RML mapper≫. In: *arXiv preprint arXiv:1903.04969* (2019).

[27] David Chaves-Fraga et al. ≪What are the Parameters that Affect the Construction of a Knowledge Graph?≫ In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer. 2019, pp. 695–713.