Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Department of Information Engineering

Master Thesis in Computer Engineering

Federated Data Analytics for Genomics Data

Master Candidate

Mirco Cazzaro

Student ID 2076745

Supervisor

Prof. Gianmaria Silvello

University of Padova

Academic Year
2023/2024

Date: 3 September 2024

To all the people that were beside me in this journey

Abstract

Biomedical data management is increasingly complex due to the variety of storage systems and evolving data models. This heterogeneity presents obstacles to data integration and querying, crucial for advancing biomedical research and healthcare. The project will involve designing a system architecture that supports the integration of heterogeneous data stores under a unified federated system. The system will also utilize principles from the Ontology-Based Data Access (OBDA) paradigm to facilitate semantic querying capabilities. By developing a federated data analytics system for genomics data, this thesis will contribute to reducing the complexities involved in biomedical data management. The system will enable more effective data integration and querying processes, thereby supporting faster and more accurate genomics research. The completion of this project will result in a prototype of a federated data analytics system capable of handling the specific needs of genomics data.

Sommario

La gestione dei dati biomedici è sempre più complessa a causa della varietà dei sistemi di archiviazione e dei modelli di dati in evoluzione. Questa eterogeneità presenta ostacoli all'integrazione e alla consultazione dei dati, cruciali per il progresso della ricerca biomedica e dell'assistenza sanitaria. Il progetto comporterà la progettazione di un'architettura di sistema che supporti l'integrazione di archivi di dati eterogenei sotto un sistema federato unificato. Il sistema utilizzerà anche i principi del paradigma dell'Accesso ai Dati Basato su Ontologie (OBDA) per facilitare le capacità di consultazione semantica. Sviluppando un sistema federato di analisi dei dati per i dati genomici, questa tesi contribuirà a ridurre le complessità coinvolte nella gestione dei dati biomedici. Il sistema permetterà processi di integrazione e consultazione dei dati più efficaci, supportando così una ricerca genomica più rapida e accurata. Il completamento di questo progetto porterà alla realizzazione di un prototipo di un sistema federato di analisi dei dati capace di gestire le esigenze specifiche dei dati genomici.

Contents

List of Figures

List of Tables

List of Code Snippets

List of Acronyms

RDF Resource Description Framework

URI Uniform Resource Identifier

KG Knowledge Graph

OWL Ontology Web Language

DBMS Database Management System

SQL Structured Query Language

OBDA Ontology-Based Data Access

API Application Program Interface

JDBC Java Database Connectivity

ODBC Open Database Connectivity

BigDAWG Big Data Analytics Working Group

FOSS Free and Open Source

ARP Advanced Relational Pushdown

SPARQL SPARQL Protocol and RDF Query Language

OBDF Ontology-Based Data Federation

W3C World Wide Web Consortium

STARQL Streaming and Temporal ontology Access with a Reasoning-based Query Language

DL Description Logic

CSV Comma Separated Values

JAR Java Archive

JSON JavaScript Object Notation

VCF Virtual Contact File

NAS Network Attached Storage

LAN Local Area Network

SMB Server Message Block

FUSE Filesystem in Userspace

GUI Graphical User Interface

VKG Virtual Knowledge Graph

HEREDITARY HEteRogeneous sEmantic Data Integration for the guT-brAin inteRplaY

BRAINTEASER BRinging Artificial INTelligencE home for a better cAre of amyotrophic lateral sclerosis and multiple SclERosis

GDPR General Data Protection Regulation

WP Work Package

ALS Amyotrophic Lateral Sclerosis

ALSFRS Amyotrophic Lateral Sclerosis Functional Rating Scale

BSBM Berlin SPARQL Benchmark

SEASHELL reSource bEnchmark for Analysis taSks in HEaLthcare data Lakes

IT Information Technology

EHR Electronic Health Records

CPU Central Processing Unit

RAM Random Access Memory

1 Introduction

1.1 State of the Art

In recent years, the increasing complexity of biomedical data has posed significant challenges to researchers and clinicians. The rapid evolution of data storage systems and the diversity of data models have contributed to these challenges, creating a heterogeneous landscape that is difficult to navigate. This complexity is particularly pronounced in the field of genomics, where the integration and analysis of diverse datasets are crucial for advancing our understanding of genetic diseases and developing personalized medical treatments. The advent of big data analytics has led to new possibilities for handling large volumes of complex biomedical data. However, the vast diversity of data sources (e.g. relational databases, hierarchical storage systems and graph-based models) necessitates the development of sophisticated data integration frameworks. These frameworks must not only accommodate the different data models but also enable seamless querying and analysis across these models. The need for such frameworks is especially crucial in genomics, where the ability to integrate and analyze data from multiple sources can significantly accelerate research and improve clinical outcomes. The focus of this thesis is on the design and implementation of a federated data analytics system specifically tailored for the integration of clinical and genomics data. This system is designed to address the challenges associated with integrating heterogeneous data sources, enabling researchers to perform complex queries across multiple datasets without the need for extensive data preprocessing or manual data integration. By leveraging the principles of OBDA, the proposed system eases semantic querying capabilities, allowing users to extract meaningful insights from the data more efficiently. OBDA is a powerful paradigm for data integration, particularly in environments characterized by data heterogeneity. Although research activities on OBDA started more than 20 years ago, it is still nowadays a subject of discussion in the academic field as well as in the industry, having periodically novel papers discussing both new frameworks or their improvements and their employment in industrial scenarios. OBDA allows for the seamless integration of relational databases into an ontology framework, enabling users to perform semantic queries that transcend the limitations of traditional data retrieval methods. The use of ontologies in OBDA provides a shared vocabulary and a formalized structure for representing knowledge within a specific domain, which is particularly beneficial in the field of genomics where data is often complex and highly interconnected. The system developed in this thesis builds upon the OBDA paradigm by integrating it into a federated data architecture. This architecture is designed to support the integration of multiple heterogeneous data sources, including relational databases, NoSQL systems, and cloud storage solutions. By creating a unified federated system, the architecture allows for real-time data retrieval and integration, eliminating the need for data deduplication and ensuring that researchers have access to the most current data available. A key component of this system is the use of a specialized ontology that models the intricate relationships between genomic data and clinical information. The ontology serves as the backbone of the system, enabling the semantic integration of data from diverse sources and facilitating complex queries that would be difficult or impossible to perform using traditional data retrieval methods. The ontology's design is informed by the specific requirements of clinical and genomics research, with a focus on ensuring interoperability and scalability as new data sources and data types are introduced. The federated architecture proposed in this thesis also incorporates advanced data federation techniques, which are essential for managing the diversity of data sources in genomics. Data federation allows for the creation of a virtual data access layer that abstracts the underlying technical details of each data source, enabling researchers to query data using standard languages like SQL without needing to know where the data is physically stored or in what format. This approach not only simplifies the data retrieval process but also minimizes the risks associated with data movement and duplication. Furthermore, the system is designed with scalability in mind, allowing it to accommodate new data sources as they become available. The system's architecture is flexible enough to integrate these new data sources seamlessly, ensuring that researchers can always work with the most comprehensive dataset possible. Another critical aspect of the system is its focus on privacy and data security. Given the sensitive nature of clinical and genomic data, especially when data is strictly related to patients' personal information, the system aims to adhere to modern data protection regulations. This is to ensure that while data is integrated and analyzed, it remains protected and secure, ensuring patient privacy and maintaining the trust of the institutions and individuals who provide the data. The thesis also explores the practical application of the proposed system within the context of the HEteRogeneous sEmantic Data Integration for the guT-brAin inteRplaY (HEREDITARY) project, a European Union funded initiative focused on integrating multimodal data to advance the understanding of brain diseases. The HEREDITARY project represents a real-world application of the federated data analytics system, demonstrating its potential to facilitate complex analyses and drive new insights in biomedical research. Finally, an extensive benchmarking phase will evaluate the performances of the proposed architecture considering both its strengths as a Database Management System (DBMS), analyzing aspects as average execution time and throughput, and the resource consumption of the system, in order to figure out whether such an application adoption may be feasible across diverse hardware environments. In summary, this thesis presents a comprehensive approach to addressing the challenges of integrating and analyzing heterogeneous clinical and genomics data. Posing its foundations on the OBDA paradigm and advanced data federation techniques, the proposed system offers a robust and scalable solution for managing the complexities of biomedical data.

1.2 Thesis Outline

The subsequent chapters are structured to thoroughly explore the necessary backgrounds, detail the methodology, demonstrate the application through a use case, and evaluate the system's performance. In particular, Chapter 2 delves into the technical and methodological characteristics of data federation systems, covering essential concepts such as the Resource Description Framework, Knowledge Graphs, and Ontologies, crucial for understanding their applications. Chapter 3 discusses the challenges of integrating diverse biomedical datasets within the HEREDITARY project, setting the stage by describing the complexities of the data landscape and emphasizing the need for robust integration tools. In Chapter 4, we review related works, focusing on existing solutions and technologies that address similar challenges, highlighting the gaps in current approaches and justifying the necessity for our proposed system. Chapter 5 presents the detailed architecture of our federated data analytics system, explaining the design choices, data sources, and integration mechanisms that enable effective data querying and retrieval across heterogeneous sources. Chapter 6 showcases the application of our system within the HEREDITARY project, providing a practical demonstration of its capabilities and effectiveness in a real-world scenario. Chapter 7 is dedicated to benchmarking the system's architecture, analyzing its performance, reliability, and scalability through quantitative data to substantiate its efficiency. The thesis concludes with Chapter 8, which summarizes the findings, outlines the significant contributions of the research, and suggests directions for future work based on the encountered limitations and the evolving landscape of data integration technologies.

2 Background

2.1 Methodological Background

2.1.1 Resource Description Framework

The Resource Description Framework (RDF) is a standard[1] by the World Wide Web Consortium (W3C) designed for data interchange on the web. Its definitive syntax was lastly defined on 2003 [1].

RDF is based on the idea of making statements about resources, expressed as triples: for example, a triple might consist of "Gene123" (subject) "hasFunction" (predicate) "DNA Repair" (object). These triples are stored in a graph, making RDF uniquely suited for modern data analytics where relationships and linkages are crucial: this structure is by design flexible, and it is used to represent information in a way that makes it easier to aggregate, integrate, and manage diverse data from various sources.

The standard utilizes Uniform Resource Identifier (URI) to ensure that each element in the triple is uniquely identifiable, allowing to link data across different datasets seamlessly. RDF also supports literal values and data types, enabling detailed descriptions of properties and values.

In summary, RDF provides a robust and flexible framework for describing and interlinking data on the web, which is crucial for any federated data system

[1] https://www.w3.org/RDF/

that aims to integrate diverse data sources effectively.

2.1.2 Knowledge Graphs

Knowledge Graphs represent an innovative way of structuring and querying interconnected data. A Knowledge Graph (KG) organizes data in a graph format, where entities (nodes) and their interrelations (edges) are defined according to a schema that encapsulates both the entities and the possible links between them. This structure allows not only to better visualize data, but is also very well suited for data exploration and analysis.

Central to the concept of KG is the idea of enhancing search and data discovery capabilities beyond simple data retrieval. By semantically linking data entities, KG allow for more intuitive and sophisticated queries that are closer to natural language questions. This capability makes them useful in complex domains like biomedical research, where users may need to uncover hidden relationships and patterns among vast datasets.

Furthermore, KG well suits in scenarios requiring data integration from disparate sources. They support the combination of structured and unstructured data and they can scale well as new data need to be integrated. This flexibility is crucial in fields such as genomics, where new data attributes and relationships are continuously being discovered and need to be rapidly integrated into existing datasets.

In practice, KG are usually powered by technologies such as RDF and other standards discussed earlier, leveraging the strengths of these frameworks to ensure robust data handling and scalability.

2.1.3 Ontologies

Ontologies are a shared vocabulary for a particular domain. They serve as a crucial tool in structuring data by defining classes where data entities can be categorized. This classification system not only standardizes data representation but also simplifies the communication between different systems and users by providing a common understanding of the domain.

Ontologies involve the use of first-order logic to define rules about the relationships between different classes. These rules allow for the logical inference of new information from the data that is already known, which can significantly deepen data analysis and querying capabilities. This set of first-order logic rules used to model ontologies composes the Ontology Web Language (OWL) [2].

With ontologies integrated within a KG, it is possible to employ inferential algorithms either at query time or at preprocessing time. This capability enables the derivation of new knowledge that isn't explicitly stated in the data but can be inferred based on the ontological representations.

The use of ontologies in a KG is crucial in complex domains like genomics. Here, the ability to infer new relationships and properties can lead to understanding intricate biological connections and processes.

In summary, ontologies provide the foundational structure for managing complex information systems. They not only facilitate a standardized approach to data handling and integration but also enhance the capability of systems to derive and utilize new knowledge effectively.

2.1.4 Data Federation

Data Federation is a technology that organizes data from multiple different and autonomous data sources and makes it accessible under a uniform data model, allowing for real-time data retrieval from various sources without requiring data deduplication. This approach creates a unified data access layer that abstracts the underlying technical details of each DBMS. Users can query this virtual layer using standard data querying languages, like SQL, without needing to know where the data is physically stored or in what format.

A key strength of Data Federation lies in its capacity to harmonize heterogeneous data sources, ranging from traditional relational databases to modern big data solutions and cloud services. This integration is seamless and dynamic, scaling well as soon as new sources come in, without requiring significant reconfiguration. Such agility is crucial in fields like genomics, where data formats and sources evolve rapidly alongside scientific advancements. Moreover, it minimizes the risks associated with data duplication and movement, such as data loss or corruption. It also ensures that data remains current, reflecting real-time changes without delay. This is particularly valuable for decision-making processes where up-to-date information is critical.

Different existing Data Federation systems may have different characteristics, and the choice of which system suits better certain requirements may depend on many different factors. A recent comparative study on Data Virtualization Systems [3] highlights and evaluates diverse features of many systems coming both from the academia as well as enterprise solutions, such as supported query languages, supported data sources, data security, exposed interfaces and software support.

2.1.5 Semantic Data Integration

Semantic Data Integration is an approach that aims to integrate data from a single relational source to an ontology that models a specific domain. This integration is achieved by defining mappings that establish semantic relationships among data entities.

The core idea behind semantic data integration involves establishing a global schema, represented by the ontology, which acts as a blueprint for how data from various sources is viewed and accessed. This ontology defines not just the entities and their attributes but also the relationships and constraints between these entities. The mapping between the ontology (global schema) and the relational data source is critical as it dictates how data is interpreted in a meaningful way.

The concept of Data Integration, firstly introduced without a semantic focus [4], has evolved with research advancements in this field. These advancements have led to the development of tools and mapping languages specifically designed for semantic data integration, laying the groundwork for a paradigm known as OBDA [5].

The OBDA framework, that has been precisely formalized [6], is composed of a chain of procedures that can be summarized as follows:

1. input query $q(\vec{x})$ is rewritten [7] into q' over the virtual ABox A' (i.e. the first-order logic specification of the ontology);
2. the rewritten query q' is unfolded using the *mappings* into an SQL query q^*;
3. the optimized SQL query q^* is evaluated over the data instance D (e.g. a relational DBMS).

Figure 2.1 outlines the aforementioned algorithm within a flowchart.

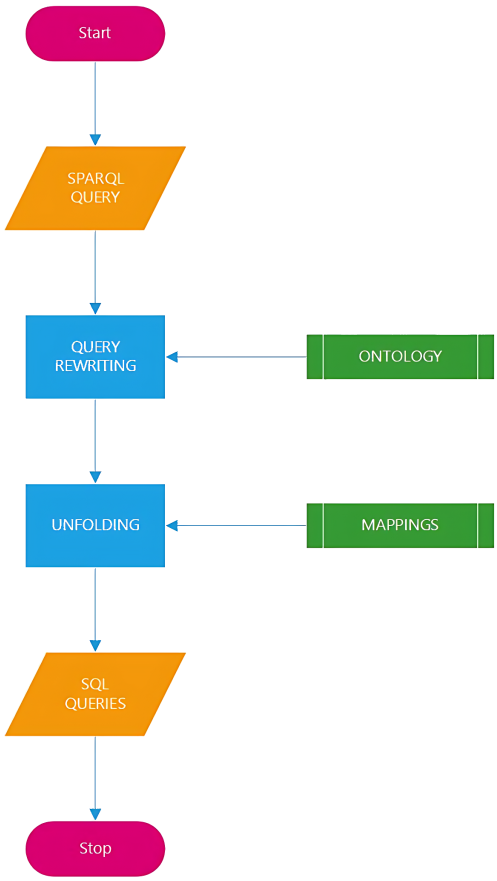

Figure 2.1: The OBDA framework

2.2 Technical Background

2.2.1 SPARQL Query Language

SPARQL, developed by the W3C, is the definitive standard for querying and managing data stored in RDF format.

As a key technology in semantic web applications, SPARQL enables sophisticated querying of KG, offering a query syntax that often resembles natural language. This user-friendly syntax facilitates intuitive data exploration and manipulation, differently from traditional relational databases that require joining tables to establish relationships.

For instance, as shown in Code 2.1, if a researcher wishes to find all genetic markers associated with a particular trait, a SPARQL query might directly reflect the question, "Which patients are associated with genetic markers X?" This parallels natural language questioning closely, making SPARQL particularly suited for domains where complex relationships must be understood and explored, such as genomics and biomedical research.

```
PREFIX bto: <https://w3id.org/brainteaser/ontology/schema/>
PREFIX BTO_resource: <https://w3id.org/brainteaser/ontology/resource/>

SELECT ?patient ?FUS ?FUS_mutation
WHERE {
   ?p a bto:Patient;
    bto:enrolledIn ?ctp.

    ?ctp bto:inClinicalTrial BTO_resource:BrainteaserALSTurin.

    OPTIONAL{
        ?p bto:tested ?FUStest.
      ?FUStest bto:hasMutation ?FUS;
                 bto:onGene NCIT:C91852.
        OPTIONAL{
            ?FUStest bto:mutationType ?FUS_mutation.
        }
    }

    BIND(SUBSTR((STR(?p)), 48) AS ?patient)
}
```

Code 2.1: Example of a SPARQL query for genetic markers data.

2.2.2 Data Federation Systems

As discussed in section 2.1.4, Data Federation Systems are sophisticated software and thus evaluable under many aspects. In this background analysis we will focus on four main aspects, considering their importance as the Data Federation System will be part of a broader framework. In particular, by the aforementioned comparison, we will present three most prominent systems considering:

- software support & documentation;
- scalability;
- logging capabilities;

- performances.

These features are also briefly summarized in Tab. 2.1.

Denodo

Denodo[2] is an enterprise virtualization platform that serves as a data federation system, integrating data from diverse sources to provide a unified view without requiring physical data deduplication. It allows for real-time access to structured and unstructured data from various sources including relational, column and No-SQL databases, web Application Program Interface (API), and flat files.

Denodo provides robust security features, including hashing, encrypting functions and user privileges, ensuring that sensitive data is protected according to compliance standards.

Moreover, it utilizes advanced query optimization techniques, such as caching and query rewriting, to enhance performance. These optimizations ensure efficient data retrieval, reducing latency and improving the overall speed of data access across the federated sources.

Although it is highly scalable, capable of accommodating new data sources, and it is provided also with a custom source wrapper template for unsupported data sources, being it an enterprise solution allows for maximum five connections, unless a premium plan is signed.

Teiid

Teiid[3] is an open-source Java component developed by Red Hat[4] that provides integrated access to multiple data sources through a single uniform API. Rather than a DBMS, Teiid is more a query engine for integrating data from multiple sources, accessible both through API and Java Database Connectivity (JDBC)/Open Database Connectivity (ODBC) interfaces.

It comes in different shapes: there are Teiid implementations as an Eclipse plugin as well as deployable packages on web containers such as Wildfly and OpenShift.

[2]https://denodo.com/en
[3]https://teiid.io/
[4]https://www.redhat.com/

One of the main issues with Teiid is poor documentation when it is not deployed alongside enterprise solutions (like OpenShift). Moreover, no major releases have been published since four years, and many open issues on the official GitHub repository[5] are not being addressed.

Dremio

Similarly to Denodo, Dremio[6] is a virtualization platform that serves as a data federation system. Although it is developed within an enterprise context, the standard version, comprehensive of most of the Dremio features, it is Free and Open Source (FOSS) under the Apache 2.0 license, combining typical enterprise software robustness with a strong community active on continuous maintenance.

Even if it is possible to use Dremio as a standalone instance (e.g. on a server), it is by design a distributed system and thus it can run on clusters up to more than 1000 nodes. In case of standalone configurations, in linux server environments it is possible to install it using packet managers, but the easiest way is to use the Docker image it comes with.

Dremio makes use of Apache Arrow to enhance processing speeds and reduce latency. Moreover, it optimizes query performance through its advanced query planner and execution engine, which can push down operations to the data source level, minimizing data movement and speeding up response times.

It provides comprehensive security features that include encryption, access controls, and data masking to ensure privacy. It also maintains detailed logs of all queries, which are crucial for compliance purposes in many fields such as the clinical domain, where patient medical data is managed alongside their personal information.

As in Denodo, but under an open-source perspective, it is possible to build custom connectors for unsupported data sources and Dremio. This is realizable through Advanced Relational Pushdown (ARP) connectors: a public repository[7] provides a Maven template, that is customizable for each data source for which a JDBC driver is available.

In conclusion, Dremio offers an open-source virtualization system with the

[5] https://github.com/teiid/teiid
[6] https://www.dremio.com/
[7] https://github.com/dremio-hub/dremio-sqllite-connector

typical robustness of enterprise software; it is highly scalable both in the sense of computation distribution over clusters and in the types of supported sources, and it has a comprehensive logging system that suits well for tracking data flow.

	Support	Free and Open Source	Well Documented	Scalability	Solid Logging Capabilities
Denodo	✓		✓	✓	✓
Teiid		✓			
Dremio	✓	✓	✓	✓	✓

Table 2.1: Comparative of Data Federation Systems

2.2.3 OBDA Systems

OBDA, as shown in Fig. 2.2, allows to seamlessly integrate a relational source within an ontology, so to add a semantic layer. Research on this specific topic has been performed not only by the academy, but also from Research & Development branches of big companies, considering the impact on the industry: a preliminary analysis [8] by Siemens[8] that lead subsequently to a custom platform, analyzed existing systems. Table 2.2 summarizes the content of this analysis.

System	Characteristics
Optique (Siemens)	Supports ontology reasoning; supports both static and streaming relational DBMS
Ontop	Supports ontology reasoning
Mastro	Supports ontology reasoning
morph-RDB	Supports ontology reasoning
D2RQ	Does not support ontology reasoning
OntoQF	Does not support ontology reasoning
Virtuoso	Does not support ontology reasoning
Spyder	Does not support ontology reasoning
Ultrawrap	Does not support ontology reasoning

Table 2.2: Comparative of OBDA Systems

Given that ontology reasoning is one of the most important requirements in the OBDA approach (without this, using this paradigm would allow just to run

[8]https://www.siemens.com/

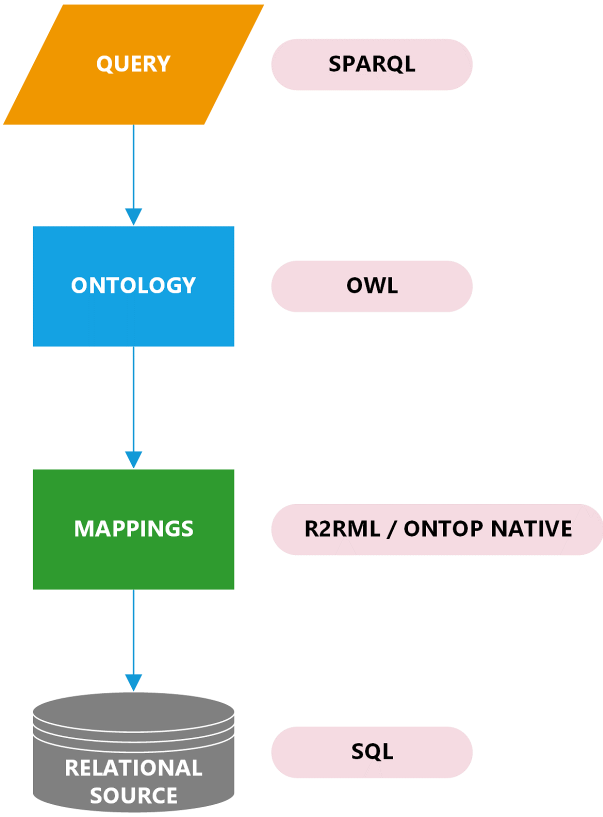

Figure 2.2: The OBDA framework

queries in SPARQL language, without any other real advantage), and considering that the Optique system is an enterprise, on-premise and closed source solution, more in-depth comparisons [9] studied Ontop and Mastro. Their comparison can be summarized as follows.

Mastro

By our knowledge, Mastro [10] was the first OBDA tool, developed at La Sapienza University of Rome. It supports a subset of SPARQL queries and integrates a custom XML-based mapping syntax. Mastro's mappings were not initially compliant with R2RML standards, which was a strong limitation in standard OBDA environments. The tool used a complex system of view predicate mappings, which may affect its adaptability to standardized environments. Although recent updates have integrated R2RML support, the system's architecture still lacks certain optimizations due to its design constraints. For instance, Mastro cannot perform advanced semantic query optimizations because it does not support detailed data constraints in its mappings. This limitation could lead to less efficient query processing and increased execution times, especially with complex queries involving multiple joins or extensive data operations. Moreover, Mastro is less efficient handling of datatype operations and IRI constructions within SQL, leading to potential slowdowns in query processing.

Ontop

Ontop [11] is an open-source and well maintained OBDA framework developed at the Free University of Bolzano. It is compliant with W3C standards, including R2RML for ontology mapping, OWL for ontology representation, and SPARQL for querying. Ontop is designed for high-performance query answering over virtualized RDF graphs. Ontop provides an efficient query answering system that leverages R2RML mappings and supports comprehensive optimizations. The system uses advanced query rewriting techniques, which are highly effective in reducing query complexity and execution time. Ontop's strengths are particularly evident in its handling of complex SPARQL queries, where it efficiently translates these into SQL, utilizing T-mappings and database integrity constraints for optimization. Regardless of its adherence to standards, Ontop comprises also a custom mapping language (Ontop Native Language).

The comparison between the two systems has been performed in two different scenarios. In general, Mastro shown faster responses in scenarios requiring extensive in terms of timings, while Ontop performs better in scenarios where a considerable number of mappings is involved for unfolding a certain query. This means that a choice on which system fits better depends on how constraining are timings in the query processing phase and on how many mappings have to be unfolded on average, that strictly depends on the heterogeneity of the underlying relational source.

2.2.4 Triple Stores

Triple stores are database management systems specifically designed to store and retrieve triples through semantic queries. Triple stores typically ingests RDF files that contains IRI's in the usual format of subject, predicate, object. These systems are optimized for storing vast amounts of triples and efficiently handling complex queries that involve traversing relationships across a network of data. Triple stores support SPARQL, enabling semantic queries that are essential for applications in data integration, knowledge management, and semantic web projects. Many comparisons, especially in the biomedical field [12], among different triple stores systems analyze their performances under different point of view such as the volume of ingested data, the implementation of an ontology within the KG, and the language used to represent it (RDF, RDFS, OWL, etc.). For the purposes of this project, where materialized triples are not used, GraphDB has been selected due to its integration with Ontop. This integration allows GraphDB to support virtual graph functionalities, meaning it can perform SPARQL queries over non-RDF relational data by translating these queries into SQL through Ontop's engine. This scenario eliminates any particular requirements from the triple store system regarding the query processing phase, considering that the task will be accomplished by the underlying OBDA system.

GraphDB

GraphDB is a robust, enterprise-ready triple store that excels in handling large volumes of data and complex queries. Developed by Ontotext[9], GraphDB is designed to facilitate efficient data integration, knowledge discovery, and semantic analytics. It uses RDF for data representation and SPARQL for querying, supporting seamless transitions between different data formats and query languages. GraphDB's integration with Ontop for virtual RDF stores and its robust performance metrics make it an ideal choice for projects that require dynamic semantic data integration without the overhead of materialized triples. Its capabilities ensure that it is not only a powerful tool for RDF data management but also a flexible solution for broader data integration challenges in semantic environments.

[9]https://www.ontotext.com/

3

Context of the Work

Handling biomedical data entails considerable challenges. Practitioners have to deal with multiple distinct storage systems, each using different data models, that may be subject to evolution over time. This introduces substantial heterogeneity, with the same data domain represented possibly through various models, such as relational [13], hierarchical [14], or graph based. Graph models are particularly prevalent in biomedical data [15] [16], as their structure effectively represents the intricate and interconnected relationships characteristic of biological systems. These systems typically operate under various database management systems (DBMSs). Further, the data is described using diverse metadata schemas, increasing the heterogeneity. This fragmentation complicates the ability to query these systems together and infer new knowledge, unless experts manually integrate them, by defining a unified data model, achieving consensus among data stakeholders, manually matching existing data to this new model, migrating data accordingly, and then modifying applications to adapt to changes in the used query language. These steps are both time-consuming and expensive. At the European level, there exist many contexts where these challenges are relevant. Therefore, in this context, we can actively pursue our objectives, grappling with collections of unstructured and heterogeneous biomedical data. In particular, the Department of Information Engineering at the University of Padova leads the EU project HEREDITARY[1], collaborating with three medical

[1]https://hereditary-project.eu/

centers that manage heterogeneous multimodal clinical and genomic data requiring integration. The HEREDITARY project emphasizes the critical need for a capable and efficient system that can seamlessly integrate diverse biomedical datasets. By setting up an effective federated architecture, we aim to automate the querying process across various data models, cutting the cost of performing analytics. This approach will facilitate a more detailed analysis of the clinical and genomic data from the three medical centers participating in HEREDITARY.

In this chapter, we will discuss the overall structure of the HEREDITARY project, and we will discuss the structure of available clinical and genomics data. By this discussion we will highlight which features are of interest for a federated data analytics platform that has to manage efficiently these sources of data.

3.1 The HEREDITARY Project

The HEREDITARY project represents a groundbreaking initiative funded by the European Union, aimed at transforming our understanding of brain diseases through an innovative convergence of multimodal heterogeneous data, such as genomic data, bioimages, clinical records and environmental data. This ambitious project is structured to exploit the power of big data and advanced analytics to tackle some of the most pressing challenges in modern healthcare. At the heart of the HEREDITARY project there is its commitment to change the paradigm of healthcare by integrating diverse data streams to unlock previously inaccessible insights. This integration involves sophisticated data linkage across various modalities. By leveraging this integrated data framework, HEREDITARY seeks to revolutionize the fields of disease detection, treatment response, and preventive healthcare.

Privacy Compliance

Security and privacy compliance stands at the fundamentals of the HEREDITARY project, particularly in handling sensitive health data. The project employs state-of-the-art secure supercomputing facilities and federated learning techniques. These methodologies ensure that while the data is extensively analyzed to produce critical health insights, it remains within the confines of local data governance laws, such as the General Data Protection Regulation (GDPR). This system not only protects individual privacy but also facilitates a collaborative environment where data does not cross organizational boundaries unnecessarily.

Technological Components

HEREDITARY is pioneering the use of advanced learning models, including deep neural networks and self-supervised learning algorithms, to analyze large heterogeneous datasets. The project's focus on semantic-aware learning methodologies, facilitated by the OBDA paradigm, allows for the seamless integration of disparate data types. These integrated datasets are utilized to perform complex queries and enhance predictive analytics capabilities, which are crucial for identifying new disease patterns and treatment possibilities.

User-Friendly Analytical Platforms

To maximize the impact of its research findings, HEREDITARY is developing interactive data-driven solutions that simplify the exploration and analysis of complex health data. The project includes the creation of a visual analytics platform that combines advanced data visualization tools with user-friendly interfaces. This initiative not only aids researchers and clinicians in hypothesis testing and decision-making but also enhances public understanding and trust in health data use.

Collaborativity and Multidisciplinarity

HEREDITARY's structure is inherently collaborative, involving multiple leading European universities and research institutions. Each participant brings unique expertise and resources, facilitating a comprehensive approach to tackling healthcare challenges. The project encourages continuous interaction among all partners, ensuring that insights and methodologies are shared and refined across different disciplines and sectors.

Focus on Neurodegenerative and Gut-Related Disorders Interplay

One of the primary research focuses of the HEREDITARY project is the investigation of neurodegenerative and gut microbiome-related disorders. By examining the gut-brain axis and its impact on diseases such as Parkinson's and

Alzheimer's, the project aims to uncover novel therapeutic and diagnostic approaches that could lead to more personalized medicine practices. These efforts are supported by sophisticated data models that predict disease progression and treatment responses, tailoring healthcare strategies to individual patient needs.

Future Perspectives

HEREDITARY is set to continue its influence beyond the project's timeline by developing sustainable strategies for health data utilization. The project's outputs are expected to inform future policy, enhance clinical practices, and continue to provide valuable insights into complex health conditions. By establishing a robust framework for data integration and analysis, HEREDITARY positions itself as a beacon for future initiatives in the realm of data-driven healthcare innovation.

HEREDITARY Project Structure

The duration of the HEREDITARY project is four years. The structural organization of the HEREDITARY project is meticulously designed to ensure efficient project management, seamless collaboration across disciplines, and rigorous pursuit of its scientific objectives. The project is divided into nine distinct Work Package (WP), each tasked with specific aspects of the project's implementation and goals.

3.2 Data Description

In the broader framework of the HEREDITARY project, the integration and analysis of detailed clinical and genomics data play a crucial role. This section delineates the structure and types of data used within the project, highlighting how this information contributes to the overarching goals of enhancing healthcare through advanced data analytics.

3.2.1 Clinical Data

Clinical data in the HEREDITARY project originates from three distinct medical centers located in Madrid, Lisbon, and Turin. Despite the geographical diversity, the data adheres to a unified schema, which ensures consistency and facilitates comprehensive data analysis across centers. This schema encompasses three main relational categories.

Static Variables

This relation provides a snapshot of patient demographics and health status at the time of their entry into the study. It includes a wide range of fields such as patient identifiers, dates of disease onset and diagnosis, vital status, demographic details (sex, ethnicity, height, weight), medical history (major traumas, surgical interventions, prevalent conditions), lifestyle factors (such as smoking habits and occupation), detailed clinical assessments of disease symptoms, familial history, genetic markers related to diseases such as Amyotrophic Lateral Sclerosis (ALS) (e.g., mutations in FUS, SOD1, TARDBP, C9orf72), and other relevant clinical data. These static variables are crucial for establishing baseline characteristics and stratifying patients based on demographic and clinical features.

ALS Functional Rating Scale Data

This relation captures dynamic, longitudinal data on the progression of ALS through a standardized functional rating scale. The fields include patient identifiers, assessment dates, and scores across multiple functional domains such as bulbar, motor, and respiratory functions, as well as detailed scores for individual tasks assessing the patient's daily living abilities.

Spirometry Measurements

The third relation contains spirometry test results, which are critical for assessing respiratory function in ALS patients. Fields include patient identifiers, test dates, and the relative forced vital capacity (FVC), which is a primary marker of pulmonary function decline in neurodegenerative conditions.

3.2.2 Genomics Data

The genomics component of the HEREDITARY dataset is sourced from synthetic data specifically designed to reflect the complexity of real-world genetic data while ensuring privacy and compliance with ethical standards. This synthetic data is provided by the CINECA project[2], which models it to mirror the variability found in populations typically studied in biomedical research. The data retains the statistical properties of original datasets without compromising individual privacy, making it ideal for research purposes.

Genotypic Data

This dataset contains information such as dataset identifiers, chromosomal positions, variant identifiers, reference and alternate alleles, and variant types. These details are crucial for identifying genetic variations that may be related with disease phenotypes observed in the clinical data. The synthetic genetic data is derived from the 1000 Genomes project, ensuring a robust representation of genetic diversity.

Phenotypic Data

Complementing the genotypic data, the phenotypic component includes variables that describe patient characteristics and health status, such as physical activity levels, alcohol frequency, smoking habits, job type, living arrangements, vital statistics like blood pressure and heart rate, and clinical diagnoses and prescriptions. The phenotypic data is modeled using the DataSynthesizer[3] tool, which is designed for privacy-preserving dataset generation.

[2]https://www.cineca-project.eu/cineca-synthetic-datasets
[3]https://github.com/DataResponsibly/DataSynthesizer

4

Related Works

Performing federated analytics tasks requires a robust architecture composed by different layers: a federation layer that allows for multiple streaming connections with diverse and heterogeneous sources (relational, no-SQL and columnar DBMSs); a virtualization layer that exposes ”virtual” relational views of non-materialized data; an integration, ontology-based layer, so to add a semantic layer to the virtualized relational views, given the importance of exploring complex relationships among genomics data.

This approach has been formalized under the concept of Ontology-Based Data Federation (OBDF) [17], and it is represented in Fig. 4.1. As far as we know, no existing off-the-shelf system implementing this framework has ever been released: who intends to apply this architecture design have to manually install different components choosing among diverse competitors, and combining them together, dealing with possible underlying platform incompatibilities. Moreover, having no off-the-shelf solutions implies having no solutions specifically tailored and optimized for dealing with genomics data.

Nonetheless, similar design proposal have been implemented in order to address both research questions as well as enterprise requirements. In this section, we will briefly discuss two solutions, one for each environment, analyzing them in details, highlighting strengths and weaknesses.

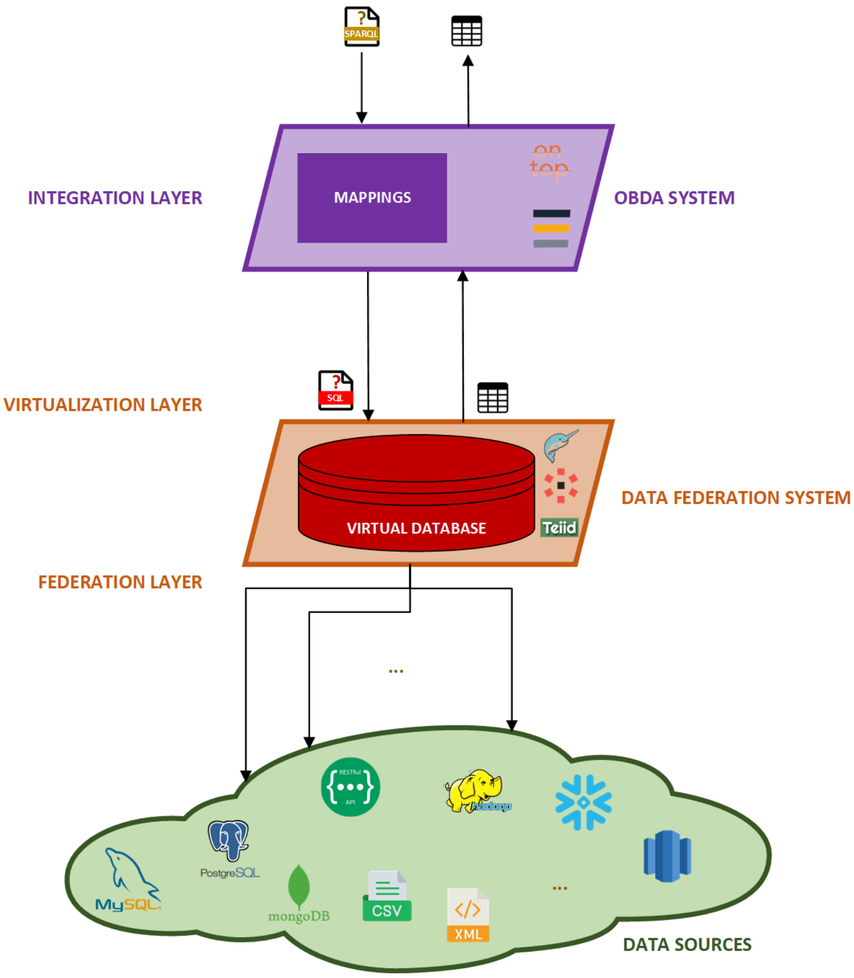

Figure 4.1: OBDF approach

4.1 BigDAWG

Big Data Analytics Working Group (BigDAWG) [18] represents a crucial milestone in the world of polystore systems, designed to address the complexities of managing heterogeneous data environments. Developed under at the Intel Science and Technology Center for Big Data, BigDAWG's architecture realizes the paradigm "one size does not fit all" in database management systems. This polystore system is engineered to support a wide set of data models and storage engines, offering a unified platform for cross-storage-system queries, real-time analytics, and complex data visualizations.

4.1.1 Architecture Overview

BigDAWG's architecture is meticulously structured into four primary layers: the base layer, the island layer, the main BigDAWG layer, and the application layer. Each of these layers serves a distinct function, creating a cohesive system capable of managing diverse data stores and facilitating seamless user interaction through applications.

Base Layer

The base layer of BigDAWG consists of the physical data stores, which include a variety of storage engines such as relational databases (e.g., PostgreSQL), array databases (e.g., SciDB), key-value stores (e.g., Apache Accumulo), and streaming databases (e.g., S-Store). Each of these storage systems is optimized for specific types of data and query workloads, adhering to the principle that no single storage engine is best suited for all types of data. This layer is responsible for the actual storage and retrieval of data, leveraging the strengths of each specialized engine to handle the respective data types effectively.

Island Layer

Above the base layer is the island layer, a critical abstraction within BigDAWG's architecture. An "island" in BigDAWG represents a domain-specific environment tailored to a particular data model and query language. For example, the relational island interfaces with traditional RDBMSs like PostgreSQL, the array island interacts with SciDB, and the text island is linked to systems like Accumulo. Each island encapsulates the syntax and semantics of its underlying data model, thus preserving the rich functionalities and optimizations offered by the native storage engines. The island layer also facilitates the casting of data between different islands, enabling queries that span multiple data models. The ability to perform these cross-island operations is central to BigDAWG's polystore capability, ensuring that users can execute queries that utilize the most appropriate data model for each part of their analysis.

Main BigDAWG Layer

The main BigDAWG layer orchestrates the interaction between the various islands and the underlying data stores. This layer includes components such as the query planner, optimizer, and the casting engine, which collectively manage the distribution of queries across the appropriate islands. The casting engine is particularly noteworthy, as it enables the movement of data between different storage systems, translating data formats and query languages as needed. For instance, data stored in an array database might be cast into a relational format to allow for SQL-based analytics, and then cast back into an array format for further processing. This layer is also responsible for maintaining metadata about the data stores and islands, including information about data distribution, query capabilities, and performance characteristics of each storage engine. This metadata is crucial for optimizing query execution, as it allows BigDAWG to intelligently route queries to the most suitable storage engine based on the query's requirements and the data's location.

Application Layer

At the top of the BigDAWG architecture is the application layer, which provides user-facing tools and interfaces for interacting with the system. This layer includes APIs for query submission, tools for data visualization, and interfaces for real-time data monitoring and complex analytics. BigDAWG supports a variety of applications, from simple SQL-based data retrievals to complex analytics involving machine learning and real-time data processing. The application layer abstracts the underlying complexity of the polystore system, offering a unified interface that allows users to focus on their data analysis tasks without needing to understand the intricacies of the underlying data storage systems.

4.1.2 Operational Principles and Innovations

BigDAWG's operational principles are grounded in the concept of federating multiple, heterogeneous data stores while preserving the specialized capabilities of each storage engine. This approach allows BigDAWG to offer a robust platform for executing complex queries across different data models, something that traditional database systems struggle to achieve.

Islands and Cross-Island Querying

The island-based architecture is one of BigDAWG's most significant innovations. By organizing data stores into islands, BigDAWG allows users to interact with data using the most appropriate data model and query language. This capability is particularly important in environments where data is stored in multiple formats, each optimized for a different type of analysis. For example, a user might store time-series data in an array database like SciDB while storing text data in a key-value store like Accumulo. BigDAWG's island architecture allows the user to query both datasets in a single operation, casting the results into a common format for further analysis. The cross-island query capability is facilitated by the casting engine, which can translate data formats and query languages on-the-fly. This ensures that data can be moved between islands without loss of fidelity or functionality. For instance, a query might start in the relational island to extract patient metadata from PostgreSQL, then cast the data into the array island to perform complex mathematical operations on waveform data stored in SciDB, and finally cast the results back into the relational island for final aggregation and reporting.

Performance Optimization

Another key feature of BigDAWG is its ability to optimize query performance [19] by leveraging the strengths of the underlying storage engines. The system's query planner uses metadata about the data stores and islands to determine the most efficient way to execute a query. For example, if a query involves a large text search, BigDAWG might choose to cast the data into the text island, where the key-value store (e.g., Accumulo) can handle the search more efficiently than a relational database. Similarly, if the query involves complex mathematical operations on large datasets, the planner might route the query to the array island, where SciDB's array-based storage engine can perform the operations more efficiently. BigDAWG also supports real-time data processing, making it suitable for applications that require immediate responses, such as monitoring systems in healthcare settings. The system's streaming capabilities, provided by engines like S-Store, allow it to handle high-velocity data streams while maintaining transactional consistency. This is particularly important in environments like medical data monitoring, where delays or errors in data processing could have serious consequences.

Use Cases and Practical Implementations

BigDAWG has been demonstrated in several complex real-world scenarios [20] [21], highlighting its versatility and robustness. One of the most prominent use cases is the management of the MIMIC II dataset, a large-scale medical database containing various types of data, including time-series waveform data, patient metadata, and clinical notes.

MIMIC II Dataset Management

The MIMIC II dataset is a prime example of a heterogeneous data environment, where different types of data require different storage and processing techniques. BigDAWG effectively manages this dataset by distributing the data across multiple storage engines, each optimized for a specific type of data. For instance, patient metadata is stored in PostgreSQL (relational island), waveform data is stored in SciDB (array island), and clinical notes are stored in Accumulo (text island). BigDAWG's ability to perform cross-island queries allows researchers to analyze the data in a holistic manner, combining insights from the different data types into a single analysis. In practical terms, this means that a query might involve extracting patient metadata from PostgreSQL, performing a Fourier transform on the waveform data in SciDB, and searching for relevant clinical notes in Accumulo. BigDAWG handles the complex task of routing the query through the appropriate islands, casting the data as needed, and optimizing the execution to minimize processing time and resource usage. This capability is particularly valuable in medical research, where researchers need to quickly and accurately analyze large datasets to draw meaningful conclusions.

Challenges and Future Directions

Despite its advanced architecture and capabilities, BigDAWG is not without challenges. One of the main issues is its prototype nature; it is not available as an off-the-shelf solution and requires significant expertise to install, configure, and extend. This limits its accessibility to organizations with the necessary technical skills and resources. Furthermore, the development of new islands to support additional data models and storage engines requires a deep understanding of both the BigDAWG system and the underlying technologies. Another challenge is the limited literature and empirical studies on BigDAWG's deployment in real-world scenarios. While the system has been successfully demonstrated in academic and research environments, there is still a lack of comprehensive studies on its performance and reliability in operational settings. This raises questions about its scalability and robustness when applied to large-scale, production environments.

4.1.3 Conclusion and Future Work

In conclusion, BigDAWG represents a significant advancement in the field of polystore systems and heterogeneous data management. Its multi-layered architecture, island-based abstraction, and cross-island querying capabilities offer a powerful solution for managing and analyzing complex datasets. However, its practical application is currently limited by the need for specialized knowledge and the lack of extensive real-world testing. Future research could focus on several key areas to reduce the barriers to adoption and expand BigDAWG's applicability. These include the development of more user-friendly installation and configuration tools, the creation of additional islands to support a wider range of data models, and the conducting of large-scale empirical studies to validate the system's performance and reliability in operational environments. Additionally, efforts could be made to improve the system's documentation and provide more comprehensive training resources to help users understand and leverage its full capabilities.

4.2 Optique

The Optique system, born from the EU-funded Seventh Framework Programme [22], represents a significant evolution in the field of OBDA systems tailored for industrial applications [23]. Unlike typical OBDA systems which focus predominantly on static data, Optique extends its capabilities to effectively handle both streaming and static data, providing a robust platform designed for real-time and historical data integration and accessibility using semantic technologies. The project involved multiple prestigious institutions and industry giants like Siemens, underscoring a collaborative effort aimed at enhancing data accessibility and integration across varied sectors. Optique documentation relatively to its development stage while project execution is publicly available[1].

[1] https://mvnrepository.com/artifact/eu.optique-project

4.2.1 Architectural Overview

Optique is designed using a layered architecture approach, ensuring scalability and extensibility [24]. At its core, the architecture comprises several distinct layers, each responsible for a specific aspect of the system's functionality:

- **Presentation Layer** : This layer includes user interfaces for both end-users and IT experts, facilitating interactions such as query formulation, system configuration, and ontology management. Key interfaces in this layer are developed to ensure user-friendliness and accessibility, allowing users to interact with the system without needing in-depth technical knowledge of the underlying semantic technologies.
- **Application Layer**: This layer represents the backbone of the Optique system, hosting critical components including the Query Formulation, Ontology and Mapping Management, and Query Answering components. Each component is designed to handle specific functionalities, from managing ontology mappings to processing complex queries over diverse data sources.
- **Data and Resource Layer**: This foundational layer manages the data sources accessed by the system, which may include relational databases, triple stores, and data streams. It also encompasses the infrastructure required to support the extensive computational needs of the system, leveraging cloud technologies to enhance scalability and performance.

4.2.2 Core Components and Functionalities

The Optique system incorporates several components that improve its performance and usability:

STARQL Query Language

At the heart of Optique's functionality is STARQL, an advanced query language that enables complex querying over both streaming and static data. STARQL supports seamless integration of streaming data queries, allowing for real-time data processing and analytics.

ExaStream Backend System

ExaStream represents the backend processing component of Optique, optimized for handling high-velocity data streams with minimal latency. Its design focuses on efficient data ingestion and query processing, ensuring that the system can manage the continuous influx of data from industrial sensors and other real-time data sources.

OptiqueVQS: Visual Query System

The Optique Visual Query System (OptiqueVQS) enhances the accessibility of the system by enabling users to construct queries visually [25]. This component is particularly useful for users without experience in query languages, as it provides an intuitive interface for building and executing queries, and thus reducing dependence on IT departments and streamlining data access processes.

4.2.3 Impact and Evaluation

The deployment of Optique has shown a scalable approach to data access, significantly reducing the reliance on IT intermediation and promoting efficient data utilization. However, despite its advanced capabilities, the integration of Optique within the Siemens ecosystem has led to its status as a proprietary system. This exclusivity could potentially restrict broader adoption and external validation, posing challenges in comparative analyses against other OBDA systems in diverse real-world settings.

5

System Architecture

As discussed in Chapter 4, there exist no off-the-shelf solution implementing the OBDF paradigm. This implies that for the specific task of dealing with clinical and genomics data, within the HEREDITARY context, it is necessary to design a system architecture, choosing components in such a way that the whole architecture results in being solid, privacy-oriented (with strong logging capabilities), and being capable to manage these kind of data. This chapter will firstly discuss the overall proposed architecture, that retraces the one presented in previous chapters. Subsequently, each layer will be presented in details, outlining how each components have been configured, reporting eventual code snippets. In order to better describe the source heterogeneity of data that may occur in contexts such as clinical and genomics, available relational data have been distributed among different sources.

5.1 System Architecture Design

The proposed architecture described in Fig. 5.1 implements the OBDF framework. It adopts Ontop as its semantic data integration layer and Dremio as its data federation layer.

Ontop have been chosen because it natively supports two high level mapping languages, that gives more freedom on their optimization, without the need to appeal to low level Description Logic (DL) languages, it is open-source and well maintained by the Free University of Bolzano data integration team, it comes embedded in GraphDB that offers solid APIs, and it comes even with an em-

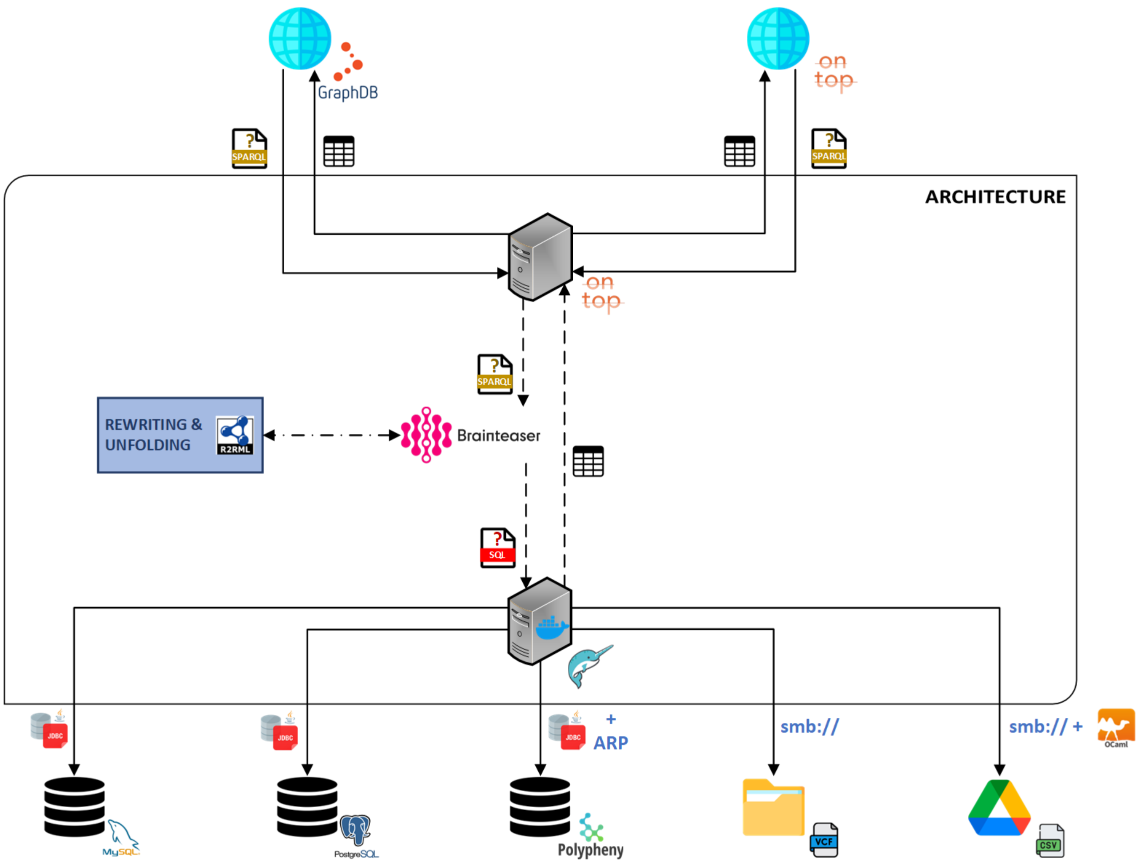

Figure 5.1: Proposed System Architecture

bedded web endpoint.

Dremio have been chosen as the virtualization and federation layer because, as discussed in the Chapter 2, it is an open-source, robust and scalable software influenced both from its enterprise nature and from a consistent community providing contributions. Moreover, within its installation methods, it is possible to set it up through a Docker image: this may set the basis for the architecture packaging, expanding the image to other software components.

The semantic integration requires an ontology well describing the domain of clinical and genomics data. Considering the reusability principle that stands at the basis of the semantic web, we identified the BRAINTEASER ontology[1] as suitable for accomplishing this task.

[1] https://brainteaser.dei.unipd.it/ontology/

5.2 Data Sources

Given the relational nature of the available data, we distributed it among five different platforms to exploit the federation capabilities of the data federation layer. The choice of the sources has been guided also by surveying commonly used ones in the biomedicine field in contexts like research and hospital diagnostic. As we will discuss, they encompasses more structured DBMSs as well as simple Comma Separated Values (CSV) files. We also included a novel DBMS system, so to investigate how new, unknown data sources can be federated as soon as they figure out.

5.2.1 MySQL

MySQL[2] is one of the most widely used relational DBMS in the world. It is a FOSS solution now distributed by Oracle[3]. It is extensively employed across a variety of applications, from small personal projects to critical enterprise environments. In our system architecture, we've chosen MySQL considering its extensive adoption, possibly also in application software used in clinics and hospitals to collect patients data. In our environment, MySQL's role is to store part of the structured clinical data presented in Chapter 3. The interfacing between MySQL and our data federation layer, Dremio, is performed through MySQL's JDBC connector. This setup allows Dremio to access and query MySQL data seamlessly. No special modifications or configurations were required to integrate MySQL with Dremio, thanks to Dremio's native support for MySQL. We simply established a new data source within Dremio by specifying the connection parameters.

5.2.2 PostgreSQL

PostgreSQL[4] stands out as a widely adopted open-source relational DBMS. It is particularly adopted within the research community due to its robustness, advanced features, and strong compliance with SQL standards. Many research institutions and academics prefer PostgreSQL for its extensive capabilities in

[2]https://www.mysql.com/it/
[3]https://www.oracle.com/it/
[4]https://www.postgresql.org/

managing complex data types and its support for sophisticated data manipulation operations. We considered PostgreSQL eligible to be a data source due to its common adoption in research contexts as a data collector. Again, PostgreSQL's role in our environment is to store another portion of the structured clinical data presented in Chapter 3. Just like with MySQL, integrating PostgreSQL with Dremio did not require any specific modifications or additional configurations.

5.2.3 Polypheny

Polypheny [26] is an open-source polystore system designed to support diverse data models, including relational, document, and graph data. It is engineered to handle mixed workloads and various query languages, making it a versatile platform suitable for dynamic data environments. In our architecture, we consider Polypheny not just as a standalone polystore but as a potential low-level federator under our main data federation layer managed by Dremio. This perspective is particularly useful because it allows us to leverage Polypheny's ability to handle multiple data models, thus enriching the flexibility and capability of our overall data management system.

Custom ARP Connector Development

Unlike the straightforward integrations with MySQL and PostgreSQL, incorporating Polypheny required a more sophisticated approach. We performed this task not only to include Polypheny as one of our data sources but also as a proof of concept about the actual possibility of developing custom ARP connectors. Dremio's ARP framework offers a powerful mechanism to extend Dremio's capability to interact with various data sources by interfacing Dremio's internal query representations into the native query language of the target data source. For Polypheny, we developed a custom ARP connector to ensure interaction between Dremio and Polypheny. The connector we developed is open-source and available for the community, which can be found at our GitHub repository [5]. These connectors rely on the target source having a JDBC driver and accepting SQL as a query language, so to have an interface to dialog with.

[5] https://github.com/mircocazzaro/dremio-polypheny-arp

Connector implementation Details

The custom ARP connector was implemented to translate SQL queries from Dremio into the query formats that Polypheny can execute directly. The ARP plugin template[6] consists of a Java Maven project, that has to be compiled, packed within a JAR file and injected, together with the target source JDBC driver, into the running Dremio instance. To customize the template, two files needs to be customize: the storage plugin configuration, which is a Java class 5.1, and the plugin ARP file, which is a YAML[7] file. The storage plugin configuration file tells Dremio what the name of the plugin should be, what connection options should be displayed in the source UI, what the name of the ARP file is, which JDBC driver to use and how to make a connection to the JDBC driver. The ARP YAML file is what is used to modify the SQL queries that are sent to the JDBC driver, allowing you to specify support for different data types and functions, as well as rewrite them if tweaks need to be made for your specific data source.

```
/* ... */

@SourceType(value = "POLYPHENY", label = "POLYPHENY")
public class PolyphenyConf extends AbstractArpConf<PolyphenyConf> {
  private static final String \ac{ARP}_FILENAME = "arp/implementation/polypheny-arp.yaml";
  private static final ArpDialect \ac{ARP}_DIALECT =
      AbstractArpConf.loadArpFile(ARP_FILENAME, (ArpDialect::new));
  private static final String DRIVER = "org.polypheny.jdbc.Driver";

  @NotBlank
  @Tag(1)
  @DisplayMetadata(label = "Polypheny host <HOST:PORT>")
  public String database;

  /* ... */

  @Tag(2)
  @DisplayMetadata(label = "Record fetch size")
  @NotMetadataImpacting
  public int fetchSize = 200;

```

[6]https://github.com/dremio-hub/dremio-sqllite-connector
[7]https://yaml.org/

```java
  @Tag(6)
  @DisplayMetadata(label = "Polypheny User Name")
  public String username = "pa";

  @Tag(7)
  @DisplayMetadata(label = "Polypheny Password")
  public String password = "";

  @Tag(4)
  @DisplayMetadata(label = "Maximum idle connections")
  @NotMetadataImpacting
  public int maxIdleConns = 8;

  @Tag(5)
  @DisplayMetadata(label = "Connection idle time (s)")
  @NotMetadataImpacting
  public int idleTimeSec = 60;

  @VisibleForTesting
  public String toJdbcConnectionString() {
    final String database = checkNotNull(this.database, "Missing database.");

    return String.format("jdbc:polypheny://%s", database);
  }


  private CloseableDataSource newDataSource() {
    return DataSources.newGenericConnectionPoolDataSource(DRIVER,
      toJdbcConnectionString(), username, password, null, DataSources.CommitMode.DRIVER_SPECIFIED_COMMIT_MODE,
            maxIdleConns, idleTimeSec);
  }
}
```

Code 5.1: The storage plugin configuration file

With respect to the storage plugin configuration Java class 5.1, being Polypheny still an embryonic project not allowing for user management, but rather having a default user ”pa” with empty password, fields are pre-filled with default values. The class specification is then interpreted at run time from Dremio, setting up a form with input fields corresponding to each @DisplayMetadata annotation. The DRIVER static and immutable variable contains the class path of the

Polypheny JDBC driver. The newDataSource() method is invoked at form submission, setting up the connection to the Polypheny instance.

5.2.4 NAS Folders

Dremio offers native support for a variety of ”relational” file types such as CSV, Excel, JavaScript Object Notation (JSON), and Virtual Contact File (VCF). These files may be physically moved within the Dremio instance, but losing any streaming capability, or by attaching a Network Attached Storage (NAS) source to it. In fact, Dremio integrates a connector to local folders, reading supported files within them. The name NAS on this typology of data source is ambiguous: the term refers to storage units usually located within the same Local Area Network (LAN) of one or more hosts accessing it, while in Dremio is used to generically refer to a folder to which it can access. This in practice means Dremio can browse local folders: thus, if NAS shared folders are mounted within the Dremio server through the Server Message Block (SMB) protocol, they can be browsed as well. In Linux environments where the standalone version of Dremio is used, its daemon must have permission to read these folders; in environments where the Docker image is employed, where a clear separation between the host file system and the Docker context occurs, folders and NAS shares must be mapped. This is obtained with Docker Volumes[8], and in particular the “Host Volume” paradigm.

```
$ docker run -v NAS_PATH/folder:/opt/dremio/folder -p 9047:9047 -p
   31010:31010 -p 45678:45678 -p 32010:32010 --name hereditary_dremio
    dremio/dremio-oss
```

Code 5.2: Docker command to run a Dremio container with a Host Volume

For the purposes of our project, we have utilized the NAS data source to store a portion of our genomics data. Specifically, we have chosen to include data in VCF files. VCF is a text file format generally used in bioinformatics for storing gene sequence variations.

[8]https://docs.docker.com/storage/volumes/

5.2.5 Google Drive

Google Drive is a widely used cloud storage service offered by Google that allows users to save files and access them from any device connected to the internet. Users can store documents, spreadsheets, and presentations, collaborate with others, and have all their work backed up safely. We chose to consider Google Drive as a data source for our system architecture primarily because of its widespread use in various contexts, including scenarios where Google Sheets are frequently adopted for data collection and management. Google Sheets, part of the Google Drive suite, is particularly popular in both academic and industrial settings for its ease of use and collaborative features. The approach to integrating Google Drive is identical to that of the NAS system, as long as Google Drive is adapted to a local host folder. This adaptation allows Dremio to interact with Google Drive as if it were interacting with local file systems, thereby simplifying access and manipulation of data stored in Google Drive. The adaptation of Google Drive to a local system is facilitated by a tool known as google-drive-ocamlfuse[9], which provides a Filesystem in Userspace (FUSE)-based file system backed by Google Drive. This tool was built using OCaml, a functional programming language known for its expressiveness, efficiency, and robustness. OCaml is utilized to handle the logical operations and data structure management, ensuring that the application runs efficiently and securely. FUSE is a software interface for Unix-like computer operating systems that lets non-privileged users create their own file systems without editing kernel code. This is used in google-drive-ocamlfuse to mount Google Drive as a file system on the user's computer.

Adaptation of Google Sheets

With the google-drive-ocamlfuse tool, Google Sheets can be adapted to local Excel files, which are natively supported by Dremio. This adaptation is crucial because it allows Dremio to perform operations on Google Sheets just as it would on local Excel files, enabling seamless data processing and integration without needing to convert or move data physically.

[9] https://github.com/astrada/google-drive-ocamlfuse

Configuration and Usage

Configuring google-drive-ocamlfuse involves setting up Google Drive as a mounted file system. Once mounted, the Google Drive folder behaves like any other directory on the local system.

```
# Installation of google-drive-ocamlfuse
$ sudo add-apt-repository ppa:alessandro-strada/ppa
$ sudo apt-get update
$ sudo apt-get install google-drive-ocamlfuse

# Authentication and mounting
$ google-drive-ocamlfuse
$ mkdir ~/google-drive
$ google-drive-ocamlfuse ~/google-drive
```

Code 5.3: google-drive-ocamlfuse Tool installation procedure

We used the Google Drive data source to store the remaining part of CSV genomics data.

5.3 Federation and Virtualization Layer

At the bottom of our system architecture we do have a software component that encompasses both the federation and virtualization functionalities. In fact, this component not only allows to seamlessly connecting to multiple diverse data sources, but also serves as a virtualizer by creating unmaterialized virtual views of the integrated data. This dual functionality is essential in our scenario for efficiently managing data access and manipulation across the disparate systems involved in our project.

5.3.1 Role of Federator

As a federator, this layer allows for extensive connectivity, facilitating interactions with various types of data sources ranging from traditional relational databases like MySQL and PostgreSQL to modern polystore systems such as Polypheny, and even cloud-based storage solutions like Google Drive. The ability to federate across these diverse sources means that data, regardless of where it is stored or in what format, can be accessed and queried as if it were located within a single, homogeneous database.

5.3.2 Role of Virtualizer

On the virtualization side, the layer enables the creation of virtual views that do not require materialization. These views treats relations exposed from underlying data sources as if they are actual tables within the same database schema: as Fig. 5.2 shows, this implies that these tables can be joined together and thus expose the most meaningful information to the upper architecture layers. Moreover, tables can be joined with pre-defined views, further enhancing the virtualization capabilities. No data is physically stored within this layer, but rather queries coming into this layer are dynamically interpreted and delegated to the source DBMSs. This approach ensures that the most current data is always presented to the user or application, without the overhead and delay associated with physical data integration. In the subsequent sections, a diagram will be included to illustrate how these virtual views are structured and managed within our system.

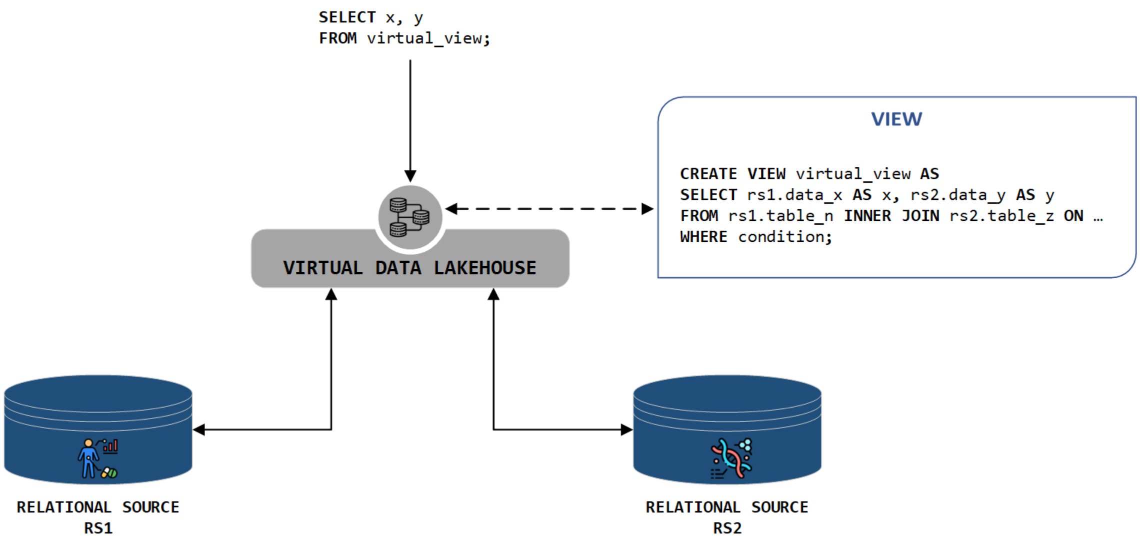

Figure 5.2: Virtual Views

5.3.3 Focus on Dremio

Web Interface

One of the main features of Dremio is its user-friendly web interface, which provides a graphical user interface (Graphical User Interface (GUI)) that simplifies the process of data management. This interface is particularly beneficial for defining and managing virtual views. Users can interact with the system through the web interface to run queries, visualize query results and configure the system settings.

JDBC Connection

Apart from its web interface, Dremio also supports JDBC connections, allowing it to integrate seamlessly with a variety of programming environments and data tools. This JDBC support is crucial for automating data processes and integrating with other applications that require programmatic access to the data federation layer. Developers can use the JDBC driver to connect directly to Dremio from their applications, enabling them to run queries programmatically and retrieve data for further processing or analysis. This capability is essential for building automated data pipelines and for applications that need to interact dynamically with the data layer.

Virtual Views Definition

Within Dremio's web interface, users can create and manage virtual views. Users can define virtual views by writing standard SQL queries that join, filter, or transform the data across these sources. For instance, a researcher might join genomic data stored in a NAS system with patient data from a MySQL database to perform analytics concurrently between genetic markers and health outcomes. In our architecture we defined different virtual views, with the intent of aligning with the ontology structure. With respect to the clinical data coming from the three medical centers, the views are limited to the union of the relations composing the dataset, given that all three of these shares the same identical schema. Only on the Amyotrophic Lateral Sclerosis view, given that data from the Madrid dataset was missing some information, we filled some of its column with null values. Views defined within Dremio are presented in code 5.4. As described before, data coming from Turin refers to a MySQL data store, data from Lisbon refers to a PostgreSQL data store, and data from Madrid is stored within a Polypheny instance.

```
# Amyotrophic Lateral Sclerosis View
SELECT * FROM LISBON.public.lisbon_alsfrs UNION SELECT * FROM TURIN
  .alsfrs_turin.Turin_alsfrs UNION SELECT MADRID.madrid_alsfrs.
  patient, MADRID.madrid_alsfrs."date", MADRID.madrid_alsfrs.tot,
  NULL AS bulbar, NULL AS motor, NULL AS respiratory, NULL AS q1,
```

```
  NULL AS q2, NULL AS q3, NULL AS q4, NULL AS q5, NULL AS q6, NULL
  AS q7, NULL AS q8, NULL AS q9, NULL AS q10, NULL AS q11, NULL AS
  q12 FROM MADRID.madrid_alsfrs;

# Spirometry Forced Vital Capacity View
SELECT * FROM LISBON.public.lisbon_spiro UNION SELECT * FROM MADRID
  .madrid_alsfrs.Madrid_spiro UNION SELECT * FROM TURIN.alsfrs_turin
  .Turin_spiro;

# Patient Static Data
SELECT * FROM LISBON.public.lisbon_static_vars UNION SELECT * FROM
  MADRID.madrid_alsfrs.Madrid_static_vars UNION SELECT * FROM TURIN.
  alsfrs_turin.Turin_static_vars;
```

Code 5.4: Virtual views over clinical data

5.4 Ontology Layer: BRAINTEASER Ontology

In our system architecture, the ontology layer is essential in modeling and managing the complex relationships between genomics data interlaced with clinical values and patients' medical records. Through an extensive review of the literature and existing resources, we identified a suitable ontology that effectively encapsulates these intricate data relationships: the BRAINTEASER Ontology [27].

5.4.1 BRAINTEASER Ontology Overview

Represented in Fig. 5.3, the BRAINTEASER Ontology is specifically designed to model the multifaceted context of genomics and clinical data. This ontology provides a structured framework that facilitates the integration of diverse data types, ranging from detailed genomic sequences to comprehensive patient medical records. By adopting this ontology within our system, we can link data across these varied domains effectively, exploiting all the potentials discussed in the background analysis, and enhancing our ability to conduct meaningful analytical analyses that require a deep understanding of both genetic and clinical factors. The ontology is part of the broader BRAINTEASER project, which aims to address complex biomedical challenges through innovative data integration techniques and advanced computational models. More information about the

BRAINTEASER project and its initiatives can be found on the official website[10].

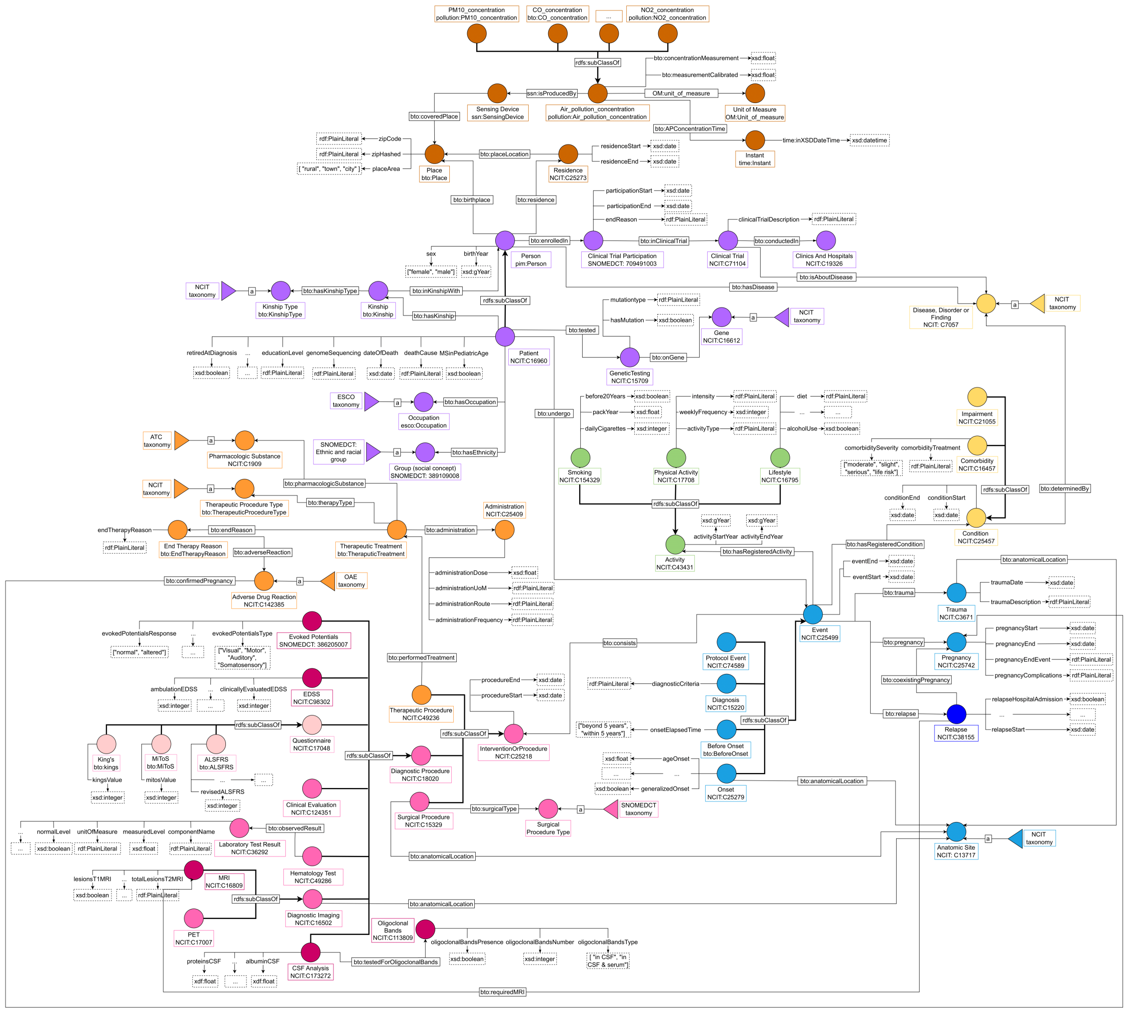

Figure 5.3: The BRAINTEASER Ontology

5.4.2 Ontology Features and Capabilities

The BRAINTEASER Ontology models meticulously various aspects of patient health data, including genetic markers, clinical symptoms, diagnostic test results, and treatment responses. This comprehensive modeling approach ensures that all relevant data dimensions are captured and can be queried effectively. One of the key strengths of the BRAINTEASER Ontology is its focus on

[10] https://brainteaser.dei.unipd.it/ontology/

interoperability. It is built to integrate seamlessly with existing medical and genomic data standards, facilitating easy data exchange and compatibility with other health information systems. As new discoveries are made and healthcare practices evolve, the ontology is structured to be scalable and flexible. It can accommodate additional data types and relationships, supporting the ongoing growth and diversification of biomedical data.

5.4.3 Ontology Implementation

In our system, the BRAINTEASER Ontology acts as the vocabulary for designing semantic queries over data. By mapping our virtual data model with this ontology, we ensure that data from disparate sources can be integrated coherently and that complex queries involves all data coming from the heterogeneous data sources.

5.5 OBDA Layer

In our architecture, The component realizing the OBDA paradigm, that aims to set up a Virtual Knowledge Graph (VKG), is Ontop. Ontop comes in different shapes: different tools are provided for accomplishing different tasks. In this section we will discuss the different Ontop tools that have been employed in our project alongside their actual role, with also code examples.

5.5.1 Ontop Mapping

Protégé is a well-known tool for ontology modelling, developed and maintained by the Stanford University. Apart from its standard capabilities, it is open to custom plugins that can be installed on demand, that expand its features. To develop ontology mappings between an ontology and a single relational data source, as in the OBDA paradigm is defined, the Ontop Mappings plugin for Protégé have been realized. Its utilization process is defined as follows:

- The ontology that has to be mapped has to be opened with Protégé;
- The Ontop Mappings plugin has three tabs: the first one is about setting up the JDBC connection with the relational source (i.e. Dremio). The JDBC driver has to be manually loaded within the Protégé "connectors" folder;
- The second tab is about defining Ontop properties. For the mapping task, no specific property needs to be defined;

- The third tab is the mappings editor. Here, there are two different input fields: one asks for a portion of the graph, in Turtle notation, with autocomplete features; the second one is about the SQL query against the relational sources. Rows coming from the relational sources are directly mapped by including fields name within curly brackets in the Turtle syntax. Note that if there are mistakes in the Turtle notation (e.g. the subgraph doesn't match with the ontology), the mapping can't be added.
- After defining all mappings, they can be validated by clicking on "Validate". SQL queries are executed and if they are wrong, or field names doesn't coincide with the one defined within curly brackets, an exception is raised;
- By saving the Protégé project, an .obda file is created in the ontology folder, containing mappings definition.

For shortness, a portion of the mappings implementation between the virtual relational schema exposed by Dremio and the BRAINTEASER Ontology is shown in Code 5.5.

```
[MappingDeclaration] @collection [[
mappingId MAPID-5b34961b80264140bb779dbd296424ac
target    bto:Patient{patient} a bto:Patient .
source    SELECT patient FROM "@mirco.cazzaro"."STATIC_VARS";

mappingId MAPID-f24debb89dc84f73a47acdf763ec0b4f
target    bto:Patient{patient} bto:alive {alive}^^xsd:boolean .
source    SELECT patient, LCASE(alive) AS alive
      FROM "@mirco.cazzaro"."STATIC_VARS"
      WHERE alive <> '';

mappingId MAPID-5b2c0dfea75d44279ce8ddde95f2a9aa
target    bto:Patient{patient} bto:sex {sex}^^xsd:string .
source    SELECT patient, LCASE(sex) AS sex
      FROM "@mirco.cazzaro"."STATIC_VARS"
      WHERE sex <> '';

mappingId MAPID-51a6eb8de7c64b7bbe4fe62d8188208c
target    bto:Patient{patient} bto:undergo bto:eventOnset1{patient} .
  bto:eventOnset1{patient} a bto:Onset ; bto:eventStart {onsetDate}^^xsd
  :datetime ; bto:bulbarOnset {onset_bulbar}^^xsd:boolean ; bto:
  axialOnset {onset_axial}^^xsd:boolean ; bto:generalizedOnset {
  onset_generalized}^^xsd:boolean ; bto:limbsOnset {onset_limbs}^^xsd:
  boolean .
source    SELECT patient, onsetDate, LCASE(onset_bulbar) AS onset_bulbar
  , LCASE(onset_axial) AS onset_axial, LCASE(onset_generalized) AS
```

```
  onset_generalized, LCASE(onset_limbs) AS onset_limbs FROM "@mirco.
  cazzaro"."STATIC_VARS";

mappingId MAPID-541428e7a83441f5bd29e52d66ffc52e
target    bto:Patient{patient} bto:undergo bto:eventOnset2{patient} .
  bto:eventOnset2{patient} a bto:Onset ; bto:ageOnset {age_onset}^^xsd:
  float .
source    SELECT patient, age_onset FROM "@mirco.cazzaro"."STATIC_VARS"
  WHERE age_onset <> '';

mappingId MAPID-ff2b4bdc90af468793b4f58105fa555f
target    bto:Patient{patient} bto:undergo bto:diagnosis{patient} . bto:
  diagnosis{patient} a bto:Diagnosis ; bto:eventStart {diagnosisDate}^^
  xsd:datetime .
source    SELECT patient, diagnosisDate FROM "@mirco.cazzaro"."
  STATIC_VARS";

mappingId MAPID-13ab39d8b67c4f34a206b7f4bcc1f442
target    bto:Patient{patient} bto:hasEthnicity bto:eth{patient} . bto:
  eth{patient} rdfs:label {ethnicity}@en .
source    SELECT patient, ethnicity FROM "@mirco.cazzaro"."STATIC_VARS"
  WHERE ethnicity <> '';

mappingId MAPID-56da0e23dc6b4cb3a0bfc475f4bcb65f
target    bto:eventOnset3{patient} bto:anatomicalLocation bto:location{
  patient} . bto:location{patient} rdfs:label {onset_location}^^xsd:
  string .
source    SELECT patient, onset_location FROM "@mirco.cazzaro"."
  STATIC_VARS" WHERE onset_location <> '';

mappingId MAPID-5ea1e8edbbb1446fb50e887c4f0605dd
target    bto:eventOnset4{patient} bto:consists bto:clinicalEvaluation{
  patient} . bto:clinicalEvaluation{patient} a bto:ClinicalEvaluation ;
  bto:predominantLimbsSide {onset_limbs_side}^^xsd:string ; bto:
  predominantLimbsImpairment {onset_limbs_impairment}^^xsd:string .
source    SELECT patient, onset_limbs_side, onset_limbs_impairment
      FROM "@mirco.cazzaro"."STATIC_VARS"
      WHERE onset_limbs_side <> '' OR onset_limbs_impairment <> '';
]]
```

Code 5.5: Mappings definition between the virtual relational schema exposed by Dremio and the BRAINTEASER Ontology

5.6 Ontop SPARQL

The aim of the OBDA approach is executing SPARQL queries over a VKG. Testing mapping correctness before their utilization in a running environment is essential. The Ontop SPARQL plugin for Protégé accomplishes this task. Given its role of a testing tool, it allows to analyze the "SQL translation" of the running SPARQL query.

5.6.1 Ontop Endpoint

The Ontop Standalone Endpoint provides a server environment where Ontop can operate independently, offering both an API and a web interface for running SPARQL queries over the virtual knowledge graph. This setup is particularly useful for environments where integration with existing data management systems is necessary, as it comes as an independent and lightweight tool.

5.6.2 Ontop with GraphDB

Integrating Ontop with GraphDB leverages the strengths of both platforms, combining Ontop's powerful OBDA capabilities with GraphDB's management functionalities. GraphDB allows to manage simultaneously different graph databases, where data is collected in independent "repositories". In particular, repositories can be standard repositories, thus ingesting RDF data, or virtual, that actually is an adaptation of the repository over an Ontop instance. This solution is particularly indicated for those scenarios where an easy-to-use web interface is necessary, allowing to export retrieved data in various formats.

6 Use Case

The proposed federated architecture in this thesis aims to run semantic queries over sparse and diverse data sources. In this chapter we will consider a practical use case of the architecture, showcasing both the flow of a query and its transformations among the diverse components of the architecture, as well as the inverse flow of the retrieved data, from the sources to the user interface.

6.1 Experimental Setup

The use case selected for this analysis involves a sophisticated SPARQL query designed to interlink and analyze clinical and recording data from the BRAINTEASER project dataset. This scenario is meticulously chosen to demonstrate the robust capability of the proposed architecture to manage and execute queries that necessitate access to multiple physical locations. The experiment is structured to conduct a thorough examination of the various components comprising the architecture, beginning with an in-depth analysis of the query itself. This initial stage focuses on inspecting the query's structure, understanding how it interacts with the data sources, and predicting its flow through the underlying system components. The goal is to highlight the interaction between the query and the data layers, offering insights into the complexities involved in federated data management. Following the structural analysis, the query's flow through the architecture will be meticulously presented and visualized. This involves analyzing each step of the query's execution, highlighting how it is rewritten, unfolded, and ultimately executed within the system. This phase aims to provide a clear and detailed schematic representation of the data flow, showcasing the dynamic interactions between the architectural components. The final phase of the experimental setup involves the actual execution of the query. During this stage, we will closely monitor and document the transformations that the query undergoes at various stages of the architecture. From its initial transformation from SPARQL into SQL by the Ontop component, through its optimization and execution in the data virtualization layer powered by Dremio, each transformation will be presented. The overarching objective of this experimental analysis is to validate the architecture's ability to efficiently manage and process complex queries across a federated data environment.

6.2 SPARQL Query over ALSFRS Relations

The selected SPARQL query, presented in Code 6.1, is specifically crafted to access and analyze the ALSFRS data, which is crucial for assessing the progression of ALS symptoms over time. This section discusses the structure of the query, its relevance to the dataset, and how it leverages the architecture's capabilities. The query employs the BRAINTEASER ontology schema, indicated by the prefix bto, to query ALSFRS data. It is designed to retrieve a comprehensive set of variables related to the ALSFRS assessments, including total scores and sub scores for bulbar, motor, and respiratory functions, as well as individual question scores from the ALSFRS-R (Revised ALS Functional Rating Scale). This query not only fetches detailed patient assessment data but also organizes it chronologically for each patient, aiding in the longitudinal analysis of disease progression. The ordered structure facilitates efficient data retrieval and subsequent analysis in a clinical research context. The chosen query is significant for several reasons: first and foremost, it accesses data unified from distinct sources, showcasing the architecture's capability to seamlessly integrate data across heterogeneous systems. Secondly, the query tests the system's capability to handle complex and extensive data structures, reflecting directly on the underlying data's relational union, which is a virtual view amalgamating identical table schemas from different databases. Finally, this same query will serve lately also as a benchmark to assess the correctness of the ontology mappings and the accuracy of data retrieval across the federated system. The ability to retrieve consistent and complete data sets across different platforms is crucial for validating the effectiveness of the federated architecture.

```
PREFIX bto: <https://w3id.org/brainteaser/ontology/schema/>

SELECT ?patient ?date ?tot ?bulbar ?motor ?respiratory ?q1 ?q2 ?q3 ?q4
 ?q5 ?q6 ?q7 ?q8 ?q9 ?q10 ?q11 ?q12 WHERE {
    ?p bto:undergo ?e.
    ?e bto:consists ?alsfrs.
    ?alsfrs a bto:ALSFRS;
           bto:procedureStart ?date;
           bto:revisedALSFRS ?tot;
           bto:bulbarSubscore ?bulbar;
           bto:motorSubscore ?motor;
           bto:respiratorySubscore ?respiratory;
           bto:alsfrs1 ?q1;
           bto:alsfrs2 ?q2;
           bto:alsfrs3 ?q3;
           bto:alsfrs4 ?q4;
           bto:alsfrs5 ?q5;
           bto:alsfrs6 ?q6;
           bto:alsfrs7 ?q7;
           bto:alsfrs8 ?q8;
           bto:alsfrs9 ?q9;
           bto:alsfrs10 ?q10;
           bto:alsfrs11 ?q11;
           bto:alsfrs12 ?q12.
    BIND(SUBSTR( (STR(?p)), 48) AS ?patient)
}
ORDER BY ?patient ?date
```

Code 6.1: SPARQL query on ALSFRS visits performed by patients over the BRAINTEASER ontology vocabulary

6.3 Query Flow

The flow of the query from its original form to the actual data retrieval across physical data sources is a multi-step process involving several layers of translation and optimization. Each phase of this transformation leverages different components of the architecture to ensure accurate and efficient query execution. Figure 6.1 represents in details this flow as the query is submitted to its final destinations (i.e. relational DBMSs).

6.3.1 Query Rewriting over Ontological Rules

The first transformation step involves expanding the SPARQL query according to the ontological rules defined in Ontop. Ontop uses these rules to interpret the query in the context of the ontology, enhancing the query by incorporating ontological knowledge. This includes inferring relationships, attributes, and classifications that are not explicitly stated in the query but are defined in the ontology. This expansion helps in making the query more comprehensive and aligned with the underlying data model, ensuring that the results are semantically consistent with the ontology's structure.

6.3.2 Query Unfolding to SQL

Once the query is expanded, Ontop performs the unfolding process. In this phase, the expanded SPARQL query is translated into SQL based on the mapping definitions that link the ontology to the virtual database schema. This step is crucial as it converts the high-level ontological query into a format that can be executed over a relational DBMS. The unfolding process ensures that the query is syntactically correct and optimized for execution against the structured schema of the data sources. The syntactical correctness of the output SQL query is strictly correlated with the correctness of the mappings.

6.3.3 Execution of SQL on Virtual Views

The final transformation occurs when the SQL query is processed by Dremio, which acts on the virtual views defined over the physical data sources. Dremio takes the SQL query and executes it against these virtual views, which represent a unified interface over the disparate databases. This phase is critical as Dremio optimizes the query's execution by deciding how best to access and retrieve the data from the underlying physical sources. This involves pushing down certain operations to the databases and merging results from the different sources.

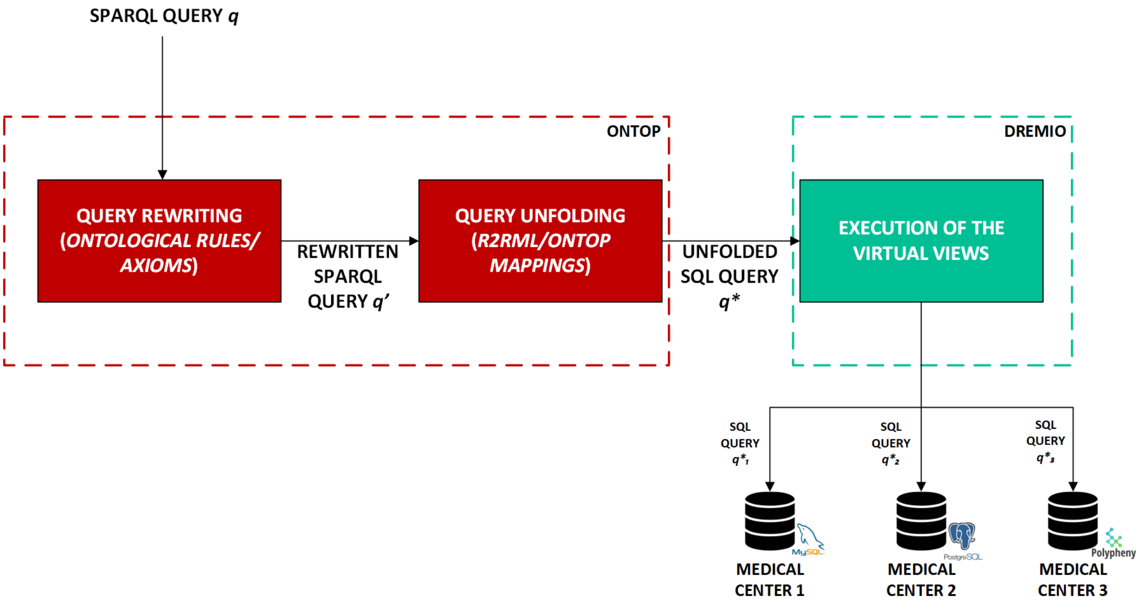

Figure 6.1: Data Flow Block Diagram of a Query Through the Federated Architecture

6.4 Exploration of Query Transformation in Ontop

In this phase we are using the Ontop framework within the Protégé environment, specifically chosen for its debugging capabilities. This setup is ideal for detailed inspection and debugging of the query transformation process, from a SPARQL query to an SQL query executable on relational databases. This decision allows for a granular inspection of how queries are translated from SPARQL to SQL, focusing on the use of ontological rules and mappings. After setting up all the components, we ran the query and we waited for the retrieval process to be completed. Given the distribution and the heterogeneity of the sources, the process was time consuming with respect to high-performance, centralized DBMSs. After assessing the result correctness by visual inspection, we analyzed the output query q* produced in output by the Ontop framework, that was submitted to Dremio. The query resulted in having long recursive patterns, particularly due to special character parsing processes. After manually inspecting and parsing these patterns in order to simplify and shorten the query repetitive body, we obtained the SQL presented in 6.2.

```
CONSTRUCT [patient, date, tot, bulbar, motor, respiratory, q1, q2, q3
    , q4, q5, q6, q7, q8, q9, q10, q11, q12]
   [date/RDF(CHARACTER VARYINGToVARCHAR(date2m51), xsd:datetime),
    q1/RDF(INTEGERToVARCHAR(q11m25), xsd:integer),
    motor/RDF(INTEGERToVARCHAR(motor1m55), xsd:integer),
    ...]
NATIVE
SELECT
    v23."bulbar1m17" AS "bulbar",
    v23."date2m51" AS "date",
    v23."motor1m55" AS "motor",
    v23."q101m44" AS "q10",
    v23."q111m42" AS "q11",
    v23."q11m25" AS "q1",
    v23."q121m41" AS "q12",
    v23."q21m24" AS "q2",
    v23."q31m23" AS "q3",

    --- ... more fields

    v23."patient26m9" AS "patient"
FROM (
    SELECT DISTINCT
        v7."bulbar" AS "bulbar1m17",
        v5."date2m51" AS "date2m51",
        v8."motor" AS "motor1m55",
        v1."patient" AS "patient26m9",
        v19."q10" AS "q101m44",
        v20."q11" AS "q111m42",
        v10."q1" AS "q11m25",

        --- ... more fields

        v6."tot" AS "tot1m45"
    FROM "clinical_data"."ALSFRS" v1

    --- ... recursive joins

    WHERE v1."patient" IS NOT NULL AND v1."date" IS NOT NULL
) v23

ORDER BY v23."date2m51" NULLS FIRST
```

Code 6.2: Parsed SQL Translation of the Original SPARQL Query

The CONSTRUCT clause is crucial in shaping the structure of the resultant RDF data, where each field is transformed to match a specific data type and linked to its RDF representation. This transformation ensures that the output aligns with the expected semantic standards, facilitating subsequent data integration and analysis. The NATIVE clause instead specifies the direct SQL translation components, signifying how the translated query interfaces with the underlying SQL database. It highlights the direct mapping of SPARQL to SQL, ensuring that the original semantic query's intent is preserved while being adapted for execution over relational data structures. The SELECT clause in SQL is constructed to directly map each variable specified in the SPARQL SELECT query. Each variable such as ?patient, ?date, ?tot, etc., corresponds to specific fields in the underlying database tables. For example, bto:procedureStart translates to selecting the date column in SQL. The SPARQL BIND function used to extract the patient ID from a URL or string is represented in SQL with a SUBSTRING function, allowing the extracted patient ID to be represented correctly in the result set. The FROM clause in SQL involves the specification of tables and joins that correspond to the triples patterns defined in the SPARQL query. The SPARQL predicate bto:undergo and bto:consists suggest relational joins between patient data and ALSFRS assessment data. This relationship is mapped to SQL joins or subqueries that fetch data from multiple related tables, capturing the relational nature of the data as specified by the ontology. The WHERE clause in SQL is essential for filtering data based on conditions expressed in SPARQL. Conditions such as ensuring data completeness or filtering based on specific criteria (like date ranges or specific patient characteristics) are directly translated from SPARQL conditions. The SQL version ensures that only relevant, complete records are processed. The ORDER BY clause in SQL mirrors the SPARQL order condition, ensuring that the results are returned in a specific order, which in this case is based on patient ID and date. This ordering is crucial for chronological analysis of patient data in medical research. The detailed translation process showcases the precise alignment from SPARQL to SQL, ensuring that the data fetched and processed adheres strictly to the semantic structure defined by the ontology.

6.5 Execution of q* on Dremio Virtual Views

In our architecture, Dremio operates as a single-instance node rather than a distributed cluster, focusing on executing SQL queries that have been transformed from SPARQL to SQL. This phase critically leverages Dremio's robust logging capabilities, which provide detailed insights into the query execution process. The Dremio query planner, which is central to its operation, allows for the export of query execution plans in JSON format. By inspecting these logs, we can observe and analyze how the query is optimized and executed. Dremio's execution engine processes the query against virtual datasets, which abstract the underlying physical data sources such as MySQL, PostgreSQL, and Polypheny. This virtualization allows for efficient data querying across heterogeneously structured data repositories without necessitating physical data aggregation or duplication. Such a setup is essential for maintaining high performance and flexibility in data handling. The detailed query plan extracted from the JSON log shows the optimizer's role in structuring the execution. It details the optimized execution paths designed to minimize the overhead associated with processing complex SQL queries. These paths reflect the strategic planning of operations to enhance query performance by reducing unnecessary data shuffles and network traffic, which is crucial in a single-node setup where resource optimization is key. The execution log provides insights into the handling of complex joins and the extensive use of URL encoding within SQL queries. The presence of multiple nested SELECT statements and intricate joins illustrates Dremio's capability to reconstruct semantic relationships that were initially expressed in the SPARQL format. This ensures that the semantic integrity and the meaning of the original queries are preserved and effectively adapted for execution over relational data structures. Furthermore, the logs reveal Dremio's use of Apache Arrow for managing data formats, ensuring efficient data processing and exchange. This choice underscores the system's design towards maximizing processing speed and minimizing latency, which is particularly beneficial in a single-node environment where all processes converge on one machine. Performance enhancement strategies such as sophisticated caching mechanisms are also evident from the logs. These mechanisms optimize the execution of repeated queries and manage frequently accessed data, thereby reducing execution times and improving the system's responsiveness. Moreover, specific SQL transformations highlighted in the log, including the decoding of URLs back to standard formats, underscore the meticulous attention to ensuring data compatibility and correctness. This aspect is critical for the subsequent stages of data analysis and integration. By executing these virtual views, Dremio facilitates real-time data retrieval and analysis, crucial for dynamic decision-making processes in clinical research environments. The detailed inspection of Dremio's query execution logs not only demonstrates the practical application of semantic web technologies in modern data architectures but also ensures that the data retrieval process is both efficient and semantically consistent.

6.6 Summary

This chapter detailed a use case that applied our federated data analytics architecture to handle complex queries across diverse data sources. The architecture facilitated the integration and querying of heterogeneous data sets using SPARQL-to-SQL translations within a federated system featuring Ontop and Dremio. The experiment demonstrated the system's capability to efficiently manage and optimize complex queries, reducing execution times and maintaining data integrity. The analysis of Dremio's logs revealed effective query optimizations and the robust handling of federated queries, showcasing the potential of the architecture to support real-time, data-intensive operations in clinical genomics research. Overall, the experiment confirmed the architecture's effectiveness in leveraging semantic web technologies and federated systems to enhance data processing and accessibility, setting a foundation for future research and applications in distributed data environments.

7 Architecture Benchmark Evaluation

Following the comprehensive exploration of our federated data architecture's design and practical use cases in earlier chapters, we now shift our focus towards a crucial aspect of system development, that is its performance and operational efficiency benchmarking evaluation. This evaluation is crucial, as it determines the feasibility and effectiveness of the architecture in real-world applications, particularly in the data-intensive field of genomics research. Benchmarking in the context of this thesis encompasses necessarily a multifaceted approach, considering both the resource consumption as well as the DBMS performance of the proposed architecture. The architecture must not only prove robustness in handling complex data interactions but also achieve this with good efficiency in terms of both time and cost. By implementing this dual-focused benchmarking approach, we aim to address two fundamental questions: how well the architecture performs under typical and peak loads, and how does it manage the computational resources at its disposal. Answering these questions will provide a comprehensive understanding of the system's operational characteristics and its suitability for deployment in real-world genomics research environments. The insights gained from this benchmarking phase are intended to provide a better understanding of the system's operational dynamics. These benchmark results will not only validate the architecture's capabilities but also highlight areas where further optimizations are necessary, ensuring the system's alignment with the high-throughput and high-accuracy demands of modern DBMS, being always close to the clinical and genomics fields' requirements. This chapter will lay out the methodologies employed in our benchmarking tests, discuss the benchmarks selected for evaluation, and detail the performance metrics that will guide our analysis of the architecture's suitability for real-world applications.

7.1 State of the Art on Benchmarking Techniques

In this section we aim to provide an overview on two state-of-the-art methodologies and their respective framework, differing in the benchmarked factors, that can provide useful insights under diverse point of view. The objective is understanding their logics and wether they can be insightful in our context.

7.1.1 The Berlin SPARQL Benchmark

The Berlin SPARQL Benchmark (BSBM) [28] was developed to evaluate the performance of RDF data management systems by comparing native RDF stores with SPARQL-to-SQL rewriters, which translate SPARQL queries into SQL queries on-the-fly. This benchmark is particularly crucial for understanding how different systems handle RDF data under various conditions, an essential factor for applications involving complex and voluminous datasets like those found in genomics. BSBM is structured around an e-commerce use case where products are offered by vendors and reviewed by consumers, creating a realistic scenario for benchmarking. The design objectives of BSBM focus on providing a meaningful comparison across different storage systems that expose SPARQL endpoints. These objectives ensure the benchmark reflects typical use-case scenarios and assesses how systems perform under realistic and concurrent client workloads.

BSBM Dataset and Query Mix

The BSBM dataset is scalable and can be generated in different sizes, allowing for comprehensive testing across systems. Having the dataset "generated" means assessing its performances on a synthetic dataset, that in particular models a typical e-commerce domain with entities like Products, Vendors, Reviews, etc. The benchmark's data generator supports creating arbitrarily large datasets by adjusting the number of products, which serves as the scale factor. This flexibility in data generation enables BSBM to simulate different data volumes and complexities, providing insights into how systems scale with increasing data sizes, while working on synthetic data gives the best combination of a common baseline for DBMSs comparisons as well as a variable dataset so not to overfit while optimizing the performances. The query mix used in BSBM simulates the search and navigation patterns of consumers looking for products, mimicking a consumer's interaction with a database. This includes queries for finding products based on various features, retrieving detailed product information, and querying for reviews. The mix consists of parameterized queries with randomly generated values to prevent caching optimizations, ensuring that the benchmark measures genuine query processing performance.

Benchmark Implementation

BSBM's implementation includes a test driver and a data generator, which together facilitate the execution of the benchmark across different systems. The test driver manages the execution of SPARQL queries over a network, simulating multiple clients, and measures the system's performance based on queries per second and average query execution time. The data generator outputs the data in both RDF and relational formats, allowing for a direct comparison of RDF stores and relational databases using SPARQL-to-SQL translation techniques.

Contributions and Impact of BSBM

BSBM has significantly contributed to the field by providing a robust framework for evaluating RDF data management systems in terms of SPARQL query performance and also standard relational DBMSs in terms of SQL query performances, becoming undoubtedly the state of the art for benchmarking DBMSs. By applying BSBM across various systems, it has revealed strengths and weaknesses in existing implementations, guiding improvements in RDF store and SPARQL-to-SQL rewriter technologies. Moreover, BSBM assists application developers in selecting appropriate storage systems based on performance metrics critical to their specific requirements.

7.1.2 SEASHELL Benchmark for Healthcare Data Lakes

In the rapidly evolving domain of healthcare informatics, the creation of a specialized benchmarking framework such as reSource bEnchmark for Analysis taSks in HEaLthcare data Lakes (SEASHELL) was necessary [29]. This benchmark addresses the pressing need to evaluate healthcare data lake infrastructures comprehensively, ensuring they are optimally configured to handle the complexities of modern medical data. As healthcare organizations increasingly rely on data-driven insights to improve patient outcomes and operational efficiencies, the ability to effectively manage and analyze vast arrays of medical data becomes crucial.

Design and Objectives of SEASHELL

SEASHELL is meticulously designed to assess the performance of healthcare data lakes, namely critical infrastructures that integrate and process diverse data types from electronic health records to genomics data. The benchmark aims to evaluate these systems on several fronts: data handling, or rather the system's capability to manage and query large and varied datasets that are typical in healthcare environments; performance efficiency, that means measuring the speed and resource efficiency of data processing tasks, which are vital for timely medical decisions, and the system scalability capabilities.

Benchmark Framework and Test Scenarios

The SEASHELL benchmark introduces a comprehensive framework designed to test data lake infrastructures under various workload scenarios, including relational analyses and machine learning tasks that mirror real-world applications such as disease prediction and patient data management. It employs a virtualized implementation of a healthcare-specific data lake architecture alongside two external cloud-based infrastructures to demonstrate its flexibility and adaptability in diverse environments. The architectural design of the SEASHELL benchmark is intentionally designed to be versatile and accurately mirror the complexities found in real-world healthcare Information Technology (IT) environments. This design choice ensures that SEASHELL can thoroughly evaluate how well a healthcare data lake manages, integrates, and analyzes different data types, both structured, like Electronic Health Records (EHR)s, and unstructured, such as medical images. Beyond just handling diverse data types, the benchmark extends its testing to encompass a range of analytical functions, from basic SQL-based querying to advanced analytics, reflecting the comprehensive data analysis tasks encountered in healthcare settings. Furthermore, the adaptability of SEASHELL is rigorously evaluated across various settings, including custom-built virtualized environments and external cloud-based platforms, ensuring its effectiveness and applicability across the diverse IT infrastructures that are typical in modern healthcare settings.

Overview of Non-Machine-Learning Tasks in SEASHELL

As outlined previously, SEASHELL comprises on non-machine-learning tasks, which are particularly interesting for what concerns benchmarking DBMSs. These tasks involve:

- Data Retrieval and Querying: Testing the efficiency and speed of basic data retrieval operations, which are foundational for any data-driven healthcare system. This includes executing complex SQL queries across large datasets to simulate the retrieval of patient information, treatment outcomes, and other critical data in real time;
- Data Processing and Transformation: Evaluating the data lake's capability to perform necessary data transformations, such as normalizing diverse data sets from various sources (like labs and medical devices) into a unified format that can be easily analyzed and stored.
- Resource Utilization: Measuring the computational and memory resources utilized during these operations to assess the cost-effectiveness of data lake architectures in a healthcare setting.

Contributions and Impact of SEASHELL

SEASHELL is a comprehensive benchmark framework that not only tests the performance of healthcare data lakes but also guides the optimization and scaling of these critical infrastructures. It is a fundamental tool for gathering detailed insights into both machine-learning and non-machine-learning capabilities, and it helps ensure that healthcare data lakes are not only capable of advanced data analysis but are also efficient and reliable for everyday medical data processing.

7.2 Server Environment Configuration

Understanding the server environment where our architecture is evaluated is crucial for benchmarking, as it provides a context to the evaluation results, that will be presented later on in this chapter. Moreover, it is important to analyze wether the server hardware and software capabilities are sufficient to accurately assess the architecture's performance under different loads.

The server's CPU, an Intel Xeon Silver 4116, features a high number of cores and threads (48 cores and 96 threads distributed across two sockets). This setup

Specification	**Details**
Operating System	Rocky Linux 8.10 (Green Obsidian)
CPU	Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz
CPU Details	48 cores, 96 threads, 24 cores per socket
RAM Memory	Total: 564GB, Available: 487GB
Storage - Locale	2.5TB (1.9TB used, 148GB available)
Kernel Version	Linux x86_64, Kernel 4.18.0

Table 7.1: Server Specifications

is excellent for multitasking and running multiple operations simultaneously, which is common in our benchmark tests. About RAM memory, our server has 564GB in total. This large amount of memory suits well for processing big datasets without the need of frequently swapping data with slower disk-based storages. Storage capacity is also important. Our server includes a 2.5TB local drive, of which 1.9TB is used, leaving 148GB free. This space is adequate for our needs during the benchmarking phase, allowing us to store and manage large amounts of data effectively. In summary, the server setup is well-suited for the demanding tasks of benchmarking our federated data architecture. The specifications ensure that we can conduct thorough and accurate performance tests, which are essential for validating the architecture's performances.

7.3 BSBM Benchmark Results over Synthetic Data

As underlined before, the BSBM framework aims to provide tools and methodologies to assess DBMS performance, both relational and RDF-based, stressing the databases in terms of concurrent queries, and gathering measurements like queries per second (QPS) and the average query execution time. The BSBM toolkit[1], provided and maintained by the Free University of Berlin, is open source and it consists of a series of JAR packages that perform different operations. In particular, we focus on two of them, that refers to the Generator class and the TestDriver class. The Generator class, and its related JAR executor, aims on generating the synthetic dataset. Users can decide by setting up command line

[1]https://sourceforge.net/projects/bsbmtools/files/bsbmtools/

arguments, the size of the dataset in terms of total rows as well as the output format, whether it has to be SQL or RDF. The schema structure of the generated relational source or the generated graph is constant (i.e. 10 tables in case of the relational source), while data in text fields or individual data properties is recreated at each generation from two text files, containing many words, in a random manner.

Synthetic Data Generation

For our experiment, we generated 100k rows in SQL format, as these is the feeding format for our architecture. The data was generated as shown in Code 7.1.

```
    $ ./generate -pc 100000 -s sql
```

Code 7.1: BSBM generator script that invokes Generator class and JAR executor

As the output SQL syntax was compatible with MySQL, in order to mimic the behavior of the architecture at its operational status, we split the schema in half and imported the two portions in two different MySQL servers. Figure 7.1, showing the benchmarking setup, represents also this scenario.

BSBM R2RML Mappings

A huge obstacle we were facing at this stage was that the tool was not intended for running in a context like the one we developed in our thesis. In other words, the BSBM procedures expects the user to generate RDF synthetic data and benchmark a triple store system, by running SPARQL queries over it, or instead generating SQL data and benchmark a relational DBMS, by running SQL queries over it. No use case were foreseen for a scenario where SPARQL queries were executed over a VKG on underlying SQL data. Luckily, a team of researchers from the Charles University in Czech Republic faced this problem in a recent scientific paper [30], developing R2RML mappings[2], specifically between the graph and relational data models.

Given that mappings, as it occurs within the Ontop Native syntax, must map portions of a graph to an SQL query, the proposed mappings were performing this operation with a standard SQL syntax, that was not suited for the Dremio

[2]https://github.com/mchaloupka/bsbm-r2rml/blob/develop/src/main/dist/rdb2rdf/mapping.ttl

SQL dialect. Given this, we had to manually inspect and edit the mappings so to make them compatible with our virtualization system. Code 7.2 shows a portion of these mappings, where we intervened by adding the Dremio space name (i.e. the equivalent of the schema in PostgreSQL) that was hosting our virtual views.

```
<#ProductFeature> a rr:TriplesMap;
rr:logicalTable [ rr:tableName "\"@mirco.cazzaro\".productfeature" ];
rr:subjectMap [
rr:template "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/
ProductFeature{nr}";
rr:class bsbm:ProductFeature;
];
rr:predicateObjectMap [
rr:predicate rdfs:label;
rr:objectMap [ rr:column "label"; ];
];
rr:predicateObjectMap [
rr:predicate rdfs:comment;
rr:objectMap [ rr:column "comment"; ];
];
rr:predicateObjectMap [
rr:predicate dc:publisher;
rr:objectMap [ rr:template "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/
v01/instances/StandardizationInstitution{publisher}"; ];
];
rr:predicateObjectMap [
rr:predicate dc:date;
rr:objectMap [ rr:column "publishDate"; ];
];
.
```

Code 7.2: BSBM Custom Mappings for Dremio SQL Syntax in R2RML

Benchmarking Setup

The whole setup, represented in Fig. 7.1 was finally ready to run the benchmark tests. The MySQL istances have set up on physical, remote servers, always in order to adhere to the paradigm of replicating a real world scenario use case.

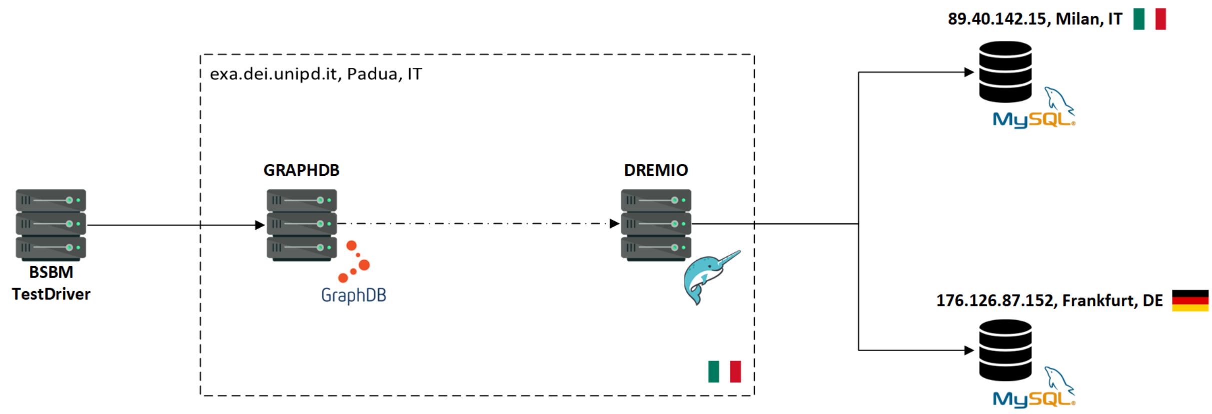

Figure 7.1: Architecture Setup for Benchmarking

Benchmark Results

We ran the TestDriver with the following parameters:

```
$ java -cp bin:lib/* benchmark.testdriver.TestDriver -runs 32 -w
4 -mt 4 -t 30000 http://localhost:7200/repositories/BSBM
```

Code 7.3: BSBM TestDriver class execution

For each query (12 in total), we ran them 32 times with 4 concurrent processes submitting queries. We excluded the first 4 queries from the measurements as a warm-up phase to eliminate any potential noise from caching mechanisms. Moreover, we set a timeout of 30 seconds for each query.

The first notable observation from results in Fig. 7.2 is that the plots are complementary: this is expectable, as a one of the main causes for a low query rate can be for sure an high response time. We can observe how Query 3 and Query 11 are performing poorly with respect to the others. As we will see in next section with other benchmarks, where we will compare resource consumption between Dremio and Ontop hosted in GraphDB, the component that usually is more resource-demanding and consuming is the virtualization system Dremio. This implies that an eventual bottleneck has to be caused at this level. By inspecting Queries 3 and 11, we can see how these two make use of extensive joins among tables that in our set up belongs to different sources as well as nested queries. This translates to a possible weakness of the proposed architecture while dealing with these case study, but that can surely be investigated and strengthen in future. In general, our architecture was able, on average, to perform 15 queries per seconds from 4 concurrent processes acting as independent users. This represent a huge milestone, considering it is an embryonic study.

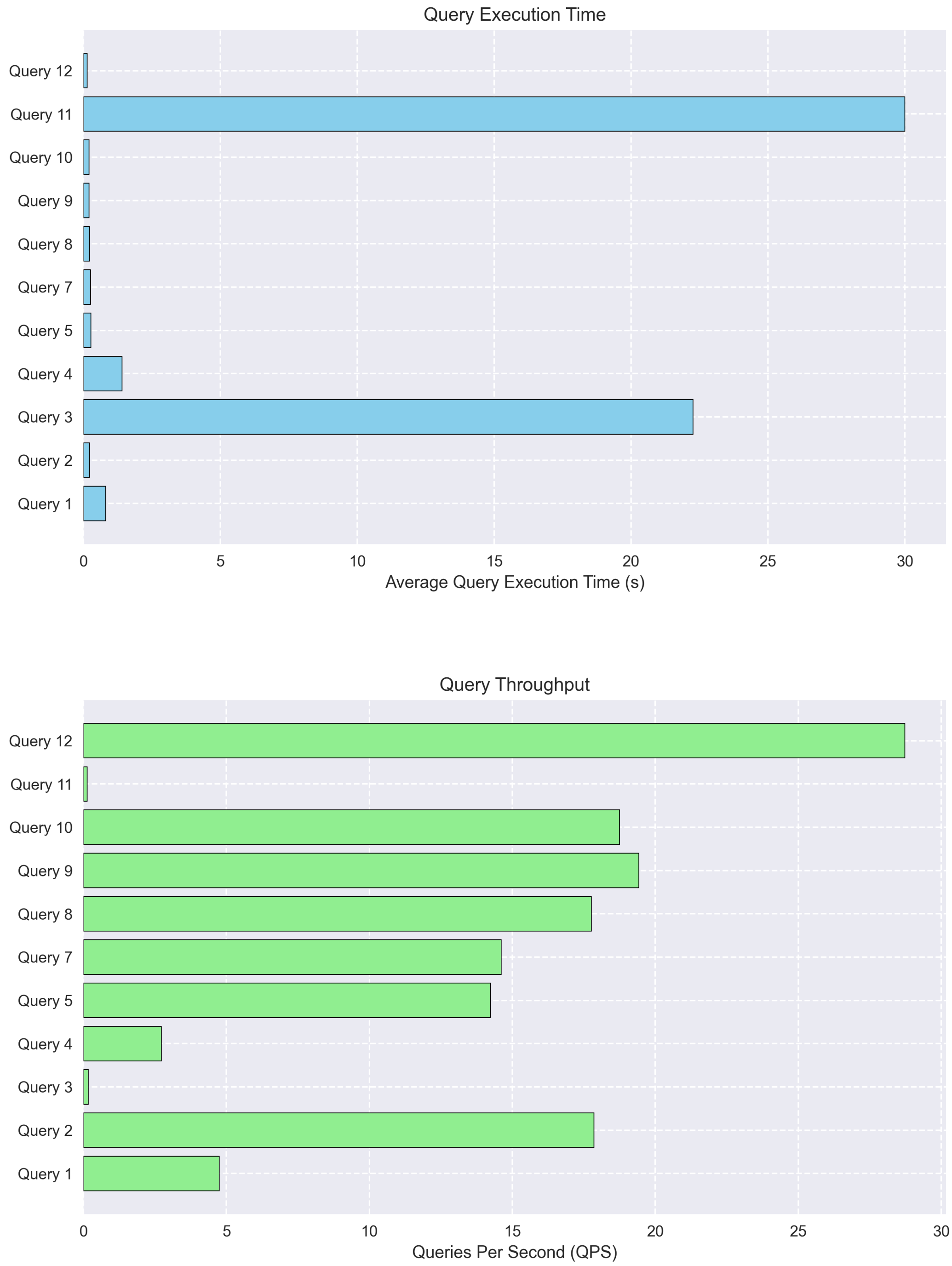

Figure 7.2: BSBM Benchmark Results

7.4 Architecture Performance Benchmarking over Clinical Data

In alignment with the methodologies presented in Section 7.1.2, we conducted an extensive evaluation of our healthcare data architecture using our available clinical data. This benchmarking initiative was critical to validate the architecture's operational efficacy, specifically targeting its ability to handle real-world healthcare analytics tasks efficiently. The benchmark framework was constructed around a custom-developed monitoring script. This script, integral to our testing methodology, was engineered to capture and log real-time CPU and RAM delta usage metrics by process ID.

```bash
#!/bin/bash

pid=$1
output_file=$2

# Get the number of CPUs
num_cpus=$(nproc)

# Time interval in seconds
interval=1

trap 'echo "Monitor stopped"; exit' SIGINT SIGTERM

if [ -z "$pid" ] || [ -z "$output_file" ]; then
    echo "Usage: $0 <PID> <output_file>"
    exit 1
fi

exec > $output_file

while true; do
    # Capture CPU times at the start of the interval
    start_utime=$(cat /proc/$pid/stat | cut -d " " -f 14)
    start_stime=$(cat /proc/$pid/stat | cut -d " " -f 15)
    start_total=$(cat /proc/stat | grep '^cpu ' | awk '{print $2+$3+$4+$5+$6+$7+$8}')

    # Capture the initial RAM usage
    start_ram=$(grep VmRSS /proc/$pid/status | awk '{print $2}')
```

```

    sleep $interval

    # Capture CPU times at the end of the interval
    end_utime=$(cat /proc/$pid/stat | cut -d " " -f 14)
    end_stime=$(cat /proc/$pid/stat | cut -d " " -f 15)
    end_total=$(cat /proc/stat | grep '^cpu ' | awk '{print $2+$3+$4+
   $5+$6+$7+$8}')

    # Capture the end RAM usage
    end_ram=$(grep VmRSS /proc/$pid/status | awk '{print $2}')

    # Calculate the deltas for CPU
    delta_process=$(( (end_utime + end_stime) - (start_utime +
   start_stime) ))
    delta_total=$(( end_total - start_total ))

    # Calculate CPU usage as a percentage
    cpu_usage=$(awk "BEGIN {printf \"%.2f\", (${delta_process} / ${
   delta_total}) * 100 * ${num_cpus}}")

    # Calculate RAM usage change if needed
    ram_usage_change=$(( end_ram - start_ram ))

    echo "$(date +'%Y-%m-%d %H:%M:%S') CPU Usage: ${cpu_usage}% | RAM
    Usage Change: ${ram_usage_change} kB"

done
```

Code 7.4: Custom Monitor Script for Performance Tracking

The benchmarking trials were designed to assess the architecture's performance by running the same analytical query we considered for the use case analysis in Chapter 6 that simulate typical data processing tasks encountered in clinical settings. During these tests, our monitoring script provided continuous feedback on system performance, enabling a comprehensive analysis of CPU and RAM usage across different operational states. Illustrated in Figures 7.3 and 7.4, the system's response to the execution of computationally intensive queries reveals significant insights into its dynamic resource allocation and management capabilities. We analyzed the output from the two monitoring daemons, synchronizing samples at the moment the query was submitted by inspecting the query logs in Dremio. This allowed us to generate superimposable

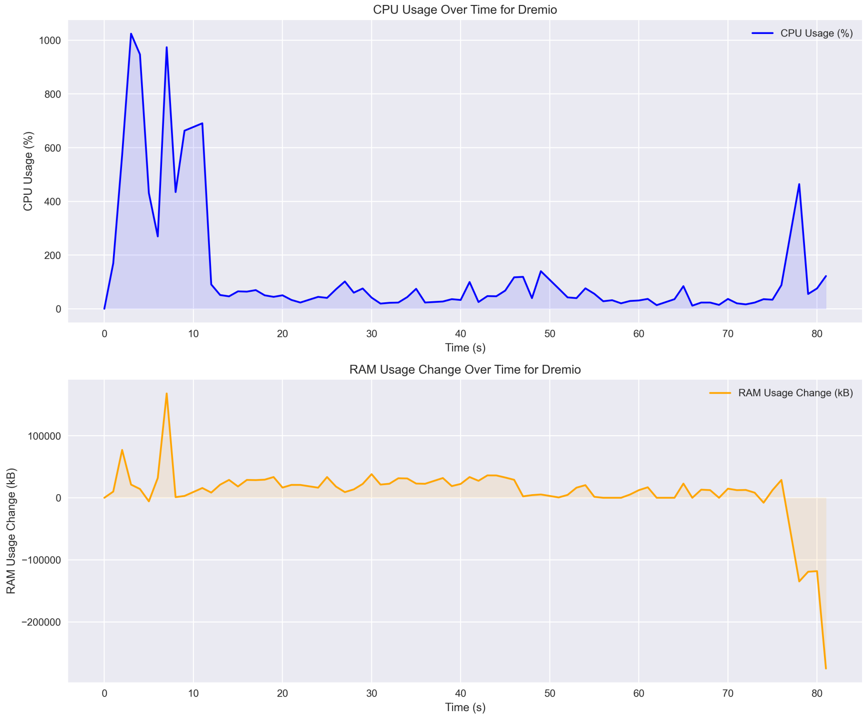

Figure 7.3: CPU and RAM usage over time for Dremio during benchmarking

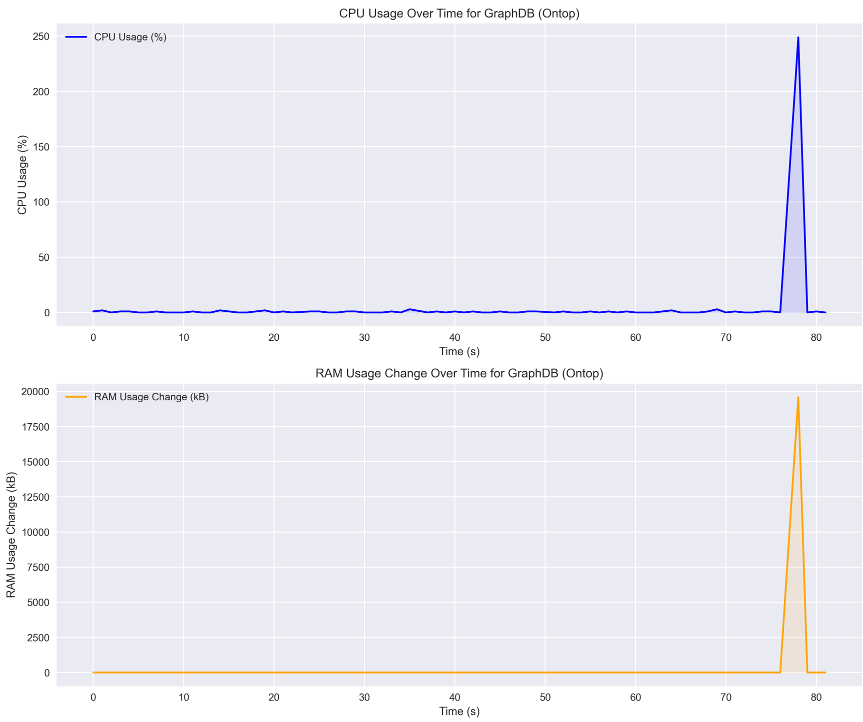

Figure 7.4: CPU and RAM usage over time for GraphDB (Ontop) during benchmarking

plots for independent analysis of CPU load and variations in RAM usage. Notably, the trends for both CPU and RAM usage appear almost flat in GraphDB, except for a spike nearly coinciding with the completion of query execution. In contrast, Dremio exhibits non-negligible resource consumption throughout the entire duration of the query execution, with spikes occurring both at the beginning and the end of the process. These phenomena may be explained by the fact that, upon submission, query execution in GraphDB does not initially involve resource-intensive activities, whereas in Dremio, it is necessary to exploit the views structure of the submitted query to begin fetching data from various sources. The spikes at the end are likely related to the combination of results from these sources and the activities involved in presenting the data to the user. The occurrence of these spikes suggests potential areas for optimization, which could be crucial for improving the system's efficiency and responsiveness, especially under peak load conditions. In conclusion, this benchmarking exercise has not only validated the robustness of our healthcare data architecture but also identified specific enhancement opportunities that can significantly improve system performance.

7.5 Summary and Conclusions

This chapter provided a detailed overview of the benchmarking process for our federated data architecture, particularly focusing on its performance and operational efficiency in the realm of genomics research. Through a series of meticulously designed benchmarks, we evaluated the architecture's ability to handle complex data interactions and manage computational resources effectively, ensuring its suitability for deployment in high-demand genomics research environments. The benchmarks detailed in this chapter utilized state-of-the-art methodologies, including the Berlin SPARQL Benchmark and the SEASHELL framework for healthcare data lakes, to provide a comprehensive assessment of the architecture's performance across various scenarios. These tests revealed both the strengths and potential areas for improvement in our system, highlighting the critical need for continual optimization.

8

Conclusions and Future Works

The research and development work undertaken in this thesis has focused on the challenging task of integrating and analyzing heterogeneous biomedical data, specifically clinical and genomics data. The primary aim has been to design a robust and scalable federated data analytics system, capable of handling the complexities associated with diverse data sources. Leveraging the OBDA paradigm and advanced data federation techniques, the system provides a unified platform that not only integrates but also semantically enriches the data from multiple, disparate sources. The system's architecture, built on Dremio as the data federation layer and Ontop for semantic data integration, has demonstrated the feasibility and effectiveness of using these technologies in a federated environment. The choice of Dremio was driven by its open-source nature, robustness, scalability, and ability to manage different types of data sources, including relational databases, polystore systems, and cloud storage solutions. Ontop was selected as well because it is FOSS, but also for its compliance with W3C standards and its capability to perform high-performance query answering over virtualized RDF graphs. Together, these technologies compose the backbone of the proposed system, enabling complex queries over diverse datasets without the need for extensive data preprocessing or manual data integration. The implementation of this system within the context of the HEREDITARY project has further validated its applicability in real-world scenarios. The federated data analytics system developed in this thesis has been designed as an initial prototype that addresses the specific needs of this project, facilitating the integration and analysis of clinical and genomics data from different medical centers across

Europe. The system's ability to handle both structured and unstructured data, its support for real-time data retrieval, and its compliance with privacy regulations such as GDPR, make it a valuable tool for biomedical research. However, the work presented in this thesis is not without limitations, and several areas for future research and development have been identified. One of the most promising directions for future work is the packetization or dockerization of the system's software components, so to transform it into a single reusable application. By containerizing the various components of the system, such as the Dremio and Ontop instances, it would be possible to simplify the deployment process and ensure consistent performance across different environments. Dockerization would also facilitate the scaling of the system, allowing it to handle larger datasets and more complex queries by distributing the workload across multiple containers. Another area for future research is strengthening the system's compliance with GDPR and other data protection regulations. While the current system has been designed with privacy and data security in mind, there is always room for improvement. Future work could focus on developing more sophisticated techniques for data anonymization and encryption, as well as implementing stricter access controls and auditing mechanisms to ensure that sensitive data is always protected. Additionally, further research could explore the integration of federated learning techniques, which would allow for the analysis of data across multiple sites without the need to share the raw data itself, thus enhancing privacy and security. Optimizing the system's performance is another critical area for future work. The benchmarking results presented in this thesis have highlighted the strengths and weaknesses of the current architecture, particularly in terms of query execution time and resource consumption. Future research could focus on refining the system's performance by utilizing the results of the benchmarking as a ground point for optimization. This could involve fine-tuning the system's configuration, improving the efficiency of the query rewriting and unfolding processes, or developing new algorithms for data federation and virtualization. Special attention should be given to optimizing the system's performance during peak loads, as these are often the most critical points in terms of resource usage and response time. In addition to these specific areas of future research, there are also broader questions that could be explored in relation to the system's overall design and functionality. For example, the system be made more user-friendly, particularly for researchers and clinicians who may not have a background in data science. As an example, it could be possible to develop a custom endpoint, developed from scratch, that interfaces with the architecture from the top, allowing to easily and visually build queries to deliver to the federated architecture, and that effectively allows to perform various analytics based on retrieved data. In conclusion, the federated data analytics system developed in this thesis represents a significant step forward in the integration and analysis of heterogeneous biomedical data. When combining advanced data federation techniques with the semantic power of OBDA, the system offers a scalable, flexible, and privacy-conscious solution for managing the complexities of clinical and genomics data. As mentioned earlier, the work is far from complete, and there are many opportunities for future research and development. Continuing to build on the foundations laid in this thesis will make it possible to create even more powerful and versatile tools for biomedical research, contributing to better healthcare outcomes and a deeper understanding of human health and disease.

References

[1] Dave Beckett and Brian McBride. "RDF/XML syntax specification (revised)". In: *W3C recommendation* 10.2.3 (2004).

[2] Deborah L McGuinness, Frank Van Harmelen, et al. "OWL web ontology language overview". In: *W3C recommendation* 10.10 (2004), p. 2004.

[3] Zhenzhen Gu, Francesco Corcoglioniti, Davide Lanti, Alessandro Mosca, Guohui Xiao, Jing Xiong, and Diego Calvanese. "A systematic overview of data federation systems". In: *Semantic Web* 15.1 (2024), pp. 107–165. DOI: 10.3233/SW-223201. URL: https://doi.org/10.3233/SW-223201.

[4] Maurizio Lenzerini. "Data Integration: A Theoretical Perspective". In: *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*. Ed. by Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis. ACM, 2002, pp. 233–246. DOI: 10.1145/543613.543644. URL: https://doi.org/10.1145/543613.543644.

[5] Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. "Ontology-Based Data Access: A Survey". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. Ed. by Jérôme Lang. ijcai.org, 2018, pp. 5511–5519. DOI: 10.24963/IJCAI.2018/777. URL: https://doi.org/10.24963/ijcai.2018/777.

[6] Elena Botoeva, Diego Calvanese, Benjamin Cogrel, Julien Corman, and Guohui Xiao. "A Generalized Framework for Ontology-Based Data Access". In: *AI*IA 2018 - Advances in Artificial Intelligence - XVIIth International Conference of the Italian Association for Artificial Intelligence, Trento, Italy, November 20-23, 2018, Proceedings*. Ed. by Chiara Ghidini, Bernardo Magnini, An-

drea Passerini, and Paolo Traverso. Vol. 11298. Lecture Notes in Computer Science. Springer, 2018, pp. 166–180. DOI: 10.1007/978-3-030-03840-3_13. URL: https://doi.org/10.1007/978-3-030-03840-3%5C_13.

[7] Konstantinos Makris, Nektarios Gioldasis, Nikos Bikakis, and Stavros Christodoulakis. "Ontology Mapping and SPARQL Rewriting for Querying Federated RDF Data Sources - (Short Paper)". In: *On the Move to Meaningful Internet Systems, OTM 2010 - Confederated International Conferences: CoopIS, IS, DOA and ODBASE, Hersonissos, Crete, Greece, October 25-29, 2010, Proceedings, Part II*. Ed. by Robert Meersman, Tharam S. Dillon, and Pilar Herrero. Vol. 6427. Lecture Notes in Computer Science. Springer, 2010, pp. 1108–1117. DOI: 10.1007/978-3-642-16949-6_32. URL: https://doi.org/10.1007/978-3-642-16949-6%5C_32.

[8] Evgeny Kharlamov, Nina Solomakhina, Özgür Lütfü Özçep, Dmitriy Zheleznyakov, Thomas Hubauer, Steffen Lamparter, Mikhail Roshchin, Ahmet Soylu, and Stuart Watson. "How Semantic Technologies Can Enhance Data Access at Siemens Energy". In: *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*. Ed. by Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig A. Knoblock, Denny Vrandecic, Paul Groth, Natasha F. Noy, Krzysztof Janowicz, and Carole A. Goble. Vol. 8796. Lecture Notes in Computer Science. Springer, 2014, pp. 601–619. DOI: 10.1007/978-3-319-11964-9_38. URL: https://doi.org/10.1007/978-3-319-11964-9%5C_38.

[9] Manuel Namici and Giuseppe De Giacomo. "Comparing Query Answering in OBDA Tools over W3C-Compliant Specifications". In: *Proceedings of the 31st International Workshop on Description Logics co-located with 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018), Tempe, Arizona, US, October 27th - to - 29th, 2018*. Ed. by Magdalena Ortiz and Thomas Schneider. Vol. 2211. CEUR Workshop Proceedings. CEUR-WS.org, 2018. URL: https://ceur-ws.org/Vol-2211/paper-25.pdf.

[10] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. "The MASTRO system for ontology-

based data access". In: *Semantic Web* 2.1 (2011), pp. 43–53. DOI: 10.3233/SW-2011-0029. URL: https://doi.org/10.3233/SW-2011-0029.

[11] Diego Calvanese, Benjamin Cogrel, Elem Guzel Kalayci, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. "OBDA with the Ontop Framework". In: *23rd Italian Symposium on Advanced Database Systems, SEBD 2015, Gaeta, Italy, June 14-17, 2015*. Ed. by Domenico Lembo, Riccardo Torlone, and Andrea Marrella. Curran Associates, Inc., 2015, pp. 296–303.

[12] Özgü Can, Emine Sezer, Okan Bursa, and Murat Osman Ünalir. "Comparing Relational and Ontological Triple Stores in Healthcare Domain". In: *Entropy* 19.1 (2017), p. 30. DOI: 10.3390/E19010030. URL: https://doi.org/10.3390/e19010030.

[13] Simon D. Harding, Jane F. Armstrong, Elena Faccenda, Christopher Southan, Stephen P. H. Alexander, Anthony P. Davenport, Adam J. Pawson, Michael Spedding, and Jamie A. Davies. "The IUPHAR/BPS guide to PHARMACOLOGY in 2022: curating pharmacology for COVID-19, malaria and antibacterials". In: *Nucleic Acids Res.* 50.D1 (2022), pp. 1282–1294. DOI: 10.1093/NAR/GKAB1010. URL: https://doi.org/10.1093/nar/gkab1010.

[14] David S. Wishart, Craig Knox, Anchi Guo, Savita Shrivastava, Murtaza Hassanali, Paul Stothard, Zhan Chang, and Jennifer Woolsey. "DrugBank: a comprehensive resource for *in silico* drug discovery and exploration". In: *Nucleic Acids Res.* 34.Database-Issue (2006), pp. 668–672. DOI: 10.1093/NAR/GKJ067. URL: https://doi.org/10.1093/nar/gkj067.

[15] Janet Piñero González, Àlex Bravo, Núria Queralt-Rosinach, Alba Gutiérrez-Sacristán, Jordi Deu-Pons, Emilio Centeno, Javier García-García, Ferran Sanz, and Laura I. Furlong. "DisGeNET: a comprehensive platform integrating information on human disease-associated genes and variants". In: *Nucleic Acids Res.* 45.Database-Issue (2017), pp. D833–D839. DOI: 10.1093/NAR/GKW943. URL: https://doi.org/10.1093/nar/gkw943.

[16] Marc Gillespie, Bijay Jassal, Ralf Stephan, Marija Milacic, Karen Rothfels, Andrea Senff-Ribeiro, Johannes Griss, Cristoffer Sevilla, Lisa Matthews, Chuqiao Gong, Chuan Deng, Thawfeek M. Varusai, Eliot Ragueneau, Yusra Haider, Bruce May, Veronica Shamovsky, Joel Weiser, Timothy Brunson, Nasim Sanati, Liam Beckman, Xiang Shao, Antonio Fabregat, Konstanti-

nos Sidiropoulos, Julieth Murillo, Guilherme Viteri, Justin Cook, Solomon Shorser, Gary D. Bader, Emek Demir, Chris Sander, Robin Haw, Guanming Wu, Lincoln Stein, Henning Hermjakob, and Peter D'Eustachio. "The reactome pathway knowledgebase 2022". In: *Nucleic Acids Res.* 50.D1 (2022), pp. 687–692. DOI: 10.1093/NAR/GKAB1028. URL: https://doi.org/10.1093/nar/gkab1028.

[17] Zhenzhen Gu, Diego Calvanese, Marco Di Panfilo, Davide Lanti, Alessandro Mosca, and Guohui Xiao. "OBDF: OBDA + Data Federation - Extended Abstract". In: *40th International Conference on Data Engineering, ICDE 2024 - Workshops, Utrecht, Netherlands, May 13-16, 2024*. IEEE, 2024, pp. 381–383. DOI: 10.1109/ICDEW61823.2024.00060. URL: https://doi.org/10.1109/ICDEW61823.2024.00060.

[18] Vijay Gadepally, Peinan Chen, Jennie Duggan, Aaron J. Elmore, Brandon Haynes, Jeremy Kepner, Samuel Madden, Tim Mattson, and Michael Stonebraker. "The BigDAWG polystore system and architecture". In: *2016 IEEE High Performance Extreme Computing Conference, HPEC 2016, Waltham, MA, USA, September 13-15, 2016*. IEEE, 2016, pp. 1–6. DOI: 10.1109/HPEC.2016.7761636. URL: https://doi.org/10.1109/HPEC.2016.7761636.

[19] Zuohao She, Surabhi Ravishankar, and Jennie Duggan. "BigDAWG polystore query optimization through semantic equivalences". In: *2016 IEEE High Performance Extreme Computing Conference, HPEC 2016, Waltham, MA, USA, September 13-15, 2016*. IEEE, 2016, pp. 1–6. DOI: 10.1109/HPEC.2016.7761584. URL: https://doi.org/10.1109/HPEC.2016.7761584.

[20] Aaron J. Elmore, Jennie Duggan, Mike Stonebraker, Magdalena Balazinska, Ugur Çetintemel, Vijay Gadepally, Jeffrey Heer, Bill Howe, Jeremy Kepner, Tim Kraska, Samuel Madden, David Maier, Timothy G. Mattson, Stavros Papadopoulos, Jeff Parkhurst, Nesime Tatbul, Manasi Vartak, and Stan Zdonik. "A Demonstration of the BigDAWG Polystore System". In: *Proc. VLDB Endow.* 8.12 (2015), pp. 1908–1911. DOI: 10.14778/2824032.2824098. URL: http://www.vldb.org/pvldb/vol8/p1908-Elmore.pdf.

[21] Matthew J. Mucklo. "Demonstration: API Federation in the BigDAWG Polystore". In: *Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB 2018 Workshops, Poly and DMAH, Rio de Janeiro, Brazil, August 31, 2018, Revised Selected Papers*. Ed. by Vijay Gadepally, Timothy G. Mattson, Michael Stonebraker, Fusheng Wang, Gang Luo, and George Teodoro.

Vol. 11470. Lecture Notes in Computer Science. Springer, 2018, pp. 93–103. DOI: 10.1007/978-3-030-14177-6_8. URL: https://doi.org/10.1007/978-3-030-14177-6%5C_8.

[22] Miltiadis D. Lytras, Evangelos Sakkopoulos, and Patricia Ordóñez de Pablos. "Semantic Web and Knowledge Management for the health domain: state of the art and challenges for the Seventh Framework Programme (FP7) of the European Union (2007-2013)". In: *Int. J. Technol. Manag.* 47.1/2/3 (2009), pp. 239–249. DOI: 10.1504/IJTM.2009.024124. URL: https://doi.org/10.1504/IJTM.2009.024124.

[23] Evgeny Kharlamov, Sebastian Brandt, Martin Giese, Ernesto Jiménez-Ruiz, Steffen Lamparter, Christian Neuenstadt, Özgür Lütfü Özçep, Christoph Pinkel, Ahmet Soylu, Dmitriy Zheleznyakov, Mikhail Roshchin, Stuart Watson, and Ian Horrocks. "Semantic Access to Siemens Streaming Data: the Optique Way". In: *Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC-2015), Bethlehem, PA, USA, October 11, 2015*. Ed. by Serena Villata, Jeff Z. Pan, and Mauro Dragoni. Vol. 1486. CEUR Workshop Proceedings. CEUR-WS.org, 2015. URL: https://ceur-ws.org/Vol-1486/paper%5C_23.pdf.

[24] Evgeny Kharlamov, Ernesto Jiménez-Ruiz, Dmitriy Zheleznyakov, Dimitris Bilidas, Martin Giese, Peter Haase, Ian Horrocks, Herald Kllapi, Manolis Koubarakis, Özgür L. Özçep, Mariano Rodriguez-Muro, Riccardo Rosati, Michael Schmidt, Rudolf Schlatte, Ahmet Soylu, and Arild Waaler. "Optique: Towards OBDA Systems for Industry". In: *The Semantic Web: ESWC 2013 Satellite Events - ESWC 2013 Satellite Events, Montpellier, France, May 26-30, 2013, Revised Selected Papers*. Ed. by Philipp Cimiano, Miriam Fernández, Vanessa López, Stefan Schlobach, and Johanna Völker. Vol. 7955. Lecture Notes in Computer Science. Springer, 2013, pp. 125–140. DOI: 10.1007/978-3-642-41242-4_11. URL: https://doi.org/10.1007/978-3-642-41242-4%5C_11.

[25] Ahmet Soylu, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ernesto Jiménez-Ruiz, Martin Giese, Martin G. Skjæveland, Dag Hovland, Rudolf Schlatte, Sebastian Brandt, Hallstein Lie, and Ian Horrocks. "OptiqueVQS: A visual query system over ontologies for industry". In: *Semantic Web* 9.5 (2018), pp. 627–660. DOI: 10.3233/SW-180293. URL: https://doi.org/10.3233/SW-180293.

[26] Marco Vogt, Alexander Stiemer, and Heiko Schuldt. "Polypheny-DB: Towards a Distributed and Self-Adaptive Polystore". In: *IEEE International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, December 10-13, 2018*. Ed. by Naoki Abe, Huan Liu, Calton Pu, Xiaohua Hu, Nesreen K. Ahmed, Mu Qiao, Yang Song, Donald Kossmann, Bing Liu, Kisung Lee, Jiliang Tang, Jingrui He, and Jeffrey S. Saltz. IEEE, 2018, pp. 3364–3373. DOI: 10.1109/BIGDATA.2018.8622353. URL: https://doi.org/10.1109/BigData.2018.8622353.

[27] Guglielmo Faggioli, Laura Menotti, Stefano Marchesin, Adriano Chiò, Arianna Dagliati, Mamede de Carvalho, Marta Gromicho, Umberto Manera, Eleonora Tavazzi, Giorgio Maria Di Nunzio, Gianmaria Silvello, and Nicola Ferro. "An Extensible and Unifying Approach to Retrospective Clinical Data Modeling: The BrainTeaser Ontology". In: *Journal of Biomedical Semantics* (2024). Accepted for publication.

[28] Christian Bizer and Andreas Schultz. "The Berlin SPARQL Benchmark". In: *Int. J. Semantic Web Inf. Syst.* 5.2 (2009), pp. 1–24. DOI: 10.4018/JSWIS.2009040101. URL: https://doi.org/10.4018/jswis.2009040101.

[29] Tommaso Dolci, Lorenzo Amata, Carlo Manco, Fabio Azzalini, Marco Gribaudo, and Letizia Tanca. "Tools for Healthcare Data Lake Infrastructure Benchmarking". In: *Information Systems Frontiers* (2024), pp. 1–22.

[30] Milos Chaloupka and Martin Necaský. "Using Berlin SPARQL benchmark to evaluate virtual SPARQL endpoints over relational databases". In: *Data Knowl. Eng.* 152 (2024), p. 102309. DOI: 10.1016/J.DATAK.2024.102309. URL: https://doi.org/10.1016/j.datak.2024.102309.

Acknowledgments

Ci tengo innanzitutto a ringraziare genitori e parenti tutti per avermi sostenuto e permesso di intrapredere questo percorso di vita.

Un ringraziamento particolare inoltre va rivolto a Beatrice, che in questi anni mi ha sempre sostenuto, supportato ma soprattutto sopportato.

Questo percorso ha posto davanti a me diversi ostacoli che