

# Efficient Extraction and Query Benchmarking of Wikipedia Data

Der Fakultät für Mathematik und Informatik  
der Universität Leipzig  
eingereichte

## DISSERTATION

zur Erlangung des akademischen Grades

**DOKTOR-INGENIEUR**  
(Dr. Ing.)

im Fachgebiet Informatik

vorgelegt

von **M.Sc. Mohamed Mabrouk Mawed Morsey**

geboren am 15. November 1980 in Kairo, Ägypten

Leipzig, den 13.4.2014



# Acknowledgement

First of all I would like to thank my supervisors, Dr. Jens Lehmann, Prof. Sören Auer, and Prof. Klaus-Peter Fähnrich, without whom I could not start my Ph.D. at the Leipzig University.

Special thanks to my direct supervisor Dr. Jens Lehmann, with whom I have started work on my Ph.D. proposal submitted to Deutscher Akademischer Austauschdienst (DAAD), in order to pursue my Ph.D. at the Agile Knowledge Engineering and Semantic Web (AKSW) group. He has continuously supported me through my Ph.D. work, giving advices and recommendations for further research steps. His comments and notes were very helpful for me particularly during writing the papers we have published together. I would like to thank him also for proofreading this thesis and for his helpful feedback, which led to improving the quality of that thesis.

Special thanks also to thank Prof. Sören Auer, whom I have first contacted asking for a vacancy to conduct my Ph.D. research at his research group. As the head of our group, Prof. Sören Auer has established a weekly meeting, called "Writing Group", for discussing how to write robust scientific papers. His notes and comments, during these meetings, were very useful for directing me to the correct way to write a good scientific paper.

I would like also to thank Prof. Klaus-Peter Fähnrich, for the regular follow-up meetings he has managed in order to evaluate the performance of all Ph.D. students. During these meetings, he has proposed several directions, for me and for other Ph.D. students as well, on how to deepen and extend our research points.

I would like to thank all of my colleagues in the Machine Learning and Ontology Engineering (MOLE) group, for providing me with their useful comments and guidelines, especially during the initial phase of my Ph.D..

I would like to dedicate this work to the souls of my parents, without whom I could not do anything in my life. With their help and support, I could take my first steps in my scientific career.

Special thanks goes to all my family members, my son Momen, my wife Hebaalla, and my dearest sisters Reham and Nermeen.

# Bibliographic Data

**Title:** Efficient Extraction and Query Benchmarking of Wikipedia Data

**Author:** Mohamed Mabrouk Mawed Morsey

**Institution:** Universität Leipzig, Fakultät für Mathematik und Informatik

**Statistical Information:** 128 pages, 29 Figures, 19 tables, 2 appendices, 98 literature references

## Abstract

Knowledge bases are playing an increasingly important role for integrating information between systems and over the Web. Today, most knowledge bases cover only specific domains, they are created by relatively small groups of knowledge engineers, and it is very cost intensive to keep them up-to-date as domains change. In parallel, Wikipedia has grown into one of the central knowledge sources of mankind and is maintained by thousands of contributors. The DBpedia (<http://dbpedia.org>) project makes use of this large collaboratively edited knowledge source by extracting structured content from it, interlinking it with other knowledge bases, and making the result publicly available. DBpedia had and has a great effect on the Web of Data and became a crystallization point for it. Furthermore, many companies and researchers use DBpedia and its public services to improve their applications and research approaches.

However, the DBpedia release process is heavy-weight and the releases are sometimes based on several months old data. Hence, a strategy to keep DBpedia always in synchronization with Wikipedia is highly required. In this thesis we propose the *DBpedia Live* framework, which reads a continuous stream of updated Wikipedia articles, and processes it. DBpedia Live processes that stream on-the-fly to obtain RDF data and updates the DBpedia knowledge base with the newly extracted data. DBpedia Live also publishes the newly added/deleted facts in files, in order to enable synchronization between our DBpedia endpoint and other DBpedia mirrors. Moreover, the new DBpedia Live framework incorporates several significant features, e.g. abstract extraction, ontology changes, and changesets publication.

Basically, knowledge bases, including DBpedia, are stored in triplestores in order to facilitate accessing and querying their respective data. Furthermore, the triplestores constitute the backbone of increasingly many Data Web applications. It is thus evident that the performance of those stores is mission critical for individual projects as well as for data integration on the Data Web in general. Consequently,

---

it is of central importance during the implementation of any of these applications to have a clear picture of the weaknesses and strengths of current triplestore implementations. We introduce a generic SPARQL benchmark creation procedure, which we apply to the DBpedia knowledge base. Previous approaches often compared relational and triplestores and, thus, settled on measuring performance against a relational database which had been converted to RDF by using SQL-like queries. In contrast to those approaches, our benchmark is based on queries that were actually issued by humans and applications against existing RDF data not resembling a relational schema. Our generic procedure for benchmark creation is based on query-log mining, clustering and SPARQL feature analysis. We argue that a pure SPARQL benchmark is more useful to compare existing triplestores and provide results for the popular triplestore implementations Virtuoso, Sesame, Apache Jena-TDB, and BigOWLIM. The subsequent comparison of our results with other benchmark results indicates that the performance of triplestores is by far less homogeneous than suggested by previous benchmarks.

Further, one of the crucial tasks when creating and maintaining knowledge bases is validating their facts and maintaining the quality of their inherent data. This task include several subtasks, and in thesis we address two of those major subtasks, specifically fact validation and provenance, and data quality. The subtask fact validation and provenance aim at providing sources for these facts in order to ensure correctness and traceability of the provided knowledge. This subtask is often addressed by human curators in a three-step process: issuing appropriate keyword queries for the statement to check using standard search engines, retrieving potentially relevant documents and screening those documents for relevant content. The drawbacks of this process are manifold. Most importantly, it is very time-consuming as the experts have to carry out several search processes and must often read several documents. We present DeFacto (Deep Fact Validation), which is an algorithm for validating facts by finding trustworthy sources for it on the Web. DeFacto aims to provide an effective way of validating facts by supplying the user with relevant excerpts of webpages as well as useful additional information including a score for the confidence DeFacto has in the correctness of the input fact. On the other hand the subtask of data quality maintenance aims at evaluating and continuously improving the quality of data of the knowledge bases. We present a methodology for assessing the quality of knowledge bases' data, which comprises of a manual and a semi-automatic process. The first phase includes the detection of common quality problems and their representation in a quality problem taxonomy. In the manual process, the second phase comprises of the evaluation of a large number of individual resources, according to the quality problem taxonomy via crowdsourcing. This process is accompanied by a tool wherein a user assesses an individual resource and evaluates each fact for correctness. The semi-automatic process involves the generation and verification of schema axioms. We report the results obtained by applying this methodology to DBpedia.

# Contents

<b>1. Introduction</b>	<b>1</b>
1.1. Motivation . . . . .	1
1.2. Contributions . . . . .	3
1.3. Chapter Overview . . . . .	5
<b>2. Semantic Web Technologies</b>	<b>7</b>
2.1. Semantic Web Definition . . . . .	7
2.2. Resource Description Framework - RDF . . . . .	7
2.2.1. RDF Resource . . . . .	8
2.2.2. RDF Property . . . . .	8
2.2.3. RDF Statement . . . . .	9
2.2.4. RDF Serialization Formats . . . . .	10
2.2.4.1. N-Triples . . . . .	10
2.2.4.2. RDF/XML . . . . .	11
2.2.4.3. N3 . . . . .	11
2.2.4.4. Turtle . . . . .	12
2.2.5. Ontology . . . . .	12
2.2.6. Ontology Languages . . . . .	13
2.2.6.1. RDFS . . . . .	13
2.2.6.2. OWL . . . . .	14
2.2.7. SPARQL Query Language . . . . .	14
2.2.8. Triplestore . . . . .	16
<b>3. Overview on the DBpedia Project</b>	<b>17</b>
3.1. Introduction to DBpedia . . . . .	17
3.2. DBpedia Extraction Framework . . . . .	19
3.2.1. General Architecture . . . . .	19
3.2.2. Extractors . . . . .	20
3.2.3. Raw Infobox Extraction . . . . .	22
3.2.4. Mapping-Based Infobox Extraction . . . . .	22
3.2.5. URI Schemes . . . . .	25
3.2.6. Summary of Recent Developments . . . . .	26
3.3. DBpedia Ontology . . . . .	27
3.4. Interlinking . . . . .	29
3.4.1. Outgoing Links . . . . .	29
3.4.2. Incoming Links . . . . .	31

<b>4. DBpedia Live Extraction</b>	<b>33</b>
4.1. Live Extraction Framework . . . . .	33
4.1.1. General System Architecture . . . . .	33
4.1.2. Extraction Manager . . . . .	34
4.2. New Features . . . . .	36
4.2.1. Abstract Extraction . . . . .	37
4.2.2. Mapping-Affected Pages . . . . .	38
4.2.3. Unmodified Pages . . . . .	39
4.2.4. Changesets . . . . .	39
4.2.5. Synchronization Tool . . . . .	40
4.2.6. New Extractors . . . . .	40
4.2.7. Delta Calculation . . . . .	41
4.2.8. Important Pointers . . . . .	41
4.3. DBpedia Live Usage . . . . .	41
<b>5. Data Quality</b>	<b>43</b>
5.1. Crowdsourcing for Data Quality . . . . .	43
5.1.1. Assessment Methodology . . . . .	43
5.1.2. Quality Problem Taxonomy . . . . .	45
5.1.2.1. Accuracy . . . . .	46
5.1.2.2. Relevancy . . . . .	48
5.1.2.3. Representational-Consistency . . . . .	49
5.1.2.4. Interlinking . . . . .	50
5.1.3. A Crowdsourcing Quality Assessment Tool . . . . .	50
5.1.4. Evaluation of DBpedia Data Quality . . . . .	51
5.1.4.1. Evaluation Methodology . . . . .	51
5.1.4.1.1. Manual Methodology . . . . .	51
5.1.4.1.2. Semi-automatic Methodology . . . . .	51
5.1.4.2. Evaluation Results . . . . .	52
5.1.4.2.1. Manual Methodology . . . . .	52
5.1.4.2.2. Semi-automatic Methodology . . . . .	53
5.2. Fact Validation . . . . .	54
5.2.1. Trustworthiness Analysis of Webpages . . . . .	57
5.2.2. Features for Deep Fact Validation . . . . .	58
5.2.3. Evaluation . . . . .	60
5.2.3.1. Training DeFacto . . . . .	60
5.2.3.2. Experimental Setup . . . . .	61
5.2.3.3. Results and Discussion . . . . .	62
<b>6. DBpedia SPARQL Benchmark</b>	<b>64</b>
6.1. Overview on Benchmarking . . . . .	64
6.1.1. Objectives of Benchmarking . . . . .	65
6.2. Dataset Generation . . . . .	66
6.3. Query Analysis and Clustering . . . . .	68

6.4. SPARQL Feature Selection and Query Variability . . . . .	70
6.5. Experimental Setup . . . . .	71
6.5.1. Benchmark Phases . . . . .	72
6.6. Benchmarking Results . . . . .	73
6.6.1. DBPSB Version 1 . . . . .	76
6.6.2. DBPSB1 Results Discussion . . . . .	77
6.6.3. DBPSB Version 2 . . . . .	82
<b>7. Related Work</b>	<b>85</b>
7.1. Semantic Data Extraction from Wikipedia . . . . .	85
7.2. RDF Benchmarks . . . . .	86
7.2.1. Existing Benchmarks . . . . .	86
7.2.2. Comparison between DBPSB and The Other Benchmarks	88
7.3. Data Quality Assessment . . . . .	89
7.4. Fact Validation . . . . .	90
<b>8. Conclusions and Future Work</b>	<b>93</b>
8.1. Conclusions . . . . .	93
8.1.1. DBpedia Live Extraction . . . . .	93
8.1.2. DBPSB . . . . .	94
8.1.3. DeFacto . . . . .	94
8.2. Future Work . . . . .	94
8.2.1. DBpedia Live Extraction . . . . .	94
8.2.2. DBPSB . . . . .	95
8.2.3. DeFacto . . . . .	96
<b>A. DBpedia SPARQL Benchmark (DBPSB) Queries</b>	<b>97</b>
A.1. DBPSB Version 1 . . . . .	97
A.2. DBPSB Version 2 . . . . .	99
<b>B. Curriculum Vitae</b>	<b>101</b>
<b>List of Tables</b>	<b>106</b>
<b>List of Figures</b>	<b>108</b>
<b>Selbständigkeitserklärung</b>	<b>120</b>



# 1. Introduction

Tim Berners-Lee defines the Semantic Web as “The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation” [Berners-Lee et al., 2001]. As Semantic Web Technology evolves many open areas emerge, which attract more research focus. Examples of these areas are ontology management, semantic data provenance, and interlinking of knowledge bases. In this work we investigate three topics of Semantic Web, specifically *DBpedia Live* knowledge base development and maintenance, triplestores benchmarking, and semantic data provenance.

## 1.1. Motivation

Knowledge bases are playing an increasingly important role for integrating information between systems and over the Web. Today, most knowledge bases cover only specific domains, they are created by relatively small groups of knowledge engineers, and it is very cost intensive to keep them up-to-date as domains change. In parallel, Wikipedia has grown into one of the central knowledge sources of mankind and is maintained by thousands of contributors. Wikipedia is the largest online encyclopedia and currently the 6<sup>th</sup> most visited website according to [alexa.com](http://www.alexa.com). The *DBpedia* (<http://dbpedia.org>) project makes use of this large collaboratively edited knowledge source by extracting structured content from it, interlinking it with other knowledge bases, and making the result publicly available. Over the past few years the DBpedia knowledge base has turned into a crystallization point for the emerging Web of Data. Several tools have been built on top of it, e.g. DBpedia Mobile<sup>1</sup>, Query Builder<sup>2</sup>, Relation Finder [Lehmann et al., 2007], and Navigator<sup>3</sup>. It is used in a variety of applications, for instance Muddy Boots, Open Calais, Faviki, Zemanta, LODr, and TopBraid Composer (cf. [Lehmann et al., 2009]).

Despite this success, DBpedia faces several obstructions as follows:

- Producing a DBpedia release requires manual effort and – since dumps of the Wikipedia database are created on a monthly basis – DBpedia has never reflected the current state of Wikipedia.
- DBpedia uses a wiki for managing the ontology and the mappings<sup>4</sup>. Con-

---

<sup>1</sup><http://beckr.org/DBpediaMobile/>

<sup>2</sup><http://querybuilder.dbpedia.org>

<sup>3</sup><http://navigator.dbpedia.org>

<sup>4</sup><http://mappings.dbpedia.org>

tributors of that wiki continuously revise the ontology and/or the mappings, which leads that the ontology kept in DBpedia gets outdated, i.e. out of synchronization with that mappings wiki.

- As DBpedia extracts data from Wikipedia, many data quality related problems and/or issues may arise, e.g. the datatype of a phone number, could be misclassified as number instead of string.

In this dissertation, each obstruction of those is discussed in detail, and solution to each one is proposed and discussed.

Basically, knowledge bases use triplestores for hosting and managing their data. It is thus evident that the performance of those stores is mission critical for individual projects as well as for data integration on the Data Web in general. Consequently, it is of central importance during the implementation of any of these applications to have a clear picture of the strengths and weaknesses of the available triplestores. Most of the contemporary triplestores mainly adopt SPARQL [Prud'hommeaux and Seaborne, 2008] as the front-end query language for their inherent data. In this thesis, we propose a generic SPARQL benchmark creation methodology. This methodology is based on a flexible data generation mimicking an input data source, query-log mining, clustering and SPARQL feature analysis. We apply the proposed methodology to datasets of various sizes derived from the DBpedia knowledge base.

The significance of knowledge bases is mainly determined by the quality of their inherent data. In other words, when the data of a knowledge base is of high quality, this ensures success and value for that knowledge base, and vice versa. The quality of the data of a knowledge base depends on several factors, e.g. the origin (the source) of data, and stability of the extraction software. One of the most important aspects of knowledge quality is provenance. While provenance is an important aspect of data quality [Hartig, 2009], to date only few knowledge bases actually provide provenance information. For instance, less than 3% of the more than 708.19 million RDF (Resource Description Framework) documents indexed by Sindice<sup>5</sup> contain metadata such such as creator, created, source, modified, contributor, or provenance.<sup>6</sup> This lack of provenance information makes the validation of the facts in such knowledge bases utterly tedious. In addition, it hinders the adoption of such data in business applications as the data is not trusted [Hartig, 2009]. In this work, we propose a fact validation approach and tool which can make use of one of the largest information sources, i.e. the Web.

---

<sup>5</sup><http://www.sindice.com>

<sup>6</sup>Data retrieved on July 9, 2013

## 1.2. Contributions

This thesis proposes approaches to tackle each of the aforementioned challenges. Regarding the first challenge, i.e. the challenge of maintaining the DBpedia data up-to-date, we develop the *DBpedia Live* framework which aims at keeping DBpedia always in synchronization with Wikipedia with a minimal delay of only a few minutes. The main motivation behind this enhancement is that our approach turns DBpedia into a real-time editable knowledge base, while retaining the tight coupling to Wikipedia. It also opens the door for an increased use of DBpedia in different scenarios. For instance, a user may like to add to her movie website a list of highest grossing movies produced in the current year. Due to the live extraction process, this becomes much more appealing, since the contained information will be as up-to-date as Wikipedia instead of being several months delayed.

Overall, we have accomplished the following contributions:

1. migration the previous incomplete DBpedia Live framework, which was PHP-based, to the new Java-based framework, which also maintains up-to-date information,
2. addition of abstract extraction capability,
3. re-extraction of mapping-affected pages,
4. flexible low-priority re-extraction of pages, which have not been modified for a longer period of time – this allows changes in the underlying extraction framework, which potentially affect all Wikipedia pages, while still processing the most recent Wikipedia changes,
5. publishing added and deleted triples as compressed N-Triples file,
6. synchronization of the ontology of DBpedia with the ontology contained in the mappings wiki,
7. building a synchronization tool, which downloads those those compressed N-Triples files, and update another triplestore with them accordingly, in order to keep it in synchronization with ours,
8. deployment of the framework on our server.

Apart from DBpedia Live framework, we also devise a data quality assessment methodology and empirically assess based on this methodology the data quality of DBpedia. We crowd-source the quality assessment of individual DBpedia resources, in order to evaluate the type and extent of data quality problems occurring in DBpedia.

With regard to the challenge of developing a benchmark for RDF triplestores, we have developed a novel benchmark called *DBPSB* (i.e. DBpedia SPARQL Benchmark). We apply this methodology on the DBpedia dataset and its SPARQL query log. However, the same methodology can be used to obtain application-specific benchmarks for other knowledge bases and query workloads. In general, our benchmark follows the four key requirements for domain specific benchmarks which are postulated in the Benchmark Handbook [Gray, 1991], i.e. it is (1) relevant, thus testing typical operations within the specific domain, (2) portable, i.e. executable on different platforms, (3) scalable, e.g. it is possible to run the benchmark on both small and very large data sets, and (4) it is understandable.

Basically, the DBPSB has the following features:

1. it is based on real data, as it uses the DBpedia knowledge base as its test dataset,
2. it uses real queries to measure the performance of the test triplestores,
3. the queries used for performance evaluation are selected carefully to cover a wide spectrum of SPARQL construct,
4. the triplestores are tested through datasets of different sizes, in order to assess their scalability.

Regarding the data provenance, we have developed the *DeFacto* (Deep Fact Validation) system, which implements algorithms for validating statements, specifically RDF triples, by finding confirming sources for it on the web. It takes a statement as input and then tries to find evidence for the validity of that statement by searching for textual information in the web. In contrast to typical search engines, it does not just search for textual occurrences of parts of the statement, but tries to find webpages which contain the actual statement phrased in natural language. It presents the user with a confidence score for the input statement as well as a set of excerpts of relevant webpages, which allows the user to manually judge the presented evidence.

DeFacto has two major use cases: (1) Given an existing true statement, it can be used to find provenance information for it. For instance, the WikiData project<sup>7</sup> aims to create a collection of facts, in which sources should be provided for each fact. DeFacto could be used to achieve this task. (2) It can check whether a statement is likely to be true, provide the user with a confidence score in whether the statement is true and evidence for the score assigned to the statement. Our main contributions are thus as follows:

1. an approach that allows checking whether a webpage confirms a fact, i.e., an RDF triple,

---

<sup>7</sup><http://meta.wikimedia.org/wiki/Wikidata>

2. an adaptation of existing approaches for determining indicators for trustworthiness of a webpage,
3. an automated approach to enhancing knowledge bases with RDF provenance data at triple level,
4. a running prototype of DeFacto, the first system able to provide useful confidence values for an input RDF triple given the Web as background text corpus.

## 1.3. Chapter Overview

Chapter 2 introduces the Semantic Web and its associated technologies which constitutes the basic scientific background required for the reader to understand the thesis. It starts by defining the Semantic Web, and afterwards it discusses the Resource Description Framework (RDF), and its components. It also explains the various RDF serialization formats, e.g. N-Triples, and the differences among them. Then, it moves to the crucial topic of Semantic Web, specifically the ontology and the various languages that can be used to develop the ontologies. Eventually, it describes the SPARQL query language and what the triplestores are and how they support the SPARQL language.

Chapter 3 gives a broad overview over the DBpedia project and how it works. First it gives a general introduction to the DBpedia project, its objectives, the languages it supports, and the applications that depend on it. Furthermore, it outlines the extraction framework, its core extractors, and the part of the Wikipedia article each extractor handles. Finally, it discusses the ontology on which DBpedia relies and the mappings between Wikipedia attributes and the ontology properties.

Chapter 4 introduces the live extraction process which aims at keeping the DBpedia data up-to-date, in order to always reflect the current state of Wikipedia. In the beginning, it outlines the general system architecture of DBpedia Live, and then it details the extraction manager, which constitutes the core of the live extraction process. Afterwards, it lists the new features of DBpedia Live, and explains each feature, how it works, and its value to DBpedia. Lastly, it gives the list of important URLs, and presents a graph indicating how important DBpedia Live is to the community.

Chapter 5 deals mainly with the problem of data quality and data validation and provenance. Firstly, it describes a data quality evaluation methodology including the assessment methodology used, the taxonomy of quality problems and issues, and the automated quality evaluation tool. It then presents the results of applying this methodology on DBpedia as a use case, in order to measure its inherent quality. Then, it introduces a data validation approach called DeFacto, and describes its architecture and how it works. Moreover, it details the trustworthiness analysis approach used in DeFacto and the features upon which DeFacto depends. Eventually, it presents the results of DeFacto.

Chapter 6 addresses the issue of benchmarking RDF triplestores. It introduces our novel RDF benchmark called DBpedia SPARQL Benchmark (DBPSB), which is based on real data and uses real queries for benchmarking. It starts by giving an overview on the benchmarking in general and its objectives. Thereafter, it explains the dataset generation strategy, and the benchmarking queries selection process. It ends by describing the experimental setup, and comparing the results of the tested triplestores for both versions of DBPSB.

Chapter 7 discusses the state-of-the-art research work related to each area of the thesis. It discusses the approaches that extract semantic data Wikipedia , and then talks about the quality assessment methodologies. Then it describes existing RDF benchmarks, and the data provenance strategies.

Eventually, Chapter 8 concludes the each point of the thesis and proposes some future work for it.

## 2. Semantic Web Technologies

This chapter gives a general overview on Semantic Web. Thereafter, it describes the RDF serialization formats, the ontology and its languages in detail. This chapter is mainly based on [Yu, 2007].

### 2.1. Semantic Web Definition

There are several different definitions about what the Semantic Web is. Tim Berners-Lee, the inventor of Semantic Web, defines it as "The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation [Berners-Lee et al., 2001]." In other words, says that Semantic Web makes the machines capable of not only presenting data but also processing them.

There is a team at the World Wide Web consortium (W3C) working on improving, and standardizing the system, and many languages, publications, tools etc., have already been developed. They have defined it as "Semantic Web is the idea of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration, and reuse of data across various applications [W3C, 2009]." In other words, Semantic Web is machine-readable and machine-understandable Web. You can also think of it as being an efficient way of representing data on the World Wide Web, or as a globally linked database.

Semantic Web depends on several technologies including Resource Description Framework (RDF), Uniform Resource Identifiers (URIs), and others. In the following section we elaborate each of these technologies.

### 2.2. Resource Description Framework - RDF

RDF is a language used to describe the information of any Web resource. It uses XML as its base standard. The Web resource can be anything, ranging from Web page to a book or a person. It plays a similar role to Semantic Web, as the role of HTML to the conventional Web.

The properties of RDF are:

- RDF is a W3C recommendation [W3C, 2004], and it is centered around metadata, i.e. this metadata on the one hand adds meaning to the Web

resource it describes, and on the other hand it can be understood and processed by machines,

- RDF can describe a Web resource regardless of the domain,
- RDF is the foundation for encoding and reusing structured metadata,
- RDF is structured which makes it machine processable. Thus machines can accomplish some valuable operations with the knowledge represented in RDF,
- RDF enables interoperability among applications exchanging machine-readable information on the Web.

### 2.2.1. RDF Resource

A resource is anything, e.g. a city, a person, or a Web site, which is described by RDF expressions. The resource is identified and characterized by what is so called **Uniform Resource Identifier (URI)**. The rationale of using URIs for identifying resources is that the name of a resource must be globally unique.

Actually, URLs (Uniform Resource Locators), commonly used for accessing Web sites, are simply a subset of URIs. Like URLs, URIs take the same format, e.g. `http://dbpedia.org/resource/William_Shakespeare`. The reason behind this is that the domain name part used in the URL, i.e. `http://dbpedia.org/` in our example, is ensured to be unique. Therefore the global uniqueness of the resource is ensured. That domain name part plays the role of a namespace. Whereas URLs always refer to Web sites, URIs may or may not refer to an actual Web site.

### 2.2.2. RDF Property

A property is a form of resource which is used to declare a trait, i.e. it can be used to denote a characteristic, an attribute, or a relation of the resource it is related to. The property is also called predicate. For instance, `http://dbpedia.org/ontology/birthPlace`, denotes the birth place of a human being. In other words, this property relates a resource representing a person, to his/her birth place.

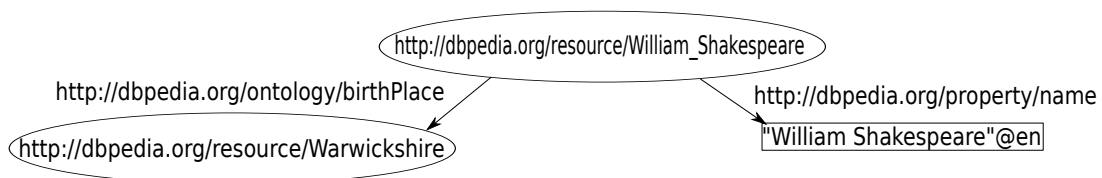


Figure 2.1.: RDF statements represented as a directed graph.



### 2.2.3. RDF Statement

An RDF statement describes a property of a specific resource. It is also called a triple. It has the following format:

”resource (subject) + property (predicate) + property value (object)” [Yu, 2007]. The property value of a statement can be another resource or a string literal.

For example:

```
<http://dbpedia.org/resource/William_Shakespeare>
  <http://dbpedia.org/ontology/birthPlace>
    <http://dbpedia.org/resource/Warwickshire>.
```

This RDF statement simply says ”The subject identified by `http://dbpedia.org/resource/William_Shakespeare` has property identified by `http://dbpedia.org/ontology/birthPlace`, whose value is equal to `http://dbpedia.org/resource/Warwickshire`”. This means that person ”William Shakespeare” has a ”birthPlace” whose value is ”Warwickshire”.

Another example:

```
<http://dbpedia.org/resource/William_Shakespeare>
  <http://dbpedia.org/property/name>
    ”William Shakespeare”@en.
```

This RDF statement says ”The subject identified by `http://dbpedia.org/resource/William_Shakespeare` has property identified by `http://dbpedia.org/property/name`, whose value is equal to ”William Shakespeare””. This means that person ”William Shakespeare” has a ”name” whose value is ”William Shakespeare”, and the trailing ”@en” is the English language tag. Actually, RDF statements can also be expressed as directed graphs, as shown in Figure 2.1 .

It is worth pointing out here that the subject or the object or both can be an anonymous resource, called ”blank node”. The blank node has no data but it plays the role of a parent of other properties, in order to allow them to appear. In order to differentiate each blank node from the others, the parser creates an *internal* unique identifier for each blank node. In other words, this identifier given to the blank node helps in identifying that node in a certain RDF document, whereas the URI given to a resource is guaranteed to be globally unique.

Since the URIs can be so large, there is a short format for writing them, by using a prefix. For instance, if we use `http://dbpedia.org/resource/` as a prefix and give it a name e.g. `dbpedia`, then resource `http://dbpedia.org/resource/William_Shakespeare` can be written as `dbpedia:William_Shakespeare`. Similarly, if `http://dbpedia.org/ontology/` is used as a prefix with name `dbo`, then property `http://dbpedia.org/ontology/birthPlace`, can be written as `dbo:birthPlace` for short. This format is very useful in writing RDF statements.

Whenever more triples, describing a specific resource are added, this means that the machine gets more knowledge about that resource. Table 2.1 shows more RDF statements about William Shakespeare. This means that the resource of William

Subject	Predicate	Object
dbpedia:William_Shakespeare	rdf:type	dbo:Person
dbpedia:William_Shakespeare	dbo:birthPlace	dbpedia:Warwickshire
dbpedia:William_Shakespeare	dbo:child	dbpedia:Judith_Quiney
dbpedia:William_Shakespeare	dbo:occupation	dbpedia:Poet
dbpedia:William_Shakespeare	dbo:deathDate	"1616-04-23"^^xsd:date
dbpedia:William_Shakespeare	dbpprop:name	"William Shakespeare"@en
dbpedia:Judith_Quiney	dbo:spouse	dbpedia:Thomas_Quiney

Table 2.1.: Sample RDF statements.

Shakespeare is the subject of other statements, which give more details about that resource. Note that the object of a particular statement can be in turn the subject of other statement(s), e.g. William Shakespeare has a child identified by URI `dbpedia:Judith_Quiney` and the knowledge base contains more information about that child as well. Note also that the object of the fifth statement is a date, so it has a trailing datatype. This small knowledge base can also be viewed as a directed graph as shown in Figure 2.2.

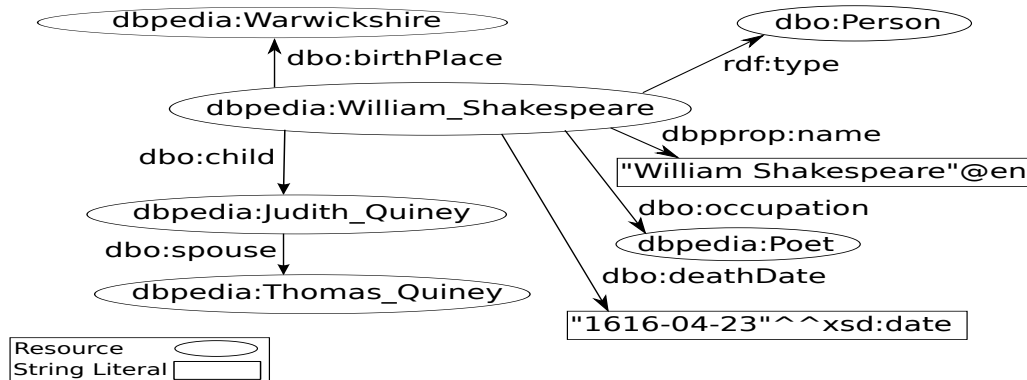


Figure 2.2.: Small knowledge base about William Shakespeare represented as a graph.

Now using those simple RDF statements you can pose complex queries to the machine, e.g. "Who is the spouse of William Shakespeare's child?".

## 2.2.4. RDF Serialization Formats

Serializing RDF data is a very crucial issue. There are several formats for serializing RDF data:

### 2.2.4.1. N-Triples

N-Triples is a simple line-based RDF serialization format. Each RDF triple is written as a separate line and terminated by a period (.). Typically files with

```

1 <http://dbpedia.org/resource/William_Shakespeare> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://dbpedia.org/ontology/Person> .
2 <http://dbpedia.org/resource/William_Shakespeare> <http://dbpedia.org/ontology/
birthPlace> <http://dbpedia.org/resource/Warwickshire> .
3 <http://dbpedia.org/resource/William_Shakespeare> <http://dbpedia.org/ontology/
child> <http://dbpedia.org/resource/Judith_Quiney> .
4 <http://dbpedia.org/resource/William_Shakespeare> <http://dbpedia.org/ontology/
occupation> <http://dbpedia.org/resource/Poet> .
5 <http://dbpedia.org/resource/William_Shakespeare> <http://dbpedia.org/ontology/
deathDate> "1616-04-23"^^<http://www.w3.org/2001/XMLSchema#date> .
6 <http://dbpedia.org/resource/William_Shakespeare> <http://dbpedia.org/property/
name> "William_Shakespeare"@en .
7 <http://dbpedia.org/resource/Judith_Quiney> <http://dbpedia.org/ontology/spouse>
<http://dbpedia.org/resource/Thomas_Quiney> .

```

Figure 2.3.: Sample N-Triples format.

```

1 <rdf:RDF xmlns:log="http://www.w3.org/2000/10/swap/log#" xmlns:rdf="http://www.
w3.org/1999/02/22-rdf-syntax-ns#">
2 <rdf:Description rdf:about="http://dbpedia.org/resource/Judith_Quiney">
3 <spouse xmlns="http://dbpedia.org/ontology/" rdf:resource="http://dbpedia.
org/resource/Thomas_Quiney"/>
4 </rdf:Description>
5
6 <Person xmlns="http://dbpedia.org/ontology/" rdf:about="http://dbpedia.org/
resource/William_Shakespeare">
7 <birthPlace rdf:resource="http://dbpedia.org/resource/Warwickshire"/>
8 <child rdf:resource="http://dbpedia.org/resource/Judith_Quiney"/>
9 <deathDate rdf:datatype="http://www.w3.org/2001/XMLSchema#date">1616-04-23<
/ deathDate>
10 <occupation rdf:resource="http://dbpedia.org/resource/Poet"/>
11 <name xmlns="http://dbpedia.org/property/" xml:lang="en">William
Shakespeare</name>
12 </Person>
13 </rdf:RDF>

```

Figure 2.4.: Sample RDF/XML format.

N-Triples have the .nt extension [Grant and Beckett, 2004]. Figure 2.3 indicates our sample triples encoded in N-Triples format.

#### 2.2.4.2. RDF/XML

RDF/XML represents RDF triples in XML format [Beckett, 2004]. RDF/XML format is more convenient for machines than N-Triples. Figure 2.4 shows our RDF example in RDF/XML format. Files containing RDF/XML data have .rdf as extension.

#### 2.2.4.3. N3

N3 stands for Notation3 and is a shorthand notation for representing RDF graphs. N3 was designed to be easily read by humans, and it is not an XML-compliant language [Berners-Lee and Connolly, 2011]. Figure 2.5 shows our RDF example in N3 format. Files containing RDF data in N3 format normally have .n3 extension.

```
1 @prefix dbo: <http://dbpedia.org/ontology/> .
2 @prefix dbpprop: <http://dbpedia.org/property/> .
3 @prefix dbpedia: <http://dbpedia.org/resource/> .
4 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5
6 dbpedia:William_Shakespeare a dbo:Person;
7     dbo:birthPlace dbpedia:Warwickshire;
8     dbo:child dbpedia:Judith_Quiney;
9     dbo:deathDate "1616-04-23"^^xsd:date;
10    dbo:occupation dbpedia:Poet;
11    dbpprop:name "William Shakespeare"@en .
12
13 dbpedia:Judith_Quiney dbo:spouse dbpedia:Thomas_Quiney .
```

Figure 2.5.: Sample N3 format.

### 2.2.4.4. Turtle

Turtle is a subset of N3. Turtle stands for Terse RDF Triple Language. Turtle files have a .ttl extension [Dave and Berners-Lee, 2011]. This particular serialization is popular among developers of the Semantic Web.

### 2.2.5. Ontology

W3C defines an ontology as "An ontology defines the terms used to describe and represent an area of knowledge." [Heflin, 2004].

This definition has several aspects that should be discussed. First, the definition states that ontology is used to describe and represent a specific area of knowledge. This means that the ontology is domain specific; it does not cover all knowledge disciplines, instead a specific area of knowledge. A domain is simply a specific area or discipline of knowledge, such as literature, medicine, or engineering.

Second, the ontology defines the terms and relationships among those terms as well. The terms are often called classes, or concepts. The relationships among these classes can be expressed using a hierarchy, i.e. superclasses represent higher-level concepts and subclasses represent finer concepts. The finer classes, or lower level classes, include all attributes and features that their superclasses have.

Third, in addition to defining the relationships among classes, there is another sort of relationships expressed by using a special set of terms called properties. These properties define various characteristics of the classes, and they can relate different classes to each other. In other words, the relationships among classes are not only hierarchical relationships (parent and child classes), but also the relationships expressed using properties.

In other words, an ontology defines a set of classes (e.g. "Person", "Book", "Writer"), and their hierarchy, i.e. which class is a subclass of another one (e.g. "Writer" is a subclass of "Person"). It also defines how those classes interact with each other, i.e. how different class are connected to each other via properties (e.g. a "Book" has an author of type "Writer" ).

Figure 2.6 indicates a sample ontology taken from the DBpedia ontology. This

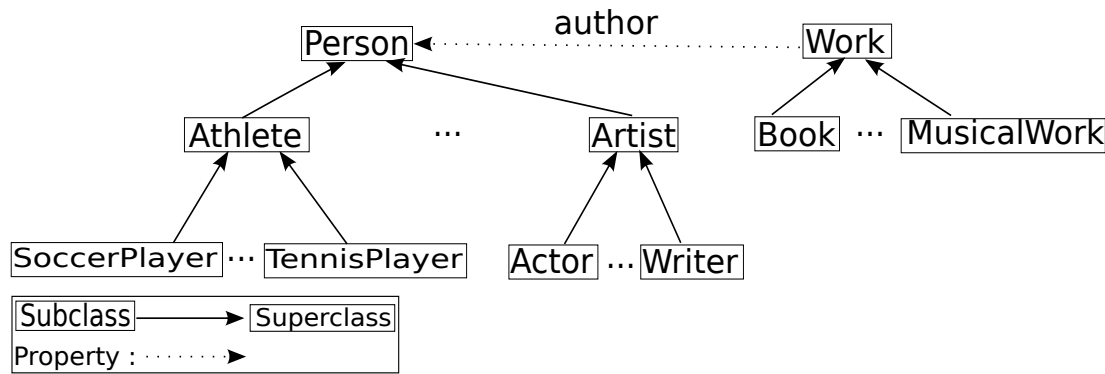


Figure 2.6.: Sample ontology snapshot taken from DBpedia ontology.

ontology says that there is a class called "Writer" which is a subclass of "Artist", which is in turn a subclass of "Person". William Shakespeare, Johann Wolfgang von Goethe, and Dan Brown are candidate instances of the class "Writer". The same applies on class "Work" and its subclasses. Note that there is a property called "author" relating an instance of class "Work" to an instance of class "Person", i.e. it relates a work to its author. For instance, the book titled "First Folio" is an instance of classes "Work" and "Book", and related via property "author" to its author "William Shakespeare" which is an instance of classes "Person", "Artist", and "Writer".

So, why do need ontologies?. The benefits of ontology are:

- it provides a common and shared understanding/definition about certain key concepts in the domain,
- it provides a way to for domain knowledge reuse,
- it makes the domain assumptions explicit,
- it provides a way to encode knowledge and semantics such that machines can understand it.

## 2.2.6. Ontology Languages

The question now is "What are the languages used to create ontologies?". Actually, there are several languages which can be used to encode ontologies, e.g. RDF Schema (RDFS), and Web Ontology Language (OWL).

### 2.2.6.1. RDFS

RDFS is an ontology language, which can be used to create vocabularies defining classes, their respective subclasses, and properties of the various RDF resources. Furthermore RDFS is a W3C recommendation [Brickley and Guha, 2004]. RDFS

ontology language also relates the properties to the classes it defines. RDFS can add semantics to the RDF predicates and resources. In other words, it describes the meaning of a specific term by defining its associated properties and what sort of objects these properties might take. It is worthy noting here that RDFS is written in RDF, so any RDFS document is a legal RDF document.

### 2.2.6.2. OWL

The Web Ontology Language (OWL) is used to create ontologies and it is also a W3C recommendation [Bechhofer et al., 2004]. It is built on RDFS. We can say that, "**OWL = RDFS + new constructs for expressiveness**" [Yu, 2007]. All RDFS classes and properties can be also used for creating OWL ontologies. OWL and RDFS have the same purpose which is describing the classes, the properties, and respective relations among those classes. However, OWL has an advantage over RDFS which is its expressiveness power. In other words, OWL can describe more complex relationships.

Due its expressiveness power, most ontology developers use OWL to develop their ontologies. As an example, showing how powerful OWL is, is that the ontology developer can create a new class as the union or intersection of two or more classes. With OWL you can also declare that two classes are representing the same thing. For instance, consider the case that there are two separate ontologies created by different developers, in the first ontology there is a class called "Poet", and in the other ontology there is a class called "PoetryWriter". Actually, those classes are equivalent to each other, in RDFS you cannot declare that those classes are equivalent, but with OWL you can.

OWL provides some powerful features for properties as well. For example, in OWL you can declare that two properties are the inverse of each other, (e.g. "author", and "isAuthorOf"). Figure 2.7 indicates a part of our ontology expressed in OWL.

Note that for property author we have defined two properties domain, and range. The domain property defines the class of instances which can be the subject of that property (author property), while the range property defines the class of instances which can be the object of that property.

OWL has many powerful features, interested reader can find more about those feature in [Bechhofer et al., 2004].

### 2.2.7. SPARQL Query Language

"The SPARQL Protocol and RDF Query Language (SPARQL) is a query language and protocol for RDF." [Clark et al., 2008]. It is a W3C standard, and it is used to ask queries against RDF graphs. SPARQL allows the user to write queries that consist of triple patterns, conjunctions (logical "and"), disjunctions (logical "or"), and/or a set of optional patterns [Wikipedia, 2013]. Examples of those optional patterns are, **FILTER**, **REGEX**, and **LANG**.

```

1 <http://dbpedia.org/ontology/Person> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
  type> <http://www.w3.org/2002/07/owl#Class> .
2 <http://dbpedia.org/ontology/Artist> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
  type> <http://www.w3.org/2002/07/owl#Class> .
3 <http://dbpedia.org/ontology/Artist> <http://www.w3.org/2000/01/rdf-schema#
  subclassOf> <http://dbpedia.org/ontology/Person> .
4 <http://dbpedia.org/ontology/Writer> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
  type> <http://www.w3.org/2002/07/owl#Class> .
5 <http://dbpedia.org/ontology/Writer> <http://www.w3.org/2000/01/rdf-schema#
  subclassOf> <http://dbpedia.org/ontology/Artist> .
6 <http://dbpedia.org/ontology/Work> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
  type> <http://www.w3.org/2002/07/owl#Class> .
7 <http://dbpedia.org/ontology/Book> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
  type> <http://www.w3.org/2002/07/owl#Class> .
8 <http://dbpedia.org/ontology/Book> <http://www.w3.org/2000/01/rdf-schema#
  subclassOf> <http://dbpedia.org/ontology/Work> .
9 <http://dbpedia.org/ontology/author> <http://www.w3.org/1999/02/22-rdf-syntax-ns#
  type> <http://www.w3.org/2002/07/owl#ObjectProperty> .
10 <http://dbpedia.org/ontology/author> <http://www.w3.org/2000/01/rdf-schema#domain
  > <http://dbpedia.org/ontology/Work> .
11 <http://dbpedia.org/ontology/author> <http://www.w3.org/2000/01/rdf-schema#range>
  <http://dbpedia.org/ontology/Person> .

```

Figure 2.7.: OWL representation of a part our ontology in N-Triples format.

```

1 PREFIX dbp: <http://dbpedia.org/resource/>
2 PREFIX dbo: <http://dbpedia.org/ontology/>
3 SELECT ?spouse
4 WHERE {dbpedia:William_Shakespeare dbo:child ?child.
5 ?child dbo:spouse ?spouse. }

```

Figure 2.8.: SPARQL query to get the spouse of Shakespeare’s child.

The SPARQL query specifies the pattern(s) that the resulting data should satisfy. The results of SPARQL queries can be result sets or RDF graphs. SPARQL has four query forms, specifically **SELECT**, **CONSTRUCT**, **ASK**, and **DESCRIBE** [Prud’hommeaux and Seaborne, 2008].

Let’s take an example to clarify the usage of SPARQL. Assume that we want to ask the query ”Who is the spouse of William Shakespeare’s child?” to our small knowledge base. SPARQL language can be used to do so. Figure 2.8 shows a SPARQL query to get information about the spouse of Shakespeare’s child.

In Figure 2.8, lines 1, and 2, defines some prefixes in order to write URIs in their short forms. Line 3, declares the variables that should be rendered to the output of that query, which is only one variable `?spouse`. Note that SPARQL variables start either with a question mark ”?”, or with a dollar sign ”\$”. Line 4 states that for statement with subject `dbpedia:William_Shakespeare` and with property `dbo:child`, we want the value of its object to be assigned to a variable called `?child`. Upon execution, this variable will take the value of `dbpedia:Judith_Quiney`. In line 5, we want variable `?child` which now has the value `dbpedia:Judith_Quiney`, to be the subject of the other statement. In other words, the statement will be `dbpedia:Judith_Quiney dbo:spouse ?spouse`. Now, variable `?spouse` is the only unknown variable of the statement, and it will take the value `dbpedia:Thomas_Quiney`. Eventually, its value will be rendered to the

output.

### 2.2.8. Triplestore

The crucial question here is, "How do we store RDF data for efficient and quick access?". Basically, RDF data is stored in what is so called triplestores. A triplestore is a software program capable of storing and indexing RDF data efficiently, in order to enable querying this data easily and effectively. A triplestore for RDF data is like Relational Database Management System (DBMS) for relational databases.

Most triplestores support SPARQL query language for querying RDF data. As there are several DBMSs in the wild, such as Oracle<sup>1</sup>, MySQL<sup>2</sup>, and SQL Server<sup>3</sup>, there are also several triplestores. Virtuoso [Erling and Mikhailov, 2007], Sesame [Broekstra et al., 2002], and BigOWLIM [Bishop et al., 2011] are typical examples of triplestores. DBpedia is using Virtuoso as the underlying triplestore.

---

<sup>1</sup><http://www.oracle.com/us/products/database/overview/index.html>

<sup>2</sup><http://www.mysql.com>

<sup>3</sup><http://www.microsoft.com/en-us/sqlserver/default.aspx>



## 3. Overview on the DBpedia Project

This chapter gives an introduction to the DBpedia project in general, including its structure, its core extractors and the role of each one, etc. It also describes the significance of the project for the community. This chapter is based on [Lehmann et al., 2013], which I co-authored together with core members of the DBpedia community.

### 3.1. Introduction to DBpedia

The DBpedia community project extracts knowledge from Wikipedia and makes it widely available via established Semantic Web standards and Linked Data best practices. Wikipedia is currently the 6th most popular website<sup>1</sup>, the most widely used encyclopedia, and one of the finest examples of truly collaboratively created content. However, due to the lack of the exploitation of the inherent structure of Wikipedia articles, Wikipedia itself only offers very limited querying and search capabilities. For instance, it is difficult to find all rivers that flow into the Rhine or all Italian composers from the 18th century. One of the goals of the DBpedia project is to provide those querying and search capabilities to a wide community by extracting structured data from Wikipedia which can then be used for answering expressive queries.

The core of DBpedia consists of an infobox extraction process, which was first described in [Auer and Lehmann, 2007]. Infoboxes are templates contained in many Wikipedia articles. They are usually displayed in the top right corner of articles and contain factual information (cf. Figure 3.1). The infobox extractor processes an infobox as follows: The DBpedia URI, which is created from the Wikipedia article URL, is used as subject. The DBpedia project was started in 2006 and has meanwhile attracted significant interest in research and practice. It has been a key factor for the success of the Linked Open Data initiative and serves as an interlinking hub for other datasets (see Section 3.4). For the research community, DBpedia provides a testbed serving real data spanning various domains and more than 100 language editions. Numerous applications, algorithms and tools have been build around or applied to DBpedia. Due to the continuous growth of Wikipedia and improvements in DBpedia, the extracted data provides an increasing

---

<sup>1</sup>See <http://www.alexa.com/topsites>. Retrieved in February 2013.

added value for data acquisition, reuse and integration tasks within organizations.

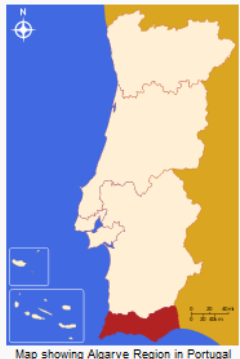
One of the reasons why DBpedia's data quality has improved over the past years is that the structure of the knowledge in DBpedia itself is maintained by its community. Most importantly, the community creates mappings from Wikipedia information representation structures to the DBpedia ontology. This ontology unifies different template structures, which will later be explained in detail – both within single Wikipedia language editions and across currently 27 different languages. The maintenance of different language editions of DBpedia is spread across a number of organizations. Each organization is responsible for the support of a certain language. The local DBpedia chapters are coordinated by the DBpedia Internationalization Committee. In addition to multilingual support, DBpedia also provides data-level links into more than 30 external datasets, which are partially also contributed from partners beyond the core project team.

```

{{Infobox settlement
 | official_name      = Algarve
 | settlement_type    = Region
 | image_map          = LocalRegiaoAlgarve.svg
 | mapsize            = 180px
 | map_caption        = Map showing Algarve
                      Region in Portugal
 | subdivision_type   = [[Countries of the
                      world|Country]]
 | subdivision_name   = {{POR}}
 | subdivision_type3  = Capital city
 | subdivision_name3  = [[Faro, Portugal|Faro]]
 | area_total_km2    = 5412
 | population_total   = 410000
 | timezone           = [[Western European
                      Time|WET]]

 | utc_offset         = +0
 | timezone_DST       = [[Western European
                      Summer Time|WEST]]

 | utc_offset_DST     = +1
 | blank_name_sec1    = [[NUTS]] code
 | blank_info_sec1    = PT15
 | blank_name_sec2    = [[GDP]] per capita
 | blank_info_sec2    = €19,200 (2006)
}}
```



Map showing Algarve Region in Portugal

<b>Country</b>	<span><span><span></span></span><span> </span></span> Portugal
<b>Capital city</b>	Faro
<b>Area</b>	
<span> </span> - Total	5,412 km <sup>2</sup> (2,089.6 sq mi)
<b>Population</b>	
<span> </span> - Total	410,000
<b>Time zone</b>	WET (UTC+0)
<span> </span> - Summer (DST)	WEST (UTC+1)
<b>NUTS code</b>	PT15
<b>GDP per capita (PPS)</b>	€ 19,200 (2006) <sup>[1]</sup>

Figure 3.1.: Mediawiki infobox syntax for Algarve (left) and rendered infobox (right).

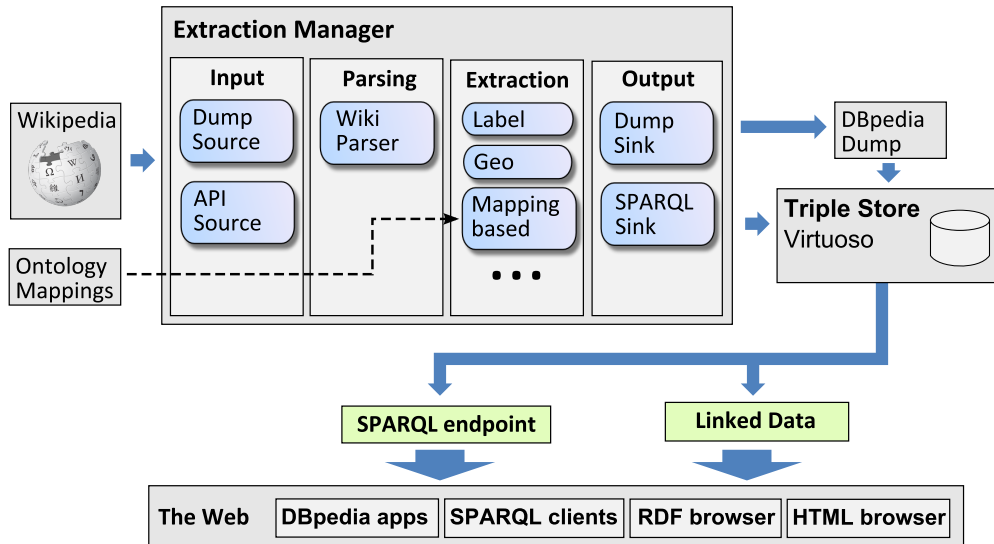


Figure 3.2.: Overview of DBpedia extraction framework.

## 3.2. DBpedia Extraction Framework

Wikipedia articles consist mostly of free text, but also comprise various types of structured information in the form of wiki markup. Such information includes infobox templates, categorization information, images, geo-coordinates, links to external web pages, disambiguation pages, redirects between pages, and links across different language editions of Wikipedia. The DBpedia extraction framework extracts this structured information from Wikipedia and turns it into a rich knowledge base. In this section, we give an overview of the DBpedia knowledge extraction framework.

### 3.2.1. General Architecture

Figure 3.2 shows an overview of the DBpedia extraction framework. The DBpedia extraction is structured into four phases:

**Input:** Wikipedia pages are read from an external source. Pages can either be read from a Wikipedia dump or directly fetched from a MediaWiki installation using the MediaWiki API.

**Parsing:** Each Wikipedia page is parsed by the wiki parser. The wiki parser transforms the source code of a Wikipedia page into an Abstract Syntax Tree.

**Extraction:** The Abstract Syntax Tree of each Wikipedia page is forwarded to the extractors. DBpedia offers extractors for many different purposes, for instance, to extract labels, abstracts or geographical coordinates. Each

extractor consumes an Abstract Syntax Tree and yields a graph of RDF statements.

**Output:** The collected RDF statements are written to a sink. Different formats, such as N-Triples are supported.

#### 3.2.2. Extractors

The DBpedia extraction framework employs various extractors for translating different parts of Wikipedia pages to RDF statements. A list of all available extractors is shown in Table 3.1. DBpedia extractors can be divided into four categories:

**Mapping-Based Infobox Extraction:** The *mapping-based infobox extraction* uses manually written mappings that relate infoboxes in Wikipedia to terms in the DBpedia ontology. The mappings also specify a datatype for each infobox property and thus help the extraction framework to produce high quality data. The mapping-based extraction will be described in detail in Section 3.2.4.

**Raw Infobox Extraction:** The *raw infobox extraction* provides a direct mapping from infoboxes in Wikipedia to RDF. As the raw infobox extraction does not rely on explicit extraction knowledge in the form of mappings, the quality of the extracted data is lower. The raw infobox data is useful, if a specific infobox has not been mapped yet and thus is not available in the mapping-based extraction.

**Feature Extraction:** The *feature extraction* uses a number of extractors that are specialized in extracting a single feature from an article, such as a label or geographic coordinates. Section 3.2.6 discusses some more of them.

Name	Description	Example
abstract	Extracts page abstracts.	dbpedia:Berlin dbo:abstract "Berlin is the capital city of (...)"
article categories	Extracts links from concepts to categories using the SKOS vocabulary.	dbpedia:Oliver_Twist dc:subject dbpedia:Category:English_novels
category label	Extracts labels for Categories.	dbpedia:Category:English_novels rdfs:label "English novels"
disambiguation	Extracts disambiguation links.	dbpedia:Alien dbo:wikiPageDisambiguates dbpedia:Alien_(film)
external links	Extracts links to external web pages.	dbpedia:Animal_Farm dbo:wikiPageExternalLink <http://books.google.com/?id=RBGmrDnBs8UC>
geo coordinates	Extracts geo-coordinates.	dbpedia:Berlin georss:point "52.5006 13.3989"
homepage	Extracts links to the official homepage of an instance.	dbpedia:Alabama foaf:homepage <http://alabama.gov/>
image	Extracts the first image of a Wikipedia page.	dbpedia:Berlin foaf:depiction <http://.../Overview_Berlin.jpg>
infobox	Extracts all properties from all infoboxes.	dbpedia:Animal_Farm dbo:date "March 2010"
interlanguage links	Extracts interwiki links.	
label	Extracts labels to articles based on their title.	dbpedia:Berlin rdfs:label "Berlin"
mappings	Extraction based on mappings of Wikipedia infoboxes to the DBpedia ontology.	dbpedia:Berlin dbo:country dbpedia:Germany
meta information	Extracts page's meta-information, e.g. editlink, and revisionlink.	dbpedia:Berlin <http://dbpedia.org/meta/editlink> <http://en.wikipedia.org/w/index.php?title=Berlin&action=edit>
page ID	Extracts page ids of articles.	dbpedia:Autism dbo:wikiPageID "25"
page links	Extracts internal links between DBpedia instances.	dbpedia:Autism dbo:wikiPageWikiLink dbpedia:Human_brain
persondata	Extracts information about persons	dbpedia:Andre_Agassi foaf:birthDate "1970-04-29"
PND	Extracts PND (Personennamendatei) data about a person.	dbpedia:William_Shakespeare dbo:individualisedPnd "118613723"
redirects	Extracts redirect links between Articles in Wikipedia.	dbpedia:Artificial_Languages dbo:wikiPageRedirects dbpedia:Constructed_language
revision ID	Extracts revision ids to articles.	dbpedia:Autism <http://www.w3.org/ns/prov#wasDerivedFrom> <http://en.wikipedia.org/wiki/Autism?oldid=495234324>
SKOS categories	Extracts information about which concept is a category and how categories are related using the SKOS Vocabulary.	dbpedia:Category:World_War_II skos:broader dbpedia:Category:Modern_history
wiki page	Extracts links to corresponding Articles in Wikipedia.	dbpedia:AnAmericanInParis foaf:isPrimaryTopicOf <http://en.wikipedia.org/wiki/AnAmericanInParis>

Table 3.1.: Overview of DBpedia extractors.

### 3.2.3. Raw Infobox Extraction

The type of Wikipedia content that is most valuable for the DBpedia extraction are infoboxes. Infoboxes are frequently used to list an article's most relevant facts as a table of attribute-value pairs on the top right-hand side of the Wikipedia page (for right-to-left languages on the top left-hand side respectively). Infoboxes that appear in a Wikipedia article are based on a template that specifies a list of attributes that can form the infobox. A wide range of infobox templates are used in Wikipedia. Common examples are templates for infoboxes that describe persons, organizations or automobiles. As Wikipedia's infobox template system has evolved over time, different communities of Wikipedia editors use different templates to describe the same type of things (e.g. `Infobox_city_japan`, `Infobox_swiss_town` and `Infobox_town_de`). In addition, different templates use different names for the same attribute (e.g. `birthplace` and `placeofbirth`). As many Wikipedia editors do not strictly follow the recommendations given on the page that describes a template, attribute values are expressed using a wide range of different formats and units of measurement. An excerpt of an infobox that is based on a template for describing automobiles is shown below:

```
{{Infobox automobile
| name           = Ford GT40
| manufacturer   = [[Ford Advanced Vehicles]]
| production     = 1964-1969
| engine         = 4181cc [[V8 engine|V-8]]
(...)
}}
```

In this infobox, the first line specifies the infobox type and the subsequent lines specify various attributes of the described entity.

An excerpt of the extracted data is as follows:<sup>2</sup>

```
dbpedia:Ford_GT40 [
  dbpprop:name "Ford GT40"@en;
  dbpprop:manufacturer dbr:Ford_Advanced_Vehicles;
  dbpprop:engine 4181, 4737, 4942, 6997;
  dbpprop:production 107 , 1964;
  (...)
] .
```

This extraction output has weaknesses: The resource is not associated to a class in the ontology and the engine and production values literal values, which are not meaningful. Those problems can be overcome by the mapping-based infobox extraction presented in the next subsection.

### 3.2.4. Mapping-Based Infobox Extraction

In order to homogenize the description of information in the knowledge base, in 2010 a community effort has been initiated to develop an ontology schema and

---

<sup>2</sup>We use `dbpedia` for `http://dbpedia.org/resource/`, `dbo` for `http://dbpedia.org/ontology/` and `dbpprop` for `http://dbpedia.org/property/` as prefixes throughout the thesis.

mappings from Wikipedia infobox properties to this ontology. The alignment between Wikipedia infoboxes and the ontology is performed via community-provided mappings that help to normalize name variations in properties and classes. Heterogeneity in the Wikipedia infobox system, like using different infoboxes for the same type of entity or using different property names for the same property (cf. Section 3.2.3), can be alleviated in this way.

This significantly increases the quality of the raw Wikipedia infobox data by typing resources, merging name variations and assigning specific datatypes to the values.

This effort is realized with the DBpedia Mappings Wiki<sup>3</sup>, a MediaWiki installation set up to enable the users to collaboratively create and edit mappings. These mappings are specified using the DBpedia Mapping Language. The mapping language makes use of MediaWiki templates that define DBpedia ontology classes and properties as well as template/table to ontology mappings. A mapping assigns a type from the DBpedia ontology to the entities that are described by the corresponding infobox. In addition, attributes in the infobox are mapped to properties in the DBpedia ontology. In the following, we show a mapping that maps infoboxes that use the `Infobox_automobile` template to the DBpedia ontology:

```

{{TemplateMapping
|mapToClass = Automobile
|mappings =
  {{PropertyMapping
  | templateProperty = name
  | ontologyProperty = foaf:name }}
  {{PropertyMapping
  | templateProperty = manufacturer
  | ontologyProperty = manufacturer }}
  {{DateIntervalMapping
  | templateProperty = production
  | startDateOntologyProperty = productionStartDate
  | endDateOntologyProperty = productionEndDate }}
  {{IntermediateNodeMapping
  | nodeClass = AutomobileEngine
  | correspondingProperty = engine
  | mappings =
    {{PropertyMapping
    | templateProperty = engine
    | ontologyProperty = displacement
    | unit = Volume }}
    {{PropertyMapping
    | templateProperty = engine
    | ontologyProperty = powerOutput
    | unit = Power }}
  }}
  (...)
}}
```

The RDF statements that are extracted from the previous infobox example are

<sup>3</sup><http://mappings.dbpedia.org>

shown below. As we can see, the production period is correctly split into a start year and an end year and the engine is represented by a distinct RDF node.

```
dbpedia:Ford_GT40 [
  rdf:type    dbo:Automobile;
  rdfs:label  "Ford GT40"@en;
  dbo:manufacturer
    dbpedia:Ford_Advanced_Vehicles;
  dbo:productionStartYear
    "1964"^^xsd:gYear;
  dbo:productionEndYear  "1969"^^xsd:gYear;
  dbo:engine [
    rdf:type  AutomobileEngine;
    dbo:displacement "0.004181";
  ]
  (...)
]
```

The DBpedia Mapping Wiki is not only used to map different templates within a single language edition of Wikipedia to the DBpedia ontology, but is used to map templates from all Wikipedia language editions to the shared DBpedia ontology. Section 3.3 shows how the infobox properties *author* and *συγγραφέας* – author in Greek – are both being mapped to the global identifier `dbo:author`. That means, in turn, that information from all language versions of DBpedia can be merged and DBpedias for smaller languages can be augmented with knowledge from larger DBpedias such as the English edition. Conversely, the larger DBpedia editions can benefit from more specialized knowledge from localized editions, such as data about smaller towns which is often only present in the corresponding language edition [Tacchini et al., 2009].

Besides hosting of the mappings and DBpedia ontology definition, the DBpedia Mappings Wiki offers various tools which support users in their work:

- **Mapping Validator:** When editing a mapping, the mapping can be directly validated by a button on the edit page. This validates changes before saving them for syntactic correctness and highlights inconsistencies such as missing property definitions.
- **Extraction Tester:** The extraction tester linked on each mapping page tests a mapping against a set of example Wikipedia pages. This gives direct feedback about whether a mapping works and how the resulting data will look like.
- **Mapping Tool:** The DBpedia Mapping Tool is a graphical user interface that supports users to create and edit mappings.



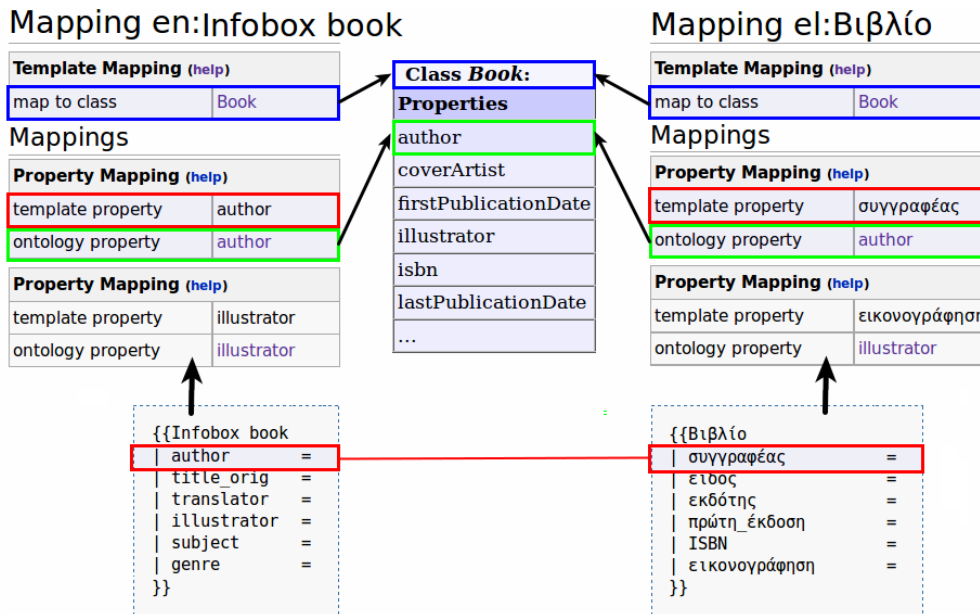


Figure 3.3.: Depiction of the mapping from the Greek and English Wikipedia templates about books to the same DBpedia Ontology class [Kontokostas et al., 2012].

### 3.2.5. URI Schemes

For every Wikipedia article, the framework introduces a number of URIs to represent the concepts described on a particular page. Up to 2011, DBpedia published URIs only under the `http://dbpedia.org` domain. The main namespaces were:

- `http://dbpedia.org/resource/` (prefix *dbpedia*) for representing article data. Apart from a few mapping cases, there is a one-to-one mapping between a Wikipedia page and a DBpedia resource based on the article title. For example, for the Wikipedia article on *Berlin*<sup>4</sup>, DBpedia will produce the URI `dbpedia:Berlin`.
- `http://dbpedia.org/property/` (prefix *dbpprop*) for representing properties extracted from the raw infobox extraction (cf. Section 3.2.3), e.g. `dbpprop:population`.
- `http://dbpedia.org/ontology/` (prefix *dbo*) for representing the DBpedia ontology (cf. Section 3.2.4), e.g. `dbo:populationTotal`.

Although data from other Wikipedia language editions were extracted, the data were extracted under the same namespaces. This was achieved by exploiting the

<sup>4</sup><http://en.wikipedia.org/wiki/Berlin>

Wikipedia *inter-language links*<sup>5</sup>. For every page in a language other than English, the page was extracted only if the page contained an inter-language link to an English page. In that case, using the English link, the data was extracted under the English resource name (i.e. *dbpedia:Berlin*).

Recent DBpedia internationalization developments showed that this approach resulted in less and redundant data [Kontokostas et al., 2012]. Thus with the *DBpedia 3.7 release*, we started to produce two types of datasets. The *localized datasets* contain all things that are described in a specific language. Within the datasets, things are identified with language specific URIs such as `http://<lang>.dbpedia.org/resource/` for article data and `http://<lang>.dbpedia.org/property/` for property data. In addition, we produce a *canonicalized dataset* for each language. The canonicalized datasets only contain things for which a corresponding page in the English edition of Wikipedia exists. Within all canonicalized datasets, the same thing is identified with the same URI from the generic language-agnostic namespace `http://dbpedia.org/resource/`.

#### 3.2.6. Summary of Recent Developments

This section summarizes the improvements of the DBpedia extraction framework since the publication of the DBpedia overview article [Lehmann et al., 2009] in 2009. One of the major changes on the implementation level is that the extraction framework has been rewritten in Scala in 2010 to improve the efficiency of the extractors by an order of magnitude compared to the previous PHP based framework. The new more modular framework also allows to extract data from tables in Wikipedia pages and supports extraction from multiple MediaWiki templates per page. Another significant change was the creation and utilization of the DBpedia Mappings Wiki as described above.

In addition, there were several smaller improvements and general maintenance: Overall, over the past four years, the parsing of the MediaWiki markup improved quite a lot which led to better overall coverage, for example, concerning references and parser functions. In addition, the collection of MediaWiki namespace identifiers for many languages is now performed semi-automatically leading to a high accuracy of detection. This concerns common title prefixes such as *User*, *File*, *Template*, *Help*, *Portal* etc. in English that indicate pages that do not contain encyclopedic content and would produce noise in the data. They are important for specific extractors as well, for instance, the category hierarchy dataset (SKOS) is produced from pages of the *Category* namespace. Furthermore, the output of the extraction system now supports more formats and several compliance issues regarding URIs, IRIs, N-Triples and Turtle were fixed.

The individual data extractors have been improved as well in both number and quality in many areas. The abstract extraction was enhanced producing more accurate plain text representations of the beginning of Wikipedia article texts.

---

<sup>5</sup>[http://en.wikipedia.org/wiki/Help:Interlanguage\\_links](http://en.wikipedia.org/wiki/Help:Interlanguage_links)

More diverse and more specific datatypes do exist (e.g. many currencies and XSD datatypes such as `xsd:gYearMonth`, `xsd:positiveInteger`, etc.) and for a number of classes and properties, specific datatypes were added (e.g. inhabitants/km<sup>2</sup> for the population density of populated places and m<sup>3</sup>/s for the discharge of rivers). Many issues related to data parsers were resolved and the quality of the `owl:sameAs` dataset for multiple language versions was increased by an implementation that takes bijective relations into account.

There are also new extractors such as extractors for Wikipedia page IDs and revisions. Moreover, redirect and disambiguation extractors were introduced and improved since. For the redirect data the transitive closure is computed while taking care of catching cycles in the links. The redirects also help regarding infobox coverage in the mapping-based extraction by resolving alternative template names. Moreover, in the past, if an infobox value pointed to a redirect, this redirection was not properly resolved and thus resulted in RDF links that led to URIs which did not contain any further information. Resolving redirects affected approximately 15% of all links, and hence increased the overall inter-connectivity of resources in the DBpedia ontology substantially.

Finally, a new heuristic to increase the connectivity of DBpedia instances was introduced. If an infobox contains a string value that is not linked to another Wikipedia article, the extraction framework searches for hyperlinks in the same Wikipedia article that have the same anchor text as the infobox value string. If such a link exists, the target of that link is used to replace the string value in the infobox. This method further increases the number of object properties in the DBpedia ontology.

### 3.3. DBpedia Ontology

The DBpedia ontology consists of 320 classes which form a subsumption hierarchy and are described by 1,650 different properties. With a maximal depth of 5, the subsumption hierarchy is intentionally kept rather shallow which fits use cases in which the ontology is visualized or navigated. Figure 3.4 depicts a part of the DBpedia ontology, indicating the relations among the top ten classes of the DBpedia ontology, i.e. the classes with the highest number of instances.

The DBpedia ontology is maintained and extended by the community in the DBpedia Mappings Wiki. Figure 3.5 depicts the growth of the DBpedia ontology over time. While the number of classes is not growing too much due to the already good coverage of the initial version of the ontology, the number of properties increases over time due to the collaboration on the DBpedia Mappings Wiki.

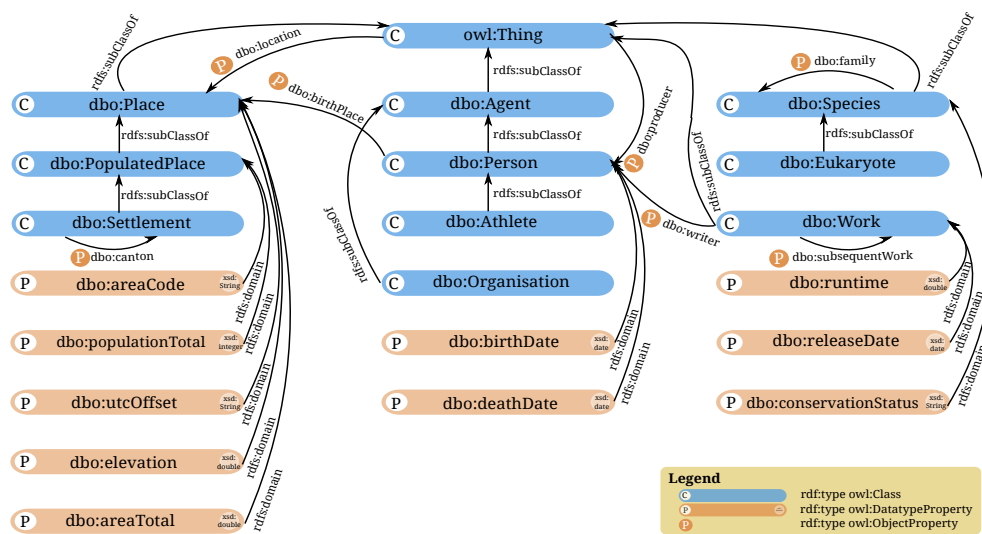


Figure 3.4.: Snapshot of a part of the DBpedia ontology.

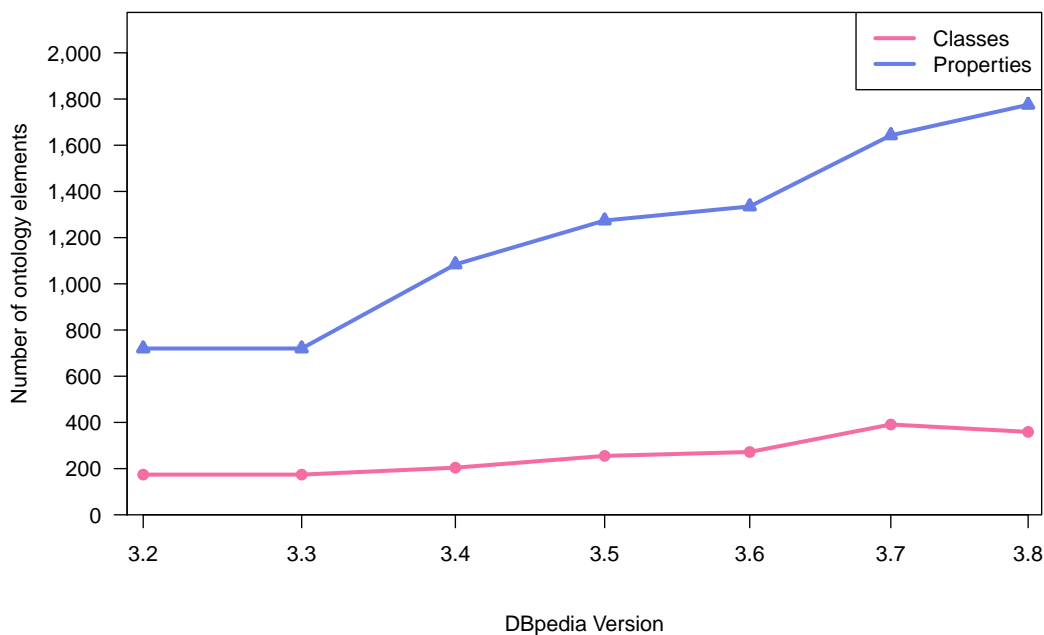


Figure 3.5.: Growth of the DBpedia ontology

```

1 SELECT * { {
2   SELECT ?ftsyear ?ftscountry (SUM(?amount) AS ?funding) {
3     ?com rdf:type fts-o:Commitment .
4     ?com fts-o:year ?year .
5     ?year rdfs:label ?ftsyear .
6     ?com fts-o:benefit ?benefit .
7     ?benefit fts-o:detailAmount ?amount .
8     ?benefit fts-o:beneficiary ?beneficiary .
9     ?beneficiary fts-o:country ?country .
10    ?country owl:sameAs ?ftscountry .
11  } } {
12  SELECT ?dbpcountry ?gdpyear ?gdpnominal {
13    ?dbpcountry rdf:type dbo:Country .
14    ?dbpcountry dbpprop:gdpNominal ?gdpnominal .
15    ?dbpcountry dbpprop:gdpNominalYear ?gdpyear .
16  } }
17  FILTER ((?ftsyear = str(?gdpyear)) &&
18          (?ftscountry = ?dbpcountry)) }

```

Figure 3.6.: SPARQL query to compare funding per year (from FTS) and country with the gross domestic product of that country.

## 3.4. Interlinking

DBpedia is interlinked with numerous external datasets following the Linked Data principles. In this section, we give an overview of the number and types of outgoing links that point from DBpedia into other datasets, as well as the external datasets that set links pointing at DBpedia resources.

### 3.4.1. Outgoing Links

Similar to the DBpedia ontology, DBpedia also follows a community approach for adding links to other third party datasets. The DBpedia project maintains a link repository<sup>6</sup> for which conventions for adding linksets and linkset metadata are defined. The adherence to those guidelines is supervised by a linking committee. Linksets, which are added to the repository are used for the subsequent official DBpedia release as well as for DBpedia-Live. Table 3.2 lists the linksets created by the DBpedia community as of April 2013. The first column names the dataset that is the target of the links. The second and third column contain the predicate that is used for linking as well as the overall number of links that is set between DBpedia and the external dataset. The last column names the tool that was used to generate the links if the tool is known. The value S refers to Silk, L to LIMES, and C to custom script.

Links in DBpedia have been used for various purposes. One example is the combination of data about European Union project funding (FTS) [Martin et al., 2013] and Data about countries in DBpedia. The query shown in Figure 3.6 compares funding per year (from FTS) and country with the gross domestic product of a country (from DBpedia).

<sup>6</sup><https://github.com/dbpedia/dbpedia-links>

Dataset	Predicate	Count	Tool
Amsterdam Museum	owl:sameAs	627	S
BBC Wildlife Finder	owl:sameAs	444	S
Book Mashup	rdf:type	9 100	
	owl:sameAs		
Bricklink	dc:publisher	10 100	
CORDIS	owl:sameAs	314	S
Dailymed	owl:sameAs	894	S
DBLP Bibliography	owl:sameAs	196	S
DBTune	owl:sameAs	838	S
Diseasome	owl:sameAs	2 300	S
Drugbank	owl:sameAs	4 800	S
EUNIS	owl:sameAs	3 100	S
Eurostat (Linked Stats)	owl:sameAs	253	S
Eurostat (WBSG)	owl:sameAs	137	
CIA World Factbook	owl:sameAs	545	S
flickr wrappr	dbpprop:hasPhoto-Collection	3 800 000	C
Freebase	owl:sameAs	3 600 000	C
GADM	owl:sameAs	1 900	
GeoNames	owl:sameAs	86 500	S
GeoSpecies	owl:sameAs	16 000	S
GHO	owl:sameAs	196	L
Project Gutenberg	owl:sameAs	2 500	S
Italian Public Schools	owl:sameAs	5 800	S
LinkedGeoData	owl:sameAs	103 600	S
LinkedMDB	owl:sameAs	13 800	S
MusicBrainz	owl:sameAs	23 000	
New York Times	owl:sameAs	9 700	
OpenCyc	owl:sameAs	27 100	C
OpenEI (Open Energy)	owl:sameAs	678	S
Revyu	owl:sameAs	6	
Sider	owl:sameAs	2 000	S
TCMGeneDIT	owl:sameAs	904	
UMBEL	rdf:type	896 400	
US Census	owl:sameAs	12 600	
WikiCompany	owl:sameAs	8 300	
WordNet	dbpprop:wordnet_type	467 100	
YAGO2	rdf:type	18 100 000	
<b>Sum</b>		<b>27 211 732</b>	

Table 3.2.: Datasets linked from DBpedia

In addition to providing outgoing links on instance-level, DBpedia also sets links on schema-level pointing from the DBpedia ontology to equivalent terms in other schemas. Links to other schemata can be set by the community within the DBpedia Mappings Wiki by using `owl:equivalentClass` in `Class` templates and `owl:equivalentProperty` in `DatatypeProperty` or `ObjectProperty` templates respectively.

In 2011 Google, Microsoft, and Yahoo! announced their collaboration on Schema.org, a collection of vocabularies for marking up content on web pages. The DBpedia 3.8 ontology contains 45 `owl:equivalentClass` and 31 `owl:equivalentProperty` links pointing at <http://schema.org> terms.

### 3.4.2. Incoming Links

DBpedia is being linked to from a variety of datasets. The overall number of links pointing at DBpedia from other datasets is 39,007,478 according to the Data Hub.<sup>7</sup> However, those counts are entered by users and may not always be valid and up-to-date.

In order to identify actually published and online datasets that link to DBpedia, we used Sindice [Oren et al., 2008]. The Sindice project crawls RDF resources on the web and indexes those resources. In Sindice, a dataset is defined by the second-level domain name of the entity's URI, e.g. all resources available at the domain `fu-berlin.de` is considered to belong to the same dataset. A triple is considered to be a link if the dataset of subject and object are different. Furthermore, the Sindice data we used for analysis only considers *authoritative* entities: The dataset of a subject of a triple must match the domain it was retrieved, otherwise it is not considered. Sindice computes a graph summary over all resources they store. With the help of the Sindice team, we examined this graph summary to obtain all links pointing to DBpedia. As shown in Table 3.4, Sindice knows about 248 datasets linking to DBpedia. 70 of those datasets link to DBpedia via `owl:sameAs`, but other link predicates are also very common as evident in this table. In total, Sindice has indexed 8 million links pointing at DBpedia. Table 3.3 lists the 10 datasets which set most links to DBpedia along with the used link predicate and the number of links.

It should be noted that the data in Sindice is not complete, for instance it does not contain all datasets that are cataloged by the DataHub<sup>8</sup>. However, it crawls for RDFa snippets, converts microformats etc. Despite the inaccuracy, the relative comparison of different datasets can still give us some insights. Therefore, we analyzed the link structure of all Sindice datasets using the Sindice cluster (see appendix for details). Table 3.5 shows the datasets with most incoming links. Those are authorities in the network structure of the web of data and DBpedia is currently ranked second.

---

<sup>7</sup>See <http://wiki.dbpedia.org/Interlinking> for details.

<sup>8</sup><http://datahub.io/>

<b>Dataset</b>	<b>Link Predicate Count</b>	<b>Link Count</b>
okaboo.com	4	2,407,121
tfri.gov.tw	57	837,459
naplesplus.us	2	212,460
fu-berlin.de	7	145,322
freebase.com	108	138,619
geonames.org	3	125,734
opencyc.org	3	19,648
geospecies.org	10	16,188
dbrec.net	3	12,856
faviki.com	5	12,768

Table 3.3.: Top 10 datasets in Sindice ordered by the number of links to DBpedia.

<b>Metric</b>	<b>Value</b>
Total links:	3,960,212
Total distinct datasets:	248
Total distinct predicates:	345

Table 3.4.: Sindice summary statistics for incoming links to DBpedia.

<b>domain</b>	<b>datasets</b>	<b>links</b>
purl.org	498	6,717,520
dbpedia.org	248	3,960,212
creativecommons.org	2,483	3,030,910
identi.ca	1,021	2,359,276
l3s.de	34	1,261,487
rkbexplorer.com	24	1,212,416
nytimes.com	27	1,174,941
w3.org	405	658,535
geospecies.org	13	523,709
livejournal.com	14,881	366,025

Table 3.5.: Top 10 datasets by incoming links in Sindice.



## 4. DBpedia Live Extraction

This chapter describes the DBpedia Live extraction framework in detail, how it works, and how it can keep DBpedia in synchronization with Wikipedia. It also details the features of the DBpedia Live system and how important it is for the community. This chapter is based on [Morse et al., 2012b].

### 4.1. Live Extraction Framework

A prerequisite for being able to perform a live extraction is an access to changes made in Wikipedia. The Wikimedia foundation kindly provided us access to their update stream, the *Wikipedia OAI-PMH*<sup>1</sup> live feed. The protocol allows to pull updates in XML via HTTP. A Java component, serving as a proxy, constantly retrieves new updates and feeds the DBpedia framework. The proxy is necessary to decouple the stream from the framework to simplify maintenance of the software. It also handles other maintenance tasks such as the removal of deleted articles and it processes the new templates, which we will introduce in Section 4.2. The live extraction workflow uses this update stream to extract new knowledge upon relevant changes in Wikipedia articles.

#### 4.1.1. General System Architecture

The general system architecture of DBpedia Live is depicted in Figure 4.1. The main components of DBpedia Live system are as follows:

- Local Wikipedia: We have installed a local Wikipedia that will be in synchronization with Wikipedia. The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) [Lagoze et al., 2008] enables an application to get a continuous stream of updates from a wiki. OAI-PMH is also used to feed updates into DBpedia Live Extraction Manager.
- Mapping Wiki: DBpedia mappings can be found at <http://mappings.dbpedia.org>. It is also a wiki. We can also use OAI-PMH to get stream of updates in DBpedia mappings. Basically, a change of mapping affects several Wikipedia pages, which should be reprocessed.

---

<sup>1</sup>Open Archives Initiative Protocol for Metadata Harvesting, cf. <http://www.mediawiki.org/wiki/Extension:OAIRepository>

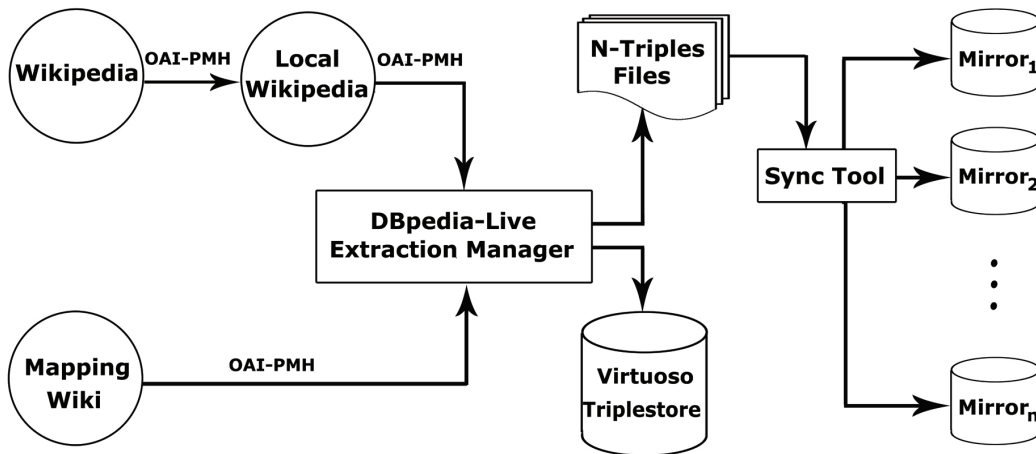


Figure 4.1.: General DBpedia Live system architecture.

- **DBpedia Live Extraction Manager:** This component is the actual DBpedia Live extraction framework. When there is a page that should be processed, the framework applies the extractors on it. After processing a page, the newly extracted triples are inserted into the backend triplestore (Virtuoso), overwriting the old triples. The newly extracted triples are also written as N-Triples file and compressed. Other applications or DBpedia Live mirrors that should always be in synchronization with our DBpedia Live can download those files and feed them into its own triplestore. The extraction manager is discussed in more detail below.
- **Synchronization Tool:** This tool is used for synchronizing other DBpedia Live mirrors with our DBpedia Live.

#### 4.1.2. Extraction Manager

Figure 3.2 gives a detailed overview of the DBpedia knowledge extraction framework and its main components.

DBpedia Live uses the same extraction manager as the core of the extraction process. In live extraction mode, article texts are accessed via the *LiveWikipedia page collection*, which obtains the current version of the article, which was preprocessed by the Java proxy from the *OAI-PMH* stream. The content is comprised of the current wikisource code, language (English only at the moment), an OAI identifier and a page revision id<sup>2</sup>. The *SPARQL-Update Destination* deletes existing triples and inserts new ones into the target triplestore. According to our measurements, about 1.4 article pages are updated each second on Wikipedia. This amounts to 120,000 page updates per day and a maximum processing time of 0.71s per page for the live extraction framework. Currently, the framework can handle up

<sup>2</sup>see here for an example <http://en.wikipedia.org/wiki/Special:Export/Algarve>

to 1.8 pages per second on a 2.8 GHz machine with 6 core CPUs (this includes consumption from the stream, extraction, diffing and loading the triples into a Virtuoso triplestore, and writing the updates into compressed files)<sup>3</sup>. Performance is one of the major engineering hurdles we had to take in order to be able to deploy the framework. The time lag for DBpedia to reflect Wikipedia changes lies between one and two minutes. The bottleneck here is the update stream, since changes normally need more than one minute to arrive from Wikipedia.

Apart from performance, another important problem is to identify which triples have to be deleted and re-extracted upon an article change. DBpedia contains a “static” part, which is not affected by the live extraction framework. This includes links to other knowledge bases, which are manually updated as well as the YAGO<sup>4</sup> and Umbel<sup>5</sup> class hierarchies, which can not be updated via the English Update Stream. We store the structure of those triples using a SPARQL graph pattern. Those static triples are stored in a separate graph. All other parts of DBpedia are maintained by the extraction framework. We redesigned the extractors in such a way that each generates triples with disjoint properties. Each extractor can be in one of three states: active, keep, and purge. Depending on the state when a Wikipedia page is changed, the triples extracted by this extractor are either updated (active), not modified (keep), or removed (purge).

In order to decide which triples were extracted by an extractor, and also to identify the triples that should be replaced we use an RDB (relational database) assisted method, which is described in more detail in [Stadler et al., 2010]. We create an RDB table consisting of 3 fields, namely `page_id`, `resource_uri`, and `serialized_data`. `Page_id` is the unique ID of the Wikipedia page. `Resource_uri` is the URI of DBpedia resource representing that Wikipedia article in DBpedia. `Serialized_data` is the JSON representation of all extracted triples. It is worth noting here that we store the extractor responsible for each group of triples along with those triples in that field. Whenever a Wikipedia page is edited, the extraction method generates a JSON object holding information about each extractor and its generated triples. After serialization of such an object, it will be stored in combination with the corresponding page identifier. In case a record with the same page identifier already exists in this table, this old JSON object and the new one are compared. The results of this comparison are two disjoint sets of triples which are used on the one hand for adding statements to the DBpedia RDF graph and on the other hand for removing statements from this graph.

We had to make a number of further changes within the DBpedia extraction framework in order to support live extraction. For instance, to parse article abstracts properly, we need to be able to resolve templates. This can only be done if (parts of) the MediaWiki database for the corresponding language edition is (are) available. For this reason we delegate updates from the stream to the

---

<sup>3</sup>see current statistics at <http://live.dbpedia.org/livestats>

<sup>4</sup><http://www.mpi-inf.mpg.de/yago-naga/yago/downloads.html>

<sup>5</sup><http://fgiasson.com/blog/index.php/2008/09/04/exploding-dbpedias-domain-using-umbel/>

MediaWiki database so that the local MediaWiki database remains in sync. In general, all parts of the DBpedia framework, which relied on static databases, files etc., needed to be exchanged so that no user interaction is required. Also, the framework had to be refactored to be language independent to make it compatible to future deployment on language specific DBpedia versions such as the Greek or German DBpedia <sup>6</sup>.

The screenshot shows the 'Mapping:Infobox book' page. At the top, there are navigation tabs: 'mapping' (selected), 'discussion', 'edit', 'history', 'delete', 'move', and 'watch'. The main heading is 'Mapping:Infobox book'. Below this, there is a paragraph explaining that this is the mapping for the Wikipedia template 'Infobox\_book' and provides links to find usages, test the mapping, and read more about mapping Wikipedia templates.

Below the text, there are four 'Property Mapping' tables, each with a 'help' link. The first table is titled 'Template Mapping' and maps the 'map to class' template property to the 'Book' ontology property. The second table maps the 'name' template property to the 'foaf:name' ontology property. The third table maps the 'title\_orig' template property to the 'foaf:name' ontology property. The fourth table maps the 'translator' template property to the 'translator' ontology property. The fifth table maps the 'author' template property to the 'author' ontology property.

Figure 4.2.: Mapping for infobox of a book.

## 4.2. New Features

The old php-based framework is deployed on one of OpenLink<sup>7</sup> servers and currently has a SPARQL endpoint at <http://dbpedia-live.openlinksw.com/sparql>. In addition to the migration to Java, the new DBpedia Live framework has the following new features:

1. Abstract extraction: The abstract of of a Wikipedia article are the first few paragraphs of that article. The new framework has the ability to cleanly

---

<sup>6</sup><http://de.dbpedia.org>

<sup>7</sup><http://www.openlinksw.com/>

extract the abstract of an article.

2. Mapping-affected pages: Upon a change in mapping, the pages affected by that mapping should be reprocessed and their triples should be updated to reflect that change.
3. Updating unmodified pages: Sometimes a change in the system occurs, e.g. a change in the implementation of an extractor. This change can affect many pages even if they are not modified. In DBpedia Live, we use a low-priority queue for such changes, such that the updates will eventually appear in DBpedia Live, but recent Wikipedia updates are processed first.
4. Publication of changesets: Upon modifications old triples are replaced with updated triples. Those added and/or deleted triples are also written as N-Triples files and then compressed. Any client application or DBpedia Live mirror can download those files and integrate and, hence, update a local copy of DBpedia. This enables that application to always in synchronization with our DBpedia Live.
5. New extractors: We have added new extractors, which enable extracting more knowledge out of the Wikipedia article.
6. Delta calculation: Via delta, the user can view the latest updates done to a specific DBpedia resource.

In the following subsections, we will describe each feature in detail.

### 4.2.1. Abstract Extraction

The abstract extractor extracts two types of abstracts:

1. Short abstract: is the first paragraph from a Wikipedia article and is represented in DBpedia by `rdfs:comment`.
2. Long abstract: is the whole text before table of contents in an article, which is represented by `dbpedia:abstract`.

The hurdle of abstract extraction is the resolution of templates. A template is a simple sequence of characters that has a special meaning for Wikipedia. Wikipedia renders those templates in a specific way.

The following example indicates a typical Wikipedia template:

#### Example 4.1 Typical Wikipedia Template

```
{{convert|1010000|km2|sp=us}}
```

This templates tells Wikipedia that the area of some country is 1010000 square kilometers, and when it is rendered, Wikipedia should display its area in both square kilometers, and square miles. So, Wikipedia will render it as “1,010,000 square kilometers (390,000 sq mi)”. DBpedia should behave similarly towards those templates.

In order to resolve those templates used in the abstract of the article, we should install a copy of Wikipedia. The required steps to install a local copy of Wikipedia are:

1. MySQL: Install MySQL server as back-end relational database for Wikipedia,
2. SQL dumps: Download the latest SQL dumps for Wikipedia, which are freely available at <http://dumps.wikimedia.org/enwiki/>,
3. Clean SQL dumps: Those SQL dumps need some adjustment, before you can insert them into MySQL, You can perform this adjustment by running “clean.sh”, which you can download from the website containing the sourcecode, see Section 4.2.8.
4. Import SQL dumps: You can now use script called “import.sh”, which is also available with the sourcecode,
5. HTTP Server: Apache server should be installed, which will provide a front-end for abstract extraction.

#### 4.2.2. Mapping-Affected Pages

Whenever a mapping change occurs, some pages should be reprocessed. For example, in Figure 4.2, if the template property called translator, which is mapped to DBpedia property translator, is changed to another property, then all entities belonging to the class Book should be reprocessed. Upon a mapping change, we identify the list of affected DBpedia entities, along with IDs of their corresponding Wikipedia pages.

Basically, the DBpedia Live framework has a priority-queue which contains all pages waiting for processing. This priority-queue is considered the backbone of our framework as several streams including the live-update stream, mapping-update stream, and unmodified-pages stream, place the IDs of their pages in this queue. DBpedia Live consumes the contents of that queue taking the priority of updates into account.

Specifically, IDs of pages fetched from the live update stream are placed in that queue with the highest priority. The IDs of pages affected by a mapping change are also placed in that queue but with lower priority.

### 4.2.3. Unmodified Pages

Naturally, there is a large variation of the update frequency of individual articles in Wikipedia. If any change in the DBpedia extraction framework occurs, e.g. a modification of the implementation of an extractor or an addition of a new extractor, this will not be a problem for the frequently updated articles as it is likely that they will be reprocessed soon.

However, less frequently updated articles may not be processed for several months and would, therefore, not reflect the current implementation state of the extraction framework. To overcome this problem, we obtain the IDs of pages which have not been modified between one and three months ago, and place their IDs in our priority-queue. Those pages have the lowest priority in order not to block or delay live extraction.

Since we use a local synchronized instance of the Wikipedia database, we can query this instance to obtain the list of such articles, which have not been modified between one and three months ago. Directing those queries against Wikipedia itself would place a too high burden on the Wikipedia servers, because the number of unmodified pages can be very large.

### 4.2.4. Changesets

Whenever a Wikipedia article is processed, we get two disjoint sets of triples. A set for added triples, and another set for deleted triples. We write those 2 sets into N-Triples files, compress them, and publish the compressed files on our server. If another triples store wants to synchronize with DBpedia Live, it can just download those files, decompress them and integrate them with its store.

The folder to which we publish the changesets has a specific structure. The folder structure is as follows:

- The parent folder contains a folder for each year while running, e.g. 2010, 2011, 2012, ....
- The folder of a year contains a folder for each month passed while running, e.g. 1, 2, 3, ..., 12.
- The folder of a month contains a folder for each day passed while running, e.g. 1, 2, 3, ..., 28/29/30/31.
- The folder of a day contains a folder for each hour passed while running, e.g. 0, 1, 2, ..., 23.
- Inside the folder of an hour, we write the compressed N-Triples files with added or removed, e.g. 000000.added.nt.gz and 000000.removed.nt.gz. This represents the 2 disjoint sets of added and/or removed triples.

To clarify that structure lets take that example:

### Example 4.2

```
dbpedia_publish/2011/06/02/15/000000.added.nt.gz  
and  
dbpedia_publish/2011/06/02/15/000000.removed.nt.gz
```

This indicates that in year 2011, in 6<sup>th</sup> month of that year, 2<sup>nd</sup> day of that month, in hour 15, 2 files were written, one for added triples, and one for removed triples.

### 4.2.5. Synchronization Tool

The synchronization tool enables a DBpedia Live mirror to stay in synchronization with our live endpoint. It downloads the changeset files sequentially, decompresses them and integrates them with another DBpedia Live mirror. As described in Section 4.2.4, 2 files are written, one for added triples, and the other for deleted ones. That tool, simply downloads both of them, creates the appropriate SPARUL INSERT/DELETE statement and executes it against the triplestore.

An interested user can download this tool and configure it properly, i.e. configure the address of his/her triplestore, login credentials for that triplestore, and so forth. Afterwards, he/she can run it to synchronize that triplestore with ours.

### 4.2.6. New Extractors

Each extractor in the DBpedia framework is responsible for handling a specific part or parts of the Wikipedia article and extracting structured data out of it. So, the more extractors the framework has, the more knowledge it generates. We have created and added two new extractors to the DBpedia framework:

1. Contributor extractor: This extractor basically extracts the contributor of the Wikipedia article, i.e. the editor of the article. This extractor is important as it can track the heavily updated articles, i.e. the articles with high number of edits, as well as the most active contributors, i.e. the editors with high number of edits.
2. Meta-Information extractor: The meta-information extractor extracts the Wikipedia page's meta-information, including edit-link, revision-link, and modification date. The edit-link holds the URL through which the user can edit the Wikipedia article, e.g. <http://en.wikipedia.org/w/index.php?title=Berlin&action=edit>. The revision-link represents the URL of the latest revision of a Wikipedia article, e.g. <http://en.wikipedia.org/w/index.php?title=Berlin&oldid=514878139>. The modification date represents the edit date of a Wikipedia article, e.g. for the Wikipedia article of Berlin the modification date is "2012-09-12T20:52:27Z", which means that this article was edited at that timestamp.



### 4.2.7. Delta Calculation

The delta calculation feature enables the user to view the latest updates done to a specific DBpedia resource. In other words, it displays a list of added, deleted, and modified triples of that resource which were generated during the last iteration of DBpedia Live over that resource. For example, to view the latest updates performed on resource "Leipzig", you should use <http://dbpedia.aksw.org:8899/delta.vsp?uri=http://dbpedia.org/resource/Leipzig>.

### 4.2.8. Important Pointers

- DBpedia project homepage: <http://dbpedia.org>.
- SPARQL-endpoint: The DBpedia Live SPARQL-endpoint can be accessed at <http://live.dbpedia.org/sparql>.
- DBpedia Live statistics: Some simple statistics are provided upon extraction on <http://live.dbpedia.org/livestats>.
- Updates: The N-Triples files containing the updates can be found at <http://live.dbpedia.org/liveupdates>.
- DBpedia sourcecode: <https://github.com/dbpedia/extraction-framework>.
- Synchronization tool: <http://sourceforge.net/projects/dbpintegrator/files/>.
- Dump files: We regularly create dumps of DBpedia Live data on monthly basis, which can be found under <http://live.dbpedia.org/dumps/>.
- Delta: The delta of a DBpedia resource is available at <http://live.dbpedia.org/delta>.
- DBpedia discussion mailing list: For reporting and discussing DBpedia-related issues, users can join our mailing list [https://sourceforge.net/mailarchive/forum.php?forum\\_name=dbpedia-discussion](https://sourceforge.net/mailarchive/forum.php?forum_name=dbpedia-discussion).

## 4.3. DBpedia Live Usage

Since its official release at the end of June 2011, DBpedia Live attracted many users, and an increasing number of requests are sent to our DBpedia Live endpoint every day. Furthermore, more users tend to use the synchronization tool to synchronize their own DBpedia Live mirrors. This leads to an increasing number of live update, i.e. changeset download, requests. 4.3 indicates the number of

daily SPARQL and synchronization requests sent to DBpedia Live endpoint in the period between August 2012 and January 2013.

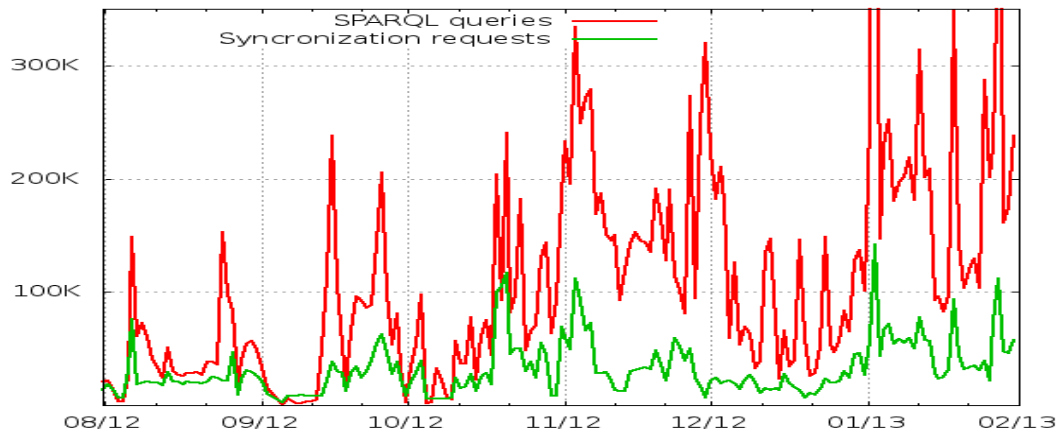


Figure 4.3.: Number of daily requests sent to the DBpedia Live for a) SPARQL queries and b) synchronization requests from August 2012 until January 2013

# 5. Data Quality

This chapter deals with the data quality evaluation and data validation methodologies. It first describes a crowdsourcing methodology for data quality evaluation, mainly centered on DBpedia. It also discusses the various types of errors and issues that can appear in semantic data in general and in DBpedia in particular, and how those errors and issues can be organized in a hierarchy. Afterwards, it introduces a data validation algorithm called "DeFacto", and how its prototype is implemented. This chapter uses material from [Zaveri et al., 2013a] and [Lehmann et al., 2012]. It is worth mentioning that this is a joint work, i.e. I have contributed as a co-author and co-developer in those tasks, in which I was involved in the following tasks:

1. Evaluating several DBpedia resources and identifying the errors that exist in them.
2. Fixing several bugs and enhancing the DBpedia framework, which leads to resolving some of the detected errors.
3. Assessing the evaluations of the participants, who evaluate random DBpedia resources and identify their problems, for correctness.
4. Developing the front-end interface of DeFacto.
5. Building the module that generates the provenance data.

## 5.1. Crowdsourcing for Data Quality

### 5.1.1. Assessment Methodology

In this section, we describe a generalized methodology for the assessment and subsequent data quality improvement of resources belonging to a dataset. The assessment methodology we propose is depicted in Figure 5.1. This methodology consists of the following four steps: 1. Resource selection, 2. Evaluation mode selection, 3. Resource evaluation and 4. Data quality improvement. In the following, we describe these steps in more detail.

**Step I: Resource selection** In this first step, the resources belonging to a particular dataset are selected. This selection can be performed in three different ways:

- *Per Class*: select resources belonging to a particular class
- *Completely random*: a random resource from the dataset
- *Manual*: a resource selected manually from the dataset

Choosing resources per class (e.g. animal, sport, place etc.) gives the user the flexibility to choose resources belonging to only those classes he/she is familiar with. However, when choosing resources from a class, the selection should be made in proportion to the number of instances of that class. Random selection, on the other hand, ensures an unbiased and uniform coverage of the underlying dataset. In the manual selection option, the user is free to select resources with problems that he/she has perhaps previously identified.

**Step II: Evaluation mode selection** The assignment of the resources to a person or machine, selected in Step I, can be accomplished in the following three ways:

- *Manual*: the selected resources are assigned to a person (or group of individuals) who will then proceed to manually evaluate the resources individually.
- *Semi-automatic*: selected resources are assigned to a semi-automatic tool which performs data quality assessment employing some form of user feedback.
- *Automatic*: the selected resources are given as input to an automatic tool which performs the quality assessment without any user involvement.

For the semi-automatic evaluation, machine learning can be applied as shown in [Bühmann and Lehmann, 2012] and provided by the *DL-Learner framework* [Lehmann, 2009, Lehmann and Hitzler, 2010, Lehmann, 2010]. Those algorithms are based on statistical analysis or refinement operators [Lehmann and Hitzler, 2007a, Lehmann, 2007, Lehmann and Haase, 2009] and have been applied to various problems [Lehmann and Hitzler, 2007b, Iglesias and Lehmann, 2011] and shown to be scalable [Hellmann et al., 2009, Hellmann et al., 2011, Lehmann et al., 2011]. The workflow can be as follows: (1) based on the instance data, generate OWL axioms which can also be seen as restrictions<sup>1</sup>, e.g. learn characteristics (irreflexivity, (inverse) functionality, asymmetry) of properties as well as definitions and disjointness of classes in the knowledge base; (2) ask queries via SPARQL or a reasoner for violations of these restrictions, e.g. in case of an irreflexive property, triples where subject and object are the same would indeed violate the characteristic of the irreflexivity. In the automatic case, a possible approach is to check for inconsistencies and other modeling problems as, e.g., described in [Lehmann and Bühmann, 2010].

---

<sup>1</sup>A local Unique Name Assumption is used therefore, i.e. every named individual is assumed to be different from every other, unless stated explicitly otherwise

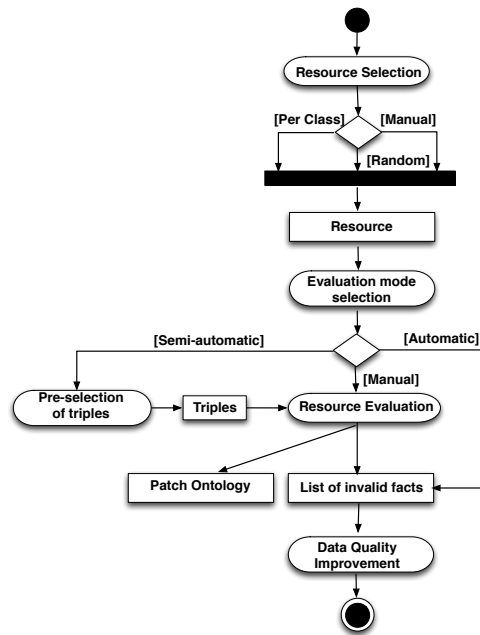


Figure 5.1.: Workflow of the data quality assessment methodology.

**Step III: Resource evaluation** In case of manual assignment of resources, the person (or group of individuals) evaluates each resource individually to detect the potential data quality problems. In order to support this step, a quality assessment tool can be used which allows a user to evaluate each individual triple belonging to a particular resource. If, in case of Step II, the selected resources are assigned to a semi-automatic tool, the tool points to triples likely to be wrong. For example, domain or range problems are identified by the tool and then assigned to a person to verify the correctness of the results.

**Step IV: Data quality improvement** After the evaluation of resources and identification of potential quality problems, the next step is to improve the data quality. There are at least two ways to perform an improvement:

- Direct: editing the triple, identified to contain the problem, with the correct value
- Indirect: using the Patch Request Ontology<sup>2</sup> [Knuth et al., 2012] which allows gathering user feedbacks about erroneous triples.

### 5.1.2. Quality Problem Taxonomy

A systematic review done in [Zaveri et al., 2013c] identified several different data quality dimensions (criteria) applicable to Linked Data. After carrying out

<sup>2</sup><http://141.89.225.43/patchr/ontologies/patchr.ttl#>

an initial data quality assessment on DBpedia (as part of the first phase of the manual assessment methodology cf. Section 5.1.4.1.1), the problems identified were mapped to this list of the identified dimensions. In particular, *Accuracy*, *Relevancy*, *Representational-consistency* and *Interlinking* were identified to be problems affecting a large number of DBpedia resources. Additionally, these dimensions were further divided into categories and sub-categories. Table 5.1 gives an overview of these data quality dimensions along with their categories and sub-categories. Additionally, we indicate whether the problems are automatically detectable (column D) and fixable (column F). If the problem is fixable, we determined whether the problem can be fixed by amending the (i) extraction framework (E), (ii) the mappings wiki (M) or (iii) Wikipedia itself (W). Moreover, the table specifies whether the problems are specific to DBpedia (marked with a ✓) or could potentially occur in any RDF dataset. For example, the sub-category *Special template not properly recognized* is a problem that occurs only in DBpedia due to the presence of specific keywords in Wikipedia articles that do not cite any references or resources (e.g. Unreferenced stub—auto=yes). On the other hand, the problems that are not DBpedia specific can occur in any other datasets. In the following, we provide the quality problem taxonomy and discuss each of the dimensions along with its categories and sub-categories in detail by providing examples.

### 5.1.2.1. Accuracy

Accuracy is defined as *the extent to which data is correct, that is, the degree to which it correctly represents the real world facts and is also free of error* [Zaveri et al., 2013c]. We further classify this dimension into the categories (i) object incorrectly extracted, (ii) datatype problems and (iii) implicit relationship between attributes.

**Object incorrectly extracted.** This category refers to those problems which arise when the object value of a triple is flawed. This may occur when the value is either (i) incorrectly extracted, (ii) incompletely extracted or (iii) the special template in Wikipedia is not recognized:

- *Object value is incorrectly extracted*, e.g.:

```
dbpedia:Oregon_Route_238    dbpprop:map
"238.0"^^http://dbpedia.org/datatype/second.
```

This resource about state highway Oregon Route 238 has the incorrect property 'map' with value 238. In Wikipedia the attribute 'map' refers to the image name as the value: map=Oregon Route 238.svg. The DBpedia property only extracted the value 238 from his attribute value and gave it the datatype 'second' assuming it is a time value, which is incorrect.

- *Object value is incompletely extracted*, e.g.:

```
dbpedia:Dave_Dobbyn    dbpprop:dateOfBirth
```

"3"^^xsd:integer.

In this example, only the day of birth of a person is extracted and mapped to the 'dateofBirth' property when it should have been the entire date i.e. day, month and year. Thus, the object value is not completely extracted.

- *Special template not properly recognized, e.g.:*

dbpedia:328\_Gudrun dbpprop:auto "yes"@en.

Certain article classifications in Wikipedia (such as “This article does not cite any references or sources.”) are performed via special templates (e.g. Unreferenced stub—auto=yes). Such templates should be listed on a black-list and omitted by the DBpedia extraction in order to prevent non-meaningful triples.

**Datatype problems.** This category refers to those triples which are extracted with an incorrect datatype for a typed literal.

- *Datatype incorrectly extracted, e.g.:*

dbpedia:Stephen\_Fry dbo:activeYearsStartYear  
"1981-01-01T00:00:00+02:00"^^xsd:gYear.

In this case, the DBpedia ontology datatype property `activeYearsStartYear` has `xsd:gYear` as range. Although the datatype declaration is correct, it is formatted as `xsd:dateTime`. The expected value is "1981"^^`xsd:gYear`.

**Implicit relationship between attributes.** This category of problems may arise due to (i) representation of one fact in several attributes, (ii) several facts encoded in one attribute or (iii) an attribute value computed from another attribute value in Wikipedia.

- *One fact is encoded in several attributes, e.g.:*

dbpedia:Barlinek dbpprop:postalCodeType "Postal code"@en.

In this example, the value of the postal code of the town of Barlinek is encoded in two attributes ‘postal code type = Postal code’ and ‘postalcode = 74-320’. DBpedia extracts both these attributes separately instead of combining them together to produce one triple, such as: `dbpedia:Barlinek dbpprop:postalCode "74-320"@en`.

- *Several facts are encoded in one attribute, e.g.:*

dbpedia:Picathartes dbo:synonym

"Galgulus Wagler, 1827 (non Brisson, 1760:preoccupied)"@en.

In this example, even though the triple is not incorrect, it contains two pieces of information. Only the first word is the synonym, the rest of the value is a reference to that synonym. In Wikipedia, this fact is represented as “synonyms = “Galgulus” *<small>* Wagler, 1827 (“non” [[Mathurin Jacques

Brisson—Brisson]], 1760: [[Coracias—preoccupied]]/⟨/small⟩””. The DBpedia framework should ideally recognize this and separate these facts into several triples.

- *Attribute value computed from another attribute value, e.g.:*

dbpedia:Barlinek dbpprop:populationDensityKm "auto"@en.

In Wikipedia, this attribute is represented as “population density km2 = auto”. The word “auto” is an indication in Wikipedia that the value associated to that attribute should be computed “automatically”. In this case, the population density is computed automatically by dividing the population by area.

Dimension	Category	Sub-category	D	F	DBpedia specific
Accuracy	Triple incorrectly extracted	Object value is incompletely extracted	–	E	–
		Object value is incompletely extracted	–	E	–
		Special template not properly recognized	✓	E	✓
	Datatype problems	Datatype incorrectly extracted	✓	E	–
	Implicit relationship between attributes	One fact encoded in several attributes	–	M	✓
		Several facts encoded in one attribute	–	E	–
Attribute value computed from another attribute value		–	E + M	✓	
Relevancy	Irrelevant information extracted	Extraction of attributes containing layout information	✓	E	✓
		Redundant attribute values	✓	–	–
		Image related information	✓	E	✓
		Other irrelevant information	✓	E	–
Representation-Consistency	Representation of number values	Inconsistency in representation of number values	✓	W	–
Interlinking	External links	External websites	✓	W	–
	Interlinks with other datasets	Links to Wikimedia	✓	E	–
		Links to Freebase	✓	E	–
		Links to Geospecies	✓	E	–
		Links generated via Flickr wrapper	✓	E	–

Table 5.1.: Data quality dimensions, categories and sub-categories identified in the DBpedia resources. Detectable (column D) means problem detection can be automated. Fixable (column F) means the issue is solvable by amending either the extraction framework (E), the mappings wiki (M) or Wikipedia (W). The last column marks the dataset specific subcategories.

### 5.1.2.2. Relevancy

Relevancy refers to *the provision of information which is in accordance with the task at hand and important to the users’ query* [Zaveri et al., 2013c]. The



only category *Irrelevant information extracted* of this dimension can be further sub-divided into the following sub-categories: (i) extraction of attributes containing layout information, (ii) image related information, (iii) redundant attribute values and (iv) other irrelevant information.

- *Extraction of attributes containing layout information, e.g.:*  
`dbpedia:Lærdalsøyri dbpprop:pushpinLabelPosition`  
`"bottom"@en`. Information related to layout of a page in Wikipedia, such as the position of the label on a pushpin map relative to the pushpin coordinate marker, in this example specified as "bottom", is irrelevant when extracted in DBpedia.
- *Image related information, e.g.:*  
`dbpedia:Three-banded_Plover dbpprop:imageCaption`  
`"At Masai Mara National Reserve, Kenya"@en`. Extraction of an image caption or name of the image is irrelevant in DBpedia as the image is not displayed for any DBpedia resource.
- *Redundant attributes value, e.g.:*  
 The resource `dbpedia:Niedersimmental_District` contains the redundant properties  
`dbo:thumbnail`, `foaf:depiction`,  
`dbpprop:imageMap` with the same value "Karte Bezirk Niedersimmental 2007.png" as the object.
- *Other irrelevant information, e.g.:*  
`dbpedia:IBM_Personal_Computer`  
`dbpedia:Template:Infobox_information_appliance`  
`"type"@en`. Information regarding a templates infobox information, in this case, with an object value as "type" is completely irrelevant.

### 5.1.2.3. Representational-Consistency

Representational-consistency is defined as *the degree to which the format and structure of information conforms to previously returned information and other datasets*. [Zaveri et al., 2013c] and has the following category:

- *Representation of number values, e.g.:*  
`dbpedia:Drei_Flüsse_Stadion dbpprop:seating`  
`Capacity "20"^^xsd:integer`. In Wikipedia, the seating capacity for this stadium has the value "20.000", but in DBpedia the value displayed is only 20. This is because the value is inconsistently represented with a dot after the first two decimal places instead of a comma.

#### 5.1.2.4. Interlinking

Interlinking is defined as *the degree to which entities that represent the same concept are linked to each other* [Zaveri et al., 2013c]. This type of problem is recorded when links to external websites or external data sources are either incorrect, do not show any information or are expired. We further classify this dimension into the following categories:

- *External websites*: Wikipedia usually contains links to external web pages such as, for example, the home page of a company or a music band. It may happen that these links are either incorrect, do not work or are unavailable.
- *Interlinks with other datasets*: Linked Data mandates interlinks between datasets. These links can either be incorrectly mapped or may not contain useful information. These problems are recorded in the following sub-categories: 1. links to Wikimedia, 2. links to Freebase, 3. links to Geospecies, 4. links generated via Flickr wrapper.

#### 5.1.3. A Crowdsourcing Quality Assessment Tool

In order to assist several users in assessing the quality of a resource, a tool has been developed called the *TripleCheckMate* tool<sup>3</sup> aligned with the methodology described in Section 5.1.1, in particular with Steps 1 – 3. To use the tool, the user is required to authenticate himself, which not only prevents Spam but also helps in keeping track of his evaluations. After authenticating himself, he/she proceeds with the selection of a resource (Step 1). He/She is provided with three options: (i) *per class*, (ii) *completely random* and (iii) *manual* (as described in Step I of the assessment methodology).

After selecting a resource, the user is presented with a table showing each triple belonging to that resource on a single row. Step 2 involves the user evaluating each triple and checking whether it contains a data quality problem. The link to the original Wikipedia page for the chosen resource is provided on top of the page which facilitates the user to check against the original values. If the triple contains a problem, he/she checks the box “is wrong”. Moreover, he/she is provided with a taxonomy of pre-defined data quality problems where he/she assigns each incorrect triple to a problem. If the detected problem does not match any of the existing types, he/she has the option to provide a new type and extend the taxonomy. After evaluating one resource, the user saves the evaluation and proceeds to choosing another random resource and follow the same procedure.

Another important feature of the tool is to allow measuring of inter-rater agreements. That is, when a user selects a random method (*Any* or *Class*) to choose a resource, there is a 50% probability that he/she is presented with a resource that was already evaluated by another user. This probability as well

---

<sup>3</sup>available at <http://github.com/AKSW/TripleCheckMate>

as the number of evaluations per resource is configurable. Allowing many users evaluating a single resource not only helps to determine whether incorrect triples are recognized correctly but also to determine incorrect evaluations (e.g. incorrect classification of problem type or marking correct triples as incorrect), especially when crowdsourcing the quality assessment of resources. One important feature of the tool is that although the tool was built for DBpedia, it is parametrizable to accept any endpoint and, with very few adjustments in the database back-end (i.e. ontology classes and problem types) one could use it for any Linked Data dataset (open or closed).

## 5.1.4. Evaluation of DBpedia Data Quality

### 5.1.4.1. Evaluation Methodology

#### 5.1.4.1.1 Manual Methodology

We performed the assessment of the quality of DBpedia in two phases: *Phase I: Problem detection and creation of taxonomy* and *Phase II: Evaluation via crowdsourcing*.

*Phase I: Creation of quality problem taxonomy.* In the first phase, two researchers independently assessed the quality of 20 DBpedia resources each. During this phase an initial list of data quality problems, that occurred in each resource, was identified. These identified problems were mapped to the 26 different quality dimensions from [Zaveri et al., 2013c]. After analyzing the root cause of these problems, a refinement of the quality dimensions was done to obtain a finer classification of the dimensions. This classification of the dimensions into sub-categories resulted in a total of 17 types of data quality problems (cf. Table 5.1) as described in Section 5.1.2.

*Phase II: Crowdsourcing quality assessment.* In the second phase, we crowdsourced the quality evaluation wherein we invited researchers who are familiar with RDF to use the TripleCheckMate tool (described in Section 5.1.3). First, each user after authenticating oneself, chooses a resource by one of three options mentioned in Section 5.1.4.1.1. Thereafter, the extracted facts about that resource are shown to the user. The user then looks at each individual fact and records whether it contains a data quality problem and maps it to the type of quality problem.

#### 5.1.4.1.2 Semi-automatic Methodology

We applied the semi-automatic method (cf. Section 5.1.1), which consists of two steps: (1) the generation of a particular set of schema axioms for all properties in DBpedia and (2) the manual verification of the axioms.

*Step I: Automatic creation of an extended schema.* In this step, the enrichment functionality of DL-Learner [Bühmann and Lehmann, 2012] for SPARQL endpoints was applied. Thereby for all properties in DBpedia, axioms expressing the (inverse) functional, irreflexive and asymmetric characteristic were generated, with

a minimum confidence value of 0.95. For example, for the property `dbo:firstWin`, which is a relation between Formula One racers and grand prix, axioms for all four mentioned types were generated: Each Formula One racer has only one first win in his career (functional), each grand prix can only be won by one Formula One racer (inverse functional). It is not possible to use the property `dbo:firstWin` in both directions (asymmetric), and the property is also irreflexive.

*Step II: Manual evaluation of the generated axioms.* In the second step, we used at most 100 random axioms per axiom type and manually verified whether this axiom is appropriate. To focus on possible data quality problems, we restricted the evaluation data to axioms where at least one violation can be found in the knowledge base. Furthermore, we tried to facilitate the evaluation by taking also the target context into account, i.e. if it exists we consider the definition, domain and range as well as one random sample for a violation. When evaluating the inverse functionality for the property `dbo:firstWin`, we can therefore make use of the following additional information:

```

1 Domain: dbo:FormulaOneRacer Range: dbo:GrandPrix
2 Sample Violation:
3 dbpedia:Fernando_Alonso dbo:firstWin
4   dbpedia:2003_Hungarian_Grand_Prix.
5 dbpedia:WikiProject_Formula_One dbo:firstWin
6   dbpedia:2003_Hungarian_Grand_Prix.
```

### 5.1.4.2. Evaluation Results

#### 5.1.4.2.1 Manual Methodology

An overview of the evaluation results is shown in Table 5.2<sup>4</sup>. Overall, only 16.5% of all resources were not affected by any problems. On average, there were 5.69 problems per resource and 2.24 problems excluding errors in the *dbprop namespace*<sup>5</sup> [Lehmann et al., 2009]. While the vast majority of resources have problems, it should also be remarked that each resource has 47.19 triples on average, which is higher than in most other LOD datasets. About 83.49% of all resources have at least one problem according to our data quality taxonomy. The tool was configured to allow two evaluations per resource and this resulted to a total of 268 inter-evaluations. We computed the inter-rater agreement for those resources, which were evaluated by two persons by adjusting the observed agreement with agreement by chance as done in Cohen’s kappa<sup>6</sup>. The inter-rater agreement results – 0.34 for resource agreement and 0.38 for triple agreement – indicate that the same resource should be evaluated more than twice in future evaluations. To assess the accuracy of the crowdsourcing evaluation, we took a random sample of 700 assessed triples (out of the total 2928) and evaluated them for correctness based on the formula in [Krejcie and Morgan, 1970] intended to be a representative of all the assessed triples. Additionally, we assumed a margin of 3.5% of error, which

<sup>4</sup>Also available at: <http://aksw.org/Projects/DBpediaDQ>

<sup>5</sup><http://dbpedia.org/property/>

<sup>6</sup>[http://en.wikipedia.org/wiki/Cohen%27s\\_kappa](http://en.wikipedia.org/wiki/Cohen%27s_kappa)

Total no. of users	58
Total no. of distinct resources evaluated	521
Total no. of resources evaluated	792
Total no. of distinct resources without problems	86
Total no. of distinct resources with problems	435
Total no. of distinct incorrect triples	2928
Total no. of distinct incorrect triples in the <i>dbprop</i> namespace	1745
Total no. of inter-evaluations	268
No. of resources with evaluators having different opinions	89
Resource-based inter-rater agreement (Cohen's Kappa)	0.34
Triple-based inter-rater agreement (Cohen's Kappa)	0.38
No. of triples evaluated for correctness	700
No. of triples evaluated to be correct	567
No. of triples evaluated incorrectly	133
% of triples correctly evaluated	81
Average no. of problems per resource	5.69
Average no. of problems per resource in the <i>dbprop</i> namespace	3.45
Average no. of triples per resource	47.19
% of triples affected	11.93
% of triples affected in the <i>dbprop</i> namespace	7.11

Table 5.2.: Overview of the manual quality evaluation.

is a bound that we can place on the difference between the estimated correctness of the triples and the true value, and a 95% confidence level, which is the measure of how confident we are in that margin of error<sup>7</sup>. From these 700 triples, 133 were evaluated incorrectly resulting in about 81% of triples correctly evaluated.

Table 5.3 shows the total number of problems, the distinct resources and the percentage of affected triples for each problem type. Overall, the most prevalent problems, such as broken external links are outside the control of the DBpedia extraction framework. After that, several extraction and mapping problems that occur frequently mainly affecting accuracy, can be improved by manually adding mappings or possibly by improving the extraction framework.

When looking at the detectable and fixable problems from Table 5.1, in light of their prevalence, we expect that approximately one third of the problems can be automatically detected and two thirds are fixable by improving the DBpedia extraction framework. In particular, implicitly related attributes can be properly extracted with a new extractor, which can be configured using the DBpedia Mappings Wiki. As a result, we expect that the improvement potential is that the problem rate in DBpedia can be reduced from 11.93% to 5.81% (calculated by subtracting 7.11% from 11.93% reported in Table 5.2). After revising the DBpedia extraction framework, we will perform subsequent quality assessments using the same methodology in order to realize and demonstrate these improvements.

#### 5.1.4.2.2 Semi-automatic Methodology

The evaluation results in Table 5.4 show that for the irreflexive case all 24 properties that would lead to at least one violation should indeed be declared as irreflexive. Applying the irreflexive characteristic would therefore help to

<sup>7</sup><http://research-advisors.com/tools/SampleSize.htm>

Criteria	IT	DR	AT %
<i>Accuracy</i>			
Object incorrectly extracted	32	14	2.69
Object value is incorrectly extracted	259	121	23.22
Object value is incompletely extracted	229	109	20.92
Special template not recognized	14	12	2.30
Datatype problems	7	6	1.15
Datatype incorrectly extracted	356	131	25.14
Implicit relationship between attributes	8	4	0.77
One fact is encoded in several attributes	670	134	25.72
Several facts encoded in one attribute	87	54	10.36
Value computed from another value	14	14	2.69
Accuracy unassigned	31	11	2.11
<i>Relevancy</i>			
Irrelevant information extracted	204	29	5.57
Extraction of layout information	165	97	18.62
Redundant attributes value	198	64	12.28
Image related information	121	60	11.52
Other irrelevant information	110	44	8.45
Relevancy unassigned	1	1	0.19
<i>Representational-consistency</i>			
Representation of number values	29	8	1.54
Representational-consistency unassigned	5	2	0.38
<i>Interlinking</i>			
External websites (URLs)	222	100	19.19
Interlinks with other datasets (URIs)	2	2	0.38
Links to Wikimedia	138	71	13.63
Links to Freebase	99	99	19.00
Links to Geospecies	0	0	0.00
Links generated via Flickr wrapper	135	135	25.91
Interlinking unassigned	3	3	0.58

Table 5.3.: Detected number of problem for each of the defined quality problems. IT = Incorrect triples, DR = Distinct resources, AT = Affected triples.

find overall 236 critical triples, for e.g. `dbpedia:2012_Coppa_Italia_Final` `dbo:followingEvent` `dbpedia:2012_Coppa_Italia_Final`, which is not meaningful as no event is the following event of itself. For asymmetry, we got 81 approved properties, for example, containing `dbo:starring` with domain `Work` and range `Actor`. Compared with this, there are also some properties where asymmetry is not always appropriate, e.g. `dbo:influenced`.

Functionality, i.e. having at most one value of a property, can be applied to 76 properties. During the evaluation, we observed invalid facts such as, for example, two different values `2600.0` and `1630.0` for the density of the moon `Himalia`. We spotted overall 199,480 errors of this type in the knowledge base. As the result of the inverse functionality evaluation, we obtained 13 properties where the object in the triple should only be related to one unique subject, e.g. there should only be one Formula One racer which won a particular grand prix, which is implicit when using the property `dbo:lastWin`.

## 5.2. Fact Validation

Characteristic	#Properties		Correct	#Violations			
	Total	Violated		Min.	Max.	Avg.	Total
Irreflexivity	142	24	24	1	133	9.8	236
Asymmetry	500	144	81	1	628	16.7	1358
Functionality	739	671	76	1	91581	2624.7	199480
Inverse Functionality	52	49	13	8	18236	1685.2	21908

Table 5.4.: Results of the semi-automatic evaluation. The table shows the total number of properties that have been suggested to have the given characteristic by Step I of the semi-automatic methodology, the number of properties that would lead to at least one violation when applying the characteristic, the number of properties where the characteristic is meaningful (manually evaluated) and some metrics for the number of violations.

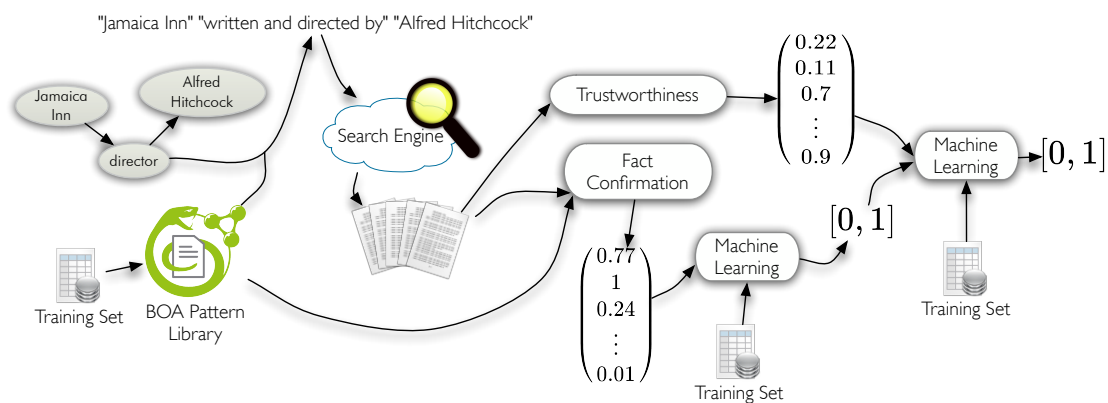


Figure 5.2.: Overview of Deep Fact Validation.

**Input and Output:** The DeFacto system consists of the components depicted in Figure 5.2. The system takes an RDF triple as input and returns a confidence value for this triple as well as possible evidence for the fact. The evidence consists of a set of webpages, textual excerpts from those pages and meta-information on the pages. The text excerpts and the associated meta information allow the user to quickly get an overview over possible credible sources for the input statement: Instead of having to use search engines, browsing several webpages and looking for relevant pieces of information, the user can more efficiently review the presented information. Moreover, the system uses techniques which are adapted specifically for fact validation instead of only having to rely on generic information retrieval techniques of search engines.

**Retrieving Webpages:** The first task of the DeFacto system is to retrieve webpages which are relevant for the given task. The retrieval is carried out by issuing several queries to a regular search engine. These queries are computed by verbalizing the RDF triple using natural-language patterns extracted by the BOA

```
1 dbpedia:Jamaica_Inn_%28film%29 dbo:director
2 dbpedia:Alfred_Hitchcock .
```

Figure 5.3.: Input data for Defacto..

framework<sup>8</sup> [Gerber and Ngonga Ngomo, 2011, Gerber and Ngomo, 2012]. As a next step, the highest ranked webpages for each query are retrieved. Those webpages are candidates for being sources for the input fact. Both the search engine queries as well as the retrieval of webpages are executed in parallel to keep the response time for users within a reasonable limit. Note that usually this does not put a high load on particular web servers as webpages are usually derived from several domains.

**Evaluating Webpages:** Once all webpages have been retrieved, they undergo several further processing steps. First, plain text is extracted from each webpage by removing most HTML markup. In essence, the algorithm decides whether the web page contains a natural language formulation of the input fact. This step distinguishes DeFacto from information retrieval methods. If no webpage confirms a fact according to DeFacto, then the system falls back on light-weight NLP techniques and computes whether the webpage does at least provide useful evidence. These indicators are of central importance because a single trustworthy webpage confirming a fact may be a more useful source than several webpages with low trustworthiness. The fact confirmation and the trustworthiness indicators of the most relevant webpages are presented to the user.

**Confidence Measurement:** In addition to finding and displaying useful sources, DeFacto also outputs a general confidence value for the input fact. This confidence value ranges between 0% and 100% and serves as an indicator for the user: Higher values indicate that the found sources appear to confirm the fact and can be trusted. Low values mean that not much evidence for the fact could be found on the Web and that the websites that do confirm the fact (if such exist) only display low trustworthiness. Naturally, DeFacto is a (semi-)automatic approach: We do assume that users will not blindly trust the system, but additionally analyze the provided evidence.

A prototype implementing the above steps is available at <http://defacto.aksw.org>. It shows relevant webpages, text excerpts and five different rankings per page. The generated provenance output can also be saved directly as RDF. For representing the provenance output, we use the W3C provenance group<sup>9</sup> vocabularies. The source code of both, the DeFacto algorithms and user interface, are openly available<sup>10</sup>.

---

<sup>8</sup><http://boa.aksw.org>

<sup>9</sup><http://www.w3.org/2011/prov/>

<sup>10</sup><https://github.com/AKSW/DeFacto>



It should be noted that we decided not to check for negative evidence of facts in DeFacto, since a) we considered this to be too error-prone and b) negative statements are much less frequent on the web. It is also noteworthy that DeFacto is a self training system on two levels: For each fact, the user can confirm after reviewing the possible sources whether he/she believes it is true. This is then added to the training set and helps to improve the performance of DeFacto. The same can be done for text excerpts of web pages: Users can confirm or reject whether a given text excerpt actually does confirm a fact. More detail of the BOA framework can be found at [Lehmann et al., 2012].

### 5.2.1. Trustworthiness Analysis of Webpages

To determine the trustworthiness of a website we first need to determine its similarity to the input triple. This is determined by how many topics belonging to the query are contained in a search result retrieved by the web search. We extended the approach introduced in [Nakamura et al., 2007] by querying Wikipedia with the subject and object label of the triple in question separately to find the topic terms for the triple. A frequency analysis is applied on all returned documents and all terms above a certain threshold that are not contained in a self-compiled stop word list are considered to be topic terms for a triple. Let  $s$  and  $o$  be the URIs for the subject and object of the triple in question and  $t$  be a potential topic term extracted from a Wikipedia page. In addition, let  $X = (s, p, o)$ . We compare the values of the following two formulas:

$$p(t|X) = \frac{|topic(t, d(X))|}{|d(X)|},$$

$$p(t|intitle(d(X), s \vee o)) = \frac{|topic(t, intitle(d(X), s) \cup intitle(d(X), o))|}{|intitle(d(X), s) \cup intitle(d(X), o)|}.$$

where  $d(X)$  is the set all web documents retrieved for  $X$ ,  $intitle(d(X), x)$  the set of web documents which have the label of the URI  $x$  in their page title.  $topic(t, d(X))$  is the set of documents which contain  $t$  in the page body. We consider  $t$  to be a topic term for the input triple if  $p(t|intitle(d(X), s) \vee intitle(d(X), o)) > p(t|X)$ . Let  $\mathcal{T}_X = \{t_1, t_2, \dots, t_n\}$  be the set of all topic terms extracted for a input triple. Defacto then calculates the trustworthiness of a webpage as follows:

**Topic Majority in the Web** represents the number of webpages that have similar topics to the webpage in question. Let  $\mathcal{P}$  be the set of topic terms appearing on the current webpage. The Topic Majority in the Web for a webpage  $w$  is then calculated as:

$$tm_{web}(w) = \left| \bigcup_{i=1}^n topic(t_i, d(X)) \right| - 1$$

where  $t_1$  is the most occurring topic term in the webpage  $w$ . Note that we subtract 1 to prevent counting  $w$ .

**Topic Majority in Search Results** calculates the similarity of a given webpage for all webpages found for a given triple. Let  $w_k$  be the webpage to be evaluated,  $v(w_k)$  be the feature vector of webpage  $w_k$  where  $v(w_k)_i$  is 1 if  $t_i$  is a topic term of webpage  $w_k$  and 0 otherwise,  $\|v\|$  be the norm of  $v$  and  $\theta$  a similarity threshold. We calculate the Topic Majority for the search results as follows:

$$tm_{search}(w) = \left| \left\{ w_i | w_i \in d(X), \frac{v(w_k) \times v(w_i)}{\|v(w_k)\| \|v(w_i)\|} > \theta \right\} \right|$$

**Topic Coverage** measures the ratio between all topic terms for  $X$  and all topic terms occurring in  $w$ :

$$tc(w) = \frac{|\mathcal{T}_X \cap \mathcal{P}|}{|\mathcal{T}_X|}$$

**Pagerank:** The Pagerank<sup>11</sup> of a webpage is a measure for the relative importance of a webpage compared to all others, i.e. higher pagerank means that a webpage is more popular. There is a positive correlation between popularity of a webpage and its trustworthiness as those pages are more likely to be reviewed by more people or may have gone under stricter quality assurance before their publication. While a high pagerank alone is certainly not a sufficient indicator for trustworthiness, we use it in combination with the above criteria in DeFacto.

### 5.2.2. Features for Deep Fact Validation

In order to obtain an estimate of the confidence that there is sufficient evidence to consider the input triple to be true, we decided to train a supervised machine learning algorithm. Similar to the above presented classifier for fact confirmation, this classifier also requires computing a set of relevant features for the given task. In the following, we describe those features and why we selected them.

First, we extend the score of single proofs to a score of web pages as follows: When interpreting the score of a proof as the probability that a proof actually confirms the input fact, then we can compute the probability that at least one of the proofs confirms the fact. This leads to the following stochastic formula<sup>12</sup>, which allows us to obtain an overall score for proofs  $scw$  on a webpage  $w$ :

$$scw(w) = 1 - \prod_{pr \in prw(w)} (1 - fc(pr))$$

<sup>11</sup><http://en.wikipedia.org/wiki/Pagerank>

<sup>12</sup>To be exact, it is the complementary even to the case that none of the proofs do actually confirm a fact.

In this formula,  $fc$  (fact confirmation) is the classifier trained in [Lehmann et al., 2012], which takes a proof  $pr$  as input and returns a value between 0 and 1.  $prw$  is a function taking a webpage as input and returning all possible proofs contained in it.

**Combination of Trustworthiness and Textual Evidence** In general, the trustworthiness of a webpage and the textual evidence we find in it, are orthogonal features. Naturally, webpages with high trustworthiness and a high score for its proofs should increase our confidence in the input fact. Therefore, it makes sense to combine trustworthiness and textual evidence as features for the underlying machine learning algorithm. We do this by multiplying both criteria and then using their sum and maximum as two different features:

$$F_{fsum}(t) = \sum_{w \in s(t)} (f(w) \cdot scw(w)) \quad F_{fmax}(t) = \max_{w \in s(t)} (f(w) \cdot scw(w))$$

In this formula  $f$  can be instantiated by all four trustworthiness measures: topic majority on the the web ( $tm_{web}$ ), topic majority in search results ( $tm_{search}$ ), topic coverage ( $tc$ ) and pagerank ( $pr$ ).  $s$  is a function taking a triple  $t$  as argument, executing the search queries explained in [Lehmann et al., 2012] and returning a set of webpages. Using the formula, we obtain 8 different features for our classifier, which combine textual evidence and different trustworthiness measures.

**Other Features** In addition to the above described combinations of trustworthiness and fact confirmation, we also defined other features:

1. The total number of proofs found.
2. The total number of proofs found above a relevance threshold of 0.5. In some cases, a high number of proofs with low scores is generated, so the number of high scoring proofs may be a relevant feature for learning algorithms.
3. The total evidence score: This is the probability that at least one of the proofs is correct, which is defined analogously to  $scw$  above:

$$1 - \prod_{pr \in prt(t)} (1 - sc(pr))$$

4. The total evidence score above a relevance threshold of 0.5. This is an adaption of the above formula, which considers only proofs with a confidence higher than 0.5.
5. Total hit count: Search engines usually estimate the number of search results for an input query. The total hit count is the sum of the estimated number of search results for each query send by DeFacto for a given input triple.

6. A domain and range verification: If the subject of the input triple is not an instance of the domain of the property of the input triple, this violates the underlying schema, which should result in a lower confidence in the correctness of the triple. This feature is 0 if both domain and range are violated, 0.5 if exactly one of them is violated and 1 if there is no domain or range violation.

### 5.2.3. Evaluation

Our main objective in the evaluation was to find out whether DeFacto can effectively distinguish between true and false input facts. In the following, we describe how we trained DeFacto using DBpedia, which experiments we used and then discuss the results of those experiments.

#### 5.2.3.1. Training DeFacto

We focus our tests on the top 60 most frequently used properties in DBpedia. The system can easily be extended to cover more properties by extending the training set of BOA to those properties. Note that DeFacto itself is also not limited to DBpedia, i.e. while all of its components are trained on DBpedia, the algorithms can be applied to arbitrary URIs. A performance evaluation on other knowledge bases is subject to future work, but it should be noted that most parts of DeFacto – except the LOD background feature described in Section 5.2 and the schema checking feature in Section 5.2.2 work only with the retrieved labels of URIs and, therefore, do not depend on DBpedia.

For training a supervised machine learning approach, positive and negative examples are required. Those were generated as follows:

**Positive Examples:** In general, we use facts contained in DBpedia as positive examples. For each of the properties we consider, we generate positive examples by randomly selecting triples containing the property. Technically, this is done by counting the frequency of the property and sending a corresponding SPARQL query with random offset to the DBpedia Live endpoint. We obtain 600 statements this way and verified them. For each statement, we manually evaluated whether it was indeed a true fact. It turned out that some of the obtained triples were incorrectly modeled, e.g. obviously violated domain and range restrictions, or could not be confirmed by an intensive search on the web within ten minutes. Overall, 473 out of 600 checked triples were facts, which we subsequently used as positive examples.

**Negative Examples:** The generation of negative examples is more involved than the generation of positive examples. In order to effectively train DeFacto, we considered it essential that many of the negative examples are similar to

true statements. In particular, most statements should be meaningful subject-predicate-object phrases. For this reason, we derive the negative examples from positive examples by modifying them, but following domain and range restrictions. Assume the input triple  $(s, p, o)$  in a knowledge base  $\kappa$  is given and let  $dom$  and  $ran$  be functions returning the domain and range of a property<sup>13</sup>. We used the following methods to generate the negative example sets dubbed *domain*, *range*, *domain-range*, *property*, *random*, *20%mix* (in that order):

1. A triple  $(s', p, o)$  is generated where  $s'$  is an instance of  $dom(p)$ , the triple  $(s', p, o)$  is not contained in  $\kappa$  and  $s'$  is randomly selected from all resources which satisfy the previous requirements.
2. A triple  $(s, p, o')$  is generated analogously by taking  $ran(p)$  into account.
3. A triple  $(s', p, o')$  is generated analogously by taking both  $dom(p)$  and  $ran(p)$  into account.
4. A triple  $(s, p', o)$  is generated in which  $p'$  is randomly selected from our previously defined list of 60 properties and  $(s, p', o)$  is not contained in  $\kappa$ .
5. A triple  $(s', p', o')$  is generated where  $s'$  and  $o'$  are randomly selected resources,  $p'$  is a randomly selected property from our defined list of 60 properties and  $(s', p', o')$  is not contained in  $\kappa$ .
6. 20% of each of the above created negative training sets were randomly selected to create a heterogenous test set.

Note that all parts of the example generation procedure can also take implicit knowledge into account. Since we used SPARQL as query language for implementing the procedure, this is straightforward by using SPARQL 1.1 entailment<sup>14</sup>. In case of DBpedia Live we did not do this for performance reasons and because it would not alter the results in that specific case.

Obviously, it is possible that our procedure of generating negative examples may also generate true statements, which just happen not to be contained in DBpedia. Similar to the analysis of the positive examples, we checked a sample of the negative examples on whether they are indeed false statements. This was the case for all examples in the sample. Overall, we obtained an automatically created and manually cleaned training set, which we made publicly available<sup>15</sup>.

### 5.2.3.2. Experimental Setup

In a first step, we computed all feature vectors, described in Section 5.2.2 for the training set. DeFacto heavily relies on web requests, which are not deterministic,

<sup>13</sup>Technically, we used the most specific class, which was explicitly stated to be domain and range of a property, respectively.

<sup>14</sup><http://www.w3.org/TR/sparql11-entailment/>

<sup>15</sup><http://aksw.org/projects/DeFacto>

	P	R	Domain F <sub>1</sub>	AUC	RSME	P	R	Range F <sub>1</sub>	AUC	RMSE
Logistic Regression	0.799	0.753	0.743	0.83	0.4151	0.881	0.86	0.859	0.844	0.3454
Naive Bayes	0.739	0.606	0.542	0.64	0.6255	0.795	0.662	0.619	0.741	0.5815
SVM	0.811	0.788	0.784	0.788	0.4609	0.884	0.867	<b>0.865</b>	0.866	0.3409
J48	0.835	0.827	<b>0.826</b>	0.819	0.3719	0.869	0.862	0.861	0.908	0.3194
RBF Network	0.743	0.631	0.583	0.652	0.469	0.784	0.683	0.652	0.75	0.4421

Table 5.5.: Classification results for trainings sets *domain* and *range*.

	P	R	Domain-Range (F <sub>1</sub> )	AUC	RSME	P	R	Property F <sub>1</sub>	AUC	RMSE
Logistic Regression	0.871	0.85	0.848	0.86	0.3495	0.822	0.818	0.818	0.838	0.3792
Naive Bayes	0.813	0.735	0.717	0.785	0.5151	0.697	0.582	0.511	0.76	0.6431
SVM	0.88	0.863	0.861	0.855	0.3434	0.819	0.816	0.816	0.825	0.3813
J48	0.884	0.871	<b>0.87</b>	0.901	0.3197	0.834	0.832	<b>0.832</b>	0.828	0.3753
RBF Network	0.745	0.687	0.667	0.728	0.4401	0.72	0.697	0.688	0.731	0.4545

Table 5.6.: Classification results for trainings sets *domain-range* and *property*.

i.e. the same search engine query does not always return the same result. To achieve deterministic behavior and to increase performance and reduce load on the servers, all web requests are cached. The DeFacto runtime for an input triple was on average slightly below 5 seconds per input triple<sup>16</sup> when using caches.

We stored the features in the arff file format and employed the Weka machine learning toolkit<sup>17</sup> for training different classifiers. In particular, we are interested in classifiers, which can handle numeric values and output confidence values. Naturally, confidence values for facts such as, e.g. 95%, are more useful for end users than just a binary response on whether DeFacto considers the input triple to be true, since they allow a more fine-grained assessment. Again, we selected popular machine learning algorithms satisfying those requirements.

We performed 10 fold cross validations for our experiments. In each experiment, we used our created positive examples, but varied the negative example sets described above to see how changes influence the overall behavior of DeFacto.

### 5.2.3.3. Results and Discussion

The results of our experiments are shown in Tables 2-4. Three algorithms – J48, logistic regression and support vector machines – show promising results. Given

<sup>16</sup>The performance is roughly equal on server machines and notebooks, since the web requests dominate.

<sup>17</sup><http://www.cs.waikato.ac.nz/ml/weka/>

	P	R	Random F <sub>1</sub>	AUC	RMSE	P	R	20% Mix F <sub>1</sub>	AUC	RMSE
Logistic Regression	0.855	0.854	0.854	0.908	0.3417	0.665	0.645	0.634	0.785	0.4516
Naive Bayes	0.735	0.606	0.544	0.853	0.5565	0.719	0.6	0.538	0.658	0.6267
SVM	0.855	0.854	0.854	0.906	0.3462	0.734	0.729	0.728	0.768	0.4524
J48	0.876	0.876	<b>0.876</b>	0.904	0.3226	0.8	0.79	<b>0.788</b>	0.782	0.405
RBF Network	0.746	0.743	0.742	0.819	0.4156	0.698	0.61	0.561	0.652	0.4788

Table 5.7.: Classification results for trainings sets *random* and *20%mix*.

the challenging tasks, F-measures up to 78.8% for the combined negative example set appear to be very positive indicators that DeFacto can be used to effectively distinguish between true and false statements, which was our primary evaluation objective. In general, DeFacto also appears to be quite stable against the various negative example sets: The algorithms with overall positive results also seem less affected by the different variations. As expected, the easiest task is to distinguish statements with random subject and object as well as random statements and true statements, whereas all other test sets are similarly difficult.

When observing single runs of DeFacto manually, it turned out that our method of generating positive examples is particularly challenging for DeFacto: For many of the facts in DBpedia only few sources exist in the Web. While it is widely acknowledged that the amount of unstructured textual information in the Web by far surpasses the available structured data, we found out that a significant amount of statements in DBpedia is difficult to track back to reliable external sources on the Web even with an exhaustive manual search. There are many reasons for this, for instance many facts are particular relevant for a specific country, such as “Person x studied at University y.”, where x is a son of a local politician and y is a country with only limited internet access compared to first world countries. For this reason, only in some cases, BOA patterns could be found: In 29 of the 527 proofs of positive examples, BOA patterns could directly be found. This number increased to 195 out of 527 when employing the WordNet expansion described in [Lehmann et al., 2012]. In general, DeFacto performs better when the subject and object of the input triple are popular on the web, i.e. there are several webpages describing them. In this aspect, we believe our training set is indeed challenging upon manual observation.

## 6. DBpedia SPARQL Benchmark

This chapter deals with the benchmarking in general, and why it is important. It details the DBPSB (DBpedia SPARQL Benchmark), its architecture, and its characteristics. Furthermore, it explains the phases of the benchmarking process, including the dataset generation, query log analysis, query clustering, query selection, and eventually the actual benchmarking process. This chapter is based on papers [Morse et al., 2011] and [Morse et al., 2012a].

### 6.1. Overview on Benchmarking

A benchmark is the process of running a computer program, or a set of programs, in order to measure and/or compare the performance of a systems(s), normally by running a number of standard tests and trials against them. The term 'benchmark' is used to refer to the benchmarking programs themselves as well. Benchmarking is usually associated with assessing performance characteristics of computer hardware, for example, the floating point operation performance of a CPU, but there are circumstances when the technique is also applicable to software. Software benchmarks are, for example, run against database management systems [Wikipedia, 2012].

There are several benchmarks for database management systems (DBMSs), but the most valuable and important one is Transaction Processing Performance Council (TPC) [TPC, 2012]. The main advantage of database benchmarks over triplestore benchmarks is that they are advanced enough to measure the performances of different DBMSs efficiently. Triplestore benchmarks still need to go through a long way to stabilize and mature. The core concern of triplestore benchmarks is to cover and test as many SPARQL features as possible.

In this thesis, we propose a generic SPARQL benchmark creation methodology. This methodology is based on a flexible data generation mimicking an input data source, query log mining, clustering and SPARQL feature analysis. We apply the proposed methodology to datasets of various sizes derived from the DBpedia knowledge base. In contrast to previous benchmarks, we perform measurements on *real* queries that were issued by humans or Data-Web applications against existing RDF data. Moreover, we do not only consider the query string but also the SPARQL feature(s) used in each of the queries.



### 6.1.1. Objectives of Benchmarking

The benchmarks aim to simulate the real workload of a system in order to assess the performance of that system under these conditions. Basically, the main objectives of benchmarks are the following:

1. assessment and/or comparison of the relative performances of the systems under test;
2. identification of the strong and weak points of each system, which in turn helps in selecting the most appropriate system for a specific purpose;
3. identification of the best and worst operating conditions of each system;
4. assistance for the system developers in order to enhance their systems;
5. assistance for building more advanced benchmarks.

In the following paragraphs we will discuss each one of those objectives in more detail.

**Assessment and/or comparison of performances of the systems:** the benchmark imposes workload on the system under test, monitors its behavior against that workload, and then measures how long it takes to handle it. That workload should mimic the real workload under which the system works as much as possible, e.g. number of queries posed per second, number of concurrent users accessing system simultaneously. When the benchmark measures the performances of several systems, then it can highlight the efficiency of each system which opens the way for improvements.

**Identification of the strong and weak points of each system:** upon running the benchmark, each system can discover its areas of strength and weakness, for instance in database benchmarks a system can be fast in running certain SQL operation, e.g. "LIKE" SQL operation, whereas it is slow for another operation, e.g. "REGEXP". This narrows down the scope of research and investigation required to be conducted by the system developers, in order improve their system.

**Identification of the best and worst operating conditions of each system:** the benchmark also assists the system maintainers to determine the most suitable running conditions of their system. This assistance has a great advantage and importance because it helps the developers to create the required system manuals, which contain the recommended conditions of their system, e.g. the recommended operating system, the recommended memory assignment, and the recommended hard drive speed.

**Assistance for the system developers to enhance their systems:** after running the benchmark on the system(s) under test, it should generate a detailed performance report for each system. This performance report should include a set of concrete items describing the environment under which the system was running and analysis of its performance. The environment parameters include the operating system, the speed of hard drives, etc. The performance parameters include the running time of the system, the queries it was able to answer and the time it needed to respond, the queries it was unable to respond. The system developers and maintainers can benefit from the report in identifying the best and worst operating conditions under which their system can work, e.g. the best underlying operating system. They can also identify the operations and capabilities that should be enhanced in their system.

**Assistance for building more advanced benchmarks:** building a benchmark can also help in building more advanced and complicated benchmarks in the future. A specific benchmark may stress and concentrate on measuring some performance aspects, e.g. the query response time, whereas it ignores some other aspects, e.g. number of concurrent users trying to access the system at the same time. A careful and extensive study of a benchmark can help in discovering the drawbacks of the benchmark itself in order to develop more powerful and complicated benchmarks. As long as the systems under test advance and improve, we need more complicated and better benchmarks to evaluate their relative performances.

## 6.2. Dataset Generation

A crucial step in each benchmark is the generation of suitable datasets. Although we describe the dataset generation here with the example of DBpedia, the methodology we pursue is dataset-agnostic.

The data generation for DBPSB is guided by the following requirements:

- The DBPSB data should resemble the original data (i.e., DBpedia data in our case) as much as possible, in particular the large number of classes, properties, the heterogeneous property value spaces as well as the large taxonomic structures of the category system should be preserved,
- The data generation process should allow to generate knowledge bases of various sizes ranging from a few million to several hundred million or even billion triples,
- Basic network characteristics of different sizes of the network should be similar, in particular the in- and outdegree,
- The data generation process should be easily repeatable with new versions of the considered dataset.

Dataset	Indegree w/ literals	Outdegree w/ literals	Indegree w/o literals	Outdegree w/o literals	No. of nodes	No. of triples
Full DBpedia	5.45	30.52	3.09	15.57	27 665 352	153 737 776
10% dataset (seed)	6.54	45.53	3.98	23.05	2 090 714	15 267 418
10% dataset (rand)	3.82	6.76	2.04	3.41	5 260 753	16 739 055
50% dataset (seed)	6.79	38.08	3.82	18.64	11 317 362	74 889 154
50% dataset (rand)	7.09	26.79	3.33	10.73	9 581 470	78 336 781

Table 6.1.: Statistical analysis of DBPSB datasets.

The proposed dataset creation process starts with an input dataset. For the case of DBpedia, it consists of the datasets loaded into the official SPARQL endpoint<sup>1</sup>. Datasets of multiple size of the original data are created by duplicating all triples and changing their namespaces. This procedure can be applied for any scale factors. While simple, this procedure is efficient to execute and fulfills the above requirements.

For generating smaller datasets, we investigated two different methods. The first method (called “rand”) consists of selecting an appropriate fraction of all triples of the original dataset randomly. If RDF graphs are considered as small world graphs, removing edges in such graphs should preserve the properties of the original graph. The second method (called “seed”) is based on the assumption that a representative set of resources can be obtained by sampling across classes in the dataset. Let  $x$  be the desired scale factor in percent, e.g.  $x = 10$ . The method first selects  $x\%$  of the classes in the dataset. For each selected class, 10% of its instances are retrieved and added to a queue. For each element of the queue, its concise bound description (CBD) [Stickler, 2005] is retrieved. This can lead to new resources, which are appended at the end of the queue. This process is iterated until the target dataset size, measured in number of triples, is reached.

Since the selection of the appropriate method for generating small datasets is an important issue, we performed a statistical analysis on the generated datasets for DBpedia. The statistical parameters used to judge the datasets are the average indegree, the average outdegree, and the number of nodes, i.e. number of distinct IRIs in the graph. We calculated both the in- and the outdegree for datasets once with literals ignored, and another time with literals taken into consideration, as it gives more insight on the degree of similarity between the dataset of interest and the full DBpedia dataset. The statistics of those datasets are given in Table 6.1. According to this analysis, the seed method fits our purpose of maintaining basic network characteristics better, as the average in- and outdegree of nodes are closer to the original dataset. For this reason, we selected this method for generating the DBPSB.

<sup>1</sup>Endpoint: <http://dbpedia.org/sparql>, Loaded datasets: <http://wiki.dbpedia.org/DatasetsLoaded>

### 6.3. Query Analysis and Clustering

The goal of the query analysis and clustering is to detect prototypical queries that were sent to the official DBpedia SPARQL endpoint based on a query-similarity graph. Note that two types of similarity measures can be used on queries, i. e. string similarities and graph similarities. Yet, since graph similarities are very time-consuming and do not bear the specific mathematical characteristics necessary to compute similarity scores efficiently, we picked string similarities for our experiments. In the query analysis and clustering step, we follow a four-step approach. First, we select queries that were executed frequently on the input data source. Second, we strip common syntactic constructs (e.g., namespace prefix definitions) from these query strings in order to increase the conciseness of the query strings. Then, we compute a query similarity graph from the stripped queries. Finally, we use a soft graph clustering algorithm for computing clusters on this graph. These clusters are subsequently used to devise the query generation patterns used in the benchmark. In the following, we describe each of the four steps in more detail.

**Query Selection** For the DBPSB, we use the DBpedia SPARQL query log which contains all queries posed to the official DBpedia SPARQL endpoint for a three-month period in 2010<sup>2</sup>. For the generation of the current benchmark, we used the log for the period from April to July 2010. Overall, 31.5 million queries were posed to the endpoint within this period. In order to obtain a small number of distinctive queries for benchmarking triplestores, we reduce those queries in the following two ways:

- *Query variations.* Often, the same or slight variations of the same query are posed to the endpoint frequently. A particular cause of this is the renaming of query variables. We solve this issue by renaming all query variables in a consecutive sequence as they appear in the query, i.e., *var0*, *var1*, *var2*, and so on. As a result, distinguishing query constructs such as REGEX or DISTINCT are a higher influence on the clustering,
- *Query frequency.* We discard queries with a low frequency (below 10) because they do not contribute much to the overall query performance.

The application of both methods to the query log data set at hand reduced the number of queries from 31.5 million to just 35,965. This reduction allows our benchmark to capture the essence of the queries posed to DBpedia within the timespan covered by the query log and reduces the runtime of the subsequent steps substantially.

---

<sup>2</sup>The DBpedia SPARQL endpoint is available at: <http://dbpedia.org/sparql/> and the query log excerpt at: <ftp://download.openlinksw.com/support/dbpedia/>.

**String Stripping** Every SPARQL query contains substrings that segment it into different clauses. Although these strings are essential during the evaluation of the query, they are a major source of noise when computing query similarity, as they boost the similarity score without the query patterns being similar per se. Therefore, we remove all SPARQL syntax keywords such as `PREFIX`, `SELECT`, `FROM` and `WHERE`. In addition, common prefixes (such as `http://www.w3.org/2000/01/rdf-schema#` for RDF-Schema) are removed as they appear in most queries.

**Similarity Computation** The goal of the third step is to compute the similarity of the stripped queries. Computing the Cartesian product of the queries would lead to a quadratic runtime, i.e., almost 1.3 billion similarity computations. To reduce the runtime of the benchmark compilation, we use the LIMES framework [Ngonga Ngomo and Auer, 2011]<sup>3</sup>. The LIMES approach makes use of the interchangeability of similarities and distances. It presupposes a metric space in which the queries are expressed as single points. Instead of aiming to find all pairs of queries such that  $sim(q, p) \geq \theta$ , LIMES aims to find all pairs of queries such that  $d(q, p) \leq \tau$ , where  $sim$  is a similarity measure and  $d$  is the corresponding metric. To achieve this goal, when given a set of  $n$  queries, it first computes  $\sqrt{n}$  so-called *exemplars*, which are prototypical points in the affine space that subdivide it into regions of high heterogeneity. Then, each query is mapped to the exemplar it is least distant to. The characteristics of metrics spaces (especially the triangle inequality) ensures that the distances from each query  $q$  to any other query  $p$  obeys the following inequality

$$d(q, e) - d(e, p) \leq d(q, p) \leq d(q, e) + d(e, p), \quad (6.1)$$

where  $e$  is an exemplar and  $d$  is a metric. Consequently,

$$d(q, e) - d(e, p) > \tau \Rightarrow d(q, p) > \tau. \quad (6.2)$$

Given that  $d(q, e)$  is constant,  $q$  must only be compared to the elements of the list of queries mapped to  $e$  that fulfill the inequality above. By these means, the number of similarity computation can be reduced significantly. In this particular use case, we cut down the number of computations to only 16.6% of the Cartesian product without any loss in recall. For the current version of the benchmark, we used the *Levenshtein* string similarity measure and a threshold of 0.9.

**Clustering** The final step of our approach is to apply graph clustering to the query similarity graph computed above. The goal of this step is to discover very similar groups queries out of which prototypical queries can be generated. As a given query can obey the patterns of more than one prototypical query, we opt for using the soft clustering approach implemented by the BorderFlow algorithm<sup>4</sup>.

<sup>3</sup>Available online at: <http://limes.sf.net>

<sup>4</sup>An implementation of the algorithm can be found at <http://borderflow.sf.net>

BorderFlow [Ngonga Ngomo and Schumacher, 2009] implements a seed-based approach to graph clustering. The default setting for the seeds consists of taking all nodes in the input graph as seeds. For each seed  $v$ , the algorithm begins with an initial cluster  $X$  containing only  $v$ . Then, it expands  $X$  iteratively by adding nodes from the direct neighborhood of  $X$  to  $X$  until  $X$  is node-maximal with respect to a function called the border flow ratio. The same procedure is repeated over all seeds. As different seeds can lead to the same cluster, identical clusters (i.e., clusters containing exactly the same nodes) that resulted from different seeds are subsequently collapsed to one cluster. The set of collapsed clusters and the mapping between each cluster and its seeds are returned as result. Applying BorderFlow to the input queries led to 12272 clusters, of which 24% contained only one node, hinting towards a long-tail distribution of query types. To generate the patterns used in the benchmark, we only considered clusters of size 5 and above.

## 6.4. SPARQL Feature Selection and Query Variability

After the completion of the detection of similar queries and their clustering, our aim is now to select a number of frequently executed queries that cover most SPARQL features and allow us to assess the performance of queries with single as well as combinations of features. The SPARQL features we consider are:

- the overall number of triple patterns contained in the query ( $|GP|$ ),
- the graph pattern constructors UNION ( $UON$ ), OPTIONAL ( $OPT$ ),
- the solution sequences and modifiers DISTINCT ( $DST$ ),
- as well as the filter conditions and operators FILTER ( $FLT$ ), LANG ( $LNG$ ), REGEX ( $REG$ ) and STR ( $STR$ ).

We pick different numbers of triple patterns in order to include the efficiency of JOIN operations in triplestores. The other features were selected because they frequently occurred in the query log. We rank the clusters by the sum of the frequency of all queries they contain. Thereafter, we select 25 queries as follows: For each of the features, we choose the highest ranked cluster containing queries having this feature. From that particular cluster we select the query with the highest frequency.

In order to convert the selected queries into query templates, we manually select a part of the query to be varied. This is usually an IRI, a literal or a filter condition. In Figure 6.1 those varying parts are indicated by `%%var%%` or in the case of multiple varying parts `%%varn%%`. We exemplify our approach to replacing varying parts of queries by using Query 9, which results in the query shown in Figure 6.1. This query selects a specific settlement along with the airport belonging to that

```

1 SELECT * WHERE {
2   { ?var0 a dbp-owl:Settlement ;
3     rdfs:label %%var%% .
4     ?var1 a dbp-owl:Airport . }
5   { ?var1 dbp-owl:city ?var0 . }
6   UNION
7   { ?var1 dbp-owl:location ?var0 . }
8   { ?var1 dbp-prop:iata ?var2 . }
9   UNION
10  { ?var1 dbp-owl:iataLocationIdentifier ?var2 . }
11  OPTIONAL
12  { ?var1 foaf:homepage ?var3 . }
13  OPTIONAL
14  { ?var1 dbp-prop:nativename ?var4 . }
15 }

```

Figure 6.1.: Sample query with placeholder.

```

1 SELECT DISTINCT ?var WHERE {
2   { ?var0 a dbp-owl:Settlement ;
3     rdfs:label ?var .
4     ?var1 a dbp-owl:Airport . }
5   { ?var1 dbp-owl:city ?var0 . }
6   UNION
7   { ?var1 dbp-owl:location ?var0 . }
8   { ?var1 dbp-prop:iata ?var2 . }
9   UNION
10  { ?var1 dbp-owl:iataLocationIdentifier ?var2 . }
11  OPTIONAL
12  { ?var1 foaf:homepage ?var3 . }
13  OPTIONAL
14  { ?var1 dbp-prop:nativename ?var4 . }
15 } LIMIT 1000

```

Figure 6.2.: Sample auxiliary query returning potential values a placeholder can assume.

settlement as indicated in Figure 6.1. The variability of this query template was determined by getting a list of all settlements using the query shown in Figure 6.2. By selecting suitable placeholders, we ensured that the variability is sufficiently high ( $\geq 1000$  per query template). Note that the triplestore used for computing the variability was different from the triplestore that we later benchmarked in order to avoid potential caching effects.

For the benchmarking we then used the list of thus retrieved concrete values to replace the `%%var%%` placeholders within the query template. This method ensures, that (a) the actually executed queries during the benchmarking differ, but (b) always return results. This change imposed on the original query avoids the effect of simple caching.

## 6.5. Experimental Setup

This section presents the setup we used when applying the DBPSB on four triplestores commonly used in Data Web applications. We first describe the

triplestores and their configuration, followed by our experimental strategy and finally the obtained results. All experiments were conducted on a typical server machine with an AMD Opteron 6 Core CPU with 2.8 GHz, 32 GB RAM, 3 TB RAID-5 HDD running Linux Kernel 2.6.35-23-server and Java 1.6 installed. The benchmark program and the triplestore were run on the same machine to avoid network latency.

**Triplestores Setup** We carried out our experiments by using the triplestores *Virtuoso* [Erling and Mikhailov, 2007], *Sesame* [Broekstra et al., 2002], *Jena-TDB* [Owens et al., 2008b], and *BigOWLIM* [Bishop et al., 2011]. The configuration and the version of each triplestore were as follows:

1. **Virtuoso:** Open-Source Edition version 6.1.2: We set the following memory-related parameters: `NumberOfBuffers = 1048576`, `MaxDirtyBuffers = 786432`.
2. **Sesame:** Version 2.3.2 with Tomcat 6.0 as HTTP interface: We used the native storage layout and set the `spoc`, `posc`, `opsc` indices in the native storage configuration. We set the Java heap size to 8GB.
3. **Jena-TDB:** Version 0.8.7 with Joseki 3.4.3 as HTTP interface: We configured the TDB optimizer to use statistics. This mode is most commonly employed for the TDB optimizer, whereas the other modes are mainly used for investigating the optimizer strategy. We also set the Java heap size to 8GB.
4. **BigOWLIM:** Version 3.4, with Tomcat 6.0 as HTTP interface: We set the entity index size to 45,000,000 and enabled the predicate list. The rule set was empty. We set the Java heap size to 8GB.

In summary, we configured all triplestores to use 8GB of memory and used default values otherwise. This strategy aims on the one hand at benchmarking each triplestore in a real context, as in real environment a triplestore cannot dispose of the whole memory up. On the other hand it ensures that the whole dataset cannot fit into memory, in order to avoid caching.

### 6.5.1. Benchmark Phases

Once the triplestores loaded the DBpedia datasets with different scale factors, the benchmark execution phase began. It comprised the following stages:

1. **System Restart:** Before running the experiment, the triplestore and its associated programs were restarted in order to clear memory caches.
2. **Warm-up Phase:** In order to measure the performance of a triplestore under normal operational conditions, a warm-up phase was used. In the warm-up phase, query mixes were posed to the triplestore. The queries posed



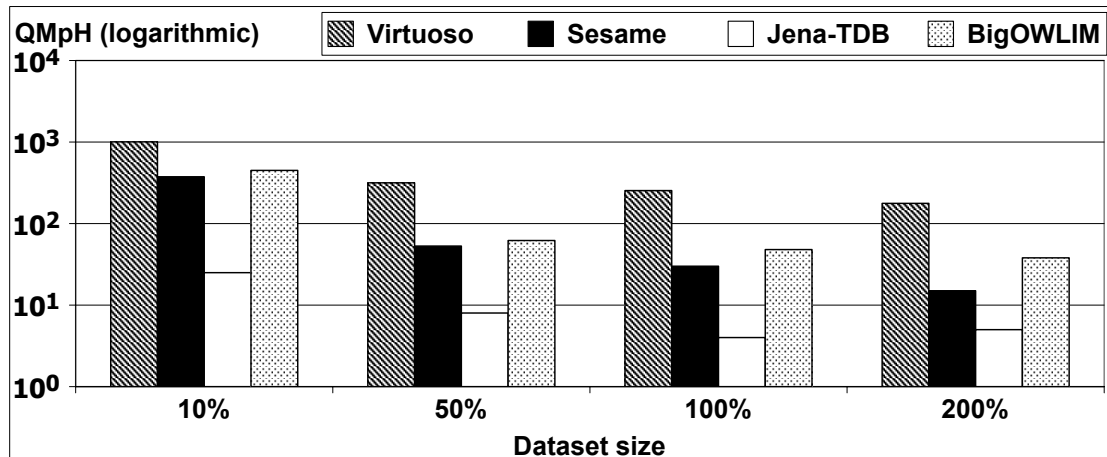


Figure 6.3.: QMPH for all triplestores of DBPSB version 1.

during the warm-up phase were disjoint with the queries posed in the hot-run phase. For DBPSB, we used a warm-up period of 20 minutes.

3. **Hot-run Phase:** During this phase, the benchmark query mixes were sent to the tested store. We kept track of the average execution time of each query as well as the number of query mixes per hour (QMPH). The duration of the hot-run phase in DBPSB was 60 minutes.

Since some benchmark queries did not respond within reasonable time, we specified a 180 second timeout after which a query was aborted and the 180 second maximum query time was used as the runtime for the given query even though no results were returned. The benchmarking code along with the DBPSB queries is freely available<sup>5</sup>.

We have created 2 versions of DBPSB, with the following specifications:

- DBPSB version 1: we used 4 different dataset sizes, i.e. 10%, 50%, 100%, and 200%, and 25 queries for performance evaluation. The query list of DBPSB version 1 can be found in Appendix A.
- DBPSB version 2: we used 3 different dataset sizes, i.e. 10%, 50%, and 100%, and 20 queries for performance evaluation. The query list of DBPSB version 2 can also be found in Appendix A.

## 6.6. Benchmarking Results

We evaluated the performance of the triplestores with respect to two main metrics: their overall performance on the benchmark and their query-based performance

<sup>5</sup><https://akswbenchmark.svn.sourceforge.net/svnroot/akswbenchmark/>

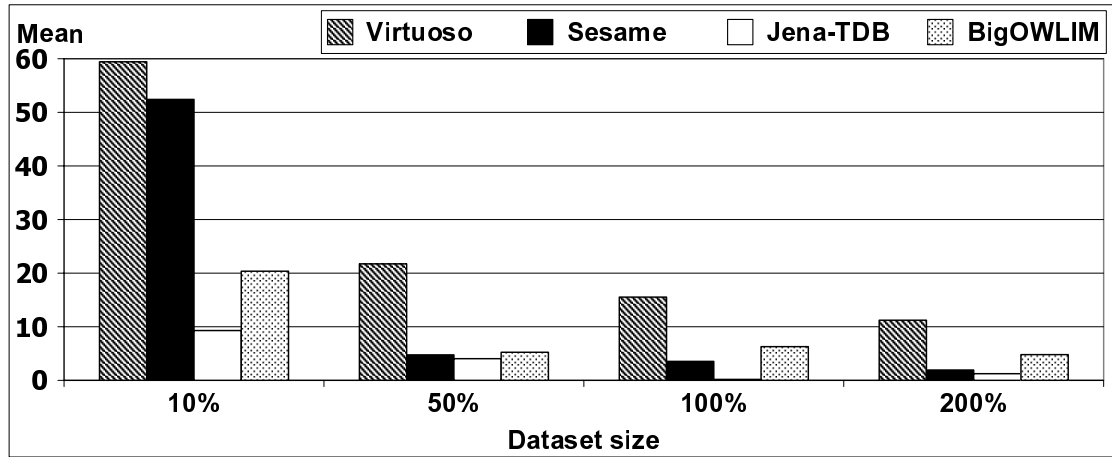


Figure 6.4.: Geometric mean of QpS of DBPSB version 1.

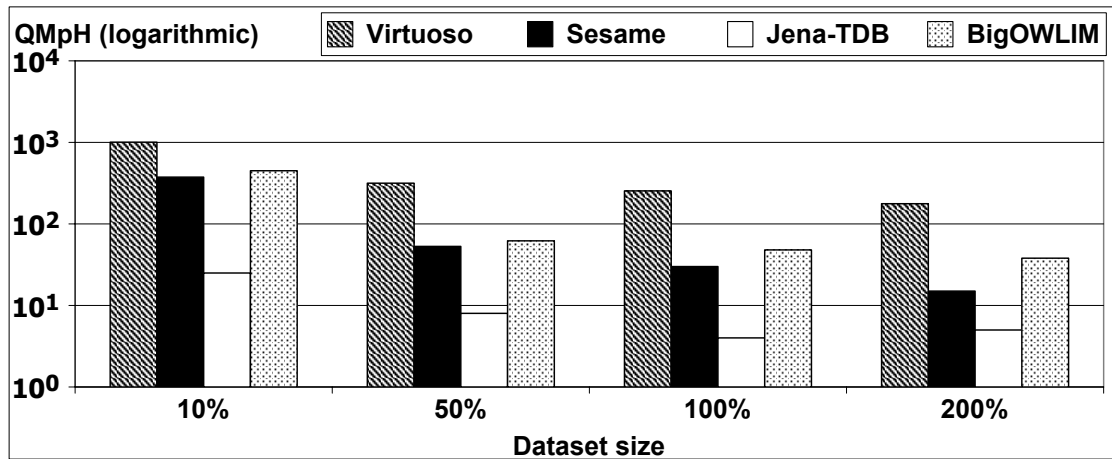


Figure 6.5.: QMpH for all triplestores of DBPSB version 2.

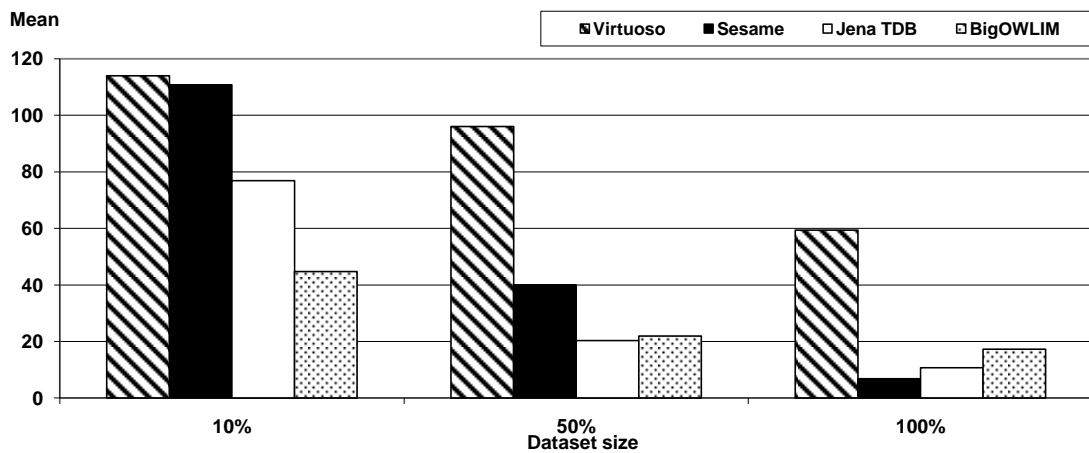


Figure 6.6.: Geometric mean of QpS of DBPSB version 2.

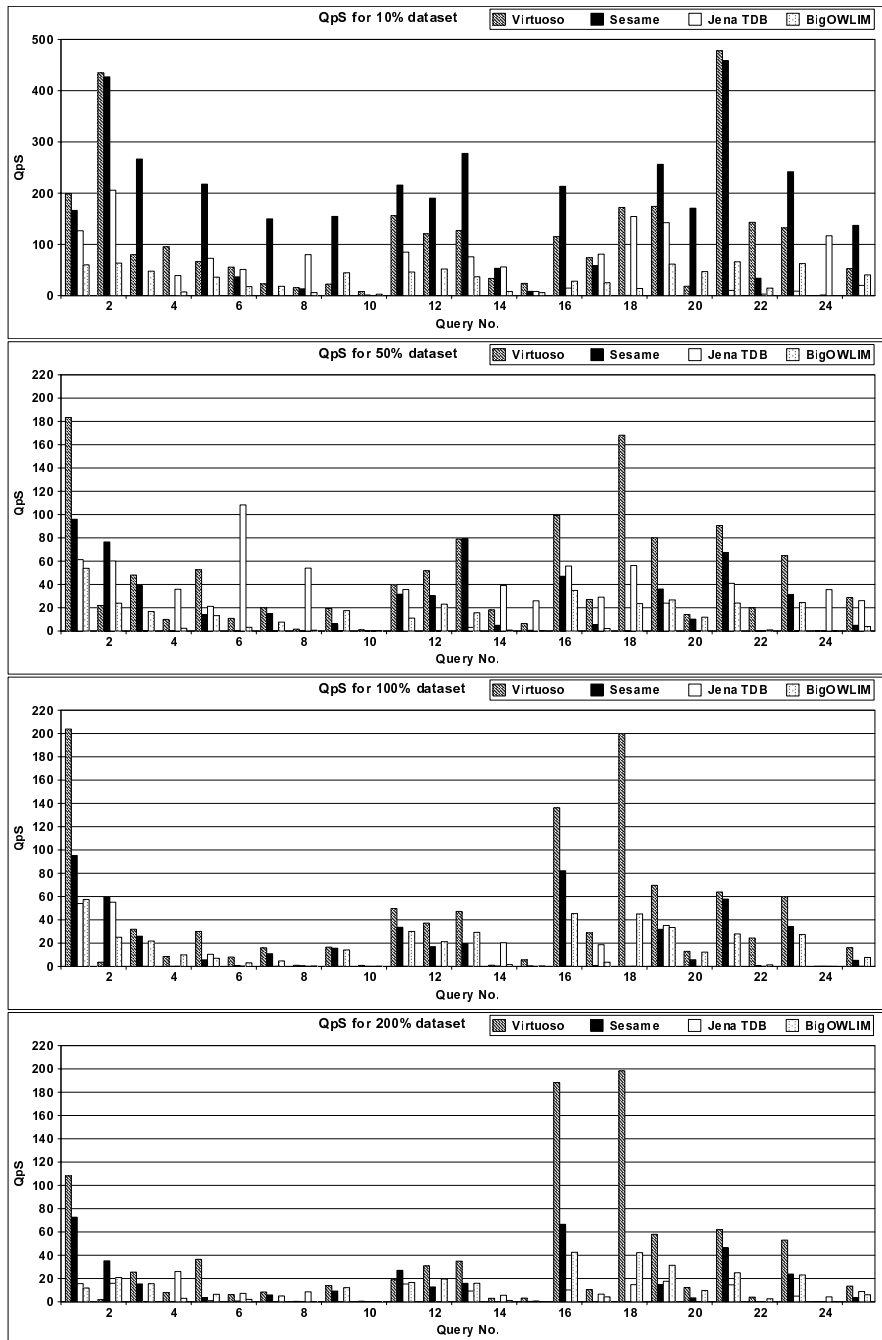


Figure 6.7.: Queries per Second (QpS) of DBPSB version 1 for all triplestores for 10%, 50%, 100%, and 200%.

The overall performance of the triplestores was measured by computing its query mixes per hour (QMpH). The metric used for query-based performance evaluation is Queries per Second (QpS). QpS is computed by summing up the runtime of each query in each iteration, dividing it by the QMpH value and scaling it to seconds.

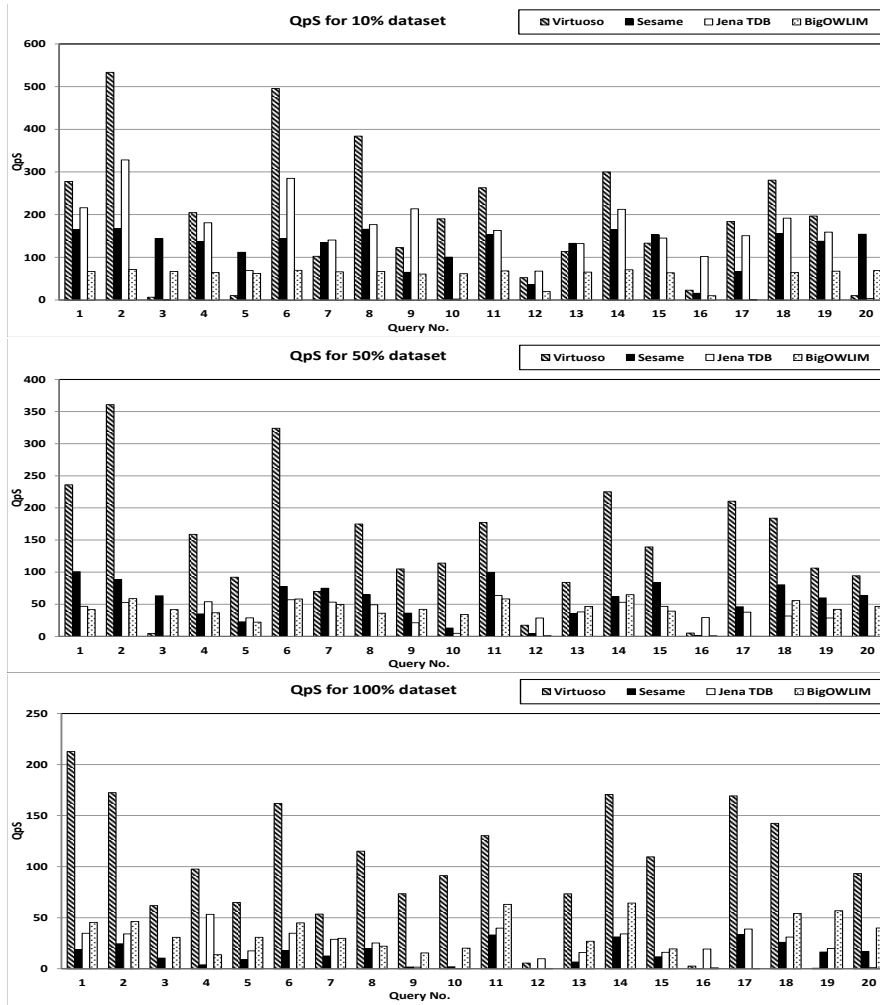


Figure 6.8.: Queries per Second (QpS) of DBPSB version 2 for all triplestores for 10%, 50% and 100%.

### 6.6.1. DBPSB Version 1

The query mixes per hour (QMpH) as shown in Figure 6.3. Please note that we used a logarithmic scale in this figure due to the high performance differences we observed. In general, Virtuoso was clearly the fastest triplestore, followed by BigOWLIM, Sesame and Jena-TDB. The highest observed ratio in QMpH between the fastest and slowest triplestore was 63.5 and it reached more than 10000 for single queries. The scalability of stores did not vary as much as the overall performance. There was on average a linear decline in query performance with increasing dataset size.

We tested the queries that each triplestore failed to executed within the 180s timeout and noticed that even much larger timeouts would not have been sufficient most of those queries. We did not exclude the queries completely from the overall assessment, since this would have affected a large number of the queries and adversely penalized stores, which complete queries within the time

frame. We penalized failure queries with 180s, similar to what was done in the SP<sup>2</sup>Bench [Schmidt et al., 2009]. Virtuoso was the only store, which completed all queries in time. For Sesame and OWLIM only rarely a few particular queries timed out. Jena-TDB had always severe problems with queries 7, 10 and 20 as well as 3, 9, 12 for the larger two datasets.

The metric used for query-based performance evaluation is Queries per Second (QpS). QpS is computed by summing up the runtime of each query in each iteration, dividing it by the QMpH value and scaling it to seconds. The QpS results for all triplestores and for the 10%, 50%, 100%, and 200% datasets are depicted in Figure 6.7.

The outliers, i.e. queries with very low QpS, will significantly affect the mean value of QpS for each store. So, we additionally calculated the geometric mean of all the QpS timings of queries for each store. The main advantage of calculating the geometric mean is that the effect of outliers is weakened. The geometric mean for all triplestores is also depicted in Figure 6.4.

Detailed results of DBPSB version 1 are indicated in tables 6.2, and 6.3.

### 6.6.2. DBPSB1 Results Discussion

This section consists of three parts: First, we compare the general performance of the systems under test. Then we look individual queries and the SPARQL features used within those queries in more detail to observe particular strengths and weaknesses of stores. Thereafter, we compare our results with those obtained with previous benchmarks and elucidate some of the main differences between them.

**General Performance** Figure 6.3 depicts the benchmark results for query mixes per hour for the four systems and dataset sizes. Virtuoso leads the field with a substantial head start of double the performance for the 10% dataset (and even quadruple for other dataset sizes) compared to the second best system (BigOWLIM). While Sesame is able to keep up with BigOWLIM for the smaller two datasets it considerably loses ground for the larger datasets. Jena-TDB can in general not deliver competitive performance with being by a factor 30-50 slower than the fastest system.

If we look at the geometric mean of all QpS results in Figure 6.4, we observe similar insights. The spreading effect is weakened, since the geometric mean reduces the effect of outliers. Still Virtuoso is the fastest system, although Sesame manages to get pretty close for the 10% dataset. This shows that most, but not all, queries are fast in Sesame for low dataset sizes. For the larger datasets, BigOWLIM is the second best system and shows promising scalability, but it is still by a factor of two slower than Virtuoso.

Query	Virtuoso			Sesame			Jena-TDB			BigOWLIM		
	QpS	SD	GM	QpS	SD	GM	QpS	SD	GM	QpS	SD	GM
1	261.6	45.3	250.1	466.3	136.2	428.8	330.4	155.5	258.9	63	8.9	61.9
2	450.9	59	445.6	427.7	15.5	427.4	255.1	80.4	236.5	64.7	4.1	64.4
3	82.8	16.2	81.3	348.3	97.3	320.7	1.4	1.9	0.6	55.3	14.8	52.4
4	138.1	48.9	122.6	10	60	0.2	71.3	61.1	52.7	20.6	21.4	11.6
5	67.7	10.9	67	287.9	65.3	269.6	116.1	70.5	93.3	46.6	17.5	42.1
6	60.5	17.9	58	49.4	5.5	48.2	82.5	58.1	65.2	19.2	5.1	18.5
7	28.5	8.5	26.7	207.1	79	183.7	1.6	2.5	0.5	26.5	13.9	22.7
8	52.8	67.4	24.4	65.7	112.9	23.6	134	75.1	108	18	21.9	9.1
9	22.9	3.9	22.7	226.9	86.8	197.8	0.6	0.6	0.4	48.9	9.1	47.4
10	8.1	0.4	8.1	1.4	0.4	1.4	0.1	0.04	0.1	2.8	0.1	2.8
11	176	36.2	171	289.7	80	265.8	125.3	67	104.9	51.5	12.6	49.3
12	124.8	20.2	123.1	309.9	118	264.8	1	3	0.1	59.6	12.3	57.2
13	129.3	16.5	128.3	367.2	101.4	337.3	190.1	77.3	157.9	46.7	16.4	43
14	83.1	29.8	74.1	179.2	134.6	116	96.2	69	74	25.8	20.5	16.8
15	128.1	67.6	90.3	162.6	148.5	72.6	97.3	100.2	43.9	43.1	23.9	28.4
16	121.5	23.6	118.7	249.9	67.6	236.4	28.8	30.5	19.9	39.8	15.7	35.5
17	102.2	29.8	95.7	186.2	109.5	135.8	115.6	63.6	97.6	42.8	18.9	36.1
18	182.8	33.1	179.1	0.5	0.1	0.5	178.3	52.6	168.1	23.3	15.5	18.3
19	199.8	47.9	191.1	302.5	69.9	286.7	200.8	88.9	174.5	62.2	4.7	61.9
20	18.9	4.1	18.6	221	63.3	203.7	0.1	0.3	0.02	50.6	8.1	49.5
21	483	48	480.6	459.4	16.7	459.1	289.6	92.5	224.9	66.1	1.8	66.1
22	206.1	60.6	190.2	241.4	173.8	140.6	38	86	8.4	52.8	19.3	44.7
23	140.3	27.4	137.1	354.7	116.9	315.3	23.9	39.7	12.6	64.1	6.1	63.6
24	1.2	0.7	0.8	32.4	100.2	3.7	173	88.1	146.4	0.3	0.2	0.2
25	62.8	19	59.2	259.8	120.6	209.5	110.5	99.8	68	52.3	12.5	49.7
1	264.5	76.1	242	153.5	136.3	116.9	64.8	13.6	63.2	56.7	10.1	55.5
2	22.4	3.2	22.2	137	98	109.9	87.6	87.9	68.6	27.7	12.7	25.6
3	55.4	19.7	51.8	81.1	92	52.6	0.1	0.1	0.1	29.4	18.9	23.3
4	44	62.7	17.6	0	0	0	41.7	19.5	38.4	3.9	3.3	3
5	60.3	14	58	46.6	70	24.8	40.2	32	30.6	21.5	15.4	16.9
6	14	7.3	12.3	6.2	2	5	162.5	69.8	141.3	5	2.3	4.3
7	23.1	7.9	21.8	62.2	52.3	42.3	0.2	0.5	0.02	10.7	5.8	9.2
8	32.5	46.9	5	15.6	39.4	2.7	88.2	67.3	67.8	8.2	13.9	1.7
9	20.4	5.3	19.9	42.3	35.3	29.1	2.3	3.8	0.8	27.2	15.1	22.4
10	1	0.1	1	0.2	0	0.2	4.3	3	1	0.28	0.02	0.28
11	97.7	56.9	73.9	45.2	48.1	36.3	37	7.1	36.3	27	10.1	24
12	62.3	22.2	57.6	56.3	58.2	42.4	0.02	0.02	0.02	34.1	18.9	28.7
13	105.1	52.1	91.7	97.5	60.4	86.8	105	94.3	44.9	19.9	10.4	17.9
14	53.9	40	38	20.4	17.6	12.3	42.8	13.2	40.9	7.9	10.6	3.3
15	51.9	32.7	33.4	33.5	66.1	11.5	43.1	38.9	32.8	8.6	13.7	2.2
16	106.7	25.4	103.4	87.7	75.4	64.8	73.2	48.3	63	38.7	12.9	36.7
17	33	11.2	30.6	20.9	19.9	13.3	31.9	9.5	30.5	10.3	8.5	6
18	203.9	57.9	190.4	0.1	0	0.1	57.6	9.6	56.9	29	12.7	26.4
19	106.6	53.6	93.5	46.1	38.1	39.7	26.8	10.2	25.3	33.7	16.9	29.9
20	15	4	14.5	37.7	30.6	28.9	0.01	0	0.01	20.5	16.2	15.2
21	189.1	138.3	135	105.6	87.3	84.3	50	16.2	46.4	28.1	13.4	25.7
22	109.1	43.9	90.3	29.5	41.4	14	1.2	1.5	0.7	15.7	15.9	6.1
23	81.2	31.4	74.4	78.3	97.4	47	1.5	2.4	0.4	32.4	18.2	27.9
24	0.2	0.1	0.2	0.9	0.4	0.7	53.7	39.3	43.6	0.06	0.05	0.04
25	35.6	10.9	33.3	45.2	57.4	26.4	37.1	16	32.4	11.5	7.6	8.6

Table 6.2.: Queries per second (QpS), geometric mean of query runtime in milliseconds (GM), and standard deviation of query runtime in milliseconds (SD), for the 10% dataset, and 50% dataset respectively of DBPSB version 1.

Query	Virtuoso			Sesame			Jena-TDB			BigOWLIM		
	QpS	SD	GM	QpS	SD	GM	QpS	SD	GM	QpS	SD	GM
1	245.9	30.9	240.9	112.7	47.1	103.9	54.5	5.9	54.2	58.3	6.1	57.9
2	3.6	0.1	3.6	81.1	45.9	69.8	67.1	40	60	31	11.8	28.7
3	42.8	21.8	37.8	32.1	14.5	28.9	0.02	0.03	0.01	23.9	7.7	22.8
4	34	50.2	14.4	0.03	0.01	0.02	4.6	9.1	0.3	29.4	22.4	18.8
5	47.9	19.7	41.7	10.7	10.6	7.6	12.5	4.9	11.5	16.1	14.5	11.1
6	8.6	2.3	8.3	4.2	2.4	2.8	45.5	46.9	9.6	4.9	2.3	4.1
7	21	12.1	18.2	21.5	37.4	13.8	0.04	0.03	0.03	7.4	4.5	6.1
8	38.3	47.7	4.8	17.8	33.7	2.7	27.1	36.7	1.3	8.3	14.5	1
9	17.8	5.7	17.1	23.3	15.9	19.5	0.01	0	0.01	15.9	6.3	14.9
10	1	0.1	0.9	0.1	0	0.1	0.01	0	0.01	0.16	0.01	0.16
11	115.7	31.7	100.2	48.5	41.9	39.4	2.9	5.7	0.2	36.2	14.4	33.2
12	47.1	25.4	41.7	25.4	12.9	21.7	0.01	0	0.01	24.6	9.7	22.9
13	89.5	73.6	64.6	43.1	41.4	28.6	0.2	0.2	0.1	38.1	17.9	33.6
14	25.1	37.3	6	3.4	5.2	1	26.3	15.7	23	28.1	26.8	8.5
15	48.4	33.2	31.1	3.1	4.7	1.3	0.04	0.04	0.02	7.9	12.2	2.5
16	137.3	14.4	136.7	98.8	49.4	89.7	29.3	30.8	6.2	48.5	10.4	47.1
17	32.8	9.3	31.2	6.3	6.5	2.7	21.7	8.1	20.3	21.5	14.8	12.3
18	208.3	27.1	205.6	0.1	0	0	44.8	34.8	9.2	47	8.3	46.1
19	99.2	67.3	81.8	40.3	20.6	35.8	46.5	24.6	40.9	34.9	7	34.2
20	14.1	4.1	13.5	8.2	6.6	6.8	0.01	0	0.01	15.1	8.3	13.5
21	98.8	76.2	78.3	85.7	69	69.2	0.1	0.1	0.1	31.3	10.5	29.7
22	115.1	52.9	93.6	5.5	8.5	2	9.5	13.3	0.6	9.5	10.6	4.6
23	76.7	38.9	67.8	41	17.8	37.6	20.2	34.4	0.8	29.1	8.4	28.2
24	0.2	0.1	0.1	0.7	0.4	0.4	17.4	15	3.7	0.05	0.05	0.03
25	30.1	11	27.1	16.5	14.5	10.3	18.1	22.5	1.2	14.3	9.4	11
1	247.1	44.4	229.9	93.2	39.2	84.4	226.9	465.5	36.6	54.5	13.4	47.7
2	1.8	0.1	1.8	45.7	27.4	39.9	17.6	6.4	16.6	26.8	11.2	24.5
3	42.8	21.1	36.6	19.8	9.1	17.7	0.03	0.02	0.02	18.1	8.2	16.7
4	26.8	46.1	12.1	0	0	0	45.9	33.6	34.4	6.8	9	4.1
5	52.7	19.8	47.4	5.5	2.7	4.7	5.7	4.3	3	11.8	10.4	8.7
6	9.4	6.1	7.6	2.4	2	1.4	34.6	47	15.1	2.9	1.3	2.6
7	14.3	9.3	11.7	7.1	2.4	6.6	0.01	0	0.01	8.7	5.3	7
8	33.6	42.9	2.8	3.8	6.7	0.4	21.8	19.9	13.9	8.8	15.7	1
9	16.1	6	15.1	9.9	3.2	9.5	0.02	0.04	0.01	14.5	5.1	13.6
10	0.5	0	0.5	0.1	0	0	0.01	0	0.01	0.16	0.02	0.16
11	114.7	57.5	73	34.3	17.2	30.5	15.9	2.7	15.6	21	7.9	19.2
12	43.8	31.8	36.1	18	8.1	15.7	0.01	0	0.01	21.1	5.4	20.4
13	64.7	51.8	48.6	21.7	13.2	18.5	13.6	9.5	11.1	20.1	10.3	18.2
14	23.9	29.5	11.4	1.7	2.8	0.4	7.2	2.9	6.5	6.3	6.5	3.4
15	65.1	51.5	36.9	3.3	4.5	1.4	1.8	1.6	1.2	3.8	8.8	0.8
16	191.2	24.1	189.8	83.6	40.3	74.9	11.1	3.1	10.6	45.4	10.6	44
17	24.1	12.2	19.4	1.6	1.8	0.8	6.9	1.7	6.8	9.1	6.7	6.8
18	212	35.8	207.3	0	0	0	19.7	7.9	17.5	44	8.3	43.1
19	84.8	62.5	69	15.9	4.3	15.3	22.4	13	19.6	34	8.9	32.7
20	13.6	4.4	12.9	8.6	17.5	4.5	0.01	0	0.01	13.1	8.2	11.5
21	93.8	77	73.7	53.6	20.8	50	15.9	4.9	15.2	26.5	6.3	25.7
22	120	80.5	74.3	20.7	66.2	2.1	0.6	0.6	0.4	11.1	11.5	5.9
23	67.6	33.3	60.1	30.3	16.7	26.9	9.2	5.9	6.9	24.1	5.5	23.6
24	0.1	0	0.1	0.3	0.2	0.2	8.3	4.8	6.4	0.05	0.05	0.02
25	23.5	15.2	19	8.1	7.8	5.2	9.2	1.6	9	11.6	9.5	8.6

Table 6.3.: Queries per second (QpS), geometric mean of query runtime in milliseconds (GM), and standard deviation of query runtime in milliseconds (SD), for the 100% dataset, and 200% dataset of DBPSB version 1.

**Scalability, Individual Queries and SPARQL Features** Our first observation with respect to individual performance of the triple stores is that Virtuoso demonstrates a good scaling factor on the DBPSB. When dataset size changes by factor 5 (from 10% to 50%), the performance of the triple store only degrades by factor 3.12. Further dataset increases (i.e. the doubling to the 100% and 200% datasets) result in only relatively small performance decreases by 20% and respectively 30%.

Virtuoso outperforms Sesame for all datasets. In addition, Sesame does not scale as well as Virtuoso for small dataset sizes, as its performance degrades sevenfold when the dataset size changes from 10% to 50%. However, when the dataset size doubles from the 50% to the 100% dataset and from 100% to 200% the performance degrades by just half.

The performance of Jena-TDB is the lowest of all triple stores and for all dataset sizes. The performance degradation factor of Jena-TDB is not as pronounced as that of Sesame and almost equal to that of Virtuoso when changing from the 10% to the 50% dataset. However, the performance of Jena-TDB only degrades by a factor of 2 for the transition between the 50% and 100% dataset, and reaches 0.8 between the 100% and 200% dataset, leading to a slight increase of its QMpH.

BigOWLIM is the second fastest triple store for all dataset sizes, after Virtuoso. BigOWLIM degrades with a factor of 7.2 in transition from 10% to 50% datasets, but it decreases dramatically to 1.29 with dataset size 100%, and eventually reaches 1.26 with dataset size 200%.

Due to the high diversity in the performance of different SPARQL queries, we also computed the geometric mean of the QpS values of all queries as described in the previous section and illustrated in Figure 6.4. By using the geometric mean, the resulting values are less prone to be dominated by a few outliers (slow queries) compared to standard QMpH values. This allows for some interesting observations in DBPSB by comparing Figure 6.3 and 6.4. For instance, it is evident that Virtuoso has the best QpS values for all dataset sizes.

With respect to Virtuoso, query 10 performs quite poorly. This query involves the features `FILTER`, `DISTINCT`, as well as `OPTIONAL`. Also, the well performing query 1 involves the `DISTINCT` feature. Query 3 involves a `OPTIONAL` resulting in worse performance. Query 2 involving a `FILTER` condition results in the worst performance of all of them. This indicates that using complex `FILTER` in conjunction with additional `OPTIONAL`, and `DISTINCT` adversely affects the overall runtime of the query.

Regarding Sesame, queries 4 and 18 are the slowest queries. Query 4 includes `UNION` along with several free variables, which indicates that using `UNION` with several free variables causes problems for Sesame. Query 18 involves the features `UNION`, `FILTER`, `STR` and `LANG`. Query 15 involves the features `UNION`, `FILTER`, and `LANG`, and its performance is also pretty slow, which leads to the conclusion that introducing this combination of features is difficult for Sesame. Adding the `STR` feature to that feature combination affects the performance dramatically and prevents the query from being successfully executed.



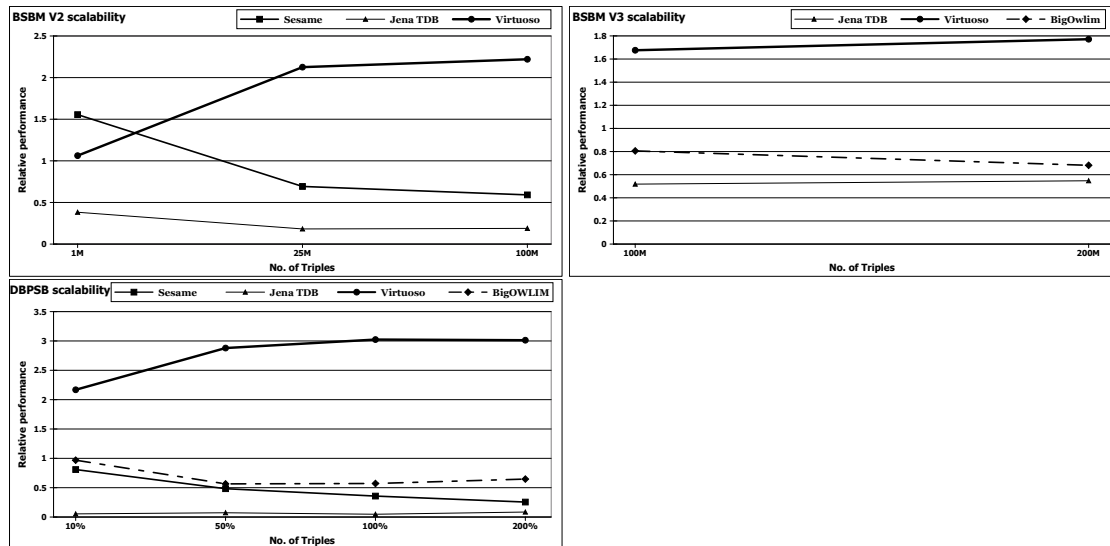


Figure 6.9.: Comparison of triple store scalability between BSBM V2, BSBM V3, DBPSB.

For Jena-TDB, there are several queries that timeout with large dataset sizes, but queries 10 and 20 always timeout. The problem with query 10 is already discussed with Virtuoso. Query 20 contains `FILTER`, `OPTIONAL`, `UNION`, and `LANG`. Query 2 contains `FILTER` only, query 3 contains `OPTIONAL`, and query 4 contains `UNION` only. All of those queries run smoothly with Jena-TDB, which indicates that using the `LANG` feature, along with those features affects the runtime dramatically.

For BigOWLIM, queries 10, and 15 are slow queries. Query 10 was already problematic for Virtuoso, as was query 15 for Sesame.

Query 24 is slow on Virtuoso, Sesame, and BigOWLIM, whereas it is faster on Jena-TDB. This is due to the fact that most of the time this query returns many results. Virtuoso, and BigOWLIM return a bulk of results at once, which takes long time. Jena-TDB just returns the first result as a starting point, and iteratively returns the remaining results via a buffer.

It is interesting to note that BigOWLIM shows in general good performance, but almost never manages to outperform any of the other stores. Queries 11, 13, 19, 21 and 25 were performed with relatively similar results across triple stores thus indicating that the features of these queries (i.e. `UON`, `REG`, `FLT`) are already relatively well supported. With queries 3, 4, 7, 9, 12, 18, 20 we observed dramatic differences between the different implementations with factors between slowest and fastest store being higher than 1000. It seems that a reason for this could be the poor support for `OPT` (in queries 3, 7, 9, 20) as well as certain filter conditions such as `LNG` in some implementations, which demonstrates the need for further optimizations.

**Comparison with Previous Benchmarks** In order to visualize the performance improvement or degradation of a certain triple store compared to its competitors, we calculated the relative performance for each store compared to the average and depicted it for each dataset size in Figure 6.9. We also performed this calculation for BSBM version 2 and version 3. Overall, the benchmarking results with DBPSB were less homogeneous than the results of previous benchmarks. While with other benchmarks the ratio between fastest and slowest query rarely exceeds a factor of 50, the factor for the DBPSB queries (derived from real DBpedia SPARQL endpoint queries) reaches more than 1 000 in some cases.

As with the other benchmarks, Virtuoso was also fastest in our measurements. However, the performance difference is even higher than reported previously: Virtuoso reaches a factor of 3 in our benchmark compared to 1.8 in BSBM V3. BSBM V2 and our benchmark both show that Sesame is more suited to smaller datasets and does not scale as well as other stores. Jena-TDB is the slowest store in BSBM V3 and DBPSB, but in our case they fall much further behind to the point that Jena-TDB can hardly be used for some of the queries, which are asked to DBpedia. The main observation in our benchmark is that previously observed differences in performance between different triple stores amplify when they are confronted with actually asked SPARQL queries, i.e. there is now a wider gap in performance compared to essentially relational benchmarks.

### 6.6.3. DBPSB Version 2

The query mixes per hour (QMpH) as shown in Figure 6.5. Note that we used a logarithmic scale in this figure due to the high performance differences we observed. In general, Virtuoso was clearly the fastest triplestore, followed by BigOWLIM, Sesame and Jena-TDB. The highest observed ratio in QMpH between the fastest and slowest triplestore was 3.2 and it reached more than 1,000 for single queries. The scalability of stores did not vary as much as the overall performance. There was on average a linear decline in query performance with increasing dataset size.

The QpS results for all triplestores and for the 10%, 50% and 100% datasets are depicted in Figure 6.8. As the outliers (i.e. queries with very low QpS) affect the mean value of QpS for each store significantly, we also computed the geometric mean of all the QpS timings of queries for each store. The geometric mean for all triplestores is also depicted in Figure 6.6.

Although Virtuoso performed best overall, it displayed very low QpS rates on Q3, Q5 and Q16. All of these queries require dealing extensively with literals (in contrast to resources). Especially Q16 combined four different SPARQL features (optional,filter,lang and distinct) which seemed to require a significant amount of processing time. BigOWLIM was mainly characterized by a good scalability as it achieves the slowest decrease of its QMpH rates over all the datasets. Still, some queries were also particularly difficult to process for BigOWLIM. Especially Q16 and Q12 which involves three resp. four SPARQL features and a lot of string manipulations were slow to run. Sesame dealt well with most queries for the 10%

Query	Virtuoso			Sesame			Jena-TDB			BigOWLIM		
	QpS	SD	GM	QpS	SD	GM	QpS	SD	GM	QpS	SD	GM
1	281.8	32.1	280.2	169	9.6	168.4	353.6	161.1	298.4	68.7	8.3	68
2	541.2	58.2	538.1	174	5.4	173.5	445.4	186.1	395.3	71.4	1.3	71.4
3	77.6	14.6	76.1	150.1	9.9	149.2	5.3	29.5	0.5	67.2	3.6	67
4	207.4	26.2	206.1	150.7	20	148	278.4	124.7	237.7	66.3	7.9	65.6
5	129.2	22	127	134.6	25.4	130	139.6	102.6	103.5	63.9	7	63.3
6	503	56.2	499.9	164.3	7	164	356	128.3	326.3	69.8	2.7	69.7
7	104.2	17.6	103.2	148.2	9.8	147.5	239.4	128.6	193.3	66.2	3.3	66.1
8	387.2	33.3	385.9	169.5	6.3	169.1	325.2	198.2	252.6	68.7	8	68
9	134.2	39.9	128.3	73.8	19.5	70.9	292.5	169.6	249.3	61.1	4.5	60.9
10	210.9	43.7	204.3	140.5	30.7	132.6	118.9	135	56.2	65.7	9.8	64.3
11	266	31	264.6	154.7	6.5	154.5	338.7	176.3	276.2	68.4	2.5	68.4
12	90.6	29.5	83	99	42.4	83.8	112.8	90.9	86.1	44.6	20.2	36.7
13	116.1	20.4	114.7	153.8	14.5	152.2	202.9	134.2	164.6	66.8	6.8	66.3
14	305.7	43	303.1	169.8	8.1	169.3	351.1	189.6	286.1	71.1	3.1	71
15	135.3	21.8	134	156.4	8.4	155.9	242.4	137.6	199.8	66	8.6	65.2
16	146.9	58.7	119.4	106.9	50.7	80.7	193.2	146.2	141.7	52.1	19.4	44.2
17	193.5	34.3	189.9	83.1	23	78.7	229.8	99.6	201.3	0.6	0	0.6
18	285.3	32.8	283.6	161.3	8.6	160.8	306.6	163.1	253	66.9	4.4	66.5
19	202.4	34.5	199.7	156.9	13.8	155.5	290.2	203.6	220.7	68.5	4.5	68.3
20	124.1	21.5	122.2	160.3	10.2	159.5	13.8	48.1	4.6	69.1	1.9	69.1
1	267.2	50.9	258.2	131.3	46.3	119	78.2	87.8	56.8	48.2	16.9	45
2	391.1	52	384.2	128.3	52.1	111.6	132.7	168.1	75.5	61.7	10.6	60.5
3	67.6	13.8	65.8	97.9	47.1	82.5	0.2	0.2	0.1	52	16.4	47.9
4	173.5	33.7	168.7	89.4	56.9	64.8	101.1	101.4	69.6	45.4	17.2	41.5
5	100.8	24.5	97.3	72.2	50.7	51.8	47.8	62.8	33.9	46.1	19.7	38.1
6	360.9	50.1	353.7	118.4	50.2	101.6	112.6	120	75.4	60.3	9.3	59.4
7	74.6	16.7	72.6	98.3	41.8	87.5	87.4	75.6	66.5	53.3	11.8	51.7
8	211.5	50.4	200.6	108.5	53.7	89.5	94.9	108.5	64.2	42.7	16.8	39.4
9	111.8	23.2	109.1	82.9	58.2	57.9	40.7	73.6	25.9	50.1	16.6	46.5
10	142	31.4	136	66.1	55.1	39.1	19.7	51.9	8.7	46.2	15.3	42.3
11	181.6	25.1	179.8	125.5	42.4	115.1	130.5	146.8	84	62.2	6.4	61.4
12	84.3	46.5	62.8	45.1	43.6	22.4	43.9	54.2	33.6	13.9	17.2	5
13	93.8	23	90.2	76.5	54.2	53.9	69.6	77	48.9	50.1	12.2	48.5
14	270.1	61.6	256.7	110.9	47.7	97.8	87.7	91.6	64.6	65.1	3.2	65
15	155.1	35.4	149.3	107.3	42.2	96.9	74.5	72.9	56.6	45.2	15	42.4
16	70.7	32.2	54.3	52.6	49.2	25.8	40.5	40.2	33.3	21.6	19.8	10
17	237	36.5	231.5	84.2	52	64.4	54.2	49	43.7	0.1	0	0.1
18	205.8	43.7	198.3	111.4	47.2	98.2	46	36.2	37.5	58.2	9.4	57.2
19	112.6	23.8	109.8	97.4	51.6	79.6	47.2	70.9	33.9	51.4	17.8	47.2
20	98.5	20	96.5	98.8	49.6	82.6	1.2	2	0.5	54.8	15.2	51.5

Table 6.4.: Queries per second (QpS), geometric mean of query runtime in milliseconds (GM), and standard deviation of query runtime in milliseconds (SD), for the 10% dataset, and 50% dataset of DBPSB version 2.

dataset. The QMpH that it could achieve yet diminishes significantly with the size of the data set. This behavior becomes especially obvious when looking at Q4, Q10 and Q12. Especially Q4 which combines several triple patterns through a UNION leads to a considerable decrease of the runtime. Jena TDB had the most difficulties dealing with the 100% data set. This can be observed especially on Q9, which contained four triple patterns that might have lead to large intermediary results. Especially in the case Jena TDB, we observed that the 8GB RAM were not always sufficient for storing the intermediary results, which led to swapping and a considerable reduction of the overall performance of the system. Detailed results of DBPSB version 2 are indicated in tables 6.4, and 6.5.

Query	Virtuoso			Sesame			Jena-TDB			BigOWLIM		
	QpS	SD	GM	QpS	SD	GM	QpS	SD	GM	QpS	SD	GM
1	270	71.3	252.6	24.5	15	21.2	103.7	144.5	52.1	53.4	17.2	49.8
2	226.5	66.5	208.2	33.6	24.6	28.1	110.8	157.6	54.4	50.7	14.7	48.5
3	71.4	22.7	67.3	12.2	5.5	11.2	0.1	0.1	0.1	37.3	14.7	34.2
4	124	39.7	114.4	4.5	2.1	4.2	119.3	118	76.7	30.9	21.6	22
5	73.9	23.4	69.8	14.4	8.2	12	36	60.3	22	46.2	17.9	40.6
6	214	64.6	196	22.9	16.9	19.7	101.3	119.1	57.3	48.8	13.6	46.8
7	63.3	21.2	59.1	22.6	14.9	17.2	73.3	92.4	43	40.6	18.8	35.4
8	266.9	135.3	202.7	41.2	25.4	31.3	81	118.3	41.2	44	21.8	35.1
9	92	34.3	84	5.4	7.2	3.3	37.7	87.3	17.4	30.7	16.7	24.4
10	123.5	35.6	114.7	2.1	0.4	2	11.7	52	1.7	35.3	15.8	31.1
11	131.9	17.8	131	38.5	13.2	35.9	108.1	135	60.6	63.4	3.2	63.3
12	53.7	41	31.2	0.4	1.1	0	24.6	47.2	13.2	9.6	14.7	2.2
13	108	47.5	93.3	8.5	3.7	7.5	36.4	66.1	20.3	34.7	15.8	31
14	234.7	83.5	210.5	47.8	23.9	40.5	102.3	134.2	54	64.7	4.1	64.6
15	168.1	70.8	144.8	16.3	11	13.6	28.5	37.1	20.6	34.1	21.8	26.4
16	70.4	42	42.5	2.1	2.6	0.8	71.2	113.8	32	30	22.6	15.8
17	227	38.5	218.5	41	18.1	37.2	105.3	108.8	61	0.3	0	0.3
18	188	60.4	172.3	30.6	12.8	28.2	101.7	145	49.1	58.1	10.6	56.6
19	128.6	43.5	108.8	28.4	18.6	22.1	78.6	147.4	31.2	60.8	11.2	59.2
20	114	38.7	105.4	19.4	7.2	18.2	4.2	4.9	2.8	43.7	13.5	41.8

Table 6.5.: Queries per second (QpS), geometric mean of query runtime in milliseconds (GM), and standard deviation of query runtime in milliseconds (SD), for the 100% dataset of DBPSB version 2.

## 7. Related Work

This chapter discusses similar work achieved, and contrasts our work to them. It orders the related work into several categories and compares each category to the closest category of our thesis.

### 7.1. Semantic Data Extraction from Wikipedia

There are several projects, which aim at extracting semantic data from Wikipedia.

YAGO2 is an extension of the YAGO knowledge base [Suchanek et al., 2008]. YAGO2 uses Wikipedia, Geonames, and WordNet as sources of data. The predecessor of YAGO2, i.e. YAGO just used Wikipedia as its main source of data. It extracts several relations (e.g. `subClassOf`, and `type`) mainly from the *category pages* of Wikipedia. Category pages are lists of articles that belong to a specific category. For instance, William Shakespeare is in the category English poets. These lists give candidates for entities (i.e. William Shakespeare), candidates for concepts (`IsA(William Shakespeare, Poet)`), and candidates for relations (e.g. `isCitizenOf(William Shakespeare, England)`). YAGO, then links the Wikipedia category hierarchy to the WordNet hierarchy, in order to enhance its classification hierarchy [Suchanek et al., 2008].

YAGO2 is the new version of the YAGO project. It uses Geonames as an additional source of data. YAGO2 introduces the integration of the spatio-temporal dimension. In contrast to the original YAGO, the methodology of building YAGO2 (and also maintaining it) is systematically designed top-down with the goal of integrating entity-relationship-oriented facts with the spatial and temporal dimensions. Moreover, YAGO represent facts in the form of subject-property-object triples (SPO triples) according to the RDF data model. YAGO2 introduces a new *spatio-temporally* model, which represents facts in the form of SPOTL tuples (i.e. SPO + Time + Location). This new knowledge representation scheme is very beneficial. For example, with the old representation scheme a knowledge base may store that a certain person is the president of a certain country, but presidents of countries change. So, it is crucial to capture the time periods during which facts of this kind actually happened [Hoffart et al., 2010]. Both YAGO and YAGO2, however, do not focus on extracting data from Wikipedia infoboxes.

KYLIN is a project based also on Wikipedia, which aims at creating or completing infoboxes by extracting information from the article text. KYLIN looks for classes of pages with similar infoboxes, determines common attributes, creates training examples, learns the extractors, and runs them on each page. Thus creating new

infoboxes or completing existing ones. It uses learning techniques to automatically fill in missing values in incomplete infoboxes. There are several problems which may exist in an infobox, such as incompleteness or inconsistency. For example, some infoboxes contain incomplete data which may, however, exist in the article text, while some other infoboxes may contain data that contradicts with the article text [Wu and Weld, 2007]. Although both DBpedia, and KYLIN work on Wikipedia infoboxes both have different objectives. DBpedia aims at extracting data from infoboxes and converting them into semantic data, whereas KYLIN tries to fill the gaps that may exist in some infoboxes.

Freebase is a large collaborative knowledge base, which uses various sources of data including Wikipedia and MusicBrainz<sup>1</sup>. It was originally developed by the software company Metaweb, which was later acquired by Google. Basically, Freebase also extracted facts from Wikipedia articles as initial content and which the users can later extend and revise [Bollacker et al., 2008]. Both DBpedia and Freebase use Wikipedia as a data source, but Freebase uses it only as a starting point in a way that its users can modify the data, whereas DBpedia aims to be closely aligned with Wikipedia.

## 7.2. RDF Benchmarks

Several RDF benchmarks were previously developed. We first present them, and then compare them to our benchmark.

### 7.2.1. Existing Benchmarks

The *Lehigh University Benchmark* (LUBM) [Pan et al., 2005] was one of the first RDF benchmarks. LUBM uses an artificial data generator, which generates synthetic data for universities, their departments, their professors, employees, courses and publications. This small number of classes limits the variability of data and makes LUBM inherent structure more repetitive. Moreover, the SPARQL queries used for benchmarking in LUBM are all plain queries, i.e. they contain only triple patterns with no other SPARQL features (e.g. `FILTER`, or `REGEX`). LUBM performs each query 10 consecutive times, and then it calculates the average response time of that query. Executing the same query several times without introducing any variation enables query caching, which affects the overall average query times.

SP<sup>2</sup>Bench [Schmidt et al., 2009] is another more recent benchmark for RDF stores. Its RDF data is based on the Digital Bibliography & Library Project (DBLP) and includes information about publications and their authors. It uses the SP<sup>2</sup>Bench Generator to generate its synthetic test data, which is in its schema heterogeneity even more limited than LUBM. The main advantage of SP<sup>2</sup>Bench

---

<sup>1</sup><http://musicbrainz.org/>

over LUBM is that its test queries include a variety of SPARQL features (such as `FILTER`, and `OPTIONAL`). The main difference between the DBpedia benchmark and SP<sup>2</sup>Bench is that both test data and queries are synthetic in SP<sup>2</sup>Bench. In addition, SP<sup>2</sup>Bench only published results for up to 25M triples, which is relatively small with regard to datasets such as DBpedia and LinkedGeoData.

Another benchmark described in [Owens et al., 2008a] compares the performance of BigOWLIM and AllegroGraph. The size of its underlying synthetic dataset is 235 million triples, which is sufficiently large. The benchmark measures the performance of a variety of SPARQL constructs for both stores when running in single and in multi-threaded modes. It also measures the performance of adding data, both using bulk-adding and partitioned-adding. The downside of that benchmark is that it compares the performance of only two triplestores. Also the performance of each triplestore is not assessed for different dataset sizes, which prevents scalability comparisons.

The Berlin SPARQL Benchmark (BSBM) [Bizer and Schultz, 2009] is a benchmark for RDF stores, which is applied to various triplestores, such as Sesame, Virtuoso, and Jena-TDB. It is based on an e-commerce use case in which a set of products is provided by a set of vendors and consumers post reviews regarding those products. It tests various SPARQL features on those triplestores. It tries to mimic a real user operation, i.e. it orders the query in a manner that resembles a real sequence of operations performed by a human user. This is an effective testing strategy. However, BSBM data and queries are artificial and the data schema is very homogeneous and resembles a relational database. This is reasonable for comparing the performance of triplestores with RDBMS, but does not give many insights regarding the specifics of RDF data management.

In general, existing SPARQL benchmark efforts such as LUBM [Pan et al., 2005], BSBM [Bizer and Schultz, 2009] and SP<sup>2</sup>Bench [Schmidt et al., 2009] resemble relational database benchmarks. Especially, the data structures underlying these benchmarks are basically relational data structures, with relatively few and homogeneously structured classes. However, RDF knowledge bases are increasingly heterogeneous. Thus, they do not resemble relational structures and are not easily representable as such. Examples of such knowledge bases are curated biomedical ontologies such as those contained in Bio2RDF [Belleau et al., 2008] as well as knowledge bases extracted from unstructured or semi-structured sources such as DBpedia [Lehmann et al., 2009, Morsey et al., 2012b] or LinkedGeoData [Auer et al., 2009, Stadler et al., 2012]. For instance, DBpedia contains thousands of classes and properties. DBpedia (version 3.6) for example contains 289,016 classes of which 275 classes belong to the DBpedia ontology. Moreover, it contains 42,016 properties, of which 1335 are in the DBpedia ontology. Also, various data types and object references of different types are used in property values. Such knowledge bases *cannot* be easily represented according to the relational data model and hence performance characteristics for loading, querying and updating these knowledge bases might potentially be fundamentally different from knowledge bases resembling relational data structures.

	LUBM	SP <sup>2</sup> Bench	BSBM V2	BSBM V3	DBPSB
<b>RDF stores</b>	DLDB-OWL,	ARQ, Redland,	Virtuoso,	Virtuoso, 4store,	Virtuoso,
<b>tested</b>	Sesame, OWL-JessKB	SDB, Sesame, Virtuoso	Jena-TDB, Jena-SDB	BigData Jena-TDB BigOwlim	Jena-TDB, BigOWLIM Sesame
<b>Test data</b>	Synthetic	Synthetic	Synthetic	Synthetic	Real
<b>Test queries</b>	Synthetic	Synthetic	Synthetic	Synthetic	Real
<b>Size of tested datasets</b>	0.1M, 0.6M, 1.3M, 2.8M, 6.9M	10k, 50k, 250k, 1M,	1M, 25M, 100M, 5M, 25M	100M, 200M	14M, 75M, 150M, 300M
<b>Dist. queries</b>	14	12	12	12	25
<b>Multi-client</b>	–	–	x	x	–
<b>Use case</b>	Universities	DBLP	E-commerce	E-commerce	DBpedia
<b>Classes</b>	43	8	8	8	239 (internal) +300K(YAGO)
<b>Properties</b>	32	22	51	51	1200

Table 7.1.: Comparison of different RDF benchmarks.

## 7.2.2. Comparison between DBPSB and The Other Benchmarks

In contrast to other benchmarks, DBPSB performs measurements on *real* queries that were issued by humans or Data Web applications against existing RDF data. In order to obtain a representative set of *prototypical queries* reflecting the typical workload of a SPARQL endpoint, we perform a query analysis and clustering on queries that were sent to the official DBpedia SPARQL endpoint. From the highest-ranked query clusters (in terms of aggregated query frequency), we derive a set of SPARQL query templates, which cover most commonly used SPARQL feature combinations and are used to generate the actual benchmark queries by parametrization. The benchmark methodology and results are also available online<sup>2</sup>. Although we apply this methodology to the DBpedia dataset and its SPARQL query log in this case, the same methodology can be used to obtain application-specific benchmarks for other knowledge bases and query workloads. Since DBPSB changes with the data and queries in DBpedia, we envision to update it in yearly increments and publish results on the aforementioned website. In general, our methodology follows the four key requirements for domain specific benchmarks as postulated in the Benchmark Handbook [Gray, 1991], i.e. it is (1) relevant, thus testing typical operations within the specific domain, (2) portable, i.e. executable on different platforms, (3) scalable, e.g. it is possible to run the benchmark on both small and very large data sets, and (4) it is understandable.

A comparison between benchmarks is shown in Table 7.1. The main difference between previous benchmarks and ours is that we rely on real data and real user queries, while most of the previous approaches rely on synthetic data. LUBM’s main drawback is that it solely relies on plain queries without SPARQL features such as `FILTER` or `REGEX`. In addition, its querying strategy (10 repeats of the same query)

<sup>2</sup><http://aksw.org/Projects/DBPSB>



allows for caching. SP<sup>2</sup>Bench relies on synthetic data and a small (25M triples) synthetic dataset for querying. The benchmark described in [Owens et al., 2008a] does not allow for testing the scalability of the stores, as the size of the data set is fixed. Finally, the BSBM data and queries are artificial and the data schema is very homogeneous and resembles a relational database.

In addition to general purpose RDF benchmarks it is reasonable to develop benchmarks for specific RDF data management aspects. One particular important feature in practical RDF triplestore usage scenarios (as was also confirmed by DBPSB) is full-text search on RDF literals. In [Minack et al., 2009] the LUBM benchmark is extended with synthetic scalable fulltext data and corresponding queries for fulltext-related query performance evaluation. RDF stores are benchmarked for basic fulltext queries (classic IR queries) as well as hybrid queries (structured and fulltext queries).

## 7.3. Data Quality Assessment

*Web data quality assessment frameworks.* There are a number of data quality assessment dimensions that have already been identified relevant to Linked Data, namely, accuracy, timeliness, completeness, relevancy, conciseness, consistency, to name a few [Bizer, 2007]. Additional quality criteria such as uniformity, versatility, comprehensibility, amount of data, validity, licensing, accessibility and performance were also introduced to be additional means of assessing the quality of LOD [Zaveri et al., 2013b]. Additionally, there are several efforts in developing data quality assessment frameworks in order to assess the data quality of LOD. These efforts are either semi-automated [Flemming, 2010], automated [Guéret et al., 2012] or manual [Bizer and Cyganiak, 2009, Mendes P.N., 2012].

Even though these frameworks introduce useful methodologies to assess the quality of a dataset, either the results are difficult to interpret, do not allow a user to choose the input dataset or require a considerable amount of user involvement.

*Concrete Web Data quality assessments* An effort to assess the quality of web data was undertaken in 2008 [Cafarella et al., 2008], where 14.1 billion HTML tables from Google’s general-purpose web crawl were analyzed in order to retrieve those tables that have high-quality relations. Additionally, there have been studies focused on assessing the quality of RDF data [Hogan et al., 2010] to report the errors occurring while publishing RDF data and the effects and means to improve the quality of structured data on the web. As part of an empirical study [Hogan et al., 2012] 4 million RDF/XML documents were analyzed, which provided insights into the level of conformance in these documents with respect to the Linked Data guidelines. Even though these studies accessed a vast amount of web or RDF/XML data, most of the analysis was performed automatically and therefore the problems arising due to contextual discrepancies were overlooked. Another study aimed to develop a framework for the DBpedia quality assessment [Kreis, 2011]. In this study, particular problems of the DBpedia extraction

framework were taken into account and integrated in the framework. However, only a small sample (75 resources) were assessed in this case and an older DBpedia version (2010) was analyzed.

*Crowdsourcing-based tasks* There are already a number of efforts which use crowdsourcing focused on a specific type of task. For example, crowdsourcing is used for entity linking or resolution [Demartini et al., 2012], quality assurance and resource management [Wang et al., 2012] or for enhancement of ontology alignments [Sarasua et al., 2012] especially in Linked Data. However, in our case, we did not submit tasks to the popular Internet marketplaces such as Amazon Mechanical Turk or CrowdFlower<sup>3</sup>. Instead, we used the intelligence of a large number of researchers who were particularly conversant with RDF to help assess the quality of one of the most important and interlinked dataset, DBpedia.

It is worth mentioning that the Linked Data life-cycle with the LOD2 stack [Auer et al., 2012] incorporates a *Quality Analysis* step. This step aims at developing techniques (components) for evaluating the quality based on concrete quality metric, such as provenance and context. *Sieve* component has a module called "Quality Assessment" module which relies on user-selected metadata as quality indicators in order to produce quality scores.

### 7.4. Fact Validation

There are three main areas related to the problem of fact validation research: The representation of provenance information in the Web of Data as well as work on trustworthiness and relation extraction. The problem of data provenance is a crucial issue in the Web of Data. While data extracted by the means of tools such as Hazy<sup>4</sup> and KnowItAll<sup>5</sup> can be easily mapped to primary provenance information, most knowledge sources were extracted by non-textual source and are more difficult to link with provenance information. In the work described in [Hartig and Zhao, 2010], Olaf Hartig and Jun Zhao developed a framework for provenance tracking. This framework provides the vocabulary required for representing and accessing provenance information on the web. It keeps track of who created a web entity, e.g. a webpage, when it was last modified etc. Recently, a W3C working group has been formed and released a set of specifications on sharing and representing provenance information<sup>6</sup>. Dividino et al. [Dividino et al., 2011] introduced an approach for managing several provenance dimensions, e.g. source, and timestamp. In their approach, they described an extension to the RDF called RDF<sup>+</sup>, which can efficiently work with provenance data. They provided a method to extend SPARQL query processing in a manner such that a specific SPARQL query can request meta knowledge without modifying the query itself.

---

<sup>3</sup><http://crowdfower.com/>

<sup>4</sup><http://hazy.cs.wisc.edu/hazy/>

<sup>5</sup><http://www.cs.washington.edu/research/knowitall/>

<sup>6</sup><http://www.w3.org/2011/prov/wiki/>

Theoharis et al. [Theoharis et al., 2011] argued how the implicit provenance data contained in a SPARQL query results can be used to acquire annotations for several dimensions of data quality. They detailed the abstract provenance models and how they are used in relational data, and how they can be used in semantic data as well. Their model requires the existence of provenance data in the underlying semantic data source. DeFacto uses the W3C provenance group standard for representing provenance information. Yet, unlike previous work, it directly tries to find provenance information by searching for confirming facts in trustworthy webpages.

The second research area related to fact validation is trustworthiness. Nakamura et al. [Nakamura et al., 2007] developed an efficient prototype for enhancing the search results provided by a search engine based on trustworthiness analysis for those results. They conducted a survey in order to determine the frequency at which the users accesses search engines and how much they trust the content and ranking of search results. They defined several criteria for trustworthiness calculation of search results returned by the search engine, such as topic majority. We adapted their approach for DeFacto and included it as one of the features for our machine learning techniques. [Pasternack and Roth, 2011a, Pasternack and Roth, 2011b] present an approach for computing the trustworthiness of web pages. To achieve this goal, the authors rely on a model based on hubs and authorities. This model allows to compute the trustworthiness of facts and websites by generating a  $k$ -partite network of pages and facts and propagating trustworthiness information across it. The approach returns a score for the trustworthiness of each fact. An older yet similar approach is that presented in [Yin et al., 2007]. Here, the idea is to create a 3-partite network of webpages, facts and objects and apply a propagation algorithm to compute weights for facts as well as webpages. The use of trustworthiness and uncertainty information on RDF data has been the subject of recent research (see e.g., [Hartig, 2008, Meiser et al., 2011]). Our approach differs from these approaches as it does not aim to evaluate the trustworthiness of facts expressed in natural language. In addition, it can deal with the broad spectrum of relations found on the Data Web.

Our fact validation approach is also related to relation extraction. Most tools that address this task rely on pattern-based approaches. Some early work on pattern extraction relied on supervised machine learning [Grishman and Yangarber, 1998]. Yet, such approaches demanded large amounts of training data, making them difficult to adapt to new relations. The subsequent generation of approaches to RE aimed at bootstrapping patterns based on a small number of input patterns and instances. For example, [Brin, 1999] presents the Dual Iterative Pattern Relation Expansion (DIPRE) and applies it to the detection of relations between authors and titles of books. This approach relies on a small set of seed patterns to maximize the precision of the patterns for a given relation while minimizing their error rate of the same patterns. Snowball [Agichtein and Gravano, 2000] extends DIPRE by a new approach to the generation of seed tuples. Newer approaches aim to either collect redundancy information (see e.g., [Yan et al., 2009] in an unsupervised manner

or to use linguistic analysis [Nguyen et al., 2007] to harvest generic patterns for relations.

# 8. Conclusions and Future Work

This chapter summarizes our research work, highlights our main contributions, and gives the general conclusion over the work. It then pinpoints the future directions in which we can move further to extend and broaden the research conducted in those areas.

## 8.1. Conclusions

Each direction of our research work has its own value and benefits. In each of the following subsections, we discuss the significance of each research direction in detail.

### 8.1.1. DBpedia Live Extraction

Due to the permanent update of Wikipedia articles, we also aim to update DBpedia accordingly. We proposed a framework for instantly retrieving updates from Wikipedia, extracting RDF data from them, and storing this data in a triplestore. Our new revision of the DBpedia Live extraction framework adds a number of features and particularly solves the following issues:

- MediaWiki templates in article abstracts are now rendered properly.
- Changes of mappings in the DBpedia mappings wiki are now retrospectively applied to all potentially affected articles.
- Updates can now be propagated easily, i.e. DBpedia Live mirrors can now get recent updates from our framework, in order to be kept in sync.

Many users can benefit from DBpedia, not only computer scientists. We have pointed out some directions how librarians and libraries can make use of DBpedia and how they can become part of the emerging Web of Data. We see a great potential for libraries to become centers of excellence in knowledge management on the Web of Data. As libraries supported the knowledge exchange through books in previous centuries, they now have the opportunity to extend their scope towards supporting the knowledge exchange through structured data and ontologies on the Web of Data. Due to the wealth and diversity of structured knowledge already available in DBpedia and other datasets on the Web of Data many other scientists, e.g. in life sciences, humanities, or engineering, would benefit a lot from such a development.

### **8.1.2. DBPSB**

We proposed the DBPSB benchmark for evaluating the performance of triplestores based on non-artificial data and queries. Our solution was implemented for the DBpedia dataset and tested with 4 different triplestores, namely Virtuoso, Sesame, Jena-TDB, and BigOWLIM. The main advantage of our benchmark over previous work is that it uses real RDF data with typical graph characteristics including a large and heterogeneous schema part. Furthermore, by basing the benchmark on queries asked to DBpedia, we intend to spur innovation in triplestore performance optimization towards scenarios, which are actually important for end users and applications. We applied query analysis and clustering techniques to obtain a diverse set of queries corresponding to feature combinations of SPARQL queries. Query variability was introduced to render simple caching techniques of triplestores ineffective.

The benchmarking results we obtained reveal that real-world usage scenarios can have substantially different characteristics than the scenarios assumed by prior RDF benchmarks. Our results are more diverse and indicate less homogeneity than what is suggested by other benchmarks. The creativity and inaptness of real users while constructing SPARQL queries is reflected by DBPSB and unveils for a certain triplestore and dataset size the most costly SPARQL feature combinations.

### **8.1.3. DeFacto**

DeFacto enables checking the validity of a given RDF triple. When given a test statement, it returns a confidence value for it as well as possible evidence for that statement. The evidence consists of a set of webpages, textual excerpts from those pages and meta-information on the pages. These text excerpts and the associated meta information allow the user to quickly get an overview over possible credible sources for the input statement. Instead of having to use search engines, browsing several webpages and looking for the relevant pieces of information in each webpage, using DeFacto the user can more efficiently review the presented information.

## **8.2. Future Work**

Each research area has its own direction(s), in which we can go move further, and expand the work.

### **8.2.1. DBpedia Live Extraction**

There are several directions in which we aim to extend the DBpedia Live framework:

*Support of other languages:* Currently, the framework supports only the English Wikipedia edition, and recently the Dutch DBpedia Live<sup>1</sup> has also been developed. We plan to extend our framework to include other languages as well. The main advantage of such a multi-lingual extension is that infoboxes within different Wikipedia editions cover different aspects of an entity at varying degrees of completeness. For instance, the Italian Wikipedia contains more knowledge about Italian cities and villages than the English one, while the German Wikipedia contains more structured information about people than the English edition. This leads to an increase of the quality of extracted data compared to knowledge bases that are derived from single Wikipedia editions. Moreover, this also helps in detecting inconsistencies across different Wikipedia and DBpedia editions.

*Wikipedia article augmentation:* Interlinking DBpedia with other data sources makes it possible to develop a MediaWiki extension that augments Wikipedia articles with additional information as well as media items (e.g. pictures and audio) from these sources. For instance, a Wikipedia page about a geographic location such as a city or a monument can be augmented with additional pictures from Web data sources such as Flickr or with additional facts from statistical data sources such as Eurostat or the CIA Factbook.

*Wikipedia consistency checking:* The extraction of different Wikipedia editions along with interlinking DBpedia with external Web knowledge builds the base for detecting inconsistencies in Wikipedia content. For instance, whenever a Wikipedia author edits an infobox within a Wikipedia article, the new content of the infobox could be checked against external data sources and information extracted from other language editions. Inconsistencies could be pointed out along with proposals on how to solve these inconsistencies. In this way, DBpedia can provide feedback to Wikipedia maintainers in order to keep Wikipedia data more consistent, which eventually may also lead to an increase of the quality of data in Wikipedia.

### 8.2.2. DBPSB

Several improvements can be envisioned in future work to cover a wider spectrum of features in DBPSB:

- Coverage of more SPARQL 1.1 features, e.g. reasoning and subqueries.
- Inclusion of further triplestores and continuous usage of the most recent DBpedia query logs.
- Testing of SPARQL update performance via DBpedia Live, which is modified several thousand times each day. In particular, an analysis of the dependency of query performance on the dataset update rate could be performed.

---

<sup>1</sup><http://live.nl.dbpedia.org/>

### **8.2.3. DeFacto**

DeFacto can be extended in manifold ways. First, BOA is able to detect natural-language representations of predicates in several languages. Thus, we could have the user choose the languages he/she understands and provide facts in several languages, therewith also increasing the portion of the Web that we search through. Furthermore, we could extend our approach to support data type properties. Moreover, DeFacto can be extended with adding the temporal dimension to it, i.e. DeFacto can be used to check if a statement was true during certain time span. On a grander scale, we aim to provide even lay users of knowledge bases with the means to check the quality of their data by using natural language input. This would support the transition from the Document Web to the Semantic Web by providing a further means to connect data and documents.



# A. DBpedia SPARQL Benchmark (DBPSB) Queries

## A.1. DBPSB Version 1

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX owl: <http://www.w3.org/2002/07/owl#>
4 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
5 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
6 PREFIX dc: <http://purl.org/dc/elements/1.1/>
7 PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
8 PREFIX dbpprop: <http://dbpedia.org/property/>
9 PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
10 PREFIX dbpedia2: <http://dbpedia.org/property/>
11 PREFIX yago: <http://dbpedia.org/class/yago/>
12 PREFIX umbelBus: <http://umbel.org/umbel/sc/>
13 PREFIX umbelCountry: <http://umbel.org/umbel/sc/>
14 PREFIX georss: <http://www.georss.org/georss/>

16 SELECT DISTINCT ?var1 WHERE { %%var%% rdf:type ?var1 . }

18 SELECT * WHERE { %%var%% ?var2 ?var1. filter(?var2 = dbp-prop:redirect ?var2 =
dbp-prop:redirect) }

20 SELECT ?var0 ?var1 ?var2 WHERE { ?var5 dbpedia-owl:thumbnail ?var0 . ?var3 rdf:
type dbpedia-owl:Person . ?var3 rdfs:label %%var%% . ?var3 foaf:page ?var1 .
OPTIONAL { ?var3 foaf:homepage ?var2 . } }

22 SELECT ?var0 ?var1 ?var2 ?var3 ?var4 WHERE { { %%var%% ?var0 ?var1 . ?var1 foaf:
name ?var3 . } UNION { ?var2 ?var0 %%var%% ; foaf:name ?var4 . } }

24 SELECT DISTINCT ?var0 ?var1 ?var2 WHERE { { ?var0 dbp-prop:series %%var1%% ; foaf
:name ?var1 ; rdfs:comment ?var2 ; rdf:type %%var0%% . } UNION { ?var0 dbp-prop:
series ?var3 . ?var3 dbp-prop:redirect %%var1%% . ?var0 foaf:name ?var1 ; rdfs:
comment ?var2 ; rdf:type %%var0%% . } }

26 SELECT DISTINCT ?var0 ?var1 ?var2 WHERE { ?var0 rdf:type <http://dbpedia.org/
class/yago/Company108058098> . ?var0 dbp-prop:numEmployees ?var1 FILTER ( xsd:
integer(?var1) >= %%var%% ) . ?var0 foaf:homepage ?var2 . }

28 SELECT DISTINCT ?var0 ?var1 ?var2 ?var3 ?var5 ?var6 ?var7 ?var10 WHERE { ?var0
rdfs:comment ?var1. ?var0 foaf:page %%var%% OPTIONAL{?var0 skos:subject ?var6}
OPTIONAL {?var0 dbp-prop:industry ?var5} OPTIONAL {?var0 dbpedia2:location ?var2}
OPTIONAL {?var0 dbpedia2:locationCountry ?var3} OPTIONAL {?var0 dbpedia2:
locationCity ?var9; dbp-prop:manufacturer ?var0} OPTIONAL {?var0 dbpedia2:
products ?var11; dbp-prop:model ?var0} OPTIONAL {?var0 <http://www.georss.org/
georss/point> ?var10} OPTIONAL {?var0 rdf:type ?var7}}

30 SELECT ?var0 ?var1 WHERE { { ?var0 rdf:type %%var1%%. ?var0 dbpedia2:population ?
var1. FILTER (xsd:integer(?var1) > %%var0%%) } UNION { ?var0 rdf:type %%var1%%. ?
var0 dbpedia2:populationUrban ?var1. FILTER (xsd:integer(?var1) > %%var0%%) } }
```

```

32 SELECT * WHERE { ?var0 a dbp-owl:Settlement; rdfs:label %%var%% . ?var1 a dbp-owl:
:Airport. {?var1 dbp-owl:city ?var0} UNION {?var1 dbp-owl:location ?var0} {?var1
dbp-prop:iata ?var2.} UNION {?var1 dbp-owl:iataLocationIdentifier ?var2. }
OPTIONAL { ?var1 foaf:homepage ?var1_home. } OPTIONAL { ?var1 dbp-prop:nativename
?var1_name.} }

34 SELECT DISTINCT ?var0 { ?var1 foaf:page ?var0. ?var1 rdf:type dbp-owl:
SoccerPlayer . ?var1 dbp-prop:position ?var2 . ?var1 dbp-prop ?var3 . ?var3 dbp-
owl:capacity ?var4 . ?var1 <http://dbpedia.org/ontology/birthPlace> ?var5 . ?var5
?var6 ?var7. OPTIONAL {?var1 dbp-owl:number ?var8.} Filter (?var6 = dbp-prop:
populationEstimate ?var6 = dbp-prop:populationCensus ?var6 = dbp-prop:statPop )
Filter (xsd:integer(?var7) > %%var1%% ) . Filter (xsd:integer(?var4) < %%var0%%
) . Filter (?var2 = 'Goalkeeper'@en ?var2 = <http://dbpedia.org/resource/
Goalkeeper_%28association_football%29> ?var2 = <http://dbpedia.org/resource/
Goalkeeper_%28football%29>) }

36 SELECT distinct ?var0 ?var1 ?var2 WHERE { {%%var%% dbp-prop:subsid ?var0 OPTIONAL
{?var2 %%var%% dbp-prop:parent} OPTIONAL{%%var%% dbp-prop:divisions ?var1}} UNION
{?var2 %%var%% dbp-prop:parent OPTIONAL{%%var%% dbp-prop:subsid ?var0} OPTIONAL
{%%var%% dbp-prop:divisions ?var1}} UNION {%%var%% dbp-prop:divisions ?var1
OPTIONAL{%%var%% dbp-prop:subsid ?var0} OPTIONAL{?var2 %%var%% dbp-prop:parent}}
}

38 SELECT DISTINCT ?var0 WHERE { ?var1 rdf:type dbp-owl:Person . ?var1 dbp-owl:
nationality ?var2 . ?var2 rdfs:label ?var0 . ?var1 rdfs:label %%var%% . FILTER (
LANG(?var0) = 'en') }

40 SELECT * WHERE {{ %%var%% rdfs:comment ?var0. FILTER (LANG(?var0) = 'en')}} UNION
{%%var%% foaf:depiction ?var1} UNION {%%var%% foaf:homepage ?var2}}

42 SELECT ?var0 ?var1 ?var2 ?var3 WHERE { ?var3 skos:subject %%var%% . ?var3 foaf:
name ?var0 . OPTIONAL { ?var3 rdfs:comment ?var1 . FILTER (LANG(?var1) = 'en') .
} OPTIONAL { ?var3 rdfs:comment ?var2 . FILTER (LANG(?var2) = 'de') . } }

44 SELECT DISTINCT ?var0 ?var1 WHERE { ?var0 rdf:type %%var%% ; rdfs:label ?var1 .
FILTER regex(str(?var1), 'pes', 'i') }

46 SELECT DISTINCT ?var0 ?var1 ?var2 ?var3 WHERE { %%var%% ?var1 ?var3 . OPTIONAL {?
var3 rdfs:label ?var2} . FILTER(langMatches(LANG(?var2),'EN')(! langMatches(LANG
(?var2),'*')) . FILTER(langMatches(LANG(?var3),'EN')(! langMatches(LANG(?var3),
'*')) . OPTIONAL {?var1 rdfs:label ?var0}}

48 SELECT DISTINCT ?var0 ?var1 { {?var0 skos:subject %%var%%.} UNION {?var0 skos:
subject <http://dbpedia.org/resource/Category:Prefectures_in_France> .} UNION {?
var0 skos:subject <http://dbpedia.org/resource/Category:German_state_capitals> .}
?var0 rdfs:label ?var1. FILTER (LANG(?var1)='fr') }

50 SELECT ?var0 ?var1 ?var2 WHERE { { %%var%% ?var0 ?var1. FILTER ( (STR(?var0) = '
http://www.w3.org/2000/01/rdf-schema#label' && LANG(?var1) = 'en') (STR(?var0) =
'http://dbpedia.org/ontology/abstract' && LANG(?var1) = 'en') (STR(?var0) = '
http://www.w3.org/2000/01/rdf-schema#comment' && LANG(?var1) = 'en') (STR(?var0)
!= 'http://dbpedia.org/ontology/abstract' && STR(?var0) != 'http://www.w3.org
/2000/01/rdf-schema#comment' && STR(?var0) != 'http://www.w3.org/2000/01/rdf-
schema#label') ) } UNION { ?var2 ?var0 %%var%% FILTER ( STR(?var0) = 'http://
dbpedia.org/ontology/owner' STR(?var0) = 'http://dbpedia.org/property/redirect'
) } }

52 SELECT ?var1 WHERE { { ?var1 rdfs:label %%var%% } UNION { ?var1 rdfs:label %%var
%% }. FILTER(regex(str(?var1),'http://dbpedia.org/resource/') regex(str(?var1),'
http://dbpedia.org/ontology/') regex(str(?var1),'http://www.w3.org/2002/07/owl')
regex(str(?var1),'http://www.w3.org/2001/XMLSchema') regex(str(?var1),'http://
www.w3.org/2000/01/rdf-schema') regex(str(?var1),'http://www.w3.org/1999/02/22-
rdf-syntax-ns')) }

54 SELECT * WHERE { ?var0 a dbp-owl:PopulatedPlace; dbp-owl:abstract ?var1; rdfs:
label ?var2; geo:lat ?var3; geo:long ?var4. {?var0 rdfs:label %%var%%.} UNION { ?

```

```

var5 dbp-prop:redirect ?var0; rdfs:label %%var%%. } OPTIONAL { ?var0 foaf:
depiction ?var6 } OPTIONAL { ?var0 foaf:homepage ?var7 } OPTIONAL { ?var0 dbp-owl
:populationTotal ?var8 } OPTIONAL { ?var0 dbp-owl:thumbnail ?var9 } FILTER (
LANGMatches( LANG(?var1), 'de') && langMatches( LANG(?var2), 'de'))}
56 SELECT * WHERE { %%var%% dbp-prop:redirect ?var0 . }
58 SELECT ?var0 WHERE { ?var1 <http://xmlns.com/foaf/0.1/homepage> ?var0 . ?var1 <
http://www.w3.org/1999/02/22-rdf-syntax-ns#type> %%var%% . }
60 SELECT ?var0 WHERE { ?var1 rdf:type dbp-owl:Person . ?var1 rdfs:label %%var%% . ?
var1 foaf:page ?var0 . }
62 SELECT * WHERE { ?var1 a dbp-owl:Organisation . ?var2 dbp-owl:foundationPlace %%
var0%% . ?var4 dbp-owl:developer ?var2 . ?var4 a %%var1%% . }
64 SELECT ?var0 ?var1 ?var2 ?var3 WHERE { ?var6 rdf:type %%var%%. ?var6 dbp-prop:
name ?var0. ?var6 dbp-prop:pages ?var1. ?var6 db-pprop:isbn ?var2. ?var6 dbp-prop
:author ?var3.}

```

## A.2. DBPSB Version 2

```

1 SELECT DISTINCT ?var0 where { %%var%% skos:subject ?var0 .}
3 SELECT DISTINCT ?var0 WHERE { %%var%% dbpprop:redirect ?var0 . }
5 SELECT ?var0 ?var1 ?var2 WHERE { ?var3 dbpedia-owl:thumbnail ?var0 . ?var3 rdf:
type dbpedia-owl:Person . ?var3 rdfs:label %%var%% . ?var3 foaf:page ?var1 .
OPTIONAL { ?var3 foaf:homepage ?var2 .} . }
7 SELECT ?var0 ?var1 WHERE { { %%var%% skos:subject ?var1 . } UNION { %%var%% skos:
subject ?var0 . ?var3 skos:broader ?var1 . } UNION { %%var%% skos:subject ?var2 .
?var1 skos:broader ?var0 . } }
9 SELECT ?var0 ?var1 ?var2 ?var3 WHERE { ?var4 dbpedia2:birthPlace %%var%% . ?var3
dbpedia-owl:birthDate ?var1 . ?var3 foaf:name ?var0 . ?var3 dbpedia-owl:deathDate
?var2 FILTER (?var1 < '1900-01-01'^^xsd:date) . }
11 SELECT * WHERE { %%var%% ?var0 ?var1. filter(?var0 = dbpedia2:redirect) }
13 SELECT DISTINCT ?var0 WHERE { { %%var%% dbpprop:writer ?var0 . } UNION { %%var%%
dbpprop:executiveProducer ?var0 . } UNION { %%var%% dbpprop:creator ?var0 . }
UNION { %%var%% dbpprop:starring ?var0 . } UNION { %%var%% dbpprop:
executiveProducer ?var0 . } UNION { %%var%% dbpprop:guest ?var0 . } UNION { %%var
%% dbpprop:director ?var0 . } UNION { %%var%% dbpprop:producer ?var0 . } UNION {
%%var%% dbpprop:series ?var0 . } }
15 SELECT ?var0 WHERE { %%var%% dbpedia-owl:abstract ?var0. FILTER langMatches(LANG
(?var0), 'en') }
17 SELECT ?var0 ?var1 WHERE { ?var2 rdfs:label %%var%% . ?var0 skos:broader ?var2 .
?var0 rdfs:label ?var1 . FILTER langMatches( LANG(?var1), 'EN') }
19 SELECT * WHERE { ?var0 a %%var%% . ?var0 foaf:givenName ?var1 FILTER regex(?var1,
'^A'). }
21 SELECT ?var0 WHERE { %%var%% a ?var1 . OPTIONAL { ?var1 rdfs:subClassOf ?var0 } .
FILTER (!bound(?var1)) . FILTER (?var0 != <http://dbpedia.org/ontology/Resource
>) . }
23 SELECT ?var0 ?var1 ?var2 ?var3 WHERE { ?var3 skos:subject %%var%% . ?var3 foaf:
name ?var0 . OPTIONAL { ?var3 rdfs:comment ?var1 . FILTER (LANG(?var1) = 'en') .
} OPTIONAL { ?var3 rdfs:comment ?var2 . FILTER (LANG(?var2) = 'de') . } }

```

```

25 SELECT DISTINCT ?var0 ?var1 WHERE { ?var2 dbpedia-owl:influenced %%var%% . ?var2
foaf:page ?var0 . ?var2 rdfs:label ?var1 filter (LANG(?var1)='en') }

27 SELECT DISTINCT ?var0 WHERE { %%var%% dbpedia2:instrument ?var0 FILTER (
langMatches (LANG(?var0), 'EN') ) }

29 SELECT * WHERE {{ %%var%% rdfs:comment ?var0. FILTER (LANG(?var0) = 'en')}} UNION
{%%var%% foaf:depiction ?var1} UNION {%%var%% foaf:homepage ?var2}}

31 SELECT DISTINCT ?var0 ?var1 WHERE { ?var2 rdf:type %%var%% OPTIONAL { ?var0 rdfs:
label ?var1 . FILTER(LANG(?var1) = 'en') . } }

33 SELECT ?var0 ?var1 ?var2 WHERE { { %%var%% ?var0 ?var1. FILTER ( (STR(?var0) = '
http://www.w3.org/2000/01/rdf-schema#label' && LANG(?var1) = 'en') (STR(?var0) =
'http://dbpedia.org/ontology/abstract' && LANG(?var1) = 'en') (STR(?var0) = '
http://www.w3.org/2000/01/rdf-schema#comment' && LANG(?var1) = 'en') (STR(?var0)
!= 'http://dbpedia.org/ontology/abstract' && STR(?var0) != 'http://www.w3.org
/2000/01/rdf-schema#comment' && STR(?var0) != 'http://www.w3.org/2000/01/rdf-
schema#label') ) } UNION { ?var2 ?var0 %%var%% FILTER ( STR(?var0) = 'http://
dbpedia.org/ontology/owner' STR(?var0) = 'http://dbpedia.org/property/redirect'
) } }

35 SELECT ?var1 WHERE { %%var%% rdfs:label ?var1 .}

37 SELECT * WHERE { ?var0 rdfs:label %%var%% ; rdf:type ?var1 . }

39 SELECT ?var0 WHERE { ?var1 rdf:type dbpedia-owl:Person . ?var1 rdfs:label %%var%%
. ?var1 foaf:page ?var0 . }

```

## B. Curriculum Vitae

### Personal Data

**Birth date:** November 15th, 1980

**Birth place:** Cairo, Egypt

**Nationality:** Egyptian

**Marital status:** Married

---

### Education

2010 – Present      Leipzig University      Leipzig, Germany  
Ph.D., Faculty of Mathematics and Computer Science, Department of Computer Science.

Thesis title: **Efficient Extraction and Query Benchmarking of Wikipedia Data**

2002 – 2006      Ain Shams University      Cairo, Egypt  
M.Sc., Faculty of Computer and Information Sciences, Computer Science Department.

Thesis title: **Intelligent Technique for Computer Virus Detection.**

1997 – 2001      Ain Shams University      Cairo, Egypt  
B.Sc., Faculty of Computer and Information Sciences, Computer Science Department.

Grade: **Excellent with honor degree.**

1994 – 1997      El Tawfekeya secondary school      Cairo, Egypt  
General secondary school certificate.

Grade: **98.25%.**

---

### Awards and Honors

- **Best research paper award at The 10th International Semantic Web Conference (ISWC2011)**, for paper "DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data".
- **Spotlight paper at The 11th International Semantic Web Conference (ISWC2012)**, for paper "DeFacto - Deep Fact Validation".

- **Outstanding paper award at The Electronic Library and Information Systems Journal**, for paper "DBpedia and the Live Extraction of Structured Data from Wikipedia".
- **DAAD scholarship**, for obtaining the Ph.D. Degree from Germany.

---

## Research Interests

- Semantic Web.
- Linked Data.
- Ontology Engineering.
- Object Oriented Analysis and Design.

---

## Publications

1. Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, **Mohamed Morsey**, Patrick van Kleef, Sören Auer, Christian Bizer. DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. Submitted to Semantic Web Journal.
2. Amrapali Zaveri, Dimitris Kontokostas, Mohamed A. Sherif, Lorenz Bühmann, **Mohamed Morsey**, Sören Auer, and Jens Lehmann. User-driven quality evaluation of dbpedia. To appear in Proceedings of 9th International Conference on Semantic Systems, I-SEMANTICS '13, Graz, Austria, September 4-6, 2013. ACM, 2013.
3. Jens Lehmann and Daniel Gerber and **Mohamed Morsey**, Axel-Cyrille Ngonga Ngomo. "DeFacto - Deep Fact Validation". In The 11th International Semantic Web Conference (ISWC2012), 2012.
4. **Mohamed Morsey**, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. "Usage-Centric Benchmarking of RDF Triple Stores". In the 26th AAAI Conference on Artificial Intelligence (AAAI 2012), 2012.
5. **Mohamed Morsey**, Jens Lehmann, Sören Auer, Claus Stadler, and Sebastian Hellmann. "DBpedia and the Live Extraction of Structured Data from Wikipedia". At The Electronic Library and Information Systems Journal, 2012.

- 
6. **Mohamed Morsey**, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo.  
"DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data". In The 10th International Semantic Web Conference (ISWC2011), 2011.
  7. **M. Mabrouk**, M. Siam, M. Hashem, and S. Arafat.  
"Mobile Agents for Computer Virus Detection", 2nd International Conference on Intelligent Computing and Information Systems (ICICIS'2005), Egypt, 2005.
  8. **M. Mabrouk**, M. Siam, M. Hashem, and S. Arafat.  
"Data Mining for Computer Virus Detection", 2nd International Conference on Intelligent Computing and Information Systems (ICICIS'2005), Egypt, 2005.
  9. **M. Mabrouk**, M. Siam, M. Hashem, and S. Arafat.  
"Neural Networks for Computer Virus Detection", 4th WSEAS International Conference on Neural Networks and Applications (WSEAS NNA 2003), Greece, 2003.

---

### Relevant Experience

- 2001 – Present      Teaching assistant, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt.
- 2002 – 2004      Teaching assistant, National Institute for Civil Aviation, Cairo, Egypt.
- 2002 – 2004      Teaching assistant, Higher Institute for Specialized Technological Studies, Cairo, Egypt.

---

### Languages Skills

- English: TOEFL iBT test with score of 94.
- German: DSH test grade 1.
- Arabic: Mother tongue.

---

### Technical and Programming Skills

- **Programming Languages Skills:**

- .NET frameworks 4.0, 3.5, 3.0, 2.0, 1.1, and 1.0, level is excellent, 10 years of experience.
- Visual C#, level is excellent, 9 years of experience.
- Visual C++ and Visual C++. Net, level is excellent, 9 years of experience.
- Java, level is excellent, 5 years of experience.
- Assembly Language, level is good, 2 years of experience.
- Delphi, level is moderate, 1 year of experience.

● **Web Programming:**

- ASP.NET 4.0.
- ASP.NET 3.5.
- ASP.NET 3.0.
- ASP.NET 2.0.
- ASP.NET 1.1.
- ASP.NET 1.0.
- JSP.
- ASP.
- AJAX.
- Java Script.
- VB script.

● **Database Systems:**

- Microsoft SQL Server 2008.
- Microsoft SQL Server 2005.
- Microsoft SQL Server 2000.
- MySQL.
- ORACLE 8.0i.

● **Other Programming Skills:**

- WPF.
- WWF.
- WCF.
- MFC.
- DotNetNuke Programming.
- Database programming using ADO, ADO.NET.



- 
- ATL Programming.
  - COM Programming.
  - DirectX 9.0 Programming.
  - ActiveX Programming.
  - SDK Programming.
- **Programming Certificates:**
    - MCAD using C#.
    - Microsoft certificate of "Web Application Development Using Microsoft Visual InterDev 6.0" using ASP.

---

## Projects

- **DBpedia Live:**

DBpedia project aims at extracting structured knowledge from Wikipedia, and making it free available on the Web. The main objective of DBpedia Live is to keep DBpedia always in synchronization with Wikipedia. It reads a continuous stream-of-updates from Wikipedia, containing the changed Wikipedia articles, process it on-the-fly and reprocesses those pages, in order to keep the DBpedia data always up to date. It is available here <http://live.dbpedia.org>. Implemented in Java, and Scala.
- **DBpedia SPARQL Benchmark (DBPSB):**

DBPSB is a general SPARQL benchmark procedure, which we apply to the DBpedia knowledge base. The benchmark is based on query-log mining, clustering and SPARQL feature analysis. In contrast to other benchmarks, we perform measurements on actually posed queries against existing RDF data. It is available here <http://aksw.org/Projects/DBPSB>. Implemented in Java.
- **Deep Fact Validation (DeFacto):**

DeFacto (Deep Fact Validation) is an algorithm for validating statements by finding confirming sources for it on the web. It takes a statement (such as "Jamaica Inn was directed by Alfred Hitchcock") as input and then tries to find evidence for the truth of that statement by searching for information in the web. It is available here <http://aksw.org/Projects/DeFacto.html>. Implemented in Java.
- **eVoucher System:**

Building and developing a system for managing and delivering vouchers to the user. I have worked in this system while I was in Technowireless software company. Implemented in C#, and ASP.NET.

- **Web Site Builder:**  
Building and developing a system for constructing web sites on the fly, i.e. the user determines his/her the requirements of the website and the system builds the HTML files for him/her. I have worked in this system while I was in ILD online software company. Implemented using classic ASP.
- **Speaker Recognition System (Graduation Project):**  
Building and developing a system that is able to identify the speaker, i.e. voice print system. Implemented in Visual C++.
- **Speech Recognition System):**  
Developing a package to accept a wave sound and apply to it several preprocessing techniques. Implemented in Visual C++.
- **Image Processing Package:**  
Developing a package to process an image including filtering, edge detection, and mathematical and logical operations. Implemented in Visual C++.
- **Graphics Package:**  
Developing a package that enables the user to draw different shapes such as lines and circles, and applying different graphics operations on them like translation and rotation. Implemented in Visual C++.
- **Students Affairs System:**  
Building a system to accept student details and applying the different operations on them like calculating the total marks of each student, and determining the failed students. Implemented in Visual Prolog.

# List of Tables

2.1. Sample RDF statements. . . . .	10
3.1. Overview of DBpedia extractors. . . . .	21
3.2. Datasets linked from DBpedia . . . . .	30
3.3. Top 10 datasets in Sindice ordered by the number of links to DBpedia. . . . .	32
3.4. Sindice summary statistics for incoming links to DBpedia. . . . .	32
3.5. Top 10 datasets by incoming links in Sindice. . . . .	32
5.1. Data quality dimensions, categories and sub-categories identified in the DBpedia resources. Detectable (column D) means problem detection can be automated. Fixable (column F) means the issue is solvable by amending either the extraction framework (E), the mappings wiki (M) or Wikipedia (W). The last column marks the dataset specific subcategories. . . . .	48
5.2. Overview of the manual quality evaluation. . . . .	53
5.3. Detected number of problem for each of the defined quality problems. IT = Incorrect triples, DR = Distinct resources, AT = Affected triples. . . . .	54
5.4. Results of the semi-automatic evaluation. The table shows the total number of properties that have been suggested to have the given characteristic by Step I of the semi-automatic methodology, the number of properties that would lead to at least one violation when applying the characteristic, the number of properties where the characteristic is meaningful (manually evaluated) and some metrics for the number of violations. . . . .	55
5.5. Classification results for trainings sets <i>domain</i> and <i>range</i> . . . . .	62
5.6. Classification results for trainings sets <i>domain-range</i> and <i>property</i> . . . . .	62
5.7. Classification results for trainings sets <i>random</i> and <i>20%mix</i> . . . . .	62
6.1. Statistical analysis of DBPSB datasets. . . . .	67
6.2. Queries per second (QpS), geometric mean of query runtime in milliseconds (GM), and standard deviation of query runtime in milliseconds (SD), for the 10% dataset, and 50% dataset respectively of DBPSB version 1. . . . .	78

6.3. Queries per second (QpS), geometric mean of query runtime in milliseconds (GM), and standard deviation of query runtime in milliseconds (SD), for the 100% dataset, and 200% dataset of DBPSB version 1. . . . .	79
6.4. Queries per second (QpS), geometric mean of query runtime in milliseconds (GM), and standard deviation of query runtime in milliseconds (SD), for the 10% dataset, and 50% dataset of DBPSB version 2. . . . .	83
6.5. Queries per second (QpS), geometric mean of query runtime in milliseconds (GM), and standard deviation of query runtime in milliseconds (SD), for the 100% dataset of DBPSB version 2. . . .	84
7.1. Comparison of different RDF benchmarks. . . . .	88

# List of Figures

2.1.	RDF statements represented as a directed graph. . . . .	8
2.2.	Small knowledge base about William Shakespeare represented as a graph. . . . .	10
2.3.	Sample N-Triples format. . . . .	11
2.4.	Sample RDF/XML format. . . . .	11
2.5.	Sample N3 format. . . . .	12
2.6.	Sample ontology snapshot taken from DBpedia ontology. . . . .	13
2.7.	OWL representation of a part our ontology in N-Triples format. . . . .	15
2.8.	SPARQL query to get the spouse of Shakespeare’s child. . . . .	15
3.1.	Mediawiki infobox syntax for Algarve (left) and rendered infobox (right). . . . .	18
3.2.	Overview of DBpedia extraction framework. . . . .	19
3.3.	Depiction of the mapping from the Greek and English Wikipedia templates about books to the same DBpedia Ontology class [Kontokostas et al., 2012]. . . . .	25
3.4.	Snapshot of a part of the DBpedia ontology. . . . .	28
3.5.	Growth of the DBpedia ontology . . . . .	28
3.6.	SPARQL query to compare funding per year (from FTS) and country with the gross domestic product of that country. . . . .	29
4.1.	General DBpedia Live system architecture. . . . .	34
4.2.	Mapping for infobox of a book. . . . .	36
4.3.	Number of daily requests sent to the DBpedia Live for a) SPARQL queries and b) synchronization requests from August 2012 until January 2013 . . . . .	42
5.1.	Workflow of the data quality assessment methodology. . . . .	45
5.2.	Overview of Deep Fact Validation. . . . .	55
5.3.	Input data for Defacto.. . . . .	56
6.1.	Sample query with placeholder. . . . .	71
6.2.	Sample auxiliary query returning potential values a placeholder can assume. . . . .	71
6.3.	QMpH for all triplestores of DBPSB version 1. . . . .	73
6.4.	Geometric mean of QpS of DBPSB version 1. . . . .	74
6.5.	QMpH for all triplestores of DBPSB version 2. . . . .	74

6.6. Geometric mean of QpS of DBPSB version 2. . . . .	74
6.7. Queries per Second (QpS) of DBPSB version 1 for all triplestores for 10%, 50%, 100%, and 200%. . . . .	75
6.8. Queries per Second (QpS) of DBPSB version 2 for all triplestores for 10%, 50% and 100%. . . . .	76
6.9. Comparison of triple store scalability between BSBM V2, BSBM V3, DBPSB. . . . .	81

# Bibliography

- [Agichtein and Gravano, 2000] Agichtein, E. and Gravano, L. (2000). Snowball: Extracting relations from large plain-text collections. In *In Proceedings of the 5th ACM International Conference on Digital Libraries*, pages 85–94.
- [Auer et al., 2012] Auer, S., Bühmann, L., Dirschl, C., Erling, O., Hausenblas, M., Isele, R., Lehmann, J., Martin, M., Mendes, P. N., van Nuffelen, B., Stadler, C., Tramp, S., and Williams, H. (2012). Managing the life-cycle of linked data with the lod2 stack. In *Proceedings of International Semantic Web Conference (ISWC 2012)*. 22
- [Auer and Lehmann, 2007] Auer, S. and Lehmann, J. (2007). What have Innsbruck and Leipzig in common? extracting semantics from wiki content. In *Proceedings of the 4th European conference on The Semantic Web: Research and Applications*, volume 4519 of *Lecture Notes in Computer Science*, pages 503–517, Berlin / Heidelberg. Springer.
- [Auer et al., 2009] Auer, S., Lehmann, J., and Hellmann, S. (2009). LinkedGeo-Data - adding a spatial dimension to the web of data. In *Proc. of 8th International Semantic Web Conference (ISWC)*.
- [Bechhofer et al., 2004] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2004). OWL Web Ontology Language Reference. Technical report, W3C, <http://www.w3.org/TR/owl-ref/>.
- [Beckett, 2004] Beckett, D. (2004). RDF/XML syntax specification (revised). W3C recommendation, W3C.
- [Belleau et al., 2008] Belleau, F., Nolin, M.-A., Tourigny, N., Rigault, P., and Morissette, J. (2008). Bio2rdf: Towards a mashup to build bioinformatics knowledge systems. *Journal of Biomedical Informatics*, 41(5):706–716.
- [Berners-Lee and Connolly, 2011] Berners-Lee, T. and Connolly, D. (2011). Notation3 (N3): A readable RDF syntax. Technical report, W3C.
- [Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.

- [Bishop et al., 2011] Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., and Velkov, R. (2011). OWLIM: A family of scalable semantic repositories. *Semantic Web*, 2(1):1–10.
- [Bizer, 2007] Bizer, C. (2007). *Quality-Driven Information Filtering in the Context of Web-Based Information Systems*. PhD thesis, Freie Universität.
- [Bizer and Cyganiak, 2009] Bizer, C. and Cyganiak, R. (2009). Quality-driven information filtering using the wiqa policy framework. *Web Semantics*, 7(1):1 – 10.
- [Bizer and Schultz, 2009] Bizer, C. and Schultz, A. (2009). The Berlin SPARQL Benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24.
- [Bollacker et al., 2008] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, New York, NY, USA. ACM.
- [Brickley and Guha, 2004] Brickley, D. and Guha, R. V. (2004). RDF Vocabulary Description Language 1.0: RDF Schema. Technical report, W3C.
- [Brin, 1999] Brin, S. (1999). Extracting patterns and relations from the world wide web. In *Selected papers from the International Workshop on The World Wide Web and Databases*, pages 172–183, London, UK. Springer-Verlag.
- [Broekstra et al., 2002] Broekstra, J., Kampman, A., and van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying RDF and RDF schema. In *Proceedings of the 2nd International Semantic Web Conference (ISWC2002)*, number 2342 in Lecture Notes in Computer Science (LNCS) 7603, pages 54–68. Springer.
- [Bühmann and Lehmann, 2012] Bühmann, L. and Lehmann, J. (2012). Universal OWL axiom enrichment for large knowledge bases. In *Proceedings of the 18th international conference on Knowledge Engineering and Knowledge Management, EKAW'12*.
- [Cafarella et al., 2008] Cafarella, M. J., Halevy, A. Y., Wang, D. Z., Wu, E., and Zhang, Y. (2008). Webttables: exploring the power of tables on the web. *Proc. VLDB Endow.*, 1(1):538–549.
- [Clark et al., 2008] Clark, K. G., Feigenbaum, L., and Torres, E. (2008). SPARQL Protocol for RDF. World Wide Web Consortium, Recommendation REC-rdf-sparql-protocol-20080115.
- [Dave and Berners-Lee, 2011] Dave, D. and Berners-Lee, T. (2011). Turtle - Terse RDF Triple Language. Technical report, W3C.



- [Demartini et al., 2012] Demartini, G., Difallah, D., and Cudré-Mauroux, P. (2012). Zencrowd: Leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *21st International Conference on World Wide Web WWW 2012*, pages 469 – 478.
- [Dividino et al., 2011] Dividino, R., Sizov, S., Staab, S., and Schueler, B. (2011). Querying for provenance, trust, uncertainty and other meta knowledge in rdf. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3).
- [Erling and Mikhailov, 2007] Erling, O. and Mikhailov, I. (2007). RDF support in the virtuoso DBMS. In Auer, S., Bizer, C., Müller, C., and Zhdanova, A. V., editors, *Proceedings of the 1st Conference on Social Semantic Web*, volume 113 of *LNI*, pages 59–68. GI.
- [Flemming, 2010] Flemming, A. (2010). Quality characteristics of linked data publishing datasources. Master’s thesis, Humboldt-Universität of Berlin.
- [Gerber and Ngomo, 2012] Gerber, D. and Ngomo, A.-C. N. (2012). Extracting multilingual natural-language patterns for rdf predicates. In *Proceedings of the 18th international conference on Knowledge Engineering and Knowledge Management, EKAW’12*, pages 87–96, Berlin, Heidelberg. Springer-Verlag.
- [Gerber and Ngonga Ngomo, 2011] Gerber, D. and Ngonga Ngomo, A.-C. (2011). Bootstrapping the linked data web. In *1st Workshop on Web Scale Knowledge Extraction @ ISWC 2011*.
- [Grant and Beckett, 2004] Grant, J. and Beckett, D. (2004). RDF test cases. W3C recommendation, World Wide Web Consortium.
- [Gray, 1991] Gray, J., editor (1991). *The Benchmark Handbook for Database and Transaction Systems (1st Edition)*. Morgan Kaufmann.
- [Grishman and Yangarber, 1998] Grishman, R. and Yangarber, R. (1998). Nyu: Description of the Proteus/Pet system as used for MUC-7 ST. In *In Proceedings of the Seventh Message Understanding Conference (MUC-7)*. Morgan Kaufmann.
- [Guéret et al., 2012] Guéret, C., Groth, P. T., Stadler, C., and Lehmann, J. (2012). Assessing linked data mappings using network measures. In *Proceedings of the 9th Extended Semantic Web Conference*, volume 7295 of *Lecture Notes in Computer Science*, pages 87–102. Springer.
- [Hartig, 2008] Hartig, O. (2008). Trustworthiness of data on the web. In *Proceedings of the STI Berlin & CSW PhD Workshop*.
- [Hartig, 2009] Hartig, O. (2009). Provenance information in the web of data. In *In Proceedings of the Linked Data on the Web (LDOW) Workshop at WWW*.

- [Hartig and Zhao, 2010] Hartig, O. and Zhao, J. (2010). Publishing and consuming provenance metadata on the web of linked data. In *Proceedings of 3rd International Provenance and Annotation Workshop*, pages 78–90.
- [Heflin, 2004] Heflin, J. (2004). OWL Web Ontology Language Use Cases and Requirements. Technical report, W3C.
- [Hellmann et al., 2009] Hellmann, S., Lehmann, J., and Auer, S. (2009). Learning of OWL class descriptions on very large knowledge bases. *International Journal on Semantic Web and Information Systems*, 5(2):25–48.
- [Hellmann et al., 2011] Hellmann, S., Lehmann, J., and Auer, S. (2011). Learning of owl class expressions on very large knowledge bases and its applications. In Semantic Services, I. and Concepts, W. A. E., editors, *Learning of OWL Class Expressions on Very Large Knowledge Bases and its Applications*, chapter 5, pages 104–130. IGI Global.
- [Hoffart et al., 2010] Hoffart, J., Suchanek, F. M., Berberich, K., and Weikum, G. (2010). YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia. Research Report MPI-I-2010-5-007, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany.
- [Hogan et al., 2010] Hogan, A., Harth, A., Passant, A., Decker, S., and Polleres, A. (2010). Weaving the pedantic web. In *Linked Data on the Web Workshop (LDOW2010) at WWW'2010*.
- [Hogan et al., 2012] Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., and Decker, S. (2012). An empirical survey of linked data conformance. *Journal of Web Semantics*, 14:14–44.
- [Iglesias and Lehmann, 2011] Iglesias, J. and Lehmann, J. (2011). Towards integrating fuzzy logic capabilities into an ontology-based inductive logic programming framework. In *Proc. of the 11th International Conference on Intelligent Systems Design and Applications (ISDA)*.
- [Knuth et al., 2012] Knuth, M., Hercher, J., and Sack, H. (2012). Collaboratively patching linked data. *Proceedings of 2nd International Workshop on Usage Analysis and the Web of Data (USEWOD 2012), co-located with the 21st International World Wide Web Conference 2012 (WWW 2012)*.
- [Kontokostas et al., 2012] Kontokostas, D., Bratsas, C., Auer, S., Hellmann, S., Antoniou, I., and Metakides, G. (2012). Internationalization of linked data: The case of the greek dbpedia edition. *Web Semantics: Science, Services and Agents on the World Wide Web*, 15(0):51 – 61.
- [Kreis, 2011] Kreis, P. (2011). Design of a quality assessment framework for the dbpedia knowledge base. Master’s thesis, Freie Universität Berlin.

- [Krejcie and Morgan, 1970] Krejcie and Morgan (1970). Determining sample size for research activities. *Educational and Psychological Measurement*, 30:607–610.
- [Lagoze et al., 2008] Lagoze, C., de Sompel, H. V., Nelson, M., and Warner, S. (2008). The open archives initiative protocol for metadata harvesting. <http://www.openarchives.org/OAI/openarchivesprotocol.html>.
- [Lehmann, 2007] Lehmann, J. (2007). Hybrid learning of ontology classes. In Perner, P., editor, *Machine Learning and Data Mining in Pattern Recognition, 5th International Conference, MLDM 2007, Leipzig, Germany, July 18-20, 2007, Proceedings*, volume 4571 of *Lecture Notes in Computer Science*, pages 883–898. Springer.
- [Lehmann, 2009] Lehmann, J. (2009). DL-Learner: learning concepts in description logics. *Journal of Machine Learning Research (JMLR)*, 10:2639–2642.
- [Lehmann, 2010] Lehmann, J. (2010). *Learning OWL Class Expressions*. PhD thesis, University of Leipzig. PhD in Computer Science.
- [Lehmann et al., 2011] Lehmann, J., Auer, S., Bühmann, L., and Tramp, S. (2011). Class expression learning for ontology engineering. *Journal of Web Semantics*, 9:71 – 81.
- [Lehmann et al., 2009] Lehmann, J., Bizer, C., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009). DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165.
- [Lehmann and Bühmann, 2010] Lehmann, J. and Bühmann, L. (2010). Ore - a tool for repairing and enriching knowledge bases. In *Proceedings of the 9th International Semantic Web Conference (ISWC2010)*, Lecture Notes in Computer Science, Berlin / Heidelberg. Springer.
- [Lehmann et al., 2012] Lehmann, J., Gerber, D., Morsey, M., and Ngonga Ngomo, A.-C. (2012). Defacto - deep fact validation. In *Proceedings of the 11th International Semantic Web Conference (ISWC2012)*.
- [Lehmann and Haase, 2009] Lehmann, J. and Haase, C. (2009). Ideal downward refinement in the EL description logic. In *Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium*.
- [Lehmann and Hitzler, 2007a] Lehmann, J. and Hitzler, P. (2007a). Foundations of refinement operators for description logics. In Blockeel, H., Ramon, J., Shavlik, J. W., and Tadepalli, P., editors, *Inductive Logic Programming, 17th International Conference, ILP 2007, Corvallis, OR, USA, June 19-21, 2007*, volume 4894 of *Lecture Notes in Computer Science*, pages 161–174. Springer. Best Student Paper Award.

- [Lehmann and Hitzler, 2007b] Lehmann, J. and Hitzler, P. (2007b). A refinement operator based learning algorithm for the  $\mathcal{ALC}$  description logic. In Blockeel, H., Ramon, J., Shavlik, J. W., and Tadepalli, P., editors, *Inductive Logic Programming, 17th International Conference, ILP 2007, Corvallis, OR, USA, June 19-21, 2007*, volume 4894 of *Lecture Notes in Computer Science*, pages 147–160. Springer. Best Student Paper Award.
- [Lehmann and Hitzler, 2010] Lehmann, J. and Hitzler, P. (2010). Concept learning in description logics using refinement operators. *Machine Learning journal*, 78(1-2):203–250.
- [Lehmann et al., 2013] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., Sahnwaldt, J. C., Stadler, C., van Kleef, P., Auer, S., Bizer, C., and Idehen, K. (2013). DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal*.
- [Lehmann et al., 2007] Lehmann, J., Schüppel, J., and Auer, S. (2007). Discovering unknown connections - the DBpedia relationship finder. In *Proceedings of 1st Conference on Social Semantic Web. Leipzig (CSSW'07), 24.-28. September*, volume P-113 of GI-Edition of *Lecture Notes in Informatics (LNI)*. Bonner Köllen Verlag.
- [Martin et al., 2013] Martin, M., Stadler, C., Frischmuth, P., and Lehmann, J. (2013). Increasing the financial transparency of european commission project funding. *Semantic Web Journal*, Special Call for Linked Dataset descriptions.
- [Meiser et al., 2011] Meiser, T., Dylla, M., and Theobald, M. (2011). Interactive reasoning in uncertain RDF knowledge bases. In Berendt, B., de Vries, A., Fan, W., and Macdonald, C., editors, *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2557–2560.
- [Mendes P.N., 2012] Mendes P.N., Mühleisen H., B. C. (2012). Sieve: Linked data quality assessment and fusion. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*.
- [Minack et al., 2009] Minack, E., Siberski, W., and Nejd, W. (2009). Benchmarking fulltext search performance of RDF stores. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications*, pages 81–95.
- [Morsey et al., 2011] Morsey, M., Lehmann, J., Auer, S., and Ngonga Ngomo, A.-C. (2011). DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data. In *Proceedings of the 10th international conference on The semantic web - Volume Part I*.

- [Morse et al., 2012a] Morse, M., Lehmann, J., Auer, S., and Ngonga Ngomo, A.-C. (2012a). Usage-Centric Benchmarking of RDF Triple Stores. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*.
- [Morse et al., 2012b] Morse, M., Lehmann, J., Auer, S., Stadler, C., and Hellmann, S. (2012b). DBpedia and the Live Extraction of Structured Data from Wikipedia. *Program: electronic library and information systems*, 46:27.
- [Nakamura et al., 2007] Nakamura, S., Konishi, S., Jatowt, A., Ohshima, H., Kondo, H., Tezuka, T., Oyama, S., and Tanaka, K. (2007). Trustworthiness analysis of web search results. In *Research and Advanced Technology for Digital Libraries, 11th European Conference*, volume 4675, pages 38–49.
- [Ngonga Ngomo and Auer, 2011] Ngonga Ngomo, A.-C. and Auer, S. (2011). Limes - a time-efficient approach for large-scale link discovery on the web of data. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Three*.
- [Ngonga Ngomo and Schumacher, 2009] Ngonga Ngomo, A.-C. and Schumacher, F. (2009). Border flow – a local graph clustering algorithm for natural language processing. In *Proceedings of the 10th International Conference on Intelligent Text Processing and Computational Linguistics (CICLING 2009)*, pages 547–558. Best Presentation Award.
- [Nguyen et al., 2007] Nguyen, D. P. T., Matsuo, Y., and Ishizuka, M. (2007). Relation extraction from wikipedia using subtree mining. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, pages 1414–1420.
- [Oren et al., 2008] Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., and Tummarello, G. (2008). Sindice.com: a document-oriented lookup index for open linked data. *Int. J. of Metadata and Semantics and Ontologies*, 3:37–52.
- [Owens et al., 2008a] Owens, A., Gibbins, N., and mc schraefel (2008a). Effective Benchmarking for RDF Stores Using Synthetic Data.
- [Owens et al., 2008b] Owens, A., Seaborne, A., Gibbins, N., and mc schraefel (2008b). Clustered TDB: A clustered triple store for jena. Technical report, Electronics and Computer Science, University of Southampton.
- [Pan et al., 2005] Pan, Z., Guo, Y., , and Heflin, J. (2005). LUBM: A benchmark for OWL knowledge base systems. In *Journal of Web Semantics*, volume 3, pages 158–182.
- [Pasternack and Roth, 2011a] Pasternack, J. and Roth, D. (2011a). Generalized fact-finding. In *Proceedings of the 20th international conference companion on World wide web, WWW '11*, pages 99–100, New York, NY, USA. ACM.

- [Pasternack and Roth, 2011b] Pasternack, J. and Roth, D. (2011b). Making better informed trust decisions with generalized fact-finding. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, pages 2324–2329. AAAI Press.
- [Prud'hommeaux and Seaborne, 2008] Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL query language for RDF. W3C recommendation, W3C.
- [Sarasua et al., 2012] Sarasua, C., Simperl, E., and Noy, N. (2012). Crowdmap: Crowdsourcing ontology alignment with microtasks. In *Proceedings of the 11th International Semantic Web Conference (ISWC2012)*, Lecture Notes in Computer Science, pages 525–541. Springer Berlin Heidelberg.
- [Schmidt et al., 2009] Schmidt, M., Hornung, T., Lausen, G., and Pinkel, C. (2009). SP2Bench: A SPARQL Performance Benchmark. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009*, pages 222–233. IEEE.
- [Stadler et al., 2012] Stadler, C., Lehmann, J., Höffner, K., and Auer, S. (2012). Linkedgeodata: A core for a web of spatial open data. *Semantic Web Journal*, 3(4):333–354.
- [Stadler et al., 2010] Stadler, C., Martin, M., Lehmann, J., and Hellmann, S. (2010). Update Strategies for DBpedia Live. In *6th Workshop on Scripting and Development for the Semantic Web Colocated with ESWC 2010 30th or 31st May, 2010 Crete, Greece*.
- [Stickler, 2005] Stickler, P. (2005). CBD - concise bounded description. Retrieved February 15, 2011, from <http://www.w3.org/Submission/CBD/>.
- [Suchanek et al., 2008] Suchanek, F. M., Kasneci, G., and Weikum, G. (2008). Yago: A large ontology from wikipedia and wordnet. *Journal of Web Semantics*, 6(3):203–217.
- [Tacchini et al., 2009] Tacchini, E., Schultz, A., and Bizer, C. (2009). Experiments with wikipedia cross-language data fusion. In *Proceedings of the 5th Workshop on Scripting and Development for the Semantic Web, ESWC*. Citeseer.
- [Theoharis et al., 2011] Theoharis, Y., Fundulaki, I., Karvounarakis, G., and Christophides, V. (2011). On provenance of queries on semantic web data. *IEEE Internet Computing*, 15:31–39.
- [TPC, 2012] TPC (2012). Transaction processing performance council website (TPC). <http://www.tpc.org>.
- [W3C, 2004] W3C (2004). Resource description framework (rdf). <http://www.w3.org/RDF/>.

- [W3C, 2009] W3C (2009). W3C semantic web activity. Última visita 8/6/2010.
- [Wang et al., 2012] Wang, J., Kraska, T., Franklin, M. J., and Feng, J. (2012). Crowder: crowdsourcing entity resolution. *Proc. VLDB Endow.*, 5:1483–1494.
- [Wikipedia, 2012] Wikipedia (2012). Benchmark — Wikipedia, the free encyclopedia. [http://en.wikipedia.org/wiki/Benchmark\\_\(computing\)](http://en.wikipedia.org/wiki/Benchmark_(computing)). [Online; accessed 12-September-2012].
- [Wikipedia, 2013] Wikipedia (2013). SPARQL — Wikipedia, The Free Encyclopedia. [Online; accessed 31-March-2013].
- [Wu and Weld, 2007] Wu, F. and Weld, D. S. (2007). Autonomously semantifying wikipedia. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 41–50, New York, NY, USA. ACM.
- [Yan et al., 2009] Yan, Y., Okazaki, N., Matsuo, Y., Yang, Z., and Ishizuka, M. (2009). Unsupervised relation extraction by mining wikipedia texts using information from the web. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 1021–1029.
- [Yin et al., 2007] Yin, X., Han, J., and Yu, P. S. (2007). Truth discovery with multiple conflicting information providers on the web. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1048–1052.
- [Yu, 2007] Yu, L. (2007). *Introduction to Semantic Web and Semantic Web services*. Chapman & Hall/CRC, Boca Raton, FL.
- [Zaveri et al., 2013a] Zaveri, A., Kontokostas, D., Sherif, M. A., Bühmann, L., Morsey, M., Auer, S., and Lehmann, J. (2013a). User-driven quality evaluation of dbpedia. In *To appear in Proceedings of 9th International Conference on Semantic Systems, I-SEMANTICS '13, Graz, Austria, September 4-6, 2013*. ACM.
- [Zaveri et al., 2013b] Zaveri, A., Rula, A., Maurino, A., Petrobon, R., Lehmann, J., and Auer, S. (2013b). Quality assessment methodologies for linked open data. *Semantic Web journal*.
- [Zaveri et al., 2013c] Zaveri, A., Rula, A., Maurino, A., Petrobon, R., Lehmann, J., and Auer, S. (2013c). Quality assessment methodologies for linked open data. Under review, available at <http://www.semantic-web-journal.net/content/quality-assessment-methodologies-linked-open-data>.

# Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

Leipzig, den 13.4.2014

Mohamed Mabrouk Mawed Morsey