



UNIVERSITÀ
DEGLI STUDI
DI PADOVA



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

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

lorem ipsum

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	xi
List of Tables	xiii
List of Algorithms	xvii
List of Code Snippets	xvii
List of Acronyms	xix
1 Introduction	1
2 Background	3
2.1 Methodological Background	3
2.1.1 Resource Description Framework	3
2.1.2 Knowledge Graphs	4
2.1.3 Ontologies	4
2.1.4 Data Federation	5
2.1.5 Semantic Data Integration	6
2.2 Technical Background	6
2.2.1 SPARQL Query Language	6
2.2.2 Data Federation Systems	8
2.2.3 OBDA Systems	11
2.2.4 Triple Stores	13
3 Context of the Work	15
3.1 The HEREDITARY Project	16
3.2 Data Description	18
3.2.1 Clinical Data	18

CONTENTS

3.2.2	Genomics Data	19
4	Related Works	21
4.1	BigDAWG	22
4.2	Optique	23
5	System Architecture	25
5.1	System Architecture Design	25
5.2	Data Sources	27
5.2.1	MySQL	27
5.2.2	PostgreSQL	27
5.2.3	Polypheny	28
5.2.4	NAS Folders	31
5.2.5	Google Drive	32
5.3	Federation and Virtualization Layer	33
5.3.1	Role of Federator	33
5.3.2	Role of Virtualizator	34
5.3.3	Focus on Dremio	34
5.4	Ontology Layer: Brainteaser Ontology	36
5.4.1	Brainteaser Ontology Overview	36
5.4.2	Ontology Features and Capabilities	37
5.4.3	Ontology Implementation	38
5.5	OBDA Layer	38
5.5.1	Ontop Mapping	38
5.5.2	Ontop SPARQL	41
5.5.3	Ontop Endpoint	41
5.5.4	Ontop with GraphDB	41
6	Use Case	43
6.1	Experimental Setup	43
6.2	SPARQL Query over ALSFRS Relations	44
6.3	Query Flow	45
6.3.1	Query Rewriting over Ontological Rules	46
6.3.2	Query Unfolding to SQL	46
6.3.3	Execution of SQL on Virtual Views	46
6.4	Exploration of Query Transformation in Ontop	47
6.5	Execution of q^* on Dremio Virtual Views	50

CONTENTS

6.6 Summary	51
7 Evaluation Results	53
8 Conclusions and Future Works	55
References	57
Acknowledgments	61

List of Figures

2.1	The OBDA framework	7
2.2	The OBDA framework	12
4.1	OBDF approach	22
5.1	Proposed System Architecture	26
5.2	Virtual Views	34
5.3	The Brainteaser Ontology	37
6.1	Data Flow Block Diagram of a Query Through the Federated Architecture	47

List of Tables

2.1	Comparative among Data Federation Systems	11
2.2	Comparative among OBDA Systems	11

List of Algorithms

List of Code Snippets

2.1	Example of a SPARQL query for genetic markers data.	7
5.1	The storage plugin configuration file	29
5.2	Docker command to run a Dremio container with a Host Volume	31
5.3	google-drive-ocamlfuse Tool installation procedure	33
5.4	Virtual views over clinical data	35
5.5	Mappings definition between the virtual relational schema ex- posed by Dremio and the Brainteaser Ontology	39
6.1	SPARQL query on ALSFRS visits performed by patients over the BRAINTEASER ontology vocabulary	45
6.2	Parsed SQL Translation of the Original SPARQL Query	47

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

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

LIST OF CODE SNIPPETS

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

GDPR General Data Protection Regulation

WP Work Package

ALS Amyotrophic Lateral Sclerosis

ALSFRS Amyotrophic Lateral Sclerosis Functional Rating Scale



Introduction



Background

2.1 METHODOLOGICAL BACKGROUND

2.1.1 RESOURCE DESCRIPTION FRAMEWORK

The Resource Description Framework (RDF) is a standard¹ 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 datatypes, 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/>

2.1. METHODOLOGICAL BACKGROUND

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 Database Management System (DBMS). Users can query this virtual layer using standard data querying languages, like Structured Query Language (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

2.1. METHODOLOGICAL BACKGROUND

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 Ontology-Based Data Access (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 optimised 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.

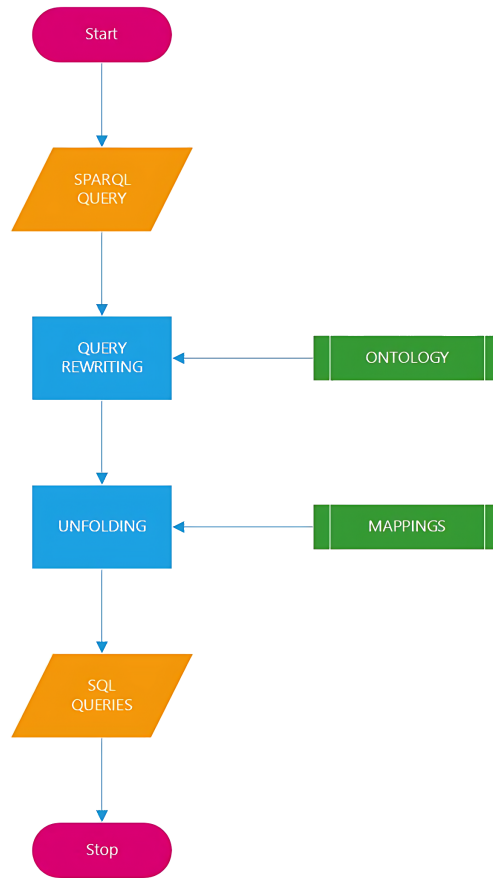


Figure 2.1: The OBDA framework

2.2 TECHNICAL BACKGROUND

2.2.1 SPARQL QUERY LANGUAGE

SPARQL Protocol and RDF 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

2.2. TECHNICAL BACKGROUND

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:BrinteaserALSTurin.

  OPTIONAL{
    ?p bto:tested ?FUS_test.
    ?FUS_test bto:hasMutation ?FUS;
              bto:onGene NCIT:C91852.
    OPTIONAL{
      ?FUS_test 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 softwares 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

Denodo² 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 (! (!)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

Teiid³ is an open-source Java component developed by Red Hat⁴ 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 Application Program Interface (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

²<https://denodo.com/en>

³<https://teiid.io/>

⁴<https://www.redhat.com/>

2.2. TECHNICAL BACKGROUND

OpenShift.

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 repository⁵ are not being addressed.

DREMIO

Similarly to Denodo, Dremio⁶ 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 repository⁷ provides a Maven template, that is customizable for each data source for which a JDBC driver is available.

⁵<https://github.com/teiid/teiid>

⁶<https://www.dremio.com/>

⁷<https://github.com/dremio-hub/dremio-sqlite-connector>

In conclusion, Dremio offers an open-source virtualization system with the typical robustness of enterprise softwares; 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 among 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 Siemens⁸ 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 among OBDA Systems

⁸<https://www.siemens.com/>

2.2. TECHNICAL BACKGROUND

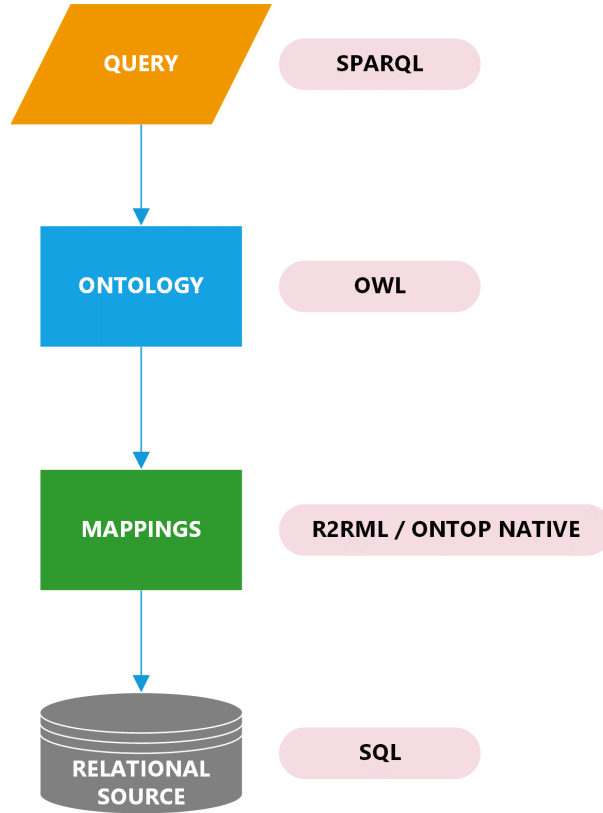


Figure 2.2: The OBDA framework

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 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

2.2. TECHNICAL BACKGROUND

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 Ontotext⁹, 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/>



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 meta-data 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 HEREDITARY¹, collaborating with three med-

¹<https://hereditary-project.eu/>

3.1. THE HEREDITARY PROJECT

ical centers that manage heterogeneous multimodal clinical and genomic data requiring integration. The HEteRogeneous sEmantic Data Integration for the guT-brAin inteRplaY (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

3.2. DATA DESCRIPTION

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 fa-

cilitates 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 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 (Amyotrophic Lateral Sclerosis) 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 syn-

3.2. DATA DESCRIPTION

thetic data is provided by the CINECA project², 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 DataSynthesizer³ 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.

4.1. BIGDAWG



Figure 4.1: OBDF approach

4.1 BigDAWG

As an example addressing research questions about managing heterogeneous data we have BigDAWG [18], developed under the Intel Science and Technology Center on Big Data. BigDAWG's architecture is composed of four layers: the base layer, the island layer, the main BigDAWG layer, and the application layer. Each layer serves a distinct purpose, from managing diverse physical data stores to facilitating user interaction through applications. This multi-layered approach allows BigDAWG to support various data models and query languages, thus offering a robust solution for cross-database queries and operations. While traditional systems such as Garlic and IBM DB2 have demonstrated the capability to handle data across different storage systems using a unified interface, BigDAWG distinguishes itself by its comprehensive support for "islands" of different data

models. This island approach not only supports operations across various data types but also enhances performance by optimizing queries based on the data model and the underlying database engine. This feature is critical in environments where performance and response times are crucial, such as in medical or real-time analytics applications. Despite its advanced architecture and capabilities, BigDAWG is not without challenges. It is not available as an off-the-shelf solution; rather, it requires certain levels of expertise if someone intends to install it or, in an even worse scenario, there exists the need to develop new islands modules. This means that significant effort would be required to adapt it to specific operational needs. Additionally, there is limited literature and empirical studies on its deployment in real-world scenarios, which poses doubts about considering its implementation. In summary, while BigDAWG represents a significant advancement in the field of database management systems, its practical application is limited by the prototype nature of its current implementation and the lack of extensive real-world testing. Future research could focus on reducing the barrier to its adoption, refining its architecture based on operational feedback, and expanding its use cases across different scenarios to fully realize its potential.

4.2 OPTIQUE

Optique [19], developed by Siemens, exemplifies an implementation of OBDA tailored to the industrial applications. Unlike conventional OBDA systems which typically focus on static data, Optique is designed specifically to handle both streaming and static data. This dual capability is crucial for environments like Siemens, where real-time data from sensors needs to be integrated with historical data for comprehensive analysis and monitoring. Optique comprises many technologies: Streaming and Temporal ontology Access with a Reasoning-based Query Language (STARQL), A language that supports complex queries over streaming and static data at the same time; ExaStream, A backend system optimized for low-latency queries on high-velocity streams; OptiqueVQS (Visual Query System), a component that improves usability by enabling users to formulate queries without prior knowledge of query languages. Although it has advanced capabilities, Optique remains a proprietary system not available for public use, which limits its adoption outside Siemens. This exclusivity may impede broader validation and benchmarking against other OBDA systems in real-

4.2. OPTIQUE

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-

5.1. SYSTEM ARCHITECTURE DESIGN

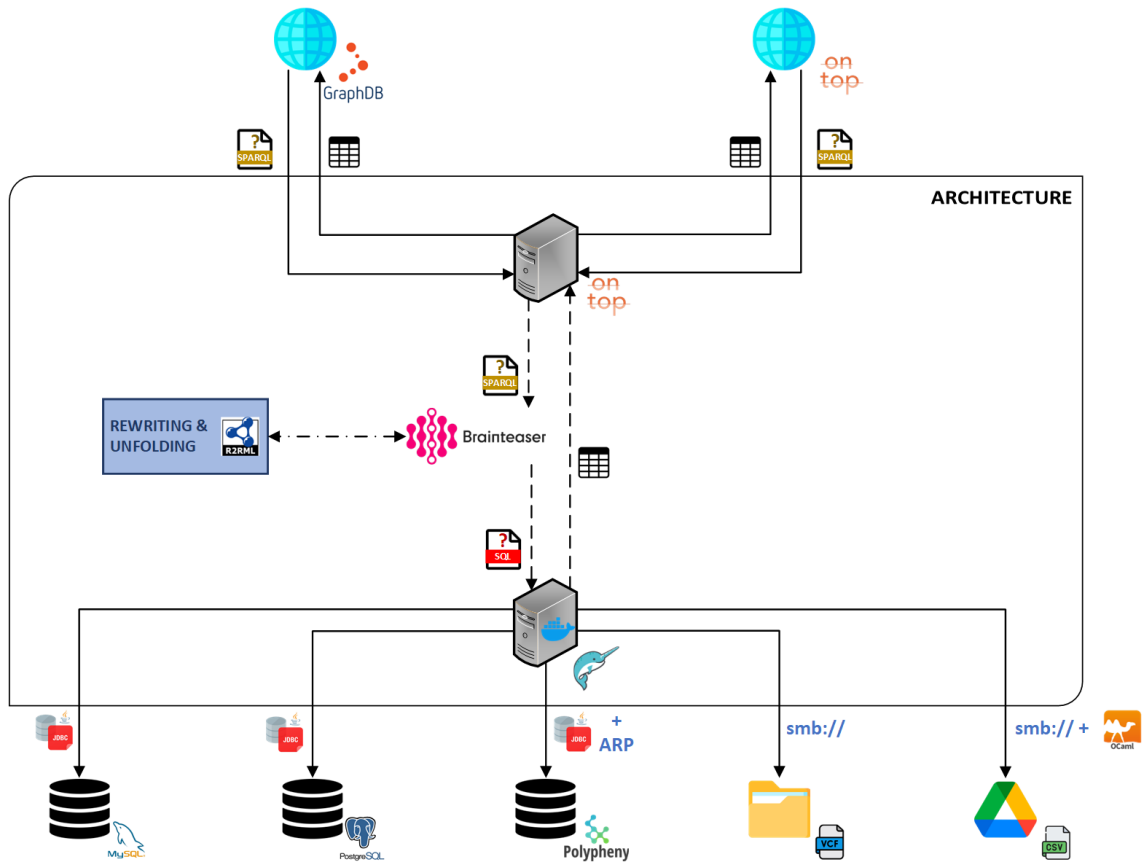


Figure 5.1: Proposed System Architecture

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 ontology¹ as suitable for accomplishing this task.

¹<https://brainteaser.dei.unipd.it/ontology/>

5.2 DATA SOURCES

Given the relational nature of available data, we distributed it among five different platforms, so 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 encompass 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

MySQL² is one of the most widely used relational DBMS (DBMS) in the world. It is a FOSS solution now distributed by Oracle³. 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 softwares 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

PostgreSQL⁴ 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/>

5.2. DATA SOURCES

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 [20] 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 Advanced Relational Pushdown (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 repository⁵. 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 template⁶ consists of a Java Maven project, that has to be compiled, packed within a Java Archive (JAR) file and injected, together with the target source JDBC driver, into the running Dremio instance. To customize the template, two files need to be customized: the storage plugin configuration, which is a Java class 5.1, and the plugin ARP file, which is a YAML⁷ 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.

```

1  /* ... */
2
3  @SourceType(value = "POLYPHENY", label = "POLYPHENY")
4  public class PolyphenyConf extends AbstractArpConf<PolyphenyConf> {
5      private static final String ARP_FILENAME = "arp/implementation/
        polypheny-arp.yaml";
6      private static final ArpDialect ARP_DIALECT =
7          AbstractArpConf.loadArpFile(ARP_FILENAME, (ArpDialect::new));
8      private static final String DRIVER = "org.polypheny.jdbc.Driver";
9
10     @NotBlank
11     @Tag(1)
12     @DisplayMetadata(label = "Polypheny host <HOST:PORT>")
13     public String database;
14
15     /* ... */
16
17     @Tag(2)
18     @DisplayMetadata(label = "Record fetch size")
19     @NotMetadataImpacting
20     public int fetchSize = 200;

```

⁶<https://github.com/dremio-hub/dremio-sqlite-connector>

⁷<https://yaml.org/>

5.2. DATA SOURCES

```
21
22 @Tag(6)
23 @DisplayMetadata(label = "Polypheny User Name")
24 public String username = "pa";
25
26 @Tag(7)
27 @DisplayMetadata(label = "Polypheny Password")
28 public String password = "";
29
30 @Tag(4)
31 @DisplayMetadata(label = "Maximum idle connections")
32 @NotMetadataImpacting
33 public int maxIdleConns = 8;
34
35 @Tag(5)
36 @DisplayMetadata(label = "Connection idle time (s)")
37 @NotMetadataImpacting
38 public int idleTimeSec = 60;
39
40 @VisibleForTesting
41 public String toJdbcConnectionString() {
42     final String database = checkNotNull(this.database, "Missing
43     database.");
44     return String.format("jdbc:polypheny://%s", database);
45 }
46
47
48 private CloseableDataSource newDataSource() {
49     return DataSources.newGenericConnectionPoolDataSource(DRIVER,
50         toJdbcConnectionString(), username, password, null, DataSources
51         .CommitMode.DRIVER_SPECIFIED_COMMIT_MODE,
52         maxIdleConns, idleTimeSec);
53 }
```

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` annota-

tion. 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 Volumes⁸, and in particular the “Host Volume” paradigm.

```
1 $ 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 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. Filesystem in Userspace (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.

```

1 # Installation of google-drive-ocamlfuse
2 $ sudo add-apt-repository ppa:alessandro-strada/ppa
3 $ sudo apt-get update
4 $ sudo apt-get install google-drive-ocamlfuse
5
6 # Authentication and mounting
7 $ google-drive-ocamlfuse
8 $ mkdir ~/google-drive
9 $ 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 virtualizator 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 VIRTUALIZATOR

On the virtualization side, the layer enables the creation of virtual views that do not require materialization. These views treat 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.

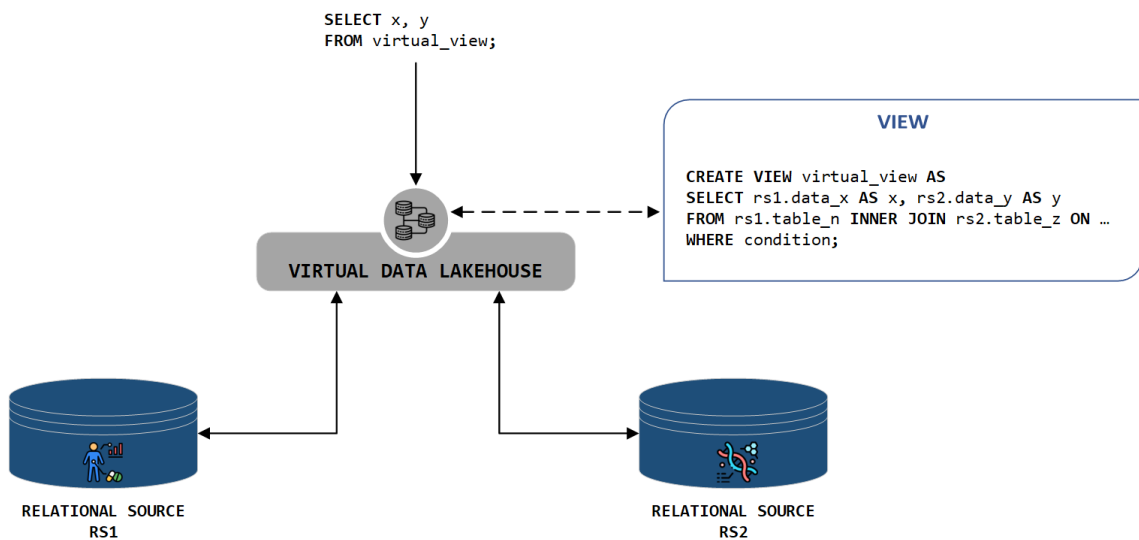


Figure 5.2: Virtual Views

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.

```

1  # Amyotrophic Lateral Sclerosis View
2  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,
```

5.4. ONTOLOGY LAYER: BRAINTEASER ONTOLOGY

```

3      NULL AS q2, NULL AS q3, NULL AS q4, NULL AS q5, NULL AS q6, NULL
4      AS q7, NULL AS q8, NULL AS q9, NULL AS q10, NULL AS q11, NULL AS
5      q12 FROM MADRID.madrid_alsfrs;
6
7  # Spirometry Forced Vital Capacity View
8  SELECT * FROM LISBON.public.lisbon_spiro UNION SELECT * FROM MADRID
   .madrid_alsfrs.Madrid_spiro UNION SELECT * FROM TURIN.alsfrs_turin
   .Turin_spiro;
6
7  # Patient Static Data
8  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.

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 website¹⁰.

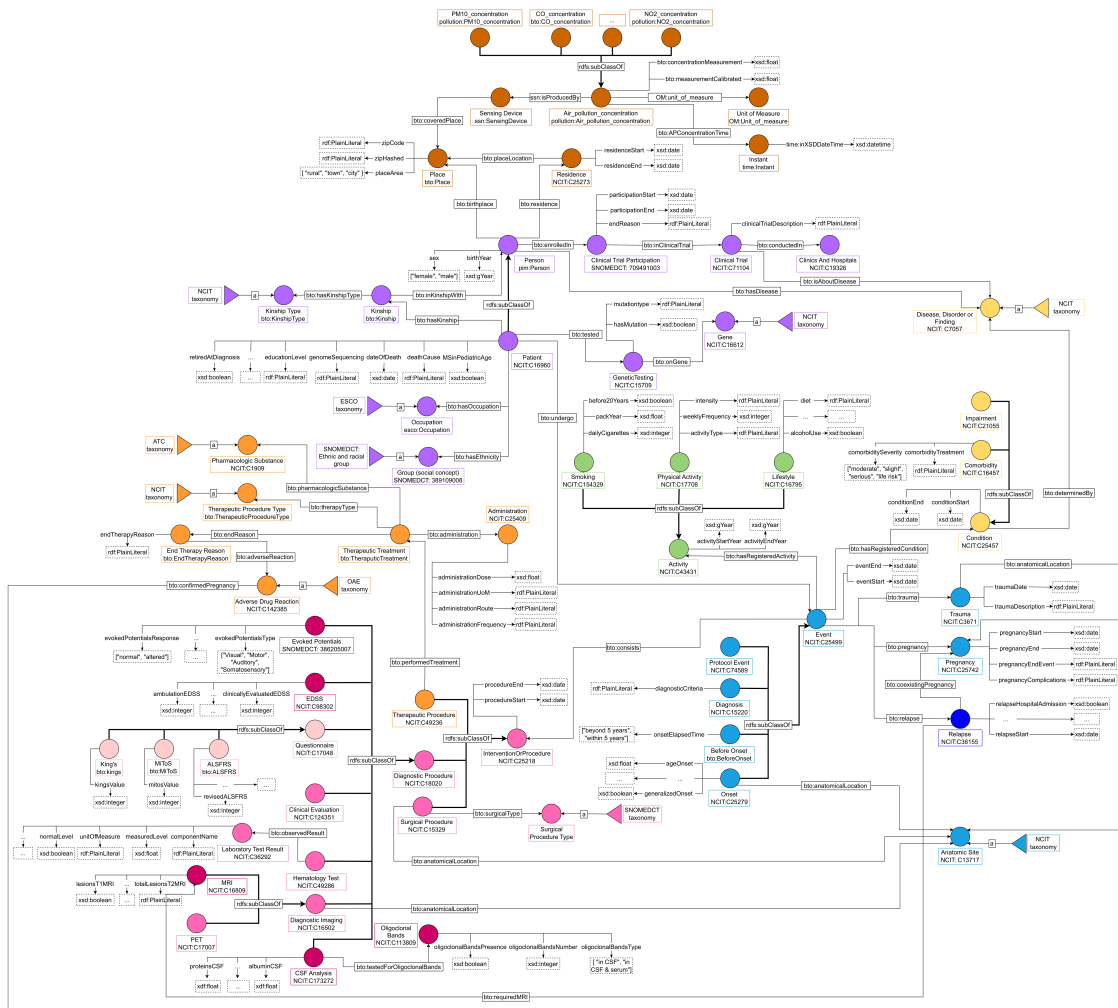


Figure 5.3: The Brainteaser Ontology

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 interoperability. It is built to integrate seamlessly with existing medical and genomic data

¹⁰<https://brainteaser.dei.unipd.it/ontology/>

5.5. OBDA LAYER

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
```

5.5. OBDA LAYER

```
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

5.5.2 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.

5.5.3 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.

5.5.4 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.



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 BRAIN-TEASER 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 pro-

6.2. SPARQL QUERY OVER ALSFRS RELATIONS

vide 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 Amyotrophic Lateral Sclerosis Functional Rating Scale (ALSFRS) data, which is crucial for assessing the progression of Amyotrophic Lateral Sclerosis (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 laterally 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 des-

6.3. QUERY FLOW

tinations (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 database management system (RDBMS). 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.

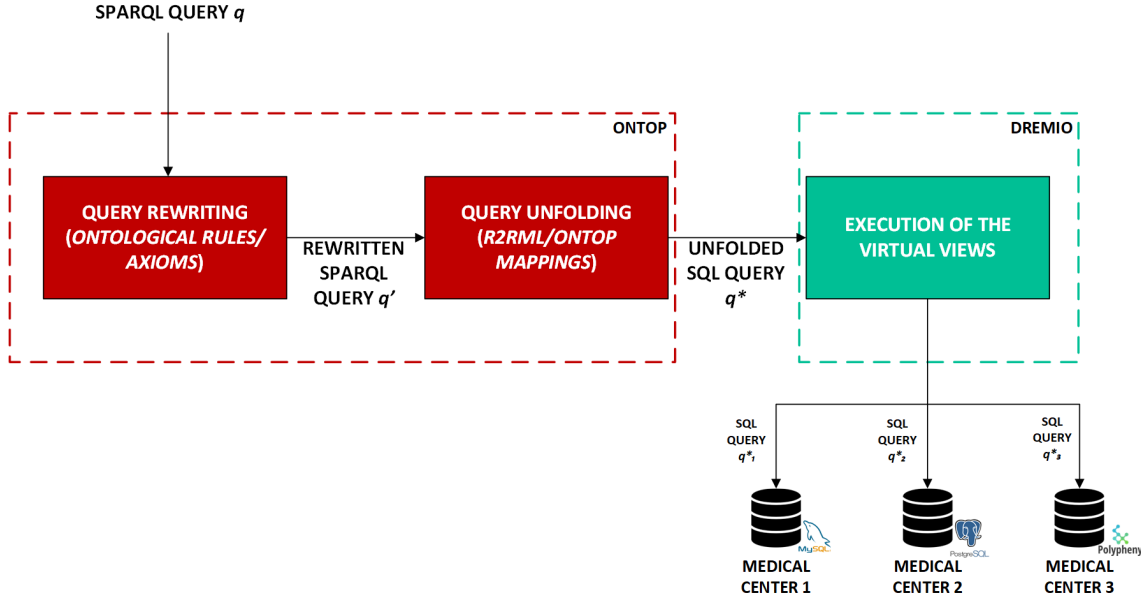


Figure 6.1: Data Flow Block Diagram of a Query Through the Federated Architecture

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 due to, in particular, to special characters 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.

6.4. EXPLORATION OF QUERY TRANSFORMATION IN ONTOP

```
1 CONSTRUCT [patient, date, tot, bulbar, motor, respiratory, q1, q2, q3
2   , q4, q5, q6, q7, q8, q9, q10, q11, q12]
3   [date/RDF(CHARACTER VARYINGToVARCHAR(date2m51), xsd:datetime),
4     q1/RDF(INTEGERToVARCHAR(q11m25), xsd:integer),
5     motor/RDF(INTEGERToVARCHAR(motor1m55), xsd:integer),
6     ...]
7 NATIVE
8 SELECT
9   v23."bulbar1m17" AS "bulbar",
10  v23."date2m51" AS "date",
11  v23."motor1m55" AS "motor",
12  v23."q101m44" AS "q10",
13  v23."q111m42" AS "q11",
14  v23."q11m25" AS "q1",
15  v23."q121m41" AS "q12",
16  v23."q21m24" AS "q2",
17  v23."q31m23" AS "q3",
18
19  --- ... more fields
20
21  v23."patient26m9" AS "patient"
22 FROM (
23   SELECT DISTINCT
24     v7."bulbar" AS "bulbar1m17",
25     v5."date2m51" AS "date2m51",
26     v8."motor" AS "motor1m55",
27     v1."patient" AS "patient26m9",
28     v19."q10" AS "q101m44",
29     v20."q11" AS "q111m42",
30     v10."q1" AS "q11m25",
31
32     --- ... more fields
33
34     v6."tot" AS "tot1m45"
35   FROM "clinical_data"."ALSFRS" v1
36
37   --- ... recursive joins
38
39   WHERE v1."patient" IS NOT NULL AND v1."date" IS NOT NULL
40 ) v23
41 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.



Evaluation Results



Conclusions and Future Works

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 et al. “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 et al. “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 et al. “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 et al. 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.

REFERENCES

- [7] Konstantinos Makris et al. "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 et al. "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 et al. 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 et al. "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 et al. "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 et al. "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 et al. “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 et al. “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 et al. “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 et al. “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 et al. “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 et al. “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] Evgeny Kharlamov et al. “Semantically-enhanced rule-based diagnostics for industrial Internet of Things: The SDRL language and case study for Siemens trains and turbines”. In: *J. Web Semant.* 56 (2019), pp. 11–29. DOI: 10.1016/J.WEBSEM.2018.10.004. URL: <https://doi.org/10.1016/j.websem.2018.10.004>.
- [20] 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 et al. IEEE, 2018, pp. 3364–3373. DOI: 10.1109/

REFERENCES

BIGDATA.2018.8622353. URL: <https://doi.org/10.1109/BigData.2018.8622353>.

Acknowledgments