A Methodology for Automating Graph Construction and Evaluation

Congcong Xing

BSc Biological Science

MSc Financial Risk Management

A Dissertation submitted in fulfilment of the

requirements for the award of

MSc

to

Dublin City University

Faculty of Engineering and Computing, School of Computing

Supervisors: Prof. Mark Roantree and Dr. Andrew McCarren

September 2021

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of MSc is entirely my own work, and that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed:

ID No.: 18214880

Date:

Acknowledgements

I would first like to thank my supervisors Prof. Mark Roantree and Dr. Andrew McCarren, for giving me the great support during my MSc. Without their constant encouragement, guidance and pursuit of perfection, this thesis would not exist.

There are too many people at DCU that I would like to thank. But I am not going to attempt to name everyone because I fear forgetting many, so I would like to extend my thanks to all the students and postdocs at the School of Computing.

Finally, I would like to thank the VistaMilk SFI Research Center and the Department of Agriculture, Food and Marine on behalf of the Government of Ireland for funding my research under grant no. SFI/12/RC/2289.

Table of Contents

List of Figures

List of Tables

Abstract

Congcong Xing

A Methodology for Automating Graph Construction and Evaluation

Graphs and graph analytics facilitate new approaches to machine learning. They also provide the ability to extract new insights from the same datasets as used in traditional machine learning experiments. For this reason, many researchers are seeking to exploit graph databases in pursuit of better performance for their predictive models. However, the construction of a graph from relational or flat models such as CSV files is not a straightforward transformation. A careful selection of nodes and relationships is required to ensure an optimal construction of the target graph. Overly large graphs can cause performance issues for a number of graph algorithms and thus, graph compression is an important part of the construction process. This research has 2 components: the usage of graphs to integrate multiple data sources and a graph transformation methodology to create the integrated schema and populate the graph. Our approach to validation uses link prediction and community detection graph analytics to evaluate the graphs built using our methodology.

Chapter 1

Introduction

Data analytics helps individuals and organizations make sense of data and in that context, better data analytics often leads to better decision making. In order to perform data analytics, a well-constructed dataset is essential and the richness of the dataset can determine the quality of the result gathered by analytics. In many cases, data integration is a fundamental part of this process as data often reside in different locations. As this is a well-understood process for many application areas, this research will also investigate the use of graphs for modeling data and graph analytics. We begin this dissertation in section 1.1, with the background for this topic where we discuss problems and our motivation for doing this research. In section 1.2, we introduce graph analytics before we outline our research plan in section 1.3. Finally, we present the structure for the dissertation in section 1.4.

1.1 Introduction and Background

Data integration is the process of combining data coming from different sources/systems to provide a panoramic view to the end-user with meaningful and valuable information [13]. Data from different sources is not connected will lead to the limitation of data extraction and analytics. Connecting different data from different sources to have a unified view of data is the process of data integration. It is im-

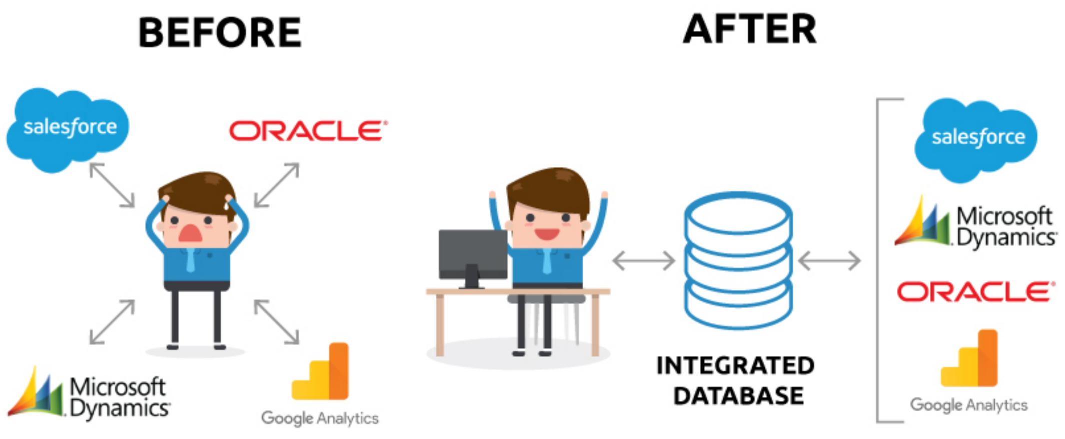


Figure 1.1: Data Integration

portant to do the data integration process as most real-world data is generated from different sources and the integration process itself will have a direct impact when extracting and analyzing heterogeneous data. There are many methods and approaches to integrate data from different sources. Different data integration methods have different features, most common processes include data collection, cleaning and reformatting, data matching, connecting, and data loading. Among them, data matching and connection steps are the key steps where the major challenges will be encountered. The data presented to the user upon completion of the integration process should maintain its integrity and therefore can be trusted. By applying data analytics on the integrated data, the end-user can often make better business decisions compared to analyze the different data produced from different systems in isolation. A simplified version of what is data integration and what we can achieve with it can be viewed in figure 1.1. On the left-hand side, we can see prior to data integration, the user has to deal with data coming from different systems which can lead to huge time and effort. On the right-hand side, the unified integrated data store containing all the information gathered from different systems will be provided to the user as a unified view for easy access which will be a much-preferred solution to many.

The data integration process is well utilized by many in fields such as the financial sector to fulfill many requirements such as market prediction and risk analysis. However, not so much for fields that traditionally were not technology-driven, such as the Agricultural (Agri) sector. The reason is simple: there was not enough data. However, with more and more organizations like Teagasc (agriculture and food development authority in Ireland) [58] setting their mission to support science-based innovation in the agri-food sector, it is time to have another look at the benefits of data integration in such field. Data warehouses [24] are a means of gathering multiple sources of data in a single repository and there have been examples of data warehouses specific to Agri data [33]. While some research eg. [56] provided solutions to create dynamic data marts from the integrated data, there is little evidence in the Agri domain of attempts to automate the integration process itself.

In order to deliver a broader bio-economy that will underpin profitability, competitiveness, and sustainability, Teagasc has partnered with many information communications technology (ICT) research institutes including the VistaMilk SFI Research Center [60]. The VistaMilk SFI Research Centre is a collaboration between Agri-Food and ICT research institutes and leading Irish/multinational food and ICT companies. It focuses on developing new and advancing existing electronic monitoring and actuation technologies. Adopting state-of-art devices, large-scale sensor/system generated data are delivered through advanced network and communication technology infrastructures.

With those modern technologies, the amount of data that is produced in Agri field makes it a good candidate for data analytics. By applying analytical techniques to those datasets acquired, the research center has already made great process for enhancing pasture-based dairy production, improved processability, and generated novel, higher-value-added products. However, a large number of data resources in the Agri field regularly originate from disparate sources and can be disconnected with different formats and types. In addition, the different data are often generated within specific areas for specific scientific focuses. We believe it is worth exploring with data integration, if the integrated dataset could be more helpful to further advance data analytics and thus, generate more valuable results. In order to archive such a goal, a solid integration solution is required.

1.1.1 Motivation and Problem Statement

The integration process for data produced in the Agri domain (and others) can be very time-consuming. That is because manual intervention is often required to handle the complex disconnected source data. In order to improve the inefficiency of that process, we want to explore what is required to deliver a robust automated system for data integration. Thus, the first question we ask is:

- Is it possible to isolate those integration processes that are heavily manual?

In other words, can we investigate where most manual effort is required in data integration? Traditionally, data integration often involves manual effort to create mappings between different datasets. Such processes can be very time-consuming and can lead to undesired results because of human errors among other issues. The user often is responsible for extracting the data from different file stores, performing the integration, and eventually having the option to load the result into a different data store for further use.

The next question we ask is:

- Can we identify precise operations that can optimize the manual process?

This process can often be improved by adopting some generic software/tools so that less manual work is required. However, with the rapidly increasing volume of dynamic data, adjustments or enhancements are still required to support the importation of new files into the system. At this point, the process has evolved to reduce the manual effort. In delivering this solution, the user will utilize some software/tools to extract data from multiple data stores and complete the data integration process. Any modifications to the source files/systems will require manual adjustments by the user or developer to the system. We believe the development of a fully automated integration system is needed to remove the manual work from the process. The goal is to reduce the time and effort required from the end-user, so they can spend more time on data analytics. In such a process, the end-user does not participate in the integration process but only serves as a consumer of the integration result. At this point, we ask our next question:

- How can we validate the integrated result?

The result of data integration is a unified data store containing information from all the processed input source information. It is essential that the integration process maintains not only the integrity of the source data but also their schema. For this, we will need to provide a solid testing strategy to ensure no data loss during the integration process and to validate the schema of the final integrated data. Until now, the solution is fully automated and it is up to the end-users how they want to use the integrated data, Which motivates our next question:

There are many data science algorithms that have been proposed by many researchers which can help to gain insight into existing datasets. With many integration solutions leveraging relational databases to persist the integrated data, it is up to the user to implement the specific algorithms as those databases rarely come with built-in functions. This might lead to inaccurate results depending on the quality of implementation. In addition, for data generated for a field like Agri, one with less domain knowledge will feel difficult to interpret the integrated datasets as the information gathered by different systems might not be descriptive enough. We believe a graph database like Neo4j [39] is able to address these challenges as it provides a well-designed user interface with visual capability and also provides an enterprise tested data science library [38] [37] for the end-user. Users can apply algorithms such as *community detection* [50] or *link prediction* [31] directly from the graph DB, which can save the user both time and effort and allow them to concentrate more on analytics.

1.2 Graph Analytics

In a simple dataset where there are few relationships within the data and little structure, tabular data provides a simple means of data storage. Even complex analytics using Agri data such as deep learning approaches [43] still used data in a flat, simple format. However, once the data structure gets more complex, especially for a dataset that is a result of combining multiple data sets, the flat model can neither represent, nor help to visualize the data. In addition, in order to apply a data science algorithm, the user often will need to code the algorithm themselves, and depending on the technical skill of the users, the quality of the end result produced can be inconsistent. Graph databases on the other hand have become common in recent years as they allow queries and analyses that provide new insights into the data. Graph databases such as Neo4j [39] provide a series of well-maintained data science libraries and instructions, which users can use to directly apply data science algorithms with minimum technical skills.

A graph (denoted as G = (V, E)) is a mathematical structure to model paired objects and their relations. It consists of a non-empty set of vertices or nodes V and a set of edges E. A vertex a represents an endpoint of an edge. An edge joins two vertices a, b and is represented by a set of vertices it connects.

Early application of graph theory is known as *Euler Circuits and The Konigsberg Bridge Problem* [8], where graphs were used to solve geographical problems. Graphs are also good to model any data with relationships and are well used in physical, biological, social network, fraud detection and cybersecurity areas for the development of the applications [26].

In the computer science sector, website networks can be represented as a whole graph, links presented as edges and each edge connects to a distinct web page node. The idea can be used in semantic web/computational linguistic and therefore to be developed in ontology matching that is crucial and well used in the data integration process. Graphs can optimize data storing, modeling, and traversing when data is in a graph structure. In fundamental science topics like chemistry, biology, physics, and mathematics, when objects can connect with each other as logical causal relationships, graphs can easily visualize and help reveal unknown interactions.

1.3 Approach

Graph databases allow researchers to obtaining new insights, even from existing data. However, most data resides in relational databases, flat files like Excel or CSV, or some form of web data. None of these formats model data as a graph with data points and relationships. Existing model transformation methods can be problematic when metadata or schema information is unavailable. Furthermore, many datasets require new levels of optimization before a good design graph schema can be formed. Thus, in order to help users to construct a robust graph for data analytics, perhaps based on data integrated from different data sources, we propose an automated integration platform that is able to take multiple data sources in tabular form and produce a well construct graph as an end result.

Our research comprises 3 components:

1. A graph-based integration platform.
2. A transformation methodology to transform an integrated relational data source to a graph format.
3. An evaluation methodology, both to test this transformation and to evaluate analytical functions on the new graph database.

1.3.1 Research Questions

At this point, we highlight the primary goal and contribution of our work: providing a fully automated data integration solution that leverages redundancy in data to organize graph data in a way that improves query performance. There are a number of research questions that must be answered as part of this research and dissertation in order to archive the overall goal.

- Can we integrate scattered datasets that is continuously being generated? One example of such datasets would generated from the modern Agri domain. Such integration solution should be both lightweight and flexible. It is one of the key elements of the proposed system that it should be intelligent enough to perform the integration on unseen files while at the same time, limit the creation of any redundant data.
- Can we verify the integration process so that we have not corrupted the data? We need to ensure the result of the integration is a true representation of the data and therefore, can be trusted.
- Can the solution automatically construct a graph from the integrated schema? The graph transformation should be done automatically by leveraging the integrated schema with minimal human overhead.
- Can we demonstrate that this can perform more powerful analytics that cannot be achieved using relational systems. We need to verify that we can run graph analytics such as centrality or community detection to extract new insights from this data.

1.4 Summary

In this chapter, we explained what is data integration and briefly discussed the benefits of performing such a process. With the constantly rising volume of heterogeneous data in the Agri section produced by modern technologies. Researchers are looking for a solid integration solution to deliver good quality integrated datasets to gain more insights. While many existing integration solutions can be adopted to Agri field, we believe the traditional data integration solution will not be sufficient as they rarely provide researchers any build-in functionalities to apply data science algorithms. Because many data science algorithms play a significant part in today's data analytics, we propose our graph based integration solution which not only ensures the data quality but also provides a well-maintained ready-to-use data science library.

The remainder of this thesis is structured as follows.

Chapter 2 provides a discussion on the state of the art in both data integration and graph model transformation.

In chapter 3 we go over the methodology that is proposed by us. In this chapter, the system architecture will be firstly outlined, followed by detailed description for each of the components in our system including *Initial Data Loader*, *Ontology Matching Processor*, *Metadata Integrator*, *Instance Data Integrator*, *Integrated Instance Data Extractor* and *Graph Data Loader.*

Chapter 4 provides a detailed explanation of the schema integration part of our system. We walk through every stage of this process, which includes *initial loading of data*, *metadata extraction*, *ontology matching*, *metadata integration* and *instance data integration.* One of the key aspects of any integration methodology is data integrity, that is, there should be no data loss upon completion of the integration. We evaluate our integration methodology by comparing the results of multiple queries executed against the original source data as well as the integrated data store.

In chapter 5, we described the final part of our system, the *graph transformation.* We give an introduction about graph transformation in general and then we go over different stages of our transformation methodology. This process includes *Meta Analysis*, *Attribute Classification* and finally *Graph Construction.*

A detailed evaluation is carried out on the graph generated using our graph transformation methodology in chapter 6. In this chapter, we firstly go over the hardware and software setup of our experiment. That is followed by a multi-dimensional evaluation process, where we examine the pros and cons of our methodology in respect of *storage*, *performance*, *data integrity* and finally, the *analytical benefits.*

The dissertation concludes with chapter 7. In this chapter, provide a summary of the methodology proposed by us, the evaluation we have done, and the results presented in this dissertation. This is followed by some future works worth exploring, which were identified during our study.

Chapter 2

Related Research

In this chapter we provide an overview of previous work related to the work carried out in this thesis. We begin with *Data Integration* in section 2.1. Following this, we examine the state of the art for *Graph Transformation* in section 2.2. In section 2.3, we summarise this chapter.

2.1 Data Integration

The aim of this section is to discuss and analyze research into data integration. A growing number of fields are trying to adopt the data integration process in order to produce richer data for data analytics.

In practice, there are often some challenges integrating data. In [61], the authors detailed computational challenges for data integration processes within the biomedical sector, a number of issues were specified including but not limited to:

- Data size;
- Format and schema design;
- Biases on data extraction;
- Collection and cleaning;

- Information efficiency;
- Data protection policies;
- Scalability.

The challenges faced in the traditional data integration process are further compounded with the introduction of big data. In [19] the authors details these challenges, in addition, they explore the progress that has been made so far to address those challenges.

Obtaining a dataset that is both high in quality as well as accuracy is hard, even in domains where high standards are applied [18, 55]. To perform data integration, the most traditional method involves a manual method providing support to semantically integrate the data. There are many limitations of manual work, for example, accuracy as well as bias and knowledge differences among different domain experts. Most importantly, as the size of the dataset increases, human effort is not enough to complete the task in a reasonable time period. Thus, many researchers are seeking automatic methods for data and knowledge fusion.

Many attempts have been made to achieve an automated solution, often through an ontological approach. The ontology approach is a semantic technology application to express data in a semantic way that deals with uncertain representations [46, 53]. For such approaches, the ontology matching/alignment process plays an important role when representing uncertainties, a reliable algorithm needs to be developed to assess these uncertainties and output a data integration process.

The ontology alignment process can be either supervised or non-supervised, In [46], the authors developed a tool to support the supervised ontology matching process. The proposed tool relies on intuitive user inputs rather than on strong skills in ontology and Semantic Web technology. As stated by the author, the primary goal of the tool is to support the systematic conversion of a given dataset into an independent and self-contained ontology in Ontology Web Language (OWL). In order to complete such a conversion, the user will have to manually characterize each of the data columns into *ID*, *Resource* or *Attribute* type through a provided user interface. The author stated such a manual alignment process is one way to limit the level of uncertainty. However, it was not tested using real-world datasets which are far more complex. In addition, because of the hard dependency on user inputs, it becomes unrealistic once the scale of the system becomes significant or in the presence of heterogeneity. We adopted a hybrid approach where we implement a threshold-based alignment solution based on graph but we use user input as an optional calibration step. Therefore we can achieve a full level of automation, but at the same time, still have the flexibility to receive contributions from domain experts to increase the result accuracy.

In [63], the authors proposed a hybrid method involving the building of both a local and global ontology base to integrate data with manually constructed mapping rules. In the proposed solution, the local ontology will firstly be built, which is derived from the local schema. By analyzing the local ontology, the system will be able to generate an OWL-based global ontology with the help of Protégé. Protégé supports the creation, visualization, and manipulation of an ontology in various representation formats and enables users to build ontology for the Semantic Web. However, in order to achieve semantic interoperability and decompose the global query into one or more sub-queries over the local data sources, the user must define the relations between the global ontology and the local ontology. These relations are named mapping rules. However, the manual effort required to define such mapping rules means the solution will not be able to scale once the data gets more complex. Our system, on the other hand, does not have such a scalability issue.

In [47], a system called *IncMap* has been developed by the authors for data integration based on ontology matching techniques. The system uses a global ontology and features a fully automated system based on the graph to create mappings between the source data files and the global ontology to complete the data integration process. The system mainly consists 5 steps including *Schema Graphs*, *Reasoning and Patterns*, *Matching Step*, *Fixpoint Computation* and *Mapping Generation*. As a first step, the system will create source and target schema using IncMap+ model.

The system will iterate over all schema elements in the relational schema and the ontology, as result, two graphs will be created. Second, the proposed system will apply reasoning techniques on the input ontology and use heuristics to annotate patterns on the source database. After that, the system will perform initial matching based on the source and target graph. Next, the system computes a fixpoint computation using the PCG which will be used to refine the calculated match scores. Finally, mappings are generated from the correspondences resulting from the PCG. Although the solution promises very positive results compared to other systems it still significantly struggles with the more complex scenarios as well as the two real-world cases (Geodata and Oil & Gas). Moreover, its primary focus is on relational database integration. Our system initially will be focusing on integrating data in tabular forms generated in the Agri domain.

In [17], the authors propose an ontology-based integration approach for web analytics within the e-commerce domain. The proposed approach is capable of collecting and integrating data from different e-shops. The integrated data is then used for advanced data mining procedures which ultimately provides better customer analytics which can lead to better sale results. The solution specifically focused on three main sources of data coming from different web tracking methods, including Google Analytics, Piwik, and specific web scraping methods in the scope of SME E-Compass project. Therefore the solution is not generic enough to be adopted by datasets generated by different systems. That is different from our proposed system which is not designed for a specific field can be utilized by sectors like Agriculture.

In [25], the authors demonstrate the urgent need of an ontology-based data integration solution for NoSQL [49] databases. They outline the maturity of such solutions in the context of RDBMS [1]. However, due to the widely used NoSQL databases such as HBase, and the lack of such solution for these types of data store, an ontology-based semantic integration system for column-oriented data has been proposed to satisfy these requirements. The system proposed leverages tools and frameworks that are currently adopted by integration systems that are primarily focusing on RDBMS. The proposed implementation consists four parts. Starting from schema generation, which is crucial for NoSQL databases as they do not have the schema concept with their dynamic nature. In order to solve this problem, the authors implemented an approach that performs both online and offline schema generation. The proposed system creates a lookup table in HBase for each table whose schema has to be extracted. Schema extraction takes place by finding the fittest individual, this process commences after the lookup table is constructed. The extracted information is then converted to suitable OWL primitives and mapped to an OWL ontology. The system then performs ontology alignment merging using COMA++ [6] and forms the global ontology. Custom SPARQL endpoints were implemented for the system to query the data through the constructed global ontology. The results of the queries can be interpreted with a reasoner. This solution is designed specifically for HBase. However, it can be altered to use other NoSQL databases. However, it cannot be used in the Agri sector as most of the data is tabular in nature and rarely stored in NoSQL Databases.

In [14], the authors demonstrated a way to represent the information stored in NoSQL databases through the use of ontologies. The process has three stages, *ontology creation for each data source*, *ontologies alignment* and finally *global ontology construction.* The implementation uses the global ontology as the query entry point, it provides a bridge query language that supports translation from SPARQL queries using specific APIs for each database source, such as Cassandra NOSQL database. The system is able to reason on the elements of the ontologies and retrieves information efficiently. However, because of the need for such bridge query language, the user will not be able to use the proposed solution unless the bridge query language has been extended to support the input source type. In addition, because of the lack of community support on such query language, the user might find certain required functionalities missing from implementation. Our proposed work will leverage a matured enterprise graph solution that provides a query language that has been used by many enterprises as well research groups.

In [30], the authors present a semi automated solution to transform heterogeneous data to OWL format as an approach to data integration. The proposed solution uses a four-step transformation approach to transform the source files into OWL files as the integration result. While the constructed OWL files are query-able and have been validated by the author to ensure data integrity. The query language itself introduces a learning curve for the end-user. Moreover, the presentation of the data is not ideal for the end-user if the user wishes solely to browse through the integrated data. In contrast, our integrated datasets will be stored in a well-constructed graph with much better accessibility and user experience.

In [42], the ontology matching tool YAM++ applies machine learning, information retrieval, and graph matching techniques to provide a higher quality matching result. However, the limitation of Yam++ is also very clear, it is only able to work with data that is in OWL format meaning a structured ontology of a dataset is required as input. Not all datasets have a supplementary OWL ontology, meaning that ontology construction is a necessary pre-step before YAM++ can be utilized for ontology matching.

2.2 Graph Transformation

The aim of this section is to discuss and analyze research into graph transformation. As explained in section 1.2, a Graph is a mathematical structure to model paired objects and their relations. It can provide different forms of analyses, offering new insights into existing data. For this reason, many researchers have attempted to implement methods to convert existing datasets into a graph model. This process is defined as graph transformation [22]. In a graph data model, objects are represented as *Nodes* and connected by *Edges*. Which can be seen as a triplet and therefore can be modeled using RDF [32]. A triple is a structure that consists of `subject-Predicate-object` [44]. Most of the solutions that are proposed focus on transforming a dataset into RDF format.

For many years, the focus for graph transformation was on converting relational data into graph models, where Primary Keys (PKs) and Foreign Keys (FKs) are used for mappings between instance data. *Direct Mapping* [4] and *R2RML* [16] are the complementary W3C recommendation (specifications) which define the language and algorithm respectively that are used to transform relational databases into RDF graphs.

In [34] and [35], the authors implement a mapping language named xR2RML to generate and add namespaces to the original data and convert data into RDF [27]. The xR2RML is built as an extension to R2RML. The mapping language relies on RML for the handling of various data formats. The mapping implementation relies on the assumption that databases provide a declarative query language to translate into RDF, and, assumes that domain ontologies already exist with classes and properties which can be used to transform a data source into RDF triples.

While a lot of data are stored and processed in relational databases, data can also be produced in other formats such as CSV flat files.

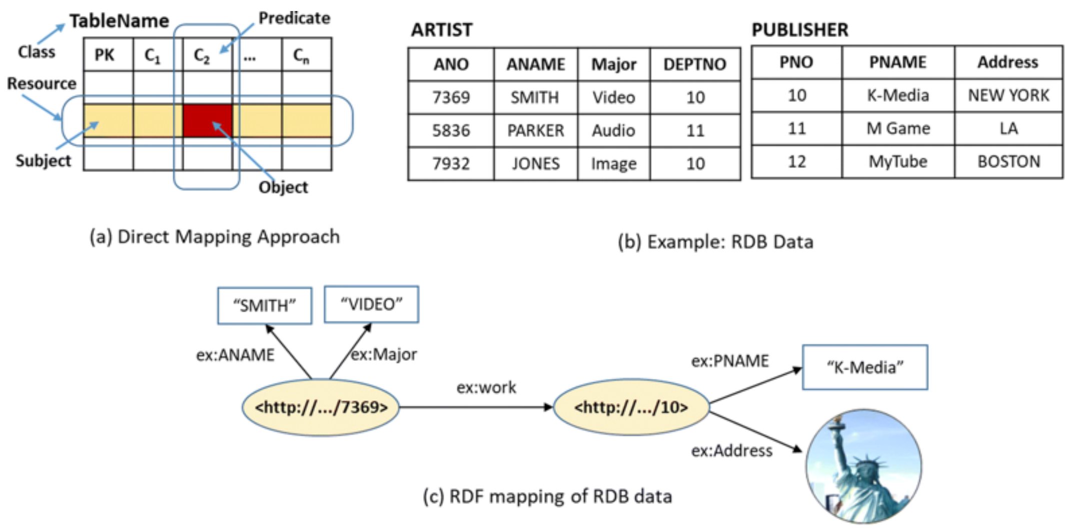
More challenges are encountered when trying converting the *plain data* into *graph formatted data*: mapping (in other words, the process of generating edges) is an issue of significant note. In a plain file/table, there are no relations between rows due to the lack of PKs and FKs, therefore, a mapping strategy is required to generate these relationships (and subsequent graph edges).

In [28], the authors provide a solution that utilizes direct mapping rules. The rules are essentially used as an RDF vocabulary. The proposed solution is shown to work on a governmental CSV dataset. While the vocabulary method can be adopted easily, it is often used within a specific domain where fixed domain knowledge is present.

The rules of Direct Mapping are straightforward, it provides a simple mapping method to transform Relational Databases (RDB) to RDF. Some key elements of the approach:

- Tables in RDB are mapped as Classes;
- Attributes are mapped to Properties;

- Rows are mapped as entities/resources with IRI creation;
- Each cell is a value;
- Foreign keys are mapped directly to the IRI of corresponding entities/resources;



ANO	ANAME	Major	DEPTNO
7369	SMITH	Video	10
5836	PARKER	Audio	11
7932	JONES	Image	10

PNO	PNAME	Address
10	K-Media	NEW YORK
11	M Game	LA
12	MyTube	BOSTON

Figure (2.1) Direct Mapping, Note. From "R2LD: Schema-based Graph Mapping of relational databases to Linked Open Data for multimedia resources data" by Zhao, Z., Han, S. & Kim, J., 2019, Multimed Tools Appl 78, 28835–28851 (2019). https://doi.org/10.1007/s11042-019-7281-5. CC BY-NC.

The example of the direct mapping process can be found in figure 2.1. The limitation of such a process is the lack of control in how the RDF is generated. In addition, the requirement of pre-defined rules poses difficulties with large scale complex datasets. Our proposed system does not mandate any human intervention. As opposed to *Direct Mapping*, *R2RML* (Relational to RDF Mapping Language) provides more flexible mappings for relation representations. Customized mapping rules are enabled in R2RML by the customized mapping file. The application of **Logic Table** breaks the restriction of the RDB structure making the target RDF data capable of meeting users' target needs and at the same time, does not modify the original RDB data structure.

In [11], the authors applied two approaches for converting CSV files to a graph. The first approach is the typical way to transform using R2RML mappings, this approach creates one TriplesMap for each column corresponding to a slice of a dimension. In the second approach, they aim to reduce the size of an R2RML mapping for statistical data by incorporating an iterator variable into RMLC. This mapping language has been equipped with several features to be able to deal with the heterogeneity of tabular data. The author stated that the only difference between those TriplesMaps is the name of the column, which provides a unique identifier to the TriplesMap object and the means of accessing the data of each column. By incorporating a variable that references the target columns, RMLC-Iterator reduces the size of the mapping while maintaining the semantics of the R2RML mapping. This variable is formalized in the mapping language by incorporating four new properties to the Logical Table object. The proposed solution also provides a mechanism to transform the RMLC mapping to R2RML mappings. The authors conclude that the second approach using RMLC mapping drastically reduces the size of the R2RML mapping document. However, we wish to highlight that the RMLC mapping still requires manual generations which can be error-prone and time-consuming.

Mapping is used by the majority of the solutions that have been proposed so far. However, In [5], the authors outline that at the enterprise level, large companies such as Facebook, store social media data in both MySQL and HBase [62] (social graph storage). Where in their MySQL database, graph interactions are stored in different tables separately as `Object` (graph node) and `Association` (graph edge). With such system architecture, tabular data can be converted to a graph directly. However, such a setup can be very difficult to achieve because of the requirements of significant hardware and software resources.

Most approaches to graph construction begin with relational datasets where Primary Keys (PKs) and Foreign Keys (FKs) are used for mappings between instance data. While this course grained approach ensures that the graph is reasonably compact, it will feature a somewhat limited number of relationships between the nodes, specifically only those mapped from PKs and FKs. Crucially, they often assume some level of schema information which is not always available with datasets acquired from various sources across the web. With a more fine-grained approach, each tuple or row cell can become a node which maximizes the relationships between nodes. However, this requires an approach to optimize graph construction by using redundant data to avoid excessively large graphs.

2.3 Summary

In this chapter, we provided an overview of different approaches for data integration. We discussed many attempts that have been carried out in the past and outlined our contributions and differences compared to other proposed solutions. In addition, we examined the state of the art in graph transformation which forms a crucial part of our overall methodology.

In the next chapter, we will explain the methodology of our proposed automated graph based data integration platform.

Chapter 3

Integration Methodology

Data Integration is a major process in data warehouse construction. It is essential for combining data from different data sources which contain heterogeneous data into a unified view of data to be used for downstream data analytics. Historically, data integration has often been performed by individuals with mostly manual efforts on data from different systems or applications. However, manual effort even with the help of some generic software tools, means that extracting the required information from disparate streams of data in a timely fashion can be extremely time-consuming, difficult, and error-prone. In order to address this problem, an automated data integration platform was developed to speed up the process of integrating data based on graph theory. By utilizing the integration platform, individuals or businesses will be able to complete ad hoc task as well as scheduled batch processing jobs by having minimum manual interventions, the result achieved will not only be accurate but also gives significantly more insight which can then be used to deliver more insightful decisions. In section 3.1, we present an integration architecture that uses an existing platform to match data between schemas, and subsequently improves the automation process. We then provide a detailed description in section 3.2 of how data is transformed using the architecture and a case study from the agricultural domain. In section 3.3, we summarise the chapter.

3.1 System Architecture

For this part of our research, we constructed an integration framework based on the existing YAM++ Graph Based Integration Platform [42]. Using this approach, data from heterogeneous sources will go through an initial processing phase at which they will be divided into two categories: metadata category and instance data category. The data that is categorized as metadata will be transformed to the ontology description format (RDF/OWL) and fed into the ontology matching platform, YAM++, which will be processed later. With help from YAM++ APIs, the resulting output of the ontology matching process will be an XML alignment file. The matching results stored in the alignment file will be used as a key driver during the metadata integration. The connected metadata will then be used as a blueprint by the platform to complete the instance data integration. We will now proceed with a detailed system flow diagram presented in figure 3.1. The platform consists of a small number of critical components to deliver the final integrated dataset and these are described briefly in the rest of this section. In order to validate our system, we will integrate a set of files from the Agri domain that contain heterogeneous data generated from different systems. The set of files will be described in detail in section 3.2. A more detailed discussion on data integration will take place in chapter 4 and graph construction is described in detail in chapter 5.

3.1.1 Ontology Matching

Ontology matching is the first step in the overall process and figure 3.1 shows the involvement of a *Loading* stage, a *Producer* stage, and finally the *Ontology Matching* process itself. In our earlier work [57], we showed that by extracting small samples of data, we could build a sufficient profile of the data to build a plan for processing or integrating that data. The same concept is applied here but in a more formal approach.

Initial Data Loader is the first component in the workflow and loads the data

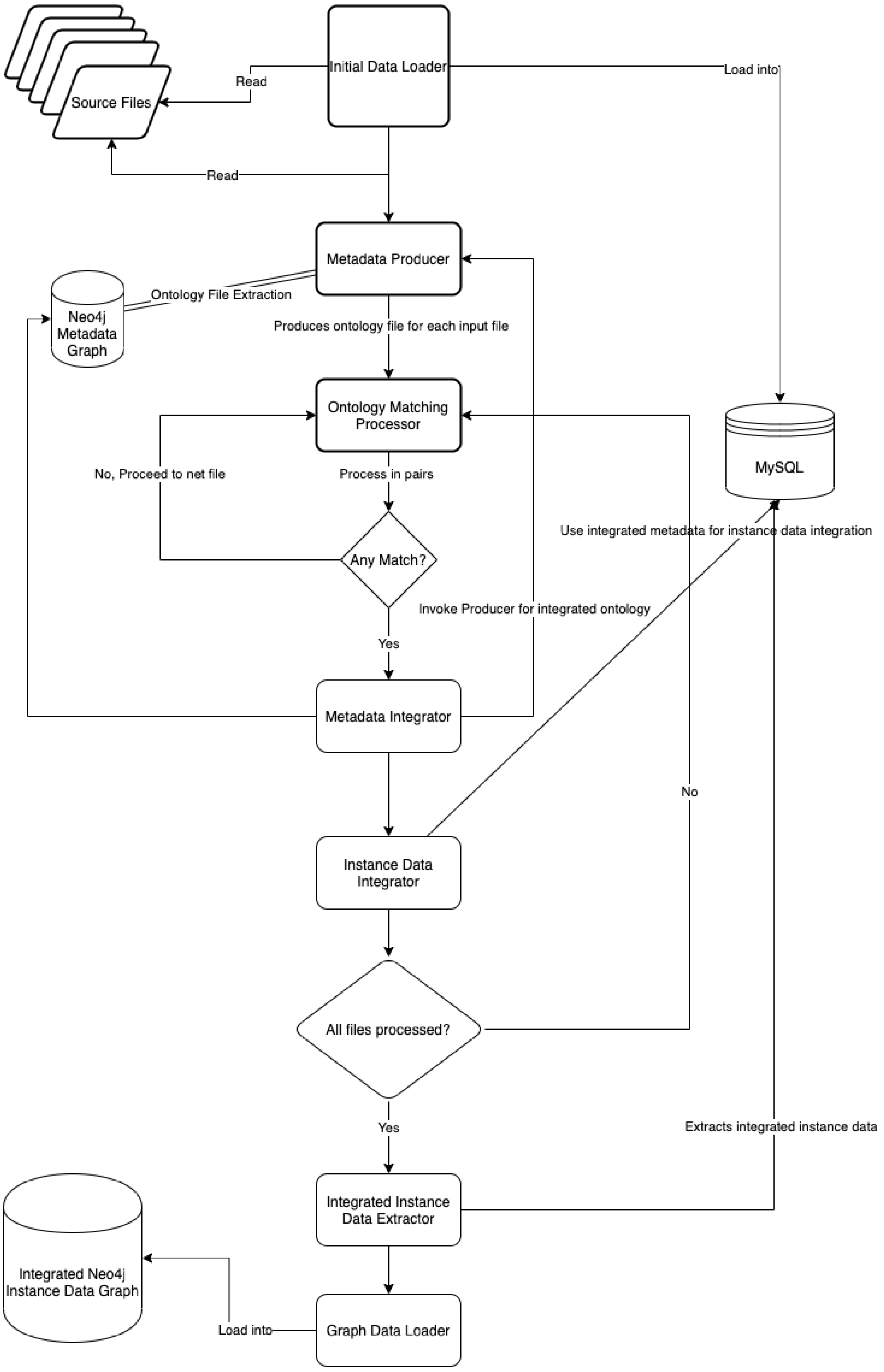

Source Files
Initial Data Loader
Read
Load into
Read
Metadata Producer
Neo4j Metadata Graph
Ontology File Extraction
Produces ontology file for each input file
Ontology Matching Processor
MySQL
No, Proceed to net file
Process in pairs
Use integrated metadata for instance data integration
Any Match?
Invoke Producer for integrated ontology
Yes
Metadata Integrator
No
Instance Data Integrator
All files processed?
Yes
Extracts integrated instance data
Integrated Instance Data Extractor
Integrated Neo4j Instance Data Graph
Load into
Graph Data Loader

Figure 3.1: System Flow Diagram

from input files into the MySQL database. **Metadata Producer** will take each one of the source files and load them into the Metadata Graph and extract ontology files to be used by Ontology Matching Processor. **Ontology Matching Processor** will process the ontology files generated by the Metadata Producer in pairs. One pair will be processed at a time: if the processor detects that two input files contain any potential connection points, the result will be passed to the Metadata Integrator, otherwise, it will proceed to the next pair in the queue. For example, for a 4-file integration process for data sources A, B, C, D, the ontology matching processor process the files in the following sequence: [A,B], [A-B, C], [A-B-C, D]. When this process has been completed, a score file will be generated for each of the pairs been processed, the score file will then be passed to the next component.

3.1.2 Metadata Integration

The *Metadata Integrator* handles the metadata integration for the Metadata Graph. Once the integration is complete, it will invoke the *Metadata Producer* to extract the extended integrated ontology file based on the integrated metadata graph. It can then be used as the new baseline for the next metadata integration step. Additionally, it also passes the metadata integration information to the Instance Data Integrator.

At this point, the system has gathered enough information to proceed with instance integration which will be the next step.

3.1.3 Data Integration

The role of the *Instance Data Integrator* is to leverage the information generated by the Metadata Integrator and perform JOIN operations using the MySQL database. As the vast majority of data is still captured using a relational or flat data model, a relational database is a good choice here. Traditionally, objects models were seen as best suited to data integration [20, 21, 52] and in this work, we selected a graph based model but a relational model was chosen as the data model for source data.

MySQL was chosen because of its wide usage in the research community. Using the original datasets, a new table will be created based on the integrated instance data. Depending on whether or not all files are fully processed, this new integrated table can be used as either the baseline for the next instance data integration step or the final integrated instance data to be fed to Integrated Instance Data Extractor. If there are input files still pending, the system will repeat the process of the Ontology Matching processor, where one of the ontology files will be the one generated by the Metadata Integrator from the previous step, and a second being the next ontology file in the list, yet to be processed.

The *Integrated Instance Data Extractor* extracts the final integrated instance data from the MySQL database in CSV file format. The reason the platform extracts the data as a CSV file and passes it to the Graph Data Loader is to increase the future re-usability of the Graph Data Loader Service, so other external systems can leverage the loader service to import any already integrated data generated by other tools. All they need to do is to put a transformation step prior to invoking the Graph Data Loading Service exposed by the platform. At this point, our system has managed to complete the instance data integration for the given source file pair and generated the integrated instance data in a tabular format.

3.1.4 Graph Construction

Graph Data Loading is the final step in the workflow. It is responsible for loading the integrated instance data in CSV file format generated by the previous component in the flow into the Neo4j Instance Data Graph. This step is not part of the data integration process and is discussed in detail in Chapter 5. The choice of Neo4j as the graph deployment application was made after a lot of analysis. It is clear that this particular graph implementation is quickly becoming one of the most powerful with strong evidence of a highly vibrant community of researchers and users [40].

3.2 Agri-Based Case Study

In this section, we present the case study that demonstrates the integration platform presented in Chapter 4.

Nowadays, agricultural (Agri) data is generated from a wide variety of sources, such as sensors, farm equipment, farmers, agricultural laboratories, and in many cases, the Web. However, these large data resources regularly originate from disparate sources and are often disconnected with different formats or types. Integrating these resources is generally the first process in the data engineering cycle and can be a manual time consuming process. These manual efforts can be attributed to the interventions required to handle the complex disconnected source data. Leveraging the graph based automated integration platform, simplifies and accelerates the integration process for the practitioner.

What was required for this part of our research was a robust evaluation of the integration process. ICBF [2] a partner in the Vistamilk [60] project was able to supply a set of analytical requirements and different Agri data sources to meet our integration challenge.

In the following case study, a list of Agri files in CSV format will be used as the source data and will be integrated using the platform shown in figure 3.2. In figure 3.2 we can see a list of input sources files on the left-hand side which go through the integration platform in the middle and eventually the constructed integrated graph can be used for data analytics purposes.

3.2.1 The Livestock Data Source

The LivestockNumbers file contains information for each of animal type that is on file, and sample data is shown in Table 3.1. The file consist of 10 data fields:

- CoverDate: Date field in the format of DD/MM/YYYY, dates ranging from year 2015 to year 2019

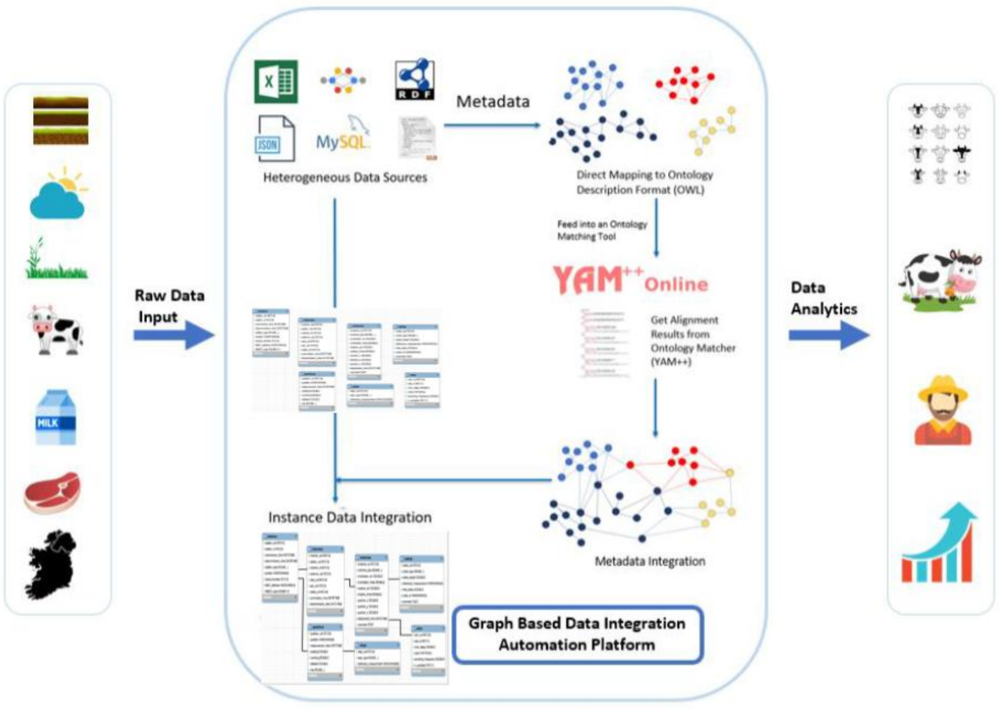

Raw Data Input
Metadata
Heterogeneous Data Sources
Direct Mapping to Ontology Description Format (OWL)
Feed into an Ontology Matching Tool
YAM++ Online
Get Alignment Results from Ontology Matcher (YAM++)
Data Analytics
Instance Data Integration
Metadata Integration
Graph Based Data Integration Automation Platform

Figure 3.2: Case Study with Agri sources

- PreviousCoverDate: Date field in the format of DD/MM/YYYY, dates ranging from year 2014 to year 2019 and contains blanks
- CoverID: Numerical identifier field
- LivestockTYpe: String type field, the content can be one of the 11 types, `>2 years`, `0 - 6 months`, `1 - 2 years`, `6 - 12 month` , `Autumn Milkers`, `Dairy Milkers Group 3`, `Dry Cows`, `Lactating Ewes`, `Spring Milkers`, `Stock Bull`, `Suckler Cows`
- NumberOfStock: Numeric field ranging from 0 to 605
- MealInTake: Numeric field ranging from 0 to 12
- SilageIntake: Numeric field ranging from 0 to 20
- GrassIntake: Numeric field ranging from 0 to 23
- AverageKgLwt: Numeric field ranging from 0 to 900 and contains blank
- TotalIntake: Numeric field ranging from 0 to 25

CoverDate	PreviousCoverDate	CoverID	LivestockTYpe	NumberOfStock	MealIntake	SilageIntake	GrassIntake	AverageKgLwt	TotalIntake
12/10/2019	01/10/2019	242781	1 - 2 years	0	0	0	6	300	6
01/10/2019	25/09/2019	240888	Spring Milkers	187	3	0	15	0	18
23/09/2019	15/09/2019	239267	6 - 12 months	0	0	0	2.4	120	2.4
23/08/2019	19/08/2019	232324	1 - 2 years	0	0	0	6	300	6
25/07/2019	22/07/2019	225195	1 - 2 years	0	0	0	6	300	6

Table 3.1: Sample Data for LivestockNumbers File

3.2.2 Animal Grazing Data Source

The ManagementDecisions file contains grazing management decisions, sample data shown in Table 3.2. This file contains a total number of 6 data fields:

- CoverID: Numerical identifier field
- CoverDate: Date field in the format of DD/MM/YYYY, dates ranging from year 2015 to year 2019

- RotationLength: Numeric field ranging from 0 to 130 and contains blanks
- ResidualHeight: Numeric field ranging from 3 to 100 and contains blanks
- DensityOfHerbage: Numeric field value can be either 240 or 250
- TargetPreGrazingYield: Numeric field ranging from 0 to 27523 and contains blanks

By cross referencing the data fields with first file `LivestockNumbers`, the possible integration points would be within `CoverID` and `CoverDate`.

CoverID	CoverDate	RotationLength	ResidualHeight	DensityOfHerbage	TargetPreGrazingYield
242781	12/10/2019	38	4	250	1578
240888	01/10/2019	38	4	250	1579
239267	23/09/2019	38	4	250	1368
232324	23/08/2019	25	4	250	1320
225195	25/07/2019	22	4	250	1168

Table 3.2: Sample Data for ManagementDecisions File

3.2.3 Paddock Data Source

The PaddockEstimations file contains estimated data for all the paddocks that are managed, sample data shown in Table 3.3. A total number of 35 data fields are in this file:

- ANONID: Numerical identifier field
- CoverID: Numerical identifier field
- CoverEstimations_PaddockID: Numerical identifier field
- CoverDate: Date field in the format of DD/MM/YYYY, dates ranging from year 2015 to year 2019
- PreviousCoverDate: Date field in the format of DD/MM/YYYY, dates ranging from year 2014 to year 2019 and contains blanks
- PaddockArea: Numeric field, ranging from 0 to 16

- DrainageCharacteristics_Description: String type filed, value can be `Well/ Moderately/ Poorly drained` or blank

- PaddockDistanceFromParlour: Numeric field ranging from 0.1 to 1000 and contains blanks

- PrincipalSoil: String type field, value can be one of 11 types or blank, e.g. `Acid Brown Earth 70`, `Gleys 75` etc.

- SoilNumber: Numeric field, value ranging from 1 to 41 and contains blanks

- AssociatedSoils: String type field, value can be one of 11 types or blank, e.g. `Brown Earth 10`, `Brown Earth 20 Gleys 5` etc.

- ParentMaterial: String type field, value can be one of 11 types or blank, e.g. `Mainly granite`, `Mostly ilurian shale` etc.

- PaddockAltitude: Numeric field ranging from 20 to 400 and contains blanks

- Gradients_Description: String type field

- Aspects_Description: String type field, includes `North`, `East`, `South`, `West`, `North East` and blanks

- HerbageEstKgDmHa: Numeric field, value ranging from -540 to 8000 and contains blanks

- Height: Numeric field, value ranging from 2.5 to 25 and contains blanks

- PreviousHerbageEstKgDmHa1: Numeric field, value ranging from -540 to 8000 and contains blanks

- PoachingEvent: Boolean field

- PoachingLevel: String type field, value can be `Lightly/Moderately Poached` or blank

- GrowthRate: Numeric field, value ranging from 0 to 1266 and contains blanks

- PaddockStatus: String type field can be such as `Grass`, `Being Grazed` and `Silage - Cut Later` etc.
- PaddockStatusPrevious: String type field can be such as `Grass`, `Being Grazed` and `Silage - Cut Later` etc.
- defoliatationoption: String type field, value includes but not limited to `Grazed`, `Cut for silage`
- GrazeDate: Date field in the format of DD/MM/YYYY, dates ranging from year 2014 to year 2019 and contains blanks
- CutDate: Date field in the format of DD/MM/YYYY, dates ranging from year 2015 to year 2019 and contains blanks
- ReseedDate: Date field in the format of DD/MM/YYYY, dates ranging from year 2015 to year 2016 and contains blanks
- SilageYieldKgDmHa: Numeric field ranging from 0 to 11111 and contains blanks
- ResidualAfterCuttingCM: Numeric field, value ranging from 0 to 200 and contains blanks
- ResidualAfterGrazingCM: Numeric field, value ranging from 1 to 25 and contains blanks
- Topping: Boolean field and contains blanks
- ResidualAfterToppingCM: Numeric field, value ranging from -2 to 20 and contains blanks
- PreviousGrazeDate: Date field in the format of DD/MM/YYYY, dates ranging from year 2011 to year 2019 and contains blanks
- ReseedMethodID: Numerical identified filed, can be 1, 2, 3 or blank
- PreGrazingYield: Numeric field, value ranging from 250 to 4000 and contains blanks

This file contains a large number of data fields, so it would take some time to analyze before making any conclusions about which field/fields should be used when implementing the integration with the previous two files. Once the analysis is done, both `CoverID` and `CoverDate` would transpire to be the integration points.

ANONID	CoverID	CoverEstimations_PaddockID	CoverDate	PreviousCoverDate	PaddockArea	...
1087943495	41554	25762	13/04/2017	06/04/2017	2.6	...
1087943495	41937	25756	19/04/2017	13/04/2017	3.8	...
2418859769	175978	102725	04/02/2019	07/01/2019	5.97	...
2487105239	180556	25842	26/02/2019	21/11/2018	1.37	...
1087943495	16011	25763	18/05/2015	11/05/2015	2.5	...

Table 3.3: Sample Data for PaddockEstimations File

3.2.4 Cow Information Data

The PbiCow file contains cow information recorded by the system, and sample data is shown in Table 3.4. The file consists of eight columns in total:

- ENC_HERD_ID: Numerical identifier field
- ENC_ANI_ID: Numerical identifier field
- DOB: Date field in the format of DD-mmm-YY, dates ranging from year 1998 to year 2018
- BREED: String field describing the breed of the cow
- Period_Begin_Date: Date field in the format of DD-mmm-YY, dates ranging from year 2000 to year 2019
- Period_End_Date: Date field in the format of DD-mmm-YY, dates ranging from year 2015 to year 2019 and contains blanks
- EBI: Numeric field ranging from -173 to 431 and contains NULL fields
- MILKSUBINDEX: Numeric field ranging from -103 to 192 and contains NULL fields

ENC_HERD_ID	ENC_ANI_ID	DOB	BREED	Period_Begin_Date	Period_End_Date	EBI	MILKSUBINDEX
359964155	2110584887	20-Mar-04	HO (68.75%), FR (18.75%), UN (12.5%)	20-Mar-04	27-Feb-17	78	-62
359968499	1503424535	27-Sep-03	HO (71.88%), FR (28.13%)	27-Sep-03	09-Feb-16	-75	-81
359969651	6252563027	07-Feb-13	HO (56.25%), FR (43.75%)	07-Feb-13	22-May-16	113	48
359970323	2682129227	22-Feb-06	HO (68.75%), FR (28.13%), UN (3.13%)	22-Feb-06	22-Dec-17	140	24
359977931	8149868573	13-Feb-16	HO (87.5%), FR (9.38%), MY (3.13%)	13-Feb-16		154	50

Table 3.4: Sample Data for PbiCow File

The fields presented in this file do not appear to have any integration points with the files above, but it does share a common field named `ENC_ANI_ID` with the `PbiLactation` file. We will see how the integration platform deals with this situation in detail in chapter 4.

3.2.5 Lactation Database

The PbiLactation file contains lactation information recorded by the system, with sample data shown in Table 3.5. The file consists 4 columns in total:

- ENC_ANI_ID: Numerical identifier field
- Calving_Date: Date field in the format of DD-mmm-YY, dates ranging from year 2013 to year 2019
- End_Lactation_Date: Date field in the format of DD-mmm-YY, dates ranging from year 2015 to year 2019 and contains blanks
- LACTATION: Numeric field ranging from 1 to 16

ENC_ANI_ID	Calving_Date	End_Lactation_Date	LACTATION
373492973	19-Feb-15	14-Dec-15	13
719997629	03-Feb-16	15-Aug-16	12
1331866073	26-May-17	15-Dec-17	14
2072385281	12-Mar-15	12-Jul-16	11
876545573	26-Feb-16	24-Jun-16	12

Table 3.5: Sample Data for PbiLactation File

From the four fields, `Calving_Date` would appear to be the most plausible field to use when integrating with files above.

At this point, it is not difficult to realize that each of the file above will provide information regarding one specific area, but in order to deliver an all around intelligent view of the overall business, the data needs to be intelligently connected together in a timely fashion.

3.3 Summary

The purpose of this chapter was to provide an outline approach to our research. This involved a high level overview of each of the steps, Ontology Matching, Metadata Integration, Data Integration, and Graph Construction, that is required to process and integrate multiple sources into a graph-based system. We have addressed all the components utilized by the system. The system is basically designed to be able to take various data files, extract the ontology files, do the data integration, and eventually completes graph construction. It is worth highlighting that the metadata integrator relies heavily on the help of ontology matching tool YAM++, but YAM++ itself can only be used as an ontology matching tool and it only supports ontology files as input. Without the other components in the system, YAM++ alone will not be able to deliver the one stop solution to simplify the data integration for users since in reality, no system will provide ontology files unless they are specifically coded for that purpose. Even with ontology files given the YAM++ result, a user will still need a robust system to finish the Instance Data Integration in order to complete the graph construction for enhanced data analytics.

We also described a real-world case study that will help to explain some of the more complex steps in our methodology. The next step is to examine each of the key components in our architecture in more detail.

In the next chapter, we will deliver a deep dive into the integration process using the data sources described in our case study.

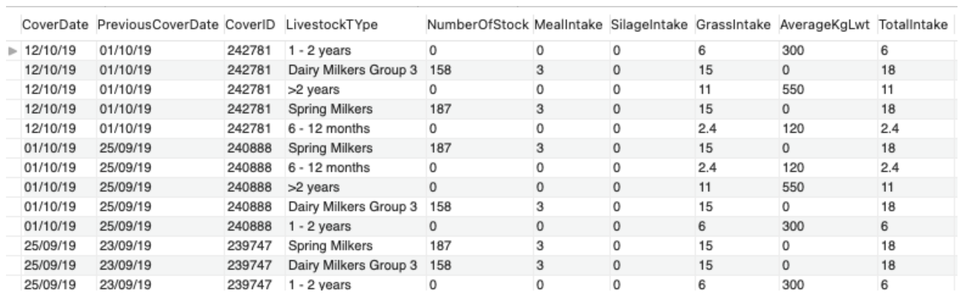
Chapter 4

Schema Integration

In this chapter, we present the integration process in detail. This process takes data in tabular form and automatically extracts ontology information. Those extracted ontology files will then be assessed to determine how well all the input files can be integrated, with help from YAM++ [42]. The score based assessment results will then be utilized to perform the automated data integration. In section 4.1, we describe the process for the initial loading of data while the metadata extraction process is described in section 4.2. Moving into section 4.3, we will discuss the ontology matching process followed by section 4.4 which describes in detail, the metadata integration process. Then, the final step is discussed in section 4.5. Those steps will be recursively invoked for all input files until all files are processed, in 4.6. In section 4.7, we present the evaluation framework and results and finally, in section 4.8, we summarise this chapter.

4.1 Initial Instance Data Loading

The first step requires the capture of all input files into a relational database and for this process, we use the MySQL database [36]. Before any integration steps take place, all the individual source files will first be loaded by the *Initial Data Loader* into the relational database, which will lead to the creation of a single dedicated



CoverDate	PreviousCoverDate	CoverID	LivestockTYpe	NumberOfStock	MealIntake	SilageIntake	GrassIntake	AverageKgLwt	TotalIntake
12/10/19	01/10/19	242781	1 - 2 years	0	0	0	6	300	6
12/10/19	01/10/19	242781	Dairy Milkers Group 3	158	3	0	15	0	18
12/10/19	01/10/19	242781	>2 years	0	0	0	11	550	11
12/10/19	01/10/19	242781	Spring Milkers	187	3	0	15	0	18
12/10/19	01/10/19	242781	6 - 12 months	0	0	0	2.4	120	2.4
01/10/19	25/09/19	240888	Spring Milkers	187	3	0	15	0	18
01/10/19	25/09/19	240888	6 - 12 months	0	0	0	2.4	120	2.4
01/10/19	25/09/19	240888	>2 years	0	0	0	11	550	11
01/10/19	25/09/19	240888	Dairy Milkers Group 3	158	3	0	15	0	18
01/10/19	25/09/19	240888	1 - 2 years	0	0	0	6	300	6
25/09/19	23/09/19	239747	Spring Milkers	187	3	0	15	0	18
25/09/19	23/09/19	239747	Dairy Milkers Group 3	158	3	0	15	0	18
25/09/19	23/09/19	239747	1 - 2 years	0	0	0	6	300	6

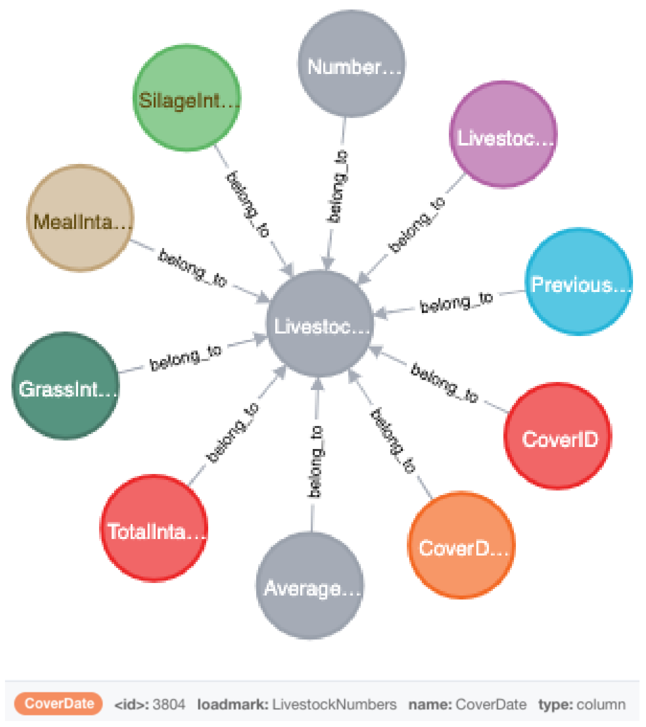
Figure 4.1: Livestock Numbers Data Source as a MySQL Table

table in the database. Using the first file processed by the loader as an example, the CSV file `LivestockNumbers` will be imported into the MySQL database as a new table, each column in the CSV file will be mapped directly to be one column in the MySQL table, shown in Figure 4.1. It is worth highlighting that the only work the user needs to complete at this stage is to input the selected input file names. The program itself will handle both the DB schema creation as well as the data loading. This process will be repeated for all the input files and it will only move to the next step of the process only when all the files have been loaded into the MySQL database. The data created in MySQL databases will be used later at *Instance Data Integration* step.

4.2 Metadata Extraction

When all source files have been imported into the relational database, the metadata producer will be invoked to proceed to the next processing step. During this step, each of the source files will be fed, one by one, into the metadata graph automatically. The graph nodes created by the platform in Neo4j are basically driven by the file name as well as the column fields. The file being processed will act as the central node, and the column fields will each map to a graph node having a `belong_to` relationship with the central node.

In order to visualize the metadata graph created by this process, Figure 4.2 shows the metadata graph for file `LivestockNumbers`. In the center of the graph, the node is named as `LivestockNumbers`. Each node will carry a number of properties which



CoverDate <id>: 3804 loadmark: LivestockNumbers name: CoverDate type: column

Figure 4.2: Livestock Numbers Metadata Graph & Properties for CoverDate Node

will be helpful to describe the node. For example, the property named `Type` will indicate whether the node was created from a table or column and the `loadmark` property captures the origin of the node creation. Figure 4.2 contains the full node information for the CoverDate node on the metadata graph created for file `LivestockNumbers`. The full list of the properties are presented in Table 4.1

Property Name	Description
id	Unique identifier of the node
loadmark	Indicates the source file where the node was created from
name	The name of the node, will be same as the column name or the file name
type	The type of the node, it can be either `column` or `table`

Table 4.1: Metadata Node Properties

Once the metadata graph has been created and fully loaded for a given source file, the platform will extract the ontology file. This process is done by leveraging the Neo4j database plugin *Neosemantics* [23], which enables the use of RDF in Neo4j. By using this plugin it enables the platform to export the graph schema in the form of an OWL Ontology. The platform will execute the commands against the Neo4j graph database and will generate the `owl:Class` definitions for each label found, the`owl:ObjectProperty` definitions for each relationship along with `rdfs:domain` and `rdfs:range` based on the labels of their start and end nodes. At this stage, all the input files will be processed individually and an ontology file will be generated for each individual file, this step is a crucial prerequisite for the next phase of processing. Note that a clear-out will be initiated before each file is processed. This is to ensure that the ontology file generated by this process only contains information relating to one specific source file.

Snippet 4.1 is the sample snippet of the ontology file generated for the `LivestockNumbers` file, and we will call it `LivestockNumbers Metadata`. Note the metadata will only contain the data fields information for a given source file and all the instance data will remain in the MySQL database and will not be processed at this stage. The full content of the ontology generated can be found in Appendix A.

Snippet 4.1: LivestockNumbers Metadata Snippet

```
<neo4j://graph.schema#SilageIntake> a owl:Class;
  rdfs:label "SilageIntake" .
<neo4j://graph.schema#CoverID> a owl:Class;
  rdfs:label "CoverID" .
<neo4j://graph.schema#AverageKgLwt> a owl:Class;
  rdfs:label "AverageKgLwt" .
```

4.3 Ontology Matching

The next step is the ontology matching processor which was developed with Yam++ [42]. Once all the files have been processed by the *Metadata Extractor*, there will be 6 ontology files generated for this case study, and each of them will be associated with the original 6 source files. The *Ontology Matching Processor* will determine how closely related each file is.

The processor evaluates ontology files in pairs, i.e. the `LivestockNumbers Metadata` and `ManagementDecisions Metadata` mentioned above are considered as an ontology file pair by the *Ontology Matching Processor*, and once evaluated, a score file is produced. During the evaluation step, the component utilizes the ontology matching tool Yam++ to deliver a score based solution. During the evaluation, the component makes an HTTP call to the API provided by Yam++, to evaluate the pair of ontology files being evaluated.

It is worth noting here that the metadata extraction capability we talked about in the previous section 4.2 is not only a valuable offering from our system but almost mandatory for real-world scenarios. This is because YAM++ is limited to only work with ontology files. In order to avoid any manual works prior to the ontology assessment, our system is essentially providing a unified end-to-end pipeline to make the overall integration process seamless for real-world data.

The sample snippet of the `ManagementDecisions Metadata` can be seen in Snippet 4.2 and the full content of the file can be found in Appendix B.

Snippet 4.2: ManagementDecisions Metadata Snippet

```
<neo4j://graph.schema#CoverID> a owl:Class;
  rdfs:label "CoverID" .
<neo4j://graph.schema#DensityOfHerbage> a owl:Class;
  rdfs:label "DensityOfHerbage" .
<neo4j://graph.schema#ResidualHeight> a owl:Class;
  rdfs:label "ResidualHeight" .
```

Essentially, all the labels of each ontology file from the pair will be utilized to perform the matching process, which is called element level ontology matching. When the matching process is completed, each of the label combinations between the two ontology files will be assigned with a matching score, with 1.0 being the highest score and 0.0 being the lowest score. When a label combination returns the highest score, this gives the metadata integrator enough confidence to decide that the two fields from two different ontology files are very likely to be matched and therefore should be integrated using that data field. An example is `CoverID` from the `Livestock Numbers Metadata` and `CoverID` from the `Management Decisions Metadata` receive a score of 1.0, indicating that the two files should be connected by this data field. Also, `CoverDate` from the `Livestock Numbers Metadata` and `ResidualHeight` from the `Management Decisions Metadata` receive a score of 0.0 indicating this combination should *not* be used by the platform to make the connection between two files.

The response of the Yam++ API will be in XML format. The *Ontology Matching Processor* will parse the result gathered from Yam++ API and generate a CSV file which will be used in the next step of the metadata integration decision making. The first score for the file coming back from Yam++ is shown in Table 4.2, with the entire file shown in Appendix K.

entity1	entity2	relation	score
CoverID	CoverID	=	1.0
CoverDate	CoverDate	=	1.0
PreviousCoverDate	CoverDate	=	1.0

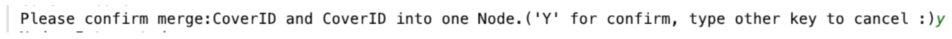
Table 4.2: First Step Metadata Score File

4.4 Metadata Integration

At this stage, the automation platform has successfully completed all the prerequisites steps for the integration process, and each of the ontology file pairs has been evaluated by the *Ontology Matching Processor* to determine whether two files can be integrated together and if so by which data field.

The *Metadata Integrator* will automatically select the combinations which have a score that is above the set minimum score threshold in order to be considered a potential connection point. In this research, the default minimum score has been set to be 0.8, which is configurable by the user. The score generated by the system will range from 0.0 to 1.0.

Once all those points have been identified by the integrator, it will proceed by



```
Please confirm merge:CoverID and CoverID into one Node.('Y' for confirm, type other key to cancel :)y
```

Figure 4.3: Calibration Feature Enabled

executing Neo4j cypher queries to integrate two individual metadata graphs on selected entities. When looking at the ontology file `LivestockNumbers Metadata` and `ManagementDecisions Metadata` generated, it can be found that a few identical fields are present in both files, thus a number has been assigned with the maximum score of 1.0. However, a field can only be used once to perform an integration. For example, if `CoverDate` from file one has been integrated with `CoverDate` from file two, then the integration between the `PreviousCoverDate` field from file `LivestockNumbers` and `CoverDate` field from file `ManagementDecisions` will not be considered eligible for future integration's. An optional calibration step is also supported by the platform, which when enabled, allows the user to confirm the integration of a particular field before merging is performed. The use case of this optional step will be demonstrated by the next metadata integration step. An example of the node integration calibration message is shown in Figure 4.3

As the result of the metadata integration process for given pair of input files along with the score file produced by the *Ontology Matching Processor*, a new integrated metadata graph will be created which contains data fields from both files being integrated and it will serve as a foundation for the next step of processing. The metadata integrator will perform the MERGE command on the fields defined by the platform to be the eligible integration point/points. Once that process is done, a new ontology file will also be extracted from this integrated metadata graph and it will be used for the next metadata integration step, this metadata graph will basically expand upon new successful metadata integration steps. The metadata graph created as a result of integration of first pair of files `LivestockNumbers` and `ManagementDecisions` is shown in Figure 4.4

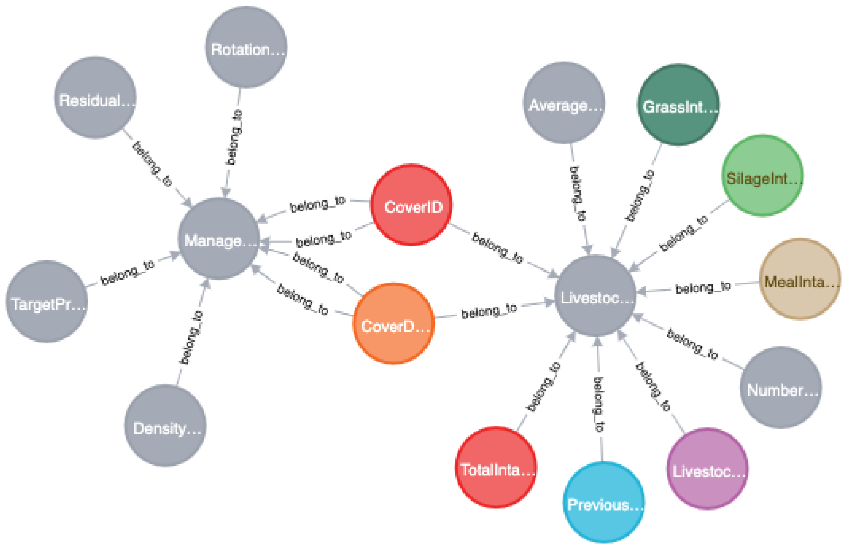


Figure 4.4: First Integrated Metadata Graph

4.5 Instance Data Integration

Unlike the *metadata integrator* which can perform integrations on multiple fields, the instance integrator requires only a single field to serve as the connection point to perform the instance data integration. The platform will store the information about which field combination has the highest matching score from the metadata integration, outlined in the first metadata integration step. As multiple combinations can achieve the maximum score of 1.0, the first one encountered by the platform will be selected as the connection point of two files for the instance data integration, i.e. data field `CoverID` from both files. The specific data field selected with the highest score will then be used by the platform to perform an outer join between the MySQL tables associated with the original two source files. A new table labeled as `Integrated Table One` will be created using the joined data. Similar to *metadata integrator*, the *instance data integrator* also provides a calibration feature that allows users to override the integration point.

4.6 Re-run of Steps For All Input Files

Once the platform has completed the integration of the first two input files, it must then process all remaining files. The above *Ontology Matching Process*, *Metadata Integration* and *Instance Data Integration* will, in sequence, be triggered repeatedly until all input source file pairs are fully processed.

Shown in Snippet 4.3 is the ontology file generated by the first step of metadata integration, with the full content of the file in Appendix C. This file will become half of the pair required to be processed next. The ontology matching processor will retrieve the next file in the queue which is `PaddockEstimations`. Its ontology file can be found in Appendix D with the corresponding sample shown in snippet 4.4. This pair of ontology files will generate a score shown in Table 4.3. By using this platform generated score file, the *metadata integrator* will be able to construct a new integrated metadata graph based on these two ontology files along with the score file passed down from the *ontology matching processor*.

Snippet 4.3: Extended Metadata with ManagementDecisions Snippet

```
<neo4j://graph.schema#TargetPreGrazingYield> a owl:Class;
  rdfs:label "TargetPreGrazingYield" .
<neo4j://graph.schema#GrassIntake> a owl:Class;
  rdfs:label "GrassIntake" .
<neo4j://graph.schema#CoverDate> a owl:Class;
  rdfs:label "CoverDate" .
```

Snippet 4.4: PaddockEstimation Metadata Snippet

```
<neo4j://graph.schema#PoachingLevel> a owl:Class;
  rdfs:label "PoachingLevel" .
<neo4j://graph.schema#CoverID> a owl:Class;
  rdfs:label "CoverID" .
<neo4j://graph.schema#GrazeDate> a owl:Class;
  rdfs:label "GrazeDate" .
```

From Table 4.3 we can see if the *metadata integrator* was configured to process on full auto mode. Some fields will be merged even though it would better if they remained as independent nodes. The optional `calibration` feature can be a great help in this case, as it allows the metadata integrator to skip the integration of the fields based on human judgment. For instance, even though the combination `NumberOfStock` and `SoilNumber` got a score of 1.0 by the ontology matching processor, it is relatively trivial for a human not to proceed with the integration of these two fields. This also applies to `TargetPreGrazeingYield` and `PreGrazeingYeild` as well as `ResidualHeight` and `Height`, where it is preferable for them to remain as independent nodes, as they all have different meanings and will provide different kinds of information. Once the calibration is done, the *metadata integrator* will proceed with the integration process, Figure 4.5.

entity1	entity2	relation	score
CoverID	CoverID	=	1.0
CoverDate	CoverDate	=	1.0
PreviousCoverDate	CoverDate	=	1.0
PreviousCoverDate	PreviousCoverDate	=	1.0
CoverDate	PreviousCoverDate	=	1.0
NumberOfStock	SoilNumber	=	1.0
TargetPreGrazingYield	PreGrazingYield	=	1.0
ResidualHeight	Height	=	1.0

Table 4.3: Second Step Metadata Score File

If a given input source file does not contain any data fields that can be used to do integration with existing integrated metadata, the platform will simply skip the file and proceed to the next file in the queue. For instance, the ontology file generated by file `PbiCow` shown in Appendix J does not contain any potential integration data points with the current integrated ontology file and will be skipped by the platform, however, it will be added to the tail of the file list, in order to be retried later. This is because once other files are added to the integrated graph, there is the possibility that this file can be integrated with the further extended graph.

The next file in line is the `PbiLactation`, the ontology file generated for it can be found in Appendix F with a sample snippet shown below in Snippet 4.5

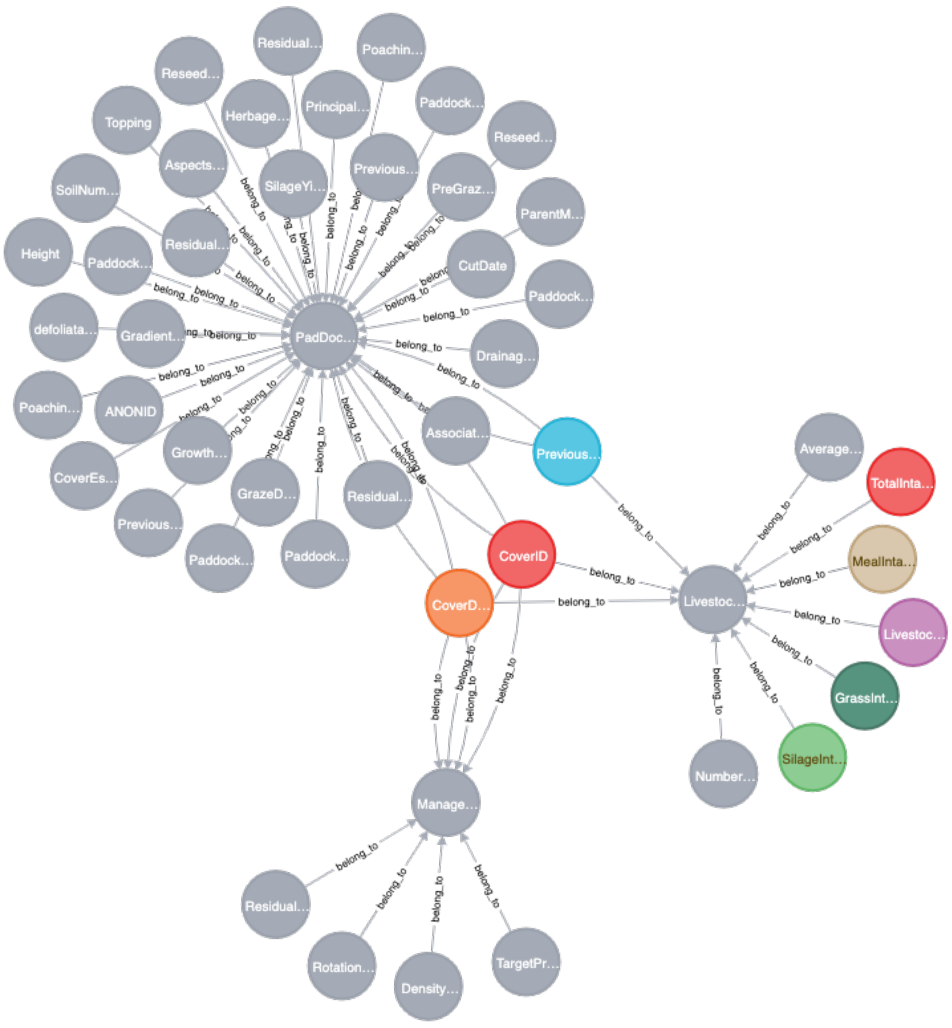


Figure 4.5: Metadata Graph Extended with PaddockEstimations

Snippet 4.5: PbiLactation Metadata Snippet

```
<neo4j://graph.schema#Calving_Date> a owl:Class;
  rdfs:label "Calving_Date" .
<neo4j://graph.schema#LACTATION> a owl:Class;
  rdfs:label "LACTATION" .
<neo4j://graph.schema#ENC_ANI_ID> a owl:Class;
  rdfs:label "ENC_ANI_ID" .
```

Adding the latest extended ontology file shown in Appendix E with sample snippet shown in Snippet 4.6 to form the pair, the *Ontology Matching Processor* will produce the score file shown in Table 4.4, once processed by metadata integrator, it will generate the further extended metadata graph shown in Figure 4.6

Snippet 4.6: Extended Metadata with PaddockEstimations Snippet

```
<neo4j://graph.schema#PreviousHerbageEstKgDmHa1> a
   owl:Class;
  rdfs:label "PreviousHerbageEstKgDmHa1" .
<neo4j://graph.schema#CoverDate> a owl:Class;
  rdfs:label "CoverDate" .
<neo4j://graph.schema#NumberOfStock> a owl:Class;
  rdfs:label "NumberOfStock" .
```

entity1	entity2	relation	score
CoverDate	Calving_Date	=	1.0

Table 4.4: Third Step Metadata Score File

The latest integrated ontology file extended by adding `PbiLactation` file can be found in appendix G, and the sample snippet shown in Snippet 4.7, it will be processed by the *ontology matching processor* along with ontology file generated by file `IndividualCase` shown in appendix H with the sample snippet shown in Snippet 4.8. This step will produce the score file shown in Table 4.5.

Snippet 4.7: Extended Metadata with PbiLactation Snippet

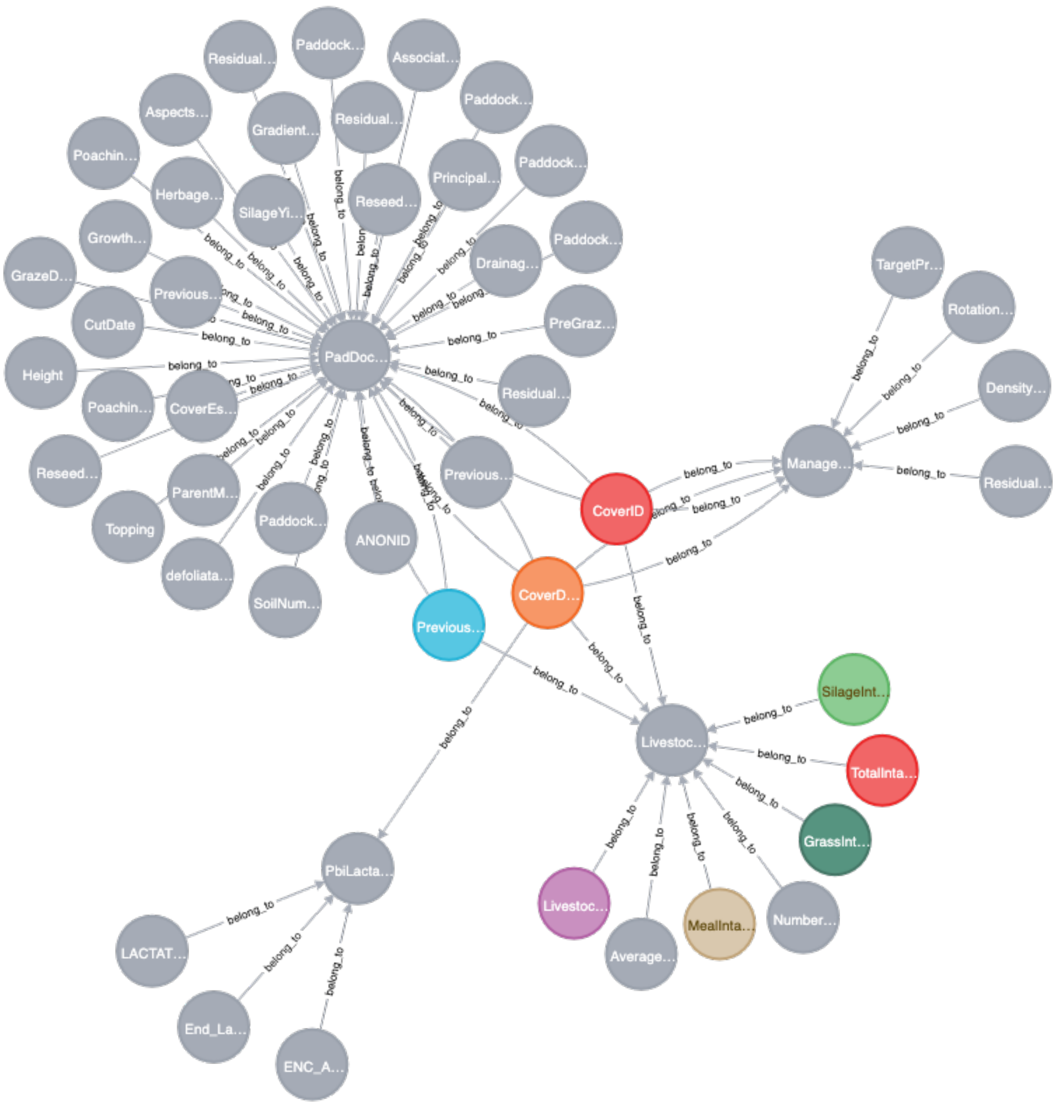


Figure 4.6: Metadata Graph Extended with PbiLactation

```
<neo4j://graph.schema#HerbageEstKgDmHa> a owl:Class;
  rdfs:label "HerbageEstKgDmHa" .
<neo4j://graph.schema#ENC_ANI_ID> a owl:Class;
  rdfs:label "ENC_ANI_ID" .
<neo4j://graph.schema#ResidualAfterGrazingCM> a owl:Class
   ;
  rdfs:label "ResidualAfterGrazingCM"
```

Snippet 4.8: IndividualCase Metadata Snippet

```
<neo4j://graph.schema#provincial_case_id> a owl:Class;
  rdfs:label "provincial_case_id" .
<neo4j://graph.schema#method_note> a owl:Class;
  rdfs:label "method_note" .
<neo4j://graph.schema#date_report> a owl:Class;
  rdfs:label "date_report" .
```

entity1	entity2	relation	score
CoverDate	date_report	=	1.0

Table 4.5: Fourth Step Metadata Score File

By performing integration on `CoverDate` from the existing integrated metadata graph and `date_report` from the `IndividualCase` file, the metadata integrator will be able to extend the graph with the new file as shown in Figure 4.7

As mentioned earlier, because the platform was not able to initially integrate the file `PbiCow`, the file was appended to the tail of the file list to be re-processed later. If we take a look at the latest integrated ontology file shown in appendix I, focusing on the relevant component in Snippet 4.9, we find a field named `ENC_ANI_ID` which was added as part of the integration of file `PbiLactation`. Now the platform should be able to make the integration based on that field if we look at the similar snippet from the ontology file generated using `PbiCow` shown in Snippet 4.10. If we look in Table 4.6, the score file generated by evaluating the pair of ontology files confirms

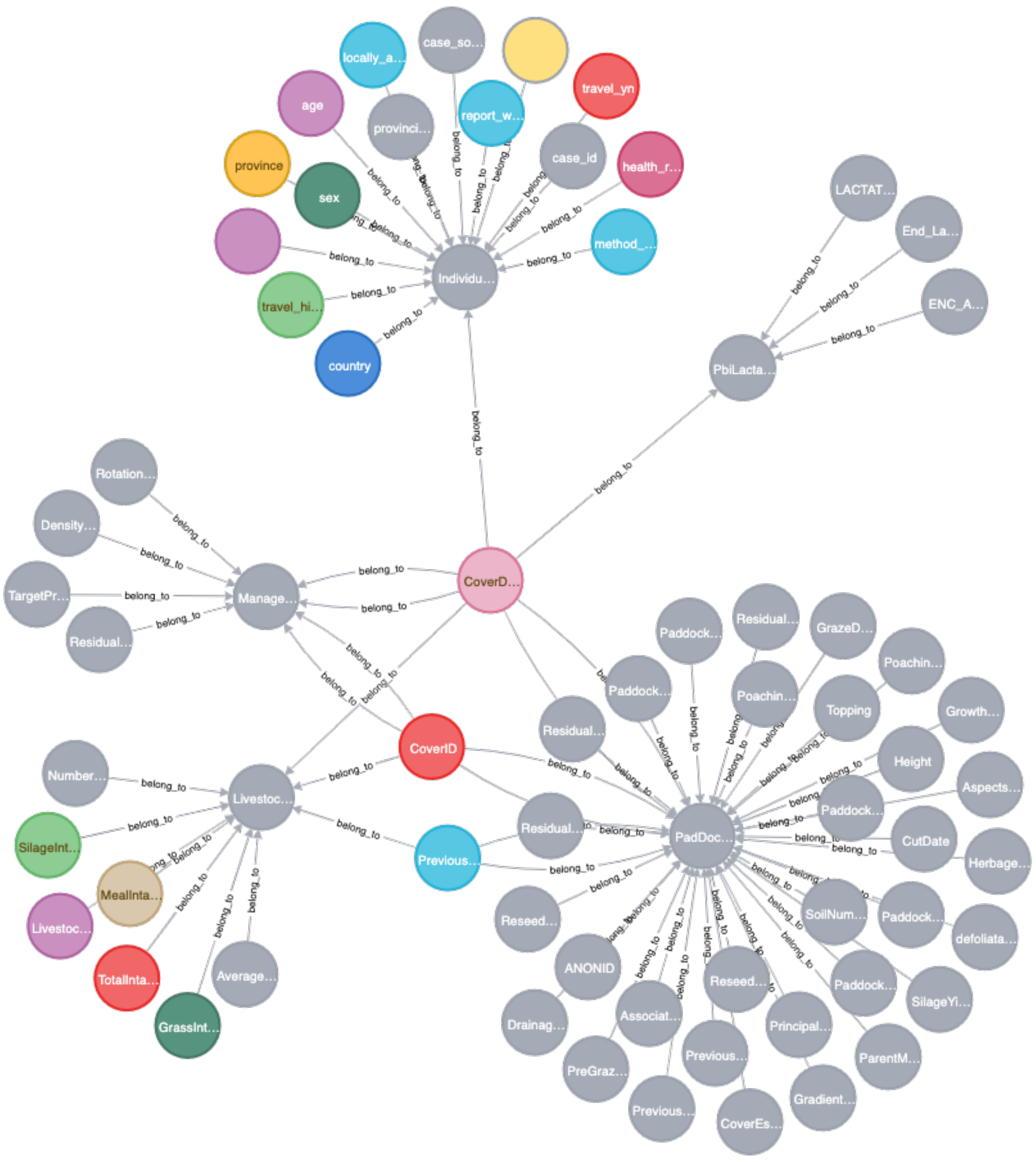


Figure 4.7: Metadata Graph Extended with IndividualCase

the same.

Snippet 4.9: Extended Metadata with IndividualCase Snippet

```
<neo4j://graph.schema#SilageIntake> a owl:Class;
  rdfs:label "SilageIntake" .
<neo4j://graph.schema#date_report> a owl:Class;
  rdfs:label "date_report" .
<neo4j://graph.schema#ENC_ANI_ID> a owl:Class;
  rdfs:label "ENC_ANI_ID" .
```

Snippet 4.10: PbiCow Metadata Snippet

```
<neo4j://graph.schema#PERIODBEGIN_DATE> a owl:Class;
  rdfs:label "PERIODBEGIN_DATE" .
<neo4j://graph.schema#ENC_ANI_ID> a owl:Class;
  rdfs:label "ENC_ANI_ID" .
<neo4j://graph.schema#ENC_HERD_ID> a owl:Class;
  rdfs:label "ENC_HERD_ID" .
```

entity1	entity2	relation	score
ENC_ANI_ID	ENC_ANI_ID	=	1.0

Table 4.6: Last Step Metadata Score File

By integrating the `PbiCow` file into the existing integrated metadata graph using field `ENC_ANI_ID`, the platform now finishes the metadata integration process. Figure 4.8 shows the final integrated metadata graph with all the input files having been fully processed. The workflow for a full demonstration of the process of the repeated components is shown in Figure 4.9 .

On completion of the integration process, a final integrated metadata graph is produced in order for consumers to check/validate the metadata integration. It illustrates a high level overview of how the data fields are being integrated, by eliminating any obstruction of the instance data, and clearly indicates the connections/relationships between each of the data fields from all the processed files.

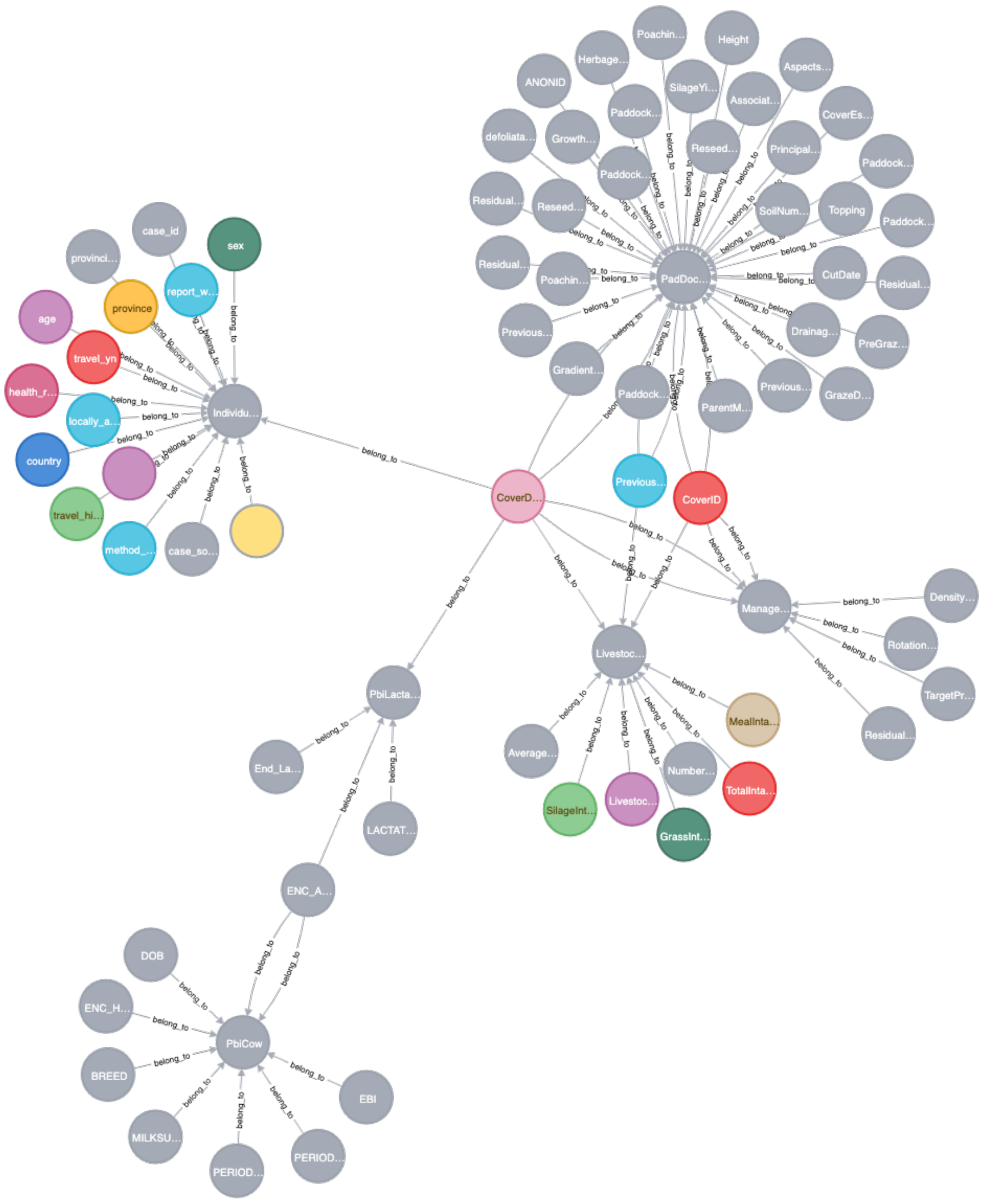


Figure 4.8: Final Integrated Metadata Graph With PbiCow Integrated

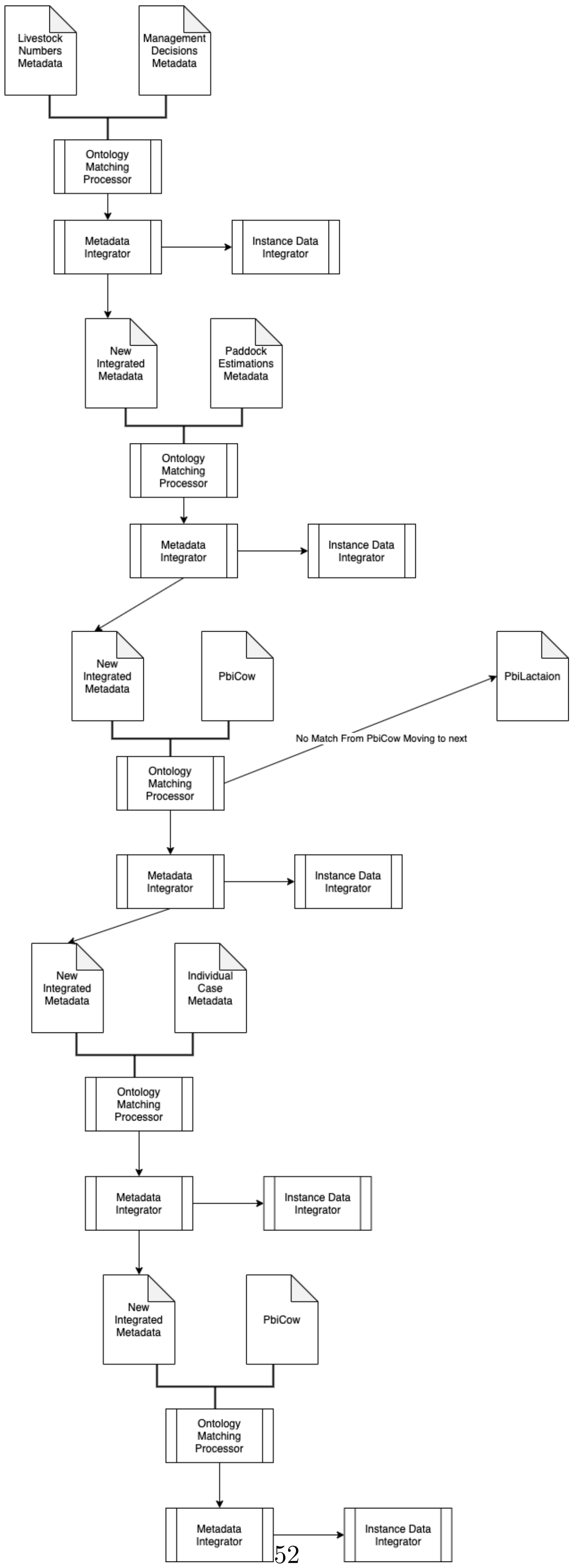


Figure 4.9: File Integration Flow In Action

At this stage, the platform has completed the full integration process with minimum human intervention and the final integrated instance data has been extracted into a CSV file and is ready to be imported into a graph.

4.7 Evaluation

It is important to measure how well our integration platform performed, as all parts of this process with the exception of the usage of YAM++ in the *Ontology Matching* phase, were developed for this dissertation.

4.7.1 Evaluation Framework

The final integrated SQL table contains 29,371 rows across 66 columns. For an evaluation, we executed SQL queries against the original SQL tables created during the initial instance data loading, with the same (although modified to reflect the new structure) queries executed against the final integrated SQL table. We would expect identical results from both sets of queries.

Five queries were selected to verify the data integrity of the integrated result against original datasets, although we believe they sufficiently demonstrate a general notion of correctness of the integrated data, it does not constitute absolute proof of correctness.

Example 4.7.1. Locate Cover with Livestock and Grazing Filter

```
SELECT CoverID FROM  ICBF.LivestockNumbers WHERE CoverID
   IN (SELECT CoverId FROM  ICBF.ManagementDecisions
   WHERE TargetPreGrazingYield  > 2000)
AND NumberOfStock > 180 AND LivestockType = 'Spring␣
   Milkers';
SELECT CoverID FROM integration_6 WHERE
   TargetPreGrazingYield  > 2000 AND NumberOfStock > 180
   AND LivestockType = 'Spring␣Milkers';
```

Example 4.7.2. Rotational Length Distribution based on Previous Cover Date Window

```
SELECT DISTINCT RotationLength FROM  ICBF.
   ManagementDecisions WHERE CoverID IN (SELECT DISTINCT
   CoverId FROM  ICBF.LivestockNumbers  WHERE STR_TO_DATE
   (PreviousCoverDate, '%d/%m/%Y') BETWEEN STR_TO_DATE('
   01/09/2019', '%d/%m/%Y') AND STR_TO_DATE('01/12/2019',
    '%d/%m/%Y'));
SELECT DISTINCT RotationLength FROM integration_6 WHERE
   STR_TO_DATE(PreviousCoverDate, '%d/%m/%Y') BETWEEN
   STR_TO_DATE('01/09/2019', '%d/%m/%Y') AND STR_TO_DATE(
   '01/12/2019',  '%d/%m/%Y')
```

Example 4.7.3. Paddock information that does not have specific livestock and with large paddock area

```
SELECT CoverEstimations_PaddockID, PaddockArea,
   PaddockStatus FROM ICBF.PaddockEstimations WHERE
   COVERID IN (SELECT DISTINCT CoverId FROM  ICBF.
   LivestockNumbers WHERE LiveStockTYpe = '>2␣years'  AND
    NumberOfStock=0) AND PaddockArea > 3;
SELECT CoverEstimations_PaddockID, PaddockArea,
   PaddockStatus FROM integration_6 WHERE LiveStockTYpe =
    '>2␣years'  AND NumberOfStock=0  AND PaddockArea > 3
   ;
```

Example 4.7.4. Filtered Cow information based on calving date

```
SELECT BREED, DOB, `PERIOD BEGIN DATE`, `PERIOD END DATE
   `, EBI, `MILK SUB INDEX` FROM ICBF.PbiCow WHERE
   ENC_ANI_ID IN (SELECT ENC_ANI_ID FROM ICBF.
   pbilactation WHERE  STR_TO_DATE(Calving_Date, '%d-%M-%
   Y') BETWEEN STR_TO_DATE('01-Jan-16', '%d-%M-%Y') AND
   STR_TO_DATE('01-Jan-18', '%d-%M-%Y'));
SELECT BREED, DOB, `PERIOD BEGIN DATE`, `PERIOD END DATE
   `, EBI, `MILK SUB INDEX` FROM integration_6 WHERE
   STR_TO_DATE(Calving_Date, '%d-%M-%Y') BETWEEN
   STR_TO_DATE('01-Jan-16', '%d-%M-%Y') AND STR_TO_DATE('
   01-Jan-18', '%d-%M-%Y') AND BREED IS NOT NULL;
```

Example 4.7.5. Lactation information for cows that are older than 20 years

```
SELECT ENC_ANI_ID, Calving_Date, End_Lactation_Date,
   LACTATION FROM ICBF.pbilactation WHERE ENC_ANI_ID IN (
   SELECT ENC_ANI_ID FROM ICBF.PbiCow WHERE STR_TO_DATE(
   DOB, '%d-%M-%Y') < STR_TO_DATE('25-May-01', '%d-%M-%Y'
   ));
SELECT ENC_ANI_ID, Calving_Date, End_Lactation_Date,
   LACTATION FROM integration_6 WHERE STR_TO_DATE(DOB, '%
   d-%M-%Y') < STR_TO_DATE('25-May-01', '%d-%M-%Y');
```

4.7.2 Evaluation Results

Example 4.7.1 to Example 4.7.5 show the queries selected that were ran against the source and final integrated tables. In each example, the query was executed against the source table, and below is the query that was executed against the final integrated table. The results for each are shown in Table 4.7. This confirmed the accuracy of our integration framework methodology.

4.8 Summary

The purpose of this chapter was to perform a detailed description of our integration platform and to describe the validation process. Our case study used files in

Query ref	Results from source	Results from integrated data	comments
Query 4.7.1	1 Column, 48 Records	1 Column, 48 Records	Identical results
Query 4.7.2	1 Column, 19 Records	1 Column, 19 Records	Identical results
Query 4.7.3	3 Columns, 36 Rows	3 Columns, 36 Rows	Identical results
Query 4.7.4	6 Columns, 22 Records	6 Columns, 22 Records	Identical results
Query 4.7.5	4 Columns, 4 Records	4 Columns, 4 Records	Identical results

Table 4.7: Validation Results

the Agri sector which were introduced in section 3.2. The evaluation showed that once files were taken from their sources and integrated, the query-based validation demonstrated that the integration took place without corrupting the underlying data.

In the next chapter, the final transformation/graph-loading step converts the annotated dataset into a graph schema. Graph database systems offer tools that import relational-like data into their graph model. In these tools, the approach is to convert each tuple from the source relation into a single node, and each value in the tuple into a property within such a node. Nodes are then linked to each other via edges, where the edges describe the Foreign to Primary key linkage between tuples.

Chapter 5

Data Model Transformation

In the previous chapter, we described how we integrated data using a graph-based platform. However, that data is still captured in a relational model and there remains a further step in transforming that relational model data to graph data. In this chapter an introduction is given in section 5.1. In section 5.2, we describe the metadata analysis. In section 5.3, we discuss attribute classification. The Graph construction process is presented in section 5.4. In section 5.5 we summarise this chapter.

5.1 Introduction

Data models provide different data views and query languages provide different tools for the analysis and querying of the same data. In a simple dataset where there are few relationships within the data and little structure, tabular data provides a simple means of data storage. In linked data [29], where data is linked as a web, a flat model can neither represent, nor help to visualize the data. However, many of the datasets provided for machine learning take the form of flat datasets, stored in relational databases. Increasingly, there are very powerful datasets stored in simple CSV files which contain none of the semantics captured in a relational database. These files are still very useful in standard machine learning algorithms.

Graph databases have become common in recent years as they allow queries and analyses that provide new insights into the data. Graph databases applications such as Neo4j [39], languages such as Cypher [15] and algorithms such as centrality [41] and community detection [50] provide different forms of analyses on the same data, offering new insights.

In a real-world scenario, business data and data from most other domains are saved in tabular format. However, in certain circumstances, some linked data has to be modeled using a graph structure in order to do graph analysis. In parallel to relational tables, graphs are another model to store data. We cannot tell which data model is the best, and the performances are related to the data itself: relational databases have been shown to have better performances when the data has little internal linkage e.g. only contains observational records, this applies for both processing speed and memory size; For linked data e.g. social media relations, a relational database can not model and visualize the internal linkages, where data is linked as a web, therefore graphs are preferable when modeling such data.

The output from the integration part of our research described in chapter 4 was an integrated tabular dataset, captured using a relational model and the RDF metadata in Neo4j format. The next step is to convert the data into graph data and for this process, we drop the RDF metadata. Our reasons can be articulated as follows: we want a generic graph and not an RDF graph; there is still a significant process involved in converting the RDF metadata graph into an actual data graph, and also we would like a process that delivers the best possible graph.

In this chapter, we present our method for converting a tabular dataset into an optimized graph database, *Cube2Graph*. In tabular datasets, each entry represents a (business) record or observation. Often, these datasets come denormalized and devoid of semantics, where data belonging to different conceptual entities are mixed together, and no constraints (e.g. PKs or FKs) are provided. We focus our attention on multi-dimensional datasets, where data can be split into two categories: dimensions and measurements. Briefly, dimension data are composed of non-overlapping values that provide a categorical description of the record; fact data provide an associated quantitative measure. We assume that the input dataset is sufficiently large to distinguish dimensions and facts where the dataset has a multi-dimensional data model.

5.2 Meta-Analysis

In this step, We analyze the tabular data to extract salient information that will drive the conversion to the graph model. A dataset in tabular format is basically a relation without schema constraints. Formally:

Definition 5.1. (Tabular Dataset). A tabular dataset is a relation `r` over a schema `R(X)`, where `R` is the name of the relation and `X` is the set of attributes in `R`.

The relation `r` is analyzed to surface the following characteristics: keys, foreign-keys, and statistical analysis of the values from each attribute $A_i \in X$.

The statistical analysis aims at determining whether the domain associated with the attribute is finite or not. As described later in this section, attributes from finite domains can be exploited in the conversion to obtain an optimized graph.

Each relation is profiled with a set of metrics listed below that will drive the graph generation:

- $M^{tot}_{a_j}$: The cardinality of the relation across all attributes (we are assuming the absence of NULL values);
- $M^{dist}_{a_j}$: The number of distinct values for attribute $a_j \in X_i$;
- $M^{mean}_{a_j}$: The mean number of duplicates for the distinct values in attribute for $a_j \in X_i$;
- $M^{rdis}_{a_j}$: Ratio of distinct values $M^{dist}_{a_j}$ to total values for the column $M^{tot}_{a_j}$;
- $M^{rdup}_{a_j}$: Ratio of mean duplicates $M^{mean}_{a_j}$, to all values $M^{tot}_{a_j}$;

In addition, we define an additional metric M_i^{limit} that is the maximum value among all $M_{a_j}^{dist}$ across all relations. While M_i^{limit} is not used in the classification part of our method, a very high number has implications for graph construction. This number defines the max number of `tolerable` distinct values for the attribute. An attribute with high value cardinality will generate a high number of nodes that in extreme numbers cause performance problems.

During the meta-analysis step, the column with the *highest* number of distinct values is recorded. This will become an independent node using the rules explained later in this section. As graph databases suffer from performance problems, early detection of problematic datasets is sensible. Our experiments showed that nodes with a high number of distinct values cause memory problems and thus, $T_{limit} \geq 150$, indicates that this limit is set to 150 values for each dimension. But we want to point out that this limit can be adjusted according to the use case, e.g. datasets, hardware setup, etc.

Initial thresholds are set in this step, with more datasets getting involved and a manual-label-check adjustment process (post-step) will then adjust the thresholds by gathering all manually labeled datasets as a data pool to train the thresholds.

5.3 Attribute Classification

In this step, every attribute is automatically classified in order to distinguish dimensional values from facts on the basis of their annotations. As the original dataset contained no semantics, we can only approximate this classification. Since there is no guarantee that the detection of dimension versus fact will be semantically accurate, we refer to categories dimensions and facts as *Independent Attributes* (`IA`) and *Dependent Attributes* (`DA`), respectively.

Our classification process is based on three parameters:

- `KB`: Domain Knowledge Base

- MV: Max number of Values
- MR: Max Ratio on distinct values

The KB is a set of attributes names that domain experts deem as IA attributes, thus overriding the automatic classification: this is necessary to simplify the management of data for the domain experts themselves, once data is in graph format.

The MV specifies the max number of absolute distinct values that are *tolerable* for an attribute before deeming it as DA.

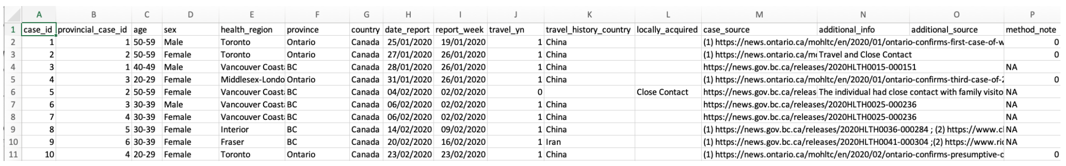
The MR parameter provides an indication of the repetitions of distinct values versus the total number of values.

Both MV and MR are used to control the level of optimization in the graph conversion.

Algorithm 5.1 Attribute Classification.

```
Classify $X$
    inputs : X, KB, MV, MR
    output: IA, DA
    foreach $A_i \in X \in r$ do
        if $A_i \in KB$
         $\vee\, COUNT(DISTINCT\ A_i) \leq MV$
         $\vee\, COUNT(DISTINCT\ A_i)/COUNT(A_i)$
         $\leq MR$
         then
            $IA \leftarrow A_i \cup IA$
        else
            $DA \leftarrow A_i \cup DA$
        end
    end
```

Intuitively, if the number of distinct values in an attribute is relatively low compared to the cardinality of the relation R, then the attribute is classified as IA; else, it is a DA. The classification algorithm is provided in Alg. 5.1. In addition to attributes associations and defined threshold, there is an additional factor that is included in the process: domain knowledge. There are some attributes that domain knowledge experts prefer to keep separate because it simplifies the management of associated values once the data is in a graph format. This knowledge base is a set of attribute



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	case_id	provincial_case_id	age	sex	health_region	province	country	date_report	report_week	travel_yn	travel_history_country	locally_acquired	case_source	additional_info	additional_source	method_note
2	1	1	50-59	Male	Toronto	Ontario	Canada	25/01/2020	19/01/2020	1	China		(1) https://news.ontario.ca/mohltc/en/2020/01/ontario-confirms-first-case-of-w			0
3	2	2	50-59	Female	Toronto	Ontario	Canada	27/01/2020	26/01/2020	1	China		(1) https://news.ontario.ca/m	Travel and Close Contact		0
4	3	1	40-49	Male	Vancouver Coast	BC	Canada	28/01/2020	26/01/2020	1	China		https://news.gov.bc.ca/releases/2020HLTH0015-000151			NA
5	4	3	20-29	Female	Middlesex-Londo	Ontario	Canada	31/01/2020	26/01/2020	1	China		(1) https://news.ontario.ca/mohltc/en/2020/01/ontario-confirms-third-case-of-2			0
6	5	2	50-59	Female	Vancouver Coast	BC	Canada	04/02/2020	02/02/2020	0		Close Contact	https://news.gov.bc.ca/releas	The individual had close contact with family visito		NA
7	6	3	30-39	Male	Vancouver Coast	BC	Canada	06/02/2020	02/02/2020	1	China		https://news.gov.bc.ca/releases/2020HLTH0025-000236			NA
8	7	4	30-39	Female	Vancouver Coast	BC	Canada	06/02/2020	02/02/2020	1	China		https://news.gov.bc.ca/releases/2020HLTH0025-000236			NA
9	8	5	30-39	Female	Interior	BC	Canada	14/02/2020	09/02/2020	1	China		(1) https://news.gov.bc.ca/releases/2020HLTH0036-000284 ; (2) https://www.c			NA
10	9	6	30-39	Female	Fraser	BC	Canada	20/02/2020	16/02/2020	1	Iran		(1) https://news.gov.bc.ca/releases/2020HLTH0041-000304 ;(2) https://www.ri			NA
11	10	4	20-29	Female	Toronto	Ontario	Canada	23/02/2020	23/02/2020	1	China		(1) https://news.ontario.ca/mohltc/en/2020/02/ontario-confirms-presumptive-c			0

Figure 5.1: Sample Data

names and values which we refer to as `KB`. All attributes in the KB are classified as IA.

Note, in the program, we use label *RECORD* to indicate that the type is IA and *NODE* to indicate that the type is DA. In example 5.3.1 we can see `case_id` is categorised as IA because of the high number of distinct values at 100% compared to the number of rows. In example 5.3.2 we can see `age` is categorised as DA because of the low number of distinct values, approximately 0.1% compared to the total number of rows in the source file.

Example 5.3.1. Classified as IA Example

```
checking column: age
age  should be NODES with good set length/ratio
age /len(set): 17 /ratio: 0.0009902720335527465 /N/A ratio: 0.0 /
```

Example 5.3.2. Classified as DA Example

```
checking column: case_id
case_id  should be RECORDS due to set[col] length limit
case_id /len(set): 17167 /ratio: 1.0 /N/A ratio: 0.0 /
```

Table 5.1 shows the result of classification step for the sample source file mentioned in figure 5.1. In this table, on the left-hand side we have the `Field Name` column which is the list of columns from the original CSV file, and on the right-hand side is the `Attribute Type` column which contains the classification result from our system for each column.

5.4 Graph Construction

With the tabular dataset annotated, the final step is the conversion to a graph model. For the graph database, we adopt, without loss of generality, a simplistic

Table 5.1: Sample Attribute Classification

Field Name	Attribute Type
case_id	DA
provincial_case_id	DA
age	IA
sex	IA
health_region	IA
province	IA
country	IA
date_report	IA
report_week	IA
travel_yn	IA
travel_history_country	IA
locally_acquired	IA
case_source	DA
additional_info	DA
additional_source	DA
method_note	IA

model where nodes can only have one value and edges only one label. More formally, we define a graph database as follows:

Definition 5.2. (Graph Database). A graph database is a graph $\mathsf{g} = (\mathrm{N}, \mathrm{E})$ where every node $n \in N$ is associated with a value, and every edge $e \in E$ is associated with a label.

Note that this definition of our graph database model is different from that used in modern graph database systems, where a graph database is, indeed, a multi-graph. In a multi-graph, in addition to linking nodes via edges, both a node and an edge can be associated with a set of (key-value) properties (values).

Graph database systems offer tools that import relational-like data into their graph model. In these tools, the approach is to convert each tuple from the source relation into a single node, and each value in the tuple into a property within such a node. Nodes are then linked to each other via edges, where the edges describe the *FKs* to *PKs* linkage between tuples.

In our method as in other approaches, the primary and foreign keys constraints are treated the same way: they become connection edges between nodes. In contrast to other approaches, our method exploits node annotations in order to reduce the number of nodes required in the graph.

Alg. 5.2 formally describes the graph construction method for a single relation. The output of the algorithm is a star-graph, as defined in Def. 5.3.

Definition 5.3. (Star-Graph Database). A star-graph database is a multi-graph $g = (N^{IND}, N^{DEP}, E)$ where an `independent node` $n \in N^{IND}$ is a node associated with a single value, a `dependent node record` $r \in N^{DEP}$ is associated with a set of pairs $\langle key, value \rangle$, and every edge $e \in E$ is a triple $\langle r, n, l \rangle$ describing a directional connection with label l between two nodes within or across in N^{IND} and N^{DEP}.

Briefly, values associated with DA attributes are loaded with a 1:1 mapping into the graph: each value becomes a so-called dependent node (N^{DEP}) in the graph; distinct values associated with IA attributes become a so-called independent node (N^{IND}). Dependent and independent nodes are then linked via graph relationships, edges, to describe the records from the original relation. Compression is achieved as only a single node class is created for each IA value which is considerably less than the total number of instances.

More formally, as per Alg. 5.2 and Def. 5.3, the star-graph is formed by a N^{DEP} node at the center of the star, where N^{IND} are satellite nodes. For the sake of clarity and brevity, but without loss of generality, we assume that a relation can only have one DA attribute. An N^{IND} node is connected to the N^{DEP} nodes via two edges:

- An edge with a label *RECORD* to capture that values belonging to the same source tuple;
- An edge with label describing the semantics of attribute connections: the attribute name from the original relation.

The graph construction process involves three steps:

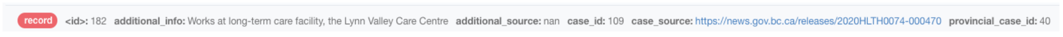
Algorithm 5.2 Graph Construction.

```
Input  : $TabularData$, $IAList$, $DAList$
Output: Constructed Graph
foreach row r in Tabular Data File do
    foreach DA in DA list do
        Load r[DA] into joint dependent node Record $record_r$;
    end
    foreach IA in IA list do
        if r[IA] does not exist in graph then
            Load r[IA] as independent node $node_n$;
        Create bi-direction edges between $node_n$ and $record_r$
    end
end
```

1. For properties which are categorized as *IA*, they will be loaded as `independent nodes` by the program;

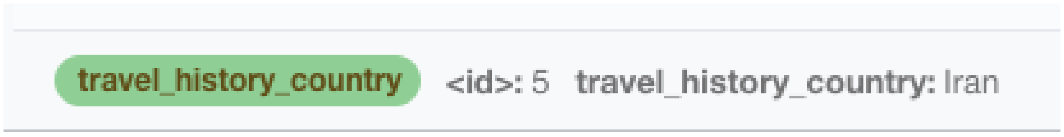
2. For properties which are categorized as *DA*, they will be loaded as joint nodes named `Record` by the program;

3. Upon completion of node creation, the program will proceed with creating `edges` between each node if there were relationships between them.



record <id>: 182 additional_info: Works at long-term care facility, the Lynn Valley Care Centre additional_source: nan case_id: 109 case_source: https://news.gov.bc.ca/releases/2020HLTH0074-000470 provincial_case_id: 40

Figure 5.2: Sample IA Node Info



travel_history_country <id>: 5 travel_history_country: Iran

Figure 5.3: Sample DA Node Info

Next, we will present some visualisation samples from the constructed graph. In Figure 5.2, we can see the node information for an N^{IND} type node, in this node, `<id>` is a unique id auto generated by Neo4j and other properties `additional_info`, `case_id`, `case_source` as well as `provicial_case_id` were categorised as IA types in the previous step described in section 5.3. In Figure 5.3 we can see the node information for an N^{DEP} type node containing one property apart from Neo4j generated `<id>` property.

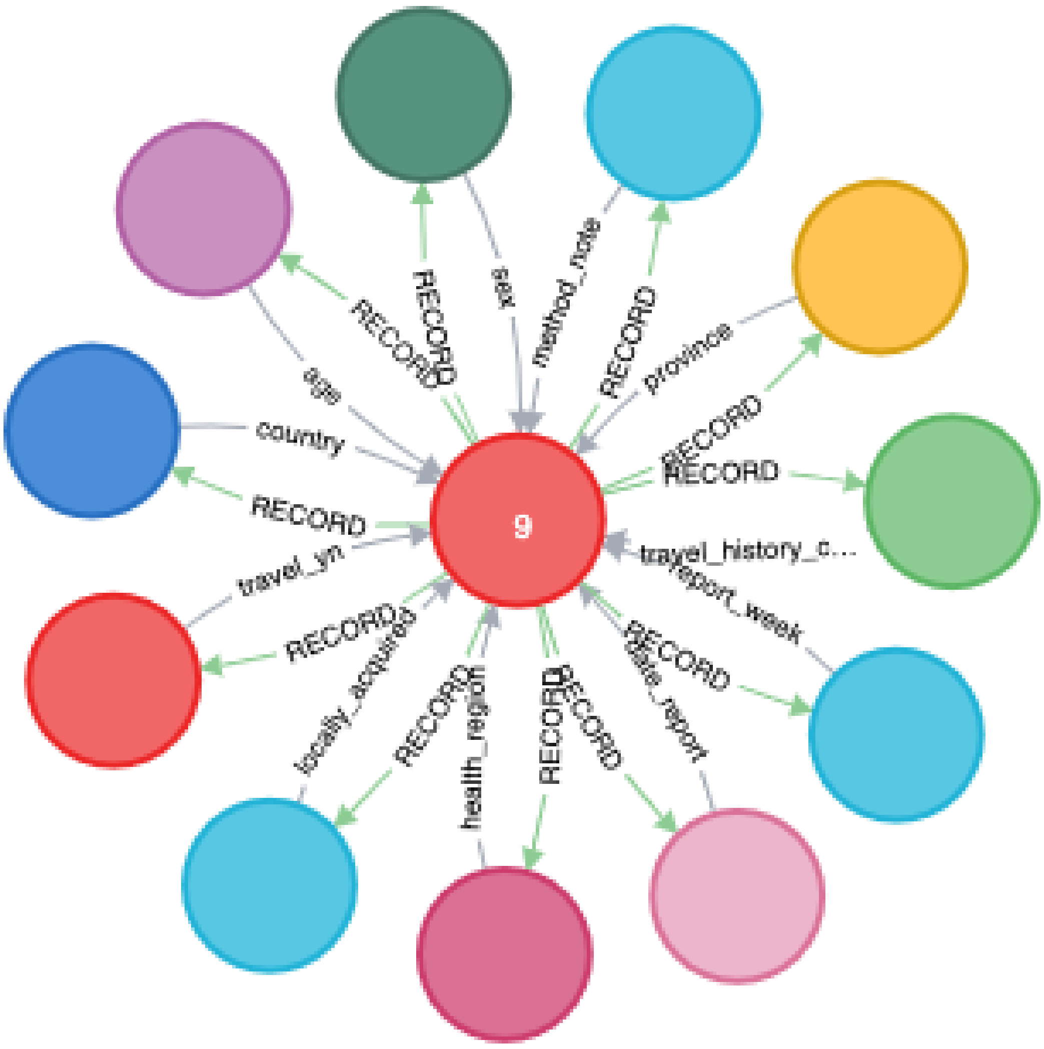


Figure 5.4: Sample DA Connected with IAs

In Figure 5.4 we can see the sample N^{DEP} node with `case_id = 9` at the center of the star connected with multiple N^{IND} node where each of the connections in the star has two edges, one labeled as *RECORD* and another labeled as the attribute name from the original source.

Figure 5.5 is an example that displays multiple DA node types connecting to the same IA node type, in this case the blue colored node in the center of the graph is *report_week* which is an *independent node* and all the red colored nodes are *dependent nodes* representing different COVID cases.

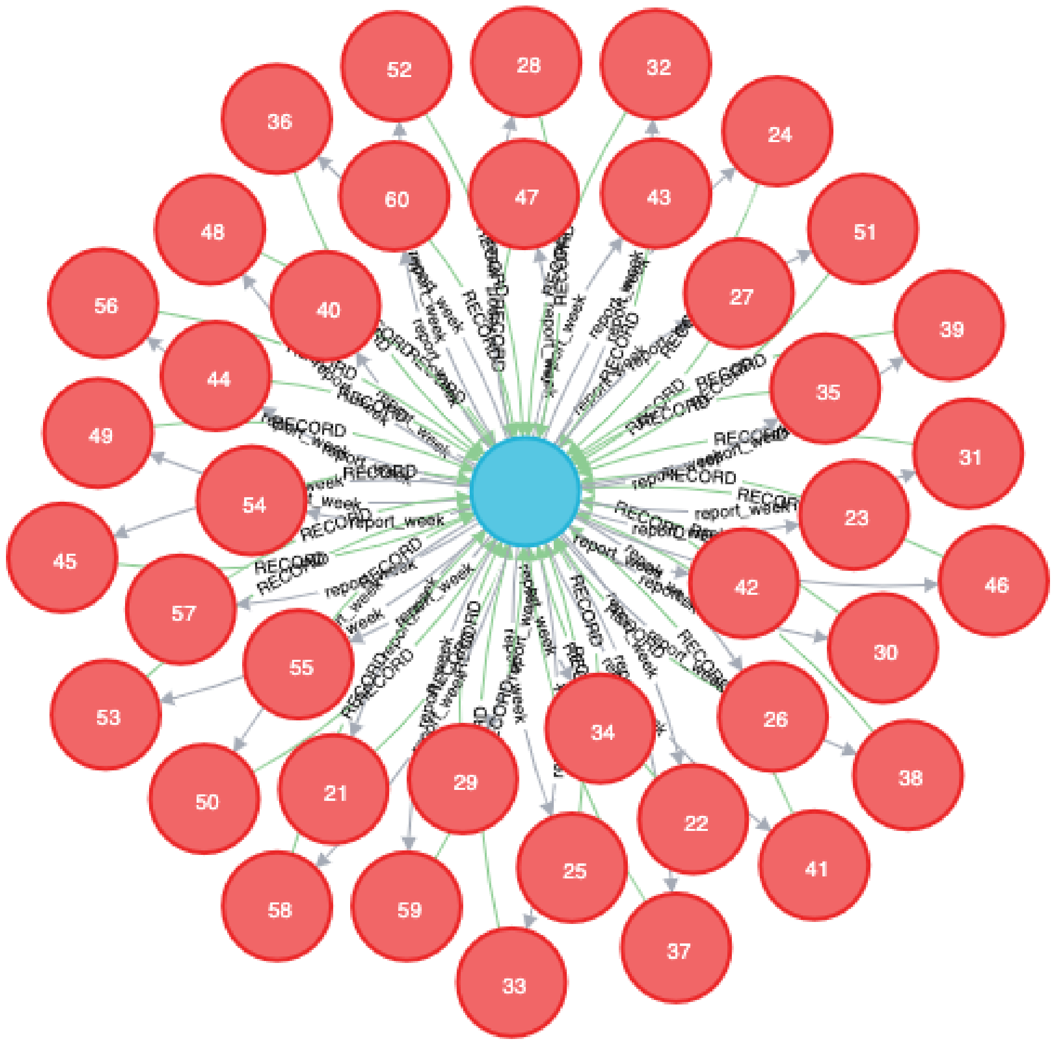


Figure 5.5: Multiple DAs connecting to same IA

In figure 5.6, we can also see an example where different DA type of nodes in the graph are connected with each other through different shared common IA type of nodes. In the graph, the *dependent nodes* labeled with `case_id` and colored in red are connected to each other via different *independent nodes* which are in different colors other than red.

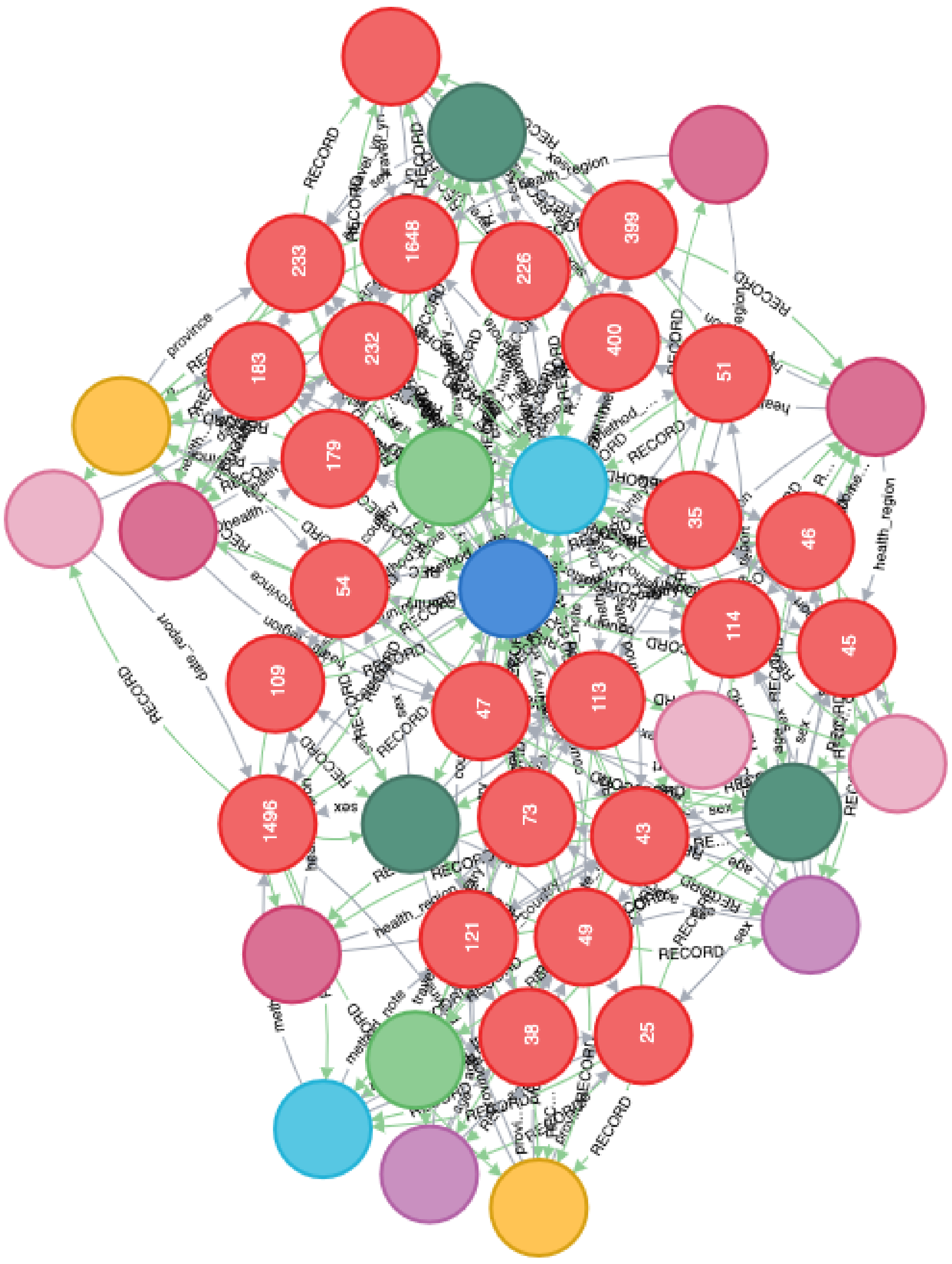


Figure 5.6: Nodes connecting with each other in graph

5.5 Summary

In this chapter, we have walked through the graph data transformation in detail. Our research provides a method to convert tabular data into a graph structured data which can be used to do graph analysis to discover relations that relational databases cannot do. Tabular data is a widely used data type, it is convenient to get other types (Json [45], XML [9], etc.) of data into a tabular type data. Between tabular data and graph data, a mapping process is required to add namespaces/connections/edges. The process to convert tabular data into graph data is a process to find internal relations. Through a classification process, the system is able to gather instructions on how to design the perspective graph schema. After generating edges based on property structures and relations, data can be loaded directly into the graph database as a graph.

With the development of Graph Technology, researchers have developed novel applications where graphs have been at the core of the solution. Graphs have given researchers a different perspective and aspect in order to deal with conventional problems and have helped in many discoveries. We believe the positive result will be achieved by combing our graph transformation methodology with our integration system.

In the next chapter, we present a detailed evaluation of the end result of the process, a graph constructed using our system originated from data in tabular form.

Chapter 6

Evaluation

In this chapter, we evaluate the graph constructed using our transformation methodology with source data in the tabular form discussed in chapter 5. We also describe the hardware and software setup for the experiment. In section 6.1, we describe the experimental setup. The results of the experiment will then be presented in section 6.2. In section 6.3, we will share our insights and findings from the results, before summarizing the chapter in section 6.4.

6.1 Experimental Setup

In this section, we will go over the setup for our experiment for both hardware and software. Our core system is developed using python [59] version 3.7. During our evaluation, the python program and MySQL database instance versioned 8.0.19 are hosted on a local machine with 8GB of RAM, 4 CPU cores as well as a solid state drive, and we leverage Microsoft AZURE [7] cloud machine with 14GB of RAM, 2 high performance CPU cores and a solid state drive for hosting a cloud based Neo4j database versioned 4.0.0 to deliver optimized performance for processing graph data loading.

Our research is largely involved in the agricultural domain. All of the datasets used in our research originated from agricultural, geographical or weather datasets. For an evaluation on the constructed graph, we used a data source outside our normal domain in order to test the robustness of our approach. As it is topical at the moment, we used one of the covid-19 datasets from Kaggle [10], consisting of 16 columns and 17,167 rows detailing individual infections in Canada. The goal is to demonstrate the topography of the graph and run comparison queries for the original source dataset and the newly constructed graph. This might appear to be out of context as we are merging two fully separate domains. However, it provides a good challenge for our approach and if successful, demonstrates that a number of core dimensions exist across all datasets that enable some level of integration eg. time and location.

able 6.1 presents the results of the **Meta-Analysis** step described in section 5.2 as columns M^{dist}, M^{mean}, $M_{a_j}^{rdis}$ and $M_{a_j}^{rdup}$, recall that M^{dist} is the total number of distinct values, $M_{a_j}^{rdis}$ is the ratio of distinct values $M_{a_j}^{dist}$ to total values for the column $M_{a_j}^{tot}$, $M_{a_j}^{rdup}$ is ratio of duplicates mean $M_{a_j}^{mean}$, to all values $M_{a_j}^{tot}$, while *Class* presents the result of the **Attribute Classification** step described in section 5.3 specifically by algorithm 5.1.

6.2 Results

Table 6.1: Covid-19 Dataset Meta Analysis

Attribute	M^{dist}	M^{mean}	$M_{a_j}^{rdis}$	$M_{a_j}^{rdup}$	Class
case_id	17167	1.0000	1.0000	0.0001	DA
provincial_case_id	8580	2.0008	0.4998	0.0001	DA
age	17	1009.8235	0.0010	0.0588	IA
sex	3	5722.3333	0.0002	0.3333	IA
health_region	83	206.8313	0.0048	0.0120	IA
province	13	1320.5385	0.0008	0.0769	IA
country	1	17167.0000	0.0001	1.0000	IA
date_report	50	343.3400	0.0029	0.02	IA
report_week	12	1430.5833	0.0007	0.0833	IA
travel_yn	3	5722.3333	0.0002	0.3333	IA
travel_history_country	75	7.4400	0.0044	0.0133	IA
locally_acquired	4	92.5000	0.0002	0.2500	IA
case_source	838	20.4857	0.0488	0.0012	DA
additional_info	174	7.9713	0.0101	0.0057	DA
additional_source	163	8.0123	0.0095	0.0061	DA
method_note	4	4291.7500	0.0002	0.2500	IA

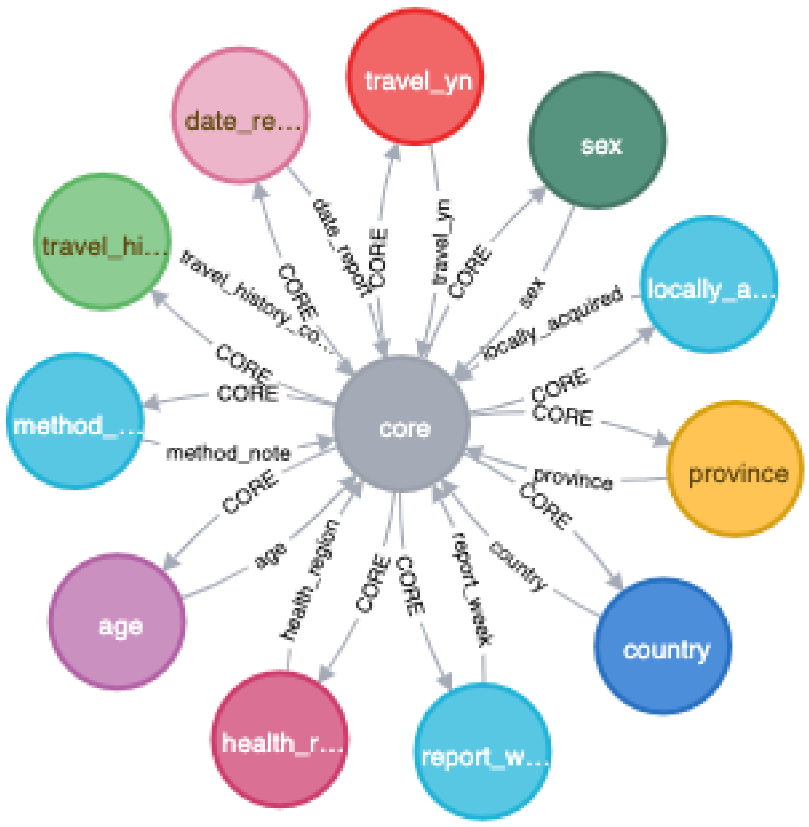


Figure 6.1: Covid-19 Graph Database Schema

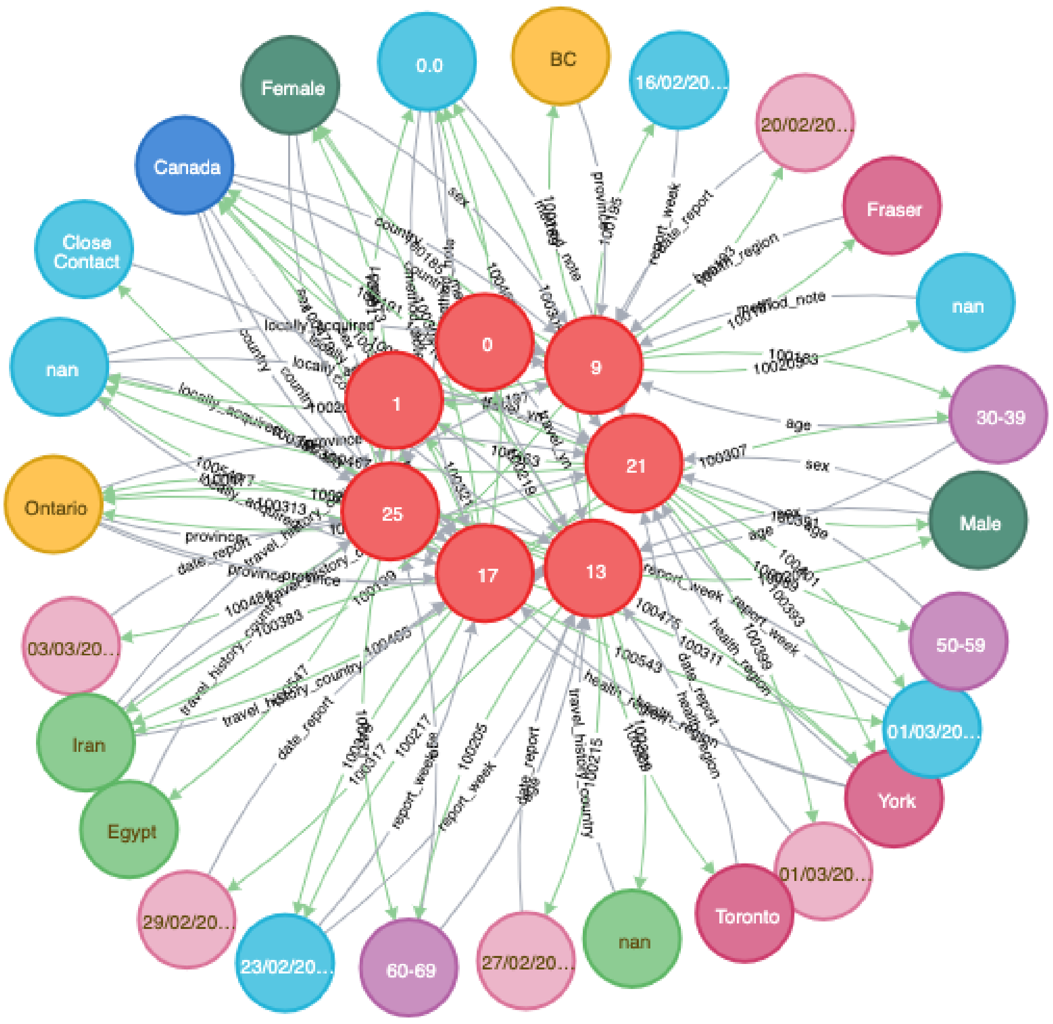


Figure 6.2: Covid-19 Graph

Figure 6.1 illustrates the schema design: a *star* where the dependent node (fact properties) form the core with relationships to all independent nodes (dimensional properties). Each fact can be related to other facts through one or more independent nodes, for example, multiple covid cases can be linked to each other when they were reported in the same week or they are based in the same province.

The constructed graph has 17,435 nodes, from 12 types (labels) and 377,674 relationships from 12 types. A partial view of the complete graph is shown in figure 6.2.

The graph is restricted to 5 random dependents nodes expanded with their connected independent nodes. The simple retrieval query is shown in example 6.2.1. Note that the reason for the relatively small number of nodes selected is for better visualization purposes. A large crowd of dependent and independent nodes being displayed in the same graph will simply make the graph unreadable.

Look at the center of the graph, the red colored nodes labeled with *ID* are 5 COVID records. They are connected with all the independent *fact* nodes associated with them.

Example 6.2.1. Retrieve 5 random dependent nodes

```
MATCH (r:record) RETURN r LIMIT 5
```

6.2.1 Storage

The first evaluation was to assess the storage requirements, as this is often a critical point especially when physical storage is limited. The queries ran on both Neo4j and MySQL to retrieve storage information are shown in Examples 6.2.2 6.2.3.

Example 6.2.2. Storage Info Retrieval Query for Neo4j

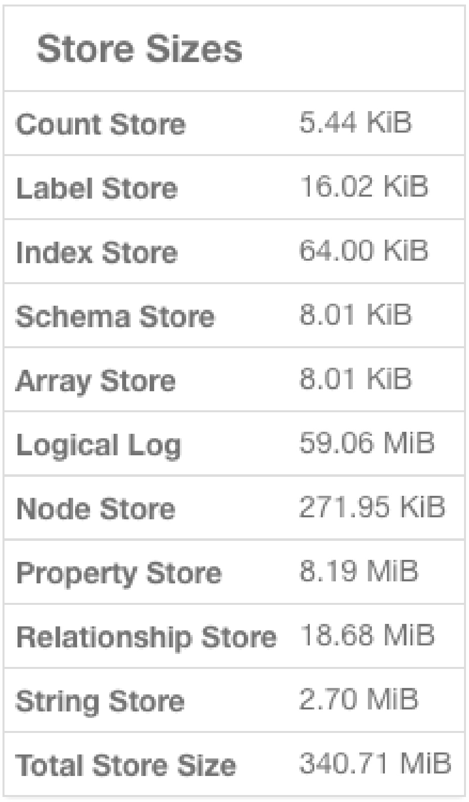
```
:sysinfo
```

Example 6.2.3. Storage Info Retrieval Query for MySQL

```
SELECT
  TABLE_NAME AS `Table Name`,
  ROUND(((DATA_LENGTH + INDEX_LENGTH) / 1024 / 1024),2)
     AS `Total Storage Size (MB)`
FROM
  information_schema.TABLES
WHERE
  TABLE_NAME = "IndividualCase"
ORDER BY
```

```
   (DATA_LENGTH + INDEX_LENGTH)
DESC;
```

The total storage required by the Neo4j graph was around 340 MB and includes the storage for both the data and the metadata, Figure 6.3. Alternatively, the MySQL database takes approximately 0.25MB in storage for the original source data shown in figure 6.4.



Store Sizes	
Count Store	5.44 KiB
Label Store	16.02 KiB
Index Store	64.00 KiB
Schema Store	8.01 KiB
Array Store	8.01 KiB
Logical Log	59.06 MiB
Node Store	271.95 KiB
Property Store	8.19 MiB
Relationship Store	18.68 MiB
String Store	2.70 MiB
Total Store Size	340.71 MiB

Figure 6.3: Neo4j Storage Size

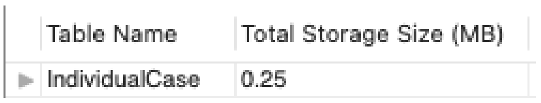


Table Name	Total Storage Size (MB)
IndividualCase	0.25

Figure 6.4: MySQL Table Storage Size

6.2.2 Data Integrity

We want to ensure we have a true representation of the data with the constructed graph. In order to perform the validation, we have selected a number of queries to be executed against both the Neo4j and MySQL databases. We expect the results from both databases to be identical. The queries are shown in Example 6.2.4, Example 6.2.5 and Example 6.2.6. In those examples, the query at the top is executed against the MySQL database, the query at the bottom was executed against Neo4j database.

Below are the expected results for each query example:

- Example 6.2.4: Integer number indicating the number of COVID cases reported in the week commencing *08/03/2020*;
- Example 6.2.5: Integer number of COVID cases that do not have the *travel_histroy_country*;
- Example 6.2.6: COVID records that are aged between *30-39* and located in *BC* area;

The graph representation in Neo4j for query result of example 6.2.4 is shown in Figure 6.5. Figure 6.6 shows the graph representation of the result for the query executed in Example 6.2.5. The graph representation for query result of example 6.2.6 is shown in Figure 6.7.

Example 6.2.4. Case Number reported for a given week

```
SELECT COUNT(*)
FROM IndividualCase
WHERE report_week = "08/03/2020"

MATCH (r:record)-[]->(w:report_week)
WHERE w.report_week = '08/03/2020'
RETURN COUNT(r)
```

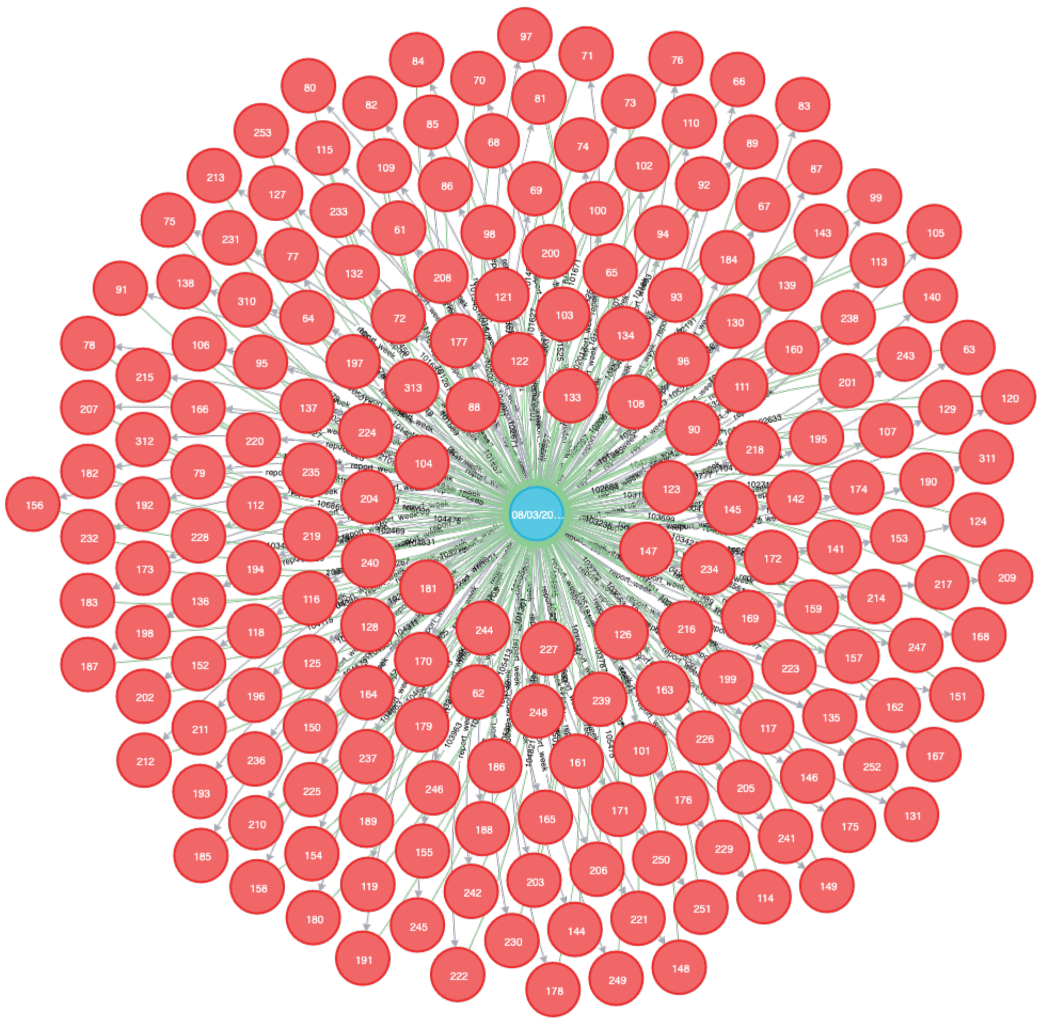


Figure 6.5: Cases reported for a given week

Example 6.2.5. Unknown Travel History

```
SELECT COUNT(*)
FROM IndividualCase
WHERE travel_history_country = "Not␣Reported"


MATCH (r:record)-[]->(t:travel_history_country)
WHERE t.travel_history_country = 'Not␣Reported'
RETURN COUNT(r)
```

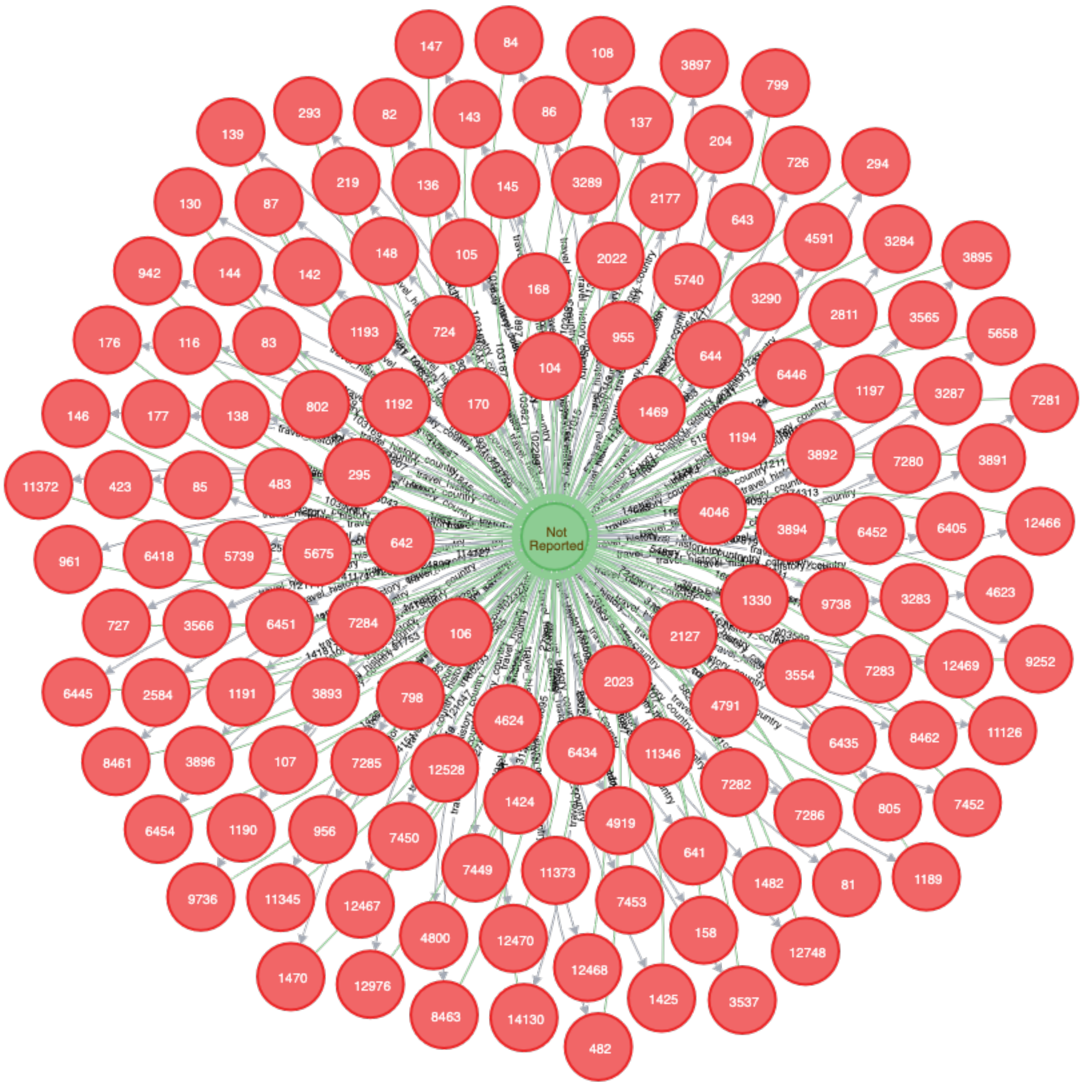


Figure 6.6: Unknown Travel History Cases

Example 6.2.6. BC Case Aged 30-39

```
SELECT * FROM IndividualCase
WHERE province = "BC"
AND age="30-39"


MATCH (p:province)-[]->(r:record)-[]->(a:age)
WHERE p.province = "BC" AND a.age = "30-39"
RETURN r
```

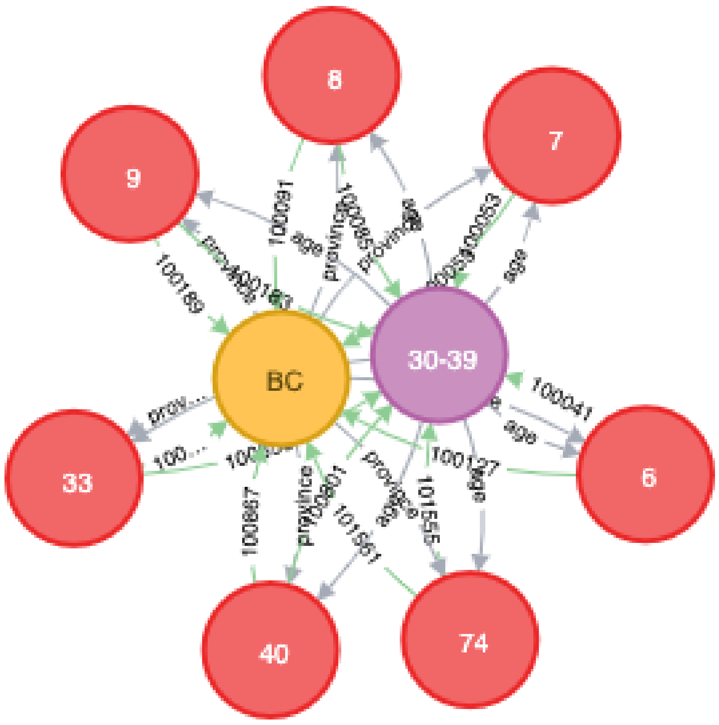


Figure 6.7: BC Case Aged 30-39

Table 6.2: Validation Results

Query	MySQL Result	Neo4j Result	Results
Case Number reported for a given week	197	197	Identical
Unknown Travel History	139	139	Identical
BC Case Aged 30-39	7 Records	7 Records	Identical

6.2.3 Analytical Evaluation

As discussed in chapter 5, the benefits of graph databases have been realized by a wider group in recent years as they allow queries and analyses that provide new insights into the data. In order to demonstrate the pros and cons of a graph database compared to a relational database, four analytical queries were selected to compare the original SQL-based dataset and the new graph database:

- *Select* and *Aggregate* were selected to favour the relational approach.
- *Linked* and *Community Detection* to favour the graph approach.

Example queries 6.2.7 and 6.2.8 determine the possibility of infection for males in the age group 30-39. Queries 6.2.9 and 6.2.10 examine the distribution of infections, seeking the difference by geographic province and by gender. Example 6.2.11 seeks to find links between infection cases. We believe the SQL equivalent is too complex to build. It requires the creation of *stored procedures*, as well as *edges table* [12], so we are not going to include it for this thesis. Example 6.2.12 is a community detection query, specific to graph databases. Before applying 6.2.12, a linkage creation pre-processing step was required to create *record-to-record* relationships based on example 6.2.11. This step used Microsoft Azure and the estimated time was based on a cloud hosted single instance of Neo4j.

Example 6.2.7. Infection-Query-Cypher

```
MATCH (n:sex)-[]-(u:record)-[]-(m:age)
WHERE n.sex='Male' and m.age='30-39'
WITH COUNT(u) as count
MATCH (a)
RETURN count*1.0/count(a) as possibility
```

Example 6.2.8. Infection-Query-SQL

```
SELECT (
  SELECT COUNT(*) FROM 'individual-level-cases'
  WHERE age='30-39' and sex='Male'
)/COUNT(*) AS POSSIBILITY
FROM 'individual-level-cases'
```

Example 6.2.9. Distribution-Query-Cypher

```
MATCH(p:province)-[]->(r:record)<-[]-(s:sex)
WITH p.province AS province,
  COUNT(r) AS number, s.sex AS sex
RETURN province, sex, number
ORDER BY province
```

Example 6.2.10. Distribution-Query-SQL

```
SELECT province, sex, COUNT(*) AS number
FROM 'individual-level-cases'
GROUP BY province, sex
ORDER BY province
```

Example 6.2.11. Linked-Query-Cypher

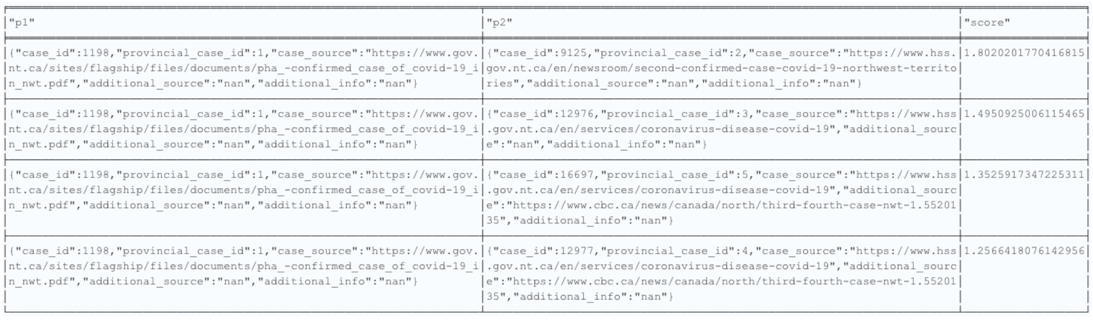
```
MATCH (p1:record)<-[]-(p:province)-[]->(p2:record)
WHERE p1.case_id=1198 AND p1.case_id <> p2.case_id
RETURN p1.case_id, p2.case_id, gds.alpha. \
  linkprediction.adamicAdar(p1, p2) AS score
ORDER BY score DESC
```

Example 6.2.12. Community-Detection

```
CALL gds.labelPropagation.stream('integration-graph')
YIELD nodeId, communityId AS Community
RETURN gds.util.asNode(nodeId) AS Node, Community
ORDER BY Community, Node
```

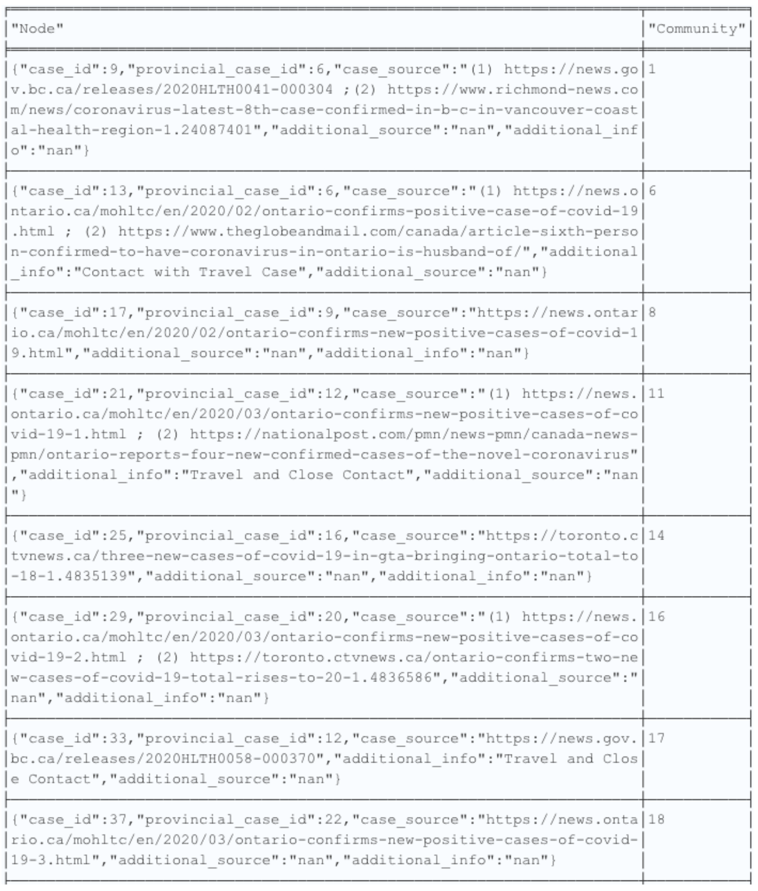
Table 6.3: Analytics Results

Query	Cypher	time(ms)	SQL	time(ms)	Results
Select	Ex3.1	36.2	Ex3.2	15.4	Identical
Aggregate	Ex3.3	58.9	Ex3.4	19.7	Identical
Linked	Ex3.5	319	Not shown	523050	N/A
Comm. Det.	Ex3.6	5000	N/A	N/A	N/A



```
|"p1"                                                            |"p2"                                                            |"score"           |
|{"case_id":1198,"provincial_case_id":1,"case_source":"https://www.gov.|{"case_id":9125,"provincial_case_id":2,"case_source":"https://www.hss.|1.8020201770416815|
|nt.ca/sites/flagship/files/documents/pha_-confirmed_case_of_covid-19_i|gov.nt.ca/en/newsroom/second-confirmed-case-covid-19-northwest-territo|                  |
|n_nwt.pdf","additional_source":"nan","additional_info":"nan"}         |ries","additional_source":"nan","additional_info":"nan"}              |                  |
|{"case_id":1198,"provincial_case_id":1,"case_source":"https://www.gov.|{"case_id":12976,"provincial_case_id":3,"case_source":"https://www.hss|1.4950925006115465|
|nt.ca/sites/flagship/files/documents/pha_-confirmed_case_of_covid-19_i|.gov.nt.ca/en/services/coronavirus-disease-covid-19","additional_sourc|                  |
|n_nwt.pdf","additional_source":"nan","additional_info":"nan"}         |e":"nan","additional_info":"nan"}                                     |                  |
|{"case_id":1198,"provincial_case_id":1,"case_source":"https://www.gov.|{"case_id":16697,"provincial_case_id":5,"case_source":"https://www.hss|1.3525917347225311|
|nt.ca/sites/flagship/files/documents/pha_-confirmed_case_of_covid-19_i|.gov.nt.ca/en/services/coronavirus-disease-covid-19","additional_sourc|                  |
|n_nwt.pdf","additional_source":"nan","additional_info":"nan"}         |e":"https://www.cbc.ca/news/canada/north/third-fourth-case-nwt-1.55201|                  |
|                                                                      |35","additional_info":"nan"}                                          |                  |
|{"case_id":1198,"provincial_case_id":1,"case_source":"https://www.gov.|{"case_id":12977,"provincial_case_id":4,"case_source":"https://www.hss|1.2566418076142956|
|nt.ca/sites/flagship/files/documents/pha_-confirmed_case_of_covid-19_i|.gov.nt.ca/en/services/coronavirus-disease-covid-19","additional_sourc|                  |
|n_nwt.pdf","additional_source":"nan","additional_info":"nan"}         |e":"https://www.cbc.ca/news/canada/north/third-fourth-case-nwt-1.55201|                  |
|                                                                      |35","additional_info":"nan"}                                          |                  |
```

Figure 6.8: Linked Query Result from Neo4j



```
╒══════════════════════════════════════════════════════════════════════╤═══════════╕
│"Node"                                                                │"Community"│
╞══════════════════════════════════════════════════════════════════════╪═══════════╡
│{"case_id":9,"provincial_case_id":6,"case_source":"(1) https://news.go│1          │
│v.bc.ca/releases/2020HLTH0041-000304 ;(2) https://www.richmond-news.co│           │
│m/news/coronavirus-latest-8th-case-confirmed-in-b-c-in-vancouver-coast│           │
│al-health-region-1.24087401","additional_source":"nan","additional_inf│           │
│o":"nan"}                                                             │           │
├──────────────────────────────────────────────────────────────────────┼───────────┤
│{"case_id":13,"provincial_case_id":6,"case_source":"(1) https://news.o│6          │
│ntario.ca/mohltc/en/2020/02/ontario-confirms-positive-case-of-covid-19│           │
│.html ; (2) https://www.theglobeandmail.com/canada/article-sixth-perso│           │
│n-confirmed-to-have-coronavirus-in-ontario-is-husband-of/","additional│           │
│_info":"Contact with Travel Case","additional_source":"nan"}          │           │
├──────────────────────────────────────────────────────────────────────┼───────────┤
│{"case_id":17,"provincial_case_id":9,"case_source":"https://news.ontar│8          │
│io.ca/mohltc/en/2020/02/ontario-confirms-new-positive-cases-of-covid-1│           │
│9.html","additional_source":"nan","additional_info":"nan"}            │           │
├──────────────────────────────────────────────────────────────────────┼───────────┤
│{"case_id":21,"provincial_case_id":12,"case_source":"(1) https://news.│11         │
│ontario.ca/mohltc/en/2020/03/ontario-confirms-new-positive-cases-of-co│           │
│vid-19-1.html ; (2) https://nationalpost.com/pmn/news-pmn/canada-news-│           │
│pmn/ontario-reports-four-new-confirmed-cases-of-the-novel-coronavirus"│           │
│,"additional_info":"Travel and Close Contact","additional_source":"nan│           │
│"}                                                                    │           │
├──────────────────────────────────────────────────────────────────────┼───────────┤
│{"case_id":25,"provincial_case_id":16,"case_source":"https://toronto.c│14         │
│tvnews.ca/three-new-cases-of-covid-19-in-gta-bringing-ontario-total-to│           │
│-18-1.4835139","additional_source":"nan","additional_info":"nan"}     │           │
├──────────────────────────────────────────────────────────────────────┼───────────┤
│{"case_id":29,"provincial_case_id":20,"case_source":"(1) https://news.│16         │
│ontario.ca/mohltc/en/2020/03/ontario-confirms-new-positive-cases-of-co│           │
│vid-19-2.html ; (2) https://toronto.ctvnews.ca/ontario-confirms-two-ne│           │
│w-cases-of-covid-19-total-rises-to-20-1.4836586","additional_source":"│           │
│nan","additional_info":"nan"}                                         │           │
├──────────────────────────────────────────────────────────────────────┼───────────┤
│{"case_id":33,"provincial_case_id":12,"case_source":"https://news.gov.│17         │
│bc.ca/releases/2020HLTH0058-000370","additional_info":"Travel and Clos│           │
│e Contact","additional_source":"nan"}                                 │           │
├──────────────────────────────────────────────────────────────────────┼───────────┤
│{"case_id":37,"provincial_case_id":22,"case_source":"https://news.onta│18         │
│rio.ca/mohltc/en/2020/03/ontario-confirms-new-positive-cases-of-covid-│           │
│19-3.html","additional_source":"nan","additional_info":"nan"}         │           │
├──────────────────────────────────────────────────────────────────────┼───────────┤
```

Figure 6.9: Community Detection Query Result from Neo4j

6.3 Analysis and Discussion

In this section, we will analyze the results shown in the previous section 6.2 and discuss our findings.

6.3.1 Storage Analysis

In figure 6.3, we can see the total storage of 340.71 MB was much bigger for the graph database in comparison to the space taken in MySQL of 0.25 MB. That is because it includes storage information that is not directly related to the data we stored, for example, the `Logical Log` required 59.06MB and is required by Neo4j and should not be included in the comparison.

We believe that the storage types that are directly related to our data which are not metadata that used by Neo4j is as follows:

- `Label Store`
- `Index Store`
- `Schema Store`
- `Node Store`
- `Property Store`
- `Relationship Store`

The total storage size of those types is roughly about 27MB, which is still considerably a much bigger number compared to the 0.25MB used to store the same data in MySQL. This is because, for every pair of nodes that have connections, there are two relationships/edges as well as multiple properties created which lead to the difference in storage requirements. In other words, the way we choose to represent the data has given us the benefits of enhanced analytical powers but has also resulted in increased storage requirements. These issues need to be considered when leveraging our system.

6.3.2 Data Integrity Analysis

From the results shown in Table 6.2, we can see for query *Case Number reported for a given week*, both MySQL and Neo4j returned 197. For query *Unknown Travel History*, both MySQL and Neo4j returned 139. Finally, 7 identical records were retrieved by both databases when query *BC Case Aged 30-39* was applied. To summarise, the identical results retrieved by various queries performed at both Neo4j database and MySQL database have proved that our system is working as expected. At this point, we can confirm that we have a true representation of the original data.

6.3.3 Analytics Result Analysis

Table 6.3 shows the results for the 4 query types in terms of computation times and a comparison of the results where possible. As expected, the *Select* and *Aggregate* queries show SQL having a better performance than Cypher queries but importantly for our validation, the result sets were equivalent. However, SQL can expect an inferior performance where JOINs are required, as Cypher has been shown to significantly out-performed SQL [48].

For the Linked-Query, we used Neo4j build-in algorithm *adamic adar algorithm*, which was introduced in 2003 by Lada Adamic and Eytan Adar to predict links in a social network [3]. The result of the query is shown in figure 6.8. The first column of the result, *p1*, is the base case with case_id equal to 1198 and province_case_id equal to 1. At the second column of the result is the *p2* which is a list of cases that are reported from the same province of case with case_id 1198. At the last column of the result is the *score*, a value of 0 indicates that two nodes are not close, while higher values indicate nodes are closer. So we can see the highest score is about 1.8 indicating the two nodes are closer when compared to other nodes in the list. If we bring Covid-19 into context, that will basically mean the possibility of the case with case_id of 9125 is linked to the case with case_id of 1198 is relative higher compare to other cases in the same geographic area. This algorithm can be useful to find connection patterns among all the COVID cases reported.

We want to highlight that it was not possible to code a meaningful community detection algorithm using SQL for this dataset, so such an algorithm can only be adopted when data is stored in a graph database. The algorithm we applied on Neo4j was the label propagation algorithm (LPA) [51], which is a fast algorithm for finding communities in a graph. It detects these communities using network structure alone as its guide and does not require a pre-defined objective function or prior information about the communities. The result of the query completed in Neo4j is shown in Figure 6.9. In the figure, The *Node* column contains the COVID case records. The *Community* column indicates the community that the record belongs to. But each record appears to have a unique community. The reason is, in order for the community detection algorithm to work, the nodes are required to have a direct relationship. However, in our graph, all the record nodes are connected to each other through common independent nodes such as *report_week* and *province*. Therefore, the community detection algorithm will not be able to bring any valuable insights for our current dataset.

6.3.4 Data Loading Performance

The loading performance between Neo4j database and MySQL database is not a like for like comparison. For the Neo4j database, the time to completion (averaged over multiple runs) for was for Meta analysis: 0.19s; Classification: 0.23 and Loading: 689.44s. The MySQL database, will only take a total of approximately 68 seconds. The loading time difference is big with Neo4j taking around 10 times longer, but that is simply because the large number of relationships created by our system and stored in Neo4j graph database is not applied in the MySQL database. Therefore, we will be unable to drill down in order to compare the two databases, but instead, we are going to discuss the loading performance from our system alone. We want to highlight that the performance of our system is heavily associated with the data we are dealing with. It is the combination of a large number of nodes, properties, and a large number of relationships that leads to the current loading performance of Neo4j graph database. The performance concern may get addressed when adopted by enterprises with machines that have large enough RAM, but for others, it will not be ideal to process data files that contain a large number of fields in conjunction with a large number of rows, unless waiting time is not a key factor.

6.4 Summary

In this chapter, we shared the hardware and software setup for our system as well as the dataset we used for our experiments. We evaluated the system in multiple dimensions including storage, data representation, analytics as well as performance. We performed analysis for the results we gathered from the experiments and talked about all the interesting findings.

In the next chapter, we draw conclusions for our research.

Chapter 7

Conclusions

As we have now reached the end of this dissertation, in section 7.1, we provide a summary of the methodology, evaluation, and results. Then in section 7.2, we suggest some areas for future research which could build upon the research presented in our work.

7.1 Thesis Summary

In this dissertation, we have outlined the benefits of graph database for different purposes and the difficulties of integrating data from different sources, and the construction of graph which often involves a large amount of time consumed by manual works. In order to solve those problems outlined, we presented our automated graph based data integration system which is able to deliver a solution that is capable of integrating multiple data sources and completes graph transformation with minimum human intervention. Thus, we can highlight the 2 key goals we set out to achieve and we will talk about them separately:

1. Automated Schema Integration
2. Automated Graph Data Model Transformation

We have discussed our schema integration methodology in detail in chapter 4. In order to evaluate our integration mythology, we used a list of files in the field of agriculture to demonstrate the result we can achieve by leveraging our integration system. At the end of the integration stage, the most fundamental factors we want to measure against our system is:

- Can the system perform integration on multiple files automatically?
- Will the system result in any data loss in the process of integration?

The evaluation result shown in section 4.7 has shown that no loss of data occurred during the integration process and our system is able to integrate the five files mentioned in 3.2. Our graph data model transformation methodology is presented in chapter 5. In order to test the robustness of our methodology, we used a file that is outside our research domain which is the covid-19 datasets from Kaggle [10].

There are a few questions we asked ourselves when performing the evaluation:

- How much storage space does the graph consume (compared to original sources)?
- Can the system guarantee data integrity?
- What is the performance of our system?
- Can we conduct graph analytics?

In order to answer these questions, we have performed a detailed evaluation primary by comparing with MySQL in 3 dimensions. In section 6.3.1, section 6.3.2 and section 6.3.4 we tried to answer above questions. In addition, we have conducted some analytics comparison between MySQL and Neo4j Graph in section 6.2.3, the result analyzed in section 6.3.3 indicates promising results when using Neo4j graph database over MySQL when it comes to data analytics.

7.2 Future Research

During the work carried out in this thesis some questions which merit further investigation arose. We now revisit these as avenues for future work.

Our system, at the moment, is only dealing with data in CSV format. The reason behind that is any data in tabular form can be easily transformed to CSV format by various tools currently on the market. However, because the goal of our system is to achieve full automation, it is worth extending this research to data models beyond the basic CSV (flat) model.

During our integration process discussed in chapter 4, we identified that where even two files share a potential integration point (same type across two files), the mismatch of the field format can lead to undesirable results. For example, both of the files can contain a date column where in the first file the format being *YYYY-MM-DD* and format being *DD-MM-YYYY* for the second file. Ideally, the format should be consistent across different files for the same type field but that is often not the case. In order to enhance our system to be more resilient, a data unification process can be added on top of the system. It should be invoked before the integration process starts and should be able to unify the data format for the same type to make consistency across all the source files.

In our graph data model, the transformation methodology presented in chapter 5 did not take into account the types of queries that might be expressed on the data. Previous research such as [54] performed an analysis of common queries before deciding on a final structure. We think this might one way to continue our graph transformation algorithm.

There are other ways to optimize our graph design. Any given attribute in the source file is either a `dependent attribute` or an `independent attribute`, and when creating nodes in a graph database, all the independent attributes are created as individual objects. However, some dimensions are *dependent dimensions* which can be a dimension group and may have similar meta features to be classified to dimensions. For example, two attributes named [county] and [country number], we can clearly see that are highly dependent on each other by conducting a quick analysis on the sample data. This suggests that ideally the two attributes should be grouped into a shared Node (one dimension) containing information for both attributes in a Graph. This is not something our system can support at the moment, but if we can enhance our system to support the concept of *independent attribute group*, not only we will be able to boost the performance and save a lot of storage in the process, but also to be able to produce a more condensed graph that is potentially more meaningful to the end user.

Bibliography

[1] Christensson, P. DBMS Definition. Retrieved 2021, Jun 22.

[2] Irish Cattle Breeding Federation. https://www.icbf.com/.

[3] Lada A. Adamic and Eytan Adar. Friends and neighbors on the web. *Soc. Networks*, 25:211–230, 2003.

[4] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, Juan Sequeda, et al. A direct mapping of relational data to rdf. *W3C recommendation*, 27:1–11, 2012.

[5] Timothy G. Armstrong, Vamsi Ponnekanti, Dhruba Borthakur, and Mark Callaghan. Linkbench: A database benchmark based on the facebook social graph. SIGMOD '13, page 1185–1196, New York, NY, USA, 2013. Association for Computing Machinery.

[6] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with coma++. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 906–908, 2005.

[7] Microsoft Azure. https://azure.microsoft.com/.

[8] Janet Heine Barnett. Early writings on graph theory: Euler circuits and the königsberg bridge problem. *Part II: Historical Projects in Discrete Mathematics and Computer Science of Resources for Teaching Discrete*

Mathematics: Classroom Project, History Modules, and Articles, MAA Notes, 74:197–208, 2005.

[9] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, François Yergeau, et al. Extensible markup language (xml) 1.0, 2000.

[10] Roche Canada. Uncover covid-19 challenge, 2020.

[11] David Chaves-Fraga, Freddy Priyatna, Idafen Santana-Perez, and Oscar Corcho. Virtual statistics knowledge graph generation from csv files. In *Emerging Topics in Semantic Technologies*, volume 36 of *Studies on the Semantic Web*, pages 235–244, 10 2018.

[12] Sara Cohen and Netanel Cohen-Tzemach. Implementing link-prediction for social networks in a database system. In *Proceedings of the ACM SIGMOD Workshop on Databases and Social Networks*, DBSocial '13, page 37–42, New York, NY, USA, 2013. Association for Computing Machinery.

[13] Stefan Conrad, Wilhelm Hasselbring, Uwe Hohenstein, Ralf-Detlef Kutsche, Mark Roantree, Gunter Saake, and Fèlix Saltor. Engineering federated information systems: Report of EFIS '99 workshop. *SIGMOD Rec.*, 28(3):9–11, 1999.

[14] Olivier Curé, M. Lamolle, and C. L. Duc. Ontology based data integration over document and column family oriented nosql. *ArXiv*, abs/1307.2603, 2013.

[15] Cypher. *Cypher Manual v4.0*, April 2020. `https://neo4j.com/docs/cypher-manual/current/`.

[16] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2rml: Rdb to rdf mapping language. *W3C recommendation*, 2012.

[17] María del Mar Roldán García, José García-Nieto, and José F. Aldana-Montes. An ontology-based data integration approach for web analytics in e-commerce. *Expert Systems with Applications*, 63:20–34, 2016.

[18] Xin Luna Dong and Theodoros Rekatsinas. Data integration and machine learning: A natural synergy. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 1645–1650, New York, NY, USA, 2018. Association for Computing Machinery.

[19] Xin Luna Dong and Divesh Srivastava. Big data integration. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 1245–1248, 2013.

[20] Manuel García-Solaco, Fèlix Saltor, and Malú Castellanos. A structure based schema integration methodology. In Philip S. Yu and Arbee L. P. Chen, editors, *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, pages 505–512. IEEE Computer Society, 1995.

[21] Piotr Habela, Mark Roantree, and Kazimierz Subieta. Flattening the metamodel for object databases. In Yannis Manolopoulos and Pavol Návrat, editors, *Advances in Databases and Information Systems, 6th East European Conference, ADBIS 2002, Bratislava, Slovakia, September 8-11, 2002, Proceedings*, volume 2435 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 2002.

[22] Reiko Heckel. Graph transformation in a nutshell. *Electronic Notes in Theoretical Computer Science*, 148(1):187–198, 2006. Proceedings of the School of SegraVis Research Training Network on Foundations of Visual Modelling Techniques (FoVMT 2004).

[23] Adam Cowley Jesús Barrasa. https://neo4j.com/labs/neosemantics/.

[24] Ralph Kimball and Margy Ross. *The data warehouse toolkit: the complete guide to dimensional modeling.* John Wiley & Sons, 2011.

[25] V K Kiran and R Vijayakumar. Ontology based data integration of nosql datastores. In *2014 9th International Conference on Industrial and Information Systems (ICIIS)*, pages 1–6, 2014.

[26] Mahesh Lal. *Neo4j graph data modeling.* Packt Publishing Ltd, 2015.

[27] Ora Lassila, Ralph R. Swick, World Wide, and Web Consortium. Resource description framework (rdf) model and syntax specification, 1998.

[28] Timothy Lebo and Gregory Todd Williams. Converting governmental datasets into linked data. In *Proceedings of the 6th International Conference on Semantic Systems*, pages 1–3, 2010.

[29] Timothy Lebo and Gregory Todd Williams. Converting governmental datasets into linked data. In *Proceedings of the 6th International Conference on Semantic Systems*, I-SEMANTICS '10, New York, NY, USA, 2010. Association for Computing Machinery.

[30] Shao Liang, Adel Taweel, Simon Miles, Yevgeniya Kovalchuk, Anastassia Spiridou, Ben Barratt, Uyen Hoang, S Crichton, Brendan Delaney, and Charles Wolfe. Semi automated transformation to owl formatted files as an approach to data integration a feasibility study using environmental, disease register and primary care clinical data. *Methods of information in medicine*, 53, 06 2014.

[31] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170, 2011.

[32] Brian McBride. *The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS*, pages 51–65. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[33] Andrew McCarren, Suzanne McCarthy, Conor O. Sullivan, and Mark Roantree. Anomaly detection in agri warehouse construction. In *Proceedings of the Australasian Computer Science Week Multiconference, ACSW 2017, Geelong, Australia, January 31 - February 3, 2017*, pages 17:1–17:10. ACM, 2017.

[34] Franck Michel. *Integrating heterogeneous data sources in the Web of data.* Theses, Université Côte d'Azur, March 2017.

[35] Franck Michel, Loïc Djimenou, Catherine Faron-Zucker, and Johan Montagnat. Translation of heterogeneous databases into rdf, and application to the construction of a skos taxonomical reference. In Valérie Monfort, Karl-Heinz Krempels, Tim A. Majchrzak, and Žiga Turk, editors, *Web Information Systems and Technologies*, pages 275–296, Cham, 2016. Springer International Publishing.

[36] AB MySQL. Mysql, 2001.

[37] Mark Needham and Amy E Hodler. A comprehensive guide to graph algorithms in neo4j. *Neo4j. com*, 2018.

[38] Mark Needham and Amy E Hodler. *Graph Algorithms: Practical Examples in Apache Spark and Neo4j*. O'Reilly Media, 2019.

[39] Neo4j. Neo4j graph platform.

[40] Neo4j. Top ten reasons for choosing neo4j.

[41] ME Newman. Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 64(1 Pt 2):016132, July 2001.

[42] DuyHoa Ngo and Zohra Bellahsene. Yam++ : A multi-strategy based approach for ontology matching task. In Annette ten Teije, Johanna Völker, Siegfried Handschuh, Heiner Stuckenschmidt, Mathieu d'Acquin, Andriy Nikolov, Nathalie Aussenac-Gilles, and Nathalie Hernandez, editors, *Knowledge Engineering and Knowledge Management*, pages 421–425, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[43] Jim O'Donoghue, Mark Roantree, and Andrew McCarren. Detecting feature interactions in agricultural trade data using a deep neural network. In Ladjel Bellatreche and Sharma Chakravarthy, editors, *Big Data Analytics and Knowledge Discovery - 19th International Conference, DaWaK 2017, Lyon, France, August 28-31, 2017, Proceedings*, volume 10440 of *Lecture Notes in Computer Science*, pages 449–458. Springer, 2017.

[44] Pieter Pauwels, Ruben Verstraeten, Ronald Meyer, and Jan Campenhout. Semantics-based design: Can ontologies help in a preliminary design phase? *Design Principles and Practices. An International Journal.*, 3:263–276, 02 2009.

[45] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of json schema. In *Proceedings of the 25th International Conference on World Wide Web*, pages 263–273. International World Wide Web Conferences Steering Committee, 2016.

[46] Salvatore Flavio Pileggi, Hayden Crain, and Sadok Ben Yahia. An ontological approach to knowledge building by data integration. In Valeria V. Krzhizhanovskaya, Gábor Závodszky, Michael H. Lees, Jack J. Dongarra, Peter M. A. Sloot, Sérgio Brissos, and João Teixeira, editors, *Computational Science – ICCS 2020*, pages 479–493, Cham, 2020. Springer International Publishing.

[47] Christoph Pinkel, Carsten Binnig, Ernesto Jimenez-Ruiz, Evgeny Kharlamov, Andriy Nikolov, Andreas Schwarte, Christian Heupel, and Tim Kraska. Incmap: A journey towards ontology-based data integration. In Bernhard Mitschang, Daniela Nicklas, Frank Leymann, Harald Schöning, Melanie Herschel, Jens Teubner, Theo Härder, Oliver Kopp, and Matthias Wieland, editors, *Datenbanksysteme für Business, Technologie und Web (BTW 2017)*, pages 145–164. Gesellschaft für Informatik, Bonn, 2017.

[48] Neo4j Graph Platform. Concepts: Relational to graph, 2019.

[49] Jaroslav Pokorny. Nosql databases: A step to database scalability in web environment. In *Proceedings of the 13th International Conference on Information Integration and Web-Based Applications and Services*, iiWAS '11, page 278–283, New York, NY, USA, 2011. Association for Computing Machinery.

[50] Xinyu Que, Fabio Checconi, Fabrizio Petrini, and John A. Gunnels. Scalable community detection with the louvain algorithm. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 28–37, 2015.

[51] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.

[52] Mark Roantree, Jessie B. Kennedy, and Peter J. Barclay. Providing views and closure for the object data management group object model. *Inf. Softw. Technol.*, 41(15):1037–1044, 1999.

[53] Mark Roantree, Jessie B. Kennedy, and Peter J. Barclay. Using a metadata software layer in information systems integration. In Klaus R. Dittrich, Andreas Geppert, and Moira C. Norrie, editors, *Advanced Information Systems Engineering, 13th International Conference, CAiSE 2001, Interlaken, Switzerland, June 4-8, 2001, Proceedings*, volume 2068 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2001.

[54] Mark Roantree and Jun Liu. A heuristic approach to selecting views for materialization. *Softw. Pract. Exp.*, 44(10):1157–1179, 2014.

[55] Mark Roantree, Dónall McCann, and Niall Moyna. Integrating sensor streams in phealth networks. In *14th International Conference on Parallel and Distributed Systems, ICPADS 2008, Melbourne, Victoria, Australia, December 8-10, 2008*, pages 320–327. IEEE Computer Society, 2008.

[56] Michael Scriney, Suzanne McCarthy, Andrew McCarren, Paolo Cappellari, and Mark Roantree. Automating data mart construction from semi-structured data sources. *Comput. J.*, 62(3):394–413, 2019.

[57] Michael Scriney, Congcong Xing, Andrew McCarren, and Mark Roantree. Representative sample extraction from web data streams. In Sven Hartmann, Josef Küng, Sharma Chakravarthy, Gabriele Anderst-Kotsis, A Min Tjoa, and

Ismail Khalil, editors, *Database and Expert Systems Applications*, pages 341–351, Cham, 2019. Springer International Publishing.

[58] Teagasc, the Agriculture and Food Development Authority. https://www.teagasc.ie/.

[59] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual.* CreateSpace, Scotts Valley, CA, 2009.

[60] VistaMilk SFI Research Centre. https://www.vistamilk.ie/.

[61] Gligorijević Vladimir and Pržulj Nataša. Methods for biological data integration: perspectives and challenges. 12(112):20150571(10):891–921, 2015.

[62] Mehul Nalin Vora. Hadoop-hbase for large-scale data. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*, volume 1, pages 601–605, 2011.

[63] Laomo Zhang, Ying Ma, and Guodong Wang. An extended hybrid ontology approach to data integration. In *2009 2nd International Conference on Biomedical Engineering and Informatics*, pages 1–4, 2009.

Appendices

Appendix A

LivestockNumbers Ontology

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<neo4j://graph.schema#belong_to> a owl:ObjectProperty;
  rdfs:domain <neo4j://graph.schema#SilageIntake>,
  <neo4j://graph.schema#NumberOfStock>,
    <neo4j://graph.schema#GrassIntake>,
    <neo4j://graph.schema#CoverDate>,
    <neo4j://graph.schema#CoverID>,
    <neo4j://graph.schema#PreviousCoverDate>,
    <neo4j://graph.schema#AverageKgLwt>,
    <neo4j://graph.schema#TotalIntake>,
    <neo4j://graph.schema#LivestockTYpe>,
    <neo4j://graph.schema#MealIntake>;
  rdfs:range <neo4j://graph.schema#LiveStockNumbers>;
  rdfs:label "belong_to" .
<neo4j://graph.schema#SilageIntake> a owl:Class;
  rdfs:label "SilageIntake" .
<neo4j://graph.schema#CoverID> a owl:Class;
  rdfs:label "CoverID" .
<neo4j://graph.schema#AverageKgLwt> a owl:Class;
  rdfs:label "AverageKgLwt" .
<neo4j://graph.schema#TotalIntake> a owl:Class;
```

```
  rdfs:label "TotalIntake" .
<neo4j://graph.schema#LivestockTYpe> a owl:Class;
  rdfs:label "LivestockTYpe" .
<neo4j://graph.schema#GrassIntake> a owl:Class;
  rdfs:label "GrassIntake" .
<neo4j://graph.schema#CoverDate> a owl:liveClass;
  rdfs:label "CoverDate" .
<neo4j://graph.schema#NumberOfStock> a owl:Class;
  rdfs:label "NumberOfStock" .
<neo4j://graph.schema#LiveStockNumbers> a owl:Class;
  rdfs:label "LiveStockNumbers" .
<neo4j://graph.schema#PreviousCoverDate> a owl:Class;
  rdfs:label "PreviousCoverDate" .
<neo4j://graph.schema#MealIntake> a owl:Class;å
```

Appendix B

ManagementDecisions Ontology

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<neo4j://graph.schema#ManagementDecisions> a owl:Class;
  rdfs:label "ManagementDecisions" .
<neo4j://graph.schema#RotationLength> a owl:Class;
  rdfs:label "RotationLength" .
<neo4j://graph.schema#TargetPreGrazingYield> a owl:Class;
  rdfs:label "TargetPreGrazingYield" .
<neo4j://graph.schema#belong$_$to> a owl:ObjectProperty;
  rdfs:domain <neo4j://graph.schema#RotationLength>,
    <neo4j://graph.schema#CoverDate>,
    <neo4j://graph.schema#CoverID>,
    <neo4j://graph.schema#ResidualHeight>,
    <neo4j://graph.schema#TargetPreGrazingYield>,
    <neo4j://graph.schema#DensityOfHerbage>;
  rdfs:range <neo4j://graph.schema#ManagementDecisions>;
  rdfs:label "belong_to" .
<neo4j://graph.schema#CoverID> a owl:Class;
  rdfs:label "CoverID" .
<neo4j://graph.schema#DensityOfHerbage> a owl:Class;
  rdfs:label "DensityOfHerbage" .
<neo4j://graph.schema#ResidualHeight> a owl:Class;
```

```
  rdfs:label "ResidualHeight" .
<neo4j://graph.schema#CoverDate> a owl:Class;
  rdfs:label "CoverDate" .
```

Appendix C

Integrated Metadata Ontology File For First Step

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<neo4j://graph.schema#RotationLength> a owl:Class;
  rdfs:label "RotationLength" .
<neo4j://graph.schema#CoverID> a owl:Class;
  rdfs:label "CoverID" .
<neo4j://graph.schema#AverageKgLwt> a owl:Class;
  rdfs:label "AverageKgLwt" .
<neo4j://graph.schema#DensityOfHerbage> a owl:Class;
  rdfs:label "DensityOfHerbage" .
<neo4j://graph.schema#TotalIntake> a owl:Class;
  rdfs:label "TotalIntake" .
<neo4j://graph.schema#LivestockTYpe> a owl:Class;
  rdfs:label "LivestockTYpe" .
<neo4j://graph.schema#belong_to> a owl:ObjectProperty;
  rdfs:domain <neo4j://graph.schema#CoverID>,
    <neo4j://graph.schema#PreviousCoverDate>,
    <neo4j://graph.schema#LivestockTYpe>,
    <neo4j://graph.schema#MealIntake>,
    <neo4j://graph.schema#SilageIntake>,
    <neo4j://graph.schema#NumberOfStock>,
    <neo4j://graph.schema#GrassIntake>,
    <neo4j://graph.schema#CoverDate>,
    <neo4j://graph.schema#RotationLength>,
    <neo4j://graph.schema#AverageKgLwt>,
    <neo4j://graph.schema#TotalIntake>,
    <neo4j://graph.schema#ResidualHeight>,
    <neo4j://graph.schema#TargetPreGrazingYield>,
    <neo4j://graph.schema#DensityOfHerbage>;
  rdfs:range <neo4j://graph.schema#LivestockNumbers>,
    <neo4j://graph.schema#ManagementDecisions>;
  rdfs:label "belong_to" .
<neo4j://graph.schema#ResidualHeight> a owl:Class;
  rdfs:label "ResidualHeight" .
<neo4j://graph.schema#PreviousCoverDate> a owl:Class;
  rdfs:label "PreviousCoverDate" .
<neo4j://graph.schema#ManagementDecisions> a owl:Class;
  rdfs:label "ManagementDecisions" .
<neo4j://graph.schema#SilageIntake> a owl:Class;
  rdfs:label "SilageIntake" .
<neo4j://graph.schema#TargetPreGrazingYield> a owl:Class;
  rdfs:label "TargetPreGrazingYield" .
<neo4j://graph.schema#GrassIntake> a owl:Class;
  rdfs:label "GrassIntake" .
<neo4j://graph.schema#CoverDate> a owl:Class;
  rdfs:label "CoverDate" .
<neo4j://graph.schema#NumberOfStock> a owl:Class;
  rdfs:label "NumberOfStock" .
<neo4j://graph.schema#LivestockNumbers> a owl:Class;
  rdfs:label "LivestockNumbers" .
<neo4j://graph.schema#MealIntake> a owl:Class;
  rdfs:label "MealIntake" .
```

Appendix D

PaddockEstimation Ontology

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<neo4j://graph.schema#PoachingLevel> a owl:Class;
  rdfs:label "PoachingLevel" .
<neo4j://graph.schema#CoverID> a owl:Class;
  rdfs:label "CoverID" .
<neo4j://graph.schema#GrazeDate> a owl:Class;
  rdfs:label "GrazeDate" .
<neo4j://graph.schema#belong_to> a owl:ObjectProperty;
  rdfs:domain <neo4j://graph.schema#HerbageEstKgDmHa>,
    <neo4j://graph.schema#PaddockArea>,
    <neo4j://graph.schema#GrowthRate>,
    <neo4j://graph.schema#AssociatedSoils>,
    <neo4j://graph.schema#CoverEstimations_PaddockID>,
    <neo4j://graph.schema#PaddockDistanceFromParlour>,
    <neo4j://graph.schema#Height>,
    <neo4j://graph.schema#PoachingLevel>,
    <neo4j://graph.schema#PrincipalSoil>,
    <neo4j://graph.schema#ResidualAfterCuttingCM>,
    <neo4j://graph.schema#Topping>,
    <neo4j://graph.schema#PreviousHerbageEstKgDmHa1>,
    <neo4j://graph.schema#CutDate>,
    <neo4j://graph.schema#SilageYieldKgDmHa>,
    <neo4j://graph.schema#Gradients_Description>,
    <neo4j://graph.schema#Aspects_Description>,
    <neo4j://graph.schema#ReseedMethodID>,
    <neo4j://graph.schema#CoverID>,
    <neo4j://graph.schema#GrazeDate>,
    <neo4j://graph.schema#ReseedDate>,
    <neo4j://graph.schema#PreviousCoverDate>,
    <neo4j://graph.schema#ANONID>,
    <neo4j://graph.schema#PaddockAltitude>,
    <neo4j://graph.schema#PaddockStatus>,
    <neo4j://graph.schema#PaddockStatusPrevious>,
    <neo4j://graph.schema#PreGrazingYield>,
    <neo4j://graph.schema#PoachingEvent>,
    <neo4j://graph.schema#ResidualAfterGrazingCM>,
    <neo4j://graph.schema#CoverDate>,
    <neo4j://graph.schema#ParentMaterial>,
    <neo4j://graph.schema#defoliatationoption>,
    <neo4j://graph.schema#ResidualAfterToppingCM>,
    <neo4j://graph.schema#SoilNumber>,
    <neo4j://graph.schema#DrainageCharacteristics_Description>,
    <neo4j://graph.schema#PreviousGrazeDate>;
  rdfs:range <neo4j://graph.schema#PadDockEstimations>;
  rdfs:label "belong_to" .
<neo4j://graph.schema#Height> a owl:Class;
  rdfs:label "Height" .
<neo4j://graph.schema#CoverEstimations_PaddockID> a owl:Class;
  rdfs:label "CoverEstimations_PaddockID" .
<neo4j://graph.schema#AssociatedSoils> a owl:Class;
  rdfs:label "AssociatedSoils" .
<neo4j://graph.schema#PoachingEvent> a owl:Class;
  rdfs:label "PoachingEvent" .
<neo4j://graph.schema#Gradients_Description> a owl:Class;
  rdfs:label "Gradients_Description" .
<neo4j://graph.schema#GrowthRate> a owl:Class;
```

```
  rdfs:label "GrowthRate" .
<neo4j://graph.schema#Topping> a owl:Class;
  rdfs:label "Topping" .
<neo4j://graph.schema#PrincipalSoil> a owl:Class;
  rdfs:label "PrincipalSoil" .
<neo4j://graph.schema#ANONID> a owl:Class;
  rdfs:label "ANONID" .
<neo4j://graph.schema#SoilNumber> a owl:Class;
  rdfs:label "SoilNumber" .
<neo4j://graph.schema#PreviousGrazeDate> a owl:Class;
  rdfs:label "PreviousGrazeDate" .
<neo4j://graph.schema#PreGrazingYield> a owl:Class;
  rdfs:label "PreGrazingYield" .
<neo4j://graph.schema#PreviousCoverDate> a owl:Class;
  rdfs:label "PreviousCoverDate" .
<neo4j://graph.schema#ResidualAfterCuttingCM> a owl:Class;
  rdfs:label "ResidualAfterCuttingCM" .
<neo4j://graph.schema#PaddockAltitude> a owl:Class;
  rdfs:label "PaddockAltitude" .
<neo4j://graph.schema#DrainageCharacteristics_Description> a owl:Class;
  rdfs:label "DrainageCharacteristics_Description" .
<neo4j://graph.schema#PreviousHerbageEstKgDmHa1> a owl:Class;
  rdfs:label "PreviousHerbageEstKgDmHa1" .
<neo4j://graph.schema#CoverDate> a owl:Class;
  rdfs:label "CoverDate" .
<neo4j://graph.schema#PaddockDistanceFromParlour> a owl:Class;
  rdfs:label "PaddockDistanceFromParlour" .
<neo4j://graph.schema#SilageYieldKgDmHa> a owl:Class;
  rdfs:label "SilageYieldKgDmHa" .
<neo4j://graph.schema#PaddockArea> a owl:Class;
  rdfs:label "PaddockArea" .
<neo4j://graph.schema#PaddockStatusPrevious> a owl:Class;
  rdfs:label "PaddockStatusPrevious" .
<neo4j://graph.schema#PadDockEstimations> a owl:Class;
  rdfs:label "PadDockEstimations" .
```

```
<neo4j://graph.schema#ParentMaterial> a owl:Class;
  rdfs:label "ParentMaterial" .
<neo4j://graph.schema#defoliatationoption> a owl:Class;
  rdfs:label "defoliatationoption" .
<neo4j://graph.schema#ResidualAfterToppingCM> a owl:Class;
  rdfs:label "ResidualAfterToppingCM" .
<neo4j://graph.schema#PaddockStatus> a owl:Class;
  rdfs:label "PaddockStatus" .
<neo4j://graph.schema#ReseedMethodID> a owl:Class;
  rdfs:label "ReseedMethodID" .
<neo4j://graph.schema#ReseedDate> a owl:Class;
  rdfs:label "ReseedDate" .
<neo4j://graph.schema#CutDate> a owl:Class;
  rdfs:label "CutDate" .
<neo4j://graph.schema#Aspects_Description> a owl:Class;
  rdfs:label "Aspects_Description" .
<neo4j://graph.schema#HerbageEstKgDmHa> a owl:Class;
  rdfs:label "HerbageEstKgDmHa" .
<neo4j://graph.schema#ResidualAfterGrazingCM> a owl:Class;
  rdfs:label "ResidualAfterGrazingCM" .
```

Appendix E

Integrated Metadata Ontology Extended With PaddockEstimation

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<neo4j://graph.schema#PoachingLevel> a owl:Class;
  rdfs:label "PoachingLevel" .
<neo4j://graph.schema#CoverID> a owl:Class;
  rdfs:label "CoverID" .
<neo4j://graph.schema#GrazeDate> a owl:Class;
  rdfs:label "GrazeDate" .
<neo4j://graph.schema#AverageKgLwt> a owl:Class;
  rdfs:label "AverageKgLwt" .
<neo4j://graph.schema#belong_to> a owl:ObjectProperty;
  rdfs:domain <neo4j://graph.schema#HerbageEstKgDmHa>,
    <neo4j://graph.schema#PaddockArea>,
    <neo4j://graph.schema#GrowthRate>,
    <neo4j://graph.schema#AssociatedSoils>,
    <neo4j://graph.schema#MealIntake>,
    <neo4j://graph.schema#CoverEstimations_PaddockID>,
    <neo4j://graph.schema#PaddockDistanceFromParlour>,
    <neo4j://graph.schema#Height>,
    <neo4j://graph.schema#PoachingLevel>,
    <neo4j://graph.schema#SilageIntake>,
    <neo4j://graph.schema#NumberOfStock>,
    <neo4j://graph.schema#PrincipalSoil>,
    <neo4j://graph.schema#ResidualAfterCuttingCM>,
    <neo4j://graph.schema#Topping>,
    <neo4j://graph.schema#TotalIntake>,
    <neo4j://graph.schema#PreviousHerbageEstKgDmHa1>,
    <neo4j://graph.schema#CutDate>,
    <neo4j://graph.schema#SilageYieldKgDmHa>,
    <neo4j://graph.schema#TargetPreGrazingYield>,
    <neo4j://graph.schema#Gradients_Description>,
    <neo4j://graph.schema#DensityOfHerbage>,
    <neo4j://graph.schema#Aspects_Description>,
    <neo4j://graph.schema#ReseedMethodID>,
    <neo4j://graph.schema#CoverID>,
    <neo4j://graph.schema#GrazeDate>,
    <neo4j://graph.schema#ReseedDate>,
    <neo4j://graph.schema#PreviousCoverDate>,
    <neo4j://graph.schema#ANONID>,
    <neo4j://graph.schema#PaddockAltitude>,
    <neo4j://graph.schema#LivestockTYpe>,
    <neo4j://graph.schema#PaddockStatus>,
    <neo4j://graph.schema#PaddockStatusPrevious>,
    <neo4j://graph.schema#PreGrazingYield>,
    <neo4j://graph.schema#GrassIntake>,
    <neo4j://graph.schema#ResidualAfterGrazingCM>,
    <neo4j://graph.schema#PoachingEvent>,
    <neo4j://graph.schema#CoverDate>,
    <neo4j://graph.schema#ParentMaterial>,
    <neo4j://graph.schema#defoliatationoption>,
    <neo4j://graph.schema#ResidualAfterToppingCM>,
    <neo4j://graph.schema#RotationLength>,
    <neo4j://graph.schema#AverageKgLwt>,
    <neo4j://graph.schema#ResidualHeight>,
    <neo4j://graph.schema#SoilNumber>,
    <neo4j://graph.schema#DrainageCharacteristics_Description>,
    <neo4j://graph.schema#PreviousGrazeDate>;
  rdfs:range <neo4j://graph.schema#PadDockEstimations>, <neo4j://graph.schema#LiveStockNumber
    <neo4j://graph.schema#ManagementDecisions>;
  rdfs:label "belong_to" .
<neo4j://graph.schema#DensityOfHerbage> a owl:Class;
  rdfs:label "DensityOfHerbage" .
<neo4j://graph.schema#LivestockTYpe> a owl:Class;
  rdfs:label "LivestockTYpe" .
<neo4j://graph.schema#Height> a owl:Class;
  rdfs:label "Height" .
<neo4j://graph.schema#CoverEstimations_PaddockID> a owl:Class;
  rdfs:label "CoverEstimations_PaddockID" .
<neo4j://graph.schema#AssociatedSoils> a owl:Class;
  rdfs:label "AssociatedSoils" .
<neo4j://graph.schema#ResidualHeight> a owl:Class;
  rdfs:label "ResidualHeight" .
<neo4j://graph.schema#PoachingEvent> a owl:Class;
  rdfs:label "PoachingEvent" .
<neo4j://graph.schema#Gradients_Description> a owl:Class;
  rdfs:label "Gradients_Description" .
<neo4j://graph.schema#GrowthRate> a owl:Class;
  rdfs:label "GrowthRate" .
<neo4j://graph.schema#PrincipalSoil> a owl:Class;
  rdfs:label "PrincipalSoil" .
<neo4j://graph.schema#Topping> a owl:Class;
  rdfs:label "Topping" .
<neo4j://graph.schema#ANONID> a owl:Class;
  rdfs:label "ANONID" .
<neo4j://graph.schema#SoilNumber> a owl:Class;
  rdfs:label "SoilNumber" .
<neo4j://graph.schema#PreviousGrazeDate> a owl:Class;
```

```
  rdfs:label "PreviousGrazeDate" .
<neo4j://graph.schema#PreGrazingYield> a owl:Class;
  rdfs:label "PreGrazingYield" .
<neo4j://graph.schema#PreviousCoverDate> a owl:Class;
  rdfs:label "PreviousCoverDate" .
<neo4j://graph.schema#ResidualAfterCuttingCM> a owl:Class;
  rdfs:label "ResidualAfterCuttingCM" .
<neo4j://graph.schema#SilageIntake> a owl:Class;
  rdfs:label "SilageIntake" .
<neo4j://graph.schema#PaddockAltitude> a owl:Class;
  rdfs:label "PaddockAltitude" .
<neo4j://graph.schema#DrainageCharacteristics_Description> a owl:Class;
  rdfs:label "DrainageCharacteristics_Description" .
<neo4j://graph.schema#GrassIntake> a owl:Class;
  rdfs:label "GrassIntake" .
<neo4j://graph.schema#PreviousHerbageEstKgDmHa1> a owl:Class;
  rdfs:label "PreviousHerbageEstKgDmHa1" .
<neo4j://graph.schema#CoverDate> a owl:Class;
  rdfs:label "CoverDate" .
<neo4j://graph.schema#NumberOfStock> a owl:Class;
  rdfs:label "NumberOfStock" .
<neo4j://graph.schema#LiveStockNumbers> a owl:Class;
  rdfs:label "LiveStockNumbers" .
<neo4j://graph.schema#PaddockDistanceFromParlour> a owl:Class;
  rdfs:label "PaddockDistanceFromParlour" .
<neo4j://graph.schema#SilageYieldKgDmHa> a owl:Class;
  rdfs:label "SilageYieldKgDmHa" .
<neo4j://graph.schema#PaddockArea> a owl:Class;
  rdfs:label "PaddockArea" .
<neo4j://graph.schema#PaddockStatusPrevious> a owl:Class;
  rdfs:label "PaddockStatusPrevious" .
<neo4j://graph.schema#PadDockEstimations> a owl:Class;
  rdfs:label "PadDockEstimations" .
<neo4j://graph.schema#TargetPreGrazingYield> a owl:Class;
  rdfs:label "TargetPreGrazingYield" .
```

```
<neo4j://graph.schema#ParentMaterial> a owl:Class;
  rdfs:label "ParentMaterial" .
<neo4j://graph.schema#defoliatationoption> a owl:Class;
  rdfs:label "defoliatationoption" .
<neo4j://graph.schema#ResidualAfterToppingCM> a owl:Class;
  rdfs:label "ResidualAfterToppingCM" .
<neo4j://graph.schema#PaddockStatus> a owl:Class;
  rdfs:label "PaddockStatus" .
<neo4j://graph.schema#MealIntake> a owl:Class;
  rdfs:label "MealIntake" .
<neo4j://graph.schema#RotationLength> a owl:Class;
  rdfs:label "RotationLength" .
<neo4j://graph.schema#TotalIntake> a owl:Class;
  rdfs:label "TotalIntake" .
<neo4j://graph.schema#ReseedMethodID> a owl:Class;
  rdfs:label "ReseedMethodID" .
<neo4j://graph.schema#ManagementDecisions> a owl:Class;
  rdfs:label "ManagementDecisions" .
<neo4j://graph.schema#ReseedDate> a owl:Class;
  rdfs:label "ReseedDate" .
<neo4j://graph.schema#CutDate> a owl:Class;
  rdfs:label "CutDate" .
<neo4j://graph.schema#Aspects_Description> a owl:Class;
  rdfs:label "Aspects_Description" .
<neo4j://graph.schema#HerbageEstKgDmHa> a owl:Class;
  rdfs:label "HerbageEstKgDmHa" .
<neo4j://graph.schema#ResidualAfterGrazingCM> a owl:Class;
  rdfs:label "ResidualAfterGrazingCM" .
```

Appendix F

PbiLactation Ontology

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<neo4j://graph.schema#belong_to> a owl:ObjectProperty;
  rdfs:domain <neo4j://graph.schema#LACTATION>,
    <neo4j://graph.schema#End_Lactation_Date>,
    <neo4j://graph.schema#Calving_Date>,
    <neo4j://graph.schema#ENC_ANI_ID>;
  rdfs:label "belong_to";
  rdfs:range <neo4j://graph.schema#PbiLactation> .
<neo4j://graph.schema#PbiLactation> a owl:Class;
  rdfs:label "PbiLactation" .
<neo4j://graph.schema#Calving_Date> a owl:Class;
  rdfs:label "Calving_Date" .
<neo4j://graph.schema#LACTATION> a owl:Class;
  rdfs:label "LACTATION" .
<neo4j://graph.schema#ENC_ANI_ID> a owl:Class;
  rdfs:label "ENC_ANI_ID" .
<neo4j://graph.schema#End_Lactation_Date> a owl:Class;
  rdfs:label "End_Lactation_Date" .
```

Appendix G

Integrated Metadata Ontology Extended With PbiLactation

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<neo4j://graph.schema#PoachingLevel> a owl:Class;
  rdfs:label "PoachingLevel" .
<neo4j://graph.schema#PbiLactation> a owl:Class;
  rdfs:label "PbiLactation" .
<neo4j://graph.schema#CoverID> a owl:Class;
  rdfs:label "CoverID" .
<neo4j://graph.schema#GrazeDate> a owl:Class;
  rdfs:label "GrazeDate" .
<neo4j://graph.schema#AverageKgLwt> a owl:Class;
  rdfs:label "AverageKgLwt" .
<neo4j://graph.schema#belong_to> a owl:ObjectProperty;
  rdfs:domain <neo4j://graph.schema#HerbageEstKgDmHa>,
    <neo4j://graph.schema#ENC_ANI_ID>,
    <neo4j://graph.schema#PaddockArea>,
    <neo4j://graph.schema#GrowthRate>,
    <neo4j://graph.schema#LACTATION>,
    <neo4j://graph.schema#AssociatedSoils>,
    <neo4j://graph.schema#MealIntake>,
    <neo4j://graph.schema#CoverEstimations_PaddockID>,
    <neo4j://graph.schema#PaddockDistanceFromParlour>,
    <neo4j://graph.schema#Height>,
    <neo4j://graph.schema#PoachingLevel>,
    <neo4j://graph.schema#SilageIntake>,
    <neo4j://graph.schema#NumberOfStock>,
    <neo4j://graph.schema#PrincipalSoil>,
    <neo4j://graph.schema#ResidualAfterCuttingCM>,
    <neo4j://graph.schema#Topping>,
    <neo4j://graph.schema#TotalIntake>,
    <neo4j://graph.schema#PreviousHerbageEstKgDmHa1>,
    <neo4j://graph.schema#CutDate>,
    <neo4j://graph.schema#SilageYieldKgDmHa>,
    <neo4j://graph.schema#TargetPreGrazingYield>,
    <neo4j://graph.schema#Gradients_Description>,
    <neo4j://graph.schema#DensityOfHerbage>,
    <neo4j://graph.schema#Aspects_Description>,
    <neo4j://graph.schema#ReseedMethodID>,
    <neo4j://graph.schema#Calving_Date>,
    <neo4j://graph.schema#CoverID>,
    <neo4j://graph.schema#GrazeDate>,
    <neo4j://graph.schema#ReseedDate>,
    <neo4j://graph.schema#PreviousCoverDate>,
    <neo4j://graph.schema#PaddockAltitude>,
    <neo4j://graph.schema#ANONID>,
    <neo4j://graph.schema#LivestockTYpe>,
    <neo4j://graph.schema#PaddockStatus>,
    <neo4j://graph.schema#PaddockStatusPrevious>,
    <neo4j://graph.schema#PreGrazingYield>,
    <neo4j://graph.schema#GrassIntake>,
    <neo4j://graph.schema#PoachingEvent>,
    <neo4j://graph.schema#ResidualAfterGrazingCM>,
    <neo4j://graph.schema#End_Lactation_Date>,
    <neo4j://graph.schema#CoverDate>,
    <neo4j://graph.schema#ParentMaterial>,
    <neo4j://graph.schema#defoliatationoption>,
    <neo4j://graph.schema#ResidualAfterToppingCM>,
    <neo4j://graph.schema#RotationLength>,
    <neo4j://graph.schema#AverageKgLwt>,
    <neo4j://graph.schema#ResidualHeight>,
    <neo4j://graph.schema#SoilNumber>,
    <neo4j://graph.schema#DrainageCharacteristics_Description>,
    <neo4j://graph.schema#PreviousGrazeDate>;
  rdfs:range <neo4j://graph.schema#PadDockEstimations>,
    <neo4j://graph.schema#PbiLactation>,
    <neo4j://graph.schema#LiveStockNumbers>,
    <neo4j://graph.schema#ManagementDecisions>;
  rdfs:label "belong_to" .
<neo4j://graph.schema#DensityOfHerbage> a owl:Class;
  rdfs:label "DensityOfHerbage" .
<neo4j://graph.schema#LivestockTYpe> a owl:Class;
  rdfs:label "LivestockTYpe" .
<neo4j://graph.schema#Height> a owl:Class;
  rdfs:label "Height" .
<neo4j://graph.schema#CoverEstimations_PaddockID> a owl:Class;
  rdfs:label "CoverEstimations_PaddockID" .
<neo4j://graph.schema#AssociatedSoils> a owl:Class;
  rdfs:label "AssociatedSoils" .
<neo4j://graph.schema#ResidualHeight> a owl:Class;
  rdfs:label "ResidualHeight" .
<neo4j://graph.schema#PoachingEvent> a owl:Class;
  rdfs:label "PoachingEvent" .
<neo4j://graph.schema#Gradients_Description> a owl:Class;
  rdfs:label "Gradients_Description" .
<neo4j://graph.schema#GrowthRate> a owl:Class;
  rdfs:label "GrowthRate" .
<neo4j://graph.schema#PrincipalSoil> a owl:Class;
  rdfs:label "PrincipalSoil" .
<neo4j://graph.schema#Topping> a owl:Class;
```

```
  rdfs:label "Topping" .
<neo4j://graph.schema#ANONID> a owl:Class;
  rdfs:label "ANONID" .
<neo4j://graph.schema#SoilNumber> a owl:Class;
  rdfs:label "SoilNumber" .
<neo4j://graph.schema#PreviousGrazeDate> a owl:Class;
  rdfs:label "PreviousGrazeDate" .
<neo4j://graph.schema#PreGrazingYield> a owl:Class;
  rdfs:label "PreGrazingYield" .
<neo4j://graph.schema#LACTATION> a owl:Class;
  rdfs:label "LACTATION" .
<neo4j://graph.schema#End_Lactation_Date> a owl:Class;
  rdfs:label "End_Lactation_Date" .
<neo4j://graph.schema#PreviousCoverDate> a owl:Class;
  rdfs:label "PreviousCoverDate" .
<neo4j://graph.schema#ResidualAfterCuttingCM> a owl:Class;
  rdfs:label "ResidualAfterCuttingCM" .
<neo4j://graph.schema#SilageIntake> a owl:Class;
  rdfs:label "SilageIntake" .
<neo4j://graph.schema#PaddockAltitude> a owl:Class;
  rdfs:label "PaddockAltitude" .
<neo4j://graph.schema#DrainageCharacteristics_Description> a owl:Class;
  rdfs:label "DrainageCharacteristics_Description" .
<neo4j://graph.schema#GrassIntake> a owl:Class;
  rdfs:label "GrassIntake" .
<neo4j://graph.schema#PreviousHerbageEstKgDmHa1> a owl:Class;
  rdfs:label "PreviousHerbageEstKgDmHa1" .
<neo4j://graph.schema#CoverDate> a owl:Class;
  rdfs:label "CoverDate" .
<neo4j://graph.schema#NumberOfStock> a owl:Class;
  rdfs:label "NumberOfStock" .
<neo4j://graph.schema#LiveStockNumbers> a owl:Class;
  rdfs:label "LiveStockNumbers" .
<neo4j://graph.schema#PaddockDistanceFromParlour> a owl:Class;
  rdfs:label "PaddockDistanceFromParlour" .
```

```
<neo4j://graph.schema#SilageYieldKgDmHa> a owl:Class;
  rdfs:label "SilageYieldKgDmHa" .
<neo4j://graph.schema#PaddockArea> a owl:Class;
  rdfs:label "PaddockArea" .
<neo4j://graph.schema#PaddockStatusPrevious> a owl:Class;
  rdfs:label "PaddockStatusPrevious" .
<neo4j://graph.schema#PadDockEstimations> a owl:Class;
  rdfs:label "PadDockEstimations" .
<neo4j://graph.schema#TargetPreGrazingYield> a owl:Class;
  rdfs:label "TargetPreGrazingYield" .
<neo4j://graph.schema#ParentMaterial> a owl:Class;
  rdfs:label "ParentMaterial" .
<neo4j://graph.schema#Calving_Date> a owl:Class;
  rdfs:label "Calving_Date" .
<neo4j://graph.schema#defoliatationoption> a owl:Class;
  rdfs:label "defoliatationoption" .
<neo4j://graph.schema#ResidualAfterToppingCM> a owl:Class;
  rdfs:label "ResidualAfterToppingCM" .
<neo4j://graph.schema#PaddockStatus> a owl:Class;
  rdfs:label "PaddockStatus" .
<neo4j://graph.schema#MealIntake> a owl:Class;
  rdfs:label "MealIntake" .
<neo4j://graph.schema#RotationLength> a owl:Class;
  rdfs:label "RotationLength" .
<neo4j://graph.schema#TotalIntake> a owl:Class;
  rdfs:label "TotalIntake" .
<neo4j://graph.schema#ReseedMethodID> a owl:Class;
  rdfs:label "ReseedMethodID" .
<neo4j://graph.schema#ManagementDecisions> a owl:Class;
  rdfs:label "ManagementDecisions" .
<neo4j://graph.schema#ReseedDate> a owl:Class;
  rdfs:label "ReseedDate" .
<neo4j://graph.schema#CutDate> a owl:Class;
  rdfs:label "CutDate" .
<neo4j://graph.schema#Aspects_Description> a owl:Class;
```

```
  rdfs:label "Aspects_Description" .
<neo4j://graph.schema#HerbageEstKgDmHa> a owl:Class;
  rdfs:label "HerbageEstKgDmHa" .
<neo4j://graph.schema#ENC_ANI_ID> a owl:Class;
  rdfs:label "ENC_ANI_ID" .
<neo4j://graph.schema#ResidualAfterGrazingCM> a owl:Class;
  rdfs:label "ResidualAfterGrazingCM" .
```

Appendix H

IndividualCase Ontology

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<neo4j://graph.schema#sex> a owl:Class;
  rdfs:label "sex" .
<neo4j://graph.schema#belong_to> a owl:ObjectProperty;
  rdfs:domain <neo4j://graph.schema#case_source>,
    <neo4j://graph.schema#health_region>,
    <neo4j://graph.schema#travel_history_country>,
    <neo4j://graph.schema#report_week>,
    <neo4j://graph.schema#province>,
    <neo4j://graph.schema#method_note>,
    <neo4j://graph.schema#provincial_case_id>,
    <neo4j://graph.schema#additional_source>,
    <neo4j://graph.schema#travel_yn>,
    <neo4j://graph.schema#date_report>,
    <neo4j://graph.schema#sex>,
    <neo4j://graph.schema#locally_acquired>,
    <neo4j://graph.schema#additional_info>,
    <neo4j://graph.schema#country>,
    <neo4j://graph.schema#case_id>,
    <neo4j://graph.schema#age>;
  rdfs:range <neo4j://graph.schema#IndividualCase>;
  rdfs:label "belong_to" .
<neo4j://graph.schema#age> a owl:Class;
  rdfs:label "age" .
<neo4j://graph.schema#province> a owl:Class;
  rdfs:label "province" .
<neo4j://graph.schema#additional_info> a owl:Class;
  rdfs:label "additional_info" .
<neo4j://graph.schema#locally_acquired> a owl:Class;
  rdfs:label "locally_acquired" .
<neo4j://graph.schema#additional_source> a owl:Class;
  rdfs:label "additional_source" .
<neo4j://graph.schema#travel_yn> a owl:Class;
  rdfs:label "travel_yn" .
<neo4j://graph.schema#travel_history_country> a owl:Class;
  rdfs:label "travel_history_country" .
<neo4j://graph.schema#case_source> a owl:Class;
  rdfs:label "case_source" .
<neo4j://graph.schema#health_region> a owl:Class;
  rdfs:label "health_region" .
<neo4j://graph.schema#provincial_case_id> a owl:Class;
  rdfs:label "provincial_case_id" .
<neo4j://graph.schema#method_note> a owl:Class;
  rdfs:label "method_note" .
<neo4j://graph.schema#date_report> a owl:Class;
  rdfs:label "date_report" .
<neo4j://graph.schema#report_week> a owl:Class;
  rdfs:label "report_week" .
<neo4j://graph.schema#IndividualCase> a owl:Class;
  rdfs:label "IndividualCase" .
<neo4j://graph.schema#country> a owl:Class;
  rdfs:label "country" .
<neo4j://graph.schema#case_id> a owl:Class;
  rdfs:label "case_id" .
```

Appendix I

Integrated Metadata Ontology Extended With IndividualCase

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<neo4j://graph.schema#PoachingLevel> a owl:Class;
  rdfs:label "PoachingLevel" .
<neo4j://graph.schema#sex> a owl:Class;
  rdfs:label "sex" .
<neo4j://graph.schema#CoverID> a owl:Class;
  rdfs:label "CoverID" .
<neo4j://graph.schema#GrazeDate> a owl:Class;
  rdfs:label "GrazeDate" .
<neo4j://graph.schema#DensityOfHerbage> a owl:Class;
  rdfs:label "DensityOfHerbage" .
<neo4j://graph.schema#LivestockTYpe> a owl:Class;
  rdfs:label "LivestockTYpe" .
<neo4j://graph.schema#belong_to> a owl:ObjectProperty;
  rdfs:domain <neo4j://graph.schema#PaddockArea>,
    <neo4j://graph.schema#GrowthRate>,
    <neo4j://graph.schema#health_region>,
    <neo4j://graph.schema#AssociatedSoils>,
    <neo4j://graph.schema#travel_history_country>,
    <neo4j://graph.schema#Height>,
    <neo4j://graph.schema#PoachingLevel>,
    <neo4j://graph.schema#travel_yn>,
    <neo4j://graph.schema#date_report>,
    <neo4j://graph.schema#SilageIntake>,
    <neo4j://graph.schema#NumberOfStock>,
    <neo4j://graph.schema#ResidualAfterCuttingCM>,
    <neo4j://graph.schema#sex>,
    <neo4j://graph.schema#Gradients_Description>,
    <neo4j://graph.schema#TargetPreGrazingYield>,
    <neo4j://graph.schema#country>,
    <neo4j://graph.schema#ReseedMethodID>,
    <neo4j://graph.schema#Calving_Date>,
    <neo4j://graph.schema#CoverID>,
    <neo4j://graph.schema#PreviousCoverDate>,
    <neo4j://graph.schema#ReseedDate>,
    <neo4j://graph.schema#ANONID>,
    <neo4j://graph.schema#PaddockStatusPrevious>,
    <neo4j://graph.schema#province>,
    <neo4j://graph.schema#provincial_case_id>,
    <neo4j://graph.schema#PreGrazingYield>,
    <neo4j://graph.schema#ResidualAfterGrazingCM>,
    <neo4j://graph.schema#End_Lactation_Date>,
    <neo4j://graph.schema#defoliatationoption>,
    <neo4j://graph.schema#ResidualAfterToppingCM>,
    <neo4j://graph.schema#locally_acquired>,
    <neo4j://graph.schema#SoilNumber>,
    <neo4j://graph.schema#DrainageCharacteristics_Description>,
    <neo4j://graph.schema#age>,
    <neo4j://graph.schema#case_source>,
    <neo4j://graph.schema#HerbageEstKgDmHa>,
    <neo4j://graph.schema#ENC_ANI_ID>,
    <neo4j://graph.schema#LACTATION>,
    <neo4j://graph.schema#MealIntake>,
    <neo4j://graph.schema#CoverEstimations_PaddockID>,
    <neo4j://graph.schema#PaddockDistanceFromParlour>,
    <neo4j://graph.schema#additional_source>,
    <neo4j://graph.schema#PrincipalSoil>,
    <neo4j://graph.schema#Topping>,
    <neo4j://graph.schema#TotalIntake>,
    <neo4j://graph.schema#PreviousHerbageEstKgDmHa1>,
    <neo4j://graph.schema#CutDate>,
    <neo4j://graph.schema#SilageYieldKgDmHa>,
    <neo4j://graph.schema#additional_info>,
    <neo4j://graph.schema#DensityOfHerbage>,
    <neo4j://graph.schema#case_id>,
    <neo4j://graph.schema#Aspects_Description>,
    <neo4j://graph.schema#GrazeDate>,
    <neo4j://graph.schema#PaddockAltitude>,
    <neo4j://graph.schema#LivestockTYpe>,
    <neo4j://graph.schema#PaddockStatus>,
    <neo4j://graph.schema#report_week>,
    <neo4j://graph.schema#method_note>,
    <neo4j://graph.schema#GrassIntake>,
    <neo4j://graph.schema#PoachingEvent>,
    <neo4j://graph.schema#CoverDate>,
    <neo4j://graph.schema#ParentMaterial>,
    <neo4j://graph.schema#RotationLength>,
    <neo4j://graph.schema#AverageKgLwt>,
    <neo4j://graph.schema#ResidualHeight>,
    <neo4j://graph.schema#PreviousGrazeDate>;
  rdfs:range <neo4j://graph.schema#PbiLactation>,
    <neo4j://graph.schema#PadDockEstimations>,
    <neo4j://graph.schema#IndividualCase>,
    <neo4j://graph.schema#LiveStockNumbers>,
    <neo4j://graph.schema#ManagementDecisions>;
  rdfs:label "belong_to" .
<neo4j://graph.schema#province> a owl:Class;
  rdfs:label "province" .
```

```
<neo4j://graph.schema#PoachingEvent> a owl:Class;
  rdfs:label "PoachingEvent" .
<neo4j://graph.schema#locally_acquired> a owl:Class;
  rdfs:label "locally_acquired" .
<neo4j://graph.schema#Gradients_Description> a owl:Class;
  rdfs:label "Gradients_Description" .
<neo4j://graph.schema#age> a owl:Class;
  rdfs:label "age" .
<neo4j://graph.schema#Topping> a owl:Class;
  rdfs:label "Topping" .
<neo4j://graph.schema#ANONID> a owl:Class;
  rdfs:label "ANONID" .
<neo4j://graph.schema#SoilNumber> a owl:Class;
  rdfs:label "SoilNumber" .
<neo4j://graph.schema#PrincipalSoil> a owl:Class;
  rdfs:label "PrincipalSoil" .
<neo4j://graph.schema#case_source> a owl:Class;
  rdfs:label "case_source" .
<neo4j://graph.schema#health_region> a owl:Class;
  rdfs:label "health_region" .
<neo4j://graph.schema#additional_info> a owl:Class;
  rdfs:label "additional_info" .
<neo4j://graph.schema#PreviousCoverDate> a owl:Class;
  rdfs:label "PreviousCoverDate" .
<neo4j://graph.schema#ResidualAfterCuttingCM> a owl:Class;
  rdfs:label "ResidualAfterCuttingCM" .
<neo4j://graph.schema#method_note> a owl:Class;
  rdfs:label "method_note" .
<neo4j://graph.schema#PaddockAltitude> a owl:Class;
  rdfs:label "PaddockAltitude" .
<neo4j://graph.schema#DrainageCharacteristics_Description> a owl:Class;
  rdfs:label "DrainageCharacteristics_Description" .
<neo4j://graph.schema#GrassIntake> a owl:Class;
  rdfs:label "GrassIntake" .
<neo4j://graph.schema#report_week> a owl:Class;
```

```
  rdfs:label "report_week" .
<neo4j://graph.schema#GrowthRate> a owl:Class;
  rdfs:label "GrowthRate" .
<neo4j://graph.schema#CoverDate> a owl:Class;
  rdfs:label "CoverDate" .
<neo4j://graph.schema#NumberOfStock> a owl:Class;
  rdfs:label "NumberOfStock" .
<neo4j://graph.schema#additional_source> a owl:Class;
  rdfs:label "additional_source" .
<neo4j://graph.schema#LiveStockNumbers> a owl:Class;
  rdfs:label "LiveStockNumbers" .
<neo4j://graph.schema#SilageYieldKgDmHa> a owl:Class;
  rdfs:label "SilageYieldKgDmHa" .
<neo4j://graph.schema#travel_history_country> a owl:Class;
  rdfs:label "travel_history_country" .
<neo4j://graph.schema#PaddockStatusPrevious> a owl:Class;
  rdfs:label "PaddockStatusPrevious" .
<neo4j://graph.schema#ParentMaterial> a owl:Class;
  rdfs:label "ParentMaterial" .
<neo4j://graph.schema#Calving_Date> a owl:Class;
  rdfs:label "Calving_Date" .
<neo4j://graph.schema#case_id> a owl:Class;
  rdfs:label "case_id" .
<neo4j://graph.schema#PreviousGrazeDate> a owl:Class;
  rdfs:label "PreviousGrazeDate" .
<neo4j://graph.schema#defoliatationoption> a owl:Class;
  rdfs:label "defoliatationoption" .
<neo4j://graph.schema#PreviousHerbageEstKgDmHa1> a owl:Class;
  rdfs:label "PreviousHerbageEstKgDmHa1" .
<neo4j://graph.schema#ReseedMethodID> a owl:Class;
  rdfs:label "ReseedMethodID" .
<neo4j://graph.schema#RotationLength> a owl:Class;
  rdfs:label "RotationLength" .
<neo4j://graph.schema#TotalIntake> a owl:Class;
  rdfs:label "TotalIntake" .
```

```
<neo4j://graph.schema#travel_yn> a owl:Class;
  rdfs:label "travel_yn" .
<neo4j://graph.schema#Aspects_Description> a owl:Class;
  rdfs:label "Aspects_Description" .
<neo4j://graph.schema#CoverEstimations_PaddockID> a owl:Class;
  rdfs:label "CoverEstimations_PaddockID" .
<neo4j://graph.schema#AssociatedSoils> a owl:Class;
  rdfs:label "AssociatedSoils" .
<neo4j://graph.schema#PreGrazingYield> a owl:Class;
  rdfs:label "PreGrazingYield" .
<neo4j://graph.schema#SilageIntake> a owl:Class;
  rdfs:label "SilageIntake" .
<neo4j://graph.schema#date_report> a owl:Class;
  rdfs:label "date_report" .
<neo4j://graph.schema#ENC_ANI_ID> a owl:Class;
  rdfs:label "ENC_ANI_ID" .
<neo4j://graph.schema#CutDate> a owl:Class;
  rdfs:label "CutDate" .
<neo4j://graph.schema#provincial_case_id> a owl:Class;
  rdfs:label "provincial_case_id" .
<neo4j://graph.schema#country> a owl:Class;
  rdfs:label "country" .
<neo4j://graph.schema#AverageKgLwt> a owl:Class;
  rdfs:label "AverageKgLwt" .
<neo4j://graph.schema#Height> a owl:Class;
  rdfs:label "Height" .
<neo4j://graph.schema#ResidualAfterGrazingCM> a owl:Class;
  rdfs:label "ResidualAfterGrazingCM" .
<neo4j://graph.schema#PaddockStatus> a owl:Class;
  rdfs:label "PaddockStatus" .
<neo4j://graph.schema#PbiLactation> a owl:Class;
  rdfs:label "PbiLactation" .
<neo4j://graph.schema#ResidualHeight> a owl:Class;
  rdfs:label "ResidualHeight" .
<neo4j://graph.schema#LACTATION> a owl:Class;
```

```
  rdfs:label "LACTATION" .
<neo4j://graph.schema#End_Lactation_Date> a owl:Class;
  rdfs:label "End_Lactation_Date" .
<neo4j://graph.schema#PaddockDistanceFromParlour> a owl:Class;
  rdfs:label "PaddockDistanceFromParlour" .
<neo4j://graph.schema#PaddockArea> a owl:Class;
  rdfs:label "PaddockArea" .
<neo4j://graph.schema#PadDockEstimations> a owl:Class;
  rdfs:label "PadDockEstimations" .
<neo4j://graph.schema#TargetPreGrazingYield> a owl:Class;
  rdfs:label "TargetPreGrazingYield" .
<neo4j://graph.schema#ResidualAfterToppingCM> a owl:Class;
  rdfs:label "ResidualAfterToppingCM" .
<neo4j://graph.schema#MealIntake> a owl:Class;
  rdfs:label "MealIntake" .
<neo4j://graph.schema#ManagementDecisions> a owl:Class;
  rdfs:label "ManagementDecisions" .
<neo4j://graph.schema#ReseedDate> a owl:Class;
  rdfs:label "ReseedDate" .
<neo4j://graph.schema#HerbageEstKgDmHa> a owl:Class;
  rdfs:label "HerbageEstKgDmHa" .
<neo4j://graph.schema#IndividualCase> a owl:Class;
  rdfs:label "IndividualCase" .
```

Appendix J

PbiCow Ontology

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<neo4j://graph.schema#belong_to> a owl:ObjectProperty;
  rdfs:domain <neo4j://graph.schema#EBI>,
    <neo4j://graph.schema#BREED>,
    <neo4j://graph.schema#ENC_ANI_ID>,
    <neo4j://graph.schema#DOB>,
    <neo4j://graph.schema#MILKSUBINDEX>,
    <neo4j://graph.schema#PERIODEND_DATE>,
    <neo4j://graph.schema#ENC_HERD_ID>,
    <neo4j://graph.schema#PERIODBEGIN_DATE>;
  rdfs:range <neo4j://graph.schema#PbiCow>;
  rdfs:label "belong_to" .

<neo4j://graph.schema#PERIODBEGIN_DATE> a owl:Class;
  rdfs:label "PERIODBEGIN_DATE" .
<neo4j://graph.schema#ENC_ANI_ID> a owl:Class;
  rdfs:label "ENC_ANI_ID" .
<neo4j://graph.schema#ENC_HERD_ID> a owl:Class;
  rdfs:label "ENC_HERD_ID" .
<neo4j://graph.schema#PbiCow> a owl:Class;
  rdfs:label "PbiCow" .
```

```
<neo4j://graph.schema#DOB> a owl:Class;
  rdfs:label "DOB" .
<neo4j://graph.schema#PERIODEND_DATE> a owl:Class;
  rdfs:label "PERIODEND_DATE" .
<neo4j://graph.schema#BREED> a owl:Class;
  rdfs:label "BREED" .
<neo4j://graph.schema#EBI> a owl:Class;
  rdfs:label "EBI" .
<neo4j://graph.schema#MILKSUBINDEX> a owl:Class;
  rdfs:label "MILKSUBINDEX" .
```

Appendix K

Yam++ API Response Example

```
<?xml version='1.0' encoding='utf-8' standalone='no'?>
<rdf:RDF xmlns='http://knowledgeweb.semanticweb.org/heterogeneity/alignment#'
         xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
         xmlns:xsd='http://www.w3.org/2001/XMLSchema#'
         xmlns:align='http://knowledgeweb.semanticweb.org/heterogeneity/alignment#'>
<Alignment>
  <xml>yes</xml>
  <level>0</level>
  <type>11</type>
  <onto1>
    <Ontology rdf:about="file:/tmp/yamppls/3TEPAF5EQQOFIB2/source.owl">
      <location>file:/tmp/yamppls/3TEPAF5EQQOFIB2/source.owl</location>
    </Ontology>
  </onto1>
  <onto2>
    <Ontology rdf:about="file:/tmp/yamppls/3TEPAF5EQQOFIB2/target.owl">
      <location>file:/tmp/yamppls/3TEPAF5EQQOFIB2/target.owl</location>
    </Ontology>
  </onto2>
    <map>
    <Cell>
      <entity1 rdf:resource='neo4j://graph.schema#CoverID'/>
      <entity2 rdf:resource='neo4j://graph.schema#CoverID'/>
      <relation>=</relation>
      <measure rdf:datatype='http://www.w3.org/2001/XMLSchema#float'>
      1.0</measure>
    </Cell>
  </map>
  <map>
    <Cell>
      <entity1 rdf:resource='neo4j://graph.schema#CoverDate'/>
      <entity2 rdf:resource='neo4j://graph.schema#CoverDate'/>
      <relation>=</relation>
      <measure rdf:datatype='http://www.w3.org/2001/XMLSchema#float'>
      1.0</measure>
    </Cell>
  </map>
    <map>
    <Cell>
      <entity1 rdf:resource='neo4j://graph.schema#PreviousCoverDate'/>
      <entity2 rdf:resource='neo4j://graph.schema#CoverDate'/>
      <relation>=</relation>
      <measure rdf:datatype='http://www.w3.org/2001/XMLSchema#float'>
      1.0</measure>
    </Cell>
  </map>
</Alignment>
</rdf:RDF>
```