Unlocking Biomedical Data for AI Health Research in Africa Using GeneNetwork

by

Bonface Munyoki Kilyungi

070707

Submitted in Partial fulfilment of the Requirements for the Degree of Master of Science in Data Science and Analytics at Strathmore University

Institute of Mathematical Sciences

Strathmore University

Nairobi, Kenya

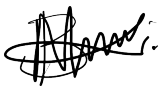
June, 2025

Declaration and Approval

Declaration

I declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of my knowledge and belief, the dissertation contains no material previously published or written by another person except where due reference is made in the dissertation itself.

Student's Name: **Bonface Munyoki Kilyungi**

Sign:  Date: 24/04/2025

Approval

The dissertation of Bonface Munyoki Kilyungi was reviewed and approved by the following:

Dr. Joseph Sevilla,
Senior Lecturer,
Strathmore University.

Dr. Godfrey Achono Madigu,
Dean, Institute of Mathematical Sciences,
Strathmore University.

Prof. Bernard Shibwabo,
Director of Graduate Studies,
Strathmore University.

Abstract

Genetic data analysis is essential for understanding biological processes and diseases. GeneNetwork (GN), an open-source platform with over 20 years of genetic and phenotypic data, relies on a complex relational database. However, the data is currently difficult to access and manipulate due to its complex underlying structures, including around 80 cross-referenced Structured Query Language (SQL) tables and various file types. This dissertation aimed to address the limitations of the GeneNetwork2 SQL database in representing and querying graph-like biological data by transforming it into the Resource Description Framework (RDF). A self-documenting Domain Specific Language (DSL) was developed using GNU Guile to automate the conversion of GN's MariaDB SQL database into RDF triples. This involved defining ontologies, mapping SQL views to RDF, and storing the data in Virtuoso. The framework's effectiveness was evaluated by comparing query performance and output quality between SQL and SPARQL. Results showed that RDF transformation significantly improved query efficiency and semantic richness. At a 99.9% confidence level, SPARQL queries exhibit statistically significant faster execution times than the equivalent SQL queries. Additionally, RDF's structured representation enabled intuitive querying and better relationship discovery, as demonstrated in retrieving mouse species details and searching GeneRIF entries. In conclusion, transforming GN's data into RDF made complex queries faster and enhanced its FAIR (Findable, Accessible, Interoperable, Reusable) properties, improving accessibility through semantic enrichment and interoperability with federated services for both human and machine agents. This transformation unlocks the full potential of the data, laying the groundwork for a more adaptable, AI-ready GN service and providing valuable insights for the broader application of RDF in biological and clinical data integration.

KEYWORDS: *Artificial Intelligence, Data Accessibility, Data Interpretation, GeneNetwork, Biological Data, Data Discovery, Resource Description Framework (RDF), Metadata*

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	**A**rtificial **I**ntelligence
API	**A**pplication **P**rogramming **I**nterface
DBMS	**D**atabase **M**anagement **S**ystem
DSL	**D**omain **S**pecific **L**anguage
EMBL	**E**uropean **M**olecular **B**iology **L**aboratory
FAIR	**F**indable **A**ccessible **I**nteroperable **R**eusable
GN	**G**ene**N**etwork
GO	**G**ene **O**ntology
GWAS	**G**enome-**W**ide **A**ssociation **S**tudy
HPO	**H**uman **P**henotype **O**ntology
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
LLM	**L**arge**L**anguage **M**odel
NCBI	**N**ational **C**enter for **B**iotechnology **I**nformation
QTL	**Q**uantitative **T**rait **L**ocus
RAD	**R**apid **A**pplication **D**evelopment
RDBMS	**R**elational **D**atabase **M**anagement **S**ystem
RDF	**R**esource **D**escription **F**ramework
SPARQL	**S**PARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage
SQL	**S**tructured **Q**uery **L**anguage
URI	**U**niform **R**esource **I**dentifier
W3C	**W**orld Wide Web **C**onsortium

Acknowledgements

I could never have reached this summit alone. Along the journey, many people have lifted me, and I am forever grateful for their unwavering support.

To my supervisors—Dr. Joseph Sevilla, Dr. Pjotr Prins, Dr. Shelby Solomon, and Dr. George Githinji—your wisdom was the compass that guided me through uncharted waters. Your insights were priceless, and your belief in me unfaltering.

A special note of gratitude goes to Mrs. Laura Mideva, whose steady hand helped me navigate Strathmore University with grace. You gave me the confidence to see this course through—may this be a bouquet of thanks, blooming in your honor.

To my brother and sister, John Nduli and Purity Mawea, and to the cherished souls in my community—Olive Murage, Joy Mwangi, Njenga Kabiru, Steve Oduor, Ikambili Kihoro, Yvonne Tamba, Ian Macharia, and Michael Kiruthi—your faith in me was my anchor. When my spirit faltered, your encouragement kept me climbing. Together, we've reached the mountaintop, and I eagerly await the moment when we can celebrate this triumph as one.

To Faiyaz Rahemtulla. Your music carried me through trying times.

To Data Science Africa, notably Dr. Ciira Maina, I extend my heartfelt gratitude for the generous grant that supported this work. Thank you for introducing me to the broader Data Science community here in Africa and for exposing me to the challenges surrounding data sharing, which I hope these pages provide meaningful insights to help alleviate.

To the GeneNetwork team, my deep gratitude. Frederick Muriithi, your work ethic and problem-solving are unmatched. Zachary Sloan, your patience and support has been graceful. Robert Williams, without you, GeneNetwork wouldn't exist—now grown, it thrives under our care. Arun Issac, your wisdom in "Scheme-y" ways shines through. Alexander Kabua, you unknowingly make me a better engineer. Shelby Solomon, your family brings me joy. Pjotr Prins, your vision through free software made this possible—we need more like you. John Nduli, your technical leadership stands tall. Thank you, GeneNetwork.

Dedicated to all the open source developers out there who fight for our freedoms through their efforts

Chapter 1

Introduction

1.1 Background

Genetic data analysis is important for understanding the underlying mechanisms of various biological processes and diseases. GeneNetwork2 ("Genenetwork", 2022) is a powerful open-source software platform that provides a range of tools and resources for analysing, visualising and storing genetic data, mostly written in the python programming language with a SQL database and flat files as a back-end (Sloan et al., 2016; Mulligan, Mozhui, Prins, and Williams, 2017). It contains over 20 years of experimental data from genetics, phenotyping, QTL and GWA studies in human and model species such as mouse and rat (Sloan et al., 2016). The size of the data is about 1 TB and growing about 20% a year. The current SQL database underlying GN can be complex and difficult to write software against, and lacks interoperability with other systems.

The Resource Description Framework (RDF) is a widely-used graph based semantic data model that can provide several benefits over SQL, including support for complex data types, reasoning and ontologies (Allemang & Hendler, 2011; Candan, Liu, & Suvarna, 2001). Representing GN's data in RDF has the potential to enable more flexible querying, data integration and reasoning capabilities, as well as better support for Linked Data and Semantic Web standards. RDF is particularly suitable for machine learning and AI because it allows machines/software to analyse and discover the structure of the data and to start reasoning on it without human intervention. So, by providing RDF, GeneNetwork2 data was made easily accessible

for AI. This dissertation made the data available for AI and made recommendations for future use of RDF in AI in a biological/clinical context.

1.2 Problem Statement

The current SQL database of GeneNetwork2 has several limitations, such as its lack of support for complex data types, reasoning and ontologies. These limitations make it difficult to integrate and query GeneNetwork2's data with other sources and to perform advanced analysis tasks, such as inferencing and querying by example.

1.3 General Objectives

The research problem is to transform GeneNetwork2's complex and non-interoperable SQL database into a graph-based RDF representation in order to overcome limitations in data integration, querying, and reasoning. Correspondingly, the main research question is: "Can GeneNetwork2's relational SQL database be effectively ported to RDF to enhance data interoperability, query performance and semantic reasoning, while maintaining output quality?" This dissertation aims to port GeneNetwork2's SQL database to RDF, and to evaluate the benefits and challenges of this approach.

1.4 Research Objectives

1. Examine previous research conducted on the GeneNetwork platform, SQL and Resource Description Framework (RDF) in order to identify any gaps that hinder effective data integration and semantic interoperability.

2. Design, implement and test a framework for the porting process, including steps, tools and resources, for porting the GeneNetwork2 database to RDF.

3. Validate the research by evaluating the results of query performance and output quality between SQL and SPARQL.

1.5 Research Questions

1. How does the SQL-based GeneNetwork platform limit data integration or semantic interoperability, and how can RDF address these gaps?

2. How can we design, implement and test a framework for porting the GeneNetwork2 database to RDF?

3. In the context of GeneNetwork2, how does the performance of RDF compare against SQL?

1.6 Scope and Limitations

The University of Tennessee Health Science Center provided the dataset used in this dissertation. Furthermore, access to one of their servers, capable of handling large databases at scale, facilitated the work conducted in this dissertation. Finally, this dissertation did not address any other issues or limitations related to GeneNetwork2 beyond the current SQL database.

1.7 Significance of the Study

This dissertation aimed to port GeneNetwork2's SQL database to RDF and highlight RDF's advantages over SQL's for graph-like data. A key contribution is the development of a domain-specific language (DSL) designed to automate the conversion of SQL views into RDF triples. Notably, this tool is database-agnostic, enabling its application beyond the GN ecosystem as a general SQL-to-RDF parser.

Through this transformation, the dissertation demonstrated that SPARQL queries consistently outperform SQL queries for complex, graph-like data, with statistically significant improvements in efficiency. Additionally, the shift to RDF enhanced the dataset's adherence to FAIR principles. This FAIR-ification not only streamlines data sharing but also lays the groundwork for AI-driven applications, such as Retrieval-Augmented Generation (RAG) models. FAIR data is crucial because it ensures datasets are more easily discoverable, accessible, interoperable, and reusable, reducing the barriers to collaboration and data sharing across disciplines. This is

particularly important in fields like bioinformatics, where integrating diverse datasets can drive new insights and accelerate scientific discovery.

Chapter 2

Literature Review

2.1 GeneNetwork

System genetics is a field of genetics that aims to understand how the interactions between genes, the environment, and other factors contribute to an organism's traits and characteristics. GeneNetwork ("Genenetwork", 2022) is a continuously updated web service for systems genetics analyses that allows researchers to analyse genetic data and identify patterns of covariation between different traits and genes (Mulligan et al., 2017). It is a toolbox that allows for the analysis of genomic data in order to identify genetic associations with complex traits. The tools has been successfully used in a number of studies, such as it's use with the extended BXD family of mice cohort in experimental systems genetics and precision medicine, as discussed in (Ashbrook et al., 2019). Additionally, the GeneNetwork web platform has been used for web-based genetic analyses as discussed in (Sloan et al., 2016). GN also allows for the upload of high-throughput experimental data, including genomic expression data obtained through microarray and RNA-sequencing technologies, as well as classical phenotypic data, such as disease-related traits (Anderson et al., 2021).

Currently, GN is scripted in GNU Guile, Python and Javascript. To guarantee bit-for-bit reproducibility and easy installation, the GN platform is packaged and deployed using GNU Guix (Sloan et al., 2016).

While GN is a powerful platform for data integration and analysis, it has one major limitation in that it relies on a SQL database to retrieve metadata for specific

experiments. This metadata includes data from the PubMed database[1], which is a widely used and comprehensive resource for scientific literature. However, using PubMed as a source of metadata for GeneNetwork may not provide the complete context for the relationships between the metadata and the data itself. As shown with GeneCup, a tool for mining PubMed and the Genome Wide Association Study (GWAS) catalog for gene-keyword relationships, using the PubMed metadata without storing any of the relationships may not capture the full context of the relationships it identifies, potentially limiting the usefulness of the tool (Gunturkun et al., 2022).

2.2 Linked Data and Semantic Web Standards

Linked Data is a set of best practices for publishing and interlinking structured data on the Web using an ontology (Bizer, Heath, & Berners-Lee, 2011). It is based on the idea of using standard Web technologies, such as HTTP and RDF, to represent and link data in a way that is both human-readable and machine-readable. Linked Data enables data from different sources to be connected and accessed in a decentralised manner, enabling data re-use and integration across a wide range of applications and domains (Bizer, Heath, & Berners-Lee, 2011).

Lee et al. (2006) proposed a set of principles for publishing data on the web in a way that enables data from different sources to be integrated into one single data space (Bizer, Heath, & Berners-Lee, 2011). The rules/principles have become known as the *"Linked Data principle"* and they are: (a) Use URIs as names for things; (b) Use HTTP URIs so that people can look up those names; (c) When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL); and (d) include links to other URIs, so that they can discover more things (Bizer, Heath, and Berners-Lee, 2011; Lee et al., 2006).

The Semantic Web is a vision for the Web in which data is represented in a machine-readable form, enabling automated processing of Web content. It is about making links, so that a person or machine can explore the web of data (Bizer, Heath, & Berners-Lee, 2011). The Semantic Web relies on a set of standards, such as RDF

[1] https://www.ncbi.nlm.nih.gov/pubmed

and OWL, to represent and exchange data in a way that is both expressive and interoperable. These standards provide a common vocabulary for data representation and enable the use of reasoning and inferencing to derive new knowledge from existing data.

The Semantic Web has several key properties which are important during application development. One notable characteristic of the semantic web is its strict separation of data from formatting and presentational aspects. This allows applications using Linked Data to handle unfamiliar vocabularies by dereferencing the URIs tha identify aforementioned vocabularies to find their definitions. Moreover, using HTTP as a standardised means for data retrieval and RDF as standardised data model simplifies data access when compared to Web APIs, which rely on heterogenous data models and access interfaces (Bizer, Heath, & Berners-Lee, 2011). Furthermore, due to the open nature of the Semantic Web, applications are not constrained to a fixed set of data (Bizer, Heath, & Berners-Lee, 2011).

In addition to their use in specific applications, Linked Data and Semantic Web standards have also been adopted as a general approach for improving data interoperability and enabling data integration. For example, the Linked Data principles have been widely adopted by government agencies and other organizations as a way to publish and share data on the Web (Bizer, Heath, & Berners-Lee, 2011). The use of Semantic Web standards has also been endorsed by the World Wide Web Consortium (W3C) as a way to enable the creation of a more interoperable and connected Web (Shadbolt, Berners-Lee, & Hall, 2006).

2.3 Resource Description Framework

The Resource Description Framework (RDF) is a widely-used semantic data model that is designed to represent and exchange information about resources on the World Wide Web (Raimond & Schreiber, 2014). It is based on the idea of representing data as a set of triples, where each triple consists of a subject, predicate and object. The subject represents the resource being described. The predicate represents the property of the resource. Finally, the object represents the value of the property (Raimond & Schreiber, 2014).

RDF has several benefits for data modelling and representation, including its ability to support complex data types, reasoning, and ontologies (Raimond & Schreiber, 2014). One of the key benefits of RDF is its support for complex data types, such as lists, sets, and graphs. This allows RDF to represent more complex relationships between resources, which is not possible with traditional data models such as relational databases (Allemang & Hendler, 2011).

Another benefit of RDF is its support for reasoning, which is the process of inferring new information from existing data. Reasoning can be used to make inferences about resources and their relationships, which can be particularly useful for data integration and querying (Allemang & Hendler, 2011).

RDF also supports the use of ontologies, which are formal definitions of the concepts and relationships in a domain. Ontologies provide a common vocabulary for data representation, which can improve data interoperability and facilitate data integration (Heath & Bizer, 2011).

One example of the use of RDF in genetics is the Gene Ontology (GO) project, which uses RDF to represent gene annotations and their relationships (Harris et al., 2004). The GO project has been widely adopted as a standard for gene annotation and is used by numerous databases and resources, including the European Molecular Biology Laboratory (EMBL) and the National Center for Biotechnology Information (NCBI).

Another example of the use of RDF in genetics is the Human Phenotype Ontology (HPO), which uses RDF to represent human phenotypes and their relationships (Robinson & Mundlos, 2010). The HPO is a widely-used resource for annotating human phenotypes and has been used in numerous studies to support the analysis and interpretation of genetic data.

In the field of biomedical research, RDF has been used to represent and integrate data from multiple sources, including clinical trials and electronic health records (Luz, de Matos Nogueira, Cavalini, & Cook, 2015). For example, a study published in the journal Health Informatics Journal used RDF to represent and integrate data from multiple sources, including clinical trials and electronic health records (Luz, de Matos Nogueira, Cavalini, & Cook, 2015). The study demonstrated the feasibility of using RDF to represent and integrate complex data from multiple sources,

and showed that RDF can support the integration and analysis of data from diverse domains.

There have also been numerous studies that have demonstrated the benefits of RDF for representing and integrating data in other domains, including biology, social media and the Semantic Web (Allemang & Hendler, 2011; Heath & Bizer, 2011; Raimond & Schreiber, 2014). These studies have shown that RDF can support the representation and integration of complex data from multiple sources, and can facilitate the use of reasoning and inferencing to derive new knowledge from existing data.

2.4 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) is a query language designed to retrieve and manipulate data stored in Resource Description Framework (RDF) format (Seaborne & Prud'hommeaux, 2008). It has become a popular choice for querying and integrating data from multiple sources, as it allows for the representation and integration of data from diverse domains, and the querying of that data using a single language.

One of the main advantages of SPARQL is its ability to query data from multiple sources and to combine data from those sources in a single query. This is made possible through the use of federated queries, which allow SPARQL to query multiple RDF graphs or datasets in a single query (Allemang and Hendler, 2011; Rakhmawati et al., 2013). This enables users to easily retrieve and integrate data from multiple sources, such as databases, ontologies and web services, without the need to manually combine the data in a separate step.

Another advantage of SPARQL is its ability to handle complex queries and to perform operations such as aggregations and data transformation. SPARQL includes a range of built-in functions and operators, as well as support for user-defined functions, which allow users to perform advanced operations on their data. Additionally, SPARQL supports the use of subqueries, which allows users to nest queries within other queries, enabling the creation of highly sophisticated queries (Allemang & Hendler, 2011).

According to Allemang and Hendler (2011), SPARQL has been widely adopted in a variety of domains, including the Semantic Web, life sciences and geospatial data management. In the Semantic Web, SPARQL is often used to query linked open data sources, such as DBpedia and Wikidata, to retrieve and integrate data from multiple sources. In the life sciences, SPARQL has been used to query biomedical ontologies and to integrate data from multiple sources, such as clinical trials and electronic health records. In the geospatial domain, SPARQL has been used to query and integrate geospatial data from various sources, such as spatial databases and web services.

There are several implementations of SPARQL, including standalone servers, libraries for various programming languages and web-based tools ("Sparql Implementations", 2022). Some popular SPARQL implementations include Apache Jena, OpenRDF Sesame, and Virtuoso.

2.5 SQL Databases and their Limitations

SQL databases are a type of database management system (DBMS) that are widely used for storing and manipulating structured data (Ramez Elmasr, 2016). It organises this data into tables that comprise rows and columns which allows for efficient querying and data manipulation (Ramez Elmasr, 2016). However, SQL databases have certain limitations that can impact their performance and scalability (Chen & Zhang, 2014).

One limitation of SQL databases is that they are not well-suited for storing large volumes of unstructured data, such as text documents or images (Chen & Zhang, 2014). While it is possible to store unstructured data in a SQL database, it requires additional processing and can be inefficient compared to using a database specifically designed for unstructured data, such as a NoSQL database (Kleppmann, 2019).

Another limitation of SQL databases is that they do not scale well when dealing with large amounts of data or high levels of concurrency (Chen & Zhang, 2014; Kleppmann, 2019). In order to scale a SQL database, it is necessary to partition the data across multiple servers or use techniques such as sharding, which can be complex and time-consuming to implement (Ramez Elmasr, 2016).

SQL databases can also be limited in their ability to handle real-time data processing and analytics (Chen & Zhang, 2014; Kleppmann, 2019). In order to perform real-time analysis on data stored in a SQL database, it is necessary to continuously update the database, which can impact performance and create additional overhead (Hector Garcia-Molina, 2014).

2.6 Related Work

Previous work in the field of biomedical informatics has demonstrated the value of Semantic Web technologies for representing and integrating heterogeneous biological datasets. Projects such as Bio2RDF (Belleau et al., 2008) and Linked Life Data (Momtchev et al., 2025) have made significant strides in publishing biological data as Linked Data using RDF. These platforms transform various biomedical datasets into a uniform RDF representation to facilitate federated queries and data integration. Similarly, the EBI RDF platform (Jupp et al., 2014) provides Linked Data views over EMBL-EBI datasets, enabling cross-dataset analysis using SPARQL endpoints.

In the domain of genetic and phenotypic data, the Monarch Initiative (Mungall et al., 2017) employs ontology-based data integration to unify diverse datasets such as HPO and GO, allowing researchers to perform semantic similarity and phenotype-driven variant prioritisation. Additionally, the Open PHACTS project (Williams et al., 2012) applied Semantic Web standards to pharmaceutical data to support drug discovery workflows.

Despite the success of these initiatives, none specifically address the transformation of legacy biomedical platforms like GeneNetwork2, which house decades of curated genetic and phenotypic data in complex SQL schemas. Although works such as Konopka (2015) and Soldatova et al. (2008) promote semantic enrichment of biomedical data, they do not provide frameworks or toolchains for automated SQL-to-RDF transformation at scale, nor do they evaluate the performance trade-offs between SPARQL and SQL in such a context.

Moreover, while Gunturkun et al. (2022) introduced GeneCup–a tool that mines PubMed and GWAS metadata for gene-keyword relationships–it does not persist

or structure these relationships using Semantic Web technologies, thereby limiting downstream interoperability and reasoning capabilities.

This dissertation addresses this gap by focusing on the reorganisation of the GeneNetwork2 SQL database into RDF using a self-documenting DSL implemented in GNU Guile. It uniquely contributes a reproducible, open-source toolchain that transforms the SQL schema into a graph-based model, stores it in a triplestore, and evaluates query performance across paradigms. This work not only enhances GN's FAIR compliance but also lays the groundwork for broader application of RDF in legacy biomedical infrastructures.

2.7 Summary

SQL databases are widely used for managing structured data due to their efficiency in querying and manipulation. However, they exhibit significant limitations in handling unstructured data, scaling to large datasets or high-concurrency environments, and supporting real-time analytics for such type of data. In contrast, Linked Data and Semantic Web technologies–particularly RDF–offer a flexible, machine-readable model for data representation. RDF expresses data as subject-predicate-object triples, forming a graph-based structure ideal for interlinking heterogeneous datasets. SPARQL enables expressive and federated queries over RDF data, making it a powerful tool for data integration across diverse domains.

Numerous projects in biomedical informatics, such as Bio2RDF, Linked Life Data, and the Monarch Initiative, have successfully adopted Semantic Web standards to improve interoperability and enable complex, cross-dataset analyses. However, these efforts primarily target newly curated datasets or specific domains and do not address the transformation of complex, legacy SQL-based platforms like GeneNetwork2. This gap–between the promise of Semantic Web technologies and their application to longstanding, production-scale biomedical systems–remains underexplored. Existing tools in the bio-informatics space lack support for automating SQL-to-RDF transformations at scale or evaluating the implications of transitioning from relational to graph-based querying. This dissertation addresses that gap by presenting a reproducible, open-source RDF transformation pipeline for GeneNetwork2,

laying the groundwork for broader adoption of Semantic Web principles in legacy biomedical infrastructures.

Chapter 3

Methodology

This chapter provides a comprehensive explanation of how each of the research objectives listed in Chapter 1 were addressed. This involves showing how the proposed solution was designed, how the research planning was carried out and how development and testing was done.

3.1 Research Approach for Objective 1

The first objective of this disseration was focused on examining previous research and studies conducted on the GeneNetwork platform, SQL and RDF in order to identify any gaps that hinder effective data integration and semantic interoperability. To meet this objective, a comprehensive literature review was conducted.

3.2 Research Approach for Objective 2

The second objective of this dissertation was to design, implement and test a framework for porting the GeneNetwork2 database to RDF. In order to accomplish this objective, a self-documenting Domain Specific Language (DSL) was developed to facilitate the translation of SQL into RDF. This DSL was used to convert SQL tables into RDF representations, subsequently enabling a comprehensive comparison of RDF performance against that of SQL.

The development methodology that was adopted in this dissertation was the *Rapid Application Development (RAD)* methodology since it prioritisnes development and prototype-building over planning. It emphasizes the end-user involvement,

prototyping, reuse and the use of automated tools (Vliet, 2007). This gives the researcher the ability to efficiently execute numerous iterations and updates to the software/model without the need to begin each iteration anew. This yields a significant advantage, as it facilitates the ability to modify the design, augment functionality and iteratively refine the software/model at a high frequency.

Vliet (2007) outlines the four fundamental steps of the Rapid Application Development (RAD) methodology as:

1. **Requirements Planning:** In contrast to traditional software development models, the RAD methodology begins by obtaining a general understanding of the requirements rather than seeking a comprehensive list of specifications from end users. The broad nature of the requirements allows for the segmentation of specific requirements at different points throughout the development cycle.

2. **Prototyping:** At this stage, actual development occurs. Prototypes are created featuring various functionalities rapidly, deviating from a strict set of requirements. Typically, these prototypes are hastily constructed to demonstrate key features. The final software artifact is only developed during the finalization stage, where the consensus on the desired outcome has been reached.

3. **Construction:** In this stage, the working model is transformed into a functional system. Usually, most bugs, issues and modifications are addressed.

4. **Cutover:** The final stage of RAD involves the final testing of the system before the system is ready to use by end users. The cutover phase encompasses intensive scale testing, technical documentation, issue tracking, final customizations and possible system simulations.

Figure 3.1 shows the aforementioned RAD life-cycle.

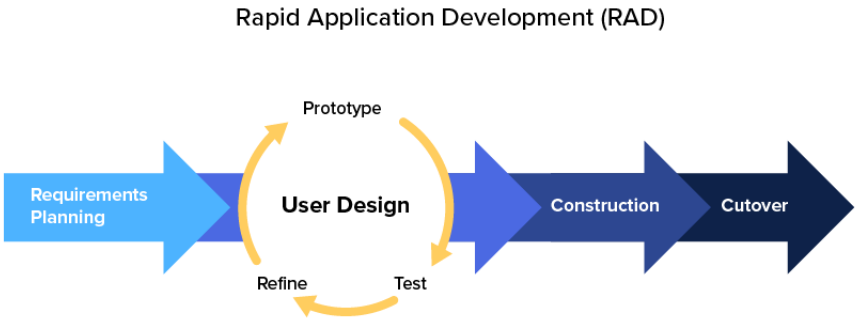


FIGURE 3.1: Rapid Application Development Life-Cycle. The development of a self-documenting Domain Specific Language (DSL) to translate SQL into RDF follows the Rapid Application Development (RAD) methodology. The process emphasizes iterative prototyping, user involvement, and rapid refinement across four stages

Source: KissFlow (2025)

The following subsections elaborate on the application of the RAD methodology within this dissertation's context.

3.2.1 Requirements Planning

The process of requirements planning is the initial stage in building the DSL. This stage encompasses activities such as conceptual ideation, during which potential representations of the DSL are brainstormed. The primary goal was to design a self-documenting DSL that is comprehensible and accessible to human end-users. Moreover, the DSL should support s-expressions and offer a declarative mapping of SQL query results to a predefined ontology. Furthermore, in this stage, relevant public and active ontologies were identified and when transforming data from SQL tables to RDF.

3.2.2 Prototyping

During the prototyping phase, a preliminary version of the DSL was developed, enabling the exploration of its design and functionality. This prototype functioned as a

proof-of-concept, and it was iteratively refined through feedback from the GeneNetwork leadership team [1], whose members possess extensive experience in programming, genetics, and genomics. They were selected for their ability to evaluate RDF progress and offer valuable insights. Figure 3.2 illustrates the parser's basic operation.

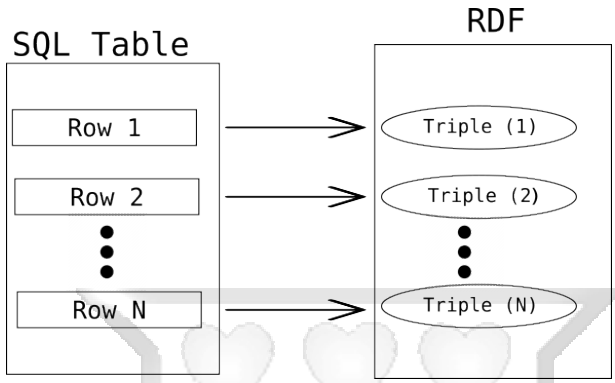


FIGURE 3.2: Parsing SQL to RDF. The parser operates on each row in a SQL table to produce a valid triple store.

During the prototyping stage, it was crucial to quickly check whether syntactical errors exist in the RDF output. To this end, *rapper* (Beckett, 2023), a Raptor RDF parser utility, was utilized to assess the syntactic correctness of the generated RDF format. This linting process ensured that the output was free of errors, permitting the data to be ingested into a virtuoso instance. After linting, data was uploaded to a running Virtuoso instance.

3.2.3 Construction

The construction phase encompassed the development of a robust, reusable tool. During this stage, numerous unforeseen challenges, including bugs, issues, and modifications, were identified and addressed. In order to prevent regressions, tests were be implemented concurrently with the development process, ensuring the tool's reliability and functionality.

[1] https://git.genenetwork.org/gn-docs/tree/general/team/leadership.md

3.2.4 Cut-off

During this final stage, the Domain Specific Language (DSL) was used to convert GeneNetwork tables into RDF format. Components of GeneNetwork that rely extensively on metadata were restructured to utilize RDF. A comparison between SQL and RDF performance was assessed.

3.3 Research Approach for Objective 3

The final objective of this dissertation was to validate this dissertation by comparing the performance of RDF against SQL. To validate the findings in this dissertation, identical queries were executed on each system, and a paired t-test was applied to the resulting runtimes. The paired t-test was used to determine whether the two paired measurements–SQL and SPARQL execution times–differed with statistical significance. We use the paired t-test because the collected data consists of paired observations and the differences between pairs are approximately normally distributed (Mishra et al., 2019). The assumption of normality in the differences is justified by the nature of performance timings. When running the same query multiple times, execution time is influenced by numerous small, independent system-level factors–such as CPU scheduling, I/O speed, cache state, and background processes. According to the Central Limit Theorem (CLT), the sum or average of these small, random effects tends toward a normal distribution, even if the individual components are not normally distributed. As a result, the distribution of performance differences typically forms a bell-shaped curve, satisfying the conditions required for the paired t-test.

Our approach tests both the null hypothesis, that there is no difference between SQL and SPARQL execution times, and the alternative hypothesis, indicating a difference in the execution time. Let $d_i = SQL_i - SPARQL_i$ represent the diffnnerence for each observation and μ_d denote the true mean difference. The null and alternative hypothosis are thus expressed as:

$$H_0 : \mu_d = 0$$

$$H_1 : \mu_d \neq 0$$

To calculate the paired t-test statistic, we first got the average performance difference of $d_i = SQL_{seconds} - SPARQL_{seconds}$ which is denoted as $\bar{d}$:

$$\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i, \text{ where n is the number of observations}$$

Next, we calculated the standard deviation, S_d of the differences:

$$S_d = \frac{1}{n-1}\sqrt{\sum_{i=1}^{n}(d_i - \bar{d})^2}$$

The degrees of freedom for the paired-test is $n - 1$, where n is the number of paired observations. The paired t-statistic was then calculated as:

$$t = \frac{\bar{d}}{S_d/\sqrt{n}}, \text{ with (n - 1) degrees of freedom}$$

Finally, We got the critical t-statistic ($t_{critical}$) using the degrees of freedom at a 99.9% confidence level ($t_{0.001}$) from the t-distribution tables. This critical value was compared with the calculated t-statistic, and the following decision rules were applied:

$$\text{If } |t| > t_{critical}, \text{ reject } H_0$$

$$\text{If } |t| \leq t_{critical}, \text{ fail to reject } H_0$$

3.4 Ethical considerations

Institutional approval was mandatory to ensure the legitimacy of the study and its findings. Strathmore University holds the required certification and granted ethical clearance for the project.

This study undertook the following ethical considerations:

- Acknowledging the source of the data that is used in the research.
- Obtaining necessary permissions and authorization to access and utilise the data.
- Clear documentation of the data collection, analysis and interpretation procedure in a clear and transparent manner that ensured reproducibility of the research findings.
- Responsibly communicate the research findings accurately, ethically, and appropriate context.

This research was subject to compliance with the provisions outlined in the Data Protection Act of Kenya, 2019, which governs data protection and privacy considerations.

3.5 Summary

This chapter detailed the methodology employed to address the research objectives outlined in the introduction. To meet the first objective, a comprehensive literature review was undertaken to understand prior work on GeneNetwork and RDF. For the second objective, a self-documenting Domain Specific Language (DSL) was developed using the Rapid Application Development (RAD) methodology. Each stage of RAD—requirements planning, prototyping, construction, and cutover—was tailored to support the DSL's iterative design and refinement. The DSL enabled structured translation of SQL tables into RDF format, allowing RDF-based systems to be benchmarked against traditional SQL systems.

To fulfil the final objective, the performance of SQL and RDF/SPARQL queries was evaluated using statistical methods, specifically a paired t-test, to determine the presence of significant differences in execution times.

Finally, ethical considerations, including institutional approval, responsible data use, and compliance with the Data Protection Act of Kenya (2019), were adhered to throughout the research process.

Chapter 4

System Design and Architecture

4.1 Introduction

This chapter presents a technical framework and architecture for implementing an embedded self-documenting DSL that is written in Scheme. The proposed DSL provides human-friendly syntax that allows a user to select tables from a SQL database, transform them into RDF, and save the output in a specified turtle file, all while generating a markdown file that has documentation about that given transformation. Given the DSL's foundation in Scheme, end users are also able to leverage Scheme's capabilities within the DSL's framework, thereby enhancing its extensibility.

4.2 Design Process

4.2.1 Design Goals

In contrast to general-purpose languages (GPLs), DSLs present certain benefits, as highlighted by Freudenthal (2010) and Spinellis (2001):

1. *Explicit Representation of Domain Knowledge:* DSLs allow domain-specific functionality to be expressed in a tangible, human-readable format at a high level of abstraction. This clarity makes software artifacts more accessible to developers, thereby simplifying the processes of development, testing and modification.

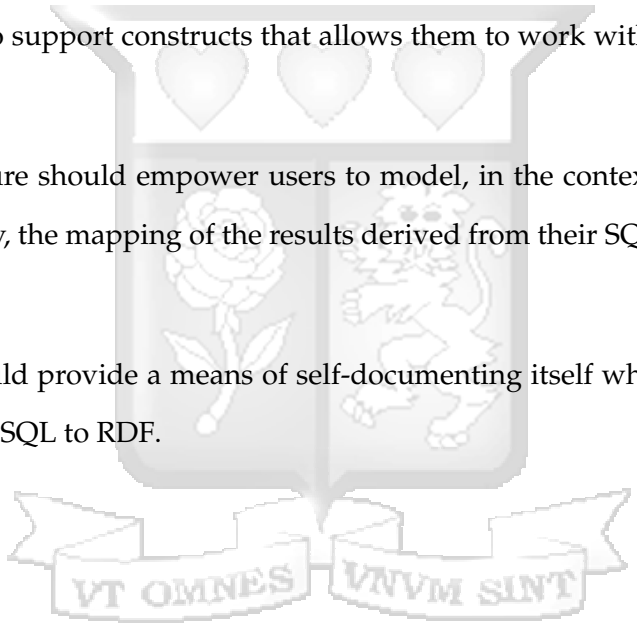
2. *Active Engagement of Domain Experts:* Programs crafted in DSL often adopt a style that aligns with the conventional formats used by domain experts. This facilitates their participation in the software's lifecycle and promotes collaboration with developers. In some cases, domain experts might directly specify, implement, verify or validate certain artifacts.

In lieu of the aforementioned advantages of a DSL, the system architecture should accomplish the following:

1. The architecture must be *database agnostic.* Since future user should be able to use this DSL with their unique databases, it is essential for the DSL to be flexible enough to support constructs that allows them to work with their unique database.

2. The architecture should empower users to model, in the context of some defined ontology, the mapping of the results derived from their SQL queries into RDF.

3. The DSL should provide a means of self-documenting itself while transforming data from SQL to RDF.



4.2.2 Architecture

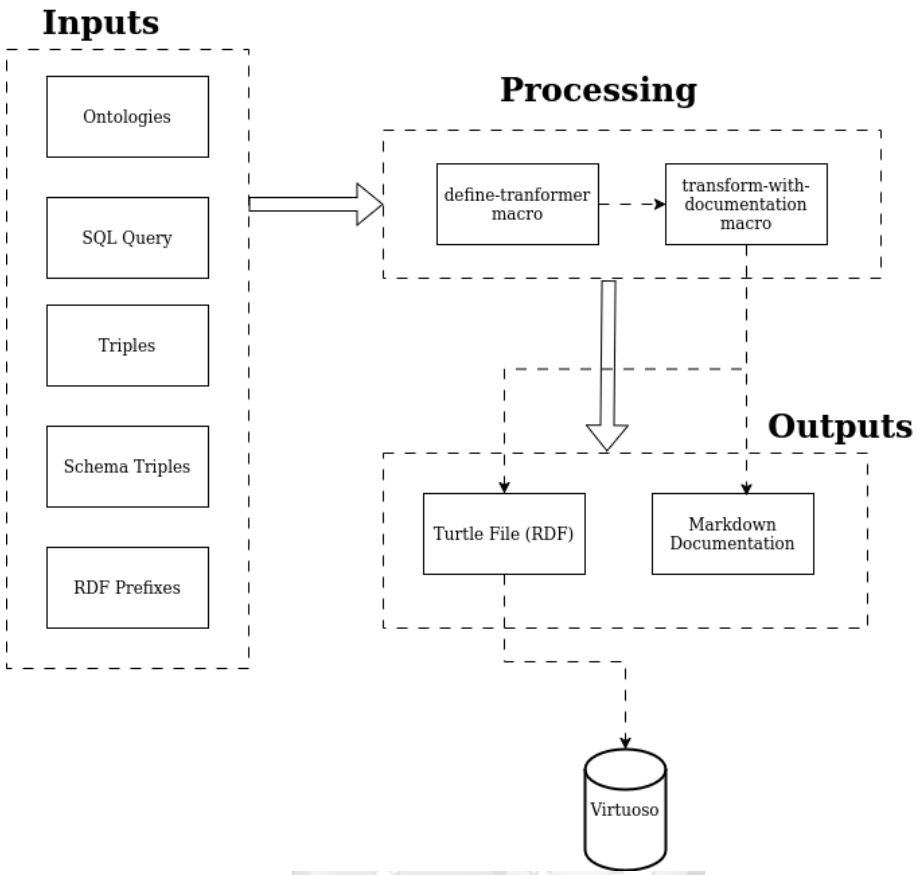


FIGURE 4.1: DSL Architecture. This shows the architecture of the DSL that converts SQL to RDF and all its major components.

Figure 4.1 shows the overarching structure of the DSL under consideration. In general, the transformation from SQL to RDF comprises the processing of distinct input components into a set of outputs: a turtle and markdown file. These inputs comprise:

1. *Ontologies*: An ontology is a formal way of representing knowledge. In the context of GeneNetwork, several ontologies are selected to server as conceptual frameworks for characterizing various entities, including experiments, investigators, species, genes and other relevant entities.

2. *SQL Query*: Acting as the most important instruction set, the SQL query is executed as a pivotal step in the transformation process.

3. *Triples Mapping*: This is a clear mapping between the results of the aforementioned SQL query and the corresponding RDF structure.

4. *User Defined Triples*: In scenarios where existing ontologies fail to encompass domain-specific concepts within the GeneNetwork context, user-defined triples are introduced to accommodate these unique conceptual dimensions.

5. *RDF Prefixes*: These are the RDF namespaces that will be appended to the beginning of the turtle file.

The "define-transformer" macro is the central driver for converting SQL to RDF. It provides an interface for specifying the SQL query, user-defined triples, and SQL-to-RDF mapping. The "define-transformer" macro is one of the inputs to the "transform-with-documentation" macro which takes objects using the "define-transformer" macro as input, along with RDF prefixes. It then parses the abstract syntax tree of the "define-transformer" macro to automatically generate documentation, and executes the transformation itself, resulting in a Turtle file.

4.3 Summary

This chapter has outlined the design and architectural framework of a Scheme-based, embedded DSL that facilitates the transformation of SQL query results into RDF while automatically generating self-documenting output. The system was built with the key design goals of being database-agnostic, ontology-aware, and user-extensible. By leveraging the expressive capabilities of Scheme and the modularity of macros such as `define-transformer` and `transform-with-documentation`, the DSL offers a streamlined approach to data translation and documentation.

The chapter began by exploring the motivation and advantages of domain-specific languages in the context of RDF transformation. It then introduced the main architectural components required to perform the translation–ontologies, SQL queries, mapping specifications, user-defined triples, and RDF prefixes. The transformation process culminates in the generation of both a Turtle file containing RDF triples and a Markdown file containing documentation for traceability and transparency.

Chapter 5

System Implementation and Testing

This chapter describes the implementation and evaluation of the self-documenting DSL designed for transforming SQL tables into RDF.

5.1 Hardware and Software Requirements

5.1.1 Hardware Requirements

The development of this DSL was conducted on a server provided by the University of Tennessee Health Science Centre. This server had the following specifications:

CPU	Dual 32-core AMD EPYC 7551 processors
Kernel	Linux Kernel version 4.9.0-14-amd64 x86_64
Memory	251.5939 GiB of RAM
Storage	24.22 TiB
Operating System	Debian GNU/Linux 9.13 (stretch)

5.1.2 Software Requirements

The software dependencies for this DSL was managed using GNU Guix[1]. The implementation was done in a scheme dialect called GNU Guile[2]. The database that was used was MariaDB and the RDF datastore utilised was Virtuoso[3].

[1] https://guix.gnu.org/
[2] https://www.gnu.org/software/guile/
[3] https://virtuoso.openlinksw.com/

5.2 DSL Implementation

This DSL uses Scheme's flexible hygienic macro system to offer end-users a human-friendly syntax for interacting with SQL databases and generating RDF output bundled with transparent documentation. The implementation of this DSL involves several key components, as illustrated in Figure 4.1 that include: defining ontologies, defining transformers, processing SQL queries, mapping SQL results to RDF, and generating documentation. The subsections below outline the implementation of these various parts.

5.2.1 Defining Transformers

A special syntax, ''define-transformer'' is introduced. This special syntax transforms a view of a database. ''define-transformer'' consists of three order-agnostic clauses: ''tables'', ''schema-triples'' and ''triples'', in the form shown in Figure 5.1 below:

```
(define-transformer function-name
  (tables (table ...) raw-forms ...)
  (schema-triples
   (subject predicate object) ...)
  (triples subject
    (verb predicate object) ...))
```

FIGURE 5.1: "define-transformer" syntax. This shows the general form of the "define-transformer" syntax. It comprises the "tables", "schema-triples" and "triples" form.

The ''tables'' clause specifies the database tables to be joined to construct the view to be transformed. *"table"* can be either of the form *"table-name"* or of the form: ''(JOIN-OPERATOR TABLE-NAME RAW-CONDITION)''. *"table-name"* is the name of the table. ''JOIN-OPERATOR'' can be one of: ''join'', ''left-join'' and ''inner-join''. *"RAW-CONDITION"* is the join condition as a raw string. This is usually something like ''USING (SpeciesId)''. *"raw-forms"* are expressions that must evaluate to strings to be appended to the SQL query. The example below shows an example of doing a left join between a Genbank table with a Species table using the "SpeciesId" as the foreign key as shown in Figure 5.2:

```
(tables (Genbank
          (left-join Species "USING (SpeciesId)")))
```

FIGURE 5.2: Using the "table" special form to select data from tables. This example use the "table" special form to do a left join between the *Genbank* and *Species* table using the *SpeciesId* column as the foreign key.

The ''schema-triples'' clause specifies the list of triples to be written once when the transform starts. An example of how this would like is demonstrated in Figure 5.3:

```
(schema-triples
   (gnc:nucleotide a skos:Concept)
   (gnt:hasSequence rdfs:domain gnc:nucleotide))
```

FIGURE 5.3: Defining special triples that will be run only once before the SQL query is executed. This example defines 2 triples. These 2 triples are are generated only once for the entire SQL query.

On the other hand, the ''triples'' clause specifies the triples to be transformed once for each row in the view. All triples have a common *"subject"*. The ''(verb predicate object)'' clauses are described below:

"verb" can either be a ''set'' or ''multiset''. For the ''set'' *"verb"*, a single triple ''(SUBJECT PREDICATE OBJECT-VALUE)'' is written where *"OBJECT-VALUE"* is the result of evaluating *"OBJECT"*. For the ''multiset'' *"verb"*, *"OBJECT"* evaluates to a list, and a triple ''(SUBJECT PREDICATE OBJECT-VALUE-ELEMENT)'' is created for each element *"OBJECT-VALUE-ELEMENT"* of that list.

The *"subject"* and *"object"* expressions in the triples clause reference database fields using a ''(field TABLE COLUMN)'' clause where *"TABLE"* and *"COLUMN"* refer to the table and column of the field being referenced. Database fields can also be referenced using ''(field TABLE COLUMN ALIAS)'' where *"ALIAS"* is an alias for that column in the SQL query.

The example in Figure 5.4 below shows how a triple can be defined:

```
(triples (field Genbank Id)
    (set gnt:hasSequence (field Genbank Sequence))
    (set gnt:belongsToSpecies (field Species Fullname)))
```

FIGURE 5.4: Defining triples from table rows. This example shows how triples are expressed declaratively from the relevant table fields. Each row returned from the SQL query will be parsed into the triples matching the form expressed here.

5.2.2 Executing a Transformer

To execute transformers, an extra special syntax, ‘‘with-documentation’’ is provided. This special syntax is shown in Figure 5.5 below:

```
(with-documentation
   (name ...)
   (connection ...)
   (table-metadata? ...)
   (prefixes ...)
   (inputs ...)
   (outputs
    `(#:documentation ...
      #:rdf ...)))
```

FIGURE 5.5: The general form of the “with-documentation” syntax. This special form glues together: defined transformers, RDF prefixes, important connection parameters to the Virtuoso store and specifies where the output is generated to.

Each part of the ‘‘with-documentation’’ is described below:

1. ‘‘name’’: This represents the title used in the generated documentation markdown file.

2. ‘‘connection’’: This contains the SQL configuration variables that will be used to connect to the database.

3. ‘‘prefixes’’: These are various ontologies that will be included at the beginning of the RDF document.

4. ‘‘inputs’’: This is a list of defined transformers that will be executed.

5. ‘‘outputs’’: This specifies the file path where the RDF and documentation files will be saved.

A more complete example is shown in Figure 5.6 below:

```
(with-documentation
   (name "Genebank Metadata")
   (connection %connection-settings)
   (table-metadata? #f)
   (prefixes
    '(("rdf:" "<http://www.w3.org/1999/02/22-rdf-syntax-ns#>")
      ("rdfs:" "<http://www.w3.org/2000/01/rdf-schema#>")))
   (inputs
    (list
     genbank))
   (outputs
    `(#:documentation "/tmp/genbank.md"
      #:rdf "/tmp/genbank.ttl")))
```

FIGURE 5.6: Transforming Genbank SQL data into RDF. This example uses the “with-documentation” special form to convert data from the Genbank table to RDF. We use the prefixes “rdf:” and “rdfs:”. As input, we use the genbank transformer. The auto-generated document is stored in “/tmp/genbank.md” and the generated turtle file is stored in “/tmp/genbank.ttl”. In this example we don’t include table metadata provided from the SQL engine in “/tmp/genbank.ttl” by setting the “table-metadata” option to “false”. Connection settings to the MariaDB is provided by “%connection-settings”.

5.3 System Testing and Validation

In this section, various tests were done to assess the performance and functionality of the self-documenting DSL designed for transformed tables into RDF format. In addition to assessing the performance of the SPARQL queries from the generated RDF, the quality of the output is also assessed, and federated queries that are impossible to do in SQL are highlighted.

5.3.1 Query Quality and Execution Performance

The following SQL query, shown in Figure 5.7 below, was run several times to count the number of publications in the GeneNetwork SQL store related to diabetes:

```
SELECT COUNT(DISTINCT Id) FROM Publication
WHERE Abstract LIKE "%diabetes%";
```

FIGURE 5.7: Finding the number of Publications whose abstract have the word "diabetes" from the GN database. This query shows the number of distinct publications tha contain the word "diabetes" in their abstract from the GN SQL database.

Similarly, Figure 5.8 below shows the corresponding SPARQL query use to count the number of publications in RDF.

```
PREFIX   dct: <http://purl.org/dc/terms/>
PREFIX   rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX fabio: <http://purl.org/spar/fabio/>

SELECT COUNT(DISTINCT ?paper) WHERE {
       ?paper rdf:type fabio:ResearchPaper ;
              dct:abstract ?abstract .
       ?abstract bif:contains "diabetes" .
}
```

FIGURE 5.8: Finding the number of publications whose abstract have the word "diabetes" from Virtuoso in SPARQL. This query shows the number of distinct publications tha contain the word "diabetes" in their abstract using SPARQL.

In the 2 queries above, we get 22 publications that have the word "diabetes" in their abstract. The performance metrics of these 2 queries is shown in Table 5.1 below:

TABLE 5.1: Comparing how fast it takes to get the number of publications with the word "diabetes" in the abstract.

SQL (s)	SPARQL (s)	$d_i = (SQL_i - SPARQL_i)$	$(d_i - \bar{d})^2$
0.09	0.014	0.076	0.00000784
0.08	0.017	0.063	0.00024964
0.10	0.016	0.084	0.00002704
0.11	0.014	0.096	0.00029584
0.09	0.015	0.075	0.00001444
		$\bar{d}_i = \sum d_i / n = 0.0788$	$S_d = 0.01124982$
			$t_{calculated,\, df=4} = \frac{0.0788}{0.01124982/\sqrt{5}}$
			≈ 14.44
			$t_{critical,\, \alpha=0.001,\, df=4} = 8.610$

The following query, shown in Figure 5.9, is more complicated and counts the number of phenotypes that are related to diabetes:

```
SELECT
  COUNT(DISTINCT Phenotype.Id)
FROM
  PublishXRef
  LEFT JOIN Publication ON
    Publication.Id = PublishXRef.PublicationId
  LEFT JOIN Phenotype ON
    Phenotype.Id = PublishXRef.PhenotypeId
WHERE
  Publication.Abstract LIKE "%diabetes%";
```

FIGURE 5.9: Finding the number of phenotypes related to diabetes from the GN database. The SQL query is more complex because it requires multiple table joins (I.e., PublishXRef, Publication, and Phenotype) and prior knowledge of the schema to understand the relationships. Without context, it's difficult to know what each table represents.

The corresponding SPARQL query, shown in Figure 5.10, is:

```
PREFIX   dct: <http://purl.org/dc/terms/>
PREFIX   rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX   gnc: <http://genenetwork.org/category/>
PREFIX fabio: <http://purl.org/spar/fabio/>

SELECT COUNT(DISTINCT ?phenotype) WHERE {
       ?paper rdf:type fabio:ResearchPaper ;
              dct:abstract ?abstract .
       ?abstract bif:contains "diabetes" .
       ?phenotype rdf:type gnc:Phenotype ;
                  dct:isReferencedBy ?paper .
}
```

FIGURE 5.10: Finding the number of phenotypes related to diabetes from the GN database. This SPARQL query is simpler and less complex compared to it's alternative SQL query—Figure 5.10—because it directly expresses relationships using intuitive RDF triples and ontology terms like *fabio:ResearchPaper* and *gnc:Phenotype*. This makes it easier to understand without needing prior knowledge of the database schema or explicit table joins, unlike the SQL query, which requires joining multiple tables and navigating their relationships manually.

In both queries above (Figure 5.9 and Figure 5.10), we get 105 phenotypes that are related to diabetes. The performance of these 2 queries is tabulated in Table 5.2 below:

TABLE 5.2: Comparing execution times for counting phenotype datasets containing "diabetes".

SQL (s)	SPARQL (s)	$d_i = (SQL_i - SPARQL_i)$	$(d_i - \bar{d})^2$
0.19	0.013	0.177	0.000154
0.18	0.015	0.165	0.00000016
0.17	0.014	0.156	0.000074
0.18	0.016	0.164	0.00000036
0.17	0.009	0.161	0.000013
		$\bar{d}_i = \sum d_i / n = 0.1646$	$S_d = 0.00777$
			$t_{calculated, df=4} = \frac{0.1646}{0.00777/\sqrt{5}}$
			≈ 47.37
			$t_{critical, \alpha=0.001, df=4} = 8.610$

A more complicated query that counts the number of datasets across different InbredSet group that have phenotypes related to diabetes is shown in Figure 5.11.

```
SELECT
  COUNT(DISTINCT InfoFiles.InfoFileId)
FROM
  InfoFiles
  LEFT JOIN PublishFreeze
    ON InfoFiles.InfoPageName = PublishFreeze.Name
  LEFT JOIN Datasets USING (DatasetId)
  LEFT JOIN InbredSet
    ON InfoFiles.InbredSetId = InbredSet.InbredSetId
  LEFT JOIN PublishXRef
    ON PublishXRef.InbredSetId = InfoFiles.InbredSetId
  LEFT JOIN Publication
    ON Publication.Id = PublishXRef.PublicationId
  LEFT JOIN Phenotype ON Phenotype.Id = PublishXRef.PhenotypeId
WHERE
  Publication.Abstract LIKE "%diabetes%";
```

FIGURE 5.11: SQL Query to count the number of datasets across different inbredset groups that have phenotypes related to diabetes.

The corresponding SPARQL query is shown in Figure 5.12.

```
PREFIX   dct: <http://purl.org/dc/terms/>
PREFIX   rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX fabio: <http://purl.org/spar/fabio/>
PREFIX   gnc: <http://genenetwork.org/category/>
PREFIX   gnt: <http://genenetwork.org/term/>
PREFIX  dcat: <http://www.w3.org/ns/dcat#>
PREFIX  xkos: <http://rdf-vocabulary.ddialliance.org/xkos#>

SELECT COUNT(DISTINCT ?dataset) WHERE {
       ?paper rdf:type fabio:ResearchPaper ;
              dct:abstract ?abstract .
       ?abstract bif:contains "diabetes" .
       ?phenotype rdf:type gnc:Phenotype ;
                  gnt:belongsToGroup ?group ;
                  dct:isReferencedBy ?paper .
       ?dataset rdf:type dcat:Dataset ;
                gnt:belongsToGroup ?group .

}
```

FIGURE 5.12: SPARQL Query to count the number of datasets across different inbredset groups that have phenotypes related to diabetes.

From the above 2 queries - Figure 5.11 and Figure 5.12 - we get 2 different dataset

counts. The SQL query returns 406 datasets, while the SPARQL query returns 419 datasets. The performance metrics of thes 2 queries are tabulated in Table 5.3 below.

TABLE 5.3: Execution times for counting diabetes-related datasets across different inbredset groups.

SQL (s)	SPARQL (s)	$d_i = (SQL_i - SPARQL_i)$	$(d_i - \bar{d})^2$
38.33	0.649	37.681	0.066241
38.46	0.124	38.336	0.734449
41.53	0.430	41.100	11.971600
38.53	0.118	38.412	0.829921
38.50	0.696	37.804	0.005929
		$\bar{d}_i = \sum d_i / n = 38.6666$	$S_d = 1.8586$
			$t_{calculated, df=4} = \frac{38.6666}{1.8586/\sqrt{5}}$
			≈ 46.51
			$t_{critical, \alpha=0.001, df=4} = 8.610$

5.3.2 Query Output Quality

Consider the SQL query, shown in Figure 5.13, which fetches all the details about the mouse species from the SQL store.

```
SELECT * FROM Species WHERE Name="Mouse" \G
```

FIGURE 5.13: SQL query to retrieve mouse details from Species table.

The corresponding results are illustrated in Figure 5.14.

```
          Id: 1
   SpeciesId: 1
 SpeciesName: Mouse
        Name: mouse
    MenuName: Mouse (Mus musculus, mm10)
    FullName: Mus musculus
      Family: Vertebrates
FamilyOrderId: 1
  TaxonomyId: 10090
     OrderId: 15
```

FIGURE 5.14: Mouse details from Species table. This example illustrates the SQL tabular result structure, which lacks semantic context for the columns unless defined elsewhere.

The corresponding sparql query is shown in Figure 5.15:

```
PREFIX gnt: <http://genenetwork.org/term/>

CONSTRUCT {
  ?species ?predicate ?object
} WHERE {
  ?species gnt:shortName "mouse";
           ?predicate ?object .
}
```

FIGURE 5.15: SPARQL SELECT query to retrieve mouse details from Species table.

The enriched results of Figure 5.15 are illustrated in Figure 5.16

```
{
  "@context": {
    "gnc": "http://genenetwork.org/category/",
    "gnt": "http://genenetwork.org/term/",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#>",
    "label": "rdfs:label",
    "prefLabel": "http://www.w3.org/2004/02/skos/core#prefLabel",
    "taxonomy-id":
     ↪  "http://www.w3.org/2004/02/skos/core#notation",
    "family": "gnt:family",
    "shortname": "gnt:shortName",
    "definedBy":
     ↪  "http://www.w3.org/1999/02/22-rdf-syntax-ns#isDefinedBy",
    "in-scheme": "http://www.w3.org/2004/02/skos/core#inScheme",
    "alt-label": "http://www.w3.org/2004/02/skos/core#altLabel"
  },
  "@id": "http://genenetwork.org/id/Mus_musculus",
  "family": "Vertebrates",
  "shortname": "mouse",
  "definedBy": {
    "@id": "http://www.wikidata.org/entity/Q83310"
  },
  "label": "Mus musculus",
  "alt-label": "Mouse",
  "in-scheme": {
    "@id": "gnc:ResourceClassificationScheme"
  },
  "taxonomy-id": {
    "@id": "http://purl.uniprot.org/taxonomy/10090"
  },
  "prefLabel": "Mouse (Mus musculus, mm10)"
}
```

FIGURE 5.16: Mouse details in json-ld format from GeneNetwork Graph in Virtuoso. Here, we can see that json-ld provides richer semantic context by clearly defining relationships and attributes through standardized vocabularies and links to relevant resources.

A common operation done in GeneNetwork is to search GeneRIF entries from NCBI to search given probesets for key terms belonging to a given species. Figure 5.17 shows how such a query in SQL looks like:

```
SELECT
  DISTINCT Species.*, GeneRIF_BASIC.*
FROM
  Species
  INNER JOIN InbredSet ON InbredSet.SpeciesId = Species.Id
  INNER JOIN ProbeFreeze ON ProbeFreeze.InbredSetId =
  ↪  InbredSet.Id
  INNER JOIN Tissue ON ProbeFreeze.TissueId = Tissue.Id
  INNER JOIN ProbeSetFreeze ON ProbeSetFreeze.ProbeFreezeId =
  ↪  ProbeFreeze.Id
  INNER JOIN ProbeSetXRef ON ProbeSetXRef.ProbeSetFreezeId =
  ↪  ProbeSetFreeze.Id
  INNER JOIN ProbeSet ON ProbeSet.Id = ProbeSetXRef.ProbeSetId
  LEFT JOIN Geno ON ProbeSetXRef.Locus = Geno.Name
  AND Geno.SpeciesId = Species.Id
  INNER JOIN GeneRIF_BASIC ON GeneRIF_BASIC.symbol =
  ↪  ProbeSet.Symbol
WHERE
  GeneRIF_BASIC.comment LIKE "%diabetes%"
  AND Species.Name = "mouse"
  AND GeneRIF_BASIC.`symbol` = BINARY "Gcg"
ORDER BY
  GeneRIF_BASIC.comment ASC
LIMIT
  1\G
```

FIGURE 5.17: SQL Search for GeneRIF entries from NCBI for the keywords "diabetes" that belong to the mouse species

The results of the code listing above in Figure 5.17 are shown in Figure 5.18.

```
           Id: 1
    SpeciesId: 1
  SpeciesName: Mouse
         Name: mouse
     MenuName: Mouse (Mus musculus, mm10)
     FullName: Mus musculus
       Family: VertebratesFamilyOrderId: 1
   TaxonomyId: 10090
      OrderId: 15
    SpeciesId: 2
        TaxID: 10116
       GeneId: 24952
       symbol: Gcg
    PubMed_ID: 26037249
   createtime: 2015-10-24 11:29:00
      comment: Data (including data from transgenic rats) suggest 1)
               celiac ganglia neurotransmission and 2) glucagon
               secretion are down-regulated in short-term
               hyperglycemia of diabetes; Wallerian degeneration
               protein mutation protects against these dysfunctions.
    VersionId: 1
1 row in set (0.01 sec)
```

FIGURE 5.18: Results of SQL search for GeneRIF from NCBI for the keywords "diabetes" that belong to the mouse species

The corresponding SPARQL query for the code listed in Figure 5.17 is shown below in Figure: 5.19:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX gnt: <http://genenetwork.org/term/>
PREFIX gnc: <http://genenetwork.org/category/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX gn: <http://genenetwork.org/id/>

CONSTRUCT {
  ?species ?speciesPred ?speciesObj .
  gnt:comment ?commentPred ?commentObj .
} WHERE {
?probeset rdf:type gnc:Probeset ;
          gnt:geneSymbol "Gcg" ;
          ?p ?o .
?symbol rdfs:comment _:node ;
        rdfs:label "Gcg" .
_:node rdf:type gnc:NCBIWikiEntry ;
       gnt:belongsToSpecies ?species ;
       rdfs:comment ?comment ;
       ?commentPred ?commentObj .
?species gnt:shortName "mouse";
       ?speciesPred ?speciesObj .
?comment bif:contains "diabetes" .
}
```

FIGURE 5.19: SPARQL Search for GeneRIF entries from NCBI for the keywords "diabetes" that belong to the mouse species

The corresponding results for Figure 5.19 are shown below:

```
{
  "@context": {
    "hasGeneId": {
      "@id": "http://genenetwork.org/term/hasGeneId",
      "@type": "@id"
    },
    "created": {
      "@id": "http://purl.org/dc/terms/created",
      "@type": "http://www.w3.org/2001/XMLSchema#datetime"
    },
    "altLabel": {
      "@id": "http://www.w3.org/2004/02/skos/core#altLabel"
    },
    "inScheme": {
      "@id": "http://www.w3.org/2004/02/skos/core#inScheme",
      "@type": "@id"
    },
    "family": {
      "@id": "http://genenetwork.org/term/family"
    },
    "shortName": {
      "@id": "http://genenetwork.org/term/shortName"
    },
```

```
    "isDefinedBy": {
      "@id": "http://www.w3.org/1999/02/22-rdf-syntax-ns#isDefin
      ↪ edBy",
      "@type": "@id"
    },
    "belongsToSpecies": {
      "@id": "http://genenetwork.org/term/belongsToSpecies",
      "@type": "@id"
    },
    "prefLabel": {
      "@id": "http://www.w3.org/2004/02/skos/core#prefLabel"
    },
    "label": {
      "@id": "http://www.w3.org/2000/01/rdf-schema#label"
    },
    "comment": {
      "@id": "http://www.w3.org/2000/01/rdf-schema#comment"
    },
    "notation": {
      "@id": "http://www.w3.org/2004/02/skos/core#notation",
      "@type": "@id"
    },
    "hasVersionId": {
      "@id": "http://genenetwork.org/term/hasVersionId"
    }
  },
  "@graph": [
    {
      "@id": "http://genenetwork.org/id/Mus_musculus",
      "family": "Vertebrates",
      "shortName": "mouse",
      "isDefinedBy": "http://www.wikidata.org/entity/Q83310",
      "label": "Mus musculus",
      "altLabel": "Mouse",
      "inScheme": "http://genenetwork.org/category/ResourceClass
      ↪ ificationScheme",
      "notation": "http://purl.uniprot.org/taxonomy/10090",
      "prefLabel": "Mouse (Mus musculus, mm10)"
    },
    {
      "@id": "http://genenetwork.org/term/comment",
      "@type": "http://genenetwork.org/category/NCBIWikiEntry",
      "belongsToSpecies":
      ↪ "http://genenetwork.org/id/Mus_musculus",
      "hasGeneId": "http://www.ncbi.nlm.nih.gov/gene?cmd=Retriev
      ↪ e&dopt=Graphics&list_uids=14526",
      "hasVersionId": 1,
      "created": "2010-10-23T06:10:00",
```

```
        "comment": {
          "@type": "http://www.w3.org/2001/XMLSchema#string",
          "@value": "Data show that in an animal model of diabetes
           ↪  induced by alloxan, progression of hyperglycemia was
           ↪  significantly attenuated in mice given the Ad-GLP-1
           ↪  vector compared with control mice."
        },
        "notation": "https://www.ncbi.nlm.nih.gov/Taxonomy/Browser
         ↪  /wwwtax.cgi?mode=Info&id=10090"
      }
    ]
}
```

FIGURE 5.20: SPARQL Search for GeneRIF entries from NCBI for the keywords "diabetes" that belong to the mouse species

From the above results, we see that the SPARQL query results offer richer contextual information. Each subject and predicate field, analogous to SQL columns, is linked to a resource providing additional definitions. Furthermore, the versatility of SPARQL extends to the ability to output results to HTML, CSV, TSV, JSON, JSON-LD and other formats [4].

5.3.3 Federated Queries in SPARQL

SQL is limited to querying tables that reside within the local server environment and does not have the capability to access or query external servers on the internet. However, RDF overcomes this limitation by allowing the use of federated queries that allow a person to query external services. The following example, shown in Figure 5.21, highlights SQL's limitation. It queries GeneNetwork to get the Wikidata identifier that can be used to query Wikidata as an external service, to get the longest recorded life span of a mouse which is 4 years.

[4] http://docs.openlinksw.com/virtuoso/rdfsparql.html#rdfsparqlimplementatioptragmassfs

```
PREFIX       gnt: <http://genenetwork.org/term/>
PREFIX       rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX       wdt: <http://www.wikidata.org/prop/direct/>

SELECT ?years {
  ?species gnt:shortName "mouse";
           rdf:isDefinedBy ?id .
SERVICE <https://query.wikidata.org/sparql> {
    ?id wdt:P4214 ?years .
  }
}
```

FIGURE 5.21: Using a Federated Query. In this example, a federated query is used to annotate GeneNetwork data using metadata from an external service, Wikidata, to fetch the longest recorded life span of a mouse.

Chapter 6

Discussion of Results

6.1 Introduction

This dissertation aimed to investigate prior research and studies concerning the GeneNetwork platform and RDF to identify the existing limitations of GeneNetwork related to SQL and explore how RDF could potentially mitigate these limitations. Additionally, a domain-specific language (DSL) was developed and implemented to transform GeneNetwork metadata from SQL to RDF, followed by performance bench marking and output quality assessments. This chapter discusses the performance benchmarks and output quality assessment that were conducted in addition to highlighting additional benefits that can be gained by using RDF to store metadata.

6.2 Performance Analysis

The conducted performance tests compared SQL and SPARQL queries for retrieving information related to diabetes from a GeneNetwork SQL store using the paired t-test statistic at a 99.9% confidence level ($\alpha = 0.001$) to evaluate the statistical significance of the observed difference. The null hypothesis posited no difference in execution times between the two query languages, while the alternative hypothesis asserted a statistically significant performance disparity.

The initial SQL query aimed to count the number of publications with abstracts containing the term "diabetes". Correspondingly, the SPARQL query transformed

this SQL query into RDF format to achieve the same result using RDF data model constructs. More complicated queries were constructed to benchmark the performance of RDF against SQL.

The results demonstrated overwhelming evidence against H_0:

(i) Table 5.1: $t_{calculated,\, df=4} \approx 14.44 > t_{critical,\, \alpha=0.001,\, df=4} = 8.610$

(ii) Table 5.2: $t_{calculated,\, df=4} \approx 47.37 > t_{critical,\, \alpha=0.001,\, df=4} = 8.610$

(iii) Table 5.3: $t_{calculated,\, df=4} \approx 46.51 > t_{critical,\, \alpha=0.001,\, df=4} = 8.610$

In all cases, the calculated t-values far exceeded the critical threshold, leading to the rejection of the null hypothesis, thereby confirming that SPARQL exhibits statistically significant faster execution times than SQL for diabetes-related queries.

As the complexity of the queries increased, SQL experienced a more pronounced slowdown compared to SPARQL. Of significance is the SQL query listed in Figure 5.11. To understand why this query is slow, we run an "EXPLAIN" on this query as shown in Figure 6.1 below.

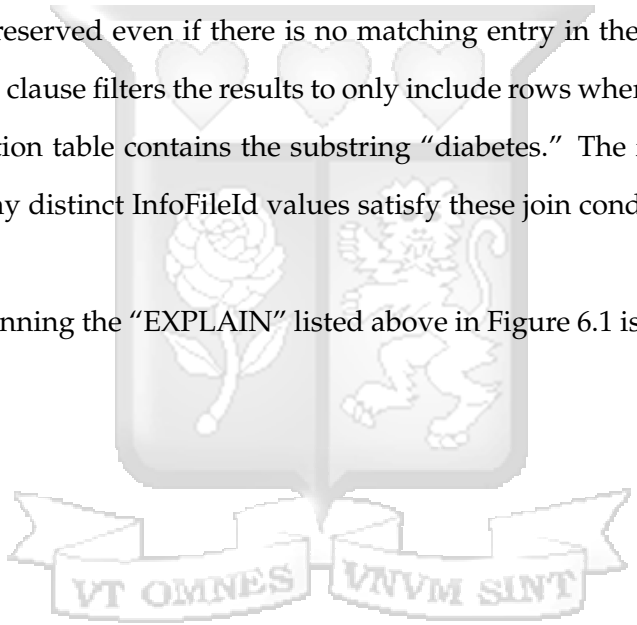
```
EXPLAIN SELECT
  COUNT(DISTINCT InfoFiles.InfoFileId)
FROM
  InfoFiles
  LEFT JOIN PublishFreeze
    ON InfoFiles.InfoPageName = PublishFreeze.Name
  LEFT JOIN Datasets USING (DatasetId)
  LEFT JOIN InbredSet
    ON InfoFiles.InbredSetId = InbredSet.InbredSetId
  LEFT JOIN PublishXRef
    ON PublishXRef.InbredSetId = InfoFiles.InbredSetId
  LEFT JOIN Publication
    ON Publication.Id = PublishXRef.PublicationId
  LEFT JOIN Phenotype ON Phenotype.Id = PublishXRef.PhenotypeId
WHERE
  Publication.Abstract LIKE "%diabetes%";
```

FIGURE 6.1: Analyzing a complex query. Here we use SQL's EXPLAIN to analyze the query.

The SQL query in Figure 6.1 performs a count of distinct "InfoFileId" values from the "InfoFiles" table, filtered based on whether the associated "Publication.Abstract" contains the word "diabetes." It begins by selecting the "InfoFiles" table—which

contains metadata for GN datasets—as the base and successively performs a series of LEFT JOINs to bring in related data from other tables. First, it joins PublishFreeze– a table that contains metadata on all datasets that have been published—on a match between "InfoFiles.InfoPageName" and "PublishFreeze.Name", followed by a join with "Datasets" using the common DatasetId. Next, it joins with "InbredSet" using the "InbredSetId" field. From there, it joins with "PublishXRef"—a cross-reference table that is usually used to link with actual sample values, again on "InbredSetId", and further connects to "Publication"—a table with information about publications—using "Publication.Id" and to "Phenotype"—a table with all the phenotypes in GN—via "Phenotype.Id". The LEFT JOIN operations ensure that records from the "InfoFiles" table are preserved even if there is no matching entry in the joined tables. Finally, the WHERE clause filters the results to only include rows where the Abstract field in the Publication table contains the substring "diabetes." The final output is a count of how many distinct InfoFileId values satisfy these join conditions and the specified filter.

The results of running the "EXPLAIN" listed above in Figure 6.1 is shown below in Figure 6.2.



```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: InfoFiles
         type: ALL
possible_keys: NULL
          key: NULL
          key_len: NULL
          ref: NULL
          rows: 720
      Extra: Using where
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: PublishFreeze
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
          rows: 70
          Extra: Using where; Using join buffer (flat, BNL join)
*************************** 3. row ***************************
           id: 1
  select_type: SIMPLE
        table: InbredSet
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 93
        Extra: Using where; Using join buffer (incremental, BNL join)
*************************** 4. row ***************************
           id: 1
  select_type: SIMPLE
        table: PublishXRef
         type: ref
possible_keys: InbredSet,record,InbredSetId
          key: record
      key_len: 2
          ref: db_webqtl.InfoFiles.InbredSetId
         rows: 181
        Extra: Using where; Using index
*************************** 5. row ***************************
           id: 1
  select_type: SIMPLE
        table: Publication
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: db_webqtl.PublishXRef.PublicationId
         rows: 1
        Extra: Using where
```

FIGURE 6.2: Results of running "EXPLAIN" on a complex query. This is the result of running "EXPLAIN" from Figure 6.1. We use "EXPLAIN" to check for performance bottlenecks on the query.

The SQL query in Figure 5.11 was slow because of inefficient table scans. The *"InfoFiles"*, *"PublishFreeze"*, and *"InbredSet"* tables are being scanned entirely (*"type:ALL"*), meaning that every row in these tables is checked without utilising indexes. Also, we have join buffers (*"BNL join"*) indicating that large portions of data must be temporarily stored and iterated through, further slowing down execution. The corresponding SPARQL query listed in Figure 5.19 operates on RDF triples, which are more efficient for graph-based queries. It leverages structured, linked data, making it easier to filter and join relationships using ontology terms(*"rdfs:label"*, *"rdf:type"*, etc.). This SPARQL query can directly locate relationships like *"species"* and *"comments"*, avoiding expensive table scans. Also, the use of *"bif:contains"* for full-text search is optimized for RDF data stores, speeding up queries involving text searches like *"diabetes."*

6.3 Output Quality and FAIR Data

FAIR data means that data is Findable, Accessible, Interoperable and Re-usable. Figure 5.14 shows the results of executing a SQL query shown in Figure 5.13. This data is not annotated. In SQL, to create extra annotations, one would have to create extra columns or create new tables to store this extra metadata. However, in RDF, the result set is richer as shown in the json-ld formats demonstrated in the previous chapter. This data is FAIR because:

1. **Findable**: The data includes unique identifiers (URIs) for each entity, such as the species "Glossophaga_soricina" and its properties, making it easily locatable on the web. Additionally, it adheres to common standards such as SKOS (Simple Knowledge Organization System) for labelling and categorisation, enhancing its discoverability. However, one challenge is that if that URIs are not properly maintained, they become obsolete and data's findability suffers. To address this issue, URIs should use persistent identifiers and adopt best practices in URI design and maintainance. For instance, leveraging services such as DOI (Digital Object Identifier) infrastructure can provide redirection mechanisms that ensure links remain functional even if the underlying resources

move. Additionally, hosting RDF data on a stable institutional or community-driven domain, and maintaining proper versioning and redirects, can prevent link rot. This way, even if the underlying data or web structure changes, the URIs continue to resolve correctly.

2. **Accessible**: The data is publicly available and accessible through the provided URIs. This ensures that researchers and interested parties can retrieve the information without restrictions, promoting transparency and openness.

3. **Interoperable**: The data is represented in RDF (Resource Description Framework), a widely used format for expressing linked data. This facilitates integration with other datasets and systems, enabling seamless interoperability across different platforms and applications.

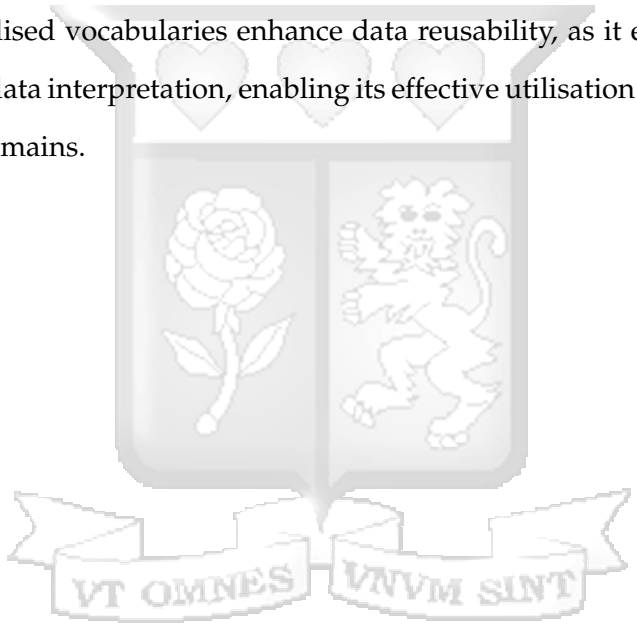
4. **Reusable**: The data is structured and annotated with descriptive properties, enhancing its usability for various purposes. By adhering to established vocabularies and standards, such as SKOS and RDF, the data becomes more interpretable and reusable in different contexts, contributing to its overall reusability.

6.4 Key Findings

In lieu of the preceding performance analysis and output assessment, the significance of leveraging RDF instead of SQL to store complex relationships becomes clear:

Firstly, RDF shows statistically significant, at 99.9% confidence level ($\alpha = 0.001$) better query execution times for tree-like data compared to SQL. This performance advantage is particularly crucial in production systems that host complex tree-like data that's hard to model where responsiveness to end users is critical. In the context of GeneNetwork, leveraging fast query execution provided by RDF enhances its search capabilities significantly.

In addition to the evident performance advantages, adopting RDF for data modelling confers a significant contribution to the principles of FAIR data. By structuring data in RDF, information becomes inherently more findable, as each entity is assigned a unique identifier, facilitating easy discovery through standardised querying methods. Moreover, RDF-encoded data enhances accessibility by providing open and transparent access points via web protocols, ensuring that data can be retrieved without constraints or barriers. Interoperability is also greatly promoted through RDF modelling, as it enables seamless integration, through federated queries, with other datasets and systems, fostering data exchange and interoperable interactions across heterogeneous environments. Additionally, RDF's semantic nature and adherence to standardised vocabularies enhance data reusability, as it ensures clarity and consistency in data interpretation, enabling its effective utilisation across diverse applications and domains.



Chapter 7

Conclusions, Recommendations and Future Work

7.1 Conclusions

This dissertation has explored the practical implementation and evaluation of a DSL designed to transform SQL tables into RDF, with a specific focus on addressing limitations within the GeneNetwork platform related to SQL queries. Through the development and testing of this DSL, several key findings and outcomes have emerged.

Firstly, the performance analysis conducted between SQL and SPARQL queries has demonstrated significant advantages of using RDF over SQL for modelling tree-like data, particularly in terms of query execution times. Across multiple benchmarks, SPARQL consistently exhibited faster response times compared to its SQL counterpart, especially as query complexity increased. This performance advantage is critical for enhancing the overall responsiveness of production systems like GeneNetwork.

Furthermore, by using RDF to make GeneNetwork data FAIR, external researchers and machine agents can access the data directly, without needing to scrape the GeneNetwork platform or contacting a database administrator for access to a restricted SQL system. Moreover, RDF enables seamless integration with external services like Wikidata, encouraging creative data modelling and data integration.

Despite its advantages, RDF adoption is not without challenges. Understanding core concepts such as triples (subject-predicate-object), URIs, and semantic relationships requires a learning curve, particularly for users unfamiliar with semantic web technologies. Additionally, mapping deeply nested or relational data into RDF triples can be complex and may lead to verbose representations. Compared to traditional relational databases, RDF may also incur higher storage overhead for large datasets, and lacks built-in mechanisms for transactional consistency across multiple updates. To mitigate this, RDF should primarily be used to store metadata—such as dataset descriptions, variable annotations, or provenance information—rather than individual sample values or raw data points, which are better suited to traditional relational databases. This separation helps manage storage overhead and performance concerns, as RDF triples can grow substantially in size for large datasets.

7.2 Recommendations

Researchers and developers within the domain of bioinformatics and data management can leverage the developed DSL to streamline the transformation of tree-like SQL data stored in flat table structures into more semantic RDF, enhancing data accessibility and interoperability. Institutions and organizations involved in data-intensive research can benefit from adopting RDF for improved data management, search capabilities, and adherence to FAIR data principles.

The generated RDF can be used to represent data as a Knowledge Graph over an extensible ontology. This knowledge graph can be used as the basis of Retrieval-Augmented Language Models that use some input sequence to search the knowledge graph and fuse the results with the language model output to generate more useful responses.

7.3 Future Work

Beyond creating a DSL that can convert SQL into RDF, a more flexible reasoning system based off a logic programming language such as Prolog, should be created to read the generated RDF. Such a system could be used for semantic reasoning and inference, leveraging the Logic programming language's logic-based approach

to derive insights and conclusions from RDF data. Future work in this direction could involve exploring the integration of this reasoning system with advanced AI techniques, such as Large Language Models (LLMs), deep learning and natural language processing, to enable more sophisticated and context-aware data models with reasoning capabilities. Additionally, efforts could focus on developing novel algorithms and inference mechanisms tailored to the specific challenges and requirements of RDF data, ultimately leading to more intelligent and adaptive systems for knowledge representation and semantic analysis.

References

Allemang, D., & Hendler, J. (2011). What is the semantic web? In *Semantic web for the working ontologist: Effective modeling in rdfs and owl* (pp. 1–12). Elsevier.

Anderson, K. R., Harris, J. A., Ng, L., Prins, P., Memar, S., Ljungquist, B., Furth, D., Williams, R. W., Ascoli, G. A., & Dumitriu, D. (2021). Highlights from the Era of Open Source Web-Based Tools [open access]. *J Neurosci*, *41*(5), 927–936. https://doi.org/10.1523/JNEUROSCI.1657-20.2020

Ashbrook, D. G., Arends, D., Prins, P., Mulligan, M. K., Roy, S., Williams, E. G., Lutz, C. M., Valenzuela, A., Bohl, C. J., Ingels, J. F., McCarty, M. S., Centeno, A. G., Hager, R., Auwerx, J., Sen, Ś., Lu, L., & Williams, R. W. (2019). The expanded bxd family of mice: A cohort for experimental systems genetics and precision medicine [open access]. *BioRxiv*, 672097. https://doi.org/10.1101/672097

Beckett, D. (2023). *Rapper - rdf parsing and serializing utility* [Raptor RDF Syntax Library]. Retrieved April 15, 2025, from https://librdf.org/raptor/rapper.html

Belleau, F., Nolin, M.-A., Tourigny, N., Rigault, P., & Morissette, J. (2008). Bio2rdf: Towards a mashup to build bioinformatics knowledge systems. *Journal of biomedical informatics*, *41*(5), 706–716.

Bizer, C., Heath, T., & Berners-Lee, T. (2011). Linked data: The story so far. In *Semantic services, interoperability and web applications: Emerging concepts* (pp. 205–227). IGI global.

Candan, K. S., Liu, H., & Suvarna, R. (2001). Resource description framework: Metadata and its applications. *Acm Sigkdd Explorations Newsletter*, *3*(1), 6–19.

Chen, C. P., & Zhang, C.-Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information sciences*, *275*, 314–347.

Freudenthal, M. (2010). Domain-specific languages in a customs information system. *IEEE software*, *27*(2), 65–71.

Genenetwork. (2022). Retrieved April 14, 2025, from https://genenetwork.org

Gunturkun, M. H., Flashner, E., Wang, T., Mulligan, M. K., Williams, R. W., Prins, P., & Chen, H. (2022). Genecup: Mining pubmed and gwas catalog for gene-keyword relationships [NIH grant support: P30 DA044223, R01 GM123489, U01 DA047638]. *G3 (Bethesda, Md.)*, *12*(5), jkac059. https://doi.org/10.1093/g3journal/jkac059

Harris, M. A., Clark, J. I., Ireland, A., Lomax, J., Ashburner, M., Ashburner, M., Foulger, R. E., Foulger, R. E., Eilbeck, K., Eilbeck, K., Lewis, S. E., Lewis, S. E., Marshall, B., Marshall, B., Mungall, C. J., Mungall, C. J., Richter, J., Richter, J., Rubin, G. M., … White, R. (2004). Gene ontology consortium: The gene ontology (go) database and informatics resource. *Nucleic acids research*, *32 Database issue*, D258–61. https://api.semanticscholar.org/CorpusID:22565487

Heath, T., & Bizer, C. (2011). Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology*, *1*(1), 1–136.

Hector Garcia-Molina, J. W., Jeffrey D. Ullman. (2014). On-line analytic processing. In *Database systems: The complete book* (Second, pp. 177–205). Pearson Education Limited.

Jupp, S., Malone, J., Bolleman, J., Brandizi, M., Davies, M., Garcia, L., Gaulton, A., Gehant, S., Laibe, C., Redaschi, N., et al. (2014). The ebi rdf platform: Linked open data for the life sciences. *Bioinformatics*, *30*(9), 1338–1339.

KissFlow, T. (2025). *What is rad? definition, meaning and benefits*. Retrieved April 14, 2025, from https://kissflow.com/application-development/rad/rapid-application-development-rad-meaning/

Kleppmann, M. (2019). Data models, and query languages. In *Designing data-intensive applications* (pp. 49–60). O'Reilly Media, Inc.

Konopka, B. M. (2015). Biomedical ontologies—a review. *Biocybernetics and Biomedical Engineering*, *35*(2), 75–86.

Lee, B., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hol-lenbach, J., Lerer, A., Sheets, D., et al. (2006). Exploring and analyzing linked data on the semantic

web. *Proceedings of the 3rd International Semantic Web User Interaction Workshop (SWUI06)*, 06.

Luz, M. P., de Matos Nogueira, J. R., Cavalini, L. T., & Cook, T. W. (2015). Providing full semantic interoperability for the fast healthcare interoperability resources schemas with resource description framework. *2015 International conference on healthcare informatics*, 463–466.

Mishra, P., Singh, U., Pandey, C. M., Mishra, P., & Pandey, G. (2019). Application of student's t-test, analysis of variance, and covariance. *Annals of cardiac anaesthesia*, *22*(4), 407–411.

Momtchev, V., Peychev, D., Primov, T., & Georgiev, G. (2025, April). *Expanding the pathway and interaction knowledge in linked life data* [Semantic Web Challenge 2009]. Ontotext. Retrieved April 14, 2025, from https://ontotext.com/documents/publications/2009/LLD_semantic_web_challenge_2009.pdf

Mulligan, M. K., Mozhui, K., Prins, P., & Williams, R. W. (2017). Genenetwork: A toolbox for systems genetics. In *Systems genetics* (pp. 75–120). Springer.

Mungall, C. J., McMurry, J. A., Köhler, S., Balhoff, J. P., Borromeo, C., Brush, M., Carbon, S., Conlin, T., Dunn, N., Engelstad, M., et al. (2017). The monarch initiative: An integrative data and analytic platform connecting phenotypes to genotypes across species. *Nucleic acids research*, *45*(D1), D712–D722.

Raimond, Y., & Schreiber, G. (2014, June). *RDF 1.1 primer* (W3C Note) (https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/). W3C.

Rakhmawati, N. A., Umbrich, J., Karnstedt, M., Hasnain, A., & Hausenblas, M. (2013). Querying over federated sparql endpoints—a state of the art survey. *arXiv preprint arXiv:1306.1723*.

Ramez Elmasr, S. B. N. (2016). Basic sql. In *Fundamentals of database systems* (Seventh, pp. 456–464). Pearson Education Limited.

Robinson, P. N., & Mundlos, S. (2010). The human phenotype ontology. *Clinical genetics*, *77*(6), 525–534.

Seaborne, A., & Prud'hommeaux, E. (2008, January). *SPARQL query language for RDF* (W3C Recommendation) (https://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/). W3C.

Shadbolt, N., Berners-Lee, T., & Hall, W. (2006). The semantic web revisited. *IEEE intelligent systems*, *21*(3), 96–101.

Sloan, Z., Arends, D., Broman, K. W., Centeno, A., Furlotte, N., Nijveen, H., Yan, L., Zhou, X., Williams, R. W., & Prins, P. (2016). Genenetwork: Framework for web-based genetics. *Journal of Open Source Software*, *1*(2), 25.

Soldatova, L. N., Aubrey, W., King, R. D., & Clare, A. (2008). The exact description of biomedical protocols. *Bioinformatics*, *24*(13), i295–i303.

Sparql implementations. (2022). Retrieved April 14, 2025, from https://www.w3.org/wiki/SparqlImplementations

Spinellis, D. (2001). Notable design patterns for domain-specific languages. *Journal of systems and software*, *56*(1), 91–99.

Vliet, H. V. (2007). Rapid application development and dsdm. In *Software engineering: Principles and practice* (pp. 57–61). John Wiley & Sons Hoboken, NJ.

Williams, A. J., Harland, L., Groth, P., Pettifer, S., Chichester, C., Willighagen, E. L., Evelo, C. T., Blomberg, N., Ecker, G., Goble, C., et al. (2012). Open phacts: Semantic interoperability for drug discovery. *Drug discovery today*, *17*(21-22), 1188–1198.

Appendices

Appendix A

Similarity Report

signature-approved-dissertation.pdf

ORIGINALITY REPORT

14%	12%	9%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	git.genenetwork.org Internet Source	3%
2	augustq.blogspot.com Internet Source	1%
3	Submitted to Strathmore University Student Paper	1%
4	Submitted to Universiti Tunku Abdul Rahman Student Paper	1%
5	Kuldeep Singh Kaswan, Jagjit Singh Dhatterwal, Anand Nayyar. "Digital Personality: A Man Forever - Volume 1: Introduction", CRC Press, 2024 Publication	1%
6	biblio.ugent.be Internet Source	1%
7	issues.genenetwork.org Internet Source	1%
8	unece.org Internet Source	<1%

9	assets.researchsquare.com Internet Source	<1 %
10	"Encyclopedia of Big Data Technologies", Springer Science and Business Media LLC, 2019 Publication	<1 %
11	www.nomenclature.info Internet Source	<1 %
12	research.wur.nl Internet Source	<1 %
13	eapj.org Internet Source	<1 %
14	repository.psa.edu.my Internet Source	<1 %
15	ouci.dntb.gov.ua Internet Source	<1 %
16	vdoc.pub Internet Source	<1 %
17	Tristan V de Jong, Yanchao Pan, Pasi Rastas, Daniel Munro et al. "A revamped rat reference genome improves the discovery of genetic diversity in laboratory rats", Cold Spring Harbor Laboratory, 2023 Publication	<1 %
18	scholarworks.sjsu.edu Internet Source	

Appendix B

Ethical Clearance Certificate

Strathmore UNIVERSITY

30th May 2024

Mr Kilyungi Bonface,
bonface.kilyungi@strathmore.edu

Dear Mr Kilyungi,

RE: Unlocking Biomedical Data for AI Health Research in Africa Using GeneNetwork as an Example

This is to inform you that SU-ISERC has reviewed and **approved** your above **SU-masters** proposal. Your application reference number is **SU-ISERC2265/24.** The approval period is from **30th May 2024 to 29th May 2025.**
This approval is subject to compliance with the following requirements:

i. Only approved documents including (informed consents, study instruments, MTA) will be used.
ii. All changes including (amendments, deviations, and violations) are submitted for review and approval by SU-ISERC.
iii. Death and life-threatening problems and serious adverse events or unexpected adverse events whether related or unrelated to the study must be reported to SU-ISERC within 72 hours of notification.
iv. Any changes anticipated or otherwise that may increase the risks or affected safety or welfare of study participants and others or affect the integrity of the research must be reported to SU-ISERC within 72 hours.
v. Clearance for the export of biological specimens must be obtained from relevant institutions.
vi. Submission of a request for renewal of approval at least 60 days prior to the expiry of the approval period. Attach a comprehensive progress report to support the renewal.
vii. Submission of an executive summary report within 90 days of completion of the study to SU-ISERC.

Before commencing your study, you will be expected to obtain a research license from National Commission for Science, Technology, and Innovation (NACOSTI) https://research-portal.nacosti.go.ke/ and obtain other clearances needed.

Yours sincerely,

Mr Ambrose Rachier,
Chairperson; SU-ISERC

Ole Sangale Rd, Madaraka Estate. PO Box 59857-00200, Nairobi, Kenya. Tel +254 (0)703 034000
Email admissions@strathmore.edu www.strathmore.edu