UiO : Department of Informatics
University of Oslo

Tabular Data Cleaning and Linked Data Generation with Grafterizer

Dina Sukhobok

Master's Thesis Spring 2016

Tabular Data Cleaning and Linked Data Generation with Grafterizer

Dina Sukhobok

May 18, 2016

Abstract

The volume of data being published on the Web and made available as Open Data has significantly increased over the last several years. However, data published by independent publishers are sliced and fragmented. Creating descriptive connections across datasets may considerably enrich data and extend their value. One way to standardize, describe and interconnect the information from heterogeneous data sources is to use Linked Data as a publishing technology.

The majority of published open datasets is in a tabular format and the process of generating valid Linked Data from them requires powerful and flexible methods for data cleaning, preparation, and transformation. Most of the time and effort of data workers and data developers is concentrated on data cleaning aspects. In spite of the number of available platforms for tabular data cleaning and preparation, no solution is focused on the Linked Data generation.

This thesis explores approaches for data cleaning and transformation in the context of the Linked Data generation and identifies their challenges. This includes reviewing typical tabular data quality issues found in the literature and practical use cases and their categorization in order to produce the requirements on designing a solution in the form of the set of data cleaning and transformation operations.

Furthermore, the thesis introduces the Grafterizer software framework, developed to assist data workers and data developers in preparing and converting raw tabular data to Linked Data with simplifying and partially automating this process. The Grafterizer framework is evaluated against existing relevant tools and systems for data cleaning. The contribution of the thesis also includes extending and evaluating reference software system to implement the needed data cleaning and transformation operations. This resulted in a powerful framework for ad-

dressing typical data quality issues and a wide range of supported data cleaning and transformation operations.

Contents

List of Figures

List of Tables

Acknowledgements

Here I would like to express my appreciation to a number of people whose help and support guided me through the work on this thesis.

First, I was very fortunate to collaborate with many amazing mentors during my time at SINTEF. The roundtable meetings and discussions have taught me a collaborative work and were an endless source of inspiration. I thank Nikolay Nikolov for significant help in development, assistance, always being open to answering my questions, and for immense patience to my lack of team working experience. For the valuable support and guidance in technical questions, I also thank Antoine Putlier. I also extend my gratitude to my external supervisor Arne Berre.

I am particularly grateful to my main supervisor Dumitru Roman, for motivation, providing background, advice on research and writing, and inspiration with new ideas. His contribution in my level of academic writing, professional knowledge, and career is immeasurable.

Last but not least I would like to thank my friends and family for their support and motivation.

Chapter 1

Introduction

The growth of the volume of information being published on the Web and made available as Open data have led to the need of interconnecting data and enriching them with semantics. This can be supported by using Linked Data as a publishing technology. At the same time, the task of data cleaning and transformation still remains one of the most time-consuming parts of data workers job. This thesis is focused on researching the approaches for data cleaning and transformation in the context of Linked Data creation and introduces a software framework, developed to support tabular data cleaning, transformation, and conversion to Linked Data.

1.1 Thesis Outline

This section shortly describes a thesis structure and provides an overview of what parts of the research are discussed in each chapter.

- *Chapter 1 - Introduction* introduces the need for the new research in data publishing and consumption process and defines the methodology intended to be used in this thesis.

- *Chapter 2 - Related Work* describes relevant related works on data quality and data quality issues, the Semantic Web and the role of data cleaning and its semantic enrichment in the data publishing process.

- *Chapter 3 - Problem Analysis* explores the challenges in resolving data quality issues, possible approaches to data cleaning and transformation, reviews existing tools and systems for data cleaning, and formulates requirements for the artifact to be developed.
- *Chapter 4 - Grafterizer: A Flexible Framework for Tabular Data Cleaning and Linked Data Generation* describes the developed artifact, its functionalities, technologies used to build it, and its user interface.
- *Chapter 5 - Evaluation* contains the evaluation of the developed artifact, describes scenarios of real-life use cases, where Grafterizer was used, and discusses the main advantages of using Grafterizer as well as its shortcomings.
- Finally, *Chapter 6 - Conclusion* summarizes this thesis, provides an estimation of the contributions in terms of expected output and acquired results, and identifies directions for the future work.

1.2 Overall Context

Data analysis activities are predicted to bring the vast majority of profit in companies in the nearest future, possible gain is estimated in billions and even trillions of dollars[1,2]. In order to benefit from knowledge discovery from data, data analysis should be performed on large quantities of data. In other words, data analysis requires a lot of reliable datasets to be published in convenient, comprehensive, and reusable form, and at the same time to be available without any restrictions. According to Suju Rajan, director of research, Yahoo Labs,

> *Many academic researchers and data scientists don't have access to truly large-scale datasets because it is traditionally a privilege reserved for large companies*[3].

[1] http://www.irishexaminer.com/lifestyle/features/dell-chief-executive-says-data-is-the-next-trillion-dollar-opportunity-370608.html last accessed May 18, 2016

[2] http://www.idc.com/getdoc.jsp?containerId=prUS40560115 last accessed May 18, 2016

[3] http://finance.yahoo.com/news/yahoo-releases-largest-ever-machine-140000758.html last accessed May 18, 2016

However, today publishing data on the Web is relatively straightforward in terms of hardware and software. The economic incentives that justify data publication come from the reduced cost of data storage and processing. Emerging technologies in data warehousing further contribute to ease of making data widely available. Taken together, these incentives lead to a large amount of data, being collected by public and private sector organizations, becoming widely available through the World Wide Web and the quantity of published information is growing exponentially [16].

Although data have been extensively collected, stored and made available, they are still not used in full capacity. There are several reasons limiting data workers in consuming Open Data. One of the primary reasons is the lack of simple approaches to interconnecting data from various publishers or even the interrelated datasets from the same publisher.

Linking Data

Why is data interlinking so relevant? The main reason for this is the fact that, very often, data analysis involves not just data directly describing the researched area, but also other related information from different sources. To give an example, a researcher could be focused on exploring a topic related to preserving the environment, such as the effective management of water resources. Collecting information for such research represents a great challenge since effective management of water resources involves the investigation of a wide range of interrelated problems. Gathering only the data directly related to water resources in one particular geographical region is not enough to reach objective conclusions. To maximize the effectiveness of the analysis, the study should include the data both directly related to water systems, and data related to the wider context of water resources management. In addition, information about water resources in adjacent geographical regions may be relevant. Thus, this way of performing the research must consider *integrated* water systems as a dynamic system of various water assets, associated social and economic processes, and corresponding institutional structures[4]. Using this approach the analysts may discover some new cause-and-

[4]https://www.unesco-ihe.org/academic-departments/integrated-water-systems-governance last accessed May 18, 2016

effect relations, that cannot be seen when exploring data scoped only to the object of the analysis.

Various organizations may be in possession of the required statistical data and other information – including governments, water industry, environmental agencies, public and private entities in water-dependent industries. The way through which aforementioned organizations publish their data may be very different, e.g., data may come in different structures, same concepts in different datasets may be described in different manners, etc. Even after transforming data to the unified form, querying them still requires a lot of preparation, e.g., collecting the data and putting them in a single database.

A great solution for the connecting structured data on the Web is provided by Linked Data. The Linked Data set of best practices for publishing and interlinking data enables data being published to be discovered and used by various applications [10].

1.3 Thesis Motivation

The process of preparing, cleaning and transforming open datasets to the Linked Data is rather challenging. The first step is to bring data into usable form, easy to manipulate and transform to Linked Data. The research literature refers to the process of data cleaning in different ways, depending on the scope of the resolved data quality issues. *Data preparation* [33, 6], *data cleansing* [31, 30], *data cleaning* [5, 35], *data wrangling* [24, 23] and *data tidying* [46] are popular ways to refer to the process of bringing data to the formats that can be easily manipulated. To avoid disambiguation, in this thesis the process of resolving data quality issues is referred to as **data cleaning**.

Once data are cleaned, they can be converted to the Linked Data format. This requires mapping data to conceptual models and provisioning the data. Since both data cleaning and data mapping are the relevant aspects of Linked Data generation, they should be performed together as two sides of one unified transformation.

When data are properly cleaned and correctly linked, it significantly increases their accessibility and reliability [6]. On the other hand, if data are not clean, and, thus contain errors and inconsistencies, it may lead to false conclusions made by data consumers (e.g., data analysts) and reduce trust towards data providers. The

process of data cleaning ensures a consistent structure of a dataset, thus making it easy for an analyst or software programs to find and extract needed variables [46], which makes data usage less time-consuming and more efficient.

At present, no unifying framework exists that supports data cleaning and data mapping as two parts of a single process resulting in Linked Data creation. The development of such a framework can significantly simplify data publication, increase the speed of data publishing and extend the value of published data, providing more opportunities for reuse Linked Open Data in various applications and contexts.

1.4 Research Questions

The questions answered in the scope of this thesis are:

- What is data quality and what data quality issues can occur?
- How is Linked Data generated from tabular data?
- What are the existing tools for tabular data cleaning and transformation? What tasks are impossible or difficult to solve using them?
- What artifact can be developed to improve current state-of-the-art? What data cleaning and transformation operations should it support and what functionalities should be provided by its user interface?
- Is the developed artifact capable of performing cleaning and transformation tasks in real-life scenarios? What improvements can be identified for the future work?

1.5 Thesis Contributions

This thesis contributes to Grafterizer – a web-based framework for data cleaning and transformations. Grafterizer is a part of DataGraft[5,6] – a powerful cloud-based

[5]DataGraft is accessible at https://datagraft.net/

[6]Github open source project is accessible at https://github.com/dapaas/dapaas.github.io

platform for data transformation, publication, and hosting. DataGraft implements the concept of *data-* and *transformation-as-a-Service*. The main goal is to let data publishers and data workers concentrate on their immediate work without the need to worry about technical details, and to simplify their work by maximizing the automation of data cleaning, transformation, and publication, as well as supporting the reuse of previously performed data transformations. The core DataGraft functionalities are transforming data, hosting it, and making it easily accessible. The platform is actively developed and extensively used in various contexts. The Grafterizer framework, as an essential part of it, provides support for data cleaning and transformation functionalities.

Summary of Thesis Contributions

The contributions of the thesis include:

- Providing a categorized summary of tabular data quality issues based on studied literature and practical use cases.
- Evaluating existing software tools and systems for tabular data cleaning and transformation against the Grafterizer framework.
- Providing a categorized summary of tabular data cleaning and transformation operations that can solve most of the common data quality issues.
- Implementing needed data cleaning and transformation operations and extending the Grafterizer framework with the user interface to support needed operations.
- Evaluating Grafterizer with real-life use case scenarios.

1.6 Research Methodology

The research process can be performed in two different forms:

Basic research is research for the purpose of obtaining new knowledge.

Applied research is research seeking solutions to practical problems [42].

This thesis represents a special case of applied research – technology research. The final goal of any technology research process is to develop an artifact (or improve the existing one) that satisfies a set of collected requirements [42]. The main steps to perform technology research are shown in Figure 1.1. This process

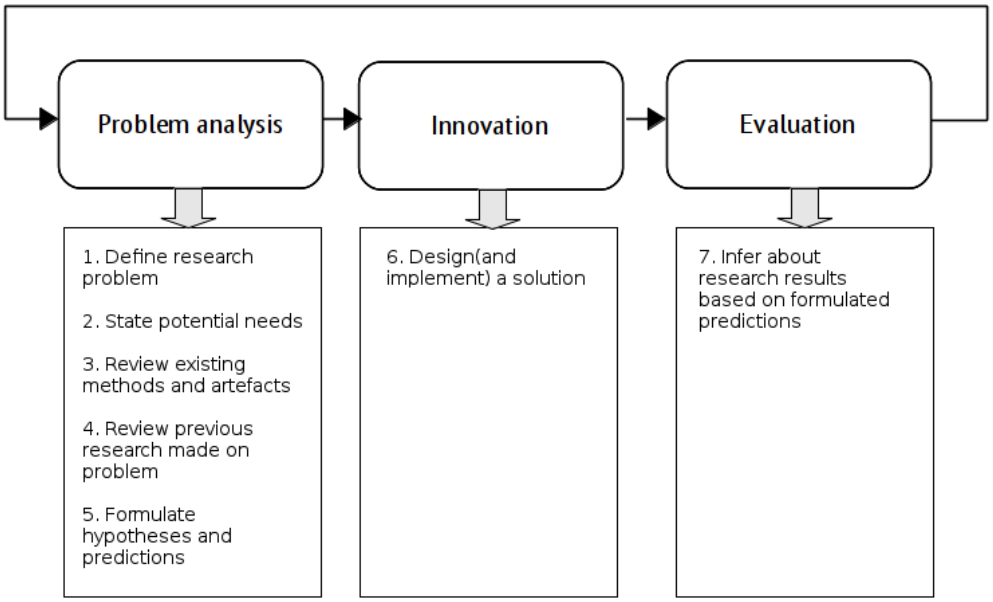

Figure 1.1: Methodology for technology research adopted in this thesis

consists of three main phases - problem analysis, innovation, and evaluation - and is iterative by nature. During the first phase, the researcher becomes acquainted with the research problem. This can be achieved by thorough literature study, discussions with specialists in the given research area, and investigation of current methods and artifacts. As a result of the aforementioned process, the researcher produces a phrased problem statement unambiguously in specific terms [27]. The final step in this phase is to state the potential needs for the artifact, i.e., to produce working hypotheses and predictions about alleviating the defined problem. It may be very helpful at this point of the process to consult with current and potential artifact users. Potential needs should be expressed in terms of success criteria. Success criteria establishment plays an important role in technological research.

This includes a set of requirements, the developed artifact should satisfy, and it not only defines precise goals of the performed research but also serves as a way of evaluating the resulting artifact.

After all the requirements have been collected and rephrased in the form of success criteria, the next phase, innovation, starts. In this phase the researcher looks for the possible problem solutions and applies them in practice. Naturally, this phase ends with producing a prototype or ready-for-use software product.

In order to estimate the performed work, one should carefully analyze the correspondence between an artifact and its requirements. This analysis represents the final phase – artifact evaluation. Based on the results of the performed evaluation, the researcher makes a conclusion about the performed work and identifies the effect of the developed product on the current state-of-the-art. The evaluation may also prepare a basis for new research.

Chapter 2

Related Work

Several years ago technological advancements led to the significant reduction in costs for data publication. This resulted in large amounts of data being generated, collected, and disseminated through the Web. The quantity of published information quickly outpaced the ability to process this information. The main reason for this was that data were presented in human-, not machine-readable, and, very often in their raw, "messy" form. These factors hinder automated data processing and increase a time needed to extract valuable information from data.

To make data easy to be processed by software programs, it is necessary to clean and standardize them. By removing data impurities, we significantly increase chances of correct data interpretation.

However, data cleaning itself doesn't make data easy to understand and does not provide meaningful descriptions of data. To cope with this problem, in 2001 the first attempts to amend the Web were launched. A new form of the Web was described in the article called "The Semantic Web" by Tim Berners-Lee, James Hendler and Ora Lassila published in the Scientific American [9]. The main idea was to make the Web content more machine-processable, and to achieve this, it was suggested to enrich available information with semantics. This approach resulted in the creation of the concept of the Semantic Web, which provides the Linked Data model.

This chapter gives a basic introduction to data cleaning and the ways to provide data description and interlinking with the help of the Semantic Web technologies.

2.1 Data Cleaning

The term "data cleaning" should be investigated in the context of data publishing and data consumption, rather than an independent concept. The reason for this is that outcome of data cleaning must answer the purpose of data publishing or consumption. In particular, prior to finding ways to resolve data quality issues, it is very important to know at what stage of work data quality should be assessed and when data cleaning should be performed. To answer these questions, it is necessary to get an overview of data publishing and data consumption activities. Knowing the place of data cleaning in their workflow, it is easier to identify possible input and desired output of the data cleaning.

2.1.1 Data Cleaning in ETL Tools

Unlike locally stored homogeneous data, data published on the Web often involve the task of integrating information from several heterogeneous data sources. In terms of information integration, it is necessary to take into account *data warehousing* technologies. Elmasri and Navathe define a data warehouse as a collection of information and a supporting system, optimized for data retrieval [14]. The entire data warehousing process is shown in Figure 2.1.

The process of integrating data from various data sources into a data warehouse is aided by Extract-Transform-Load (ETL) tools. It covers collecting data from input sources, possibly cleaning and transformation (i.e. reformatting them to match the global schema) and their loading. The second step of the ETL flow ("transform") constitutes the main focus of this thesis. The output of this step should be accurate data which is complete, consistent, and unambiguous [13]. Typically, commercial ETL tools have rather basic data cleaning capabilities and there is usually no support for automated detection of data errors and inconsistencies [35].

According to [12], data cleaning is estimated to take 50-80% of the development time and cost in data warehousing projects.

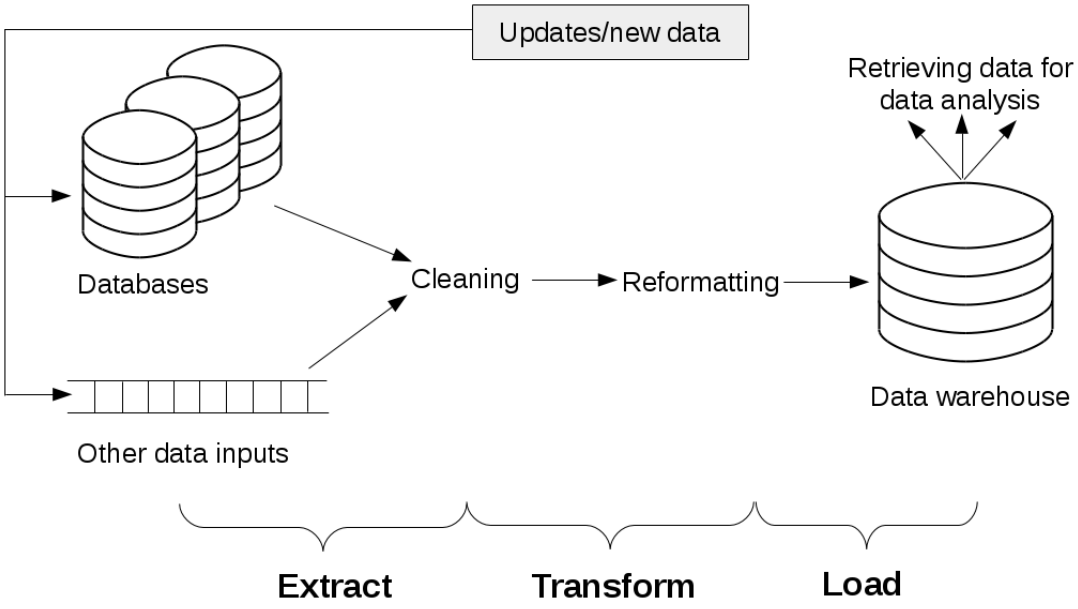

Figure 2.1: Overview of data warehousing [14]

2.1.2 Data Cleaning in Data Analysis

Not only data publishers encounter the challenge of data cleaning. This task is also an inevitable part of work in any data analysis task.

Data analysis technologies include simple statistical analysis, more complex multidimensional analysis, data mining and knowledge discovery in databases (KDD), and are aimed at extracting useful knowledge from explored data. The knowledge discovery process has several phases [14]:

1. Data selection
2. Data cleaning
3. Data enrichment
4. Data transformation or encoding
5. Data analysis itself

6. Reporting and display of the discovered information.

It is easy to see that first four phases are very similar to data processing in the data publishing process, and data analysts (also referred to as "data scientists" or "business analysts") also work with data cleaning. A recent survey of about 80 data scientists, performed by CrowdFlower, known provider of a data enrichment platform for data science teams, inspected various aspects of data scientists' work. The results of this survey clearly identify data cleaning as the most time-consuming and less enjoyable part of their work (Figure 2.2).

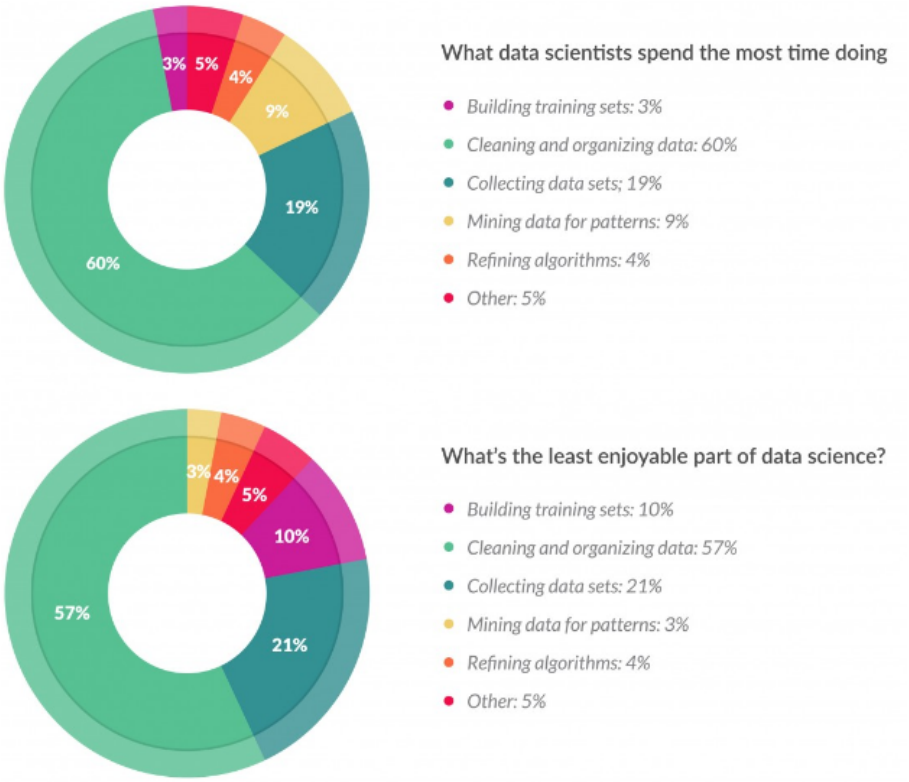

Figure 2.2: Results of the survey of data scientists[1]

[1] http://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#514c427c7f75 last accessed May 18, 2016

To find a way to alleviate data publishers' and data scientists' work in data cleaning, it is necessary to know, how to define and measure data quality. But before studying a data quality, it is important to note, that data quality is closely related to the type of data, for which quality is measured. Therefore, we should first explore existing types of data.

2.1.3 Types of Data

Depending on how the data are organized, three types of data can be distinguished:

Structured data concerns, in the first place, tables in relational databases, and is characterized by strict adherence of data to the associated schema. The structured data represents only 10% of all electronic data[2].

Semi-structured data may have some certain structure, but are not organized as strictly, as structured data. Some distinguishable characteristics of semi-structured data include: data attributes not known at the design time and therefore data are not associated with pre-defined schema; attributes have different representation among data entries; missing attributes for some entries. Usually, in a semi-structured data, schema information is mixed in with data values, which is why data of this type are often referred to as *self-describing data*. Examples of semi-structured data are CSV tabular formats, XML and JSON documents. Data stored in NoSQL databases are also considered as semi-structured data.

Unstructured data , as their name suggests, don't have any organization at all. Typical examples are text documents or multimedia content [14].

Semi-structured and unstructured data at present comprise most of the information available on the Web and the massive growth of data of these types[3] has inevitable implications for data publishing and data analysis technologies.

The most well-known and widely accepted (by data scientists) data presentation form is a tabular format, which is semi-structured data. Statistical data that

[2] http://www.ibmbigdatahub.com/blog/do-not-ignore-structured-data-big-data-analytics last accessed May 18, 2016

[3] http://www.datasciencecentral.com/profiles/blogs/structured-vs-unstructured-data-the-rise-of-data-anarchy last accessed May 18, 2016

data publishers make openly available is also mostly in a tabular format. Thus, the research performed in this thesis is focused on tabular data as input and on generated Linked Data as output. Therefore, in the first place, data quality should be considered in the context of tabular data.

Prior to a more detailed investigation of data quality, it is important to provide a basic vocabulary, describing structure and semantics of typical tabular dataset.

Most of the statistical datasets are **tables** and are composed of **rows** and **columns**. Columns in tabular data are almost always labeled with **column headers**.

Table 2.1: A typical example of tabular dataset

Name	**Age**	**Gender**
Alice	28	female
Bob	34	male

Datasets are intended to represent some part of the real world and each element in a dataset should be mapped to objects of a real world. Therefore, tabular data should be published in accordance with the following rules:

1. Each row represents an **entity**, which can be, for example, a person, place, physical object or an event. Entities have a unique existence in the real world.

2. Each column header represents an **attribute** of an entity.

3. Each column value represents a **value** of the corresponding attribute of an entity.

4. Each table represents a **collection** of entities.

5. All entities in a collection have the same **entity type**.

To exemplify the statements made above, Table 2.1 provides a collection of data about two entities of a type Person, having attributes Name, Age and Gender with values of these attributes represented as values of corresponding columns. This

small vocabulary for tabular data's structural elements is used for exemplifications in the rest of the thesis.

2.1.4 Data Quality

When speaking about data quality, it is often to think about criteria such as data accuracy (usually syntactic accuracy). The most common examples of syntactic accuracy violation are misspellings, such as typos and phonetic errors. Indeed, these errors are present in most data and they significantly affect data quality, but accuracy is not the only side of data quality.

The term "data quality" has been extensively studied in many areas, such as statistics, business management, and computer science. In computer science, data quality has been intensively studied since the beginning of the 1990's when the problem of measuring and improving the quality of electronic data emerged [5]. At present, data quality is considered as a multidimensional concept. That means that each specific aspect of data quality is captured by a data quality dimension (sometimes also referred to as data quality criteria). Hence, in order to measure overall data quality, each data quality dimension should be assessed.

The literature on data quality includes many taxonomies for data quality dimensions [6, 31]. With respect to the problem statement of this thesis and expected input data, the following data quality dimensions are identified as most important:

Accuracy measures the distance between value v of a real-life entity attribute and value v' representing the same attribute in a dataset as a column value. For example, when data describe certain infrastructure component, attribute value specifying its type $v' = Bridge$ is correct while attribute value $v' = Brdg$ is incorrect. Usually, two types of accuracy are distinguished:

- *Syntactic accuracy* defines whether value v' belongs to the domain range D, which is defined for the entity being represented. Thus, value $v' = Road$ is syntactically correct, even though the described entity is a bridge because this value corresponds to the domain of infrastructure types. Syntactic accuracy can be identified when domain range is specified, i.e., it requires values to have certain data types.

- *Semantic accuracy* defines whether value v' corresponds to true value v. To continue the example given above, value $v' = Road$ is incorrect, since it describes the real-life entity $v = Bridge$. Another name for semantic accuracy is *correctness*.

Syntactic and semantic accuracy may coincide, since when syntactic accuracy is violated, it affects also a semantic accuracy. Clearly, the violation of just semantic accuracy is typically more complex to detect than the violation of syntactic accuracy. This may require complex comparison algorithms, analyzing different tables describing various aspects of the same entities, and often involves analyst's judgment.

Completeness is defined as the extent to which a given dataset describes the corresponding part of a real world. Completeness may be measured in different ways:

- *Measurement completeness* measures the presence of null values in certain columns of a row (absence of value for certain attributes of entity).
- *Entity completeness* measures the presence of null values in all columns in a row (presence of empty attributes).
- *Attribute completeness* measures the presence of null values in certain columns (absence of values for certain attribute for all entities in the collection).
- *Collection completeness* measures the presence of null values in the entire dataset (collection of entities).

Consistency captures the presence of contradictions and can further be divided into two types – consistency of values within a dataset and consistency of values between different datasets. Contradictions take place when schema integrity constraints are violated. For example, a schema constraint may require the attribute "Age" of an entity "Employee" to hold employee's age as the difference between current date and value of attribute "Birth date". If column values for attributes "Age" and "Birth date" of the same entity do not hold this constraint, they are inconsistent.

Uniqueness dimension measures redundancy of entities, described in a dataset. When an entity is stored in a dataset two or more times, it means that the data source contains *duplicates*. The duplication problem increases significantly when multiple data sources need to be integrated, which often happens both during data publication and data consumption. In this case, the datasets often contain redundant data in different representations [35]. Clearly, to be able to identify duplication, the entities should be assigned a primary key (one-attribute or composite), unambiguously distinguishing the described entities. Uniqueness is sometimes considered as a special case of consistency when the primary key schema constraint is violated. However, due to the frequency of occurrence of uniqueness violation and special way of detection of duplicates problem, in scope of this thesis uniqueness is presented by separate dimension.

The aforementioned data quality dimensions are used to measure a quality of semi-structured tabular data, that needs to conform to some schema. Therefore, these data quality measurements refer to the dataset schema types and constraints.

Linked Data can be considered as structured data. To evaluate the Linked Data quality, one more data quality dimension should be taken into account – schema quality dimension. Schema quality dimension is characterized by the following:

Correctness with respect to the model. Prior to the creation of relevant vocabularies and Linked Data generation, data workers normally develop a data model. Concepts and their attributes described in the data model should be represented correctly in associated schema. Thus, for example, entities or observations should form a concept, that is unique and has its own distinguishable identifier.

Correctness with respect to requirements is observed when the schema requirements in terms of model categories are represented correctly. For example, if each order should have exactly one customer, the type of relationship between entities Order and Customer should be "one-to-one", not "one-to-many".

The minimalization dimension means conciseness of the Linked Data schema and minimization of redundant schema and data elements.

The completeness of a schema measures the extent to which schema includes all the necessary elements, i.e. attributes related to described entity or observation [5].

Another data quality dimension, applicable exclusively to Linked Data, is *linkability completeness* measuring the number of interlinked instances in a dataset [4].

2.1.5 Data Anomalies

When a dataset does not satisfy given data quality criteria, it means that it contains **data anomalies**. In order to provide higher data quality, these anomalies should be detected and removed.

The sources of problems with the data may differ. The most common reason of erroneous data is human errors during the manual production of the data. Another source for data quality issues is data schema evolution over time, which can cause misinterpretation of new entity types or attributes. Finally, automated data generation, such as information derived from sensors, carries its own issues, such as errors due to the inferences from the environment or wrong calibration [25].

Although data quality issues differ from dataset to dataset, it is possible to identify some common data anomalies. In order to use a systematic approach to data cleaning and to alleviate further usage of the developed method for data cleaning, it is necessary to explore possible data quality criteria violations and categorize them. To obtain a list of possible data anomalies, the research literature on data quality, statistics, and data cleaning has been studied, including literature containing interview results with data workers [23] and literature using pure logical reasoning to describe data quality issues [31, 35, 46].

Anomalies taxonomies in the research literature tend to be rather generic and describe data quality problems for all types of data. According to the scope of this thesis, we investigate only data anomalies inherent in tabular data aimed to be transformed and published as Linked Data. There are several ways to categorize data anomalies:

- By the scope of a data quality problem
- By the violated data quality dimension.

Addressing the scope of a data quality problem, data anomalies may occur in single or multiple column values, in column headers, rows within a table or across several tables (see Figure 2.3). Since column headers do not represent data themselves but define the data structure, anomalies in column headers are treated as schema quality issues.

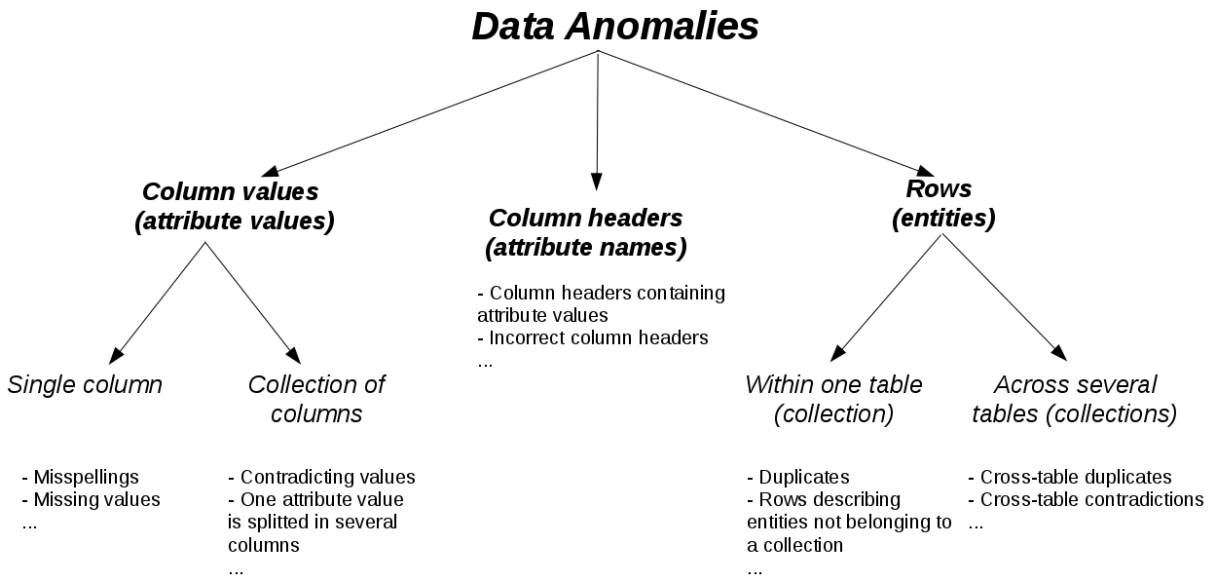

Figure 2.3: Data anomalies categorized by scope of data quality problems

When data anomalies are categorized according to the affected data quality dimension they can be divided into data anomalies violating accuracy, consistency, completeness, uniqueness, and those, that affect several data quality dimensions.

A summary of extracted data anomalies for the purpose of this thesis is described in Table 2.2.

Table 2.2: Summary of data anomalies

Scope	Problem	Data anomaly example	Reasoning	Affected data quality dimension
Column values	*Illegal values*	bdate = 30.02.1987	Values outside of domain range. Date 30.02.1987 is illegal date	**Accuracy**: syntactic and semantic
	Erroneous values	bdate = 15.02.1987	Syntactically correct values, not contradicting with other column values, but representing wrong attribute values for the entity. The most difficult anomaly to identify	**Accuracy**: semantic
	Inconsistent column values	date = 30.02.1987, age = 18	Date of birth and age are inconsistent	**Consistency**
	Missing values	$person_1$ = (name = "Alice Smith", age = null)	One or several column values are missing	**Completeness** **Consistency***
Column headers	*Column headers containing attribute values*	observationEmpNo = (2014 = 123, 2015 = 157, 2016 = 170)	Observation about number of employees in company contains values of attribute "year" as column headers	**Schema quality**: correctness with respect to the model
	Incorrect column headers	$person_1$ = (name = "Alice Smith", age = "female")	Column header is inconsistent with actual attribute it holds	**Schema quality**: correctness with respect to the model
	Column headers not related to model	$person_1$ = (name = "Alice Smith", petName = "Polly")	Dataset describes attributes not relevant in scope of the collection	**Schema quality**: correctness with respect to the model
Column values, Column headers	*Multiple values stored in one column*	$order_1$ = (number = 12345, address = "New York, Harrison Street, 507")	Data anomaly takes place under the assumption, that the data model requires storing address in several attributes - city, street and house number	**Consistency**, **Schema quality**: correctness with respect to the model
	Single value is splitted across multiple columns	$order_1$ = (number = 12345, city = "New York", address = "Harrison Street", houseNo = 507)	Data anomaly takes place under the assumption, that the data model requires values of address attribute to be stored in one column	**Consistency**, **Schema quality**: correctness with respect to the model

(Continued on Next Page)

Scope	Problem	Data anomaly example	Reasoning	Affected data quality dimension
Rows	*Duplicate rows*	$person_1$ = (name = "Alice Smith", id = "12345") $person_2$ = (name = "Bob Johnson", id = "12345")	Uniqueness of entity with primary key ID is violated	**Uniqueness**
	Row, describing entity not belonging to a collection	$person_1$ = (name = "Alice Smith", id = "12345") $person_2$ = (name = "MyCompany, Inc.", id = "12346")	Same collection contains data about physical persons and company, i.e. another type of entity - legal person. Type of described entity should follow the schema.	**Consistency**

*Consistency violation because of missing values takes place in case of missing primary key value. In this way row describes an entity, which has not a unique existence in the real world and therefore doesn't satisfy defined schema.

2.2 The Semantic Web

A major part of Web content is not machine-accessible, i.e., although information itself is available, it is still a challenge to process and interpret it completely automatically. Difficulties and limitations in managing available data include searching, extracting, and maintaining data.

The Semantic Web approach is aimed at representing the information contained in World Wide Web in a way that is more advantageous in terms of data consumption. The Semantic Web technologies are promoted by the World Wide Web Consortium (W3C) – an international standardization organization, developing Web standards. This section provides information on basic Semantic Web technologies used in this thesis.

One important term that is closely related to organizing data and represents an essential part of the Semantic Web is **an ontology**. Ontology can be defined as an explicit and formal specification of a conceptualization [19]. In the context of the Semantic Web, the terms "ontology" and "vocabulary" are often used as synonyms and describe concepts and relationships between concepts. Concepts can be thought of as classes (types, categories) of entities of the real world. For example, these can be persons, places, etc. The advantages of having a good data ontology include the support for interlinking data and standardization of terms for concepts and relationships between them. The number of ontologies have been developed and are freely available on the Web. Data publishers, wishing to take advantage of the Semantic Web are encouraged to adopt and extend existing vocabularies to support data interlinking and standardization.

Another concept, that should be explained in the context of data organization in the Semantic Web is **data graph**. Data graphs consist of resources and relationships between them. The basic building block of a data graph is a three-part statement, commonly called a **triple**. The three parts composing a triple are **subject**, **predicate** and **object**. This way of constructing statements is very natural and is analogous to constructing sentences in a natural language, where subjects, verbs, and objects are used to express statements. Since information is stored in the form of triples, knowledge bases of the Semantic Web data are often called **triplestores**.

The easiest way to illustrate the terms introduced above is to use a simple

graph. Figure 2.4 illustrates one statement that can be constructed from Table 2.1, where the subject is *Alice*, the predicate is *gender* and the object is *female*.

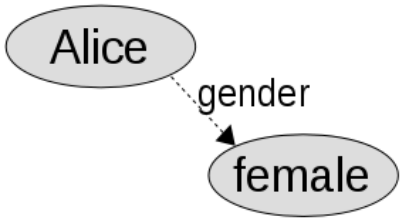

Figure 2.4: An example of data graph triple

Resource Description Framework

The formal language used to define basic graph structures in the Semantic Web is RDF (Resource Description Framework). RDF statements can use different syntax to be represented and interexchanged, e.g., XML syntax, N-Triples, Terse RDF Triple Language (Turtle) etc. The following code describes the graph given above in terms of a simple RDF/XML statement:

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:characteristic="https://www.example.com/PersonCharacteristics#">

 <rdf:Description rdf:about="https://www.example.com/Person#Alice">
   <characteristic:gender rdf:resource="https://www.example.com/genders#female" />
 </rdf:Description>
</rdf:RDF>
```

Listing 2.1: A simple RDF statement

As its name suggests, the Resource Description Framework describes resources. Resources may be any entities of the real world. To express the unique existence

of a described entity, every resource is identified by a **Uniform Resource Identifier (URI)**. URI's are organized with the help of the **namespaces**. On line 5 in the code in Listing 2.1, a namespace *characteristic* is defined, which has a namespace URI *"https://www.example.com/PersonCharacteristics#"*. The purpose of having namespaces in RDF, as in any other language, is to avoid naming conflicts. Thus, the data graph with fully qualified URIs looks like the graph depicted in Figure 2.5.

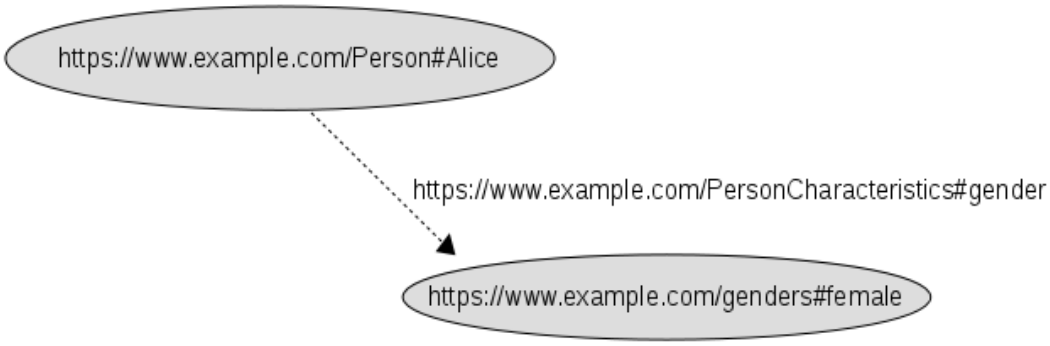

Figure 2.5: The data graph with fully qualified URIs

It is necessary to emphasize, that RDF is a standard language, used to express data, but it doesn't define the semantics of data. This is done with the help of Resource Description Framework Schema (RDFS) and Web Ontology Language (OWL). RDF Schema describes a domain in terms of classes and properties and supports defining hierarchical relationships by using subclasses and subproperties. The Web Ontology Language helps to express how the described data relates to other data on the Web.

Querying Semantic Data

It is expected, that data based on the Semantic Web specifications is easier to search and extract. Knowledge bases for Semantic Web data (triplestores) can be queried with the help of SPARQL (SPARQL Protocol and RDF Query Language) [18]. SPARQL is rather similar to SQL, whose syntax is familiar to many data workers and developers.

SPARQL supports four forms of queries:

SELECT queries return a sequence of values defined by a query pattern in the form of a table.

CONSTRUCT queries return RDF graph as a specified subset of the queried data.

ASK queries return a boolean value, answering whether or not a query pattern has a solution.

DESCRIBE queries return RDF graph, where the data variables to be returned are defined not by a client, but by query endpoint.

```
1 PREFIX characteristic: <https://www.example.com/PersonCharacteristics#>
2
3 SELECT ?person
4 WHERE {
5 ?person characteristic:name "Alice" .
6 }
```

Listing 2.2: Example of SPARQL query

Listing 2.2 gives an example of a SELECT query. It is easy to see, that SPARQL SELECT queries, just as SQL queries, have SELECT and WHERE clauses. The important difference here is the presence of PREFIX keyword, identifying namespace used in the query.

Linked Open Data

Publishing data as a Linked Open Data covers two aspects: first, making data open, i.e. available to everyone without any restrictions. This can be achieved by publishing data under open licenses. The second aspect, linking data, implies creating relationships between entities described in data expressed in machine-readable form. The set of guidelines for publishing Linked Data on the Web was defined by Tim Berners-Lee [4] and has following recommendations:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).
4. Include links to other URIs so that they can discover more things [8].

The web of Linked Open Data is often visualized as a linked data cloud (Figure 2.6). The graph nodes in this figure represent datasets that have been published in Linked Data format. The depicted nodes are based on the metadata collected by contributors to the Data Hub[5] data management platform and metadata extracted from a crawl of the Linked Data on the Web conducted in April 2014.

[4] https://www.w3.org/DesignIssues/LinkedData.html last accessed May 18, 2016
[5] https://datahub.io/

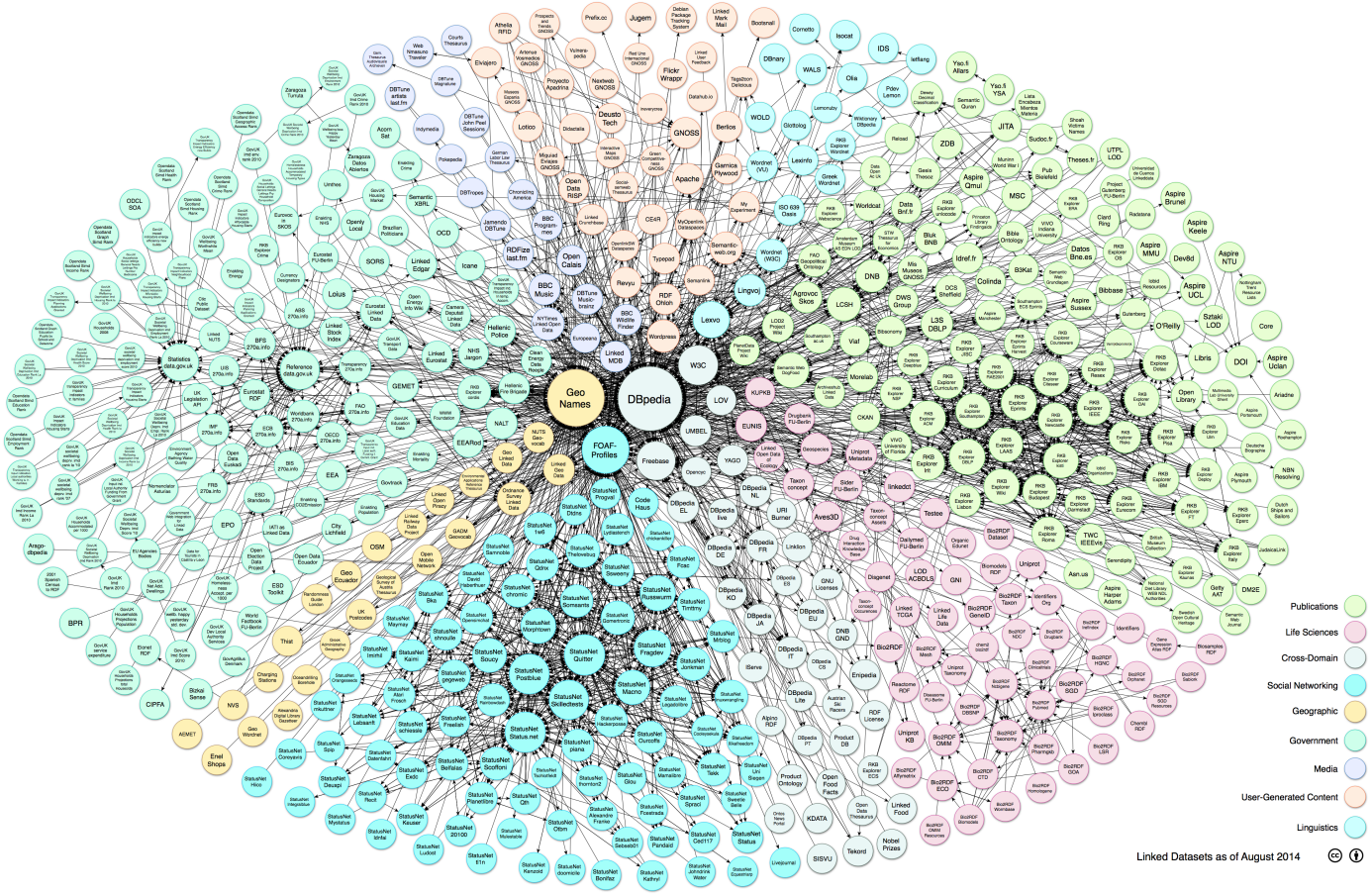

Figure 2.6: Linked Open Data cloud [6]

[6] http://lod-cloud.net/ last accessed May 18, 2016

Chapter 3

Problem Analysis

In the previous chapter, we defined data cleaning as a process of detecting and removing data anomalies. This chapter provides more details on how these operations are performed and concludes with a set of explicit requirements for the artifact developed as part of this thesis to simplify data cleaning and transformation process.

3.1 Data Cleaning and Transformation Cycle

Comprehensive data cleaning comprises four general phases and is iterative by nature. The phases of data cleaning are:

1. *Data auditing* aimed to detect data anomalies.

2. *Definition of transformation workflow* aimed to suggest a way to remove data anomalies.

3. *Execution of transformation workflow* aimed to apply suggested transformation to data.

4. *Verification of the executed transformation* aimed to evaluate the results.

The generalized goal of this thesis is to develop an artifact supporting data cleaning and transformation to RDF. Hence, an input of raw tabular data should undergo

these four phases of data cleaning and be transformed to clean data either in tabular or Linked Data format. The data cleaning cycle, as a part of a developed artifact, is depicted in Figure 3.1.

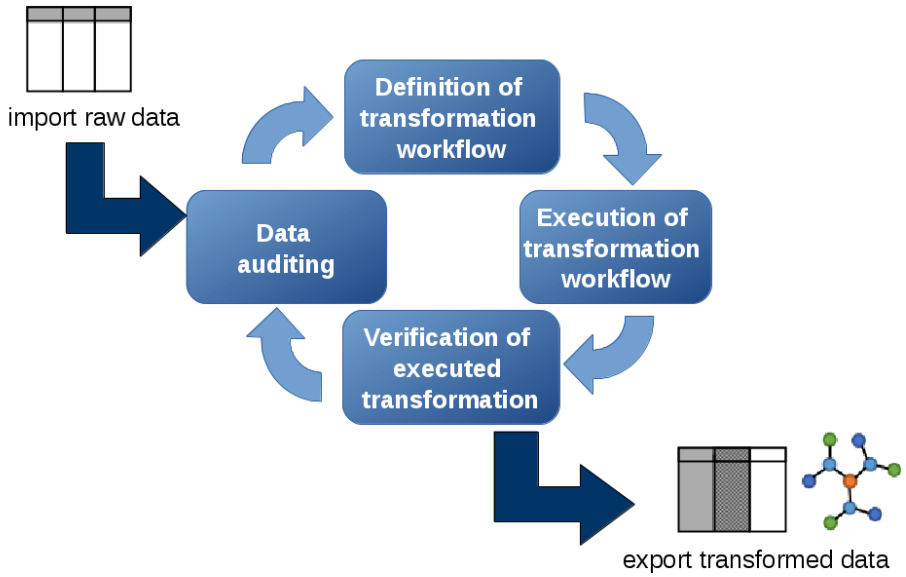

Figure 3.1: Data cleaning and transformation cycle as a part of a developed artifact

Data Auditing

The first step in the data cleaning and transformation process is data auditing, which is when data anomalies associated with a dataset are detected. The process of identifying missing and erroneous values, duplicates, contradictions with schema constraints, and other data anomalies is very time-consuming. The automation of this phase can significantly reduce the overall time required for data cleaning. Various automated routines have already been developed for identifying and solving data quality issues. However, fully automated approaches to data

cleaning suffer from a number of limitations. Many of the developed algorithms for automated data anomalies discovery are able to identify potential data quality issues. But nevertheless, user participation cannot be completely excluded and human judgment is crucially important in the process of evaluating identified data quality issues and choosing an appropriate method to fix them [26].

Automated data auditing is closely related to the schema definition. A number of data quality dimensions described in Section 2.1.4 are defined as dataset schema types and constraint violations. Therefore, automated methods of detecting data quality issues require data worker to specify detailed schema on data.

When automated data auditing cannot help with detecting data anomalies, data workers need to inspect the dataset manually. In this scenario, an important role is played by appropriate data visualization [25]. In particular, good data ordering makes it easier to scan data values in order to identify anomalies. Thereby, in a sorted dataset, it is easier to notice "fuzzy" duplicates that can be missed by automated duplicate detectors, since similar records will appear adjacent to each other, or to identify extreme values, since they will appear at the beginning/end of a sorted dataset. Visualization of raw data and extensive support of custom dataset reordering significantly simplifies the process of manual data inspection, which is performed in most cases, even when the automated data auditing systems are available.

As a result of the first step of the data cleaning process, there should be a list of data anomalies residing in the audited dataset.

Definition of a Transformation Workflow

After the data have been audited, and information about data anomalies is known, the second phase, the definition of a transformation workflow, begins. During this phase, the data worker specifies operations to perform on data in order to eliminate data anomalies, enrich the data, or transform it into a form more suitable for further audit, publication, or consumption. This step implies close interaction with data worker since there are many ways to resolve data quality issues and, therefore, data worker should assign precise cleaning logic. Thus, in the case of missing data values, some default value may be used to replace null values, rows with missing values can be removed from the dataset, lacking values may be

calculated out from known information, etc.

During this phase data may also either be extended with additional attributes, making further data consumption more useful, or narrowed to a set of summaries on data. In the case of creating Linked Data from tabular data, the explicit mapping of tabular values to a schema vocabulary should also be specified in this phase.

The result of the definition of a transformation workflow phase is a set of operations, containing all necessary details about which steps should be executed on a dataset to obtain the desired result.

Execution of Transformation Workflow

The third step, transformation workflow execution, is performed after the definition of a transformation workflow. This step is executed automatically and the implementation of the workflow should be applied in an efficient manner to all the data that is intended to be transformed. In the context of this thesis, the expected input data are assumed to be large-scale datasets.

The output of this step is transformed data. Depending on the defined transformation workflow and desired form of the output, it can be either tabular data or RDF data.

Verification of Executed Transformation

The last step of data cleaning and transformation cycle is verification of executed transformation. Although some authors [36, 29] do not distinguish verification as a separate step of data cleaning and transformation, this phase is very important since it gives an evaluation of the performed transformation and determines a plan for further action. After the verification of the performed transformation, in some cases, transformed data may satisfy the defined data quality criteria, and cleaning and transformation cycle is finished. In other cases, it may still have some data quality issues, and the cycle of data cleaning and transformation starts again from the phase of auditing data to eliminate the rest of data anomalies.

3.2 Overview of Existing Approaches and Products

In the process of performing any type of technological research, it is very important to get acquainted with existing solutions for the researched problem. This helps to determine the work already done in the researched area, simplifies identifying present challenges and difficulties, and thus helps to set up the requirements for the new solution. Currently, available software products for data cleaning and transformation can be divided into several groups:

1. Spreadsheet software
2. Command line interface (CLI) tools
3. Programming languages and libraries for statistical data analysis
4. Complex systems designed to be used for interactive data cleaning and transformation in ETL process.

The first group is comprised of spreadsheet software tools. Indeed, spreadsheets are well-known to the most of the data workers, have a simple intuitive interface, and require no advanced technical skills for their usage. Examples of spreadsheet tools that can be used for tabular data cleaning are *Libre Office Calc*[1], *Microsoft Excel*[2], *Google Sheets*[3] and many others. A number of guidelines and recommendations are available for learning how to clean tabular data with spreadsheets. One good example is *"A Gentle Introduction to Data Cleaning"* series from the School of Data community of data workers[4]. An important feature, that makes spreadsheets very attractive for data workers is that spreadsheets are represented visually in tools. They display the input data, provide very simple and intuitive interface for data manipulation and reflect the performed changes on data immediately, providing the user with the possibility to verify the changes.

However, despite their simplicity and interactive design, spreadsheet software products have a number of limitations and disadvantages.

[1] https://www.libreoffice.org/discover/calc/

[2] http://www.microsoftstore.com/store/msusa/en_US/pdp/Excel-2016/productID.323021400

[3] https://www.google.com/sheets/about/

[4] http://schoolofdata.org/courses/#IntroDataCleaning last accessed May 18, 2016

Firstly, working with spreadsheets is error-prone. Perhaps, the one most well-known error made during spreadsheet data transformation occurred in Reinhart and Rogoff's austerity-justifying paper [37]. Two Harvard economists published a highly influential piece of work, which contained a wrong conclusion due to an erroneous Excel spreadsheet formula. Transformation workflow definition errors in spreadsheets are rather difficult to identify – data and transformation code are mixed together, significantly hindering the process of code review. Furthermore, conventional spreadsheets are typically limited in functionality and so are incapable of coping with the most sophisticated data quality problems. One more substantial disadvantage of spreadsheet tools is that they are not suitable for processing truly large amounts of data.

To conclude, spreadsheet software tools were not initially created for data cleaning, and, although they have a simple and attractive interface and allow to perform basic data transformations, these environments are not entirely suitable for processing large amounts of data.

Another broad group of tools for cleaning tabular data are command-line tools. Such tools are typically reliable, provide a broad set of functionalities, give an ability to automate data cleaning and conversion, and allow to make this task repeatable. Repeatability can significantly reduce time and cost data needed for transforming data.

One good example is *csvkit* [17], which is a suite of command-line tools for working with tabular data in CSV format. CSV is a common format used in many business, scientific and statistical applications. csvkit supports basic dataset reordering, filtering data, merging data from several datasets and generating summaries on columns. Another command-line tool for more detailed cleaning is *CSVfix*[5]. Compared to csvkit, this tool has more capabilities to manipulate the data, such as merging several columns into one, applying a standard function to the column values etc.

Although the aforementioned command line interface tools provide good functionality for data cleaning and in some cases even able to handle large volumes of input data, they suffer from lack of convenient user interface .

[5]http://neilb.bitbucket.org/csvfix/manual/csvfix16/csvfix.html

The tools from a third group, programming languages and libraries, include, for example, *Agate* [6] Python library for data analysis. This library provides powerful data cleaning and data analysis capabilities. In the context of data transformations, as part of data analysis, it is worth to mention the *R*[7] programming language for statistical computing and the data manipulation tools based on this language, e.g., *dplyr*[8] and *tidyr*[9]. The disadvantage of the tools from this group is that they require users to have considerable knowledge in programming.

Examples of relevant commercial ETL tools supporting powerful and efficient data transformations include *Pentaho Data Integration*[10] and *Trifacta Wrangler*[11]. These systems are designed specifically to support an ETL process and offer a number of useful data manipulation functionalities. However, they were not created to support Linked Data capabilities, and are thus of limited relevance to this thesis.

The system, most closely related to the research performed in this thesis, is *OpenRefine*[12], which is a free, open-source tool for data cleaning. OpenRefine provides an interactive user-friendly interface suitable for users with any level of technical competence. Through the installed *RDF Refine plugin*[13], OpenRefine makes it possible to assign RDF mappings and generate RDF data out of input tabular data. Nevertheless, the tool has notable limitations. Transformation functions are tightly coupled to the application's core and are thus not exposed as an API. Furthermore, the transformation engine uses a multi-pass approach to data transformation operations, thus, data manipulation is very memory-intensive, wich prevents usage with large-scale datasets.

After the evaluation of state-of-the-art solutions for data cleaning and transformation, it is possible to make a plan for the contribution to be made in the scope of this thesis. The resulted artifact should provide broad capabilities for data cleaning and transformation, support Linked Data generation, have a convenient user interface and be able to handle large-scale datasets. The next section provides

[6] https://agate.readthedocs.org/en/1.3.1/
[7] https://www.r-project.org/
[8] https://cran.r-project.org/web/packages/dplyr/index.html
[9] https://blog.rstudio.org/2014/07/22/introducing-tidyr/
[10] http://community.pentaho.com/projects/data-integration/ last accessed May 18, 2016
[11] https://www.trifacta.com/products/wrangler/ last accessed May 18, 2016
[12] http://openrefine.org/ last accessed May 18, 2016
[13] http://refine.deri.ie/ last accessed May 18, 2016

detailed requirements and success criteria for the product to be developed.

3.3 Requirements and Success Criteria

The final part of problem analysis is establishing a set of requirements for the software product to be developed. These requirements represent the descriptions of what the system should do, the services that it provides and the constraints on its operation. Functional requirements for a system depend, among other things, on the expected users of the software [43]. We divide the expected users of the developed platform in two broad groups:

- **Data publishers**, whose goal is to clean data and to prepare it for publication in tabular or Linked Data format.
- **Data consumers**, including data scientists, who perform data analysis, and developers, who create new applications and services requiring intensive use of published data.

Developers comprise a group of users, who have knowledge and experience in programming, and may be willing to embed data cleaning and transformation functionalities in their applications. Thus, they benefit from the software product exposing an API for some of its routines.

Other groups of users may have different experience and level of knowledge in programming. Although data scientists and data publishers may work in close collaboration with the IT department of their company, they are typically domain experts and their technical competence may vary [23]. These users have a broad knowledge about the data they work with, how they were assembled, how they should be interpreted, and which calculations should be performed on these data. Typically, data scientists and data publishers have to spend a large amount of time on repeating the same sequence of modifications on data when data is updated or new datasets are collected. These users will benefit from an interactive GUI and the possibility to re-use transformations with different input data.

In summary, the developed platform should have a comprehensive GUI to be appropriate for data publishers and data consumers who don't have much experience with using programming languages in their work. At the same time, it

should allow executing routines defined in an API or write custom code directly, to support data scientists who are proficient in using advanced technologies for data manipulation and developers, who wish to use platform functionalities in external applications. The desired components for the new software product and the way they associate with users are shown in Figure 3.2.

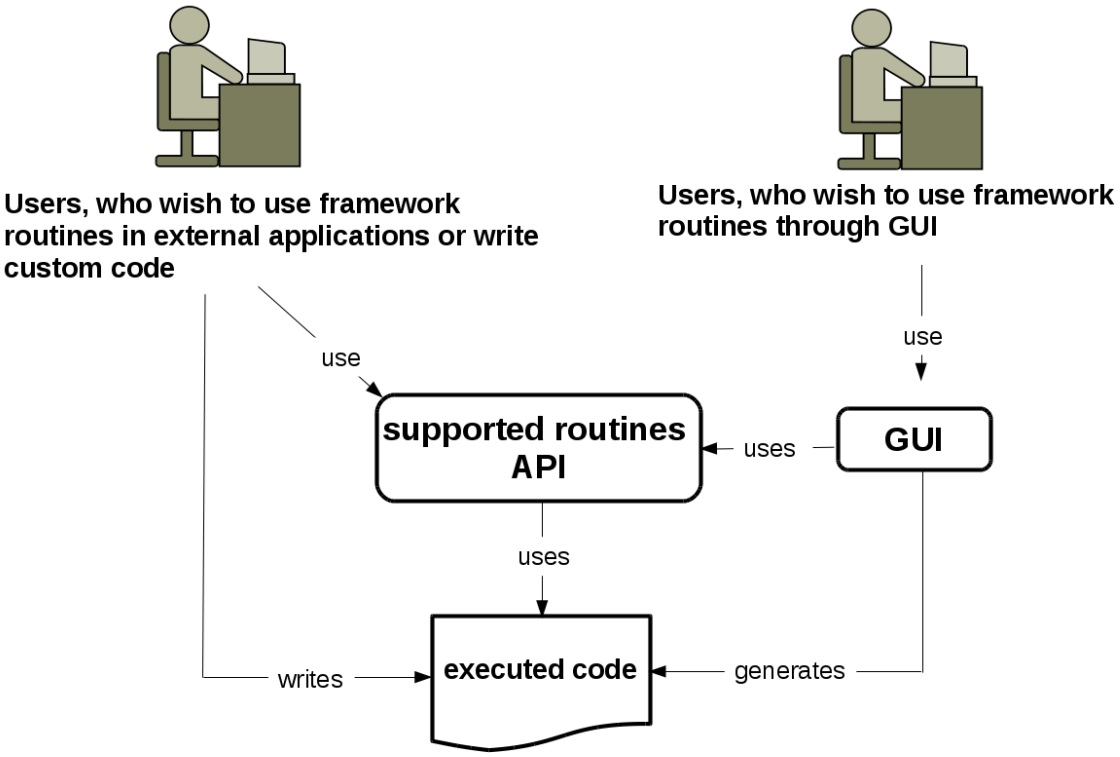

Figure 3.2: Main components of the new product

As a part of the technology research in this thesis, the aforementioned requirements are hereby stated in a way, which will make it possible to evaluate the software after the development.

Thus, the framework should:

R1. Provide routines to address the most typical data quality issues with tabular data.

R2. Provide routines to automate the generation of RDF data out from tabular data.

R3. Provide routines that are available for use by external applications.

R4. Provide routines that can handle large volumes of data.

R5. Have a comprehensive user interface that will make it possible to easily use the basic framework capabilities for users with no proficiency in programming.

R6. Have a comprehensive user interface that will simplify writing, debugging and use of custom code to perform complex data manipulations for data scientists with experience in programming.

R7. Provide input data visualization that will help in the visual detection of data anomalies.

R8. Support an interactive transformation workflow with automatic execution and instant feedback on changes to data.

R9. The transformation workflow should be easy to reuse.

Another source for discovering requirements is real-life scenarios [43]. After the prototype development is finished, it will be tested with several use cases, which may reveal new requirements for the tool or necessary corrections to existing ones.

Chapter 4

Grafterizer: A Flexible Framework for Tabular Data Cleaning and Linked Data Generation

4.1 Framework Overview

Grafterizer was developed as a web-based framework for data cleaning and transformations and integrated in the DataGraft platform. The core goal of Grafterizer is to support cleaning of tabular data and transforming it to an RDF graph. The general process it supports is shown in Figure 4.1.

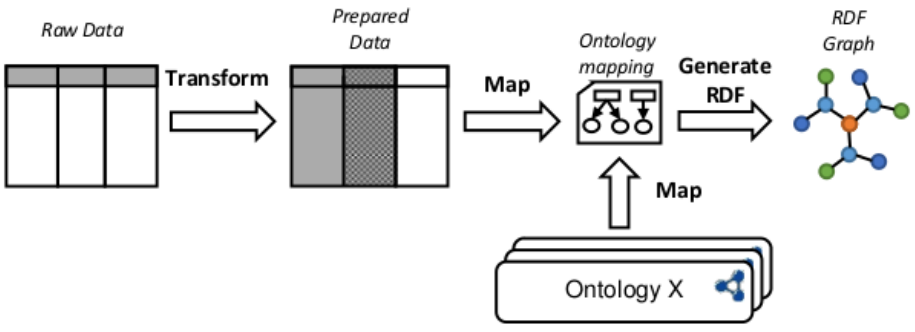

Figure 4.1: The process of generating a semantic graph from tabular data

Grafterizer is designed to support two types of transformations: *tabular-to-tabular* and *tabular-to-RDF*. Tabular-to-tabular transformations take tabular data in CSV format as an input and produce transformed tabular data in CSV format as an output. Tabular-to-RDF transformations take tabular data in CSV format as an input and produce RDF data in N-triples serialization format as an output. The way these two types of transformations can be performed in Grafterizer is discussed in Section 4.4.

4.2 Core Components

The two main components closely related to Grafterizer are *Grafter*[1] – a software library and domain specific language (DSL) for creating and executing data transformations – and *Graftwerk*[2] – a back-end service that supports exposing Grafter transformations "as-a-service" and provides a RESTful API for executing these transformations.

Graftwerk provides two core features:

1. Executing a specified transformation on the entire dataset and returning the results in chosen format (either tabular or RDF).

2. Executing a specified transformation on the subset of data and returning the results for this subset.

The second feature is essential in allowing an interactive live preview of the transformation.

Grafter is a powerful software library and DSL for producing linked data graphs from tabular data, which provides extensive support for tabular-to-tabular data conversions and powerful ETL data transformations, suitable for handling large datasets. Grafter was developed by Swirrl[3], a company focused on developing Linked Open Data solutions for the public sector. The Grafter suite of tools is implemented in Clojure[4] – a functional programming language and Lisp dialect

[1] http://grafter.org/
[2] https://github.com/proDataMarket/graftwerk
[3] http://www.swirrl.com/
[4] https://clojure.org/

that runs on the Java virtual machine (JVM). The use of the JVM allows Clojure to have access to the numerous libraries, available for JVM-based languages. Using a functional programming language, such as Clojure, is also is an advantage in terms of data processing. The benefits of this choice include:

- Functional programs typically operate on *immutable data structures*. Since the data structures cannot be modified, they can be shared without need to ensure concurrency, which allows more efficient memory use.
- Most of functional languages, including Clojure, support *lazy evaluation*. Lazy evaluation implies deferring the computation of values until they are needed, which helps to avoid unnecessary computations and allows to use infinite data structures.
- Functional languages use *higher-order functions*. This means ability to process code as data and improves program modularity.

The grafter consists of several modules, that encapsulate its various functionalities. The schema of Grafter's architecture and its interaction with Grafterizer can be seen in Figure 4.2.

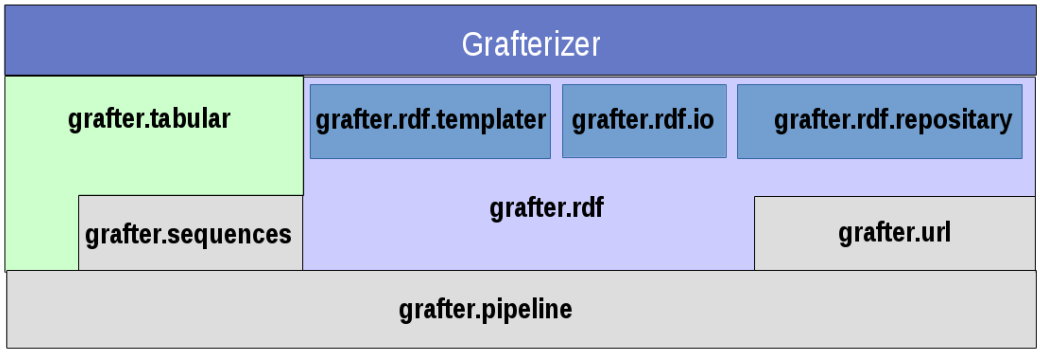

Figure 4.2: Grafter's architecture and Grafterizer

The two modules that are most relevant for this thesis are defined in namespaces **grafter.rdf** and **grafter.tabular**. These modules are concentrated on supporting two fundamental sides of Grafter transformations: cleaning tabular data (tabular-to-tabular type of conversion) and transformation to Linked Data (tabular-to-RDF type of conversion).

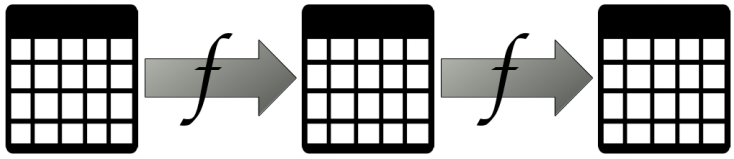

Figure 4.3: Pipes, performing tabular-to-tabular transformations

4.2.1 Grafter Pipes

The tabular data transformation process in Grafter is realized through a pipeline abstraction, i.e., each step of a transformation is defined as a **pipe** – a function that performs simple data conversion on its input and produces an output. These functions are composed together in a pipeline, whereby the output of each pipe serves as input to the next (see Figure 4.3).

Pipeline functions may be combined arbitrarily, whereby each combination produces another pipeline. Each of these pipes is a pure Clojure function from a *Dataset* to a *Dataset* , where *Dataset* is a data structure used to handle data in Grafter. The example can be seen in a code sample below:

```
1 -> (read-dataset data-file)
2    (drop-rows 1)
3    (make-dataset move-first-row-to-header)
4    (mapc {:gender {"f" (s "female")
5                    "m" (s "male")}}))
```

Listing 4.1: Example of Grafter's pipeline

Each line from the example above is a function call that takes a Dataset and, optionally, other parameters as an input, and returns a modified Dataset.

The functions' uniformity in terms of input and output makes it very intuitive for users to use pipelines for data manipulation.

4.2.2 Grafter Grafts

In order to publish dataset as Linked Data, the cleaned dataset needs to be converted into a graph structure. While cleaning a dataset as tabular-to-tabular transformation is handled by pipes, the tabular-to-RDF transformation is performed with help of another type of Grafter function – **graft** (see Figure 4.4). A graft takes a Dataset as input and produces a sequence of *Quads*. Quad is a data unit similar to triples consisting of a subject, predicate, object and extended by context.

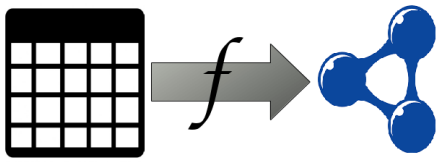

Figure 4.4: Graft, performing tabular-to-RDF transformations

Quads are formed from rows in a dataset. In addition to the dataset itself, grafts receive a mapping specification, where quads elements (subjects, predicates, objects and context) and their relations to dataset columns are described. The example of a graft template can be seen in a code sample below:

```
1 (def make-graph
2   (graph-fn [{:keys [name sex age person-uri gender]}]
3             (graph (base-graph "example")
4                    [person-uri
5                     [rdf:a foaf:Person]
6                     [foaf:gender sex]
7                     [foaf:age age]
8                     [foaf:name (s name)]])))
```

Listing 4.2: Example of Grafter's pipeline

This sample of code shows use of **graph-fn** function. In the second line the arguments to this function, which are dataset columns involved in the mapping, are specified. The lines 3-8 represent the body of the function. The body of the function is structured like this:

```
(graph graph-uri
       [subject1 [predicate1 object1]
                         [predicate2 object2]]
       [subject2 [predicate3 object3]])
```

Listing 4.3: The structure of graph-fn function

The *graph-uri* variable defines a context for the generated quads, whereas their subjects predicates and objects are defined in nested vectors. The graph-fn function generates the sequence of quads according to the specified mapping for the each row of the dataset.

4.3 The Grafterizer Transformation Functions

The Grafter library provides a set of useful functions for processing tabular data[5]. However, some of the common data anomalies listed in Table 2.2 either cannot be solved with help of functions provided by Grafter, or require more complex, data-dependent combinations of Grafter functions. The intuitive solution to this problem is to extend the Grafter library for tabular transformations with routines performing the basic data transformation operations.

The designed set of routines should be usable as an extension to the Grafter DSL and suitable for use when implementing graphical user interface components. Hence, this set of routines may be referred to as application programming interface (API), and the general guidelines for API development can be used during the design and implementation process[6].

Thus, the developed set of routines must satisfy the following requirements:

1. *Completeness* implies that the set of routines should be complete and support all the necessary routines.

2. *Understandable and unambiguous routine names and specifications* make easier to memorize the API routines and understand purpose of their use.

3. *Readability of resulting code* makes the API easier to maintain.

[5] http://api.grafter.org/docs/0.6.0/grafter.tabular.html

[6] http://people.mpi-inf.mpg.de/~jblanche/api-design.pdf last accessed May 18, 2016

4. *Safety of use.* A well-designed routines should be capable of anticipating, detecting and resolution of errors during the execution.

5. *Extensibility* implies ease of adding new routines to the set of already exposed.

To ensure the first requirement, completeness, it is necessary to know the full set of tabular data transformation operations, that should be supported. Based on performed research on data anomalies, and, taking into consideration the features supported by existing related systems, the basic operations on tabular data were identified. This set is partially covered by Grafter capabilities, however, a significant part of essential data transformation operations do not have corresponding functions in the Grafter API (see Table 4.1). The missing operations comprise the set of routines that are developed as part of this thesis.

The understandable and unambiguous routine names and specifications should be defined after obtaining the full set of operations to be supported. The specification of developed routines among with descriptions and examples is summarized in Appendix B.

The readability of resulting code, safety of use and extensibility depend on the correctness and good style of implementation. The developed routines have been deployed at Clojars – a public repository for open source Clojure libraries[7].

[7] https://clojars.org/grafterizer/tabular_functions

Table 4.1: Summary of basic tabular transformations

Scope	Name	Description	Function name	Supported by Grafter	Application
Rows	Add Row	Create a new record in a dataset	add-row	no	Dataset enrichment
	Take Rows	Extract a selected row (sequence of rows)	take-rows rows	yes	Resolves anomaly: *"Rows, describing entities not belonging to a collection"*
	Drop Rows	Delete a selected row (sequence of rows)	drop-rows rows	yes	Resolves anomaly: *"Rows, describing entities not belonging to a collection"*
	Shift Row	Change a row's position inside a dataset	shift-row	no	Reordering, simplifies anomaly detection
	Filter Rows	Filter rows for exact matches, regular expressions, empty values, etc.	grep	yes	Resolves anomaly: *"Rows, describing entities not belonging to a collection", "Missing values"**
	Remove Duplicates	Remove similar rows based on certain column or set of columns	remove-duplicates	no	Resolves anomaly: *"Duplicate rows"*
Entire dataset	Sort Dataset	Sort dataset by given column names in given order	sort-dataset	no	Reordering, simplifies anomaly detection

(Continued on Next Page)

Scope	Name	Description	Function name	Provided by Grafter	Application
Entire dataset	Reshape Dataset (Melt)	Move columns to rows	melt	yes	Resolves anomaly: *"Column headers containing attribute values"*
	Reshape Dataset (Cast)	Move rows to columns by categorizing and aggregating	cast	no	Dataset enrichment, simplifies anomaly detection
	Group and Aggregate	Group values by column or multiple columns and perform aggregation (get minimum, maximum or average value, count or sum values in every group) on the rest of columns	group-rows	no	Dataset enrichment, simplifies anomaly detection
Columns	Add Column	Add a column with a manually specified value	add-column add-columns	yes	Dataset enrichment
	Derive Column	Add a column with values computed from other columns	derive-column	yes	Dataset enrichment
	Take Columns	Take selected column(s)	columns	yes	Resolves anomaly: *"Column headers not related to model"*
	Drop Columns	Drop selected column(s)	remove-columns	no	Resolves anomaly: *"Column headers not related to model"*

(Continued on Next Page)

Scope	Name	Description	Function name	Provided by Grafter	Application
Columns	Shift Column	Change columns' order	shift-column	no	Reordering, simplifies anomaly detection
	Merge Columns	Merge columns using custom separator	merge-columns	no	Resolves anomaly: *"Single value is splitted across multiple columns"*
	Split Column	Split column using custom separator	split	no	Resolves anomaly: *"Multiple values stored in one column"*
	Rename Columns	Change column headers	rename-columns	yes	Resolves anomaly: *"Incorrect column headers"*
	Map Columns	Apply function to all values in a column	mapc	yes	Resolves anomalies: *"Illegal values", "Erroneous values", "Inconsistent column values", "Missing values", ""*
	Convert Datatype†	Cast values in a column to specified datatype	convert-datatype	no	Resolves anomalies: *"Illegal values"*

*Missing values anomaly can be resolved by filtering rows if data worker's cleaning logic implies removing rows with missing values.
†Convert datatype function is oriented to XSD datatypes and therefore is used during RDF mapping rather than during tabular cleaning.

4.4 The Grafterizer Graphical User Interface Design

The Grafter library is primarily targeted at software developers. As discussed in Section 3.3, a significant part of expected Grafterizer framework users are domain experts, who may have no or little experience in software development. However, one of the fundamental requirements for the framework is to have a comprehensive user interface. Thus, an essential part of the practical work in this thesis is the implementation of Grafterizer's GUI.

Grafterizer is designed as a web application and implemented in Angular JS[8]. The list of technologies used in the development process includes:

- **Grafter** library and DSL for data transformations.
- **Grafwerk** back-end service for executing Grafter transformations.
- **AngularUI**[9] suite for building user interfaces for developers using Angular JS.
- **Angular Material**[10] UI framework and implementation of Google's Material Design[11] Specification.
- **jsedn**[12] library for parsing and generation of Grafter/Clojure code.

The resulting framework supports building Grafter transformations in an easy and interactive way. The basic functionalities are as follows:

Defining and editing data transformation workflows – operations of a transformation workflow can be easily added, edited, reordered or removed. All operations are defined with parameters, which can be changed at any time when designing the transformation.

[8] https://angularjs.org/
[9] http://angular-ui.github.io/
[10] https://material.angularjs.org/latest/
[11] https://www.google.com/design/spec/material-design/introduction.html#introduction-goals
[12] https://github.com/shaunxcode/jsedn

Live preview – the framework interactively displays the transformed dataset, thus supporting a real-time evaluation of a defined transformation workflow. Live preview feature also supports error reporting by notifying users about errors during transformation execution in a pop-up window.

Sharing and reusing the data transformation workflows – this feature is supported by the integration with the DataGraft platform. Transformations created with the help of Grafterizer are stored as a sequence of operations on data and can be reused and copied.

Grafterizer's user interfacfe(Figure 4.5) consists of a preview panel (on the right side) and transformation definition panel (on the left side).

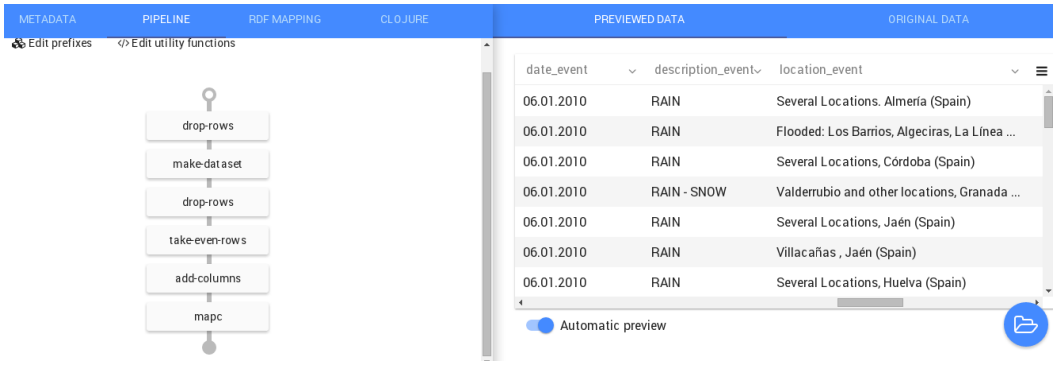

Figure 4.5: The screenshot of Grafterizer's GUI

The transformation definition allows specifying transformation metadata and the two types of Grafter's transformations: tabular data cleaning and converting tabular data to RDF. Furthermore, it displays the generated Clojure (Grafter) code.

Tabular-to-tabular transformations are specified in a visual presentation of a Grafter pipeline, clearly demarcating the order of pipeline functions, used to process the data. These functions include both Grafter routines and routines from the extensions to Grafter developed as part of this thesis. The interface allows users to specify parameters to functions (Figure 4.6) with a possibility to edit them later on. When these parameters are edited, the preview shows the result immediately. The user is also able to preview the state of the dataset on each step of the transformation. A possibility to edit pipeline functions' parameters, change the order

of pipeline functions and get immediate feedback simplifies a process of defining the transformation workflow and the transformation workflow verification.

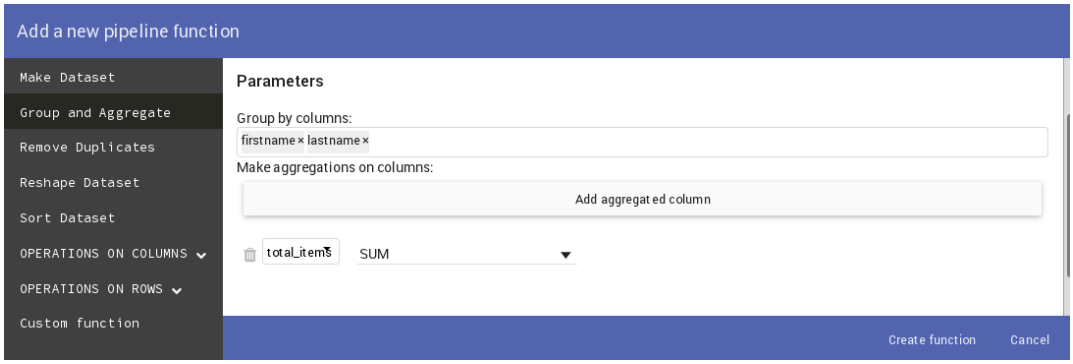

Figure 4.6: Adding a new pipeline function

Due to the inevitable variations in data quality issues, operations on data have not been limited by the functions listed in previous section. Grafterizer allows users to define their own custom functions on data (Figure 4.7) and include them in the transformation pipeline. Having user-defined custom code as independent utility functions provides an essential flexibility in transforming data, helps to encapsulate transformation logic and makes it possible to reuse the utility functions at different points of a transformation.

Figure 4.7: Adding a new utility function

The general cycle of data cleaning and transformation, discussed in Section 3.1, corresponds well to Grafterizer's workflow. The first step of the process, data auditing, is performed by visual inspection of a dataset, which is supported by the interactive preview functionality and aided by various reordering functions. The definition of a transformation workflow is implemented as the process of composing a transformation pipeline. The execution of a transformation workflow takes place simultaneously with pipeline composition and the results are instantly visualized in a live preview, thus making it possible to verify the transformation.

After data quality issues are solved, the dataset can be transformed to the Linked Data graph. The RDF triple patterns that should appear in the resulting linked data are designed by the user, whereby all triples' subjects, predicates and objects are specified manually through a mapping procedure. During the mapping process column headers are mapped to RDF nodes in order to produce a set of triples that corresponds to each data row (Figure 4.8).

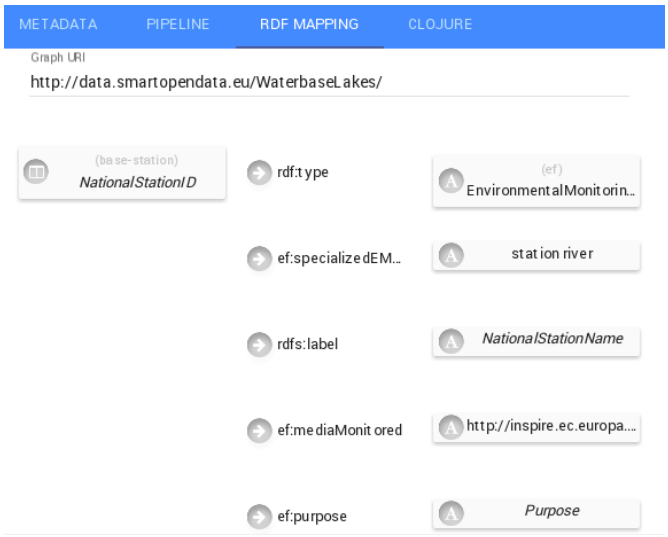

Figure 4.8: RDF mapping in Grafterizer

Grafterizer supports reuse of existing RDF ontologies by providing a searchable catalog of vocabularies and makes it possible to manage individual namespace prefixes. Each column in a dataset can be mapped as a URI node with namespace prefix assigned by user or literal node with a specified datatype. Grafterizer also provides support for error handling when casting to datatypes. To support

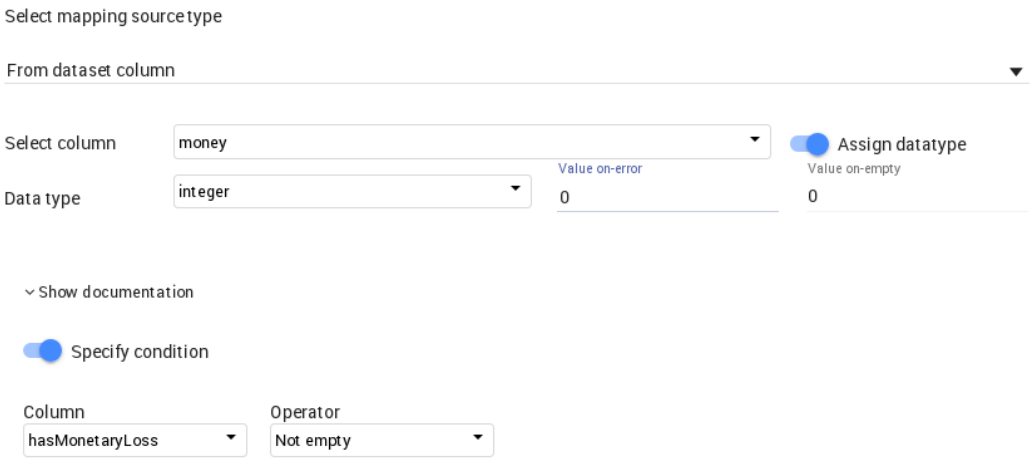

Figure 4.9: Casting to datatypes and assigning conditions during RDF mapping

this, a routine for casting values to specified datatype was implemented among with other developed Clojure functions. In addition, users may assign condition(s) under which a triple or entire sub-graph should be generated (Figure 4.9).

If the RDF mapping is present in the transformation, Grafterizer automatically performs tabular-to-RDF transformation. If the mapping is omitted, the executed transformation will return tabular dataset.

To summarize, the innovation phase as a part of the technology research in this thesis resulted in:

- Specifying the set of tabular data transformation operations that should be supported.
- Implementing the subset of tabular data transformation operations, that are not supported by Grafter DSL.
- Implementing the Grafterizer user interface in collaboration with the team of software developers.

Chapter 5

Evaluation

This section performs a detailed evaluation, whereby the developed framework is compared with relevant software products with similar capabilities, chosen approach to Linked Data generation is evaluated with resprect to the alternative one, and the software is tested with real-life datasets.

5.1 Comparative Evaluation of Data Cleaning Capabilities

During the process of collecting the requirements for the artifact, several related software products were examined. These tools served as the input to the specification of operations on data that should be supported by the developed solution and made it possible to evaluate Grafterizer in the context of subset of tools with similar functionalities.

The software products used in the evaluation were chosen with respect to their capabilities to perform data cleaning and transformation operations, expected input (tabular data), the basic purpose of their operation and their popularity among data workers. To define candidate products for the comparison, several recommendations from various communities were taken into account[1].

[1] https://theodi.org/blog/tools-for-working-with-csv-files last accessed May 18, 2016
https://multimedia.journalism.berkeley.edu/tutorials/cleaning-data/ last accessed May 18, 2016
http://govhack-toolkit.readthedocs.org/technical/tabular-data/ last accessed May 18,

The relevant tools chosen for the comparison can be categorized as follows:

- Command line interface tools:
  - *csvkit 0.9.1*[2] – suite of utilities for converting to and working with CSV data format.
  - *CSVfix 1.6*[3] – command-line stream editor for CSV data.
- Programming language library for data analysis:
  - *Agate 1.3.1*[4] – Python data analysis library.
- Spreadsheet software:
  - *Microsoft Excel 2016*[5] – spreadsheet software for data manipulation and data analysis.
- Complex systems designed to be used for interactive data cleaning and transformation in ETL process:
  - *Trifacta Wrangler*[6] – an interactive tool for data cleaning and transformation, part of Trifacta's data preparation platform.
  - *OpenRefine 2.6*[7] – a desktop application for data cleaning and transformation .
  - *Grafterizer 0.4.1*[8] – a web-based framework for data cleaning and transformations, part of DataGraft platform for data transformation and publishing.

2016
http://digitalarchaeology.msu.edu/kb/2013/09/17/data-cleaning-transformation-management-tools/ last accessed May 18, 2016
https://www.quora.com/What-are-the-best-data-cleansing-tools last accessed May 18, 2016

[2] https://csvkit.readthedocs.org/

[3] http://neilb.bitbucket.org/csvfix/manual/csvfix16/csvfix.html

[4] https://agate.readthedocs.org/en/1.3.1/

[5] http://www.microsoftstore.com/store/msusa/en_US/pdp/Excel-2016/productID.323021400

[6] https://www.trifacta.com/products/wrangler/

[7] http://openrefine.org/

[8] https://datagraft.net/

Since the investigated software tools were designed for slightly different target user groups, the inspected features were categorized in correspondence with their effect on data. The categories are:

- *Data reordering* as a set of operations changing input data order.
- *Data extraction* as a set of operations taking a subset of data elements from input data.
- *Data manipulation* as a set of operations changing values in input data.
- *Data enrichment* as a set of operations adding new values to a dataset.
- *Data examining* as a set of operations performing some analysis on a dataset without changing input data.

The thorough comparative summary of features available in most popular tools for data manipulation is given in Table 5.1.

Table 5.1: Comparative summary of basic features supported by most used data cleaning and transformation tools

Feature		Solutions without GUI			Advanced data manipulation systems with GUI			
		Agate (Python library)	**csvkit (CLI tool)**	**CSVfix (CLI tool)**	**MS Excel (Spreadsheet)**	**Trifacta Wrangler**	**OpenRefine**	**Grafterizer**
Reordering	Sort data by one or several columns	*order_by* method	*csvsort* command, impossible to choose between sort types	*sort* command		impossible to sort by several columns and to choose between sort types		
	Reorder columns	Requires listing all the columns in the desired order	Requires listing all the columns in the desired order	Requires listing all the columns in the desired order				
	Reorder rows manually based on their positions							
	Reshape dataset	*pivot*, *normalize* and *denormalize* methods			May require different ways of implementation depending on data		May require different ways of implementation depending on data	
Extraction	Take a subset of rows by value or condition (filtering)	*where* method	*csvgrep* command	*find* command				
	Take a subset of rows by their positions	*limit* method, hard to choose rows by numbers	only allows to select rows to database		possible using macros			
	Remove duplicates	*unique* method		*unique* command				
	Take a subset of columns		*csvcut* command	*exclude* command				

(Continued on Next Page)

Feature		Solutions without GUI			Advanced data manipulation systems with GUI			
		Agate (Python library)	**csvkit (CLI tool)**	**CSVfix (CLI tool)**	**MS Excel (Spreadsheet)**	**Trifacta Wrangler**	**OpenRefine**	**Grafterizer**
Manipulation	Modify a column by applying a standard or custom function	possible as: compute new column → remove old column → rename new column		*eval* and *edit* commands				
	Rename columns manually	*rename* method		*summary* command				
	Rename columns by applying a function to a current column name							
	Merge values from several columns			*merge* command			possible using GREL code	
	Split values from one column to several columns			*split_char* command				

(Continued on Next Page)

Feature		Solutions without GUI			Advanced data manipulation systems with GUI			
		Agate (Python library)	**csvkit (CLI tool)**	**CSVfix (CLI tool)**	**MS Excel (Spreadsheet)**	**Trifacta Wrangler**	**OpenRefine**	**Grafterizer**
	Edit cell values manually (refer particular cell)							
Enrichment	Concatenate several datasets	*merge* method	*csvstack* command	*file_merge* command	possible using macros		possible using GREL code	currently unavailable in Grafterizer UI, but possible to use via *join-dataset* function
	Relational join of several datasets	implemented in *agate-lookup* extension	*csvjoin* command	*join* command	possible using MS Query		possible using GREL code	currently unavailable in Grafterizer UI, but possible to use via *join-dataset* function
	Group and aggregate for one or several columns	*group_by* method	*csvgroup* command	*summary* command	May require different ways of implementation depending on data		possible using GREL code	
	Add new column (computed or with custom values)	*compute* method		only possible to add empty or fixed string				
	Add new row with manually specified values						only possible through adding new column and then transposing a dataset	

(Continued on Next Page)

Feature		Solutions without GUI			Advanced data manipulation systems with GUI			
		Agate (Python library)	**csvkit (CLI tool)**	**CSVfix (CLI tool)**	**MS Excel (Spreadsheet)**	**Trifacta Wrangler**	**OpenRefine**	**Grafterizer**
Examining	Get statistics on a dataset, identify outliers	*stdev_outliers* and *Summary* methods	*csvstat* command	*stat* command			Faceting may help in identifying outliers	
	Check for spelling errors					possible through finding similar records	possible through finding similar records	
	Find similar records							

63

To summarize, Grafterizer supports most of basic functionalities provided by other relevant data cleaning and transformation tools.

5.2 Comparative Evaluation with the R2RML Approach

As part of our comparative evaluation of the RDF mapping approach we examine R2RML (RDB to RDF Mapping Language) – a language for expressing customized mappings from relational databases to RDF datasets [11]. In contrast to the approach in Grafterizer, the R2RML approach requires the input data to be stored in a relational database.

The R2RML approach is a rather attractive because the W3C recommendations provide standardised specifications for converting from relational databases to semantic web data [11, 3] and a number of tools are available for performing such a conversion [44]. However, due to the requirement to define a database schema and to convert data into the relational database standards, it becomes inflexible when more complex data transformations are required. Building a comprehensive user interface using R2RML significantly depends on the details of the implementation.

To conduct a comparison, we examined data provided by the Linked Open NPD FactPages project[9], that performs conversion of the Norwegian Petroleum Directorate's FactPages[10] to Linked Open Data using the R2RML [41]. In terms of the project, 70 tabular datasets (in the form of CSV files) were cleaned and loaded to a MySQL[11] relational database, after this a D2RQ[12] map was generated and used to dump database to RDF. Some steps of the data manipulation in this project required a great effort:

- Creating a relational schema for the database. This required manual specifying of the database's tables, columns, column datatypes, primary and foreign keys.

[9] http://sws.ifi.uio.no/project/npd-v2/ last accessed May 18, 2016
[10] http://factpages.npd.no/factpages/
[11] http://www.mysql.com/
[12] http://http://d2rq.org/

- Prior to the loading data files into a relational database, they had to be cleaned, whereby, for example, date and time values were converted into the correct format for MySQL; values used to indicate null ("NA", "not available", "n/a", "N/A", "" and "NULL") set to the database null values; values used to indicate unknown values ("?", "Not known") were transformed to the standard string "UNKNOWN"; empty date values were converted to "9999-12-31" date value.

- The identifiers of classes and properties in automatically generated D2RQ map are too related to the database terms, and corresponding URIs are not informative. For this reason, D2RQ map was edited manually [41].

The aforementioned project conclusion expresses some difficulties of data transformation:

> *Significant amounts of additional (largely manual) effort were needed to produce good quality RDF data, but this was due at least in part to quality issues relating to the source data [41].*

To find out whether or not Grafterizer can solve these difficulties, a subset of 5 tabular datasets was cleaned, mapped to the schema corresponding to D2RQ map used in the aforementioned project and converted to an RDF graph. Example screenshots for the created Grafterizer pipeline and RDF mapping can be seen in Figure 5.1 and Figure 5.2 correspondingly.

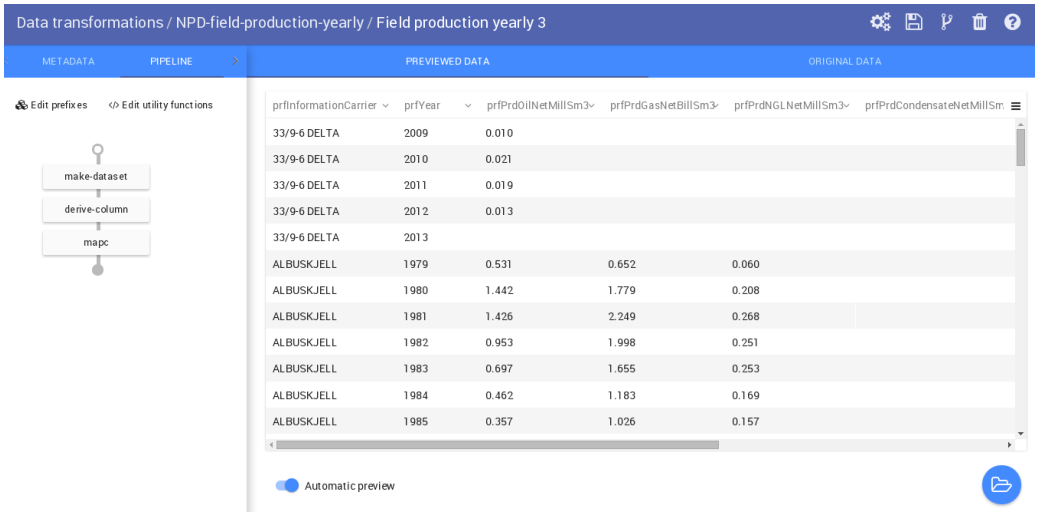

Figure 5.1: The screenshot of NPD Fact Pages Grafterizer pipeline

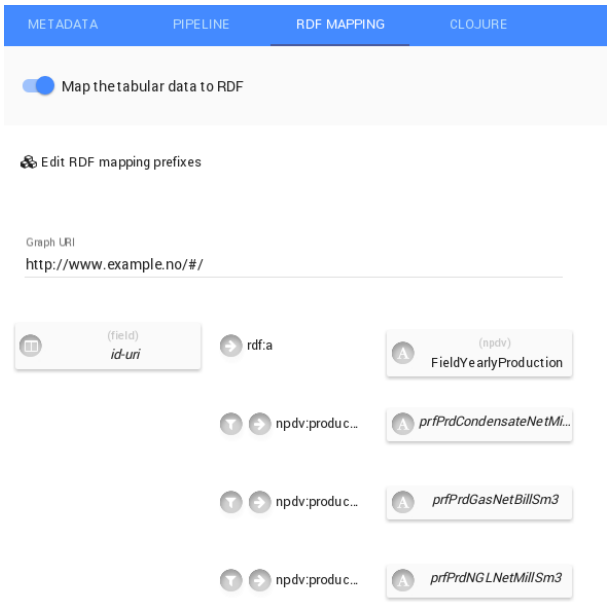

Figure 5.2: The screenshot of NPD Fact Pages RDF mapping

The data cleaning operations, that were performed manually in the Linked Open NPD FactPages project, were performed with help of Grafterizer data cleaning functions. This allows the defined transformation to be reused if the datasets will be changed over time. Furthermore, Grafterizer provides a convenient interface for the datatype conversion with the possibility to specify the output values that will be used for empty or erroneous input values. This datatype conversion can be defined directly during the RDF mapping. In addition, the RDF generation with Grafterizer doesn't require having a relational database that excludes difficulties with creating the database schema.

Thus, the conducted comparison makes it possible to conclude that the approach implemented in Grafterizer can significantly simplify tabular data cleaning and allows faster data transformation by avoiding the overhead of having a step of storing data in relational database.

5.3 Use Case Testing

This section contains short descriptions of real-life use case scenarios, in which the Grafterizer framework was used to clean the input data and convert it to RDF form. Each subsection contains basic information about the project, the number of input tabular datasets, the encountered limitations and resulting improvements in Grafterizer.

PLUQI: Personalized and Localized Urban Quality Index

Personalised Localised Urban Quality Index (or PLUQI for short) is a Web application, that was developed by Saltlux[13] company. PLUQI provides a visualization of level of well-being and sustainability for the set of cities based on individual preferences (Figure 5.3). One of the challenges associated with this project was to collect and integrate together various data related to life quality in the points of interest. The collected data was represented by tabular datasets, which had to be cleaned and converted to Linked Data.

[13] http://saltlux.com/

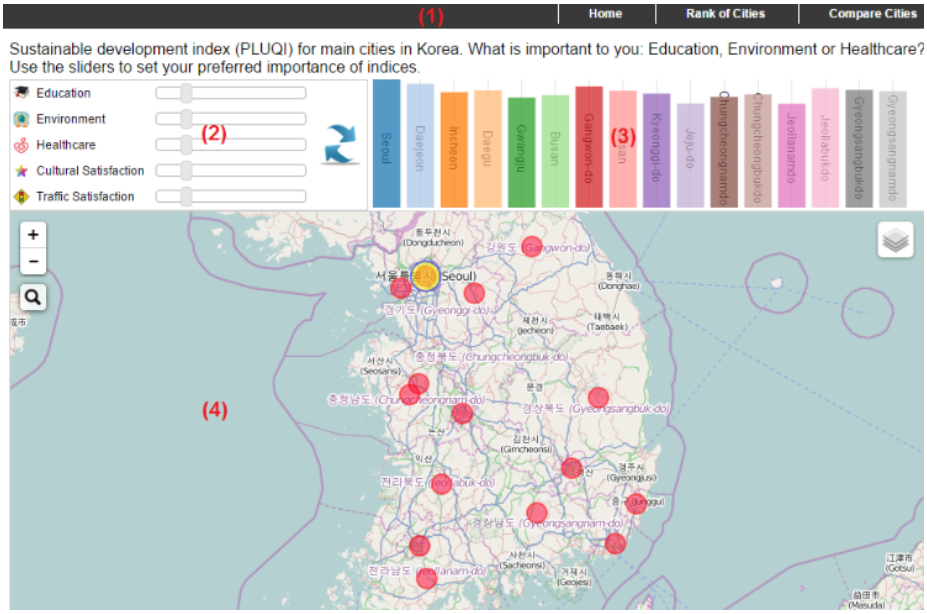

Figure 5.3: PLUQI application screenshot

The main domains, considered in terms of life quality are:

- Daily life satisfaction: weather, transportation, community, etc.
- Healthcare level: number of doctors, hospitals, suicide statistics, etc.
- Safety and security: number of police stations, fire stations, crimes per capita, etc.
- Financial satisfaction: prices, income, housing, savings, debt, insurance, pension, etc.
- The level of opportunity: jobs, unemployment, education, re-education, economic dynamics, etc.
- Environmental needs and efficiency: green space, air quality, etc.

The Grafterizer framework was used to prepare data and transform it to RDF. Input datasets include:

- Cultural facilities: 4 datasets

- Traffic equipment: 1 dataset
- Green space: 2 datasets
- Highschools: 1 dataset
- Crime statistics: 1 dataset
- Highschools: 1 dataset.

Environmental Data (Smart Open Data)

Grafterizer was used to perform data transformations on datasets in the SmartOpenData[14] project. The datasets hold information from the biodiversity and environment protection domains. Raw tabular data were cleaned and transformed to RDF according to a model defined specifically for the project. Input datasets for SmartOpenData project come from a total of 47 CSV files, which includes 42 TRAGSA Pilot files and 5 ARPA Pilot files. During the Smart Open Data use case scenario, the required data transformations were performed as well via another system for data transformation – OpenRefine.

InfraRisk: Identifying Critical INFRAstructure at RISK from Natural Hazards

InfraRisk[15] is a framework to identify and track the impact of natural hazards on infrastructure networks (e.g. roads, rails). The knowledge base for the project was created from 6 tabular datasets, cleaned and transformed to RDF with the help of Grafterizer.

Other Examples

Other cases of using Grafterizer to clean tabular data and convert it to RDF include, for example, processing property-related data provided by Statsbygg[16] , the

[14] http://www.smartopendata.eu/
[15] http://www.infrarisk-fp7.eu/
[16] http://www.statsbygg.no/

Norwegian government's key advisor in construction and property affairs. In this case data about public buildings in Norway was integrated with external information about accessibility in buildings. The goals of this integration are to enrich the currently available information about public buildings in Norway, and provide a more efficient mechanism to share this data with external organisations and the general public.

One more example of use, that is worth to mention is transforming data from the CITI-SENSE[17] project. In this scenario, Grafterizer was used to clean and transform air quality sensor data.

Resulting Improvements and Evaluation Summary

During testing with these use cases, the need for implementing additional functionalities was identified. The PLUQI use case scenario has revealed a need to implement an interface for parameterizing user-defined utility functions. The Smart Open Data use case scenario revealed the need to implement logic and interface for sorting the dataset and removing duplicates. The improvements in RDF mapping part as a result of tool evaluation with this use case scenario were in implementing a user interface for creating language-tagged string literals. The work with the Infrarisk use case scenario revealed the need for assigning RDF mappings conditions. The conditions evaluate specified column values for each row, whereas the output of conditional statement defines whether an RDF statement will be created for this row.

After improving Grafterizer with the aforementioned functionalities, it was possible to perform the data cleaning for all used input tabular datasets and generate valid RDF data from them.

[17] http://www.citi-sense.eu/

Chapter 6

Conclusion

As a problem statement for this thesis, there were identified challenges of simple integration of heterogeneous data sources. One attractive high-level solution to easily and seamlessly merge the datasets is to follow the Semantic Web specifications and publish data in the Linked Data format. This requires converting information about entities and relationships between them into the standardized machine-readable RDF form.

At the same time, most the data consumers and data publishers spend too much time and effort for data cleaning. Currently, the process of generating Linked Open Data is additionally hindered by the absence of the unified framework for data cleaning and transformation to RDF. In addition, the complexity of available tools limits the opportunities for data manipulation for data workers with a lack of programming experience.

To alleviate these problems, this thesis contributes with essential improvements to an available framework for data cleaning and transformation to graph data, providing the necessary functionalities to cope with common data cleaning problems, while at the same time remaining simple – enough that non-programmers can use it and flexible – enough that data developers can easily work with it. The results of this thesis [45] have been accepted to the Extended Semantic Web Conference[1] (ESWC) – the international conference on Semantic Web technologies.

[1] http://2016.eswc-conferences.org/

6.1 The Evaluation of Performed Work in Accordance with the Requirements

To evaluate the performed work, it is necessary to refer to the set of requirements, defined in Section 3.3. According to these requirements, the developed artifact should:

R1. *Provide routines to address the most typical data quality issues with tabular data*. This requirement is satisfied with respect to the data quality issues discussed in Section 2.1.5. The developed artifact supports the set of tabular operations to resolve most common tabular data quality issues (Section 4.3).

R2. *Provide routines to automate the generation of RDF data out from tabular data.* The developed artifact allows generation of RDF data out from tabular data through the Grafter **graph-fn** function (Section 4.2.2).

R3. *Provide routines that are available for use by external applications.* The routines for tabular data cleaning, as well as the routines for generating RDF data are implemented as openly available and well-documented functions, which allows to use them by external applications.

R4. *Provide routines that can handle large volumes of data.* The developed routines can be effectively executed on large volumes of data. However, the Grafterizer user interface has the limitation on the size of uploaded data (data files shouldn't be larger than 10 MB each).

R5. *Have a comprehensive user interface that will make it possible to easily use the basic framework capabilities for users with no proficiency in programming.* The Grafterizer user interface supports specifying a transformation without using custom code. Specifying the transformation only through the GUI for the functions discussed in Section 4.3 can eliminate the most common data quality issues. However, more sophisticated transformations still require the user to have an experience in programming.

R6. *Have a comprehensive user interface that will simplify writing, debugging and use of custom code to perform complex data manipulations for data*

scientists with experience in programming. The Grafterizer user interface supports writing, debugging and using a custom code in the transformation through the "Edit utility functions" capability (Section 4.4).

R7. *Provide input data visualization that will help in the visual detection of data anomalies.* The Grafterizer interactive preview supports data visualization, which helps in the visual detection of data anomalies (Section 4.4). Several supported transformation functions also may be of help in the detection of data anomalies (Sort Dataset, Shift Row, Shift Column, etc.). However, Grafterizer doesn't support the automated anomaly detection. This functionality can be identified as a direction for future work.

R8. *Support an interactive transformation workflow with automatic execution and instant feedback on changes to data.* The Grafterizer interactive preview provides an immediate feedback on changes on data, that simplifies the verification of executed transformation.

R9. *The transformation workflow should be easy to reuse.* Grafterizer treats the defined set of operations on tabular data, declared custom utility functions and RDF mappings as a single transformation object. Thus, being integrated into DataGraft, it makes possible to reuse and share the transformation.

The performed evaluation results in the conclusion that Grafterizer is a viable product, capable of performing complex data cleaning and transformation operations.

6.2 Directions for Future Work

Although the developed artifact satisfies the stated requirements and capable of performing data cleaning and transformation with real-life datasets, there are still many research opportunities. This section outlines potential areas for improvement in future versions of Grafterizer.

6.2.1 Automated Documentation of Data Quality

One functionality that can significantly further ease the consumption of published data is providing automated documentation of data quality. Linked Data provides capabilities that allow data publishers to specify machine-readable information about the quality of their data. Providing the data quality measurements is stated as one of the most important recommendations for publishing data on the Web that are described in [28] and can be expressed in terms of an existing Data Quality vocabulary[2] .

With the help of the aforementioned vocabulary the following information can be provided in Linked Data format:

1. *Statistics* computed on dataset can indicate such observations as:
    - A number of distinct external resources linked to the dataset.
    - A number of distinct external resources used in the dataset.
    - A number of distinct literals.
    - A number of languages used in the dataset.
2. *Availability* – how data can be accessed now and over the time.
3. *Processability* – measures the level on which data is machine-readable.
4. *Accuracy* – measures how correct published data represents a corresponding real-life entity.
5. *Consistency* – measures contradictions within a dataset.
6. *Relevance* – describes whether published data includes an appropriate amount of data.
7. *Completeness* – describes whether dataset contains all data items representing a corresponding real-life entity.
8. *Conformance* – describes whether dataset follows accepted standards.
9. *Timeliness* – measures data actuality [1].

[2] https://www.w3.org/TR/vocab-dqv/

By enriching published data with a machine-readable data quality assessment, data publishers significantly improve the process of dataset consumption. The automated generation of dataset quality information in terms of the aforementioned vocabulary is an important feature for data publishers, who clean and transform their data to RDF with the help of Grafterizer framework.

6.2.2 Automated Anomaly Detection

Another feature that could further simplify data cleaning process with Grafterizer is automated anomaly detection.

Numerous algorithmic techniques, which are able to aid anomaly detection in data have been developed by a various database and machine learning communities. These include methods for automated type inference [15], detecting erroneous and extreme values [21, 22] and schema matching [20, 34]. Some of the aforementioned techniques are applicable with tabular data and can serve as parts of functioning applications, e.g., type suggestion in *messytables*[3] and schema matching in *CSV Lint*[4].

Future work should provide an interactive service for automated anomaly detection. This feature should enable users to perform a basic spell checks, identify similar records and define and validate a schema on the dataset (data types for columns, acceptability of null values, uniqueness and other relevant information).

6.2.3 Intelligent Vocabulary Suggestion

One more feature that Grafterizer framework may benefit from is intelligent vocabulary suggestion. Implementing the capabilities of automated RDF vocabulary suggestion may significantly simplify the process of mapping tabular data to RDF and thus reduce the complexity and time necessary for tabular-to-RDF data conversion.

Capabilities to enable this functionality are provided by the ABSTAT linked data summaries[5] Web service and are available for integration.

[3] https://messytables.readthedocs.org/en/latest/ last accessed May 18, 2016
[4] http://csvlint.io/about last accessed May 18, 2016
[5] http://abstat.disco.unimib.it:8880/

The future service can be included as part of the RDF mapping process in Grafterizer. It should provide the end-users with suggestions on the classes of RDF statement subjects and objects, as well as suggestions on properties based on dataset column names and context information.

6.3 Thesis Summary

The work performed in this thesis is described as follows:

1. Chapter 1 argues for the need for the new research in data publication and consumption process.

2. Prior to the practical contribution itself, the related work was discussed in Chapter 2. This includes studying a literature on data quality and defining data quality through the *set of data quality dimensions*. Next, the data quality issues were introduced as the *categorized set of data anomalies*, which should be resolved in order to increase the level of data quality. After data quality problems were examined, the technologies that can further enrich the data and increase its usability were studied.

3. The next stage of work is performing a problem analysis, that concludes in identifying the *set of requirements* and a *general workflow* for the artifact to be developed. This was done in Chapter 3.

4. Chapter 4 discusses a practical contribution through presenting the *Grafterizer*, an interactive framework for data cleaning and transformation. The system's novelty is in unifying two parts of Linked Data generation, data cleaning and converting to RDF, and in the interactive user interface, simplifying data transformation. The functionalities of the framework cover resolving all groups of data anomalies identified during the problem analysis.

5. The evaluation of the developed software was reported in Chapter 5, where the referenced software was evaluated against existing relevant tools and

systems with similar functionalities. The evaluation also includes comparing Grafterizer approach to Linked Data generation to the alternative approach (R2RML). An essential part of the evaluation is also an execution of a set of the use case scenarios with the help of Grafterizer framework.

6. Finally, the overall thesis evaluation in accordance with the stated requirements was performed and directions for future work were identified in Chapter 6.

To conclude, the work performed in this thesis allowed to make essential extensions to Grafterizer. At present, Grafterizer is a complete framework, capable of performing complex data cleaning and transformation operations, having a convenient user interface with interactive preview and supporting the reuse of the defined transformations.

Appendices

Appendix A

List of Acronyms

Table A.1: List of acronyms

Acronym	**Full name**
API	Application programming interface
CSV	Comma separated values (format)
DSL	Domain specific language
ETL	Extract transform load
GUI	Graphical user interface
JSON	JavaScript Object Notation (format)
JVM	Java virtual machine
KDD	Knowledge discovery in databases
LOD	Linked Open Data
OWL	Web Ontology Language
RDB	Relational database
RDF	Resource description framework
RDFS	Resource description framework
REST	Representational state transfer
R2RML	RDB to RDF Mapping Language
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSD	XML Schema Definition

Appendix B

Specification of the Developed Routines in Clojure

add-row

```
(add-row dataset [& values])
(add-row dataset position [& values])
```

Inserts new row into a dataset. Two options are available:

1. Takes a dataset and vector containing field values and appends new row to the end of a dataset, e.g.

   Given original dataset

:col1	:col2	:col3
1	2	3
4	5	6
7	8	9

   function returns the following result:

```
(add-row dataset [10 11 12]) ; =>
```

:col1	:col2	:col3
1	2	3
4	5	6
7	8	9
7	11	12

2. Takes a dataset, row index and vector containing field values and inserts new row at the specified position. If position index is negative or greater then total number of rows in a dataset, the new row will be apended to a dataset.

   Example Given original dataset

:col1	:col2	:col3
1	2	3
4	5	6
7	8	9

   function returns the following result:

```
(add-row dataset 1 [10 11 12]) ; =>
```

:col1	:col2	:col3
1	2	3
10	11	12
4	5	6
7	8	9

```
(add-row dataset -2 [10 11 12])  ; =>
```

:col1	:col2	:col3
10	11	12
1	2	3
4	5	6
7	8	9

For both options if number of parameters denoting field values is less than current number of columns in a dataset, lacking values for columns will remain empty. If number of parameters denoting field values is greater than number of columns in a dataset, rest of the values will be discarded.

cast

```
(cast dataset variable value f)
```

Cast function is reverse to melt. Given a dataset, variable-column, value-column and name of aggregation function, it forms column headers by identifying distinct variables and populates these columns by taking values and performing specified aggregation on them. Other columns are treated as pivot keys.

For most common aggregations there exists a set of pre-defined functions:

- MIN
- MAX
- SUM
- AVG
- COUNT
- COUNT-DISTINCT

Example. Given original dataset:

:company-name	:position	:total-employed
Cisco	Jr.Software developer	22
Cisco	Sr.Software developer	10
Cisco	Intern	2
Oracle corporation	Assist.manager	2
Oracle corporation	Sr.Software developer	38
IBM	Assist.manager	2
IBM	Jr.Software developer	8
Cisco	Assist.manager	3
IBM	Sr.Software developer	5
IBM	Intern	4

function returns the following result:

```
(cast :position :total-employed "SUM") ; =>
```

:company-name	:Jr.Software developer	:Sr.Software developer	:Intern	:Assist.manager
Cisco	22	10	2	3
IBM	8	5	4	2
Oracle corporation		38		2

group-rows

```
(group-rows dataset colnames colnames-functions)
```

Given a dataset, vector of column names and set of maps of form colname function-or-separator-name and creates a new dataset containg rows grouped by colnames from vector and the result of applying functions to correspondent column values. If function name is not recognized as a common aggregation function, argument will be used as a separator for merged values Each function in a map should take sequence of values as a parameter and return a single value.

For most common aggregations there exists a set of pre-defined functions:

- MIN
- MAX
- SUM
- AVG
- COUNT
- COUNT-DISTINCT

Example 1. Given original dataset:

:firstname	:lastname	:order_num	:total_items	:total_cost
Alice	Smith	11111	5	150
Bob	Johnson	857	7	70
Alice	Smith	11112	30	340
Alice	Williams	505	1	170
Bob	Johnson	858	3	370
Mary	Williams	1543	1	15

function returns the following result:

```
(group-rows dataset [:firstname :lastname]
#_=> #{ {:total_items "SUM"}      ; total number of items person ordered
#_=>    {:total_cost "AVG"}       ; average total cost per one order
#_=>    {:order_num "COUNT"}      ; number of orders person made
#_=>    {:total_cost "MAX"}})     ; maximum total cost per one order =>
```

:firstname	:lastname	:order_num_COUNT	:total_cost_AVG	:total_items_SUM	:total_cost_MAX
Alice	Smith	2	245	35	340
Bob	Johnson	2	220	10	370
Alice	Williams	1	170	1	170
Mary	Williams	1	15	1	15

Example 2. Given original dataset:

:name	:phone-number
Alice	123-45-67
Bob	777-88-99
Alice	111-11-11

function returns the following result:

```
(group-rows dataset [:name] #{{:phone-nuber ",␣"}}) ; =>
```

:name	:phone-number
Alice	123-45-67, 111-11-11
Bob	777-88-99

join-dataset

```
(join-dataset dataset filename concat-type)
(join-dataset dataset filename fkey id value)
```

Joins two datasets together. Two options are available:

1. Takes a dataset, filename and type of concatenation (either :v to concatenate datasets vertically – append data from file to the right side of given dataset or :h to concatenate datasets horizontally – append data from file to the bottom of given dataset).

   Throws an error if number of columns/rows is not appropriate

   Example 1. Given original dataset:

:name	:age	:gender
Alice	18	female
Bob	30	male

   and file *left-part.csv* with content:

   ```
   email,               country
   alice@example.com,   Norway
   bob@example.com,     Norway
   ```

   function returns the following result:

```
(join-dataset "left-part.csv" :v) ; =>
```

:name	:age	:gender	:email	:country
Alice	18	female	alice@example.com,	Norway
Bob	30	male	bob@example.com,	Norway

Example 2. Given the same original dataset and file 'other-persons.csv' with content:

```
name,  age,  gender
John,  38,   male
Mary,  27,   female
```

function returns the following result:

```
(join-dataset "other-persons.csv" :h) ; =>
```

:name	:age	:gender
Alice	18	female
Bob	30	male
John	38	male
Mary	27	female

2. Takes a dataset, filename, column that acts as foreign key in original dataset and columns for id and value in file.

   Builds a lookup table from file and maps values in original dataset appropriately.

   Example. Given original dataset:

:name	:age	:position
Alice	18	25
Bob	30	7

and file *position-codes.csv* with content:

```
code,        description
   1,            manager
   7,           engineer
   8,    senior engineer
  25,         accountant
```

function returns the following result:

```
(join-dataset "position-codes.csv"
#_=>           :position "code" "description") ; =>
```

:name	:age	:position
Alice	18	accountant
Bob	30	engineer

merge-columns

```
(merge-columns dataset columns separator)
(merge-columns dataset columns separator newname)
```

Merges several columns in one using specified separator between columns.
Two options are available:

1. Takes a dataset, vector of columns and separator and merges columns together. Column containing the result of the merge gets the same name as the first column in the list of arguments.

   Example. Given original dataset:

:name	:city	:country	e-mail
Alice	Oslo	Norway	alice@example.com
Bob	Trondheim	Norway	bob@example.com

   function returns the following result:

```
(merge-columns [:city :country] ", ") ; =>
```

:name	:city	e-mail
Alice	Oslo, Norway	alice@example.com
Bob	Trondheim, Norway	bob@example.com

2. Takes a dataset, vector of columns, separator and new column name and merges columns together.

Example. Given original dataset:

:name	:city	:country	e-mail
Alice	Oslo	Norway	alice@example.com
Bob	Trondheim	Norway	bob@example.com

function returns the following result:

```
(merge-columns [:city :country] ",␣" :place) ; =>
```

:name	:place	e-mail
Alice	Oslo, Norway	alice@example.com
Bob	Trondheim, Norway	bob@example.com

remove-columns

```
(remove-columns dataset cols)
(remove-columns dataset indexFrom indexTo)
```

Removes columns from a dataset.
Two options are available:

1. Takes a dataset and vector of column names and creates a new dataset containing all columns except of those that were specified

   Example. Given original dataset:

:col1	:col2	:col3	:col4
1	2	3	4
5	6	7	8
9	10	11	12

   function returns the following result:

   ```
   (remove-columns [:col1 :col4]) ; =>
   ```

:col2	:col3
2	3
6	7
10	11

2. Takes a dataset and two indices and creates a new dataset containing all columns execept of columns having indices within the specified interval (including both points)

Example. Given original dataset:

a	b	c	d	e	f
0	0	0	0	0	0
1	1	1	1	1	1
2	2	2	2	2	2

function returns the following result:

```
(remove-columns 2 4) ; Remove columns having indices within the interval [2, 4] =>
```

a	b	f
0	0	0
1	1	1
2	2	2

remove-duplicates

```
(remove-duplicates dataset)
(remove-duplicates dataset colnames)
```

Removes duplicates from a dataset.
Two options are available:

1. Given a dataset sorts it and looks for rows having the same values across all columns and leaves only one instance from each set of such rows, other rows(duplicates) will be removed from a dataset.

   Example. Given original dataset:

:name	:age	:gender
Alice	18	female
Bob	30	male
Alice	28	female
Alice	18	female
Bob	32	male

function returns the following result:

```
(remove-duplicates dataset) ; =>
```

:name	:age	:gender
Alice	18	female
Alice	28	female
Bob	30	male
Bob	32	male

2. Given a dataset and a column(sequence of columns) looks for rows having the same values in the specified field(s) and leaves only the first encountered row in this sequence. Dataset should be sorted in desired order before function is called

   Example. Given original dataset:

:name	:age	:gender
Alice	18	female
Bob	30	male
Alice	28	female
Alice	18	female
Bob	32	male

   function returns the following result:

```
(->  (sort-dataset dataset  [:name :age] :alpha :desc)
#_=> (remove-duplicates [:name :gender])) ; Dataset is first sorted in
#_=> ; a such way, that the records about the same person are given in descending
#_=> ; order by age =>
```

:name	:age	:gender
Alice	28	female
Bob	32	male

shift-row

```
(shift-column dataset column)
(shift-column dataset column position-to)
```

Changes row's position inside a dataset.
Two options are available:

1. Takes a dataset and row index and moves this row to the end of a dataset, data rows with indices greater than specified index will be moved one position up.

   Example. Given original dataset:

:col1	:col2	:col3
1	2	3
4	5	6
7	8	9
10	11	12

   function returns the following result:

```
(shift-row dataset 1) ; =>
```

:col1	:col2	:col3
1	2	3
7	8	9
10	11	12
4	5	6

2. Takes a dataset and two row indices and moves row from index #1 to index #2. Other rows will be shifted appropriately.

   Example. Given original dataset:

:col1	:col2	:col3
1	2	3
4	5	6
7	8	9
10	11	12

   function returns the following result:

```
(shift-row dataset 1 3) ; =>
```

:col1	:col2	:col3
1	2	3
7	8	9
10	11	12
4	5	6

shift-column

```
(shift-column dataset column)
(shift-column dataset column position-to)
```

Changes column's position inside a dataset.

Two options are available:

1. Takes a dataset and column name/index and moves this column to the last position, data columns with indices greater than specified index will be moved one position left.

   Example. Given original dataset:

:a	:b	:c	:d
1	2	3	a
4	5	6	b
7	8	9	c

   function returns the following result:

```
(shift-column :b) ; =>
```

   which is equivalent to

```
(shift-column 1) ; =>
```

:a	:c	:d	:b
1	3	a	2
4	6	b	5
7	9	c	8

2. Takes a dataset, column name/index and index where this column should be moved, moves given column to the specified index. Other columns will be shifted appropriately.

   Example. Given original dataset:

:a	:b	:c	:d
1	2	3	a
4	5	6	b
7	8	9	c

   function returns the following result:

```
(shift-column :c 0) ; =>
```

which is equivalent to

```
(shift-column 2 0) ; =>
```

:c	:a	:b	:d
3	1	2	a
6	4	5	b
9	7	8	c

sort-dataset

```
(sort-dataset dataset colnames-sorttypes)
```

Sorts dataset by given column names in given order. Column names and types of sorting are given in a vector. Sorting priority is defined by order of column name – sorting type pair. Sorting by multiple columns works as follows: if several rows have equal columns (first in the vector of given columns) according to the given comparator type, these rows will be sorted by second column and second comparator, if both first and second are equal, sorting will be performed by the third column and third comparator etc.

Type of comparator used for sorting is defined as one of following:

- **:ascalpha**, **:descalpha** for alphabetical sorting (in ascending and descending order correspondingly);
- **:ascnum**, **:descnum** for numerical sorting;
- **:asclen**, **:desclen** for sorting by field length;
- **:ascdate**, **:descdate** for sorting dates.

Example 1. Given original dataset:

:a	:b	:c	:d
2	string	1	01.01.2015
111	string	3	03.11.2015
44	longer string	9	03.03.2013
3	the longest string	6	25.12.2015

calling function with different parameters results in following datasets:

```
(sort-dataset dataset [{:a :ascalpha}]) ; sort by column :a in ascending alphabetical order
    =>
```

:a	:b	:c	:d
111	string	3	03.11.2015
2	string	1	01.01.2015
3	the longest string	6	25.12.2015
44	longer string	9	03.03.2013

Example 2. Given the same dataset, sort by column :a in descending alphabetical order

```
(sort-dataset dataset [{:a :descalpha}]) ; =>
```

:a	:b	:c	:d
44	longer string	9	03.03.2013
3	the longest string	6	25.12.2015
2	string	1	01.01.2015
111	string	3	03.11.2015

Example 3. Given the same dataset, sort by column :a in ascending numerical order

```
(sort-dataset dataset [{:a :ascnum}]) ; =>
```

:a	:b	:c	:d
2	string	1	01.01.2015
3	the longest string	6	25.12.2015
44	longer string	9	03.03.2013
111	string	3	03.11.2015

Example 4. Given the same dataset, sort by column :b in ascending order by field length

```
(sort-dataset dataset [{:b :asclen}]) ; =>
```

:a	:b	:c	:d
2	string	1	01.01.2015
111	string	3	03.11.2015
44	longer string	9	03.03.2013
3	the longest string	6	25.12.2015

Example 5. Given the same dataset, sort by column :d in ascending order by date

```
(sort-dataset dataset [{:b :asclen}]) ; =>
```

:a	:b	:c	:d
44	longer string	9	03.03.2013
2	string	1	01.01.2015
111	string	3	03.11.2015
3	the longest string	6	25.12.2015

Example 6. Given the same dataset, sort by column :b in ascending order by field length, for equal values arrange by column :a in ascending order by length

```
(sort-dataset dataset [{:b asclen} {:a :asclen}])  ; =>
```

:a	:b	:c	:d
2	string	1	01.01.2015
111	string	3	03.11.2015
44	longer string	9	03.03.2013
3	the longest string	6	25.12.2015

Example 7. Given the same dataset, sort by column :a in ascending order by field length, for equal values arrange by column :b in ascending order by length

```
(sort-dataset dataset [{:a asclen} {:b :asclen}])  ; =>
```

:a	:b	:c	:d
2	string	1	01.01.2015
3	the longest string	6	25.12.2015
44	longer string	9	03.03.2013
111	string	3	03.11.2015

split-column

```
(split-column dataset colname separator)
```

Given a dataset, column name and separator splits specified column into multiple by separator. New columns get names of a form [original-column-name]_splitted_0, [original-column-name]_splitted_1, ...

Example. Given original dataset:

:name	:address	:email
Alice	New York, Harrison Street, 507	alice@example.com
Bob	Richmond, Main Street, 17	bob@example.com
Mary	NY, Harrison Street, 29, H0512	mary@example.com

function returns the following result:

```
(split-column :address #",␣") ; =>
```

:name	:address_splitted_0	:address_splitted_1	:address_splitted_2	:address_splitted_3	:email
Alice	New York	Harrison Street	507		alice@example.com
Bob	Richmond	Main Street	17		bob@example.com
Mary	NY	Harrison Street	29	H0512	mary@example.com

convert-literal

```
(convert-literal x dtype & {:keys [on-empty on-error lang-tag], :or {on-error false,
    on-empty 0, lang-tag nil}})
```

Converts given value to the specified datatype.
Supported datatypes (with corresponding xsd types) :

Argument	Datatype	Value space
"byte"	xsd:byte	-128…+127 (8 bit)
"short"	xsd:short	-32768…+32767 (16 bit)
"double"	xsd:double	64-bit floating point numbers
"decimal"	xsd:decimal	Arbitrary-precision decimal numbers
"integer"	xsd:int	-2147483648…+2147483647 (32 bit)
"long"	xsd:long	-9223372036854775808…+9223372036854775807 (64 bit)
"float"	xsd:float	2-bit floating point numbers
"boolean"	xsd:boolean	true, false
"date"	xsd:dateTime	Date and time with timezone
"string"	xsd:string	Character strings

Optional keys:

:on-error – specifies value, that should be used to replace non-valid arguments. By default function replaces all non-valid values with 0 for all numeric types, "false" for data type boolean and "31.12.2099" for dates

:on-empty – specifies value, that should be used to replace empty(nil) arguments. By default function replaces all empty values with 0 for all numeric types, "false" for data type boolean and "31.12.2099" for dates

:lang-tag – specifies a language tag used with string literals

By default for conversions to data type boolean following values are converted to false:

- false (as boolean);
- nil;
- "" (empty string);
- "false" (as string);
- "0" (as string);
- 0 (as integer).

References

[1] Riccardo Albertoni, Antoine Isaac, and Christophe Guéret. *Data on the Web Best Practices: Data Quality Vocabulary*. Tech. rep. http://www.w3.org/TR/2015/WD-vocab-dqv-20151217. W3C, 2015.

[2] G. Antoniou, F. van Harmelen, and R. Hoekstra. *A Semantic Web Primer*. Cooperative information systems. MIT Press, 2012. ISBN: 9780262018289. URL: https://books.google.no/books?id=tYb6AQAAQBAJ.

[3] Marcelo Arenas et al. *A Direct Mapping of Relational Data to RDF*. Tech. rep. https://www.w3.org/TR/rdb-direct-mapping/. W3C, 2012.

[4] C. Batini and M. Scannapieco. *Data and Information Quality: Dimensions, Principles and Techniques*. Data-Centric Systems and Applications. Springer International Publishing, 2016. ISBN: 9783319241067. URL: https://books.google.no/books?id=kJ_WCwAAQBAJ.

[5] Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 3540331727.

[6] Carlo Batini et al. "From Data Quality to Big Data Quality." In: *J. Database Manage.* 26.1 (Jan. 2015), pp. 60–82. ISSN: 1063-8016. DOI: 10.4018/JDM.2015010103. URL: http://dx.doi.org/10.4018/JDM.2015010103.

[7] I. Berlocher, S. Kim, and T. Lee. "Use case implementation." In: *v1.DaPaaS Deliverable D5.2*. 2014. URL: http://bit.ly/1lb5uzJ.

[8] Tim Berners-Lee. "Linked Data." In: 2006. URL: www.w3.org/DesignIssues/LinkedData.html.

[9] Tim Berners-Lee, James Hendler, Ora Lassila, et al. “The semantic web.” In: *Scientific american* 284.5 (2001), pp. 28–37.

[10] Christian Bizer, Tom Heath, and Tim Berners-Lee. “Linked data-the story so far.” In: *Semantic Services, Interoperability and Web Applications: Emerging Concepts* (2009), pp. 205–227.

[11] Souripriya Das, Seema Sundara, and Richard Cyganiak. *R2RML: RDB to RDF Mapping Language*. Tech. rep. https://www.w3.org/TR/r2rml/. W3C, 2012.

[12] Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning*. 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 2003. ISBN: 0471268518.

[13] Shaker H Ali El-Sappagh, Abdeltawab M Ahmed Hendawi, and Ali Hamed El Bastawissy. “A proposed model for data warehouse ETL processes.” In: *Journal of King Saud University-Computer and Information Sciences* 23.2 (2011), pp. 91–104.

[14] Ramez Elmasri and Shamkant Navathe. *Fundamentals of Database Systems*. 6th. USA: Addison-Wesley Publishing Company, 2010. ISBN: 0136086209, 9780136086208.

[15] Kathleen Fisher and David Walker. “The PADS Project: An Overview.” In: *Proceedings of the 14th International Conference on Database Theory*. ICDT ’11. Uppsala, Sweden: ACM, 2011, pp. 11–17. ISBN: 978-1-4503-0529-7. DOI: 10.1145/1938551.1938556. URL: http://doi.acm.org/10.1145/1938551.1938556.

[16] John F. Gantz et al. “The Diverse and Exploding Digital Universe: An Updated Forecast of Worldwide Information Growth Through 2011.” In: *IDC*. 2008.

[17] Christopher Groskopf and contributors. *csvkit*. 2015. URL: https://csvkit.readthedocs.org/.

[18] The W3C SPARQL Working Group. *SPARQL 1.1 Overview*. Tech. rep. https://www.w3.org/TR/sparql11-overview/. W3C, 2013.

[19] Thomas R Gruber. “A translation approach to portable ontology specifications.” In: *Knowledge acquisition* 5.2 (1993), pp. 199–220.

[20] Laura M. Haas et al. “Clio Grows Up: From Research Prototype to Industrial Tool.” In: *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’05. Baltimore, Maryland: ACM, 2005, pp. 805–810. ISBN: 1-59593-060-4. DOI: 10.1145/1066157.1066252. URL: http://doi.acm.org/10.1145/1066157.1066252.

[21] Joseph M Hellerstein. “Quantitative data cleaning for large databases.” In: *United Nations Economic Commission for Europe (UNECE)* (2008).

[22] Victoria Hodge and Jim Austin. “A Survey of Outlier Detection Methodologies.” In: *Artif. Intell. Rev.* 22.2 (Oct. 2004), pp. 85–126. ISSN: 0269-2821. DOI: 10.1023/B:AIRE.0000045502.10941.a9. URL: http://dx.doi.org/10.1023/B:AIRE.0000045502.10941.a9.

[23] Sean Kandel. “INTERACTIVE SYSTEMS FOR DATA TRANSFORMATION AND ASSESSMENT.” PhD thesis. STANFORD UNIVERSITY, June 2013.

[24] Sean Kandel et al. “Research Directions in Data Wrangling: Visualizations and Transformations for Usable and Credible Data.” In: *Information Visualization Journal* 10 (4 2011), pp. 271–288. URL: http://vis.stanford.edu/papers/data-wrangling.

[25] Sean Kandel et al. “Research Directions in Data Wrangling: Visuatizations and Transformations for Usable and Credible Data.” In: *Information Visualization* 10.4 (Oct. 2011), pp. 271–288. ISSN: 1473-8716. DOI: 10.1177/1473871611415994. URL: http://dx.doi.org/10.1177/1473871611415994.

[26] Sean Kandel et al. “Profiler: Integrated Statistical Analysis and Visualization for Data Quality Assessment.” In: *Advanced Visual Interfaces*. 2012. URL: http://vis.stanford.edu/papers/profiler.

[27] C.R. Kothari. *Research Methodology: Methods and Techniques*. New Age International (P) Limited, 2004.

[28] Bernadette Farias Lóscio, Caroline Burle, and Newton Calegari. *Data on the Web Best Practices*. Tech. rep. http://www.w3.org/TR/2016/WD-dwbp-20160112. W3C, 2016.

[29] Jonathan I Maletic and Andrian Marcus. “Data Cleansing: Beyond Integrity Analysis.” In: *IQ*. Citeseer. 2000, pp. 200–209.

[30] A. Manning. *Databases for Small Business: Essentials of Database Management, Data Analysis, and Staff Training for Entrepreneurs and Professionals*. Apress, 2015. ISBN: 9781484202784. URL: https://books.google.no/books?id=CU3BrQEACAAJ.

[31] Heiko Müller and Johann-Christph Freytag. *Problems, methods, and challenges in comprehensive data cleansing*. Professoren des Inst. Für Informatik, 2005.

[32] M. Tamer Ozsu. *Principles of Distributed Database Systems*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007. ISBN: 9780130412126.

[33] D. Pyle. *Data Preparation for Data Mining*. Data Preparation for Data Mining v. 1. Morgan Kaufmann Publishers, 1999. ISBN: 9781558605299. URL: https://books.google.no/books?id=hhdVr9F-JfAC.

[34] Erhard Rahm and Philip A. Bernstein. “A Survey of Approaches to Automatic Schema Matching.” In: *The VLDB Journal* 10.4 (Dec. 2001), pp. 334–350. ISSN: 1066-8888. DOI: 10.1007/s007780100057. URL: http://dx.doi.org/10.1007/s007780100057.

[35] Erhard Rahm and Hong Hai Do. “Data Cleaning: Problems and Current Approaches.” In: *IEEE Data Engineering Bulletin* 23 (2000), p. 2000.

[36] Vijayshankar Raman and Joseph M. Hellerstein. “Potter’s Wheel: An Interactive Data Cleaning System.” In: *Proceedings of the 27th International Conference on Very Large Data Bases*. VLDB ’01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 381–390. ISBN: 1-55860-804-4. URL: http://dl.acm.org/citation.cfm?id=645927.672045.

[37] Carmen M. Reinhart and Kenneth S. Rogoff. *Growth in a Time of Debt*. Working Paper 15639. National Bureau of Economic Research, 2010. DOI: 10.3386/w15639. URL: http://www.nber.org/papers/w15639.

[38] B. Roberts. “Software tools integrated into platform.” In: *DaPaaS Deliverable D4.2*. 2015. URL: http://bit.ly/1PcM8v7.

[39] B. Roberts and R. Moynihan. “Documented methodology and guidelines.” In: *DaPaaS Deliverable D4.1*. 2014. URL: http://bit.ly/1NMU8lJ.

[40] A. Simov et al. “Open Data PaaS prototype.” In: *v.2. DaPaaS Deliverable D2.3*. 2015. URL: http://bit.ly/1Jj86s4.

[41] Martin G. Skjæveland, Espen H. Lian, and Ian Horrocks. “Publishing the Norwegian Petroleum Directorate’s FactPages as Semantic Web Data.” In: *The Semantic Web – ISWC 2013*. Ed. by Harith Alani et al. Vol. 8219. LNCS. Springer Berlin Heidelberg, 2013, pp. 162–177. URL: SkLiHo-ISWC2013.pdf.

[42] Ida Solheim and Ketil Stølen. *Technology Research Explained*. Tech. rep. SINTEF A313. SINTEF, 2007.

[43] I. Sommerville. *Software Engineering*. International Computer Science Series. Pearson, 2011. ISBN: 9780137053469. URL: https://books.google.no/books?id=l0egcQAACAAJ.

[44] Dimitrios-Emmanuel Spanos, Periklis Stavrou, and Nikolas Mitrou. “Bringing Relational Databases into the Semantic Web: A Survey.” In: *Semant. web* 3.2 (Apr. 2012), pp. 169–209. ISSN: 1570-0844. DOI: 10.3233/SW-2011-0055. URL: http://dx.doi.org/10.3233/SW-2011-0055.

[45] Dina Sukhobok et al. “Tabular Data Cleaning and Linked Data Generation with Grafterizer.” In: *to appear in post-conference proceedings of ESWC*. 2016. URL: http://2016.eswc-conferences.org/program/posters-demos.

[46] Hadley Wickham. “Tidy data.” In: *The Journal of Statistical Software* 59 (10 2014). URL: http://www.jstatsoft.org/v59/i10/.