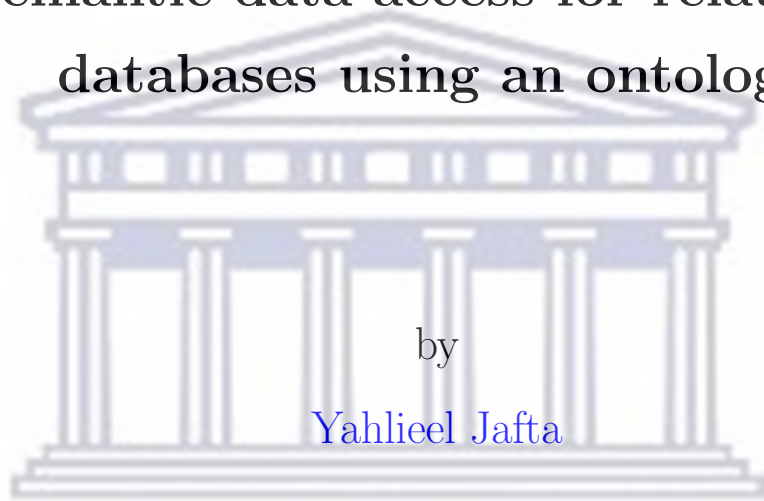


UNIVERSITY OF THE WESTERN CAPE

Semantic data access for relational databases using an ontology



by

Yahlieel Jafta

UNIVERSITY *of the*
WESTERN CAPE

A thesis submitted in fulfilment for the
degree of Master of Science

in the

Faculty of Natural Sciences

Department of Computer Science

Supervisor: Prof Louise Leenen

Co-supervisor: Prof Thomas Meyer


March 2024

Declaration of Authorship



UNIVERSITY *of the*
WESTERN CAPE

I, YAHLIEEL JAFTA, declare that this thesis “*Semantic data access for relational databases using an ontology*” is my own work, that it has not been submitted before for any degree or assessment at any other university, and that all the sources I have used or quoted have been indicated and acknowledged by means of complete references.

Signed:  _____

Date: 27 March 2024 _____

Abstract

Data analysis-based decision-making is performed daily by domain experts. As data grows in size and heterogeneity, accessing relevant data becomes challenging. In an Ontology-based data access (OBDA) approach, ontologies are advocated as a suitable formal tool to address complex data access. This technique falls within the Semantic Web domain, combining a domain ontology with a data source by using a declarative mapping specification to enable data access using a domain vocabulary. In this research, we investigate this approach by: a) studying the theoretical background that enables this technique; b) conducting a literature review on the existing open source tools that implement OBDA; c) implementing OBDA on a “real-world” relational dataset using an OBDA tool; and d) providing results and analysis of query answering. We selected Ontop (<https://ontop-vkg.org>) among various OBDA tools to illustrate how this technique enhances the data usage of the GitHub community. Ontop is an open-source tool applying OBDA in the domain of relational databases. We used the GHTorrent dataset, a relational database, in combination with the SemanGit ontology for our implementation. We perform a set of queries to highlight a subset of the features of this data access approach. The results from the queries look positive and can assist various use cases related to GitHub data with a semantic approach and integrate data from platforms integrating with GitHub directly or indirectly. The feature of querying in domain vocabulary without the need to understand the underlying data and schema stands out and provides benefits in practice. However, we observe the practical impediments in the manual development of a domain ontology and creating a mapping specification, the most complicated OBDA design-time task. Finally, we discuss the selected queries and conclude with future research.

Acknowledgements

I want to extend my sincere gratitude to the following individuals and organizations for their support and contributions to my MSc thesis in Computer Science:

I sincerely appreciate the guidance, expertise, and mentorship provided by my supervisor, Louise Leenen, and co-supervisor, Thomas Meyer. Their insights and encouragement were invaluable throughout this research.

I owe my family and friends a debt of gratitude for their support, patience, and understanding during the challenging journey of my MSc studies.

I am thankful for the resources the University of the Western Cape and the Computer Science Department provided, which greatly facilitated my research.

This thesis would not have been possible without the collective efforts and support of the individuals and organizations mentioned above. Thank you all for being an essential part of my academic journey.



Publication

- **Title:** Investigating Ontology-based data access with GitHub

Authors: Yahlieel Jafta, Louise Leenen, Thomas Meyer

Jafta, Y., Leenen, L., Meyer, T. (2023). Investigating Ontology-Based Data Access with GitHub. In: Pesquita, C., et al. The Semantic Web. ESWC 2023. Lecture Notes in Computer Science, vol 13870. Springer, Cham. https://doi.org/10.1007/978-3-031-33455-9_38, pp. 644–660 [52]



UNIVERSITY *of the*
WESTERN CAPE

Contents

Declaration of Authorship	i
Abstract	ii
Acknowledgements	iii
Publication	iv
List of Figures	viii
Abbreviations	x
1 Introduction	1
1.1 Problem Statement	1
1.2 Research Question	5
1.3 Research Objectives	6
1.4 Methodology	6
1.4.1 Problem identification and motivation	6
1.4.2 Objectives for a solution	7
1.4.3 Design and development	7
1.4.4 Evaluation	7
1.4.5 Communication	7
1.5 Thesis Structure and Outline	8
2 Background	9
2.1 Semantic Web	9
2.2 Conceptualization	10
2.3 Ontology	12
2.4 Description Logic	14
2.5 OWL	15
2.6 Connecting ontologies and databases	20
2.6.1 SPARQL	20
2.6.2 OBDA Framework	23
2.6.3 Query answering	24
2.6.4 Mapping	25

2.7	Conclusion	26
3	Literature Review	27
3.1	Introduction	27
3.2	Literature Review Methodology	28
3.3	Use cases	30
3.3.1	Manufacturing/Machine Diagnoses	30
3.3.2	Oil and Gas	31
3.3.3	Biomedical	31
3.3.4	Biology	31
3.3.5	Healthcare	31
3.3.6	Services	32
3.3.7	Maritime	32
3.3.8	Big Data	32
3.4	Data sources, Ontologies and Mappings	33
3.4.1	Data sources	33
3.4.2	Ontologies	33
3.4.3	Mappings	33
3.5	Optimization	34
3.6	Evaluation and Results	35
3.7	Discussion	36
3.8	Conclusion	36
4	Ontology-Based Data Access Tool, Dataset and Ontology	37
4.1	Introduction	37
4.2	OBDA tool	38
4.2.1	Ontop system	39
4.2.2	Query representation	40
4.2.3	SPARQL to SQL translation	40
4.2.3.1	SPARQL to IQ	43
4.2.3.2	IQ to SQL	45
4.3	The GHTorrent Dataset	45
4.3.1	GHTorrent Data Collection	47
4.3.2	GHTorrent Limitations	50
4.4	SemanGit Ontology	51
4.4.1	SemanGit limitations	53
4.4.2	SemanGit extensions	53
4.4.2.1	Class definitions	54
4.4.2.2	Property definitions	55
4.5	Conclusion	56
5	Implementation	57
5.1	Introduction	57
5.2	Preliminaries	57
5.3	Database setup	58
5.4	Mapping GHTorrent to the SemanGit Ontology	58
5.4.1	Mapping assertions	58

Figure 5.1 mapping assertions	59
Figure 5.2 mapping assertions	60
Figure 5.3 mapping assertions	61
Figure 5.4 mapping assertions	62
Figure 5.5 mapping assertions	63
Figure 5.6 mapping assertions	64
Figure 5.7 mapping assertions	64
5.5 Querying GHTorrent with SPARQL	66
5.6 Discussion	76
5.7 Conclusion	77
6 Conclusion	78
6.0.1 Research sub-question 1	79
6.0.2 Research sub-question 2	79
6.0.3 Research sub-question 3	80
6.0.4 Research sub-question 4	80
6.1 Future work	81
6.2 Concluding Comments	82
A Literature review summary	83
B Mapping specifications	106
C Ontology	113
Bibliography	117




List of Figures

1.1	OBDA Architecture [117]	4
2.1	A sample conceptualization of the GitHub domain with a user, commit, and repository [117].	11
2.2	RDF graph model based on table 2.1	22
3.1	Percentage of reviewed papers by year	28
3.2	OBDA/OBDI generic structure	29
3.3	Number of papers per domain	30
4.1	GHTorrent MySQL Database Schema	48
4.2	SemanGit Ontology summary (classes and properties)	52
5.1	User entity mapping	59
5.2	Repository (Project) entity mapping	61
5.3	Commit entity mapping	61
5.4	Project commits mapping	62
5.5	Pull Request entity mapping	63
5.6	Pull Request commits mapping	64
5.7	Pull Request event history mapping	64
5.8	Merged Pull Requests mapping	65
5.9	SPARQL query retrieving commits of authors from the “rails” GitHub repository.	66
5.10	Angular and React repo commits by year	71
5.11	Angular and React top 10 contributors by commits	73
5.12	Angular commit author commits vs. pull requests	74
B.1	Commit map	106
B.2	Commit Comment map	106
B.3	Follow map	107
B.4	Issue map	107
B.5	Issue Label map	107
B.6	Organization member map	107
B.7	Programming language map	108
B.8	Project map	108
B.9	Project commit map	108
B.10	Project label map	108
B.11	Project programming language map	109
B.12	Pull Request map	109

B.13 Pull Request comment map	109
B.14 Pull Request commit map	109
B.15 Pull Request history map	110
B.16 Pull Request merge map	110
B.17 Pull Request user map	110
B.18 Pull Request user map	110
B.19 User map	111
B.20 User commit map	111
B.21 User programming languages map	111
B.22 Repository milestone map	111
B.23 Watcher map	112
C.1 SemanGit Ontology summary (classes and properties)	114
C.2 SemanGit Ontology metrics	115
C.3 Ontology visualisation using WebVOWL [74].	116

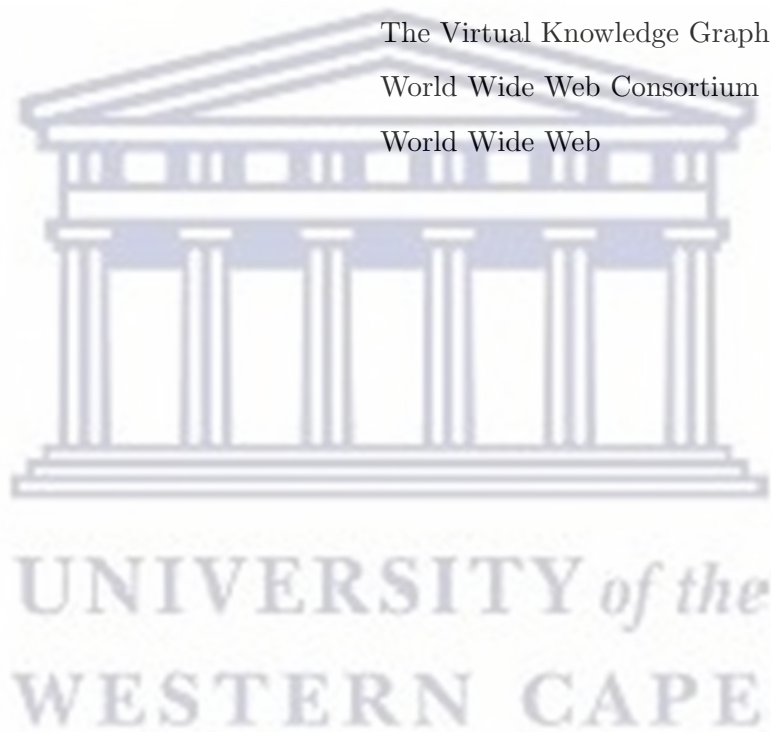


Abbreviations



ADNI	Alzheimer Disease Neuroimaging Initiative
BSBM	Berlin SPARQL Benchmark
CQs	Conjunctive Queries
DBMS	Database Management System
DL	Description Logic
dbDist	DISTINCT by the database engine
obdaDist	DISTINCT by the OBDA engine
ETL	Extract, Transform, and Load
F-Logic	Feature-based Description Logic
HDF5	Hierarchical Data Format version 5
HDFS	Hadoop Distributed File System
IQ	Intermediate Query
IRIs	Internationalized Resource Identifiers
KG	Knowledge Graph
LLMs	Large Language Models
MWS	Materialised Window Signatures
OBDA	Ontology-based Data Access
OBDI	Ontology-based Data Integration
OWL	Web Ontology Language
PR	Pull Request
REST	Representational state transfer
RDF	Resource Description Framework
RDFS	RDF Schema
RDMS	Relational Database Management Systems
RDB2RDF	Relational Databases to RDF
RML	RDF Mapping Language

R2RML	RDB to RDF Mapping Language
Scalable Semantic Analytics Stack	SANSA
SemanGit	Semantic Git
SPARQL	SPARQL Protocol and RDF Query Language
SQO	Semantic Query Optimization
SUS	System Usability Scale
TDB	Triple Database
Turtle	Terse RDF Triple Language
VCS	Version Control System
VKG	The Virtual Knowledge Graph
W3C	World Wide Web Consortium
WWW	World Wide Web



Chapter 1

Introduction

1.1 Problem Statement

Information retrieval is a critical process in organizations for extracting insights to achieve strategic organizational objectives. In various domains, clients require access to domain-specific services exported by systems [14]. Information has become the focal point of society and a driver of innovation in analytics with the intent to obtain valuable insights to drive decision-making. Large enterprises today use several information systems, each with its database to store input and functional data [42]. Domain experts use data analysis to inform decision-making. However, gaining access to the required data is becoming a challenge due to data access generally being performed by technical experts who translate the requirements of domain experts into the necessary analytical output, creating a bottleneck at scale [61]. While big data analytics plays a pivotal role in the modern-day enterprise, data scientists required to produce the analytical output spend significant time on the data preparation and integration phase [31].

Let us take the importance of data management in business into consideration. We can deduce that the challenge of data collection and providing adequate access at a reasonable cost is of utmost importance for businesses today [31]. Even though data integration is a known problem in data management, the need to address the problem of data access and integration in the climate of big data is increasing and poses a significant challenge. Data access and integration pose some challenges. One of these challenges is the complexity related to modeling data integration applications. Another challenge is the need for more data integrity across various data sources, which can result in inconsistent data or data redundancy. Processing and reasoning on queries become more complex as a result of this. According to Gusenkov et al. [42], the problem of data integration is closely related to the problem of developing intelligent search systems

(intelligent data access), given that both these problems address the underlying factor of providing users (i.e., data scientists, domain experts, application end-users or machines) access to heterogeneous data.

The two main ways to handle access to heterogeneous data are procedural and declarative [68].

- The procedural methodology takes a bottom-up approach by addressing the problem at the data source level. Generally, this includes specific software service solutions. A few approaches apply the procedural methodology, such as federated databases [102] and mediators [21]. Federated databases involve defining relationships between various databases where mediators consist of modules at the client level that convert the data from various sources into a suitable form. Both these approaches are expensive to maintain. Whenever there is a change in the schema or structure of the underlying data sources, the software that facilitates data access must be updated.
- The declarative approach, also called a top-down approach, defines a global representation or shared conceptualization valid for the domain of interest underlying the data sources. This conceptualization links intentional domain terms to actual data. These terms are then specified to access information [14].

The central theme of this thesis revolves around the declarative approach. To realize such a solution, an approach known as Ontology-based data access (OBDA) is advocated for; a technique that utilizes ontologies as a suitable formal tool for data access [92]. An ontology is a formal specification of the concepts within a given domain and the relationships between these concepts [39]. It is formalizing domain knowledge and expressing the concepts in a shared vocabulary. This conceptualization can be used to reason about domain concepts and perform inference, thus giving rise to interesting applications and solutions such as data access [14] and data integration [42]. Given the heterogeneity of data in “real-world” information systems, the formal specification of domain knowledge plays a significant role in providing a unified representation of heterogeneous data. In the literature, this formalism of domain knowledge is advocated for and applied in the space of problems around data integration and developing intelligent search systems [42]. This thesis focuses on the latter; however, the literature considers this a unified problem [42].

Given Codd’s relational algebra and the normalization theory [22], today, we see various relational database management systems (RDMS) being used within the industry. In the last decade, the database research community created the foundation for using columnar

storage [1], allowing for efficient storage and processing of large data sets. As a result, RDMS has seen considerable growth, especially the wide adoption of database systems offered as cloud services [1]. In OBDA, the ontology expresses a shared conceptualization of the domain of interest at a high level of abstraction independent from the data sources. While an ontology is a good candidate for realizing this conceptualization, RDMS are natural candidates for the management of the data layer given the maturity of RDMS.

Roughly every five years, a group of database researchers meet to do a self-assessment of the database community and produce a report. In the most recent assessment, the 2019 Seattle Report on Database Research [1], they produced a set of challenges facing the community and data scientists. We list a few relevant to the interest of this thesis.

- The researchers emphasize a clear trend toward heterogeneous computation, which will have an impact on traditional data warehouse storage. The database community is in transition to a data lake-oriented architecture for analytics. A data lake is a central repository intended to store, process, and secure many semistructured and unstructured data. Identifying joinable data with other relevant data sets within a data lake remains challenging.
- Data integration and data wrangling is 80-90% of the challenge data scientists face and remains an ongoing problem for decades. To address this problem, the researchers recommend that the community focus on the end-to-end data-to-insights pipeline [1], such as visualizing the answer to a user query.
- Understanding the context of data and the processes working on it, including provenance. Data provenance refers to the traceability of data such that the source can be determined, including all transformations, to enable evaluation of its quality and authenticity [32]. For further details on data provenance, we refer the reader to [10, 32]. A recent study [113] made some advancements in this area in the context of ontologies.

The virtual knowledge graph (VKG) approach, referred to in the literature as OBDA, has become a well-known view for accessing and integrating data sources [119]. In this approach, the data sources are virtualized through mappings and an ontology, which is presented as a unified knowledge graph (KG) that end-users can query using domain vocabulary [119]. A knowledge graph is a knowledge base that represents a network of real-world entities in a graph structure¹. Given the rise of knowledge discovery applications, users are increasingly required to write complex database search requests to retrieve information. OBDA provides end-users access to data in a form that does not

¹<https://www.ibm.com/cloud/learn/knowledge-graph>

require deep-level database-related technical skills and schema knowledge [42]. When the data gets queried, the user query is translated over the ontology into SQL queries over the database. This is depicted in figure 1.1. Domain experts can express information needs in domain terms they are familiar with, without any background knowledge on the way the data is structured at the source, and to receive answers that are understood [59].

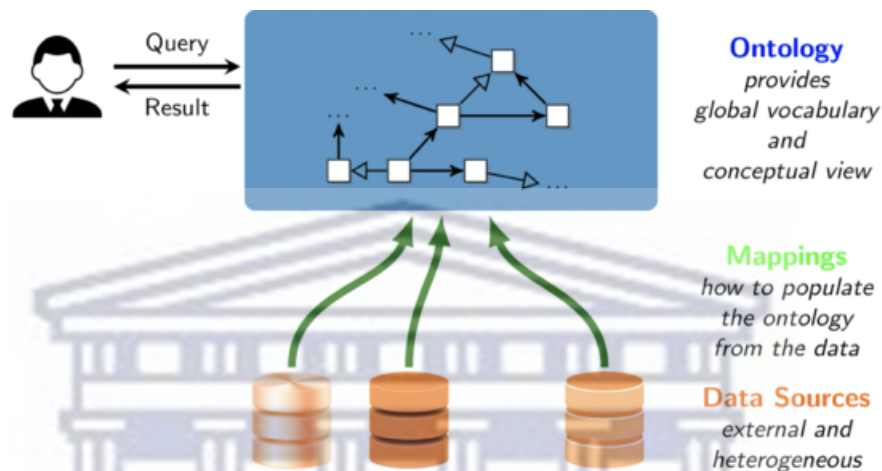


FIGURE 1.1: OBDA Architecture [117]

The mapping specification layer is responsible for binding the ontology and the data sources. This is achieved by linking the classes and properties in the ontology to SQL views over the data in the database. The ontology, combined with the mappings, exposes a VKG, which can be queried using SPARQL, the standard query language in the Semantic Web [12]. To distinguish KG's from ontologies, we note the following. An ontology describes the entity types and their relationships in the absence of data, whereas a KG is instantiated once the ontology is populated with instance data. The ontology provides the schema of the entities and relationships, and the KG contains the entity data enriched by the ontology that provides the context.

This research investigates the approach of OBDA and applies an implementation to a relational database populated with real-world data using a well-known OBDA tool. The tool of choice is Ontop, a state-of-the-art open-source OBDA system released under the Apache license that links relational databases with their applicable domain ontology. The terms in an ontology are linked to the underlying data sources using a formal mapping specification, which exposes the relational database as a virtual Resource Description Framework (RDF) graph [12]. Such a virtual graph can then be queried via SPARQL (SPARQL Protocol and RDF Query Language) by translating the SPARQL queries into SQL queries over the relational database. In order to realize such an implementation, a data source and related domain ontology are required.

GitHub is a popular software project hosting platform for version control and collaboration using the *Git* [73] protocol. Git is an open-source tool developers use to manage their source code in their local environment, while GitHub is an online service to which developers who use Git can connect to synchronize local source code. GitHub provides an extensive public REST API that enables researchers to retrieve both the commits to the projects' repositories and events generated through user actions on projects. Analysts are mining the data stored in GitHub's event logs, attempting to understand how the users interact with the application and collaborate on software projects.

To address the need for empirical software engineering studies in the scope of Github, the GHTorrent [35] project was established. GHTorrent aims to create a scalable offline mirror of GitHub's event streams and persistent data and offer it to the research community as a service. GHTorrent has used the GitHub REST API to gather data from GitHub over several years, resulting in a better source for GitHub data due to the request rate limitation on the REST API. The raw JSON responses returned from the REST API are stored in MongoDB and MySQL databases. The MySQL database is used for this research since it is a relational model. Furthermore, there exists an RDF-linked dataset for GHTorrent called the SemanGit [66]. The SemanGit dataset was systematically built by transforming the GHTorrent dataset into linked data, and the SemanGit ontology was subsequently developed.

This research closely investigates the OBDA technique in the context of intellectual search systems. This is executed with the following set of resources:

- GHTorrent (GitHub) dataset, MySQL database instance
- SemanGit ontology
- Using Ontop, a state-of-the-art VKG system to configure and link the ontology and database.

1.2 Research Question

The main research question posed in this research is phrased as follows: "How effective is OBDA with real-world data?"

This research question can be broken down into the following sub-questions:

1. How does OBDA scale in terms of performance and implementation on real-world datasets?

2. How do OBDA implementations compare in terms of successful results?
3. What are the current limitations of OBDA?
4. What improvements can be made?

1.3 Research Objectives

The following research objectives will be used to obtain answers to the research sub-questions mentioned in the previous section, the answers to which will culminate in an answer to the main research question posed in the previous section:

1. Investigate the theoretical background of the OBDA technique and document design/implementation decisions.
2. Conduct a literature review to investigate OBDA scalability.
3. Investigate tools implementing OBDA.
4. Implement OBDA on a real-world dataset using an OBDA tool and relational database.
5. Provide results and analysis of query answering.

1.4 Methodology

This study applies the Design Science Research (DSR) methodology to evaluate OBDA over real-world data, specifically relational databases. Research applying DSR has been based on several process models [114]. The model we follow is based on the work by Peffers et al. [90], a widely used model with over 10,000 citations. It consists of the following components; “problem identification and motivation, objectives for a solution, design and development, evaluation, and communication” [90].

The thesis is organized around these components as follows:

1.4.1 Problem identification and motivation

Given the importance of data management in industry and the continued rise of Big Data, the need to address the problem of data access and integration is increasing and poses a significant challenge. The primary focus of this thesis is to investigate the

limitations and prospects of OBDA as a way of querying relational databases using ontologies.

1.4.2 Objectives for a solution

The study begins with an in-depth exploration of the theoretical foundations of OBDA. This involves understanding the underlying principles, techniques, and methodologies related to mapping relational data to ontologies and developing query-answering systems based on these mappings. A comprehensive literature review is conducted to analyze existing real-world implementations of OBDA. This review aims to identify the strengths, weaknesses, and practical implications of various approaches taken in industry and academia.

1.4.3 Design and development

Building upon the theoretical background and insights from the literature review, we develop an implementation of OBDA on real-world data. In this step, we employ the Ontop tool, a state-of-the-art technology in the OBDA domain, to establish a mapping specification between the SemanGit ontology and the MySQL relational database instance. This serves as the basis for the execution of experiments to perform OBDA evaluation.

1.4.4 Evaluation

We conducted a series of experiments to evaluate the effectiveness and efficiency of the developed artifact. Real-world data is used to simulate practical scenarios, and the experimental results are analyzed. This evaluation identifies the strengths and limitations of the developed OBDA implementation. The reflection process allows for insights into potential improvements and future research directions.

1.4.5 Communication

The knowledge gained from this study is shared with the academic community to contribute to the progression of the OBDA field. This is achieved through the development of an artifact and an associated publication [52]. The artifact, representing the culmination of theoretical, practical, and experiments, serves as a tangible contribution to the

OBDA domain. The publication provides an account of the research, including problem identification, conceptual foundations, development, and empirical findings.

1.5 Thesis Structure and Outline

The structure of the thesis is presented in the form of providing the theoretical foundations of OBDA, followed by the investigative procedure and practical implementation. Chapters 2–3 of the thesis provide the theoretical background on OBDA, semantic technologies, and the related work in the literature. These two chapters set the foundation for understanding the theoretical landscape of OBDA and are used as the basis for the implementation. This is continued by discussing the Ontop system, the selected GHTorrent dataset, and the SemanGit ontology in Chapter 4. In Chapter 5, we document the implementation, including the mapping specification configuration, query answering, and result analysis. Each chapter builds on the previous chapter, which outlines the literature required to understand the OBDA paradigm and a real-world example in the form of an implementation using real-world data.

Chapter 6 concludes the thesis by providing the answers to the sub-research questions, thereby providing an answer to the main research question posed in the thesis. The chapter closes with several directions for future work.



UNIVERSITY *of the*
WESTERN CAPE

Chapter 2

Background

This chapter discusses the background material on Ontology-Based Data Access (OBDA) and what differentiates this approach from a theoretical perspective and in practice. The chapter is structured as follows. We discuss the Semantic Web, the broader research field that OBDA forms a part of, in section 2.1. We follow up by describing the theoretical foundations for ontological conceptualizations, the core of this research, in sections 2.2 and 2.3. We discuss the languages employed to define ontologies in sections 2.4 and 2.5. Finally, we conclude by discussing the steps taken to apply OBDA in practice.

2.1 Semantic Web

The Semantic Web is a diverse field of research introduced in 2001 by Tim Berners-Lee as an extension to the current World Wide Web (WWW). As a component of Web 3.0¹, it has been standardized by the World Wide Web Consortium (W3C). Web 3.0 is a concept in the domain of the evolution of the WWW [96]. The evolution of the WWW has gone through two iterations, namely Web 1.0 and Web 2.0.

Web 1.0 was a platform for publishing static, well-designed information in text and images without interaction between the information and the user. Web 2.0, the current WWW, was an extension of Web 1.0, which increased cooperation between organizations, users, programmers, and service providers, allowing them to reuse and contribute information. In the current climate of the WWW, which is overflowing with exabytes of data, machines are still unable to automate harvesting all this information or carrying out complex tasks with it [96].

¹<https://www.w3.org/standards/semanticweb/>

The vision of the Semantic Web is to bring structure and meaning to the content on web pages, to enable an environment for automated processes, also referred to as “intelligent software agents”, to roam the pages on the internet and execute highly sophisticated tasks. Let us demonstrate this vision with a concrete example in the context of GitHub.

Example 2.1. *Bob is a new intern starting at a software development company. Working on his first project, he starts pushing software changes for his first task to the organization’s repository associated with a GitHub milestone. GitHub milestones are used to track the progress of commits on a GitHub project.*

In this context, an *intelligent agent* “visiting” the web page of the organization’s repository will know the metadata on the web page, encoded as keywords (project name, programming language) in the HTML markup as it is done today. Additionally, the “agent” will also know that Bob is a new user on the team and can associate the task with a project milestone whose deadline is on Friday.

An argument is made for *semantics* being the most crucial factor for advancing the Web to its next phase [103]. The Semantic Web is an expansion of the present WWW, wherein data is given unambiguous meaning [103], which is an enabler of task coordination between people and machines [7]. Data integration is the foundation of the Semantic Web [87]. Data that is “data about data” is transformed through the use of metadata into meaningful information that can be located, evaluated, and delivered by software agents [87]. The metadata are embedded into web pages, which enables software agents to decipher the meaning of content on the web. Ontologies, a semantic technology, are a fundamental building block of the Semantic Web [56] and are expected to be significant in assisting automated processes in accessing information [48]. We discuss ontologies in section 2.3.

2.2 Conceptualization

Knowledge representation in a formal declarative structure originates from conceptualizing the domain of interest. Based on Genesereth and Nilsson [30], this includes the concepts over a domain of interest and the relations that link them. The universe of discourse refers to the entities being discussed.

Based on example 2.1, in figure 2.1, we depict a sample conceptualization for the GitHub domain.

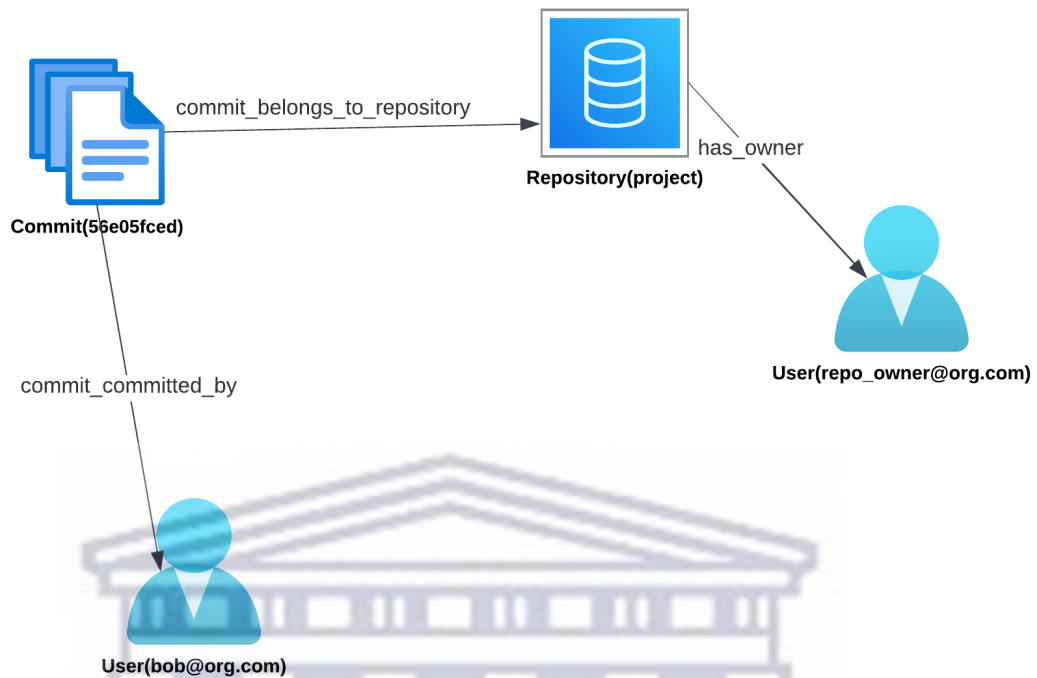


FIGURE 2.1: A sample conceptualization of the GitHub domain with a user, commit, and repository [117].

By observing these relations defined here, we can see that they represent a specific snapshot in time or a specific state of the “world” over the universe of discourse. However, a conceptualization deals with the underlying meaning of the concepts and not a specific “world” state within the universe of discourse. For example, the meaning of *commit_committed_by* in figure 2.1 is defined in the way a **user** interacts with a **commit**. Thus, as proposed by Guarino [40], a conceptualization deals with the intensional relations of the concepts and not the extensional relations between the objects that reflect a specific world of the universe. We now formalize what we refer to when we say “world”, based on [41].

Definition 2.1. (World) In respect of a specific system S , we want to model a specific world state, where S consists of the instances for the concepts that comprise the system. A world consists of a set of ordered states corresponding to the progression of S over time. Decoupling the time aspect from this, a world state conforms to a world [41].

Definition 2.2. A conceptualization is a triple $\mathbf{C} = (D, W, \mathbf{R})$ where,

- D is a set called the universe of discourse
- W is a set of possible worlds
- \mathbf{R} is a set of conceptual relations in the domain $\langle D, W \rangle$

Example 2.2. We now define our example in 2.1 in this context.

- $D = \{56e5fced, bob@org.com, repo_owner@org.com, project\}$
- $W = \{w1, w2, w3, \dots\}$
- $R = \{Commit^1, User^1, Repository^1, User^1, commit_committed_by^2, commit_belongs_to_repository^2, has_owner^2\}$

To keep the example brief, we assume that the unary conceptual relations $Commit^1$, $User^1$, $Repository^1$ and $User^1$ map to the same extensions in every possible world state. We do not apply this assumption to the binary relations $commit_committed_by^2$, $commit_belongs_to_repository^2$ and has_owner^2

- for all worlds w in W : $Commit^1(w) = \{\dots, 56e5fced, \dots\}$
- for all worlds w in W : $User^1(w) = \{\dots, bob@org.com, \dots, repo_owner@org.com, \dots\}$
- for all worlds w in W : $Repository^1(w) = \{\dots, project, \dots\}$
- $commit_committed_by^2(w_1) = \{\dots, (56e5fced, bob@org.com), \dots\}$
- $commit_committed_by^2(w_2) = \{\dots, (56e5fced, repo_owner@org.com), \dots\}$
- $commit_committed_by^2(w_3) = \dots$
- $commit_belongs_to_repository^2(w_1) = \{\dots, (56e5fced, project), \dots\}$
- $commit_belongs_to_repository^2(w_2) = \dots$

2.3 Ontology

An ontology is an explicit specification of a domain conceptualization [39]. It is a formal representation of knowledge by a set of concepts or terms within a domain and the relations between them. It is the process of formalizing knowledge and expressing the concepts and their relations in a given domain of interest, which, as a result, defines a shared domain vocabulary that is interpretable by both people and machines [39]. We now discuss this explicit specification in line with the running example in 2.1 by highlighting the theoretical foundations.

According to Guarino [41], ontologies are required to facilitate communication between humans and machines and inter-machine and inter-human communication. Consequently, ontologies present a sharable and reusable knowledge base, empowering the

expansion of domain knowledge. This expansion of a knowledge base is achieved by inferences made on the existing knowledge base. Inference is a tool to improve the quality of data by discovering new relationships and performing automated analyses on the content of the data to extend the existing knowledge or identify any data inconsistencies. Automated reasoning algorithms achieve this, a vital enabler of utilizing ontologies in practice [45].

Motivations for using ontologies include:

- Knowledge analysis at the domain level.
- Establishing a shared vocabulary for a domain allows information to be shared between people and machines.
- Integration of knowledge from various domains.
- Defining domain assumptions in an explicit manner, which simplifies maintenance when assumptions change.
- Distinguishing domain knowledge from operational knowledge.
- Extending the knowledge base by revealing implicit domain knowledge via automated reasoning.

In human correspondence, a *language* describes elements of a particular conceptualization. To express a GitHub *commit*(56e5fcd) that was committed by *user* (bob@org.com), we have to use a specific formal symbol (*commit_committed_by*) that represents a particular conceptual relation. In this context, Guarino [41] classifies this as the agent *committing* to a conceptualization utilizing a specific language. To realize this as an explicit specification, how the conceptual relations are *interpreted* according to the conceptualization becomes the focal point. For example, how do we formally ensure that the interpretation of the conceptual relation *commit_committed_by* is interpreted according to the committed conceptualization? Once a specific conceptualization has been committed, we must ensure that each possible world state conforms to the intended conceptualization. This is where ontologies emerge as an explicit specification of a domain conceptualization. The conceptualization is explicitly defined in a constrained language, intensionally utilizing suitable axioms [41]. These axioms are specified using a formal language, enabling machine-readable conceptualization expressions.

2.4 Description Logic

Researchers have proposed various ontology languages in the literature based on various formalisms. One of the earlier formalisms is Frame-based languages such as Frame Logic (F-Logic) and logic-based languages in view of First Order Logic (FO). F-Logic combines the declarative style, concise syntax, and clearly defined semantics of logic-based languages with the advantages of conceptual modeling that come from object-oriented frame-based languages [3]. F-Logic relies on representing knowledge as frames and their semantics being defined operationally, compared to logic-based languages, FO and Description Logics (DLs), that apply the formal semantics of their underlying logic [99]. Given the high expressive power of FO, it provides limited automated reasoning capabilities in practice due to high computational cost. DLs, a fragment of FO, generally do not suffer from this limitation and are mostly decidable [99].

DLs are a class of knowledge representation languages that can represent knowledge of an application domain in a formal structure [5], such as an ontology. DLs make this possible by providing the mechanics to represent the relationships between entities in a given domain of interest. These entities are concepts, roles, and individuals. Concepts or domain terms represent sets of individuals, roles represent binary relations between the concepts, and individual names represent single individuals in the domain [65]. The representation of the domain is expressed as a formal definition of the concept expressions built from atomic concepts, for domain terms and the relationships between them as atomic roles. In DLs, the atomic concepts are known as unary predicates and atomic roles as binary predicates [5].

DLs generally distinguish domain knowledge into two parts, a terminological part and an assertional part called the TBox and the ABox, respectively. The combination of these parts is called a knowledge base (KB). The TBox represents knowledge about the structure of the domain, such as an ontology or database schema, capturing a set of universally quantified assertions expressing generic properties of domain concepts and roles [26]. The ABox represents knowledge about concrete instances or individual objects, such as a database instance.

Consider an example where an ontology for the GitHub domain application is developed from a vocabulary of git protocols. Concepts such as **user**, **repository**, and **project** can be captured. Respectively, these concepts represent the set of all users, repositories, and projects. Roles such as **hasOwner** represent the relationship between projects and users. This can also be defined as the TBox axioms describing the concepts and the relationships between them.

Example 2.3. *Example of TBox axioms describing concepts and relationships in the GitHub domain.*

- $project \sqsubseteq Repository$
- $GithubProject \sqsubseteq Repository \sqcap \exists hasowner.User$

The first axiom, which says that all **projects** are **repositories**, is expressed by the concept *inclusion*, saying that the concept `Repository` subsumes the concept `Project`. One of the intriguing features of DLs is the capacity to construct a statement that links concepts and roles. For example, a GitHub project, which is a **repository**, has an owner that is a **user** establishing a relationship between **repository** and a role **hasOwner**. The concept subsumption captures this relationship and conjunction in the second axiom in example 2.3. In summary, example 2.3 briefly describes a GitHub project, describing it as a `Repository` with an owner as a `User`. Furthermore, this description can be supplemented with “general knowledge” and background knowledge from the domain of interest. For example, a repository cannot be considered both private and public.

These descriptions are contained within the TBox. The ABox is the assertional part and would represent the instances or individuals for the statements. ABox axioms capture the knowledge about these instances, the concepts to which they belong, and how they relate to each other [65]. For example:

Example 2.4. Jaff owns an open-source Github project called OpenCode Ontology. The following ABox axioms describes the concept assertions contained in this description: $User(Jaff)$, $Project(OpenCode\ Ontology)$, asserting that Jaff is a user, OpenCode Ontology is the project, and a role assertion $hasOwner(OpenCode\ Ontology, Jaff)$ describing the relationship between the individual Jaff owning the project instance, OpenCode Ontology.

An overview of the semantics of DLs can be found in the literature [5, 65].

2.5 OWL

Ontologies are defined by an ontology language that allows for writing formal conceptualizations of domain models [4]. For a language to achieve this, the following requirements must be met; “a well-defined syntax, well-defined semantics, efficient reasoning support, sufficient expressive power, and convenience of expression” [4].

The Resource Description Framework (RDF) is a language for representing information about resources on the WWW, with each resource being identified by a Uniform Resource identifier [80]. RDF represents scenarios where information needs to be processed by systems or applications. RDF allows the model of basic statements about resources as a graph of nodes and arcs representing the resources and their properties and values [80]. RDF also provides an XML-based syntax (RDF/XML) for documenting and sharing these graphs and allows the representation of some ontological knowledge. Models are structured via typed hierarchies, subclass, and subproperty relationships, domain and range restrictions, and instances of classes.

For instance, with example 2.1, we can utilize RDF Schema (RDFS), which describes the vocabulary that is used in RDF descriptions [84], to model the following;

- define classes such as **user**, **repository**, **github_user**, **commit** and **cpython**;
- express that **cpython** is an instance of **repository**
- declare that **commit_belongs_to_repository** is a property relating the two classes **commit** (domain) and **repository** (range);
- express that **github_user_is_org** is a property, with **github_user** as its domain and boolean as its range.

However, to meet the requirements for an ontology language, a language that is richer than RDFS is required. Features such as a local scope of properties, class disjointness, cardinality restrictions, and special characteristics of properties [4] are required together with practical, efficient reasoners and being sufficiently expressive to express large classes of knowledge.

OWL was created to address the limitations of the RDFS. OWL distinguishes it from RDF because it upholds a rich set of inferences [48]. With an initial long list of design goals to satisfy various use cases and requirements, as outlined in [44], the expectations for OWL meant that the expressiveness required was beyond what was provided by DLs, such as coupling information with classes and properties [48].

Using OWL, we can apply a set of extensions to the RDF expressions, such as;

- state that **user** and **repository** are disjoint classes;
- declare that the class **organization** is defined exactly as those members of the class **github_user** that have “true” as a value for the **github_user_is_org** property.

The initial version of OWL, intended as an extension to RDFS, is, however, unable to meet the requirement of the trade-off between expressive power and efficient reasoning. This is due to the expressive primitives present in RDF, namely the “`rdfs: Class` (the class of all classes) and `rdf : Property` (the class of all properties)” [4]. Computational complexity increases drastically if the logic of OWL is extended with these primitives. Fulfilling each one of the requirements required for OWL at once would have created a formalism where specific reasoning problems are undecidable. This led to OWL being decoupled into three components or sublanguages to address the requirements separately while retaining upward compatibility with RDF and RDFS. In order of expressiveness, the sublanguages are OWL Full, OWL DL, and OWL Lite. OWL Full is upward compatible with RDF and RDFS, supporting all the RDF and RDFS combinations [48]. OWL DL and OWL Lite are extensions that support limited combinations of RDF and RDFS, such as not allowing classes to be used as individuals [48].

In existing work [15], the researchers investigated the data complexity of query answering using DLs. This analysis concluded that the DL-Lite family is the set of logics allowing for conjunctive query answering. The first set of DLs is custom-made for query answering over large data sources. As part of the work completed in the European TONES [92] project, the researchers employed an approach to provide relational database access through ontologies by using a combination of features and extensions from a subset of ontology languages from the DL-Lite family to achieve low computational complexity of inference. The ontology language produced as an output of this work to facilitate data access to RDMS was called DL-Lite_A. DL-Lite_A comprises features from two languages in the DL-Lite family, namely DL-Lite_F and DL-Lite_R. DL-Lite_F enables specifying the main modeling components of conceptual models, and DL-Lite_R incorporates the DL fragment of RDFS. The DL-Lite_A fragment extends DL-Lite_R with functional properties; however, to remain in LOGSPACE for query answering over large data sets it requires restrictions on the interaction between properties used in different types of axioms ².

Since 2009, there has been a second version of OWL due to the challenges faced in the initial version, OWL 1. These challenges relate to the efficiency and scalability of the reasoning process. OWL 2 [46] consists of different profiles, OWL 2 EL, OWL 2 QL, OWL 2 RL, OWL 2 DL, and OWL 2 Full, which vary in their reasoning complexity. The profiles OWL 2 EL, OWL 2 QL, and OWL 2 RL are fragments of OWL 2 that have polynomial reasoning time [105]. The reasoning complexity of OWL 2 DL is undecidable with a complexity of N2EXPTIME [105].

In the second version of OWL, the OWL 2 QL profile (based on the DL-Lite family of DLs) was designed so that the query answering is performed in LOGSPACE in relation

²https://www.w3.org/TR/owl2-profiles/#OWL_2.QL

to the size of the data source³. LOGSPACE, or Logarithmic Space, is a class of computational complexity that includes decision problems that can be solved by a Turing machine using a deterministic algorithm and a logarithmic amount of writable memory space. The design was intended to query data in a relational database management system (RDMS) through an ontology via a query rewriting component. Rewriting the query into an SQL query processed by the RDBMS engine system without affecting the underlying data. OWL 2 QL is the intersection of RDFS and OWL 2 DL, providing features to express conceptual models, including UML and ER diagrams. We note that we are using OWL 2 QL for our implementation of OBDA.

We now note the important features directly from the OWL 2 QL specification.

OWL 2 QL syntactic restrictions ⁴	
Subclass Expressions	Superclass Expressions
a class	a class
existential quantification (ObjectSomeValuesFrom) where the class is limited to <i>owl:Thing</i>	intersection (ObjectIntersectionOf)
existential quantification to a data range (DataSomeValuesFrom)	existential quantification to a class (ObjectSomeValuesFrom)
	existential quantification to a data range (DataSomeValuesFrom)

OWL 2 QL supports the following axioms in line with the syntactic restrictions.

- subclass axioms (SubClassOf)
- class expression equivalence (EquivalentClasses)
- class expression disjointness (DisjointClasses)
- inverse object properties (InverseObjectProperties)
- property inclusion (SubObjectPropertyOf not involving property chains and SubDataPropertyOf)
- property equivalence (EquivalentObjectProperties and EquivalentDataProperties)
- property domain (ObjectPropertyDomain and DataPropertyDomain)
- property range (ObjectPropertyRange and DataPropertyRange)
- disjoint properties (DisjointObjectProperties and DisjointDataProperties)
- symmetric properties (SymmetricObjectProperty)

- reflexive properties (ReflexiveObjectProperty)
- irreflexive properties (IrreflexiveObjectProperty)
- asymmetric properties (AsymmetricObjectProperty)
- assertions other than individual equality assertions and negative property assertions (DifferentIndividuals, ClassAssertion, ObjectPropertyAssertion, and DataPropertyAssertion)

The following concepts are not supported⁵.

- existential quantification to a class expression or a data range (ObjectSomeValuesFrom and DataSomeValuesFrom) in the subclass position
- self-restriction (ObjectHasSelf)
- existential quantification to an individual or a literal (ObjectHasValue, DataHasValue)
- enumeration of individuals and literals (ObjectOneOf, DataOneOf)
- universal quantification to a class expression or a data range (ObjectAllValuesFrom, DataAllValuesFrom)
- cardinality restrictions (ObjectMaxCardinality, ObjectMinCardinality, ObjectExactCardinality, DataMaxCardinality, DataMinCardinality, DataExactCardinality)
- disjunction (ObjectUnionOf, DisjointUnion, and DataUnionOf)
- property inclusions (SubObjectPropertyOf) involving property chains
- functional and inverse-functional properties (FunctionalObjectProperty, InverseFunctionalObjectProperty, and FunctionalDataProperty)
- transitive properties (TransitiveObjectProperty)
- keys (HasKey)
- individual equality assertions and negative property assertions

⁵https://www.w3.org/TR/owl2-profiles/#OWL2_QL

2.6 Connecting ontologies and databases

The approach of OBDA, also referred to as the Virtual Knowledge Graph (VKG), where utilizing an ontology to facilitate access to information, can profitably benefit enterprise data integration and the Semantic Web by enabling clients to rely on a domain vocabulary to gain access to application and system services [92]. In this technique, the ontology provides a semantic layer over the data layer, enabling a conceptual view of an information system [92]. This conceptual view creates an abstraction of the information system, hiding the details of the underlying data layer. Queries that would generally be large and complex, such as cross-database queries, can be constructed at the ontology level without the need to understand how each database schema is structured or how the data is stored [57].

To realize such a system initially required an understanding of which fragments of OWL 1 (OWL 1 DL or OWL 1 Lite) are practical in real-world systems to produce ontologies that are suitable for the real-world environment. As indicated in section 2.5, given the sheer size of big data, none of the fragments (OWL 1 DL and OWL 1 Lite) was suitable, given that both are coNP-hard in relation to data complexity. This resulted in the development of the OWL 2 QL ontology language to enable query answering over RDMS in real-world systems. To handle data access without materializing all assertions that an ontology can derive, OWL QL 2 was selected as the ontology language and identified as a suitable fragment of OWL that can achieve this. This means that data is fetched as queries are posed instead of being stored in memory. By delegating the processing of queries to the RDMS, the underlying Knowledge Graph (KG) remains virtual and exposes up-to-date information at query time.

2.6.1 SPARQL

The virtualization of a Knowledge Graph (KG) enables the querying of data using SPARQL, a W3C standard query language in the Semantic Web [12], by performing a translation of the SPARQL queries into SQL queries over data sources. SPARQL is a graph-matching query language with specifications that provides languages and protocols to query and manipulate RDF graphs⁶. It is fully integrated into the Semantic Web and expects data to be structured as RDF graphs and resources to be identified by Internationalized Resource Identifiers (IRIs) [27].

To support the heterogeneity of data on the WWW in various domains with different vocabularies, the SPARQL specification consists of four different query forms: *SELECT*,

⁶<https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/>

CONSTRUCT, *DESCRIBE*, and *ASK*. The *SELECT* and *CONSTRUCT* query forms are used where the domain vocabulary is known for the underlying data points. The difference between these forms is the format of returned results. The *SELECT* form delivers results in tabular XML format, but the *CONSTRUCT* form returns RDF data. If the vocabulary is unknown, but the IRIs are known, then the *DESCRIBE* query form is applicable. A *DESCRIBE* query returns an RDF graph describing the requested resource. When the ability to answer a specific query is unknown, an *ASK* query form is used. This query form returns “yes” or “no” depending on whether at least one answer can be given by the data endpoint [27]. Furthermore, SPARQL supports queries whose answers are not explicitly represented in the KG but which can be implicitly inferred using automated reasoning [48].

As described in [27], a SPARQL query consists of five components: optional *PREFIX* namespace declarations, a query result clause, optional *FROM* or *FROM NAMED* clauses, a *WHERE* clause, and optional query modifiers.

- The optional *PREFIX* declarations are used to introduce shortened namespaces for IRIs, similar to that of XML namespaces.
- The query result clause can take one of the four forms *SELECT*, *CONSTRUCT*, *DESCRIBE*, and *ASK*.
- The optional *FROM* or *FROM NAMED* clauses defines the dataset being queried.
- The *WHERE* clause is specified in a triple pattern set to select the triples forming the result.

Let us demonstrate a SPARQL *SELECT* query with an example based on a sample GitHub User database table represented as a KG.

TABLE 2.1: Sample data from GitHub User table

id	login	city	...
1011	tosch	Hiroshima	...
1042	jmettraux	City of Johannesburg	...

```
User:1011 User:login "tosch".
User:1011 User:city "Hiroshima".
User:1042 User:login "jmettraux".
User:1042 User:city "City of Johannesburg".
```

LISTING 2.1: Data assertions from table 2.1 in RDF triple format

Table 2.1 represents a sample dataset from the GitHub User table. We show how this table can be represented in the RDF format, using “triples” in listing 2.1. An RDF triple, or semantic triple, consists of three components⁷:

- the subject, which is an RDF URI reference or a blank node
- the predicate, which is an RDF URI reference
- the object, which is an RDF URI reference, a literal or a blank node

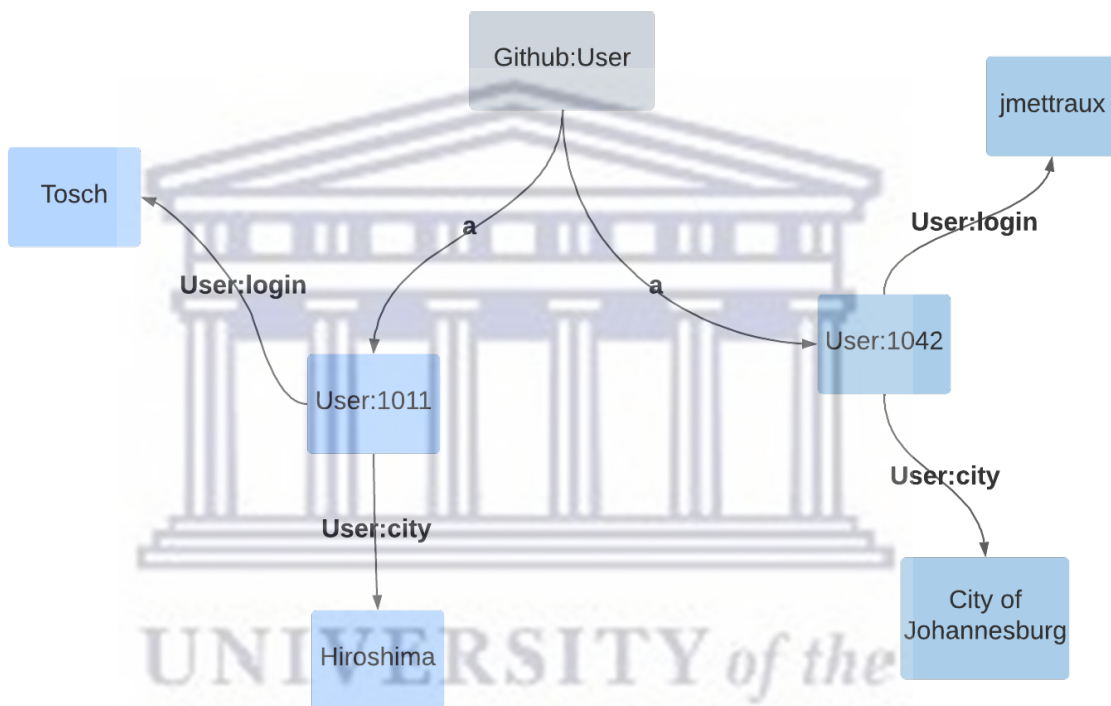


FIGURE 2.2: RDF graph model based on table 2.1

A graph structure is formed when a set of triples are joined together, with the subjects and objects as nodes and the predicates as edges. In figure 2.2, we show this data model in a graph structure.

```

# User is a class
  Github:User    a owl:Class .
# login is a property
  User:login    a owl:Property .
  User:login    rdfs:domain Github:User .
  User:login    rdfs:range xsd:string .
# city is a property
  User:city     a owl:Property .
  User:city     rdfs:domain Github:User .
  
```

⁷<https://www.w3.org/TR/rdf-concepts/>

```

    User:city      rdfs:range xsd:string.
# Instances
    User:1011 a GitHub:User.
    User:1042 a GitHub:User.
# Properties
    User:1011 User:login "tosch".
    User:1011 User:city "Hiroshima".

    User:1042 User:login "jmettraux".
    User:1042 User:city "City of Johannesburg".

```

LISTING 2.2: RDFS (RDF Schema) representation for table 2.1 schema

In this context, the triples represent the database table row data. However, we can also model the table schema in RDFS as shown in listing 2.2. Here, the User table schema and the data are defined in a single specification. Listing 2.2 defines the User class and each property (*login and city*) it contains. The subject is explicitly defined as the domain (User table), while the range indicates the data type.

```

select ?userLogin ?userCity
from graph:GitHub
where {
    ?user a GitHub:User.
    ?user User:login ?userLogin.
    ?user User:city ?userCity.
    ?userCity User:city 'Hiroshima'}.
}

```

LISTING 2.3: SPARQL user query

TABLE 2.2: Listing 2.3 result

<i>userLogin</i>	<i>userCity</i>
tosch	Hiroshima

In listing 2.3, we show a SPARQL query, where each line in the query is a pattern to be matched from the database. The query's body is a collection of triples with variables preceded by the "?" symbol through a *SELECT* operator. First, variables are assigned values so that the query body triples match the KG's triples. The query answer is constructed by processing the assigned variable values. We only select the *userLogin* and *userCity* variables in this case. The answer to this query can be seen in table 2.2.

2.6.2 OBDA Framework

The OBDA framework consists of an extensional instance, the data source, an intensional schema, the ontology [116], and the link between the two consisting of a mapping

specification.

Definition 2.3. Formally, the extensional instance is represented as the *data source* \mathbf{D} conforming to the data source schema \mathbf{S} . The intensional schema is defined as the OBDA specification $\mathbf{P} = (\mathbf{O}, \mathbf{M}, \mathbf{S})$ [116] where,

- \mathbf{O} is an ontology
- \mathbf{M} a mapping from \mathbf{S} to \mathbf{O}
- \mathbf{S} the data source schema

An OBDA specification \mathbf{P} is instantiated by a database \mathbf{D} compliant with the schema \mathbf{S} . The pair (\mathbf{P}, \mathbf{D}) is an OBDA instance or an instance of a VKG. The RDF graph, denoted $\mathbf{M}(\mathbf{D})$, is the set of triples produced by combining \mathbf{M} and \mathbf{D} . Thus, the exposed virtual RDF graph, denoted $\mathbf{G}_{\mathbf{P},\mathbf{D}}$, provides the semantics of an OBDA instance (\mathbf{P}, \mathbf{D}) and comprises the triples derived from the triples in $\mathbf{M}(\mathbf{D})$ by applying the axioms in \mathbf{O} [117].

2.6.3 Query answering

The most fundamental reasoning task in the OBDA approach is query answering over the KG [116]. Query answering is performed by utilizing SPARQL as a query language. A SPARQL query \mathbf{q} over the OBDA instance (\mathbf{P}, \mathbf{D}) essentially returns the answer to \mathbf{q} over the KG $\mathbf{G}_{\mathbf{P},\mathbf{D}}$, inline with the standard SPARQL semantics [117]. The primary method for query answering in this approach is query reformulation, which prevents physical materialization of the KG $\mathbf{G}_{\mathbf{P},\mathbf{D}}$. The SPARQL query \mathbf{q} expressed over the KG is reformulated into a SQL query \mathbf{Q} that can be directly executed on \mathbf{D} [117]. During the query reformulation process, the SPARQL query \mathbf{q} is processed through a set of transformations, which include rewriting the query \mathbf{q} with respect to the ontology \mathbf{O} and unfolding it inline with the mapping \mathbf{M} . The answers returned by the SQL query \mathbf{Q} , after execution on \mathbf{D} , are returned and transformed into RDF terms based on the mapping \mathbf{M} .

In practice, a direct implementation of query rewriting and unfolding suffers from high computational costs. Many optimizations have been developed to improve performance, such as compiling the ontology and mappings offline during the bootstrap phase, utilizing database constraints to simplify queries, or using a query cost estimation and selecting the appropriate rewrite mechanisms. Optimization techniques vary between various OBDA system implementations and are not discussed in detail here. We will discuss the

relevant optimization performed on the selected OBDA tool for this research in Chapter 4.

2.6.4 Mapping

The mapping specification is the most complex component that is at the core of an OBDA system. The mapping \mathbf{M} connecting the ontology \mathbf{O} to the database is responsible for specifying how the ontology assertions are populated by the data from the source \mathbf{D} . It is considered the most complicated part of setting up an OBDA system as it involves writing individual queries consistent with the ontology's vocabulary for each database table and column [12]. While the development and maintenance of ontologies is a well-established topic with considerable research [107], the engineering of mapping specifications is still an emerging technology. Given the complexity, as stated above, mapping engineering is a tedious and demanding procedure. It requires deep knowledge of both the domain of interest and how the underlying data sources are structured. Several mapping engineering methodologies and tools have been proposed to address this challenge. The authors in [117] group the contributions into two categories: mapping bootstrappers and editors.

A mapping bootstrapper attempts to automate or semi-automate a mapping specification for a relational data source. This is often based on the W3C direct mapping (DM) standard⁸, which defines an RDF graph representation of the data in a relational database. Following a predetermined set of rules, DM specifies how to generate the appropriate RDF graph, mapping a table to a novel class, a column to a novel data property, and a foreign key to a novel object property [117]. However, the generated ontology and mappings are data source specific, whereas a domain ontology aims at being usable across multiple data sources within a domain.

Mapping editors are either textual or graphical. Based on the W3C RDB2RDF Mapping Language (R2RML) (a W3C standard for mapping relational databases to RDF data sets) or alternative syntax. These languages are widely used in ontology editors like Protégé⁹ or text editors like Stardog Studio¹⁰. Although the text editors provide an environment for mapping engineering, they do not support features such as syntax highlighting and require deep-level knowledge about the underlying mapping language [117]. In graphical editors, users define mappings using a user interface (UI); however, this approach suffers from a lack of intuitive UI design and overloads the user (mapping engineer) with information in the UI [117].

⁸<https://www.w3.org/TR/rdb-direct-mapping/intro>

⁹<https://protege.stanford.edu/>

¹⁰<https://www.stardog.com/>

2.7 Conclusion

In this chapter, we discussed and highlighted the key theoretical background of OBDA. We briefly discussed the Semantic Web, which encompasses this technique. We discussed the concept of an ontology and the language employed to define it. Finally, we discussed the steps to make this viable in a real-world setting, the challenges, and how ontologies are employed to provide data access utilizing a mapping specification.



Chapter 3

Literature Review

3.1 Introduction

Applications of knowledge graphs are gradually gaining momentum due to their agility and flexibility to apply to various data models [18]. This flexibility enables the integration of heterogeneous sources and data schemas. Throughout recent years, much attention has been on converting legacy data to RDF knowledge graphs. Given the broad impact and implementations of relational database management systems (RDMS), naturally, the focus shifted in this direction. The two main approaches for this were to materialize all data within a given data source as RDF triples or on-the-fly data access using a query language such as SPARQL and delegating the actual retrieval of the data to the data source engine [82]. The latter is called the Virtual Knowledge Graph (VKG) approach. After converting data into knowledge graphs, processing is done using domain ontologies with automated reasoning capabilities. In this chapter, a review of past implementations in the area of semantic data access using the knowledge graph approach is carried out.

This chapter is organized as follows: Section 3.2 discusses the methodology used to survey and review the literature, which defines the discussion framework used in all subsequent sections; Section 3.3 describes the relevant use cases and data sets; Section 3.4 describes the objective in the various papers; Sections 3.5 – 3.7 describe the implementation approaches, systems, and ontologies used; Section 3.8 summarizes the results obtained in various studies; and Section 3.9 then provides a discussion of the procedures used, explicitly pointing out the challenges as reported in the papers. Finally, we conclude in Section 3.10

3.2 Literature Review Methodology

This chapter reviews relevant literature to reveal the different implementation approaches authors take to enable Ontology-based data access (OBDA) or Ontology-based data integration (OBDI) to various data sources. The review covers several aspects that relate to the use case, objectives, challenges, and approaches used in various domains devised in the literature for this purpose. The review focuses explicitly on implementations of OBDA and OBDI on relational and heterogeneous data sources.

Google Scholar was used to collect papers from 2017 to 2023, with the following keywords: (“Ontology-based data access” AND “relational databases” AND “data integration”) AND (“Ontology-based data access” AND “OBDA systems” AND “large-scale”) AND (“Ontology-based data access” AND “scalability” AND “performance”). The results obtained for these queries were about 1213 in total, and of these, only papers were selected with an implementation use case applying OBDA or OBDI. We excluded workshop papers and removed duplicate papers. Even though this review focuses on relational data sources and OBDA, the methods used to achieve this can be extended and applied to other data sources and data processing approaches. Hence, the conclusions and discussions of this chapter can be adapted to other approaches.

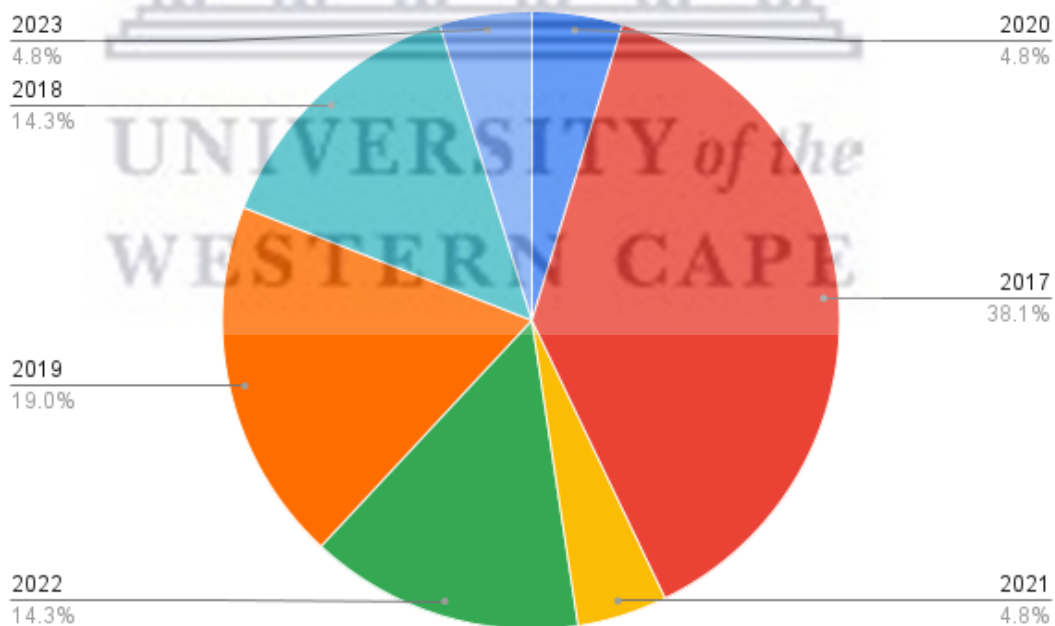


FIGURE 3.1: Percentage of reviewed papers by year

The total number of reviewed papers is 20, combining conferences and journal papers published from 2017–2023. We note that we only looked at recent work while investigating the topic of OBDA in this context. We explored older papers before 2017 to

understand the domain, such as [11, 58, 92]. However, for this review, we only focused from 2017 onwards to focus on more recent work in this area. Figure 3.1 shows the reviewed studies annually.

Figure 3.2 shows the generic structure of OBDA or OBDI systems. This structure, together with our interest in the scalability of such systems in practice, was used as a guideline to construct a set of questions to categorize and analyze the 20 papers systematically. The questions are as follows:

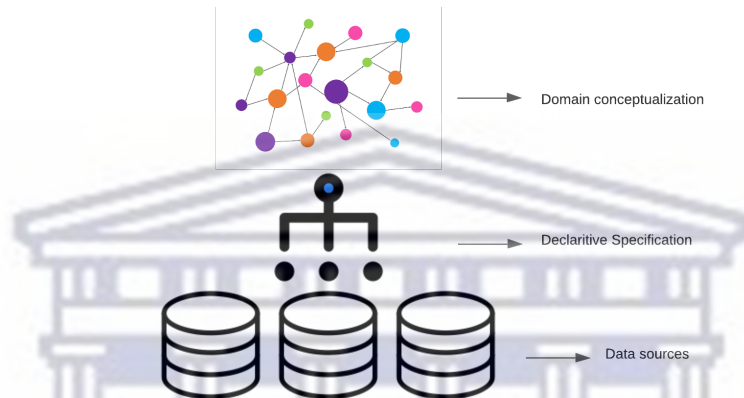


FIGURE 3.2: OBDA/OBDI generic structure

1. What are the real-world use cases and objectives of OBDA systems in various domains?
2. What are the data sources and techniques used for ontology and mapping engineering?
3. What are the query processing algorithms and optimization strategies employed in OBDA implementations?
4. What scalability challenges arise when dealing with large ontologies and datasets?
5. What evaluation criteria are used to assess the performance of OBDA systems, and what are their limitations and gaps?
6. What are the current challenges and open research questions in OBDA, and what future research directions can address these challenges?

These questions provide the framework for the rest of this paper. Sections 3.3–3.8 address questions 1–7 in sequence. Section 3.8 discusses the findings in the previous seven sections. Section 3.10 provides conclusions.

3.3 Use cases

This section provides a breakdown of use cases in the 20 studies reviewed, which we categorize by domain. The general use case of OBDA is for data access or integration across heterogeneous data sources. We now discuss how this is applied in various domains. Figure 3.3 provides a breakdown of the domains in the review.

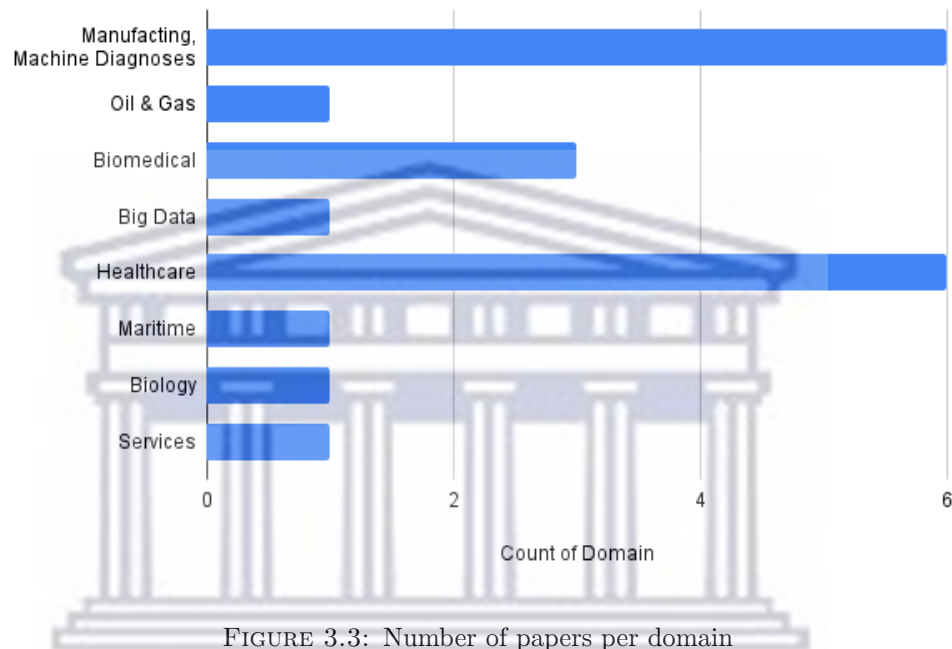


FIGURE 3.3: Number of papers per domain

3.3.1 Manufacturing/Machine Diagnoses

The studies in the manufacturing domain focus on improving existing production by utilizing OBDA to enhance data access and integration to improve manufacturing production quality. The data used are heterogeneous and generated from manufacturing processes and equipment, such as its configuration, location, and weariness. In environments where manufacturing equipment is integrated with IoT devices, OBDA is used to aggregate and integrate data using background knowledge about factory equipment, such as locations of sensors, structure, and characteristics of conveyors to manage energy consumption and management of tools. Furthermore, OBDA is applied to solve existing bottlenecks, such as accessing and integrating heterogeneous data. 30% of the papers in this review are in the domain of manufacturing [53, 60–62, 91, 98].

3.3.2 Oil and Gas

We have one study in the Oil and Gas industry where exploratory geologists are expected to locate new exploitable accumulations of oil or gas in specific locations by analyzing data about these areas on time. Such data sources are frequently dispersed across heterogeneous and self-evolving systems or have been changed over time to meet the needs of the applications they support. The study aims to create a solution to the data access problem at scale while keeping the constraints of OBDA in mind [59].

3.3.3 Biomedical

The biomedical domain consists of various disparate data sources, such as data from various proteomics studies or patient health records dispersed in multiple databases and document repositories. From the three studies we reviewed, the focus was on linking multiple data sources to assist biologists in obtaining relevant knowledge from diverse data sources to understand and explain biological processes of interest [86], as well as assisting healthcare professionals who are unable to locate important information in a fast and error-free manner [109, 112].

3.3.4 Biology

The biology domain contains a vast amount of data sources. The use case of OBDA in this context is the integration and semantic enrichment of heterogeneous biological databases. Furthermore, by linking multiple biological databases, researchers will be able to jointly query (i.e., conjunctive queries) these databases using a single query language [104].

3.3.5 Healthcare

The healthcare domain provides several use cases for OBDA. There is an emphasis on establishing a knowledge base to enable decision-making and analysis in epidemic science, focusing on spatio-temporal and social reasoning. Access to diverse rare disease datasets and semantic-level data integration is critical for advancing research and safeguarding sensitive data. Furthermore, the incorporation of ontology reasoning skills into medical information databases intends to facilitate data retrieval and querying of patient records from disparate sources. Data integration, particularly through ontology-based methodologies, is critical for retrieving diverse healthcare data more quickly and easily. The significance of semantic interoperability in post-genomic clinical trials involving

various universities is highlighted. Relational Databases to RDF (RDB2RDF) systems provide real-time transformation of RDF datasets to maintain data freshness and provide an integrated view. 30% of the papers in this review are from the healthcare domain [34, 43, 85, 93, 110, 120].

3.3.6 Services

Organizations in the Services domain focus more on understanding the value of examining how their business processes are carried out in the real world to drive continuous improvement. Process mining is an approach that has gained traction to address this and is highlighted in the paper [17] we reviewed in this domain. For more information on process mining, we direct the user to [111]. In the paper we reviewed, OBDA is applied in this scenario where event logs generated from business processes are examined and extracted from relational data sources. An ontology describes the logs and is used within an OBDA system to extract enriched log information from the raw data.

3.3.7 Maritime

In the Maritime domain, data retrieval, integration, and reasoning with these data sets are complex due to the variety of data in diverse sources, the heterogeneity of data formats, and the volume of data [97]. In this context, the OBDA/OBDI approach supports using distributed knowledge bases, which use modular ontologies for data retrieval, integration, and reasoning from diverse and heterogeneous data sources. Allowing for delegating data retrieval, integration, and reasoning responsibilities [97].

3.3.8 Big Data

Since the advent of Big Data, heterogeneous data has become more prevalent. The paper we reviewed in this domain focused on providing implementation details of applying the OBDA paradigm to the NoSQL and Data Lake technology field. Cassandra, MongoDB, Couchbase, and Neo4j are examples of non-relational or NoSQL databases. This diversity contributes to one of the most significant Big Data challenges: variety. In the paper reviewed, the authors give a detailed implementation of applying the OBDA to the NoSQL and Data Lake domain [79].

3.4 Data sources, Ontologies and Mappings

This section summarizes the data sources, approaches to developing ontologies, and mappings used in the 20 studies reviewed.

3.4.1 Data sources

The data sources in the review include relational and non-relational databases. Data is stored in file formats such as Excel, CSV, and the Hierarchical Data Format version 5 (HDF5) file formats. Additional sources include real-time streaming sensor data and distributed data. The data sources are based on real-world data, except the study in [43], which includes a synthetic dataset. Different approaches to data management were performed, such as the study in [53], which performs Extract, Transform, and Load (ETL) to raw data and stores them within a database that is used in the OBDA system, where other studies work with multiple data sources without moving or transforming the data.

3.4.2 Ontologies

Approximately 75% of the studies document the manual development of ontologies as part of the OBDA solution [17, 34, 53, 59–62, 85, 86, 91, 97, 98, 104, 109, 110, 120, 121]. Development includes workshops with domain experts and technical staff to define the conceptual model and generally occurs over a lengthy period. 15% of the studies reused existing ontologies [79, 93, 104, 112], and one study did not use a conceptual model and opted for a solution that generates an RDF graph directly from the data sources [43].

3.4.3 Mappings

The RDF Mapping Language (RML) or RDB to RDF Mapping Language (R2RML) is widely used to map the data sources to the conceptual model in the reviewed papers. A few papers take a different approach, such as mapping to streaming data and supporting real-time analytics [60, 61]. In [34], the authors developed a bespoke SQL generator that uses database-specific mappings based on the structure of the database. The authors in [43] utilized the mapping language from the D2RQ platform. The D2RQ System is an implementation that allows virtual access to relational databases as read-only RDF graphs¹. Additionally, two studies used software-based solutions to map data to the

¹<http://d2rq.org/>

relevant ontology using the Apache Jena open-source framework². The study in [110] implemented a software solution using the Jena API to create mappings from a set of mapping rules based on a global schema of three different data sources. In [109], the authors mapped the data to the ontology using a Java-based software tool and stored the resultant RDF data in the Triple Database (TDB) triple store. TDB is a component of Jena for RDF storage and query³.

3.5 Optimization

The query volume, the size, and complexity of the data sources, ontology, mappings, and the stability and performance of the underlying system all impact scalability. Several approaches are used in the literature to optimize the performance of OBDA implementations according to the domain and presented challenges.

Optimizations focus on query rewriting, unfolding, and execution, given that OBDA query processing involves rewriting, unfolding, and query execution [59]. Query rewriting and unfolding often return duplicate results due to redundancy and inefficiency of rewriting/unfolding. This is due to ontology classes or properties that can participate in multiple mappings via multiple sub-classes [59].

In implementations where relational databases are used, the optimizations are applied at the database level using constraints such as primary key and foreign key definitions and strategic indexes to speed up query lookups. This is especially prevalent where OBDA tools are used, such as Ontop⁴. We provide more details of Ontop in chapter 4 section 4.2.1. The database optimizations assist the query rewriting procedure based on the mapping specifications as the constraints are used to generate the optimal queries, such as avoiding self-joins.

Distributed computing is crucial in heterogeneous environments where data are in multiple sources. In these settings, optimizations involve optimizing storage and query execution, as well as caching techniques. This includes running OBDA systems within a cluster and utilizing the Hadoop Distributed File System (HDFS)⁵ and SPARK⁶ distributed analytics engine.

A direct comparison of these approaches across the literature is difficult since each paper implementation is different, with varied requirements and operating environments.

²<https://jena.apache.org/>

³<https://jena.apache.org/documentation/tdb/>

⁴<https://ontop-vkg.org/>

⁵<https://hadoop.apache.org/>

⁶<https://spark.apache.org/>

However, analyses per domain and underlying system environment can be a research topic. We did not explore this in the literature review.

We summarise the optimizations from the literature in table [A.2](#).

3.6 Evaluation and Results

Based on the literature, we find three primary metrics to evaluate the OBDA/OBDI system implementations. These are query efficiency, effectiveness, and usability of the system. 50% of the papers we reviewed looked at the performance of the system by measuring the run time of queries with relevance to the scale of the data [[53](#), [59–62](#), [79](#), [86](#), [93](#), [98](#), [104](#)]. 15% performed evaluations based on the effectiveness [[97](#), [109](#), [110](#), [121](#)] and usability [[62](#), [91](#), [112](#)] respectively. The effectiveness is based on how well the system solves the particular use cases, and usability evaluation was done by involving key stakeholders in the evaluation phase. One paper focused on evaluating the effectiveness of the mapping technique [[43](#)], and 15% did not mention any evaluation [[17](#), [34](#), [85](#)]. We note that the study in [[62](#)] evaluated both query performance and usability of the system.

The reported results indicate satisfactory performance and motivate the potential of OBDA and OBDI. Generally, query performance grows linearly with respect to the size of the data. For large queries, query execution is slower but completed in a reasonable time, given the context of practical use cases. Only one paper [[97](#)] reported on the computational efficiency of reasoning over the data sources. The authors applied reasoning to identify complex events while retrieving data from multiple sources. The data retrieval process took place at various intervals within a distributed framework. Although this process was the most demanding task during runtime for incoming data, it did not compromise the system's overall effectiveness. More information about this approach is available in the same source [[97](#)].

Finally, we note that the environment, deployment configuration, and optimizations all impact the results, and we did not do a comparative analysis given the difficulty of this task. We summarise the evaluations and results from the literature in table [A.3](#).

3.7 Discussion

The preceding sections demonstrate a notable, expanding literature on OBDA covering a wide range of domains, data sources, and contexts. Based on the literature, from a top-down view, the OBDA approach can be divided into the “materialized” and “virtual” approaches. The materialized approach stores the underlying data as a knowledge graph in RDF triple stores. In contrast, the virtual approach maintains the data in its original location and relies on query translation techniques to query the data sources. Utilizing tools that encapsulate OBDA is becoming more prevalent, given 50% of the papers utilize tools such as Ontop, Squerall, and Optique.

The results indicate that delivering a practical application of OBDA requires a robust implementation, given that the scalability is affected by the volume of queries, the complexity and scope of the ontology, and the stability and performance of the underlying system.

We are also interested in reasoning over data sources with this approach; however, this was out of the scope of this review. We note that only one paper [97] reported on reasoning over the data sources. We consider this as future work on how reasoning challenges are addressed in practical settings.

We summarize the reviewed papers in appendix A.

3.8 Conclusion

In OBDA, using knowledge graphs to access heterogeneous data, 20 relevant papers are reviewed by examining the use cases, the data sources, ontology(s) and mappings used, optimization, evaluation, and finally, the results. Implementations of OBDA are applied in different domains with varied objectives and different approaches based on the environment. The OBDA studies considered have limited effectiveness in solving the overall challenges of data access and integration at scale. The results from the studies indicate query performance growing linearly as the data scales and requires bespoke optimization solutions based on the domain, the use case, and the configuration of computational infrastructure. This review aims to make researchers aware of the performance of such systems in practice and the associated challenges.

Chapter 4

Ontology-Based Data Access Tool, Dataset and Ontology

4.1 Introduction

The purpose of a Virtual Knowledge Graph (VKG) query answering system, synonymous with an Ontology-Based Data Access (OBDA) system, is to provide access to various data sources. In the context of this research, we focus on access to a relational database by utilizing an ontology. The purpose of the ontology is to provide a domain vocabulary familiar to users and to provide additional background domain knowledge to the underlying data. Using a mapping specification, the ontology vocabulary's terms are associated with the specific data source. Consequently, an OBDA system has the following components: a) queries describing the data requirements of users, b) an ontology, c) a mapping specification, and d) the data sources [119]. In this chapter, we document the components we selected for investigating and implementing OBDA, as well as the OBDA tool, the dataset, and the ontology. We discuss the selected OBDA tool, Ontop¹, the GHTorrent² dataset and the Semantic Git (SemantGit)³ ontology. The chapter is structured accordingly. We describe each of these, outlining the development history and why we opted for these. The implemented mapping specification and user queries are discussed in Chapter 5, where we outline the implementation using the components discussed in this chapter.

¹<https://ontop-vkg.org>

²<https://ghtorrent.org>

³<https://github.com/SemantGit/SemantGit/tree/master/Documentation/ontology>

4.2 OBDA tool

In both academia and industry, more than a dozen Virtual Knowledge Graph (VKG) query answering systems have been developed [117]. To select a suitable query-answering system for our implementation, we looked at systems that are open-source with the ability to perform ontological reasoning. In Xiao et al. [117], the authors reported on the most important query-answering systems that are compliant with industrial standards and in terms of query performance. The report includes systems that are both open-source and proprietary, irrespective of ontological reasoning capacity. The systems include D2RQ⁴, Mastro [13], Morph [94], Ontop [12], Oracle Spatial and Graph⁵, Stardog⁶ and Ultrawrap [101]. From this list of query answering systems, D2RQ, Morph and Ontop are open-source.

D2RQ is a framework for accessing relational databases using virtual read-only RDF graphs. It provides RDF-based access to relational database material without requiring replication into an RDF store. The core feature is a declarative mapping language that defines the relation between an ontology and a relational database. It has an engine that integrates with the Jena Semantic Web toolkit to enable mappings to be used to rewrite Jena API calls to SQL queries. We direct the user to <http://d2rq.org/> for further details.

Morph, formerly ODEMapster, is an OBDA system that transforms relational databases into RDF (RDB2RDF). The Ontology Engineering Group developed it based on the R2RML specification⁷. Morph optimizes SQL queries using techniques like self-join removal and elimination. Real-world queries from various Spanish and EU projects have tested Morph's capabilities⁸. For more details on Morph, refer to [94].

We note that both D2RQ and Morph projects do not support ontology inference and have not actively been maintained since January 2015 and June 2022 respectively. Given the lack of inference support, we opted for the Ontop system as the tool of choice.

The D2RQ and Morph projects lack support for ontology inference and have not received active maintenance since January 2015 and June 2022, respectively. Due to this absence of inference support, we have chosen the Ontop system as our preferred tool.

⁴<http://d2rq.org/>

⁵<https://www.oracle.com/database/technologies/spatialandgraph.html>

⁶<https://www.stardog.com/>

⁷<http://www.w3.org/TR/r2rml/>

⁸<https://github.com/oeg-upm/morph-rdb>

4.2.1 Ontop system

Ontop is an open-source, Java-based OBDA system released under the Apache 2 license. It has been developed at the Free University of Bozen-Bolzano and is commercially supported by the company Ontopic [117]. By utilizing mappings to connect the terms (classes and properties) in an ontology to the data sources, the Ontop system makes relational databases accessible as virtual RDF graphs [12]. Ontop has undergone four major releases since its inception in 2009 and is still actively maintained, establishing it as the most mature and state-of-the-art OBDA open-source system [119].

The first major release, Ontop v1, was based on answering queries, specifically conjunctive queries (CQs). The queries consist of conjunctions of unary and binary atoms for class and property assertions [119]. OWL QL 2 was used as the ontology language and identified as a suitable fragment of OWL that can be handled by VKG systems without the need for materializing all assertions that can be derived from the ontology. In this version, the mapping specification was based on a Datalog rewriting algorithm that compiles a conjunctive query (CQ) and an OWL 2 QL ontology into a union of CQs. Datalog is a declarative logic programming language that has been used in deductive database work and various other data access-related applications [78]. At query time, the algorithm translates CQs based on OWL 2 QL ontologies into SQL queries. When the generated CQs are evaluated over the database, it yields the same results as the CQ mediated by the OWL 2 QL ontology [119]. In the process of rewriting, “query atoms can be replaced by their definitions from the mapping” [119]. This is also known as query unfolding. To achieve efficient query performance, v1 relied upon Semantic Query Optimization (SQO). SQO is the semantic analysis of SQL queries and the use of database integrity constraints, such as primary and foreign keys, to reduce the size and complexity of queries [12]. An interesting observation by the authors in [12] is that even though rewriting and unfolding steps are considered distinct steps from a theoretical point of view, they should be combined in practice. For example, a mapping can be combined with the subclass and sub-property relations of the ontology, and the generated mapping specification (or T-mapping) can be constructed and optimized before any query is processed, performing the expensive SQO only during the bootstrap process.

Subsequent versions of Ontop support the W3C recommendations for SPARQL and the W3C RDB2RDF Mapping Language (R2RML) mappings. R2RML is a W3C standard for mapping relational databases to RDF data sets. R2RML is widely used and part of a collection of two standards to map the data of relational databases to RDF. To achieve these recommendations, various challenges arose due to the Datalog implementation of v1. To support the standardized mapping specification recommendations, the evolution

to a variant of relational algebra in place of the Datalog approach was initiated. Supporting non-monotonic (OPTIONAL, MINUS), cardinality (DISTINCT), and aggregation (SUM, MIN, MAX, GROUP BY) features are difficult to model since SPARQL is based on a rich algebra that is beyond the expressivity of CQs with Datalog. The challenges are further described in [119]. As a consequence, a large portion of Ontop was rewritten with Datalog being replaced with “a relational-algebra-type representation” [119]. The Intermediate Query (IQ) language, an algebra-based data structure that unifies both SPARQL and relational algebra, was the result of the rewrite and was released in 2019 as the third stable major release of Ontop, v3. In 2020, Ontop v4 was released following compliance improvements and additional features that were added to v3. We now discuss the core features in Ontop v4 based on the work of Xiao et al. [119], specifically query representation, SPARQL to SQL translation, and query optimization.

4.2.2 Query representation

Ontop represents queries by encoding them in the IQ language [118]. The IQ language provides a consistent representation from the mapping for both user SPARQL queries and generated SQL queries. In the IQ language, RDF datasets are modeled following the triples and a quaternary relation quad model in SPARQL, where a set of triples are in the form $(\mathbf{s-p-o})$ and a collection of these sets in a named graph \mathbf{g} are represented as quadruples in the form $(\mathbf{s-p-o-g})$. Similarly, Ontop models this by using atomic expressions in the form $\mathbf{triple(s, p, o)}$ and $\mathbf{quad(s, p, o, g)}$, where $\mathbf{s, p, o}$, and \mathbf{g} are constants or variables. In relational algebra, these expressions would have to be constructed by combining the SELECTION and PROJECTION operators. PROJECTION is used to *project* (π), the required attributes of distinct data (tuples) from a relation. SELECTION is used to *select* (σ), the required data (tuples) from a relation with optional conditions, where π is used for variable names and σ to handle constants and variable matching [119].

4.2.3 SPARQL to SQL translation

Regarding relational algebra expressions, the Ontop system uses a compact representation of queries to encode SPARQL queries. Based on [119], we highlight the features in the Ontop system for query translation based on the following mapping syntax,

$$T_1(x, y) \rightsquigarrow: b\{x\} : p \quad y, \quad T_2(x, y) \rightsquigarrow: b\{x\} : q \quad y$$

where T_1 and T_2 are database tables with x and y being the attributes or columns, the first attribute for both tables being the primary key of type TEXT and the second attribute non-nullable and of type TEXT and DECIMAL for T_1 and T_2 respectively. Translating the mappings on the left side of the \rightsquigarrow into an IQ produces SQL queries in the form of atomic expressions $T_1(x, y)$ and $T_2(x, y)$, where the variables x and y imply the π operation in relational algebra. The right side represents the subject-predicate object mappings for the properties $:p$ and $:q$.

As noted in section 4.2.1, Ontop supports the R2RML mapping language. An R2RML mapping consists of a set of **rr :TriplesMap** classes. The **rr :TriplesMap** class has the following three properties: **rr :logicalTable**, **rr :subjectMap**, and, **rr :predicateObjectMap**. A **rr :TriplesMap** specifies a rule for translating each row of a logical table (database table) to zero or more RDF triples⁹.

- **rr :logicalTable** defines the logical table (database table).
- **rr :subjectMap** defines the target class and the URI generation format.
- **rr :predicateObjectMap** defines the target property and the object generation by means of the **rr :objectMap**, where the value of **rr :objectMap** is a **rr :constant**, **rr :column** or **rr :template**.

In R2RML, IRIs, blank nodes, and literals are constructed using templates. Also referred to as string templates, a template is a format string that can be used to construct strings from various components, including referencing database column names. R2RML templates are enclosed in curly braces¹⁰ and serve as placeholders to be replaced by values from the database. In the case of IRI templates, *safe separators*¹¹ are used to support different values of parameters to populate a template placeholder. Literals are mapped to a specific datatype called a “datatype-able” term map. However, if literals need to be constructed from more than one column, in the case of a *xsd:date* type, then a safe separator is used. For example, if a date value is spread across three integer columns (day, month, and year), the “-” separator is considered safe. In IQ, non-constant RDF terms such as namespace IRIs are not a formal part of the RDF data model and are constructed using the binary function **rdf** with a TEXT lexical value and term type as its arguments [119]. Based on the mapping syntax example, the subjects of both triples are IRIs of the same template and are built using the same template function. In the example, the IRI equates to `:b1` when $x = 1$. Before being used as lexical values, database values must be transformed into text. The DECIMAL attribute in T_2 is mapped to the

⁹<https://www.w3.org/TR/r2rml/#dfn-predicate-object-map>

¹⁰<https://www.w3.org/TR/r2rml/>

¹¹<https://www.w3.org/TR/r2rml/#dfn-safe-separator>

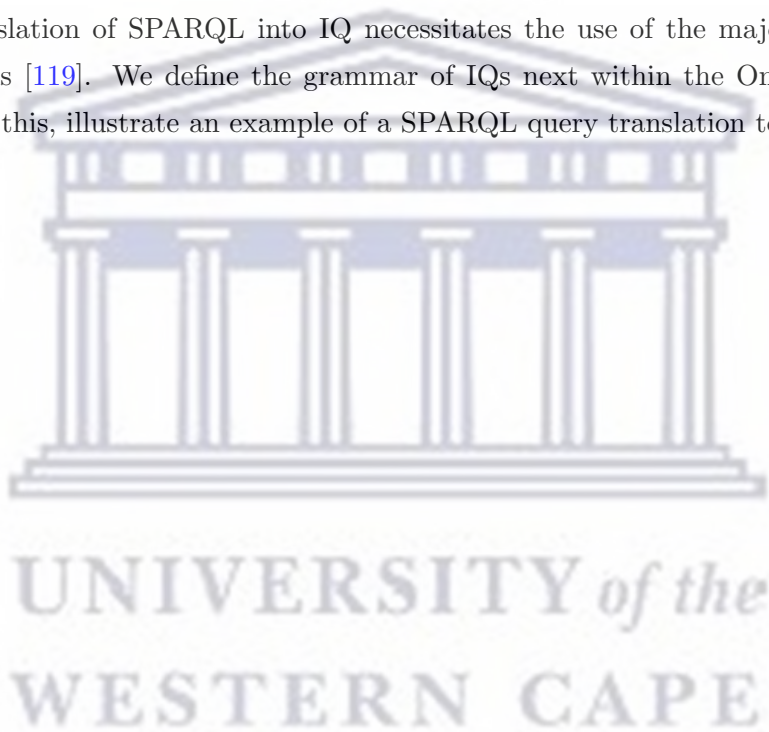
xsd:decimal type. The following is the resulting IQ representation of the mapping assertions:

$$T_1(x, y) \rightsquigarrow \text{triple}(\text{rdf} : b\{ \}(x), \text{IRI} : p \quad \text{rdf}(y, \text{xsd} : \text{string}))$$

$$T_2(x, y) \rightsquigarrow \text{triple}(\text{rdf} : b\{ \}(x), \text{IRI} : q \quad \text{rdf}(d2t(y), \text{xsd} : \text{decimal}))$$

where before being used as lexical values, the database values must be converted into text. Here, it is done by the unary function **d2t()** for the decimal value.

The translation of SPARQL into IQ necessitates the use of the majority of algebraic operations [119]. We define the grammar of IQs next within the Ontop context and, based on this, illustrate an example of a SPARQL query translation to SQL.



Definition 4.1. The following grammar describes IQs [119];

$$\begin{aligned} \phi := & P(t) \mid PROJ_{\tau}^x \phi \mid AGG_{\tau}^x \phi \mid DISTINCT \phi \mid ORDERBY_x \phi \mid \\ & SLICE_{i,j} \phi \mid FILTER_{\beta} \phi \mid JOIN_{\beta}(\phi_1, \dots, \phi_k) \mid LEFTJOIN_{\beta}(\phi_1, \phi_2) \mid \\ & UNION(\phi_1, \dots, \phi_k) \end{aligned}$$

where,

- **P** is a relation name (triple, graph instance, or a database table name)
- **t** a tuple of terms
- **x** a tuple of variables
- τ a substitution
- $i, j \in \cup \{0, +\infty\}$ are values for the offset and limit
- β a boolean term

Remark 4.2. Relations are sets of tuples in the standard relational model, which by definition do not contain “duplicate” entries [38]. However, practical implementations of relational database management systems (RDBMS) diverged from the “pure” relational model by permitting duplicate tuples in query responses, making bags (multisets) the predominant collection type in query processing rather than sets. This decision was based on the performance costs related to duplicate elimination. Instead, duplicate elimination is performed only if the user explicitly requests it via the SQL “**DISTINCT**” keyword. In line with this, the algebraic operators above operate on bags of tuples and are interpreted using bag semantics. For more background details on bag semantics, we refer the interested reader to [38] and [88] in the OBDA context.

4.2.3.1 SPARQL to IQ

We illustrate query translation in the following example with the **commit** table and a subset of the columns from the GHTorrent MySQL database instance. The following tuples can be found in a **commit**:

TABLE 4.1: Commit table

<i>id</i>	<i>sha</i>	<i>author_id</i>	<i>created_at</i>
1	ecf5851798cce783c59...	1	2012-06-01 20:33:21
2	f45f724213278770052...	1105	2012-06-23 03:39:30
3	adfed8c15dceec3c040...	3	2012-07-23 07:47:16

For this example, the mapping assumes the following ontology properties: Data properties `:commit_sha` and `:commit_created_at` maps to the “sha” and “created_at” columns, and the object property `:commit_author` maps to “author_id”. These properties construct for each commit three triples to specify the hash key identifier, the author of the commit, and the timestamp of when it was created. Based on this, the mapping specification is as follows.

$$\begin{aligned} \text{Commit}(x, h, -, -) &\rightsquigarrow \text{triple}(\text{rdf}(:c\{ \}(x), IRI), :commit_sha, \text{rdf}(h, \text{xsd} : \text{string})), \\ \text{Commit}(x, -, a, -) &\rightsquigarrow \text{triple}(\text{rdf}(:c\{ \}(x), IRI), :commit_author, \text{rdf}(:a\{ \}/\{ \}(a), IRI)), \\ \text{Commit}(x, -, -, d) &\rightsquigarrow \text{triple}(\text{rdf}(:c\{ \}(x), IRI), :commit_created_at, \text{rdf}(d, \text{xsd} : \text{date})) \end{aligned}$$

We use the following SPARQL query to determine the number of commits made by each author using these tuples.

```
SELECT ?author (COUNT(?c) AS ?cnt)
WHERE
{
  ?c a :commit
  ?c :commit_author ?author .
}
GROUP BY ?author
```

LISTING 4.1: SPARQL author commit count query

We get the following IQ upon unfolding of the SPARQL query:

$$\begin{aligned} &AGG_{cnt/SPARQL_Count(c)}^{?author} \quad JOIN \\ &PROJ_{?author,?c}^{?author/rdf(:a\{ \}/\{ \}(a),IRI), \quad ?c/rdf(:c\{ \}(x_1),IRI)} \quad \text{Commit}(x_1, -, a, -) \end{aligned}$$

After unfolding, the next step is lifting the projections (PROJ) and simplifying the functional terms.

$$\begin{aligned} &PROJ_{?author/rdf(:a\{ \}(a_1),IRI), \quad ?cnt/rdf(i2t(n),xsd:integer)}^{?author,?cnt} \quad \text{Commit}(x_1, -, a, -) \\ &AGG_{?n/Count(x_1)}^{a_1} \\ &JOIN(\text{Commit}(x_1, a_1, -, -) \end{aligned}$$

The “_” symbol is used in place of attributes/columns not projected.

4.2.3.2 IQ to SQL

The latest version of Ontop, v4, generally transforms database values into RDF terms by applying top-level projection [119]. Given that Database Management System (DBMS) vendors generally modify their SQL implementation to suit their needs, the ANSI/ISO SQL standards are only lightly adhered to, making it challenging to generate SQL interoperable across DBMS vendors. Instead, the Ontop v4 model supported each SQL dialect in a granular way, such that it supports the datatypes, conventions (attributes, table identifiers), function semantics, clause restrictions, and data catalog structure. Ontop is implemented in the Java programming language and contains Java factory classes representing the various SQL dialects, where the dialect-specific implementations are provided through dependency injection. Furthermore, Ontop supports user-defined SQL via the queries in the mapping specification [119].

```
SELECT v1.'author_id', COUNT(*) AS v0
FROM 'commit' v1
WHERE v1.'author_id' IS NOT NULL
GROUP BY v1.'author_id'
```

LISTING 4.2: Generated SQL query for listing 4.2

4.3 The GHTorrent Dataset

The primary research question, as defined in section 1.2 of chapter 1, is based on the effectiveness of the OBDA approach on real-world data. Using a real-world dataset to investigate OBDA is a primary requirement for this research, along with ensuring it contains a relational model to demonstrate OBDA in the context of RDBMS using the Ontop system.

GitHub is a popular software repository hosting platform for version control and has seen wide adoption in the last few years. It is based on the decentralized open-source and version control tool Git [73]. Git, created by Linus Torvalds, began as a revision management system for coordinating the development of the Linux kernel in 2005. Its functionality, portability, efficacy, and third-party acceptance have progressed significantly over time, making it the market leader in its domain [106]. Because Git manages the revisions, pushing a revision to a remote repository or pulling a revision from a remote repository into your local repository is seamless. However, as this scales, the maintenance of a repository and its servers can be overwhelming. Maintenance includes ensuring server connectivity, server security configuration, creating user accounts, and

providing user support. These tasks can be delegated to a third-party provider like GitHub to handle these difficulties. Software development teams typically use Git on their local machines to manage source code, while GitHub is an online service to which developers connect and synchronize their local source code changes. Containing more than 128 million open-source repositories as of February 2020¹², GitHub is one of the most significant internet sources of software artifacts [54].

Even though GitHub has a close relationship with Git, it provides many additional features specifically aimed at managing the online collaboration and social interactions of projects. We note a few of the features relevant to the dataset and provide some context.

Pull Request, also referred to as a merge request, is a request to merge code or file changes made on a separate clone or branch of the central repository into the base branch.

Watching; the watching feature allows users to subscribe to a particular repository and receive notifications for activities performed. More information on available features is available in the literature [24, 71, 72].

GitHub has over eighty-three million developers across more than four million organizations contributing to more than two-hundred million repositories¹, making it a substantial source for software repository data. Several well-known open-source projects have chosen GitHub to host their code base. These include:

- TensorFlow¹³, a Google-developed open-source software library designed for numerically intensive tasks and large-scale machine learning and deep learning support.
- Linux¹⁴, the open-source operating system created by Linus Torvalds.
- d3¹⁵, a JavaScript library for using web standards to visualize data.
- Vue¹⁶, a progressive JavaScript framework for building web-based user interfaces that can be adopted incrementally.

GitHub is a place where developers can demonstrate their skills to peers and potential employers and the platform where social coding elements were first introduced [37]. GitHub presents a wealth of research opportunities [37]. In recent years, several works have been published that focused on mining GitHub data to fulfill various research objectives. These include [2, 23, 25, 33, 51, 55, 63, 95, 115], among others.

¹²<https://towardsdatascience.com/githubs-path-to-128m-public-repositories-f6f656ab56b1>

¹<https://github.com/about>; Accessed: 2022-10-23

¹³<https://github.com/tensorflow/tensorflow>

¹⁴<https://github.com/torvalds/linux>

¹⁵<https://github.com/d3/d3>

¹⁶<https://github.com/vuejs/core>

The acquisition and curation of data from software repositories is a typical requirement to support empirical studies on software engineering [35], and GitHub is an attractive source for this as it provides access to its internal public data via a Representational state transfer (REST) application programming interface (API)¹⁷ [36]. However, access to the REST API is capped at a request limitation of 15,000 requests per hour. Given this limitation, extracting large amounts of data to support research depending on this data is a pretty cumbersome procedure. The GHTorrent project was created to grant access to public data on GitHub. It aims to provide a platform for researchers and developers to gather insights and analytics from vast amounts of open-source software data. GHTorrent is an offline mirror of Github's event streams and persistent data (for public projects) made available to the research community as a service. It was curated over several years and is still actively maintained.

4.3.1 GHTorrent Data Collection

The GHTorrent project has been mining data from GitHub since 2013 using a decentralized data collection process to collect data from the GitHub REST API. Given the challenges associated with the REST API request limit per user authentication token, the creator [35] of GHTorrent developed the process from the ground up. Data mining is performed in parallel using multiple access tokens. Collaboration between researchers is made possible through this decentralisation. To avoid duplicate requests in this workflow, a caching strategy is implemented. GHTorrent uses a MongoDB database to cache the results per entity, making it possible to query the raw data [35]. A mirroring algorithm is implemented to resolve the data into the appropriate schema structure. A recursive dependency resolution is the foundation of the mirroring algorithm, where for each retrieved entity, a set of dependencies is defined to ensure a logical flow based on the data schema figure 4.1 [35]. The result of this is a better source of structured GitHub data. The raw JSON responses returned from the REST API are stored in a MongoDB¹⁸ and MySQL¹⁹ database, respectively.

Figure 4.1 shows the MySQL database schema. This illustration is based on the MySQL dump dated June 01, 2019. In total, there are more than 125486232 repositories available in the data dump. In this research, we opted to work with the MySQL data instance since it is a relational model.

A description of the tables in the MySQL relational model:

¹⁷<https://docs.github.com/en/rest>

¹⁸<https://www.mongodb.com/>

¹⁹<https://www.mysql.com/>

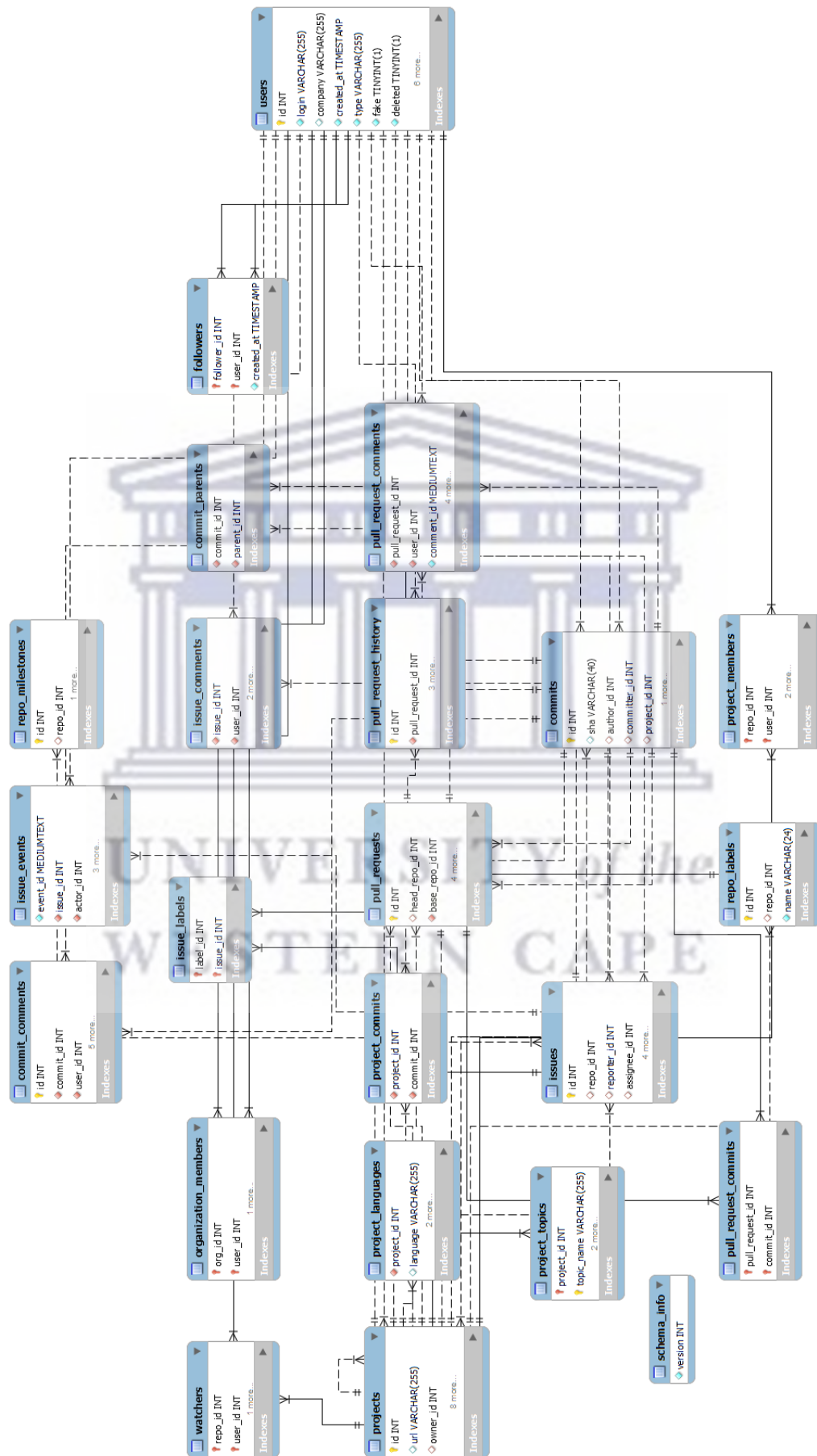


FIGURE 4.1: GHTorrent MySQL Database Schema

- The *user* table contains data about GitHub users. There are two types of users, USER and ORG, representing a user or organization, respectively. Personal data (emails and real names) are excluded from the table in the data dump.
- *organization_members* describes members (users) related to organizations.
- The *projects* table contains information about repositories.
- The *project_members* table describes users with commit access to the repository.
- The *project_languages* table describes the languages that are used in the repository.
- The *commits* table stores the unique commits.
- The *commit_parents* table stores the parent commit(s) for each commit.
- The *project_commits* table contains the commits belonging to the history of a project.
- The *commit_comments* table contains code review comments on commits.
- The *followers* table describes a follower to a user.
- The *watchers* table contains data about users that have starred (watched) a project.
- The *pull_requests* table contains information about the events where developers are ready to begin merging their code with the main project repository.
- The *pull_request_history* table contains information about the events that occurred in the lifetime of pull requests.
- The *pull_request_commits* table contains information about the commits associated with pull requests.
- The *pull_request_comments* table contains code review comments on a commit associated with pull requests.
- The *issues* table contains the issues associated with a repository.
- The *issue_events* table contains the events associated with issues on a repository.
- The *issue_comments* table contains discussion comments against pull requests or issues.
- The *repo_labels* table stores the labels to be assigned to an issue affecting a particular repository.

- The *issue_labels* table stores the labels that have been assigned to an issue.

More details are available on the GHTorrent website ²⁰.

4.3.2 GHTorrent Limitations

We now note the limitations concerning the dataset as reported in [35]:

- Entities that are added to the GitHub event stream are reported, but deletions are not. GitHub does not report item Time stamps for the watchers/stars and followers entities. As a workaround, when a follow/watch action is performed, the timestamp of the event that is generated is used by GHTorrent.
- Issues and pull requests are associated on GitHub. When a pull request is created, an associated issue is also created. As a result, pull request conversation comments must constantly be retrieved from multiple sources, namely pull request comments for code reviews and issue comments for pull requests.
- GHTorrent uses a Git user name resolution to connect a user table entry to a commit table entry. Since Git allows users to set up custom user names as their commit names, GitHub can report the same username across all entities. If the commit user cannot be resolved, for example, because the commit user does not belong to a GitHub user or the Git username is incorrectly configured, GHTorrent will generate a fake user entry with as much information as possible.
- The tracking of pull request commits is not always accurate, as they can be merged using external tools outside the GitHub environment.
- To open a bug in the GitHub bug tracker, all that is required is a textual description. Bug statuses are tracked by specific labels that are explicitly configured. This means that bugs' characteristics cannot be compared across projects similarly.
- As GitHub evolves, entity names are updated, and API endpoints get updated. For example, the watchers entity has been renamed to stargazers. If any changes affect the relational schema, subsequent data dumps will not be compatible with the previous schema.
- Network-related errors during data curation might result in missing data. Known instances of missing events include several days at the beginning of March 2012, when an error in the event mirroring script went unnoticed, and from the middle

²⁰<https://ghtorrent.org/relational.html>

of October 2012 to the middle of November 2012, when the data collection process was adapted to the newly imposed requirement for authenticated API requests.

- The `pull_request_history` and `issue_events` tables might contain duplicate records due to the REST API returning slightly modified results when they are queried at different time moments.

4.4 SemanGit Ontology

Utilizing an ontology that applies to the domain of the underlying data source is an essential step in OBDA. By applying it on the domain level, an ontology can enrich incomplete data with background domain knowledge via inferencing [117]. The GHTorrent dataset falls within the domain of version control systems, specifically the Git Version Control System (VCS). A VCS keeps track of changes made to a file or set of files over time. Selected files can be restored to their previous state using this feature, promoting easy recovery of files and errors [19]. As part of the investigative procedure to identify the dataset, we had to remember the ontology used. For this, we could develop or reuse an existing ontology for the domain of interest. While investigating a suitable dataset for the research, we found a novel RDF dataset based on the GHTorrent called Semantic Git (SemanGit). Based on a Git ontology, SemanGit is the first collection of linked data extracted from GitHub [66]. The SemanGit ontology has been identified as suitable for this research as it was developed and used as the underlying ontology for the RDF-linked dataset created from the GHTorrent dataset. As of April 2019, the SemanGit RDF dataset has over 21 billion triples. More details on creating the RDF dataset can be found in [66].

Remark 4.3. In the SemanGit project, data (in .csv file format) from monthly GHTorrent data dumps are extracted, using a Java-based software middleware, into RDF triples and stored within a graph database (triple store). In OBDA, the ontology is used directly over the data source via a mapping specification, keeping the data in its original state. Additionally, OBDA can integrate several data sources and should thus not be viewed as a specific data source.

There are various Git providers, such as GitHub, GitLab, Bitbucket, and SourceForge, to name a few. The SemanGit ontology was created with the Git protocol features as a base and additional protocol features specific to the GitHub platform. This allows the ontology to be extensible and support the unique features of specific Git providers. Currently, the SemanGit ontology is only extended to support GitHub as it was built to support the RDF format of the GHTorrent linked dataset. In order to accomplish this,

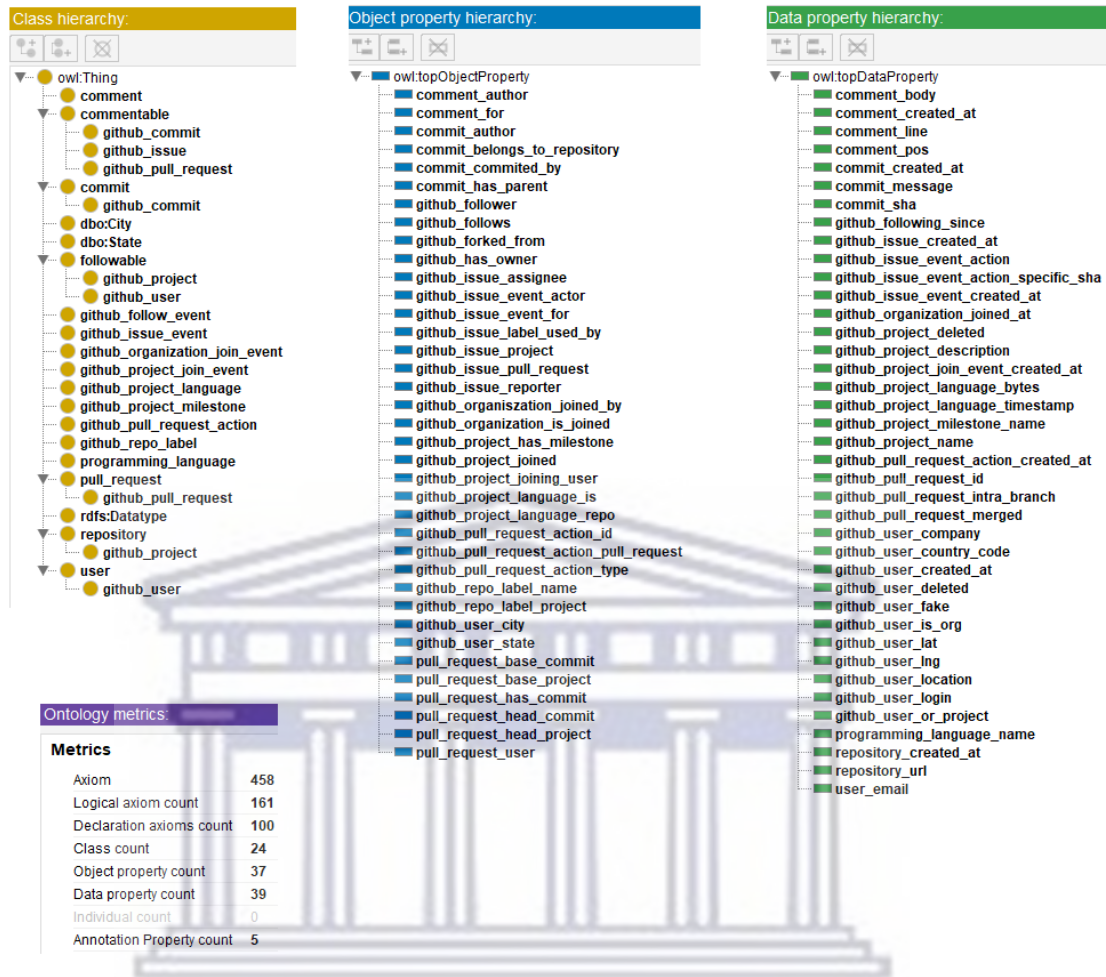


FIGURE 4.2: SemanGit Ontology summary (classes and properties)

the ontology makes a distinction between Git conventions and features that are specific to each provider. For example, based on Git, the author of a commit is represented by a “Name [email]” pair, whereas GitHub represents a commit author as a user containing additional attributes such as location, country code, creation date [66]. As a result, the ontology is hierarchical, with Git protocol features forming the base classes and provider-specific extensions as classes that inherit from them. For example, the ontology captures the Git convention `user` with a single data property `user_email` as a base class. The concept `github_user` is a subclass of `user` with all the additional object and data properties specific to GitHub. Any specific provider features not in the protocol convention form a stand-alone class that does not inherit from a Git base class. In the Semangit ontology, all the GitHub-specific classes and properties are denoted with the “*github_*” prefix.

A summary of the ontology is depicted in figure C.1. The full SemanGit ontology is available on GitHub²¹.

²¹<https://github.com/SemanGit/SemanGit/blob/master/Documentation/ontology/>

4.4.1 SemanGit limitations

The SemanGit ontology is considered to be a base ontology as it models the fundamental Git protocols with extensions to support a set of GitHub social concepts. Two types of classes can be distinguished in OWL, *Primitive* and *Defined* classes. Classes with at least one set of necessary and sufficient requirements are defined classes; they have a definition, and every individual who meets the definition belongs to the class. Primitive classes lack any sets of necessary and sufficient requirements [47]. Let us illustrate this within the context of the SemanGit ontology.

In the SemanGit ontology, *github_pull_request* is a subclass of *pull_request* which says that if something is a *github_pull_request* it is necessarily a *pull_request*. According to GitHub's REST API, every pull request (*github_pull_request*) is also considered an *issue*, but not all issues are considered pull requests. Given the ontology description, if we consider an instance of a *pull_request*, the knowledge captured is not sufficient to determine that the pull request instance is a member of the class *github_pull_request* and that it is an *issue*. We must alter the conditions to make this possible by extending the necessary conditions to necessary AND sufficient conditions. This means that the requirements for being a member of the class *github_pull_request* are not only necessary but also sufficient to establish that any given instance that satisfies the conditions must be a member of the class *github_pull_request*. Thus, the classes in the SemanGit ontology are considered to be primitive. Furthermore, the ontology lacks inverse relations and object property characteristics. Through the use of property characteristics, OWL makes it possible to enrich the meaning of properties [47]. We now discuss and outline the extensions made to enrich the ontology.

4.4.2 SemanGit extensions

The approach for extending the SemanGit ontology is based on the methodology defined in "Ontology Development 101: A Guide to Creating Your First Ontology" by Noy et al. [89]. It is an iterative development process that repeats continuously to enhance the ontology. It consists of the following sub-processes [89]:

- Determine the domain and scope by defining a set of competency questions.
- Explore the reuse of existing ontologies.
- Listing key terms in the ontology.
- Create the classes and class hierarchy.

- Create the properties of classes.
- Create features for the defined properties.
- Create instances.

While newer methodologies for developing ontologies exist [57, 64], the approach we employed sufficed for our needs as a foundational ontology already existed.

We renamed the classes and properties by removing the underscores and using Upper-CamelCase for class names and lowerCamelCase for property names. We also focused on enriching existing class and property definitions. Since we are reusing an existing ontology, the domain (*Git protocols*) and scope (*GitHub*) of the ontology are known, with the fundamental concepts being defined. Considering this, we are only focusing on the sub-processes related to the extension of class and property definitions. The instances are defined in the underlying database instance.

4.4.2.1 Class definitions

The class extensions applied were minimal. We have converted the class descriptions of *GithubProject* and *GithubPullRequest* to *definitions*.

- If something is an instance of a *GithubProject* then it is necessary that it is a *Repository* and it is also necessary that it has exactly one owner that is a member of the class *User*.
 - $GithubProject \sqsubseteq Repository \sqcap \exists hasowner.User \sqcap (= 1githubHasOwner.User)$
- If something is an instance of a *GithubPullRequest* then it is necessary that it is a *PullRequest* and it is also necessary that it has **exactly** 1 issue that is a member of the class *GithubIssue*.
 - $GithubPullRequest \sqsubseteq (PullRequest \sqcap GithubIssue) \sqcap \exists githubPullRequestIssue.GithubIssue \sqcap (= 1githubPullRequestIssue.GithubIssue)$
- Furthermore, the *User* and *Repository* classes are disjoint from each other.
 - $User \sqsubseteq \neg Repository$

4.4.2.2 Property definitions

“Each object property may have a corresponding **inverse property**. If some property links individual **a** to individual **b**, then its inverse property will link individual **b** to individual **a**” [47]. The following inverse properties were added. We note that the domain and range of each property was already defined.

- *githubOwnerOf* inverse of *githubHasOwner*
- *hasAuthoredComment* inverse of *commentAuthor*
- *hasAuthoredCommit* inverse of *commitAuthor*
- *hasCommittedCommit* inverse of *committedBy*
- *repositoryHasCommit* inverse of *belongsToRepository*

We summarize the property characteristic updates in table 4.2.

TABLE 4.2: Property characteristics

<i>Property</i>	<i>Characteristics</i>
commentAuthor	Functional
commitAuthor	Functional
commitBelongsTo_repository	Functional
commitCommittedBy	Functional
commitHasParent	Functional; Asymmetric; Irreflexive
githubForkedFrom	Functional
githubHasOwner	Functional
githubIssueEvent_actor	Functional
githubIssueEvent_for	Functional
githubIssueProject	Functional
githubIssuePull_request	Functional
githubOwnerOf	Inverse Functional
githubUserCity	Functional
githubUserState	Functional
hasAuthoredComment	Inverse Functional
hasAuthoredCommit	Inverse Functional
hasCommittedCommit	Inverse Functional
pullRequestBaseProject	Functional
pullRequestHeadProject	Functional
pullRequestUser	Functional
repositoryHasCommit	Inverse Functional

4.5 Conclusion

In this chapter, we discussed the Ontop OBDA system and the specific mechanics (theoretical and practical) it uses to represent SPARQL queries and transform them into SQL queries using the IQ language. We illustrated this with sample data in the GitHub context and discussed a query optimization approach implemented within Ontop. We outlined the dataset (GHTorrent) by exploring the motivations for its existence, the data collection process, and its limitations. We reviewed the ontology (SemantGit) used to investigate the OBDA approach. We outlined a brief history of why the ontology was created and the development and design decisions. Finally, we discussed the ontology's limitations and documented the extensions that were made to enrich the ontology class and property definitions.



Chapter 5

Implementation

5.1 Introduction

Implementing an Ontology-based Data Access (OBDA) system for a relational database requires a set of components. These include a domain ontology, a relational database instance, and a mapping specification linking these two. We used the Ontop¹ OBDA tool to facilitate the mapping between the SemanGit ontology and the GHTorrent MySQL database instance (dated June 01, 2019). The SemanGit ontology is a result of establishing a collection of linked data extracted from GitHub, using GHTorrent [66]. GHTorrent is an offline mirror of GitHub's event streams and persistent data for public repositories. In this chapter, we document the creation of the mapping assertions using Ontop and the query experiments. We highlight the data setup procedure, linking it to the ontology and the issues we encountered. Finally, we outline the queries used in the experiments and analyze them to explore the benefits of OBDA.

5.2 Preliminaries

We used a host computer with an AMD Ryzen 9 5900X 12-Core Processor running at 3.70 MHz using 16GB of RAM, running Windows 10 Pro version 21H2 for the query experiments. The GHTorrent MySQL database instance was used and installed on a Gigabyte GP-AG42TB AORUS 2TB M.2 2280 PCI-E 4.0 Solid State Drive. To create and manage the mapping assertions for the ontology and database, we used the open source Protégé ontology editor². Protégé enables the management of mappings and

¹<https://ontop-vkg.org/>

²<https://protege.stanford.edu/>

querying from within the Protégé editor using the Ontop plugin³. We used version 5.5.0 of Protégé and version 4.1.1 of Ontop.

5.3 Database setup

The dataset was obtained from the GHTorrent downloads page⁴. The MySQL data dump from June 2019 was downloaded and used as the dataset for the implementation. During the data setup procedure, the type of storage component used impeded the performance. Once we extracted and imported the data into a local MySQL instance on the host machine, general read performance in the MySQL workbench was very slow due to the size of the tables, even with the necessary indexes applied. To overcome this, we upgraded the storage and dedicated the storage for the sole purpose of the database without additional operating system programs and files.

5.4 Mapping GHTorrent to the SemanGit Ontology

A fundamental component of realizing an OBDA system is the mapping specification. The mapping connecting the ontology to the database involves writing individual queries that must be consistent with the vocabulary of the ontology for each database table and column [12]. We now describe a set of mapping assertions created for connecting the SemanGit ontology to the tables of interest in the GHTorrent MySQL database instance.

5.4.1 Mapping assertions

A mapping assertion comprises three components: a unique mapping identifier, a target, and a source. The target is a set of RDF triple patterns defined in the Terse RDF Triple Language (Turtle)⁵ syntax that captures the data returned by the source, with the source being a regular SQL query. Turtle is a format that enables the expression of an RDF graph using a compact and intuitive text representation by utilizing abbreviations for frequently used patterns and data types. It is a syntax for serializing RDF data into a text-based format that machines and people can write and read. Turtle employs a simple syntax based on the subject-predicate-object structure of RDF triples to provide a compact representation of RDF data that may be used for transferring and storing RDF data on the Web.

³<https://protegewiki.stanford.edu/wiki/Ontop>

⁴<https://ghtorrent.org/downloads.html>

⁵<https://www.w3.org/TeamSubmission/turtle/>

The mapping assertions construct a part of the knowledge graph (KG) as defined in the target part by populating the RDF triple pattern answer variables with the corresponding answer in the result set of the source SQL query. The answer variables are enclosed in braces “{” and “}”. We note that the order of the property mappings needs to match the order of the columns returned from the source SQL.



FIGURE 5.1: User entity mapping

Figure 5.1 shows the mapping assertions for the `User` class and `users` database table.

The mappings (figure 5.1) maps the “:User/{id}” Internationalized Resource Identifier (IRI) to the class “:User”. The “{id}” placeholder represents a unique identifier for each user. The rest of the specification defines the user’s properties and maps them to specific columns in the database `users` table.

Figure 5.1 mapping assertions

- We map the “:User/{id}” IRI to the class “:User”, where *id* is the primary key in the users table.
- “:githubUserLogin” maps to the *login* column, with data type “xsd:string”.
- “:githubUserCompany” maps to the *company* column, with data type “xsd:string”.
- “:githubUserCreatedAt” maps to the *created_at* column, with data type “xsd:dateTime”.
- “:githubUserFake” maps to the *fake* column, with data type “xsd:boolean”.
- “:githubUserDeleted” maps to the *deleted* column, with data type “xsd:boolean”.
- “:githubUserLng” maps to the *long* column, with data type “xsd:float”.

- “:githubUserLat” maps to the *lat* column, with data type “xsd:float”.
- “:githubUserCountryCode” maps to the *country_code* column, with data type “xsd:string”.
- “:githubUserState” maps to the *state* column, with data type “dbo:State”.
- “:githubUserCity” maps to the *city* column, with data type “dbo:City”.
- “:githubUserLocation” maps to the *location* column, with data type “rdfs:string”.
- “:githubUserIsOrg” maps to the *is_organization* column, with data type “xsd:boolean”.

TABLE 5.1: UserMap

Property / Class	Column	Data Type	IRI
:User	id	:User	:User/{id}
:githubUserLogin	login	xsd:string	
:githubUserCompany	company	xsd:string	
:githubUserCreatedAt	created_at	xsd:dateTime	
:githubUserFake	fake	xsd:boolean	
:githubUserDeleted	deleted	xsd:boolean	
:githubUserLng	long	xsd:float	
:githubUserLat	lat	xsd:float	
:githubUserCountryCode	country_code	xsd:string	
:githubUserState	state	dbo:State	
:githubUserCity	city	dbo:City	
:githubUserLocation	location	rdfs:string	
:githubUserIsOrg	is_organization	xsd:boolean	

In figure 5.1, we define what an organization is considered to be, where an organization, according to the dataset, is a user database entry with the “type” column populated with the value “ORG”. The mapping specification defines one property for the user: “:githubUserIsOrg”. This property is linked to the value “true” (of type “xsd:boolean”) where the column *type* has the value “ORG” in the *users* table. The purpose of this property is to indicate whether the user is an organization. This enables the KG to assert whether a user is an organization based on the boolean value of the *githubUserIsOrg* property.

Figure 5.2 mapping assertions

- We map the “:Repository/{id}” IRI to the class “:Repository”, where *id* is the primary key in the projects table.
- “:repositoryUrl” maps to the *url* column, with data type “xsd:anyURI”.
- “:githubHasOwner” maps to the *owner_id* column, which maps the :User/{owner_id} IRI to the class “:User”.

Mapping ID:

Target (Triples Template):

```

:Repository/{id} a :Repository ; :repositoryUrl {url}^xsd:anyURI ; :githubHasOwner
:User/{owner_id} ; :githubProjectName {name}^xsd:string ; :githubProjectDescription
{description}^xsd:string ; :repositoryCreatedAt {created_at}^xsd:dateTime ; :githubForkedFrom
:Repository/{forked_from} ; :githubProjectDeleted {deleted}^xsd:boolean .

```

Source (SQL Query):

```

select id, url, owner_id, name, description, created_at, forked_from, deleted from projects

```

FIGURE 5.2: Repository (Project) entity mapping

- “:githubProjectName” maps to the *name* column, with data type “xsd:string”.
- “:githubProjectDescription” maps to the *description* column, with data type “xsd:string”.
- “:repositoryCreatedAt” maps to the *created_at* column, with data type “xsd:dateTime”.
- “:githubForkedFrom” maps to the *forked_from* column, which maps the :Repository/{forked_from} IRI to the class “:Repository”.
- “:githubProjectDeleted” maps to the *deleted* column, with data type “xsd:boolean”.

TABLE 5.2: ProjectMap

Property / Class	Column	Data Type	IRI
:Repository	id	:Repository	:Repository/{id}
:repositoryUrl	url	xsd:anyURI	
:githubHasOwner	owner_id	:User	:User/{owner_id}
:githubProjectName	name	xsd:string	
:githubProjectDescription	description	xsd:string	
:repositoryCreatedAt	created_at	xsd:dateTime	
:githubForkedFrom	forked_from	:Repository	:Repository/{forked_from}
:githubProjectDeleted	deleted	xsd:boolean	

Mapping ID:

Target (Triples Template):

```

:Commit/{id} a :Commit ; :commitSha {sha}^xsd:string ; :commitAuthor :User/{author_id} ;
:committedBy :User/{committer_id} ; :commitCreatedAt {created_at}^xsd:dateTime .

```

Source (SQL Query):

```

select id, sha, author_id, committer_id, created_at from commits

```

FIGURE 5.3: Commit entity mapping

Figure 5.3 mapping assertions

- We map the “:Commit/{id}” IRI to the class “:Commit”, where *id* is the primary key in the commits table.

- “:commitSha” maps to the *sha* column, with data type “xsd:string”.
- “:commitAuthor” maps to the *author_id* column, which maps the :User/{author_id} IRI to the class “:User”.
- “:commitCommittedBy” maps to the *committer_id* column, which maps the :User/{committer_id} IRI to the class “:User”.
- “:commitCreatedAt” maps to the *created_at* column, with data type “xsd:dateTime”.

TABLE 5.3: CommitMap

Property / Class	Column	Data Type	IRI
:Commit	id	:Commit	:Commit/{id}
:commitSha	sha	xsd:string	
:commitAuthor	author_id	:User	:User/{author_id}
:commitCommittedBy	committer_id	:User	:User/{committer_id}
:commitCreatedAt	created_at	xsd:dateTime	



FIGURE 5.4: Project commits mapping

In figure 5.4, we define the mapping assertions for the associative table linking commits to a repository. An associative table is used for many-to-many relationships between two tables. In this context, the **project_commits** table represents the commits belonging to the history of a project. Multiple projects can share the same commits if one is a fork of the other⁶, where a fork is a copy of a repository.

Figure 5.4 mapping assertions We map the “:belongsToRepository” property based on the domain and range (Commit - Repository) to the *commit_id* and *project_id* columns, which maps a commit to its repository and a repository to its commits.

TABLE 5.4: ProjectCommit Map

Property	Domain	Range	Columns	IRI
:belongsToRepository	Commit	Repository	commit_id project_id	:Commit/{commit_id} :Repository/{project_id}

⁶<https://gitorrent.org/relational.html>

Mapping ID: PullRequestMap

Target (Triples Template):

```

:PullRequest/{id} a :PullRequest ; :pullRequestHeadProject :Repository/{head_repo_id} ;
:pullRequestBaseProject :Repository/{base_repo_id} ; :pullRequestHeadCommit
:Commit/{head_commit_id} ; :pullRequestBaseCommit :Commit/{base_commit_id} ; :id
{pullreq_id}^^xsd:integer ; :githubPullRequestIntraBranch {intra_branch}^^xsd:boolean .

```

Source (SQL Query):

```

select id, head_repo_id, base_repo_id, head_commit_id, base_commit_id, pullreq_id, intra_branch
from pull_requests

```

FIGURE 5.5: Pull Request entity mapping

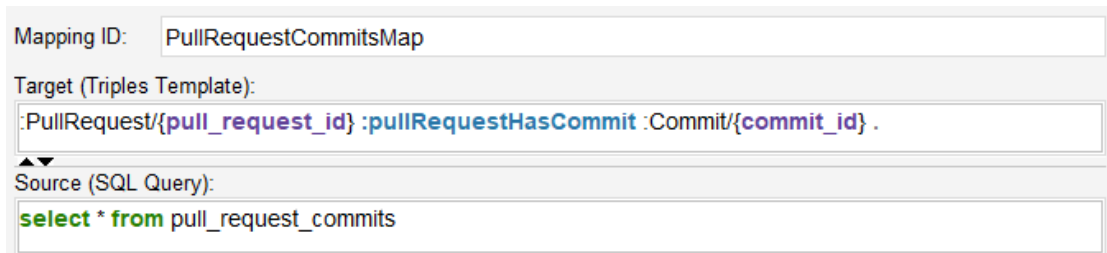
Figure 5.5 mapping assertions

- We map the “:PullRequest/{id}” IRI to the class “:PullRequest”, where *id* is the primary key in the pull_requests table.
- “:pullRequestHeadProject” maps to the *head_repo_id* column, which maps the :Repository/{head_repo_id} IRI to the class “:Repository”.
- “:pullRequestBaseProject” maps to the *base_repo_id* column, which maps the :Repository/{base_repo_id} IRI to the class “:Repository”.
- “:pullRequestHeadCommit” maps to the *head_commit_id* column, which maps the :Commit/{head_commit_id} IRI to the class “:Commit”.
- “:pullRequestBaseCommit” maps to the *base_commit_id* column, which maps the :Commit/{base_commit_id} IRI to the class “:Commit”.
- “:githubPullRequestId” maps to the *pullreq_id* column, with data type “xsd:integer”.
- “:githubPullRequestIntraBranch” maps to the *intra_branch* column, with data type “xsd:boolean”.

TABLE 5.5: PullRequestMap

Property / Class	Column	Data Type	IRI
:PullRequest	id	:PullRequest	:PullRequest/{id}
:pullRequestHeadProject	head_repo_id	:Repository	:Repository/{head_repo_id}
:pullRequestBaseProject	base_repo_id	:Repository	:Repository/{base_repo_id}
:pullRequestHeadCommit	head_commit_id	:Commit	:Commit/{head_commit_id}
:pullRequestBaseCommit	base_commit_id	:Commit	:Commit/{base_commit_id}
:githubPullRequestId	pullreq_id	xsd:integer	
:githubPullRequestIntraBranch	intra_branch	xsd:boolean	

In figure 5.6, we define the mapping assertions for the associative table linking commits to a pull request.



```

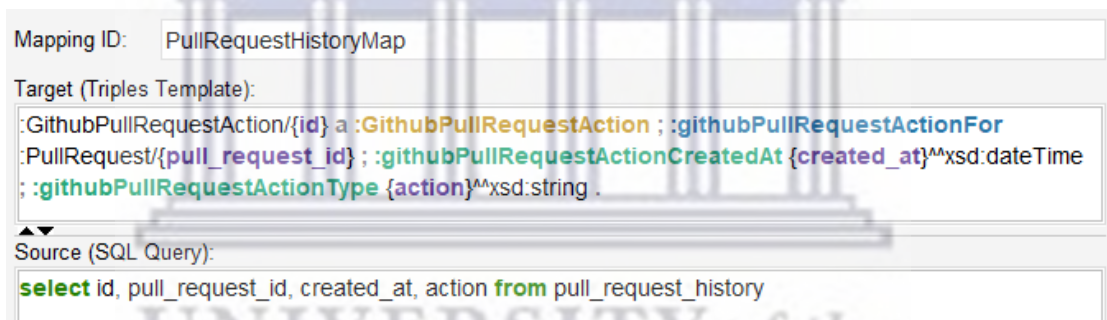
Mapping ID: PullRequestCommitsMap
Target (Triples Template):
:PullRequest/{pull_request_id} :pullRequestHasCommit :Commit/{commit_id} .
Source (SQL Query):
select * from pull_request_commits
  
```

FIGURE 5.6: Pull Request commits mapping

Figure 5.6 mapping assertions We map the “:pullRequestHasCommit” property based on the domain and range (PullRequest - Commit) to the *pull_request_id* and *commit_id* columns, which maps a commit to its associated pull request.

TABLE 5.6: PullRequestCommitsMap

Property	Domain	Range	Columns	IRI
:pullRequestHasCommit	PullRequest	Commit	pull_request_id commit_id	:PullRequest/{pull_request_id} :Commit/{commit_id}



```

Mapping ID: PullRequestHistoryMap
Target (Triples Template):
:GithubPullRequestAction/{id} a :GithubPullRequestAction ; :githubPullRequestActionFor
:PullRequest/{pull_request_id} ; :githubPullRequestActionCreatedAt {created_at}^^xsd:dateTime
; :githubPullRequestActionType {action}^^xsd:string .
Source (SQL Query):
select id, pull_request_id, created_at, action from pull_request_history
  
```

FIGURE 5.7: Pull Request event history mapping

In figure 5.7, we define the mapping assertions for the *pull_request_history* table. Since this table stores the events in the lifetime of a pull request, we map it to the “:GithubPullRequestAction” class in the ontology. Each event is the result of an action on a pull request, and thus, we associate the events with the *GithubPullRequestAction* class.

Figure 5.7 mapping assertions

- We map the “:GithubPullRequestAction/{id}” IRI to the class “:GithubPullRequestAction”, where *id* is the primary key in the *pull_request_history* table.
- “:githubPullRequestActionFor” maps to the *pull_request_id* column, which maps the *:PullRequest/{pull_request_id}* IRI to the class “:pullRequest”.
- “:githubPullRequestActionCreatedAt” maps to the *created_at* column, with data type “xsd:dateTime”.

- “:githubPullRequestActionType” maps to the *action* column, with data type “xsd:string”.

TABLE 5.7: PullRequestHistoryMap

Property / Class	Column	Data Type	IRI
:GithubPullRequest-Action	id	:GithubPullRequestAction	:GithubPullRequestAction/{id}
:githubPullRequest-ActionFor	pull_request_id	:PullRequest	:PullRequest/{pull_request_id}
:githubPullRequest-ActionCreatedAt	created_at	xsd:dateTime	
:githubPullRequest-ActionType	action	xsd:string	

Mapping ID: PullRequestMergedMap

Target (Triples Template):
 :PullRequest/{pull_request_id} :githubPullRequestMerged {is_merged}^^xsd:boolean .

Source (SQL Query):
 select pull_request_id,
 CASE action
 WHEN 'merged' THEN 'true'
 ELSE 'false'
 END as is_merged
 from pull_request_history where action = 'merged'

FIGURE 5.8: Merged Pull Requests mapping

In figure 5.8, we define what is considered a “merged” pull request, where a merged pull request according to the dataset is a *pull_request_history* database entry with the “action” column populated with the value “merged”. The mapping specification defines one property for the pull_request, which is “:githubPullRequestMerged”. This property is linked to the value of the column *is_merged* (of type “xsd:boolean”), which is based on the value of the *action* column in the **pull_request_history** table. The value “merged” is the truth value in this case. The purpose of this property is to indicate whether a pull request is merged. This enables the KG to assert whether a pull request is merged based on the boolean value of the *github_pull_request_merged* property.

TABLE 5.8: PullRequestMergedMap

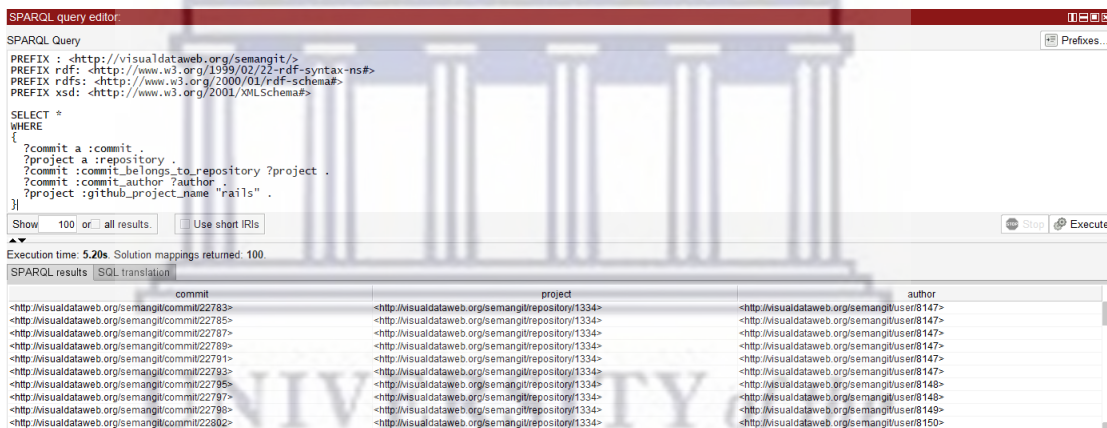
Property / Class	Column	Data Type	IRI
:githubPullRequestMerged	is_merged	xsd:boolean	:PullRequest/{pull_request_id}

Remark 5.1. We note that the subject of triples using the “a” predicate is the IRI of the described resource. In RDF, resources are identified by IRIs, which are unique strings

that identify the resource on the web. The subject of a triple using the “a” predicate is the IRI of the described resource, and the object is the IRI of the class or type the resource belongs to. The “a” predicate is a shorthand for the complete predicate “rdf:type”⁷.

5.5 Querying GHTorrent with SPARQL

To investigate the value of OBDA, we perform query answering over the VKG using a select set of queries. We investigate querying from the point of view of end-users trying to extract some insights from the data using domain terms and analyze the produced SQL queries. As shown in figure 5.9, we use the Protégé ontology editor in combination with the Ontop plugin, which provides a graphical user interface for specifying SPARQL queries and visualizing the query results.



The screenshot shows the SPARQL query editor interface. The query is as follows:

```

SPARQL Query
PREFIX : <http://visualdataweb.org/semangit/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT *
WHERE
{
  ?commit a :commit .
  ?project a :repository .
  ?commit :commit_belongs_to_repository ?project .
  ?commit :commit_author ?author .
  ?project :github_project_name "rails" .
}

```

The results table shows 100 solution mappings returned. The columns are commit, project, and author. The commit column contains URIs like <http://visualdataweb.org/semangit/commit/22783>. The project column contains URIs like <http://visualdataweb.org/semangit/repository/1334>. The author column contains URIs like <http://visualdataweb.org/semangit/user/8147>.

FIGURE 5.9: SPARQL query retrieving commits of authors from the “rails” GitHub repository.

We performed query answering over the generated virtual knowledge graph using a select set of SPARQL queries based on a user not being informed of specific data encoding schemes and the schema structure of the data source. We look at the SQL query generation and showcase a sample analysis from a subset of the queries.

```

SELECT ?commit ?author
WHERE
{
  ?commit a :Commit .
  ?project a :Repository .
  ?commit :belongsToRepository ?project .
  ?commit :commitAuthor ?author .
  ?project :githubProjectName "cpython" .
}

```

⁷<https://www.w3.org/TR/rdf11-primer/>

```
}

```

LISTING 5.1: Github commits belonging to “cpython” repository

```
SELECT v5.'author_id1m25' AS 'author_id1m25', v5.'commit_id1m5' AS
'commit_id1m5'
FROM (SELECT DISTINCT v2.'author_id' AS 'author_id1m25',
v1.'commit_id' AS 'commit_id1m5', v1.'project_id' AS 'project_id1m5'
FROM 'project_commits' v1, 'commits' v2, 'projects' v3
WHERE (
    v2.'author_id' IS NOT NULL AND v1.'commit_id' = v2.'id'
    AND v1.'project_id' = v3.'id' AND 'cpython' = v3.'name'
)
) v5

```

LISTING 5.2: SQL query example listing 5.1 is based on.

The first query, listing 5.1, is based on a SQL query example from the GHTorrent website⁸, which we implement in SPARQL. This query selects all the commits for a repository named “cpython”. The generated SQL query can be seen in listing 5.2. Here, we are asking for all the commits and commit authors belonging to a repository named “cpython”. We observe the generated SQL performing various joins on the relevant table columns, which is very concrete compared to the abstract SPARQL query. In both SPARQL and SQL queries, the input parameter is “cpython”, but knowledge of the lookup procedure is not applicable in the case of SPARQL since it deals with a higher level of abstraction.

```
SELECT *
WHERE
{
  ?organization a :User.
  ?organization :githubUserCountryCode "za".
  ?organization :githubUserIsOrg true.
}

```

LISTING 5.3: Select GitHub organizations with country code “za”

```
SELECT v1.'id' AS 'id1m52'
FROM 'users' v1
WHERE (
    'ORG' = v1.'type' AND 'za' = v1.'country_code'
)

```

LISTING 5.4: Generated SQL for listing 5.3

In listing 5.3, we select the GitHub organizations with country code “za”. In reference to chapter 4.3, GitHub identifies organizations and users as a **User** entity with

⁸<https://ghtorrent.org/relational.html>

a **type** column to distinguish whether an entity is an organization or a standard user. To model this in the ontology, the SemanGit ontology contains a data property named “*github_user_is_org*” with a domain and range of “*github_user*” and the “boolean” datatype respectively. In figure 5.1, we show how this property is mapped to the database. Listing 5.4 shows the SQL query translated from the SPARQL query in listing 5.3. Here we observe the inclusion of the generated `'ORG' = v1.'type'` **WHERE** clause, which is a result of the “UserMap” mapping specification in figure 5.1. In this query, we illustrate the case of not needing to know how an organization is defined in the data source. Here we observe, selecting the organization subset by using the “*github_user_is_org*” property in the SPARQL clause (where *github_user_is_org* is *true*), unfolds in the `'ORG' = v1.'type'` SQL clause after query translation (listing 5.4).

```
SELECT *
WHERE
{
  ?com a :Commit .
  ?pr a :PullRequest .
  ?repo a :Repository .
  ?author a :User .
  ?com :belongsToRepository ?repo .
  ?com :commitAuthor ?author .
  ?author :githubUserCountryCode "za" .
  ?pr :pullRequestHasCommit ?com
}
```

LISTING 5.5: Contributions (Pull Requests) of users with country code “za”

```
SELECT DISTINCT v2.'author_id' AS 'author_id1m25',
v1.'commit_id' AS 'commit_id1m5',
v1.'project_id' AS 'project_id1m5',
v4.'pull_request_id' AS 'pull_request_id1m14'
FROM 'project_commits' v1, 'commits' v2,
'users' v3, 'pull_request_commits' v4
WHERE (
  v1.'commit_id' = v2.'id'
  AND v2.'author_id' = v3.'id'
  AND v1.'commit_id' = v4.'commit_id'
  AND 'za' = v3.'country_code'
)
```

LISTING 5.6: Generated SQL for listing 5.5

In listing 5.5, we are asking for all the contributions made by South African users. Such information can be valuable to parties interested in the open-source contributions of software developers within a given region.

```
SELECT ?author (COUNT (?commit) AS ?commit_count)
```

```

WHERE
{
  ?commit a :Commit .
  ?commit :commitAuthor ?author .
}
GROUP BY ?author

```

LISTING 5.7: Number of commits per author

```

SELECT v1.'author_id' AS 'author_id1m25', COUNT(*) AS 'v0'
FROM 'commits' v1
WHERE v1.'author_id' IS NOT NULL
GROUP BY v1.'author_id'

```

LISTING 5.8: Generated SQL for listing 5.7

In listing 5.7, we retrieve the number of commits per author from the **commit** table. With this example, we illustrate query translation, which includes an aggregate function with the commit table and a subset of the columns. The translated MySQL query can be seen in listing 5.8.

```

SELECT DISTINCT ?member
WHERE {
  VALUES ?project { repo:27601818 }
  ?member :githubUserFake false .
  ?pr :pullRequestBaseProject ?project .
  ?pr :githubPullRequestMerged true .
  ?pr :pullRequestUser ?member .
}

```

LISTING 5.9: Select core team members of Vue js project based on Pull Request contributions

```

SELECT DISTINCT v1.'id' AS 'id1m51' FROM 'users' v1,
'pull_requests' v2, 'pull_request_history' v3,
'pull_request_history' v4
WHERE (
  (v1.'fake' = 0) AND v2.'id' = v3.'pull_request_id'
  AND v2.'id' = v4.'pull_request_id'
  AND v1.'id' = v4.'actor_id' AND 27601818 = v2.'base_repo_id'
  AND 'merged' = v3.'action'
)

```

LISTING 5.10: Generated SQL for listing 5.9

Listing 5.9 retrieves authentic users contributing to the popular GitHub repository Vue⁹ based on merges of a **Pull Request** (PR). Authentic users can own repositories and

⁹<https://github.com/vuejs/vue>

perform actions such as managing issues, pull requests, and commits. Unauthentic users only show up as commit authors or committers. The *fake* column is used to identify these types of users in the user table. A PR is a request to merge code changes made on a separate branch of the central repository into the base branch. The database table “*pull_request_history*” stores all the actions associated with a PR, including the user and type of action. We observe in the translated SQL query, listing 5.10, the lookup into this table without explicitly defining it in the SPARQL query (listing 5.9). This is a result of the mapping specification for the object property *githubPullRequestMerged* (see figure 5.8), which is populated based on the “merged” action related to a pull request that is stored in the “*pull_request_history*” table. The generated SQL query contains two self-joins on the “*pull_request_history*” table. The Ontop system uses unique constraints (primary key) for removing self-joins. In the mapping, we reference a non-unique constraint column (*pull_request_id*) for the *pull_request_history* table. As a test, we observed that the self-join was removed when using the primary key in the mapping.

We now highlight some additional queries related to repository contributions. We investigate two popular GitHub repositories, Angular and React. Angular, developed at Google, is a web application development framework that uses Typescript/JavaScript and other languages to create mobile and desktop web apps. React, a JavaScript library for building user interfaces was developed at Meta (formerly known as Facebook).

```
SELECT ?repo_name ?year (COUNT(?commit) AS ?commits)
WHERE
{
  ?commit :belongsToRepository ?project .
  ?project :githubProjectName ?repo_name .
  ?commit :commitCreatedAt ?date .
  FILTER (?project IN (repo:3905191, repo:12159636))
}
GROUP BY ?repo_name (year(?date) AS ?year)
```

LISTING 5.11: Number of commits per year for Angular and React repositories

```
SELECT v7.'name1m39' AS 'name1m39', v7.'v2' AS 'v2', COUNT(*) AS 'v4'
FROM (SELECT v5.'name1m39' AS 'name1m39',
EXTRACT(YEAR FROM v5.'created_at1m32') AS 'v2'
FROM (SELECT DISTINCT v1.'commit_id' AS 'commit_id1m5',
v3.'created_at' AS 'created_at1m32', v2.'name' AS 'name1m39',
v1.'project_id' AS 'project_id1m5'
FROM 'project_commits' v1, 'projects' v2, 'commits' v3
WHERE (
(v1.'project_id' = 3905191 OR v1.'project_id' = 12159636)
AND v1.'project_id' = v2.'id' AND v1.'commit_id' = v3.'id'
)
) v5
```



```
) v7
GROUP BY v7.'name1m39', v7.'v2'
```

LISTING 5.12: Generated SQL for listing 5.11

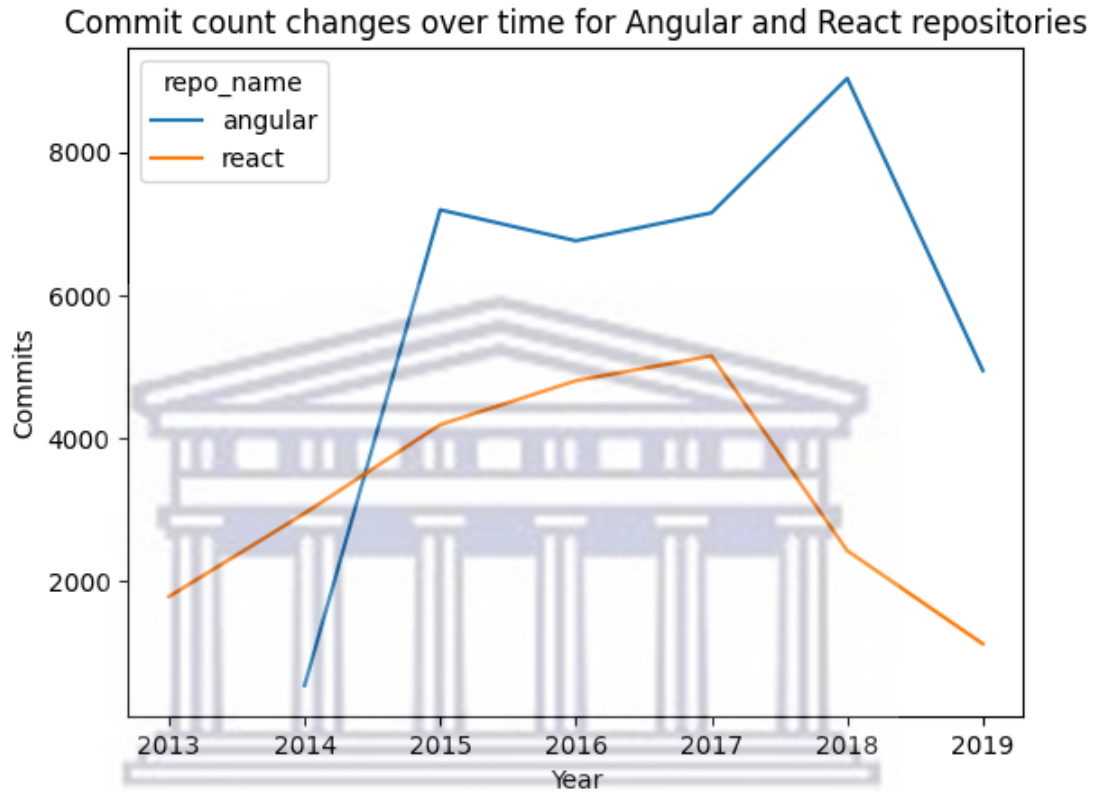


FIGURE 5.10: Angular and React repo commits by year

In listing 5.11, we select the commits for the Angular and React repositories. We group the results by repository and year to see how the number of commits changed. To help visualize this activity, we use a line plot on the results obtained. This can be seen in figure 5.10.

```
SELECT *
WHERE {
  {
    SELECT ?project ?committer (COUNT(?commit) as ?commits)
    WHERE {
      ?commit :belongsToRepository ?project .
      ?commit :commitAuthor ?committer .
      ?committer :githubUserFake false .
      FILTER (?project IN (repo:3905191))
    }
    GROUP BY ?project ?committer
    ORDER BY DESC(?commits)
    LIMIT 10
```

```

}
UNION
{
  SELECT ?project ?committer (COUNT(?commit) as ?commits)
  WHERE {
    ?commit :belongsToRepository ?project .
    ?commit :commitAuthor ?committer .
    ?committer :githubUserFake false .
    FILTER (?project IN (repo:12159636))
  }
  GROUP BY ?project ?committer
  ORDER BY DESC(?commits)
  LIMIT 10
}
}

```

LISTING 5.13: Select top 10 commit contributors for Angular and React repositories

```

SELECT v17.'author_id1m7' AS 'author_id1m7', v17.'v6' AS 'v6', v17.'v8'
AS 'v8'
FROM (SELECT v7.'author_id1m7' AS 'author_id1m7',
'http://visualdataweb.org/semangit/repository/3905191' AS 'v6', v7.'v8'
AS 'v8'
FROM (SELECT v5.'author_id1m7' AS 'author_id1m7', COUNT(*) AS 'v8'
FROM (SELECT DISTINCT v2.'author_id' AS 'author_id1m7',
v1.'commit_id' AS 'commit_id1m5'
FROM 'project_commits' v1, 'commits' v2, 'users' v3
WHERE (
(v3.'fake' = 0)
AND v1.'commit_id' = v2.'id'
AND v2.'author_id' = v3.'id'
AND 3905191 = v1.'project_id'
)
) v5
GROUP BY v5.'author_id1m7'
ORDER BY COUNT(*) DESC
LIMIT 10) v7
UNION ALL
SELECT v15.'author_id1m7' AS 'author_id1m7',
'http://visualdataweb.org/semangit/repository/12159636' AS 'v6', v15.'v8'
AS 'v8'
FROM (SELECT v13.'author_id1m7' AS 'author_id1m7', COUNT(*) AS 'v8'
FROM (SELECT DISTINCT v10.'author_id' AS 'author_id1m7',
v9.'commit_id' AS 'commit_id1m3'
FROM 'project_commits' v9, 'commits' v10, 'users' v11
WHERE (
(v11.'fake' = 0)
AND v9.'commit_id' = v10.'id'

```

```

    AND v10.'author_id' = v11.'id'
    AND 12159636 = v9.'project_id'
  ) v13
GROUP BY v13.'author_id1m7'
ORDER BY COUNT(*) DESC
LIMIT 10) v15
) v17

```

LISTING 5.14: Generated SQL for listing 5.11

Number of commits by the top 10 commit authors between the Angular and React repository

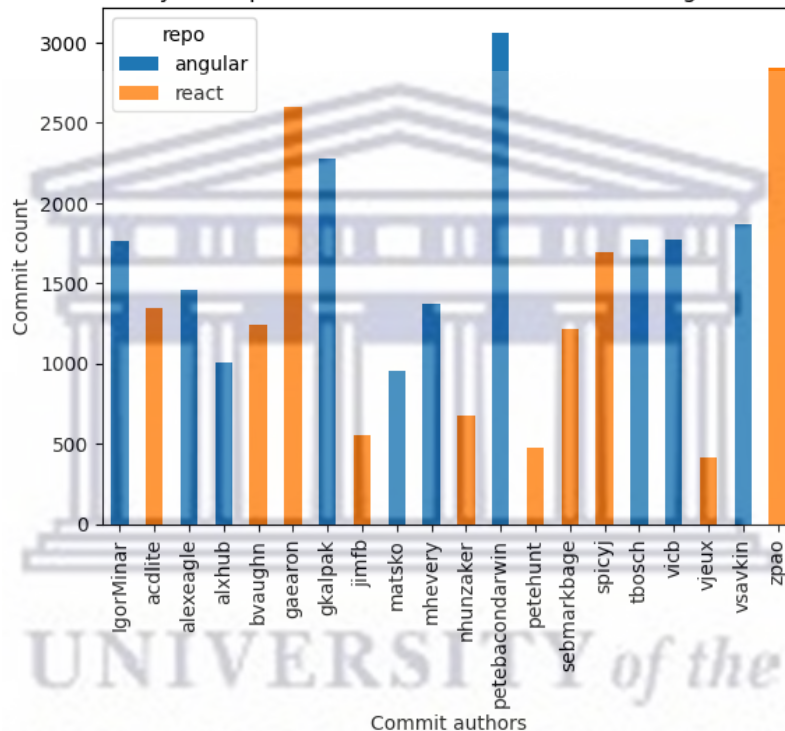


FIGURE 5.11: Angular and React top 10 contributors by commits

In listing 5.13, we select the top 10 contributors (commit authors) for the Angular and React repositories. We only select the authors (users) that are authentic, where “:githubUserFake” is false. The results can be seen in figure 5.11.

```

SELECT ?author (COUNT(DISTINCT ?commit) as ?commits) (COUNT(DISTINCT ?pr)
AS ?prs)
WHERE {
  BIND (repo:12159636 AS ?repo)
  ?commit :belongsToRepository ?repo .
  ?commit :commitAuthor ?author .
  ?pr :pullRequestBaseProject ?repo .
  ?pr :pullRequestUser ?author .
}
GROUP BY ?author

```

LISTING 5.15: Select commit count and pull request count for Angular commit authors

```

SELECT v6.'author_id1m25' AS 'author_id1m25', COUNT(DISTINCT(v6.'id1m10'))
AS 'v3',
COUNT(DISTINCT(v6.'commit_id1m5')) AS 'v4'
FROM (SELECT DISTINCT v2.'author_id' AS 'author_id1m25',
v1.'commit_id' AS 'commit_id1m5', v3.'id' AS 'id1m10'
FROM 'project_commits' v1, 'commits' v2, 'pull_requests' v3,
'pull_request_history' v4
WHERE (
v1.'commit_id' = v2.'id' AND v3.'id' = v4.'pull_request_id'
AND v2.'author_id' = v4.'actor_id' AND 12159636 = v1.'project_id'
AND 12159636 = v3.'base_repo_id')
) v6
GROUP BY v6.'author_id1m25'

```

LISTING 5.16: Generated SQL for listing 5.11

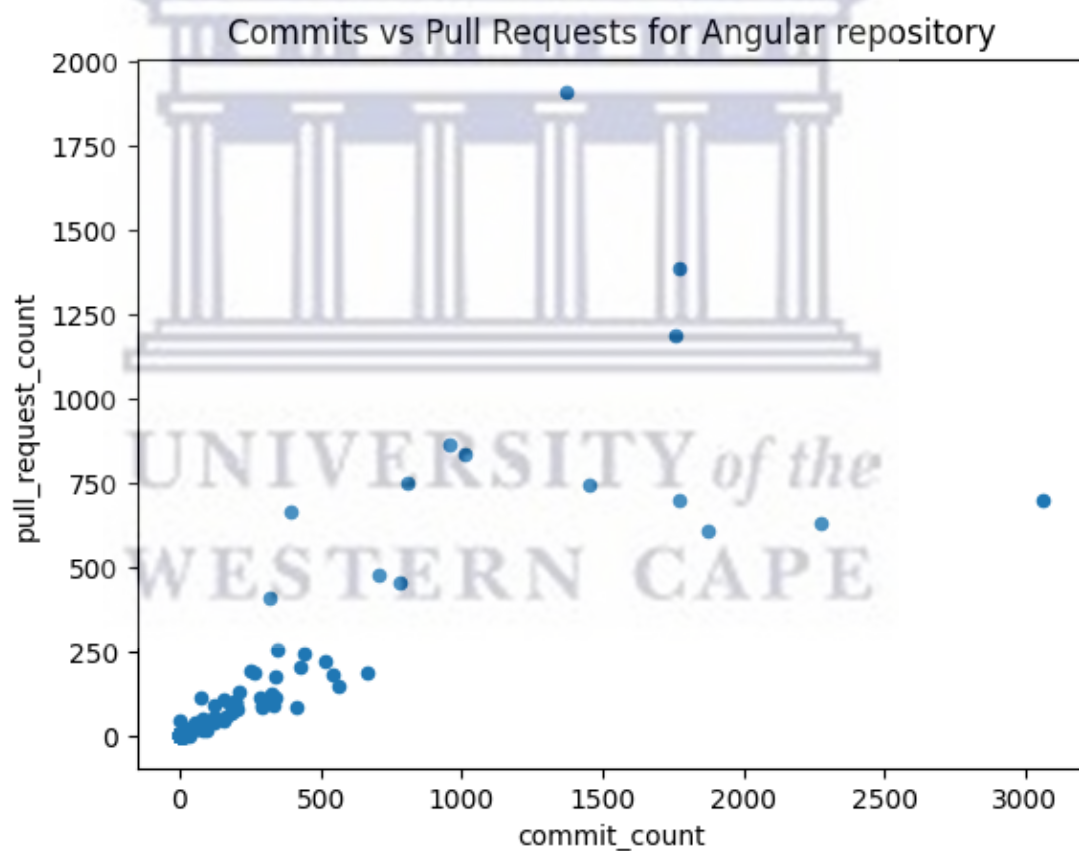


FIGURE 5.12: Angular commit author commits vs. pull requests

In listing 5.15, we select the number of commits and pull requests for the Angular repository for comparison. The results can be seen in the plot figure 5.12.

```

SELECT ?author (COUNT(DISTINCT ?commit) as ?commits)
(COUNT(DISTINCT ?pr) AS ?prs)
WHERE {
BIND (repo:12159636 AS ?repo)

```

```

?repo :repositoryHasCommit ?commit .
?author :hasAuthoredCommit ?commit .
?pr :pullRequestBaseProject ?repo .
?pr :pullRequestUser ?author .
}
GROUP BY ?author

```

LISTING 5.17: Angular repository contributor commits and pull requests

```

SELECT v6.'author_id1m7' AS 'author_id1m7',
COUNT(DISTINCT(v6.'id1m28')) AS 'v3',
COUNT(DISTINCT(v6.'commit_id1m5')) AS 'v4'
FROM (SELECT DISTINCT v2.'author_id' AS 'author_id1m7',
v1.'commit_id' AS 'commit_id1m5', v3.'id' AS 'id1m28'
FROM 'project_commits' v1, 'commits' v2,
'pull_requests' v3, 'pull_request_history' v4
WHERE (
v1.'commit_id' = v2.'id'
AND v3.'id' = v4.'pull_request_id'
AND v2.'author_id' = v4.'actor_id'
AND 12159636 = v1.'project_id'
AND 12159636 = v3.'base_repo_id'
)
) v6
GROUP BY v6.'author_id1m7'

```

LISTING 5.18: Generated SQL for listing 5.17

The query above, listing 5.17, retrieves all the contributors with their total number of commits and pull requests. We expect to receive results for this query by reasoning over the axioms in the ontology that declare inverse properties, even if we did not include any explicit mapping assertions for the object properties *repositoryHasCommit* and *hasAuthoredCommit*. We use the two axioms, that *repositoryHasCommit* is the inverse property of *belongsToRepository*, and that *hasAuthoredCommit* is the inverse property of *commitAuthor*, in this scenario.

We repeated each query ten times and took the mean average of the execution time. We compared the execution within the Protégé SPARQL query editor against running the generated SQL directly in the MySQL Command-line client. We did not notice a significant difference in the execution times. Each query was executed against the entire database by selecting “all results” in the Protégé SPARQL query editor. We show the SPARQL query execution time in table 5.9. We note that the queries with no data captured for execution time were not completed promptly, with exceptional long running times. As a result, the run time was not captured. We captured no result

TABLE 5.9: Query execution times

Query	Execution Time (s)	Number of triples
Listing 5.1	—	66951
Listing 5.3	9.515	859
Listing 5.5	—	780954
Listing 5.7	—	—
Listing 5.9	0.2202	290
Listing 5.11	0.6877	13
Listing 5.13	775.7	20
Listing 5.15	678.5	1110
Listing 5.17	680.8	1110

for execution time and the number of triples for listing 5.7 due to timing out with an out-of-memory exception.

5.6 Discussion

We performed a set of queries to highlight a subset of the features of the OBDA approach. During the execution of the experiments, the feature of querying in domain vocabulary without the need to understand the underlying database data encoding and schema as well as utilizing the ontology axioms during query executions does stand out. The results look positive and can assist various use cases related to GitHub data with a semantic approach. The ontology enables a more precise understanding of the relationships between different data elements, allowing for intelligent data querying.

We observe two potential benefits for the GitHub community. Firstly, context-aware queries enable users to express queries considering the relationships between entities, eliminating the need to comprehend the database structure. Simplifying query formulation allows users to concentrate on query semantics rather than database intricacies. The ontology facilitates expressing queries in domain vocabulary, thereby improving the ability to explore and analyze GitHub data. Secondly, OBDA enhances interoperability between various information systems interacting with GitHub data. A standardized representation of GitHub data enables the integration with information systems to ensure data consistency during exchanges.

However, the practical impediments we observe are in the manual development of a domain ontology and the creation/maintenance of a mapping specification which affects scalability. Furthermore, the queries that required a lookup over a large subset of the data did not yield results promptly and, in some cases, timed out (listings 5.1, 5.5 and 5.7). We discuss the challenges in chapter 6.

5.7 Conclusion

This chapter reported on implementing OBDA using Ontop, the SemanGit ontology, and the GHTorrent MySQL database instance. We documented the data setup procedure and mapping assertions and outlined a set of queries used in the experiments. Finally, we discussed our observations and the impediments we encountered during the experiments. We concluded that OBDA provides benefits in practice. However, it is still an emerging technology and needs to mature more. Given that the development and maintenance of an ontology and mapping specification require deep knowledge of the domain and application of interest, it is an expensive endeavor, especially in a heterogeneous production environment that is highly scalable. Thus, the research remains active in trying to solve these practical problems.



Chapter 6

Conclusion

This research aimed to investigate the question, “How effective is ontology-based data access (OBDA) with real-world data?” The context for this inquiry was the following:

We studied the background material on OBDA from a theoretical perspective. We examined the semantic web, the broader subject this technique fits. We discussed the concept of an ontology and the language used to define it. Finally, we covered the methods needed to make this feasible in a real-world situation, the challenges, and how ontologies are used to give data access via a mapping specification.

We performed a literature review based on a large set of research studies that implement data access and data integration in semantic data access using the OBDA approach. Twenty relevant studies in the domain of OBDA employing knowledge graphs to access heterogeneous data were examined by reviewing the use cases, data sources, ontology(s) and mappings employed, optimization, assessment, and ultimately the outcomes.

In addition, we reviewed existing tools that implement this approach and produced an implementation of OBDA using an ontology from the “Git” domain, a GitHub dataset (GHTorrent), and the Ontop OBDA tool. We reviewed the Ontop tool and the specific mechanics (both theoretical and practical) it employs to encode SPARQL queries and translate them into SQL queries. We demonstrated this with a GitHub dataset and presented a query optimization strategy developed within Ontop. We described the dataset by investigating its purpose, data-gathering procedure, and constraints. We reviewed the ontology that was utilized to study the OBDA technique. We provided a brief history of why the ontology was formed, its development, design decisions, constraints, and the extensions made to the ontology. The data setup procedure, mapping assertions, and a set of queries were described in the implementation. Finally, we examined the findings from the experiments as well as the challenges we faced.

The main research question was divided into four sub-questions. We provide the key findings obtained for each question as follows:

6.0.1 Research sub-question 1

“How does OBDA scale in terms of performance and implementation on real-world datasets?”

In assessing the scalability of OBDA within this research, we performed a performance assessment of our implementation. This evaluation generated quantitative metrics offering insights into OBDA efficiency. Notable performance indicators included query execution time, which depended on the volume of triples returned (refer to Table 5.9). Queries were systematically diversified to simulate accessing various data subsets, revealing performance degradation for queries handling larger subsets. The observed variation in query response times underscores scalability’s impact on the system’s overall performance. Additionally, we analyzed memory usage to discern the system’s behavior under substantial query workloads.

We note that the query volume, the ontology’s size and complexity, and the underlying system’s stability and performance all impact scalability. While it is possible to scale OBDA systems in a production environment, in contrast to traditional database systems, it is a complex endeavor that requires deep knowledge to develop and maintain domain ontologies and mapping specifications that do not suffer semantic loss. On the other hand, traditional relational database systems have lower complexity, are scalable, and have defined best practices to achieve good performance in production, given the level of maturity. Compared to existing large systems, OBDA currently falls short in complexity, cost-effectiveness, and maturity. However, OBDA allows for a more detailed understanding of the connections between diverse data sets. This allows for more intelligent and accurate data queries. Thus, the trade-off between scalability and the reasoning capacity of OBDA needs to be considered.

6.0.2 Research sub-question 2

“How do OBDA implementations compare in terms of successful results?”

The results reported in the literature from the review in chapter 3 indicate satisfactory performance and motivate the potential of OBDA and OBDI. Generally, query performance grows linearly with respect to the size of the data. For complex queries, query execution is slower but completed in a reasonable time, given the context of practical use cases. We note that the environment, deployment configuration, and optimizations

all impact the results, and we did not do a comparative analysis given the difficulty of this task.

The results from our implementation in chapter 5 look positive and can assist various use cases related to GitHub data with a semantic approach and integrate data from platforms integrating with GitHub directly or indirectly. The ontology enables a more precise understanding of the relationships between different data elements, allowing for intelligent data querying. However, as the scale of the queries grows, we observe long query run times that do not return results promptly. In some instances, we are running out of memory.

6.0.3 Research sub-question 3

“What are the current limitations of OBDA?”

We observe practical impediments in the manual development of a domain ontology and the creation of a mapping specification, which affects scalability. Even though ontologies can be adapted to changing requirements and represent different levels of abstraction, it still requires significant expertise to change the ontology and the mapping assertions while maintaining scalability. Ontology maintenance is a well-known research topic [28, 75] and involves managing changes to ensure consistency and relevance over time. Maintenance can be triggered by changes in the domain or adapting to new use cases and requirements and is predominantly manually performed. For more information, we refer the reader to [28, 75, 83]. Furthermore, it is challenging to keep ontologies and mappings up to date with changes in data sources while maintaining semantic equivalence between the original data and associated ontologies in a specific domain [76]. Also, we note that actualizing OBDA within the context of an information system requires careful consideration for the implementation of a suitable user interface (UI) to facilitate the SPARQL query construction from ontology vocabulary, where users of such a system are querying from a client-facing UI and not writing SPARQL queries.

6.0.4 Research sub-question 4

“What improvements can be made?”

Given these challenges highlighted in this study, the research in this field is very active:

- (Semi-)Automating ontology development using an approach called Ontology Learning (OL), where machine learning techniques are applied to represent knowledge

from heterogeneous data sources. Recent work in this area includes various proposals to apply OL in the scope of relational databases [6, 67, 70, 77], as well as a survey of the recent methods and tools of the OL from relational databases [76].

- Additional approaches for automating mapping specifications between ontologies and data sources using algorithmic techniques. In the work by Calvanese et al. [16], the authors proposed an algorithm to automatically detect and map a relational schema to ontology mapping patterns.
- Using distributed systems for data management [29].
- Applying caching techniques to store knowledge graphs in memory [81].
- SPARQL query scalability optimizations for large RDF data sets [50, 100, 108, 122].

A different approach would be to look at the field of Natural Language Processing to assist in query formulation, such as querying knowledge graphs in natural language, which integrates techniques from machine learning algorithms, specifically Large Language Models (LLMs) and knowledge graphs. This is, however, out of the scope of this research, and we refer the interested reader to [20, 49, 69].

We also mention Ontopic Studio¹, a more recent no-code mapping editor to link databases and data lakes with knowledge graphs. This tool enables the creation and editing of knowledge graphs from relational databases utilizing a UI. We direct the reader to <https://ontopic.ai/en/ontopic-studio/> for further detail.

Based on the answers to the research sub-questions, we construct an answer to the thesis's primary research question, "How effective is OBDA with real-world data?". We conclude this thesis as follows: "The OBDA studies considered in this thesis have limited effectiveness in solving the overall challenges of data access and integration at scale and require bespoke solutions in various domains and environments. They are effective at specific use cases but lack maturity from an implementation and maintenance point of view. Finally, as databases are not static but change over time, the process of mapping between ontologies and databases and the process of querying must take these temporal features into consideration".

6.1 Future work

There are opportunities for this work to be extended and applied for specific use cases applicable to GitHub and OBDA. This includes publishing the extended ontology and

¹<https://ontopic.ai/en/>

making this work publicly available to the GitHub community via an interface and API endpoint for further evaluation. Maintenance of the extended ontology will be ongoing and can take several directions depending on the scope of use cases. Another consideration is to harness the capabilities of LLMs to facilitate natural language understanding, enabling querying and parsing in a natural language context. Moreover, this research can contribute to the broader domain of artificial intelligence by aiding in knowledge extraction from heterogeneous data.

6.2 Concluding Comments

This research illustrates the application of ontologies and knowledge graphs to solve large-scale data integration and querying. Given that OBDA is still an emerging technique, the research demonstrates the importance of interacting with data using domain vocabulary. However, the practical challenges provide several directions for future work. Investigating the theoretical challenges associated with OBDA, such as scalability, expressiveness, and reasoning complexity, emerges as another crucial direction for advancing the field.

The artifacts of this research can be found at <https://github.com/yahlieel/SemanGit>, which is a fork of the original SemanGit repository² and includes the extended ontology and mapping specification implemented.

²<https://github.com/SemanGit/SemanGit>

Appendix A

Literature review summary

This appendix contains tables that summarize the findings of the literature review.



No.	Domain	Use case	Objective	Ref
1	Manufacturing, Machine Diagnoses	Analyses of product quality during manufacturing.	Addressing the challenges of access to data generated during product manufacturing.	[53]
2	Manufacturing, Machine Diagnoses	Enabling direct data access for engineers in a Big Data environment.	Enable direct data access using a hybrid approach, including classical OBDA, which supports archived data, static relational data, and live streaming data.	[61]
3	Oil and Gas	Statoil data access is performed by geologists who often pass the requirements for the data to technical experts. OBDA is applied to address the bottleneck this creates at scale.	Develop a solution to address the data access problem at scale while considering the limitations of OBDA at the time of the publication.	[59]
4	Biomedical	A study of semantic proteomics data integration linking four data sources.	The aim is to help biologists get relevant knowledge from multiple data sources to understand and explain the biological processes of interest.	[86]
5	Big Data	Implementation of OBDA in the context of the Semantic Data Lake.	To address the challenges of Query translation, federated query execution, and data silos, the Squerall framework is implemented for querying data lakes.	[79]

No.	Domain	Use case	Objective	Ref
6	Healthcare	Computational epidemiology seeks to develop computational methods to study the distribution and determinants of health-related states or events (including disease) and the application of this study to the control of diseases and other health problems.	Develop a knowledge base that facilitates the development of decision support and analytical environments to support epidemic science.	[43]
7	Healthcare	Access to multiple rare disease datasets is important as it will lead to new research opportunities and analysis over larger cohorts.	The application of semantic web technologies and federated queries provides a novel infrastructure that can readily incorporate additional registries, thus providing access to harmonized data relating to unprecedented numbers of patients with rare diseases while meeting data privacy and security concerns.	[85]
8	Healthcare	Adding ontology reasoning capabilities to medical information database access.	Allow for the storage of knowledge about the medical field to make it possible to intuitively retrieve data from a complex relational database.	[34]
9	Healthcare	Automatic ontology-based data integration method that can be effectively deployed and used in healthcare.	The proposed system helps the doctor query the patient records stored across various data sources without knowing the query required to access them.	[110]

No.	Domain	Use case	Objective	Ref
10	Healthcare	Extant cancer survival analyses have primarily focused on individual-level factors due to limited data availability from a single source.	The authors proposed an ontology-based approach to integrate heterogeneous datasets addressing key data integration challenges and simultaneously study as many cancer risk factors as possible.	[120]
11	Maritime	Distributed knowledge bases make data retrieval, integration, and reasoning with these data challenging.	Support the use of distributed knowledge bases for retrieving, integrating, and reasoning with data from disparate and heterogeneous. data sources.	[97]
12	Biology	An ontology-based federated approach for data integration in the Biology domain.	Enable researchers to jointly query three heterogeneous databases using a common query language.	[104]
13	Biomedical	Access to patient data has become a major bottleneck for healthcare professionals who struggle to find the relevant information in a timely way and without missing critical clinical information.	A novel hybrid semantic and text-based system that Ahus commissioned to provide integrated access to patient health records scattered in several databases and document repositories.	[112]
14	Biomedical	Public biomedical data distributed in large databases worldwide are far from being “standardized” to exploit the latest machine learning technologies to analyze data. This is the case of neurodegenerative diseases and the Alzheimer’s Disease (AD) in whose context specialized data collections such as the one by the Alzheimer’s Disease Neuroimaging Initiative (ADNI) is maintained.	The objective of this work is to build a computational ontology from the ADNI data collection and to provide a means for populating the ontology with the actual data in the ADNI. These two components make it possible to query the ADNI database semantically to support data extraction more intuitively.	[109]

No.	Domain	Use case	Objective	Ref
15	Manufacturing, Machine Diagnoses	Streaming analytics that requires integration and aggregation of heterogeneous and distributed streaming and static data is a typical task in many industrial scenarios, including the case of industrial IoT where several pieces of industrial equipment such as turbines in Siemens are integrated into an IoT.	Extend OBDA to become analytics, source, and cost-aware.	[60]
16	Manufacturing, Machine Diagnoses	Semantic technologies can help address the challenges with authoring, reusing, and maintaining signal processing rules.	The authors propose to extend the traditional data-driven approach to diagnostics with an OBDA layer and a new rule language to what they call Semantic Rule-based Diagnostics.	[98]
17	Manufacturing, Machine Diagnoses	The digitization of the industry requires information models describing assets and information sources of companies to enable the semantic integration and interoperable exchange of data.	The objective is to produce an information model centered around machine data and describe all relevant assets, key terms, and relations in a structured way, using existing and newly developed RDF vocabularies.	[91]
18	Healthcare	Semantic interoperability is essential when carrying out post-genomic clinical trials where several institutions collaborate since researchers and developers need an integrated view and access to heterogeneous data sources.	The objective is to use RDB2RDF systems that provide RDF datasets as a unified view.	[93]

No.	Domain	Use case	Objective	Ref
19	Services	Process mining aims at discovering, monitoring, and improving business processes by extracting knowledge from event logs.	Utilize a framework and methodology to extract XES event logs from relational data sources.	[17]
20	Manufacturing, Machine Diagnoses	Rule-based diagnostic systems to minimize the maintenance cost and downtime of equipment poses significant challenges in rule authoring, reuse, and maintenance by engineers.	The authors propose an approach to address the problems of Rule-based diagnostic systems by relying on the OBDA approach.	[62]

TABLE A.1: Chapter 3 literature review summary.



UNIVERSITY of the
WESTERN CAPE

No.	Data sources	Optimizations	Ref
1	PostgreSQL database	Manual construction of database constraints applied in Ontop to support non-primary/foreign key constraints	[53]
2	Streaming data, Static data	Parallelism were applied to live-stream operations by inter-query parallelism, executing queries on distributed compute nodes.	[61]
3	Seven databases, Exploration and Production Data Store (EPDS)	Query rewriting optimization, Query unfolding optimization. Structural optimizations - Formulate query joins inside the unions and special functions (such as URI construction) as high as possible in the query tree. Detect and remove inefficient joins between sub-queries. Semantic optimizations - Remove redundant unions and joins, detect unsatisfiable or trivially satisfiable conditions, etc., using database constraints.	[59]
4	UniProt Knowledgebase, String (Search Tool for the Retrieval of Interacting Genes/Proteins), Protein Data Bank, Pubmed	The IPDS is stored using HDFS. A caching strategy is applied using SPARK..	[86]
5	Apache Cassandra, Mongo, Apache Parquet, CSV, MySQL	The Squerall framework make use of the underlying implementations of SPARK and Presto.	[79]
6	Synthetic Population - Household, person, activity (Relational) Contact network and output (File) Experimental (Relational)	The study uses a combination of tuple-based and value-based mapping using D2RQ.	[43]

No.	Data sources	Optimizations	Ref
7	CSV tabular data on the antineutrophil cytoplasmic antibody (ANCA) - Associated Vasculitides (AAV) disease.	The Apache Jena Fuseki SPARQL server was used to store the generated knowledge graph and enable federated querying.	[85]
8	Relational Database	N/A	[34]
9	Excel, SQL Server, MongoDB	System implementation adhere to storage optimization principles.	[110]
10	Patients' demographic, tumor, treatment, and survival information from the 1996–2010 data of Florida Cancer Data System (FCDS) (Relational), Census tract-level poverty information from the 2000 U.S. census data (Relational), 1996-2010 county-level smoking rates from the Behavioral Risk Factor Surveillance System (BRFSS) of the Centers for Disease Control and Prevention (Relational)	N/A - Used the Ontop system	[120]
11	Two PostgreSQL databases	Parallelization	[97]
12	Three data sources: UniProt (RDF), BGee (MySQL) and OMA (HDF5)	N/A	[104]
13	DIPS, Metavision and DIPS Archive	N/A - Used the PreOptique (based on Optique) system.	[112]
14	CSV tabular data	N/A	[109]

No.	Data sources	Optimizations	Ref
15	Static and Streaming Data	<p>Query optimizations on live streams: In-memory indexing</p> <p>Query optimizations on archived information: Efficient storage of archived streams, Elastic infrastructure that automatically distributes analytical computations and data over a computational cloud.</p>	[60]
16	PostgreSql database	Developed a new Semantic Rule-based Diagnostic language to serve the diagnostic tasks required.	[98]
17	Sensor Data, Manufacturing Execution System data	Data sources are replicated and synchronized periodically.	[91]
18	Relational database	<p>Optimization techniques to the query translation algorithm include: Self-join elimination, Phantom triple pattern introduction</p>	[93]
19	Relational database	N/A - Used the Ontop system	[17]
20	TeraData, MS SQL Server, SAP HANA, IBM Maximo	Translation of semantic diagnostic programs into SQL queries and then execution of generated queries.	[62]

TABLE A.2: Chapter 3 data sources and optimizations summary.

No.	Evaluation Details	Results	Ref
1	Dataset 1 - 3.15GB Dataset 2 - 31GB, Dataset 3 - 59GB Query catalog of 13 queries. Queries range from performing joins and applying filters, to nested sub-queries and complex aggregation. Used Ontop and Scalable Semantic Analytics Stack (SANSA).	Ontop outperformed SANSA and supported more queries. Most Queries (q1–q5, q6, q7, q12, and q13) execution times scale sub-linearly, with most running in less than one second even over the largest dataset DS3. The evaluation showed that complex queries can be answered with the OBDA approach within a reasonable timeframe.	[53]

No.	Evaluation Details	Results	Ref
2	<p>Streaming and static data was used containing measurements produced by 100,000 thermocouple sensors installed.</p> <p>Two STARQL queries were adopted for evaluation:</p> <p>Query 1: Calculates the Pearson correlation between two live streams.</p> <p>Query 2: Computes the Pearson correlation of a live stream with a varying number of archived streams.</p> <p>Applied Parallelism between live streams, and parallelism between live and archived streams.</p>	<p>Query 1: Executed with varying numbers of concurrent queries (1 to 1024) between different pairs of live streams, using a fixed window size of 60 tuples on non-overlapping windows and 128 ExaStream worker nodes.</p> <p>The system's throughput increased linearly with query numbers, peaking at 4,250,226 tuples/s when matching the available cores (256), but more queries caused core sharing and reduced throughput.</p> <p>Query 2: Executed with varying numbers of available VM-workers (1 to 16), using a fixed live-stream velocity of 1 tuple/min and a fixed window size of 1 hour (60 tuples), comparing the current live stream window against 100,000 archived ones.</p> <p>Each node calculated the Pearson coefficient between its subset of archived measurements and the live stream. Intra-query parallelism notably decreased processing time due to a surplus of archived windows compared to available workers.</p>	[61]

No.	Evaluation Details	Results	Ref
3	<p>Query catalog of 73 queries. 73%, are either linear or three-shaped conjunctive queries, the others contain aggregate functions and negation.</p> <p>The system was deployed across seven large and intricate data sources: EPDS, Recall, CoreDB, GeoChemDB, OpenWorks, Compass, and NPD FactPages.</p> <p>The study aimed to measure performance gains from optimizations, particularly focusing on eliminating duplicates and employing OBDA Constraints. Queries were executed with different DISTINCT strategies: no DISTINCT, DISTINCT by the database engine (dbDist), and DISTINCT by the OBDA engine (obdaDist).</p>	<p>In the noDist experiment, 17 out of 60 queries timed out. Of the successful 43, execution times varied from less than 1s to 2m to 6m, with an average of 36.5s and a median of 12.5s. The maximum unfolding time was 187ms.</p>	[59]

UNIVERSITY of the
WESTERN CAPE

No.	Evaluation Details	Results	Ref
4	<p>The study evaluates the performance of IPDS query processing, focusing on query rewriting and execution. It examines the total response time of test queries across four proteomics data sources. Seven queries involve searching one protein with multiple fields, while five queries involve searching multiple proteins simultaneously with fixed fields.</p>	<p>The study analyzes query rewriting and execution performance in IPDS, and the impact of caching.</p> <p>Query rewriting times increase with more fields queried, ranging from 1.035 to 6.417 seconds for 7 queries. For multiple queries, rewriting time rises from 9.918 to 196.468 seconds for 5 to 100 queries.</p> <p>Execution times for single searches vary from 29.115 to 154.47 seconds for 2 to 14 fields. For multiple searches, it ranges from 1995.835 to 3673.239 seconds for 5 to 100 calls, showing a quadratic increase.</p> <p>Without caching, 100 query calls take 75223.095 seconds, while with caching, it takes 3673.239 seconds.</p>	[86]

UNIVERSITY of the
WESTERN CAPE

No.	Evaluation Details	Results	Ref
5	<p>Compares the performance of two query engines, Spark and Presto. Evaluating accuracy as well as query performance.</p> <p>Datasets: Synthetic data generated at three scales (500K, 1.5M and 5M) based on the Berlin SPARQL Benchmark (BSBM) [9].</p> <p>Five SQL table dumps used from BSBM: Product, Producer, Offer, Review, and Person.</p> <p>Data is pre-processed to extract tuples and stored at three different scales in Cassandra, MongoDB, Parquet, CSV and MySQL.</p> <p>Ten queries are used for evaluation ¹.</p>	<p>Accuracy: Squerall results was 100% identical to MySQL. MySQL timed out at the 1.5m data scale. Returned results (not timing out) for Spark and Presto were identical.</p> <p>Performance: In data scale 0.5M, query performance is superior across all the queries, with an increase of up to 800%. In data scale 1.5M and 5M, Presto-based is superior in all queries besides Q1, with an increase of up to 1300%.</p> <p>Presto-based Squerall performed significantly better than Spark-based. Presto emphasizes ad hoc querying as a fundamental feature, while Spark only partially addresses this aspect [79].</p>	[79]

¹<https://github.com/EIS-Bonn/Squerall>

No.	Evaluation Details	Results	Ref
6	<p>Evaluated query execution to measure the strength of mapping approaches over various types of RDF graphs. Mapping approaches (value-based and tuple-based): D2RQ with the Oracle database, D2RQ with Postgres database, Jena TDB, and Virtuoso tools.</p> <p>20 Queries: 10 queries collected by interviewing various epidemiologists.</p> <p>Created a set of five benchmark queries based on BSBM version 3.1 and five D2RQ benchmark queries.</p>	<p>For both virtual and materialized RDF graphs, value-based mapping outperforms tuple-based mapping, except for queries returning large numbers of triples.</p> <p>Virtuoso with tuple-based mapping shows better performance in scenarios where queries yield large triple counts.</p> <p>SPARQL queries containing regular expressions perform faster with tuple-based mapping and the Oracle tool for virtual RDF graphs.</p> <p>For materialized graphs, value-based mapping is faster for regular expression queries and provides similar performance for both Jena TDB and Virtuoso tools.</p> <p>Value-based mapping facilitates faster execution of queries involving multiple data sources due to preserved relationships. Queries reliant on a single data source show consistent execution times regardless of mapping.</p> <p>Tuple-based mapping's neglect of primary and foreign key relationships leads to data duplication and slower performance in complex queries.</p> <p>Preservation of relationships in value-based mapping enhances performance in complex queries.</p>	[43]

No.	Evaluation Details	Results	Ref
7	<p>The study applies OBDA in a federated setting to provide a novel infrastructure that can integrate various data registries, while maintaining data privacy and security requirements.</p> <p>No performance evaluation was performed.</p>	N/A	[85]
8	<p>The study documents the development and implementation of an OBDA for medical data access system.</p> <p>No performance evaluation was performed.</p>	N/A	[34]
9	<p>The study reports on the time it takes to generate schema mappings, symptom generation and data retrieval across three data sources, Excel, SQL and MongoDB.</p>	<p>The generation of schema and mapping for about 16,000 records took about 15 seconds, whereas for the time taken for 7500 records is 6 seconds.</p> <p>The symptom generation process took about 12 s for about 16,000 records. Whereas the time taken for 7500 records is 4 seconds.</p> <p>The time taken to retrieve output from a set of 28,000 records took about 6 seconds, whereas for the time taken for 14,000 records is 4 seconds.</p> <p>There is a linear increase in query execution time along with the increasing size of the dataset.</p>	[110]

No.	Evaluation Details	Results	Ref
10	<p>The study developed an Ontology for Cancer Research Variables (OCRV) and created mapping axioms for data integration across data sources.</p> <p>Implemented a data pipeline using the Ontop platform for querying, extracting, and transforming relational database data for integrative analysis.</p> <p>No performance evaluation was performed.</p>	<p>The following key integration challenges are addressed with the solution:</p> <ul style="list-style-type: none"> Using a shared, controlled vocabulary to make data understandable to both human and computers. Explicitly modeling the semantic relationships makes it possible to compute and reason with the data. Linking patients to contextual and environmental factors through geographic variables. Being able to document the data manipulation and integration processes clearly in the ontologies. 	[120]

UNIVERSITY *of the*
WESTERN CAPE

No.	Evaluation Details	Results	Ref
11	<p>Two data sources: Hermes and Aminess, both PostGIS/- PostgreSQL.</p> <p>Hermes provides dynamic data about vessels' movement. Aminess data source provides both static and dynamic data.</p> <p>Static data comprises positions of 452 ports, 48 restricted regions, and details of 38,530 registered vessels.</p> <p>Dynamic data includes critical points from AIS messages and weather forecasts from May 1st, 2015, to Sept. 1st, 2015, totaling 2,745,776 records.</p> <p>The system is configured to retrieve dynamic data at various update intervals. Each stage of the system has been evaluated across different update periods, ranging from 120 to 10,800 seconds (3 hours).</p>	<p>Time required for system initialization and static data retrieval remains constant regardless of the update period.</p> <p>Retrieval time for dynamic data remains small compared to overall update time.</p> <p>Distributed computation of dynamic data triples scales well with low increase rate as update period increases.</p> <p>Ontop OBDA demonstrates scalability, as the increase in dynamic data triples with update period has a lower impact on retrieval time.</p> <p>Distribution of computations to distinct workers contributes to scalability.</p> <p>Recognition of complex events becomes the most time-consuming task for update periods exceeding 3960 seconds.</p>	[97]

No.	Evaluation Details	Results	Ref
12	<p>Data set consists of three Databases. UniProt RDF KnowledgeBase, high-quality sequence and functional information on proteins.</p> <p>OMA, a database of orthology inferences. Bgee, a database of curated gene expression patterns in animals.</p> <p>Evaluated performance with 12 federated queries that illustrate real use cases requiring information across the three databases.</p> <p>Evaluated three queries against each dataset combination.</p>	<p>Average query run-time of up to 6 seconds for 9 out of 12 queries, with less than half a second for three out of these. - Hold for queries with higher complexity (number of triple patterns).</p> <p>The longest run time is 349.18 seconds returning 2269 results (triples), as a result of having to scan the entire search space in the OMA database.</p>	[104]
13	<p>Evaluation was performed based on system usability. Participants were requested to complete a questionnaire after testing PreOptique.</p> <p>The employed questionnaire has two sections: the first one corresponds to the System Usability Scale (SUS) ².</p>	<p>The study computed the SUS scores for the participants' responses, obtaining 86.0 in average with a standard deviation of 10.7.</p> <p>The study only tested the system with a copy of the production databases for 10 patients. However, the authors conclude that it should not entail scalability issues, since query complexity does not change with the number of patients.</p> <p>While the number of participants in the usability study is relatively low, experts in the field report that only five participants are needed on average to find 85% of usability problems in a design.</p>	[112]

²<https://digital.gov/2014/08/29/system-usability-scale-improving-products-since-1986/>

No.	Evaluation Details	Results	Ref
14	The study define a computational ontology representing a logic-based formal conceptual model of the Alzheimer Disease Neuroimaging Initiative (ADNI) data collection.	Developed a detailed computational ontology for clinical multi-modal datasets from the ADNI repository. Implemented a mechanism to populate the ontology with ADNI data. Facilitates complex queries to ADNI files, enabling acquisition of new diagnostic knowledge about Alzheimer's disease.	[109]
15	Evaluated five queries against optimizations performed. Query 1: Computes an equality join on the Wid and Time attributes between two live-streams. Query 2: Computes the Pearson correlation of a live stream with a varying number of archived streams. Queries 3 and 4: Variations of Query 2 but, computing similarity based on either the average or the minimum values within a window. Query 5: Calculates the Pearson correlation between two live streams. Adaptive indexing optimisation - Query 1 Materialised Window Signatures (MWS) optimisation - Query 2 - 5	The MWS optimisation reduces the time for the Pearson query by 8.18%. The join between the live stream and the large Measurements relation, consuming 69.58% of the query time, is unavoidable. For the other two queries, the CPU overhead was reduced of the query, and the optimiser further prunes this join from the query plan as it is no longer necessary. Parallelism between live and archived streams - Query 5: Intra-query parallelism results in significant decrease of the time required to perform the join operation. Locality-sensitive hashing (LSH) optimisation - Query 5: One can observe a significant decrease in the overall query execution time when we adopt the combination of the MWS and LSH techniques for computing correlation between live and archived streams.	[60]

No.	Evaluation Details	Results	Ref
16	<p>Evaluation is based on how well the SQL translation approach scales.</p> <p>Four manufacturing diagnosis tasks (queries) T1 to T4 was executed on each dataset.</p> <p>Data consists of transaction data from 15 days of the conveyor's run, which is scaled up to 40 conveyors.</p> <p>Producing 10 datasets with 4 to 40 conveyors each.</p>	<p>1.3GB dataset: T1 - 3s, T2 - 10s, T3 - 24s, T4 - 30s</p> <p>2.6GB dataset: T1 - 6s, T2 - 21s, T3 - 53s, T4 - 64s</p> <p>3.9GB dataset: T1 - 10s, T2 - 31s, T3 - 80s, T4 - 101s</p> <p>5.2GB dataset: T1 - 14s, T2 - 49s, T3 - 103s, T4 - 128s</p> <p>6.5GB dataset: T1 - 18s, T2 - 57s, T3 - 136s, T4 - 170s</p> <p>7.8GB dataset: T1 - 23s, T2 - 71s, T3 - 171s, T4 - 225s</p> <p>9.1GB dataset: T1 - 29s, T2 - 83s, T3 - 205s, T4 - 274s</p> <p>10.4GB dataset: T1 - 35s, T2 - 97s, T3 - 234s, T4 - 331s</p> <p>11.7GB dataset: T1 - 39s, T2 - 108s, T3 - 248s, T4 - 367s</p> <p>13GB dataset: T1 - 46s, T2 - 121s, T3 - 273s, T4 - 412s</p> <p>The running time grows linearly with respect to the data size.</p> <p>The most challenging query T4 was answered in 7 min (running over 40 conveyors).</p>	[98]
17	<p>A questionnaire was designed and distributed to stakeholders involved in the information modeling project to gather anonymous feedback.</p>	<p>Stakeholder feedback varied on the information modeling project, with some optimistic about semantic technologies, while others remained skeptical. Expectations included enabling autonomous systems and reducing interfaces.</p> <p>Bottlenecks from stakeholders includes the lack of standardized ontologies and available IT personnel. Consequently, the company is seeking IT personnel with semantic technology expertise.</p>	[91]

No.	Evaluation Details	Results	Ref
18	<p>Five selected queries was used for evaluation.</p> <p>Comparing morph-RDB [94] with D2R [8], with regards to the total time required for the execution of the SPARQL queries.</p> <p>The study assessed query performance in cold and warm modes, with the former involving server restarts and cache clearing before each query. Average execution times were normalized against native queries. Notably, the evaluation method applies only to morph-RDB and native queries due to limitations with the D2R Server, which generates multiple SQL queries and conducts in-memory joins.</p>	<p>The study found that morph-RDB outperformed D2R Server for accessing relational data using SPARQL.</p> <p>Query performance is good in general, with better results achieved using morph-RDB. However, a subset of queries are still time-consuming due to arithmetic operations in the SPARQL query and its translation into SQL.</p>	[93]
19	<p>The study utilises a framework to extract event logs from relational data sources.</p> <p>No performance evaluation was performed.</p>	N/A	[17]

No.	Evaluation Details	Results	Ref
20	<p>Evaluation is based on the efficiency of the SQL code generated by the OBDA component.</p> <p>The data was collected from 29 sensors installed on trains, along with relevant train information. Subsequently, the data was scaled in both the number of sensors and time dimensions.</p> <p>The study evaluated four diagnostic tasks (queries) across each scaled dataset.</p>	<p>The running time of queries grows linearly with respect to the growth of the data.</p> <p>The evaluation indicates that diagnostic engineers can reduce their time spent by up to 66% by using ontologies. Consequently, this semantic solution enables engineers to concentrate more on analyzing diagnostic output rather than on the current tasks of understanding and collecting data for creating data-driven diagnostic rules [62].</p>	[62]

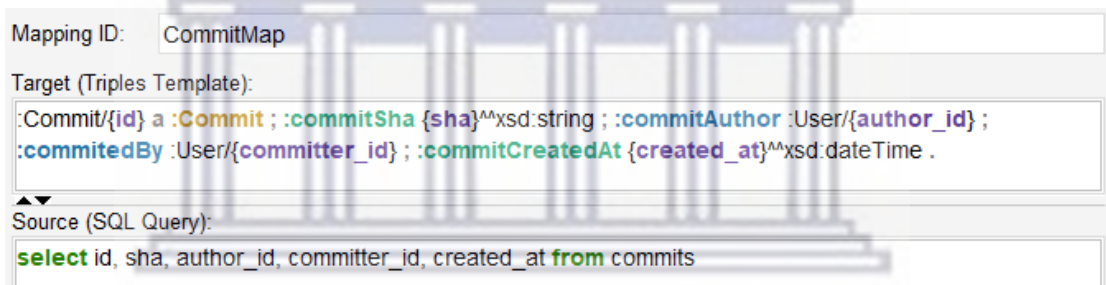
TABLE A.3: Chapter 3 literature review evaluation summary.

UNIVERSITY of the
WESTERN CAPE

Appendix B

Mapping specifications

This appendix lists the mapping specifications created for the implementation in chapter 5.



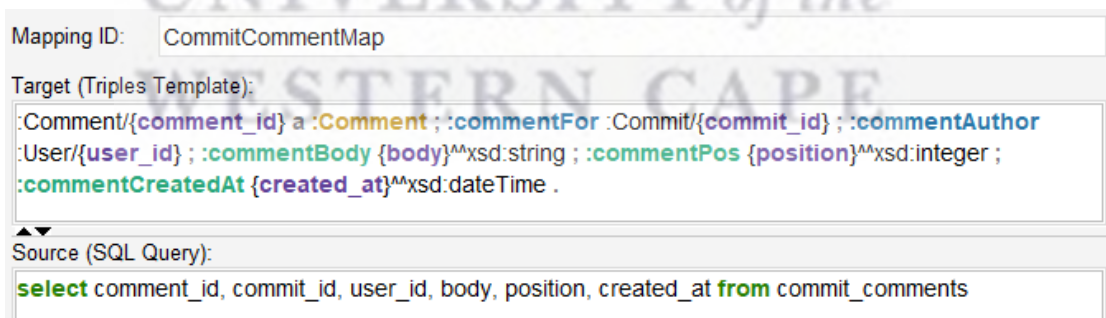
The screenshot shows a web interface for defining a mapping. It has three main sections: 'Mapping ID', 'Target (Triples Template)', and 'Source (SQL Query)'. The 'Mapping ID' field contains 'CommitMap'. The 'Target' field contains a SPARQL-like template: `:Commit/{id} a :Commit ; :commitSha {sha}^xsd:string ; :commitAuthor :User/{author_id} ; :committedBy :User/{committer_id} ; :commitCreatedAt {created_at}^xsd:dateTime .`. The 'Source' field contains an SQL query: `select id, sha, author_id, committer_id, created_at from commits`. There are expand/collapse arrows between the sections.

```
Mapping ID: CommitMap
```

```
Target (Triples Template):
:Commit/{id} a :Commit ; :commitSha {sha}^xsd:string ; :commitAuthor :User/{author_id} ;
:committedBy :User/{committer_id} ; :commitCreatedAt {created_at}^xsd:dateTime .
```

```
Source (SQL Query):
select id, sha, author_id, committer_id, created_at from commits
```

FIGURE B.1: Commit map



The screenshot shows a web interface for defining a mapping. It has three main sections: 'Mapping ID', 'Target (Triples Template)', and 'Source (SQL Query)'. The 'Mapping ID' field contains 'CommitCommentMap'. The 'Target' field contains a SPARQL-like template: `:Comment/{comment_id} a :Comment ; :commentFor :Commit/{commit_id} ; :commentAuthor :User/{user_id} ; :commentBody {body}^xsd:string ; :commentPos {position}^xsd:integer ; :commentCreatedAt {created_at}^xsd:dateTime .`. The 'Source' field contains an SQL query: `select comment_id, commit_id, user_id, body, position, created_at from commit_comments`. There are expand/collapse arrows between the sections.

```
Mapping ID: CommitCommentMap
```

```
Target (Triples Template):
:Comment/{comment_id} a :Comment ; :commentFor :Commit/{commit_id} ; :commentAuthor
:User/{user_id} ; :commentBody {body}^xsd:string ; :commentPos {position}^xsd:integer ;
:commentCreatedAt {created_at}^xsd:dateTime .
```

```
Source (SQL Query):
select comment_id, commit_id, user_id, body, position, created_at from commit_comments
```

FIGURE B.2: Commit Comment map

Mapping ID:	FollowerMap
Target (Triples Template):	<code>:GithubFollowEvent/{follower_id}/{user_id} a :GithubFollowEvent ; :githubFollower :User/{follower_id} ; :githubFollows :User/{user_id} ; :githubFollowingSince {created_at}^^xsd:dateTime .</code>
Source (SQL Query):	<code>select * from followers</code>

FIGURE B.3: Follow map

Mapping ID:	IssueMap
Target (Triples Template):	<code>:GithubIssue/{id} a :GithubIssue ; :githubIssueProject :Repository/{repo_id} ; :githubIssueReporter :User/{reporter_id} ; :githubIssueAssignee :User/{assignee_id} ; :githubIssuePullRequest :PullRequest/{pull_request_id} ; :githubIssueCreatedAt {created_at}^^xsd:dateTime .</code>
Source (SQL Query):	<code>select id, repo_id, reporter_id, assignee_id, pull_request_id, created_at from issues</code>

FIGURE B.4: Issue map

Mapping ID:	IssueLabelMap
Target (Triples Template):	<code>:GithubRepoLabel/{label_id} a :GithubRepoLabel ; :githubIssueLabelUsedBy :GithubIssue/{issue_id} .</code>
Source (SQL Query):	<code>select label_id, issue_id from issue_labels</code>

FIGURE B.5: Issue Label map

Mapping ID:	OrganizationMemberMap
Target (Triples Template):	<code>:GithubOrganizationJoinEvent/{org_id}/{user_id} a :GithubOrganizationJoinEvent ; :githubOrganizationJoinedBy :User/{user_id} ; :githubOrganizationJoinedAt {created_at}^^xsd:dateTime .</code>
Source (SQL Query):	<code>select * from organization_members</code>

FIGURE B.6: Organization member map

Mapping ID:	ProgrammingLanguageMap
Target (Triples Template):	<code>:ProgrammingLanguage/{language} a :ProgrammingLanguage .</code>
Source (SQL Query):	<code>select language from project_languages</code>

FIGURE B.7: Programming language map

Mapping ID:	ProjectMap
Target (Triples Template):	<code>:Repository/{id} a :Repository ; :repositoryUrl {url}^xsd:anyURI ; :githubHasOwner :User/{owner_id} ; :githubProjectName {name}^xsd:string ; :githubProjectDescription {description}^xsd:string ; :repositoryCreatedAt {created_at}^xsd:dateTime ; :githubForkedFrom :Repository/{forked_from} ; :githubProjectDeleted {deleted}^xsd:boolean .</code>
Source (SQL Query):	<code>select id, url, owner_id, name, description, created_at, forked_from, deleted from projects</code>

FIGURE B.8: Project map

Mapping ID:	ProjectCommit
Target (Triples Template):	<code>:Commit/{commit_id} :belongsToRepository :Repository/{project_id} .</code>
Source (SQL Query):	<code>select commit_id, project_id from project_commits</code>

FIGURE B.9: Project commit map

Mapping ID:	ProjectLabelMap
Target (Triples Template):	<code>:GithubRepoLabel/{id} a :GithubRepoLabel ; :githubRepoLabelProject :Repository/{repo_id} ; :githubRepoLabelName {name}^xsd:string .</code>
Source (SQL Query):	<code>select id, repo_id, name from repo_labels</code>

FIGURE B.10: Project label map

Mapping ID:	ProjectLanguageMap
Target (Triples Template):	<code>:GithubProjectLanguage/{project_id}/{language} a :GithubProjectLanguage ; :githubProjectLanguageRepo :Repository/{project_id} ; :githubProjectLanguages :ProgrammingLanguage/{language} ; :githubProjectLanguageBytes {bytes}^^xsd:integer ; :githubProjectLanguageTimestamp {created_at}^^xsd:dateTime .</code>
Source (SQL Query):	<code>select * from project_languages</code>

FIGURE B.11: Project programming language map

Mapping ID:	PullRequestMap
Target (Triples Template):	<code>:PullRequest/{id} a :PullRequest ; :pullRequestHeadProject :Repository/{head_repo_id} ; :pullRequestBaseProject :Repository/{base_repo_id} ; :pullRequestHeadCommit :Commit/{head_commit_id} ; :pullRequestBaseCommit :Commit/{base_commit_id} ; :id {pullreq_id}^^xsd:integer ; :githubPullRequestIntraBranch {intra_branch}^^xsd:boolean .</code>
Source (SQL Query):	<code>select id, head_repo_id, base_repo_id, head_commit_id, base_commit_id, pullreq_id, intra_branch from pull_requests</code>

FIGURE B.12: Pull Request map

Mapping ID:	PullRequestCommentMap
Target (Triples Template):	<code>:Comment/{comment_id} a :Comment ; :commentFor :PullRequest/{pull_request_id} ; :commentAuthor :User/{user_id} ; :commentBody {body}^^xsd:string ; :commentPos {position}^^xsd:integer ; :commentCreatedAt {created_at}^^xsd:dateTime .</code>
Source (SQL Query):	<code>select comment_id, pull_request_id, commit_id, user_id, body, position, created_at from pull_request_comments</code>

FIGURE B.13: Pull Request comment map

Mapping ID:	PullRequestCommitsMap
Target (Triples Template):	<code>:PullRequest/{pull_request_id} :pullRequestHasCommit :Commit/{commit_id} .</code>
Source (SQL Query):	<code>select * from pull_request_commits</code>

FIGURE B.14: Pull Request commit map

Mapping ID: PullRequestHistoryMap

Target (Triples Template):

```
:GithubPullRequestAction/{id} a :GithubPullRequestAction ; :githubPullRequestActionFor
:PullRequest/{pull_request_id} ; :githubPullRequestActionCreatedAt {created_at}^^xsd:dateTime
; :githubPullRequestActionType {action}^^xsd:string .
```

Source (SQL Query):

```
select id, pull_request_id, created_at, action from pull_request_history
```

FIGURE B.15: Pull Request history map

Mapping ID: PullRequestMergedMap

Target (Triples Template):

```
:PullRequest/{pull_request_id} :githubPullRequestMerged {is_merged}^^xsd:boolean .
```

Source (SQL Query):

```
select pull_request_id,
CASE action
WHEN 'merged' THEN 'true'
ELSE 'false'
END as is_merged
from pull_request_history where action = 'merged'
```

FIGURE B.16: Pull Request merge map

Mapping ID: PullRequestUserMap

Target (Triples Template):

```
:PullRequest/{pull_request_id} a :PullRequest ; :pullRequestUser :User/{actor_id} .
```

Source (SQL Query):

```
select pull_request_id, actor_id from pull_request_history
```

FIGURE B.17: Pull Request user map

Mapping ID: PullRequestUserMap

Target (Triples Template):

```
:PullRequest/{pull_request_id} a :PullRequest ; :pullRequestUser :User/{actor_id} .
```

Source (SQL Query):

```
select pull_request_id, actor_id from pull_request_history
```

FIGURE B.18: Pull Request user map

Mapping ID:

Target (Triples Template):

```
:User/{id} a :User ; :githubUserLogin {login}^xsd:string ; :githubUserCompany {company}^xsd:string ;
:githubUserCreatedAt {created_at}^xsd:dateTime ; :githubUserFake {fake}^xsd:boolean ;
:githubUserDeleted {deleted}^xsd:boolean ; :githubUserLng {long}^xsd:float ; :githubUserLat
{lat}^xsd:float ; :githubUserCountryCode {country_code}^xsd:string ; :githubUserState
{state}^xsd:string ; :githubUserCity {city}^xsd:string ; :githubUserLocation {location}^xsd:string ;
:githubUserIsOrg {is_organization}^xsd:boolean .
```

Source (SQL Query):

```
select id, login, company, created_at, fake, deleted, `long`, lat, country_code, state, city, location,
CASE type
WHEN 'ORG' THEN 'true'
ELSE 'false'
END as is_organization
from users
```

FIGURE B.19: User map

Mapping ID:

Target (Triples Template):

```
:commit/{id} :commit_committed_by :user/{committer_id} .
```

Source (SQL Query):

```
select id, committer_id from commits
```

FIGURE B.20: User commit map

Mapping ID:

Target (Triples Template):

```
:User/{user_id} :programsInLanguage :GithubProjectLanguage/{project_id}/{language} .
```

Source (SQL Query):

```
select distinct c.author_id as user_id, pls.language, proj.id as project_id from projects as proj
join project_languages as pls on proj.id = pls.project_id
join commits as c on c.project_id = proj.id
```

FIGURE B.21: User programming languages map

Mapping ID:

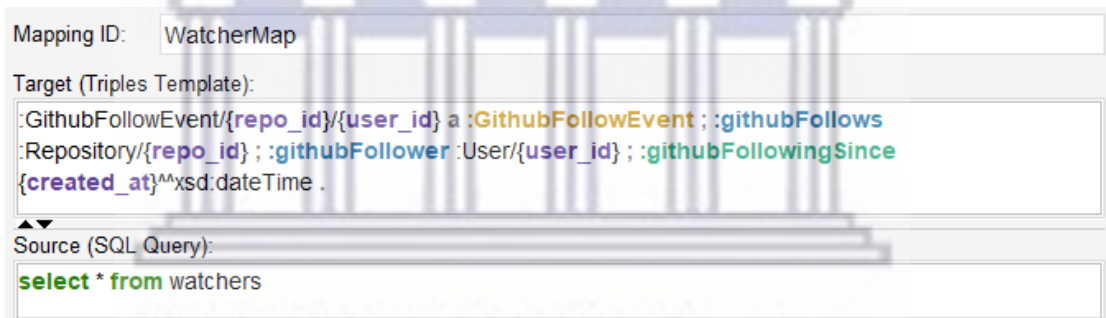
Target (Triples Template):

```
:GithubProjectMilestone/{id} a :GithubProjectMilestone ; :milestoneOf :Repository/{repo_id} ;
:githubProjectMilestoneName {name}^xsd:string .
```

Source (SQL Query):

```
select * from repo_milestones
```

FIGURE B.22: Repository milestone map



The screenshot shows a configuration window for a mapping named 'WatcherMap'. It is divided into three sections: 'Mapping ID', 'Target (Triples Template)', and 'Source (SQL Query)'. The 'Target' section contains a complex SPARQL-like template with variables and literals. The 'Source' section contains a simple SQL query.

Mapping ID: WatcherMap

Target (Triples Template):

```
:GithubFollowEvent/{repo_id}/{user_id} a :GithubFollowEvent ; :githubFollows :Repository/{repo_id} ; :githubFollower :User/{user_id} ; :githubFollowingSince {created_at}^^xsd:dateTime .
```

Source (SQL Query):

```
select * from watchers
```

FIGURE B.23: Watcher map

Appendix C

Ontology

This appendix contains a summary of the extended SemanGit ontology described in chapter 4.



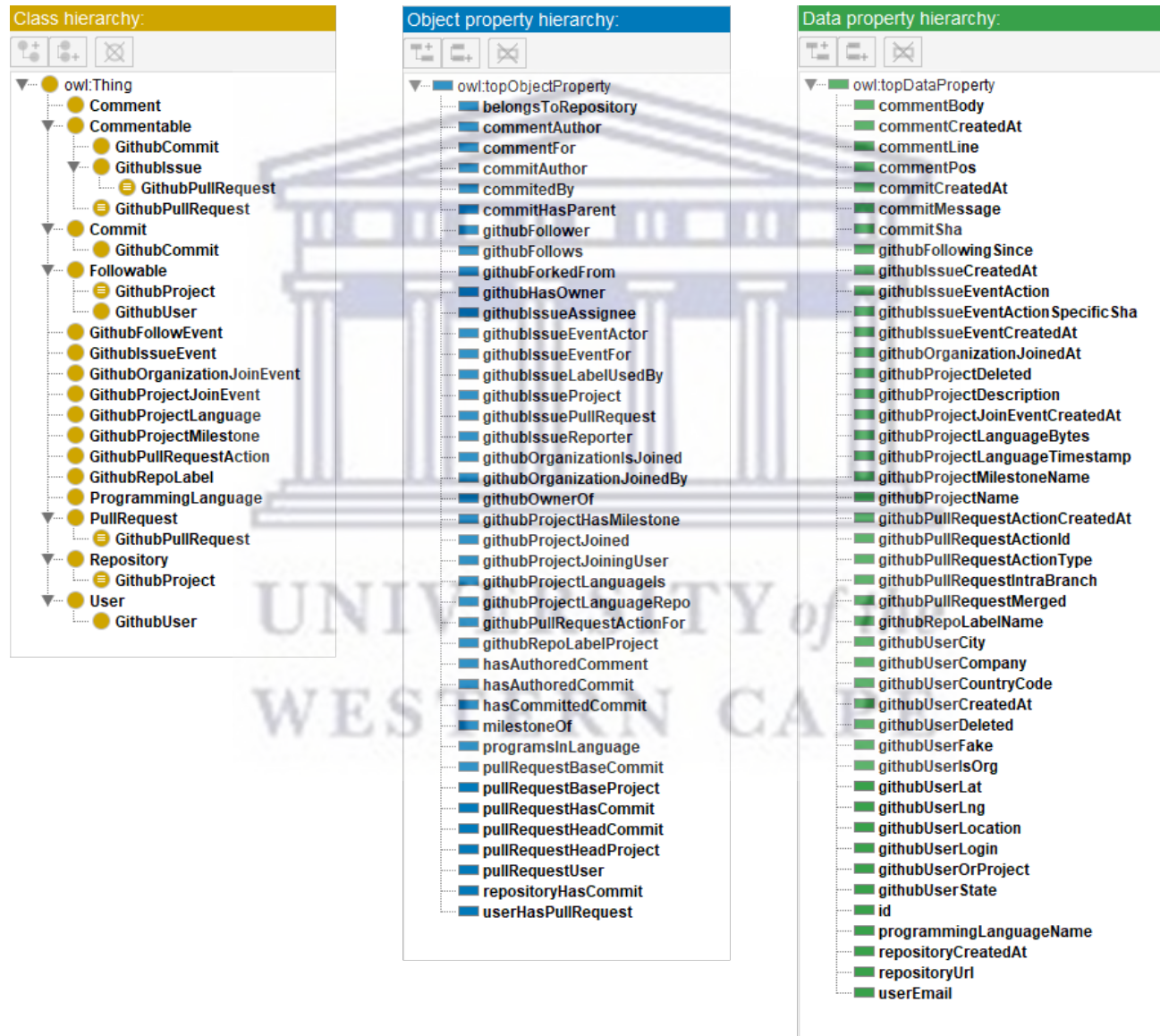


FIGURE C.1: SemanGit Ontology summary (classes and properties)

Ontology metrics:	
Metrics	
Axiom	413
Logical axiom count	202
Declaration axioms count	108
Class count	21
Object property count	40
Data property count	44
Individual count	0
Annotation Property count	5
Class axioms	
SubClassOf	7
EquivalentClasses	2
DisjointClasses	1
GCI count	0
Hidden GCI Count	2
Object property axioms	
SubObjectPropertyOf	0
EquivalentObjectProperties	0
InverseObjectProperties	6
DisjointObjectProperties	0
FunctionalObjectProperty	12
InverseFunctionalObjectProperty	4
TransitiveObjectProperty	0
SymmetricObjectProperty	0
AsymmetricObjectProperty	1
ReflexiveObjectProperty	0
IreflexiveObjectProperty	1
ObjectPropertyDomain	40
ObjectPropertyRange	40
SubPropertyChainOf	0
Data property axioms	
SubDataPropertyOf	0
EquivalentDataProperties	0
DisjointDataProperties	0
FunctionalDataProperty	0
DataPropertyDomain	44
DataPropertyRange	44
Individual axioms	
ClassAssertion	0
ObjectPropertyAssertion	0
DataPropertyAssertion	0
NegativeObjectPropertyAssertion	0
NegativeDataPropertyAssertion	0
SameIndividual	0
DifferentIndividuals	0
Annotation axioms	
AnnotationAssertion	103
AnnotationPropertyDomain	0
AnnotationPropertyRangeOf	0

FIGURE C.2: SemanGit Ontology metrics

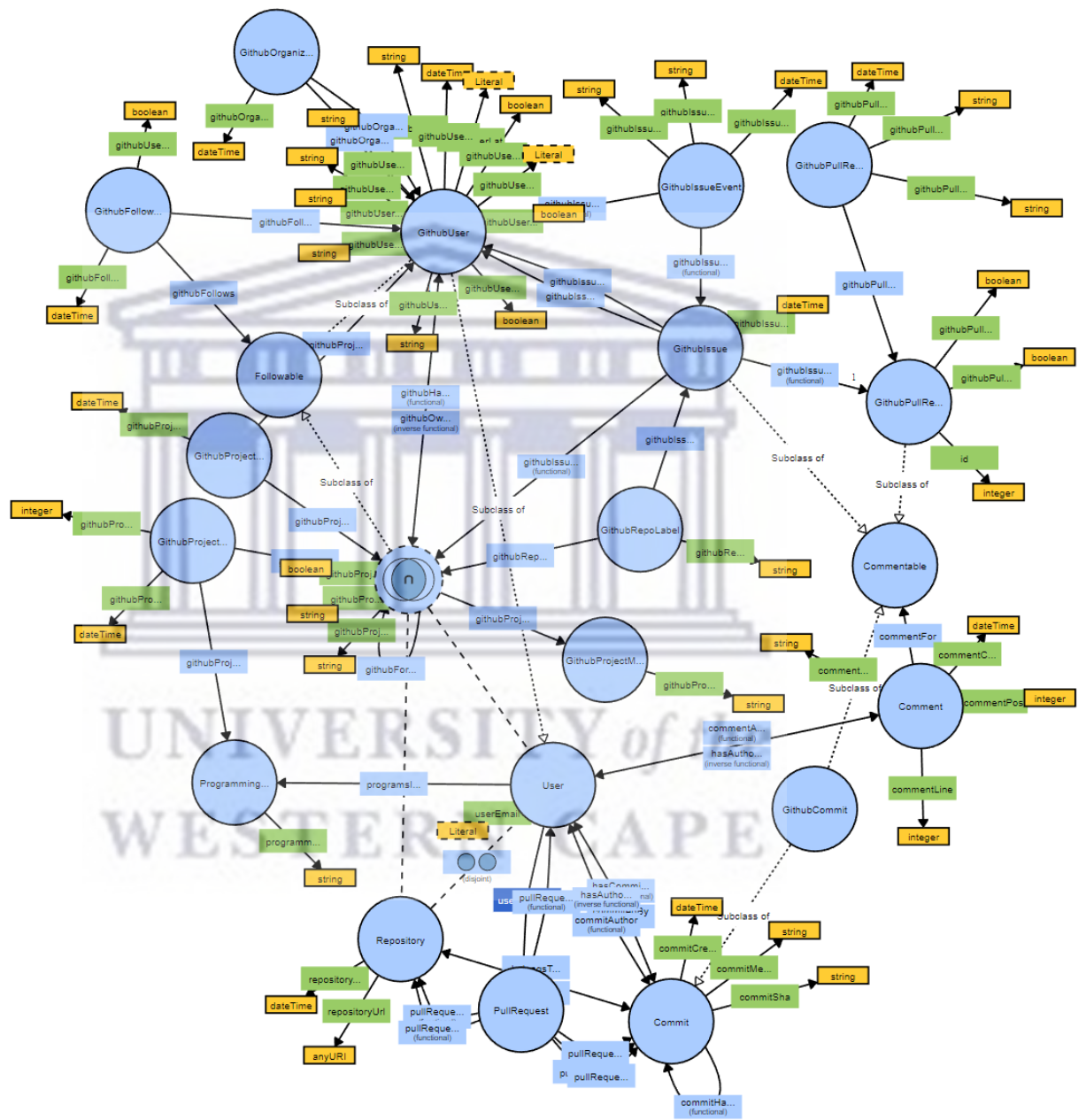


FIGURE C.3: Ontology visualisation using WebVOWL [74].

Bibliography

- [1] D. Abadi, A. Ailamaki, D. Andersen, P. Bailis, M. Balazinska, P. Bernstein, P. Boncz, S. Chaudhuri, A. Cheung, A. Doan, L. Dong, M. J. Franklin, J. Freire, A. Halevy, J. M. Hellerstein, S. Idreos, D. Kossmann, T. Kraska, S. Krishnamurthy, V. Markl, S. Melnik, T. Milo, C. Mohan, T. Neumann, B. Chin Ooi, F. Ozcan, J. Patel, A. Pavlo, R. Popa, R. Ramakrishnan, C. Ré, M. Stonebraker, and D. Suciu, “The Seattle Report on Database Research,” *SIGMOD Rec.*, vol. 48, no. 4, p. 44–53, feb 2020. [Online]. Available: <https://doi.org/10.1145/3385658.3385668>
- [2] M. AlMarzouq, A. AlZaidan, and J. AlDallal, “Mining GitHub for research and education: challenges and opportunities,” *International Journal of Web Information Systems*, vol. 16, no. 4, pp. 451–473, Jan 2020. [Online]. Available: <https://doi.org/10.1108/IJWIS-03-2020-0016>
- [3] J. Angele, M. Kifer, and G. Lausen, “Ontologies in F-Logic,” in *Handbook on Ontologies*, ser. International Handbooks on Information Systems, S. Staab and R. Studer, Eds. Springer, June 2009, pp. 45–70. [Online]. Available: https://ideas.repec.org/h/spr/ihichp/978-3-540-92673-3_2.html
- [4] G. Antoniou and F. van Harmelen, *Web Ontology Language: OWL*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 67–92. [Online]. Available: https://doi.org/10.1007/978-3-540-24750-0_4
- [5] F. Baader, I. Horrocks, C. Lutz, and U. Sattler, *Introduction to description logic*. Cambridge University Press, 2017. [Online]. Available: <https://doi.org/10.1017/9781139025355>
- [6] B. Ben Mahria, I. Chaker, and A. Zahi, “A novel approach for learning ontology from relational database: from the construction to the evaluation,” *Journal of Big Data*, vol. 8, no. 1, p. 25, Jan 2021. [Online]. Available: <https://doi.org/10.1186/s40537-021-00412-2>

- [7] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 34–43, 2001.
- [8] C. Bizer and R. Cyganiak, "D2r server-publishing relational databases on the semantic web," in *Poster at the 5th international semantic web conference*, vol. 175, 2006.
- [9] C. Bizer and A. Schultz, "The berlin sparql benchmark," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 5, no. 2, pp. 1–24, 2009.
- [10] P. Buneman, S. Khanna, and T. Wang-Chiew, "Why and where: A characterization of data provenance," in *Database Theory — ICDT 2001*, J. Van den Bussche and V. Vianu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 316–330.
- [11] J.-P. Calbimonte, O. Corcho, and A. J. Gray, "Enabling ontology-based access to streaming data sources," in *The Semantic Web–ISWC 2010: 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7–11, 2010, Revised Selected Papers, Part I 9*. Springer, 2010, pp. 96–111. [Online]. Available: https://doi.org/10.1007/978-3-642-17746-0_7
- [12] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, and G. Xiao, "Ontop: Answering SPARQL queries over relational databases," *Semantic Web*, vol. 8, no. 3, pp. 471–487, 2017. [Online]. Available: <https://doi.org/10.3233/SW-160217>
- [13] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo, "The MASTRO system for ontology-based data access," *Semantic Web*, vol. 2, no. 1, pp. 43–53, 2011. [Online]. Available: <https://doi.org/10.3233/SW-2011-0029>
- [14] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati, "Ontology-based Database Access." in *SEBD*, 2007, pp. 324–331.
- [15] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Data complexity of query answering in description logics," *Artificial Intelligence*, vol. 195, pp. 335–360, 2013. [Online]. Available: <https://doi.org/10.1016/j.artint.2012.10.003>
- [16] D. Calvanese, A. Gal, N. Haba, D. Lanti, M. Montali, A. Mosca, and R. Shraga, "ADaMaP: Automatic Alignment of Relational Data Sources Using Mapping Patterns," in *International Conference on Advanced Information Systems Engineering*, Springer. Springer International Publishing, 2021, pp. 193–209. [Online]. Available: https://doi.org/10.1007/978-3-030-79382-1_12

- [17] D. Calvanese, T. E. Kalayci, M. Montali, and S. Tinella, “Ontology-Based Data Access for Extracting Event Logs from Legacy Data: The onprom Tool and Methodology,” in *Business Information Systems: 20th International Conference, BIS 2017, Poznan, Poland, June 28–30, 2017, Proceedings 20*, Springer. Springer International Publishing, 2017, pp. 220–236. [Online]. Available: https://doi.org/10.1007/978-3-319-59336-4_16
- [18] D. Calvanese, D. Lanti, T. M. De Farias, A. Mosca, and G. Xiao, “Accessing scientific data through knowledge graphs with Ontop,” *Patterns*, vol. 2, no. 10, p. 100346, 2021.
- [19] S. Chacon and B. Straub, *Pro git*. Springer Nature, 2014. [Online]. Available: <https://doi.org/10.1007/978-1-4302-1834-0>
- [20] Y.-H. Chen, E. J.-L. Lu, and T.-A. Ou, “Intelligent SPARQL query generation for natural language processing systems,” *IEEE Access*, vol. 9, pp. 158 638–158 650, 2021. [Online]. Available: <https://doi.org/10.1109/ACCESS.2021.3130667>
- [21] S. Cluet, C. Delobel, J. Siméon, and K. Smaga, “Your mediators need data conversion!” in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, 1998, pp. 177–188. [Online]. Available: <https://doi.org/10.1145/276305.276321>
- [22] E. F. Codd, “A relational model of data for large shared data banks,” *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970. [Online]. Available: <https://doi.org/10.1145/362384.362685>
- [23] J. Coelho, M. T. Valente, L. L. Silva, and E. Shihab, “Identifying unmaintained projects in github,” in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–10. [Online]. Available: <https://doi.org/10.1145/3239235.3240501>
- [24] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, “A Systematic Mapping Study of Software Development With GitHub,” *IEEE Access*, vol. 5, pp. 7173–7192, 2017. [Online]. Available: <https://doi.org/10.1109/ACCESS.2017.2682323>
- [25] O. Dabic, E. Aghajani, and G. Bavota, “Sampling Projects in GitHub for MSR Studies,” in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 560–564. [Online]. Available: <https://doi.org/10.1109/MSR52588.2021.00074>
- [26] G. De Giacomo and M. Lenzerini, “TBox and ABox reasoning in expressive description logics.” *KR*, vol. 96, no. 316-327, p. 10, 1996. [Online]. Available: <https://api.semanticscholar.org/CorpusID:182613>

- [27] E. Della Valle and S. Ceri, "Querying the semantic web: SPARQL," in *Handbook of Semantic Web Technologies*, 2011. [Online]. Available: https://doi.org/10.1007/978-3-540-92913-0_8
- [28] L. Di-Jorio, S. Bringay, C. Fiot, A. Laurent, and M. Teisseire, "Sequential patterns for maintaining ontologies over time," in *On the Move to Meaningful Internet Systems: OTM 2008*, R. Meersman and Z. Tari, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1385–1403.
- [29] C. Franke, S. Morin, A. Chebotko, J. Abraham, and P. Brazier, "Distributed semantic web data management in HBase and MySQL cluster," in *2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 2011, pp. 105–112. [Online]. Available: <https://doi.org/10.1109/CLOUD.2011.19>
- [30] M. R. Genesereth and N. J. Nilsson, "CHAPTER 2 - Declarative Knowledge," in *Logical Foundations of Artificial Intelligence*, M. R. Genesereth and N. J. Nilsson, Eds. San Francisco (CA): Morgan Kaufmann, 1987, pp. 9–44. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780934613316500082>
- [31] G. D. Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati, "Using Ontologies for Semantic Data Integration," in *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*. Springer, 2018, pp. 187–202. [Online]. Available: https://doi.org/10.1007/978-3-319-61893-7_11
- [32] B. Glavic, "Big data provenance: Challenges and implications for benchmarking," in *Specifying Big Data Benchmarks*, T. Rabl, M. Poess, C. Baru, and H.-A. Jacobsen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 72–80.
- [33] M. Golzadeh, A. Decan, D. Legay, and T. Mens, "A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments," *Journal of Systems and Software*, vol. 175, p. 110911, 2021. [Online]. Available: <https://doi.org/10.1016/j.jss.2021.110911>
- [34] H. Gorskis, L. Aleksejeva, and I. Polaka, "Ontology-Based System Development for Medical Database Access," in *ENVIRONMENT. TECHNOLOGIES. RESOURCES. Proceedings of the International Scientific and Practical Conference*, vol. 2, 2017, pp. 24–29. [Online]. Available: <https://doi.org/10.17770/etr2017vol2.2572>
- [35] G. Gousios, "The GHTorrent dataset and tool suite," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 233–236. [Online]. Available: <https://doi.org/10.1109/MSR.2013.6624034>

- [36] G. Gousios and D. Spinellis, “GHTorrent: GitHub’s data from a firehose,” in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 2012, pp. 12–21. [Online]. Available: <https://doi.org/10.1109/MSR.2012.6224294>
- [37] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman, “Lean GHTorrent: GitHub data on demand,” in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 384–387. [Online]. Available: <https://doi.org/10.1145/2597073.2597126>
- [38] T. J. Green, *Bag Semantics*. Boston, MA: Springer US, 2009, pp. 201–206. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_979
- [39] T. Gruber, “Ontology,” *Encyclopedia of Database Systems*, 2008. [Online]. Available: <https://cir.nii.ac.jp/crid/1570009751340313600>
- [40] N. Guarino, *Formal ontology in information systems: Proceedings of the first international conference (FOIS’98), June 6-8, Trento, Italy*. IOS press, 1998, vol. 46. [Online]. Available: <https://dl.acm.org/doi/10.5555/521669>
- [41] N. Guarino, D. Oberle, and S. Staab, “What is an ontology?” in *Handbook on ontologies*, S. Staab and R. Studer, Eds. Springer Berlin Heidelberg, 2009, pp. 1–17. [Online]. Available: https://doi.org/10.1007/978-3-540-92673-3_0
- [42] A. Gusenkov, N. Bukharaev, and E. Birialtsev, “On ontology based data integration: problems and solutions,” vol. 1203, no. 1. IOP Publishing, apr 2019, p. 012059. [Online]. Available: <https://dx.doi.org/10.1088/1742-6596/1203/1/012059>
- [43] S. Hasan, E. A. Fox, K. Bisset, and M. V. Marathe, “EpiK: A Knowledge Base for Epidemiological Modeling and Analytics of Infectious Diseases,” *Journal of Healthcare Informatics Research*, vol. 1, no. 2, pp. 260–303, Dec 2017. [Online]. Available: <https://doi.org/10.1007/s41666-017-0010-9>
- [44] J. Heflin, “OWL Web Ontology Language Use Cases and Requirements,” *W3C Recommendation*, vol. 10, no. 10, pp. 1–12, 2004. [Online]. Available: <https://www.w3.org/TR/webont-req/>
- [45] S. Heymans, L. Ma, D. Anicic, Z. Ma, N. Steinmetz, Y. Pan, J. Mei, A. Fokoue, A. Kalyanpur, A. Kershenbaum *et al.*, “Ontology Reasoning with Large Data Repositories,” in *Ontology Management*. Springer US, 2008, pp. 89–128. [Online]. Available: https://doi.org/10.1007/978-0-387-69900-4_4
- [46] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, S. Rudolph *et al.*, “OWL 2 Web Ontology Language Primer (Second Edition),” *W3C recommendation*, 2012. [Online]. Available: <https://www.w3.org/TR/owl2-primer/>

- [47] M. Horridge, S. Jupp, G. Moulton, A. Rector, R. Stevens, and C. Wroe, “A practical guide to building “owl” ontologies using protégé 4 and co-ode tools edition1.2,” *The university of Manchester*, vol. 107, 2009.
- [48] I. Horrocks, P. F. Patel-Schneider, and F. Van Harmelen, “From SHIQ and RDF to OWL: The making of a web ontology language,” *Journal of Web Semantics*, vol. 1, no. 1, pp. 7–26, 2003. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1570826803000027>
- [49] N. Hu, Y. Wu, G. Qi, D. Min, J. Chen, J. Z. Pan, and Z. Ali, “An empirical study of pre-trained language models in simple knowledge graph question answering,” *World Wide Web*, pp. 1–32, May 2023. [Online]. Available: <https://doi.org/10.1007/s11280-023-01166-y>
- [50] J. Huang, D. J. Abadi, and K. Ren, “Scalable SPARQL querying of large RDF graphs,” *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 1123–1134, aug 2011. [Online]. Available: <https://doi.org/10.14778/3402707.3402747>
- [51] N. Imtiaz, J. Middleton, J. Chakraborty, N. Robson, G. Bai, and E. Murphy-Hill, “Investigating the effects of gender bias on GitHub,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, ser. ICSE ’19. IEEE, 2019, pp. 700–711. [Online]. Available: <https://doi.org/10.1109/ICSE.2019.00079>
- [52] Y. Jafta, L. Leenen, and T. Meyer, “Investigating Ontology-Based Data Access with GitHub,” in *The Semantic Web*, C. Pesquita, E. Jimenez-Ruiz, J. McCusker, D. Faria, M. Dragoni, A. Dimou, R. Troncy, and S. Hertling, Eds. Cham: Springer Nature Switzerland, 2023, pp. 644–660. [Online]. Available: https://doi.org/10.1007/978-3-031-33455-9_38
- [53] E. G. Kalaycı, I. Grangel González, F. Lösch, G. Xiao, A. ul Mehdi, E. Kharlamov, and D. Calvanese, “Semantic Integration of Bosch Manufacturing Data Using Virtual Knowledge Graphs,” in *The Semantic Web–ISWC 2020: 19th International Semantic Web Conference, Athens, Greece, November 2–6, 2020, Proceedings, Part II 19*. Springer, 2020, pp. 464–481. [Online]. Available: https://doi.org/10.1007/978-3-030-62466-8_29
- [54] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, “The promises and perils of mining GitHub,” in *Proceedings of the 11th working conference on mining software repositories*, ser. MSR 2014, 2014, pp. 92–101. [Online]. Available: <https://doi.org/10.1145/2597073.2597074>
- [55] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, “Predicting issue types on GitHub,” *Science of Computer Programming*, vol. 205, p. 102598,

2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167642320302069>
- [56] Y.-B. Kang, S. Krishnaswamy, W. Sawangphol, L. Gao, and Y.-F. Li, “Understanding and improving ontology reasoning efficiency through learning and ranking,” *Information Systems*, vol. 87, p. 101412, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0306437917306476>
- [57] C. M. Keet, *An introduction to ontology engineering*. University of Cape Town, 2018.
- [58] E. Kharlamov, D. Hovland, E. Jiménez-Ruiz, D. Lanti, H. Lie, C. Pinkel, M. Rezk, M. G. Skjæveland, E. Thorstensen, G. Xiao *et al.*, “Ontology Based Access to Exploration Data at Statoil,” in *The Semantic Web-ISWC 2015: 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II 14*. Springer International Publishing, 2015, pp. 93–112. [Online]. Available: https://doi.org/10.1007/978-3-319-25010-6_6
- [59] E. Kharlamov, D. Hovland, M. G. Skjæveland, D. Bilidas, E. Jiménez-Ruiz, G. Xiao, A. Soylu, D. Lanti, M. Rezk, D. Zheleznyakov *et al.*, “Ontology Based Data Access in Statoil,” *Journal of Web Semantics*, vol. 44, pp. 3–36, 2017, Industry and In-use Applications of Semantic Technologies. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1570826817300276>
- [60] E. Kharlamov, Y. Kotidis, T. Mailis, C. Neuenstadt, C. Nikolaou, Ö. Özçep, C. Svingos, D. Zheleznyakov, Y. Ioannidis, S. Lamparter *et al.*, “An ontology-mediated analytics-aware approach to support monitoring and diagnostics of static and streaming data,” *Journal of Web Semantics*, vol. 56, pp. 30–55, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1570826819300010>
- [61] E. Kharlamov, T. Mailis, G. Mehdi, C. Neuenstadt, Ö. Özçep, M. Roshchin, N. Solomakhina, A. Soylu, C. Svingos, S. Brandt *et al.*, “Semantic access to streaming and static data at Siemens,” *Journal of Web Semantics*, vol. 44, pp. 54–74, 2017, industry and In-use Applications of Semantic Technologies. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1570826817300124>
- [62] E. Kharlamov, O. Savković, M. Ringsquandl, G. Xiao, G. Mehdi, E. G. Kalayc, W. Nutt, M. Roshchin, I. Horrocks, and T. Runkler, “Diagnostics of Trains with Semantic Diagnostics Rules,” in *Inductive Logic Programming:*

- 28th International Conference, ILP 2018, Ferrara, Italy, September 2–4, 2018, Proceedings 28*. Springer, 2018, pp. 54–71. [Online]. Available: https://doi.org/10.1007/978-3-319-99960-9_4
- [63] T. Kinsman, M. Wessel, M. A. Gerosa, and C. Treude, “How do software developers use GitHub Actions to automate their workflows?” in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 420–431. [Online]. Available: <https://doi.org/10.1109/MSR52588.2021.00054>
- [64] K. I. Kotis, G. A. Vouros, and D. Spiliotopoulos, “Ontology engineering methodologies for the evolution of living and reused ontologies: status, trends, findings and recommendations,” *The Knowledge Engineering Review*, vol. 35, p. e4, 2020.
- [65] M. Krötzsch, F. Simancik, and I. Horrocks, “A Description Logic Primer,” *arXiv preprint arXiv:1201.4089*, vol. abs/1201.4089, 2012. [Online]. Available: <https://api.semanticscholar.org/CorpusID:1221862>
- [66] D. O. Kubitza, M. Böckmann, and D. Graux, “SemanGit: A Linked Dataset from git,” in *The Semantic Web – ISWC 2019*. Springer, 2019, pp. 215–228. [Online]. Available: https://doi.org/10.1007/978-3-030-30796-7_14
- [67] B. Lakzaei and M. Shamsfard, “Ontology learning from relational databases,” *Information Sciences*, vol. 577, pp. 280–297, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0020025521006654>
- [68] M. Lenzerini and C. Daraio, *Challenges, Approaches and Solutions in Data Integration for Research and Innovation*. Cham: Springer International Publishing, 2019, pp. 397–420. [Online]. Available: https://doi.org/10.1007/978-3-030-02511-3_15
- [69] S. Liang, K. Stockinger, T. M. de Farias, M. Anisimova, and M. Gil, “Querying knowledge graphs in natural language,” *Journal of big data*, vol. 8, pp. 1–23, 2021. [Online]. Available: <https://doi.org/10.1186/s40537-020-00383-w>
- [70] C.-h. Liao, Y.-f. Wu, and G.-h. King, “Research on Learning OWL Ontology from Relational Database,” in *Journal of Physics: Conference Series*, vol. 1176, no. 2. IOP Publishing, 2019, p. 022031. [Online]. Available: <https://dx.doi.org/10.1088/1742-6596/1176/2/022031>
- [71] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, and S. Liu, “Exploring the characteristics of issue-related behaviors in github using visualization techniques,” *IEEE Access*, vol. 6, pp. 24 003–24 015, 2018. [Online]. Available: <https://doi.org/10.1109/ACCESS.2018.2810295>

- [72] J. Liu, J. Li, and L. He, “A Comparative Study of the Effects of Pull Request on GitHub Projects,” in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2016, pp. 313–322. [Online]. Available: <https://doi.org/10.1109/COMPSAC.2016.27>
- [73] J. Loeliger and M. McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development.* ” O’Reilly Media, Inc.”, 2012.
- [74] S. Lohmann, V. Link, E. Marbach, and S. Negru, “WebVOWL: Web-based visualization of ontologies,” in *Knowledge Engineering and Knowledge Management: EKAW 2014 Satellite Events, VISUAL, EKM1, and ARCOE-Logic, Linköping, Sweden, November 24-28, 2014. Revised Selected Papers. 19.* Springer, 2015, pp. 154–158.
- [75] M. Luczak-Rösch, “Towards agile ontology maintenance,” in *The Semantic Web - ISWC 2009*, A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 965–972.
- [76] C. Ma and B. Molnr, “Ontology learning from relational database: Opportunities for semantic information integration,” *Vietnam Journal of Computer Science*, vol. 9, no. 01, pp. 31–57, 2022. [Online]. Available: <https://doi.org/10.1142/S219688882150024X>
- [77] C. Ma and B. Molnár, “Use of ontology learning in information system integration: a literature survey,” in *Intelligent Information and Database Systems: 12th Asian Conference, ACIIDS 2020, Phuket, Thailand, March 23–26, 2020, Proceedings 12.* Springer, 2020, pp. 342–353. [Online]. Available: <https://api.semanticscholar.org/CorpusID:212564890>
- [78] M. Madsen and O. Lhoták, “Fixpoints for the masses: programming with first-class Datalog constraints,” *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–28, 2020. [Online]. Available: <https://doi.org/10.1145/3428193>
- [79] M. N. Mami, D. Graux, S. Scerri, H. Jabeen, S. Auer, and J. Lehmann, “Squerall: Virtual Ontology-Based Access to Heterogeneous and Large Data Sources,” in *The Semantic Web–ISWC 2019: 18th International Semantic Web Conference, Auckland, New Zealand, October 26–30, 2019, Proceedings, Part II 18.* Springer, 2019, pp. 229–245. [Online]. Available: <https://api.semanticscholar.org/CorpusID:204754566>

- [80] F. Manola, E. Miller, B. McBride *et al.*, “RDF primer,” *W3C recommendation*, vol. 10, no. 1-107, p. 6, 2004. [Online]. Available: <https://www.w3.org/TR/rdf-primer/>
- [81] M. A. Martínez-Prieto, M. Arias Gallego, and J. D. Fernández, “Exchange and Consumption of Huge RDF Data,” in *Extended Semantic Web Conference*. Springer Berlin Heidelberg, 2012, pp. 437–452. [Online]. Available: https://doi.org/10.1007/978-3-642-30284-8_36
- [82] H. E. Massari, S. Mhammedi, N. Gherabi, and M. Nasri, “Virtual OBDA Mechanism Ontop for Answering SPARQL Queries Over Couchbase,” in *International Conference on Advanced Technologies for Humanity*. Springer, 2021, pp. 193–205. [Online]. Available: https://doi.org/10.1007/978-3-030-94188-8_19
- [83] N. Matentzoglou, D. Goutte-Gattat, S. Z. K. Tan, J. P. Balhoff, S. Carbon, A. R. Caron, W. D. Duncan, J. E. Flack, M. Haendel, N. L. Harris, W. R. Hogan, C. T. Hoyt, R. C. Jackson, H. Kim, H. Kir, M. Larralde, J. A. McMurry, J. A. Overton, B. Peters, C. Pilgrim, R. Stefancsik, S. M. Robb, S. Toro, N. A. Vasilevsky, R. Walls, C. J. Mungall, and D. Osumi-Sutherland, “Ontology Development Kit: a toolkit for building, maintaining and standardizing biomedical ontologies,” *Database*, vol. 2022, p. baac087, 10 2022. [Online]. Available: <https://doi.org/10.1093/database/baac087>
- [84] B. McBride, “The resource description framework (RDF) and its vocabulary description language RDFS,” in *Handbook on ontologies*. Springer Berlin Heidelberg, 2004, pp. 51–65. [Online]. Available: https://doi.org/10.1007/978-3-540-24750-0_3
- [85] K. McGlenn, M. A. Rutherford, K. Gisslander, L. Hederman, M. A. Little, and D. O’Sullivan, “FAIRVASC: A semantic web approach to rare disease registry integration,” *Computers in Biology and Medicine*, vol. 145, p. 105313, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0010482522001056>
- [86] C. Messaoudi, R. Fissoune, and H. Badir, “IPDS: A semantic mediator-based system using Spark for the integration of heterogeneous proteomics data sources,” *Concurrency and Computation: Practice and Experience*, vol. 33, no. 1, p. e5814, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:219513264>
- [87] R. D. Morris, “Web 3.0: Implications for online learning,” 2011. [Online]. Available: <https://doi.org/10.1007/s11528-011-0469-9>

- [88] C. Nikolaou, E. V. Kostylev, G. Konstantinidis, M. Kaminski, B. C. Grau, and I. Horrocks, “The Bag Semantics of Ontology-Based Data Access,” *arXiv preprint arXiv:1705.07105*, 2017. [Online]. Available: <https://doi.org/10.48550/arXiv.1705.07105>
- [89] N. F. Noy, D. L. McGuinness *et al.*, “Ontology development 101: A guide to creating your first ontology,” 2001. [Online]. Available: <https://api.semanticscholar.org/CorpusID:500106>
- [90] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A Design Science Research Methodology for Information Systems Research,” *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007. [Online]. Available: <https://doi.org/10.2753/MIS0742-1222240302>
- [91] N. Petersen, L. Halilaj, I. Grangel-González, S. Lohmann, C. Lange, and S. Auer, “Realizing an RDF-Based Information Model for a Manufacturing Company – A Case Study,” in *The Semantic Web–ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part II 16*. Springer, 2017, pp. 350–366. [Online]. Available: https://doi.org/10.1007/978-3-319-68204-4_31
- [92] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati, “Linking Data to Ontologies,” in *Journal on data semantics X*. Springer, Berlin, Heidelberg, 2008, pp. 133–173. [Online]. Available: https://doi.org/10.1007/978-3-540-77688-8_5
- [93] F. Priyatna, R. Alonso-Calvo, S. Paraiso-Medina, and O. Corcho, “Querying clinical data in HL7 RIM based relational model with morph-RDB,” *Journal of biomedical semantics*, vol. 8, no. 1, pp. 1–12, 2017. [Online]. Available: <https://doi.org/10.1186/s13326-017-0155-8>
- [94] F. Priyatna, O. Corcho, and J. Sequeda, “Formalisation and experiences of R2RML-based SPARQL to SQL query translation using Morph,” in *Proceedings of the 23rd international conference on World wide web*, 2014, pp. 479–490. [Online]. Available: <https://doi.org/10.1145/2566486.2567981>
- [95] A. Rastogi, N. Nagappan, G. Gousios, and A. van der Hoek, “Relationship between geographical location and evaluation of developer contributions in github,” in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–8. [Online]. Available: <https://doi.org/10.1145/3239235.3240504>

- [96] R. Rudman and R. Bruwer, “Defining web 3.0: opportunities and challenges,” *The Electronic Library*, vol. 34, 2016. [Online]. Available: <https://doi.org/10.1108/EL-08-2014-0140>
- [97] G. Santipantakis, K. Kotis, and G. A. Vouros, “OBDAIR: Ontology-Based Distributed framework for Accessing, Integrating and Reasoning with data in disparate data sources,” *Expert Systems with Applications*, vol. 90, pp. 464–483, 2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0957417417305705>
- [98] O. Savković, E. Kharlamov, M. Ringsquandl, G. Xiao, G. Mehdi, E. G. Kalayc, W. Nutt, and I. Horrocks, “Semantic diagnostics of smart factories,” in *Semantic Technology: 8th Joint International Conference, JIST 2018, Awaji, Japan, November 26–28, 2018, Proceedings 8*. Springer, 2018, pp. 277–294. [Online]. Available: https://doi.org/10.1007/978-3-030-04284-4_19
- [99] T. Schneider and M. Šimkus, “Ontologies and Data Management: A Brief Survey,” *KI-Künstliche Intelligenz*, vol. 34, no. 3, pp. 329–353, 2020. [Online]. Available: <https://doi.org/10.1007/s13218-020-00686-3>
- [100] G. Sejdiu, D. Graux, I. Khan, I. Lytra, H. Jabeen, and J. Lehmann, “Towards a Scalable Semantic-Based Distributed Approach for SPARQL Query Evaluation,” in *International Conference on Semantic Systems*. Springer, Cham, 2019, pp. 295–309. [Online]. Available: https://doi.org/10.1007/978-3-030-33220-4_22
- [101] J. F. Sequeda and D. P. Miranker, “Ultrawrap: SPARQL execution on relational data,” *Journal of Web Semantics*, vol. 22, pp. 19–39, 2013. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1570826813000383>
- [102] A. P. Sheth and J. A. Larson, “Federated database systems for managing distributed, heterogeneous, and autonomous databases,” *ACM Computing Surveys (CSUR)*, vol. 22, no. 3, pp. 183–236, 1990. [Online]. Available: <https://doi.org/10.1145/96602.96604>
- [103] A. P. Sheth and C. Ramakrishnan, “Semantic (Web) Technology in Action: Ontology Driven Information Systems for Search, Integration, and Analysis,” *IEEE Data Engineering Bulletin*, vol. 26, no. 4, p. 40, 2003.
- [104] A. C. Sima, T. Mendes de Farias, E. Zbinden, M. Anisimova, M. Gil, H. Stockinger, K. Stockinger, M. Robinson-Rechavi, and C. Dessimoz, “Enabling semantic queries across federated bioinformatics databases,” *Database*, vol. 2019, 11 2019. [Online]. Available: <https://doi.org/10.1093/database/baz106>

- [105] G. Singh, S. Bhatia, and R. Mutharaju, “OWL2Bench: A Benchmark for OWL 2 Reasoners,” in *International semantic web conference*. Springer International Publishing, 2020, pp. 81–96. [Online]. Available: https://doi.org/10.1007/978-3-030-62466-8_6
- [106] D. Spinellis, “Git,” *IEEE software*, vol. 29, no. 3, pp. 100–101, 2012. [Online]. Available: <https://doi.org/10.1109/MS.2012.61>
- [107] S. Staab and R. Studer, *Handbook on ontologies*. Springer Science & Business Media, 2010. [Online]. Available: <https://doi.org/10.1007/978-3-540-92673-3>
- [108] C. Stadler, G. Sejdiu, D. Graux, and J. Lehmann, “Sparklify: A Scalable Software Component for Efficient Evaluation of SPARQL Queries over Distributed RDF Datasets,” in *International Semantic Web Conference*. Springer, 2019, pp. 293–308. [Online]. Available: https://doi.org/10.1007/978-3-030-30796-7_19
- [109] F. Taglino, F. Cumbo, G. Antognoli, I. Arisi, M. D’Onofrio, F. Perazzoni, R. Voyat, G. Fiscon, F. Conte, M. Canevelli *et al.*, “An ontology-based approach for modelling and querying Alzheimer’s disease data,” *BMC Medical Informatics and Decision Making*, vol. 23, no. 1, pp. 1–15, 2023. [Online]. Available: <https://doi.org/10.1186/s12911-023-02211-6>
- [110] R. Thirumahal, G. Sudha Sadasivam, and P. Shruti, “Semantic Integration of Heterogeneous Data Sources Using Ontology-Based Domain Knowledge Modeling for Early Detection of COVID-19,” *SN Computer Science*, vol. 3, no. 6, p. 428, 2022. [Online]. Available: <https://doi.org/10.1007/s42979-022-01298-4>
- [111] W. Van Der Aalst, *Process mining: Data Science in Action*. Springer Berlin, Heidelberg, 2016, vol. 2. [Online]. Available: <https://doi.org/10.1007/978-3-662-49851-4>
- [112] G. Vega-Gorgojo, L. Slaughter, and M. Giese, “Seeing the whole picture: integrated pre-surgery reports with PreOptique,” *Journal of Biomedical Semantics*, vol. 10, pp. 1–15, 2019. [Online]. Available: <https://doi.org/10.1186/s13326-019-0197-1>
- [113] P. S. Vikas Trikha and S. Kothari, “Managing Data Provenance in the Semantic Web,” *INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH TECHNOLOGY (IJERT)*, vol. 8, no. 3, 2019.
- [114] J. Vom Brocke, A. Hevner, and A. Maedche, “Introduction to Design Science Research,” *Design science research. Cases*, pp. 1–13, 2020. [Online]. Available: https://doi.org/10.1007/978-3-030-46781-4_1

- [115] J. Wachs, M. Nitecki, W. Schueller, and A. Polleres, “The Geography of Open Source Software: Evidence from GitHub,” *Technological Forecasting and Social Change*, vol. 176, p. 121478, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0040162522000105>
- [116] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati, and M. Zakharyashev, “Ontology-based data access: A survey.” *International Joint Conferences on Artificial Intelligence*, 7 2018, pp. 5511–5519. [Online]. Available: <https://doi.org/10.24963/ijcai.2018/777>
- [117] G. Xiao, L. Ding, B. Cogrel, and D. Calvanese, “Virtual knowledge graphs: An overview of systems and use cases,” *Data Intelligence*, vol. 1, no. 3, pp. 201–223, 2019. [Online]. Available: <https://doi.org/10.1162/dint.a.00011>
- [118] G. Xiao, R. Kontchakov, B. Cogrel, D. Calvanese, and E. Botoeva, “Efficient handling of SPARQL optional for OBDA,” in *International Semantic Web Conference*. Springer, Cham, 2018, pp. 354–373. [Online]. Available: https://doi.org/10.1007/978-3-030-00671-6_21
- [119] G. Xiao, D. Lanti, R. Kontchakov, S. Komla-Ebri, E. Güzel-Kalaycı, L. Ding, J. Corman, B. Cogrel, D. Calvanese, and E. Botoeva, “The Virtual Knowledge Graph System Ontop,” in *International Semantic Web Conference*. Springer, Cham, 2020, pp. 259–277. [Online]. Available: https://doi.org/10.1007/978-3-030-62466-8_17
- [120] H. Zhang, Y. Guo, Q. Li, T. J. George, E. Shenkman, F. Modave, and J. Bian, “An ontology-guided semantic data integration framework to support integrative data analysis of cancer survival,” *BMC medical informatics and decision making*, vol. 18, no. 2, pp. 129–147, 2018. [Online]. Available: <https://doi.org/10.1186/s12911-018-0636-4>
- [121] H. Zhang, Y. Guo, Q. Li, T. J. George, E. A. Shenkman, and J. Bian, “Data Integration through Ontology-Based Data Access to Support Integrative Data Analysis: A Case Study of Cancer Survival,” in *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2017, pp. 1300–1303. [Online]. Available: <https://doi.org/10.1109/BIBM.2017.8217849>
- [122] X. Zhang, M. Zhang, P. Peng, J. Song, Z. Feng, and L. Zou, “A Scalable Sparse Matrix-Based Join for SPARQL Query Processing,” in *Database Systems for Advanced Applications*, G. Li, J. Yang, J. Gama, J. Natwichai, and Y. Tong, Eds. Cham: Springer International Publishing, 2019, pp. 510–514. [Online]. Available: https://doi.org/10.1007/978-3-030-18590-9_77