CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Identifying Related Clinical Trials by Using Graph Database and Clustering

Master's thesis in Computer Science and Engineering

Razan Ghzouli

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

Identifying Related Clinical Trials by Using Graph Database and Clustering

Razan Ghzouli

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Identifying Related Clinical Trials by Using Graph Database and Clustering

Razan Ghzouli

© Razan Ghzouli, 2019.

Supervisor: Graham Kemp, Department of Computer Science and Engineering
Advisor: Daniel Gouda, AstraZeneca
Examiner: Alexander Schliep, Department of Computer Science and Engineering

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in $\LaTeX$
Gothenburg, Sweden 2019

Identifying Related Clinical Trials by Using Graph Database and Clustering

Razan Ghzouli
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

In this work we investigate two methods to find similar clinical trials; creating a graph database to migrate clinical trials meta-data from relational database, and clustering clinical trials. We succeeded in identifying similar groups of clinical trials using the clustering method. However, we were not able to evaluate migrating the meta-data into graph database method due to limitations of the chosen software.

Acknowledgements

I would like to extend my gratitude to both my supervisor, Dr.Graham Kemp, and my examiner, Dr.Alexander Schliep. Their supervision, and advice during the thesis helped in forming this work. The preceding work by AstraZeneca team, Kerstin Forsberg and Daniel Goude, on the MetaDataHub and the CDI projects was invaluable for the success of this thesis since their data was used in it.

Razan Ghzouli, Gothenburg, September 2019

Contents

List of Figures

List of Tables

Listings

1 Introduction

Pharmaceutical companies spend large amounts of money and decades on research to develop new drugs. According to The Pharmaceutical Research and Manufacturers of America organization (PhRMA) in 2013, it takes at least 10 years on average and around a cost of $2.6 billion to bring a new approved medicine to patients[1]. These researches generate enormous amounts of data. One type of data generated is acquired and analyzed data from clinical trials. This generated data need to be managed and utilized in an efficient manner to make the most of these investments.

In the last few years, there has been an increased interest across the pharmaceutical industry in clinical trial data sharing and reusing to explore new scientific innovation, and decrease the time to develop and move a new drug to the market [1].

One of the first steps to share and reuse clinical trial data is to pool similar studies that investigated specific compound, lab measurements, or other interesting factor to scientists. Usually clinical research scientists suggest to pool a group of old studies (legacy studies), that they think are similar, and hand them to data scientists and other IT staff to prepare for further investigation. However, pooling suggested similar studies might take months and sometimes fails due to different clinical trial data structures and heterogeneity of the collected data.

We propose to use multiple big data techniques to discover clinical trials which are similar from a data structure point of view, to make data-driven pooling suggestions, that can lead to shorter pooling time and a higher rate of success.

1.1 Problem Description

As mentioned in the introduction, pharmaceutical companies are heading toward reusing their legacy studies data. However, pooling scientists' suggested studies sometimes fail since different clinical trial data structures are not taken into consideration from the beginning.

[1]`https://www.phrma.org/advocacy/research-development/clinical-trials`

The usage of different standards to design clinical trials is one of the reasons for heterogeneity of collected data in clinical trials and their structure. Groups like the Clinical Data Interchange Standards Consortium (CDISC) and the National Cancer Institute Enterprise Vocabulary Services (NCI EVS) have been key drivers of these standards[2].

The way clinical research data is handled and structured over time has dramatically changed. CDISC and NCI EVS have both been working endlessly to draft standards, and drive the pharmaceutical industry to become more standardized [2]. CDISC standards are widely used to design and structure study planning and data collection, tabulation, analysis, and submission to the US Food and Drug Administration (FDA) and other regulatory agencies internationally[3].

The CDISC effort focuses on defining clinical trials datasets and meta-data standards to build formal data models that use a controlled terminology standard [3]. In CDISC standards glossary V12.0 (2018)[4] dataset is defined as "a collection of structured data in a single file" from the same domain, e.g., all data collected regarding demographic information like sex and ethnicity are stored in demographic dataset, abbreviated as DEM dataset or DM dataset.

Eventhough using standards was suppose to make pharmaceutical companies and regulatory organizations work more efficient, many companies have implemented inefficient meta-data management solutions using outdated technologies. They stored each domain dataset in a separate SAS file (collected demographic information in a DM dataset, lab results information in a LAB dataset), similar to Excel files, noticing that one standard might define around hundred of datasets. This resulted in multiple spreadsheets per study, leading to disconnected meta-data repository and difficulty to understand the relationships between collected elements [4]. Figure 1.1 shows an example of multiple studies' data management systems, where each block has sometimes hundred of files that belong to a study, and each file contains hundred to thousands of records of collected data from one domain, e.g., demographic information (DM), lab results information (LAB), device events information (DE), etc. It is obvious how the number of disparate files quickly becomes unmanageable with thousands of studies.

Standards define what each dataset should collect, which are normally represented as columns in a dataset. These columns are called variables. Number of variables per dataset might vary from twenty to hundreds. Figure 1.2 shows some of the demographic dataset variables according to one of CDISC standards' versions in meta-data form. The first column holds dataset name, in this case demographic dataset, abbreviated to DM. The second column shows DM dataset collected vari-

[2]`https://www.cancer.gov/research/resources/terminology/cdisc/`

[3]`https://www.cdisc.org/standards/foundational/`

[4]`https://www.cdisc.org/standards/glossary/`

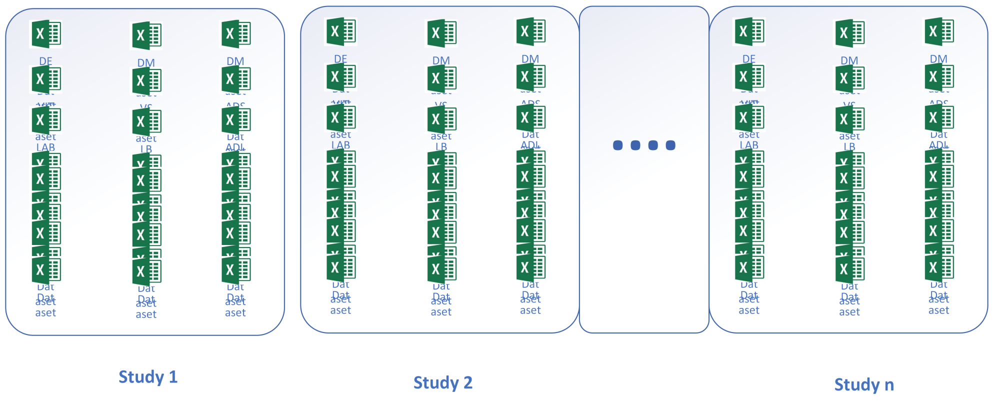
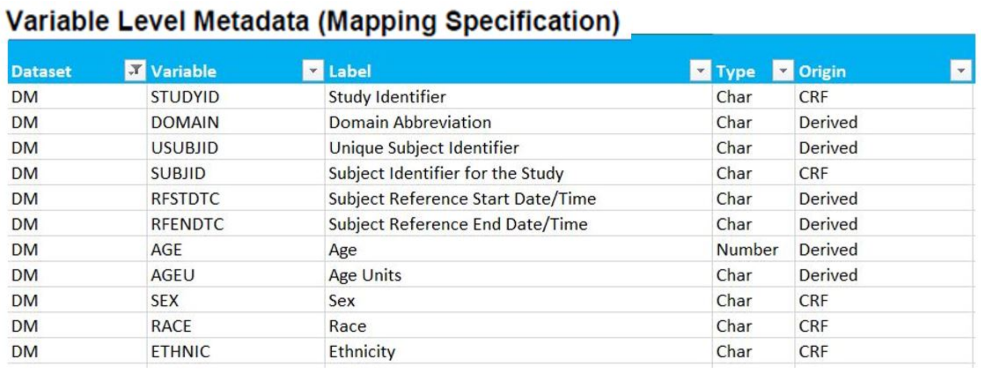


Figure 1.1: This figure shows multiple studies' data management systems. Each block represents a study repository. Each file represents a dataset that collects data with the same characteristics, also called dataset domain, (demographic dataset (DM), laboratory test results dataset (LAB), device events dataset (DE), etc.)

ables abbreviated when possible, e.g., STUDYID is study identifier. The third column shows the full form of the variable abbreviation. All collected variables by DM dataset are related to demographic information, e.g., AGE, SEX, RACE, ETHNIC, etc, in addition to other variables that identify the study (STUDYID) and participating subjects in clinical trial (SUBJID).

Homogeneity of collected variables is one of the factors in finding similar clinical trials to pool, which can be verified by first checking if these trials have collected the same variables. Usually data scientists need to query each domain datasets' variables per study, and compare them with the other studies' datasets variables. This process can be done when pooling a couple of studies, but it is time consuming. However, when scaling this process to ten, a hundred, or even thousands of studies, it becomes impossible to query with the current data management systems. Another factor of complexity and variability of pooled studies with the current data management systems is with each new version of standards, organizations might use different abbreviation to variables of the same meaning in older versions.

This problem was piling up through the years, but with the rising of legacy studies data reusing trend, companies had to face the challenge of finding similar studies across these disparate files. A few years ago AstraZeneca performed two projects called *MetaDataHub* and *Clinical Data Index (CDI)*, where a number of studies were searched, and locations for data and key documents were saved in a database. The CDI project focused on extracting studies' meta-data, and storing them in a relational database (RDB) [5, 6]. The extracted meta-data contained each clinical trial datasets and their variables, resulting in RDB tables similar to figure 1.2 all stored in one data management system.


Variable Level Metadata (Mapping Specification)

Dataset	Variable	Label	Type	Origin
DM	STUDYID	Study Identifier	Char	CRF
DM	DOMAIN	Domain Abbreviation	Char	Derived
DM	USUBJID	Unique Subject Identifier	Char	Derived
DM	SUBJID	Subject Identifier for the Study	Char	CRF
DM	RFSTDTC	Subject Reference Start Date/Time	Char	Derived
DM	RFENDTC	Subject Reference End Date/Time	Char	Derived
DM	AGE	Age	Number	Derived
DM	AGEU	Age Units	Char	Derived
DM	SEX	Sex	Char	CRF
DM	RACE	Race	Char	CRF
DM	ETHNIC	Ethnicity	Char	CRF

Figure 1.2: This figure shows what demographic dataset (DM) should collect according to one of CDISC standards' versions in meta-data form. The first column holds dataset name, in this case demographic dataset, abbreviated to DM. The second column shows DM dataset collected variables abbreviated when possible. Third column shows the full form of the variable abbreviation.

The CDI project was the first step to find similar studies by addressing questions about what variables are captured in the studies. However, with around 20,000 studies and each one of them having hundred of datasets and variables, they ended up with millions of variables to query in RDB.
With data growing in volume, this challenge becomes a big data one, and requires big data technology. We propose using different big data methods to find similar studies. The first step is visualizing studies using a directed graph to understand the relationship between datasets and variables in a clinical trial. The second step is storing studies' extracted meta-data in a graph database instead of a relational database, to discover relations between studies and find similar ones. The final step is clustering studies, to find groups of similar studies in terms of their collected variables.

1.2 Objectives and Research Questions

In this project we aim to make pooling clinical trials data faster and more efficient by identifying similar clinical trials from data structure perspective to have data driven pooling suggestions. In the current data management systems, it is difficult to identify which of many studies can be pooled, and judge quickly how easy or difficult it might be to combine data from a particular set of studies. So, we propose to use a big data technology to store data to make the former easier to accomplish, in addition to applying a clustering algorithm to find groups of similar trials.
By the end of the project, we want to answer the below questions:

- Can studies' meta-data (datasets and variables) visualization using a directed graph help to understand the relation between datasets and variables better in order to build a graph database?
- Can a graph database provide an efficient aggregated view of the different data related to clinical trials to discover relations between studies that the current RDBs cannot provide?
- How many groups of studies can be discovered if we cluster studies according to their collected variables?

1.3 Report structure

The contents of this thesis are organized as follow. After the introduction, chapter 2 introduces some of the concepts and methods used later in the implementation chapter. Chapter 3 explains the implemented methods to answer our research questions. In chapter 4, we present our results. The last chapter 5 provides an overview of the thesis, discusses what has been accomplished, and the lessons learned.

2
Background

In this chapter, we will explain some of the mentioned concepts in the report in addition to the methods used. This will lay the foundation for the methods used in the implementation chapter.

2.1 Clinical Trial

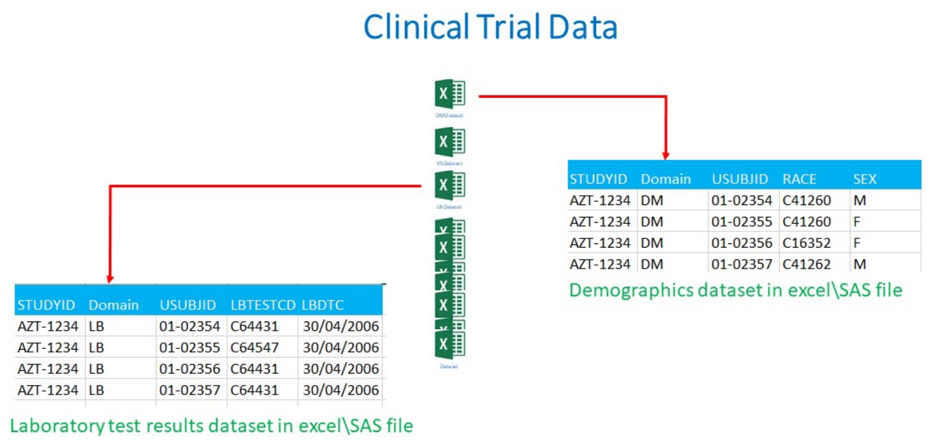
According to CDISC standards glossary V12.0 (2018)[1] clinical trial is defined as *"research investigation involving human subjects that is designed to answer specific questions about the safety and efficacy of a biomedical intervention (drug, treatment, device) or new ways of using a known drug, treatment, or device)"*. In AstraZeneca, there are around 20,000 clinical trials stored in different data management systems due to legacy organizations mergers and acquisitions.

2.1.1 Clinical Trial Data

According to the former CDISC glossary, clinical trial data is defined as *"data collected in the course of a clinical trial"*. As mentioned in section 1.1, CDISC is one of the standardization organizations that issues how clinical trials should be designed and structured and what type of data should be collected.
Usually a standard defines clinical trial data in terms of datasets and variables. CDISC standards glossary V12.0 (2018) defines dataset as *"collection of structured data in a single file"*. Each dataset file contains data from the same domain, e.g., all data collected about untoward medical occurrence in participating subjects in a clinical trial, like any unintended symptom, are stored in adverse event (AE) dataset file. In the previous CDISC glossary, a variable is defined as *"any entity that varies; any attribute, phenomenon, or event that can have different qualitative or quantitative values"*. In a clinical trial dataset file, a variable corresponds to a column. Figure 2.1 displays clinical trial data. The tables show datasets, where each row is a record for a participating subject and each column is

[1]`https://www.cdisc.org/standards/glossary/`

a variable collected. Both datasets have common variables that identify the clinical trial (STUDYID) and participating subjects (USUBJIS), in addition to other variables that belong only to specific dataset (RACE variable in DM dataset and LBTESTCD variable in LB dataset).


Clinical Trial Data
DMDataset
VS Dataset
LB Dataset
Dataset
Demographics dataset in excel\SAS file
Laboratory test results dataset in excel\SAS file

STUDYID	Domain	USUBJID	RACE	SEX
AZT-1234	DM	01-02354	C41260	M
AZT-1234	DM	01-02355	C41260	F
AZT-1234	DM	01-02356	C16352	F
AZT-1234	DM	01-02357	C41262	M

STUDYID	Domain	USUBJID	LBTESTCD	LBDTC
AZT-1234	LB	01-02354	C64431	30/04/2006
AZT-1234	LB	01-02355	C64547	30/04/2006
AZT-1234	LB	01-02356	C64431	30/04/2006
AZT-1234	LB	01-02357	C64431	30/04/2006

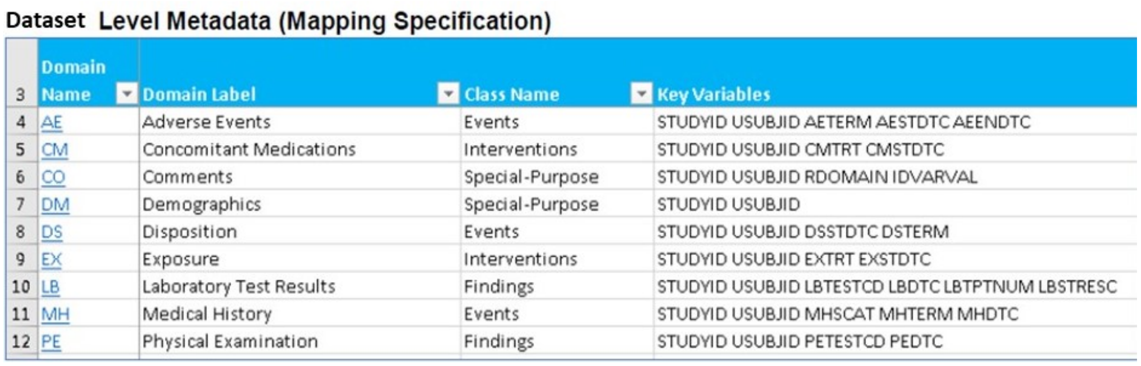
Figure 2.1: This figure shows a clinical trial data. Each file represents a dataset (demographic dataset (DM), laboratory test results dataset (LB), etc.). The tables, with the red arrows pointing at them, are what dataset file usually contains. The right table is DM dataset and the left one is LB dataset, where each column correspond to a collected variable. STUDYID variable column contains the clinical trial unique ID. Domain variable column contains dataset name. USUBJID variable column contains a unique subject identifier for each subject participant. In the DM dataset table, the race variable column contains a code that correspond to each subject race, e.g., C41260 is Asian. Each row in the dataset is a record for participating subjects in the clinical trial.

Meta-data is defined as "data about data", more accurately, meta-data could be defined as *"the information required to contextualize and understand a given data element"* [4]. Within clinical trial, meta-data has multiple levels, but we are interested in only two of them:

- **Dataset level meta-data** describes the properties of a dataset. Figure 2.2 shows dataset level meta-data for a clinical trial, where rows contain datasets available in the clinical trial. Domain name column contains the dataset's abbreviated name, which is usually used to name the dataset file in the data management system.
- **Variable level meta-data** describes variable level properties. Figure 1.2

in section 1.1 shows an example of variable level meta-data for DM dataset. Each row is a variable record that was represented as a column in the clinical trial dataset shown in Figure 2.1.

In the CDI project, data scientists and IT staff worked on indexing clinical trial SAS datasets for studies in different data management systems to extract their meta-data for dataset and variable levels. They stored each level meta-data in different RDB tables. Figure 2.3 represents the transformation process from SAS dataset files into meta-data level RDB.



Dataset Level Metadata (Mapping Specification)

3	Domain Name	Domain Label	Class Name	Key Variables
4	AE	Adverse Events	Events	STUDYID USUBJID AETERM AESTDTC AEENDTC
5	CM	Concomitant Medications	Interventions	STUDYID USUBJID CMTRT CMSTDTC
6	CO	Comments	Special-Purpose	STUDYID USUBJID RDOMAIN IDVARVAL
7	DM	Demographics	Special-Purpose	STUDYID USUBJID
8	DS	Disposition	Events	STUDYID USUBJID DSSTDTC DSTERM
9	EX	Exposure	Interventions	STUDYID USUBJID EXTRT EXSTDTC
10	LB	Laboratory Test Results	Findings	STUDYID USUBJID LBTESTCD LBDTC LBTPTNUM LBSTRESC
11	MH	Medical History	Events	STUDYID USUBJID MHSCAT MHTERM MHDTC
12	PE	Physical Examination	Findings	STUDYID USUBJID PETESTCD PEDTC

Figure 2.2: This figure shows dataset level meta-data for a clinical trial. Each row in the table is a record for a dataset in the clinical trial. The domain label column contains the dataset names, and the domain name column contains the abbreviated form for each name.

2.1.2 Clinical Trial Data Pooling and Reusing

In this section two initiatives are mentioned as examples of multiple efforts to pool clinical trial data for reusing, and how big data technology can help in facilitating it.

As mentioned in [7], there have been multiple initiatives to create pool analysis datasets in the health domain to reuse research data because the right dataset is essential to obtain the right insights in data science. As a start, it is important for data scientists to have a good understanding of the availability of relevant datasets as well as the content and structure of these datasets, before starting the actual data analysis to catalyze scientific innovation.

The most challenging step is understanding the structure of different clinical trials to create pool analysis datasets. One of the initiatives to reuse clinical trial data is The Yale University Open Data Access (YODA) project [8]. The YODA founders highlighted in [8] how the proactive preparation of a catalogue of available trials for a large company with a multitude of marketed products is time and resource intensive. However, it is important to understand the differences and similarities

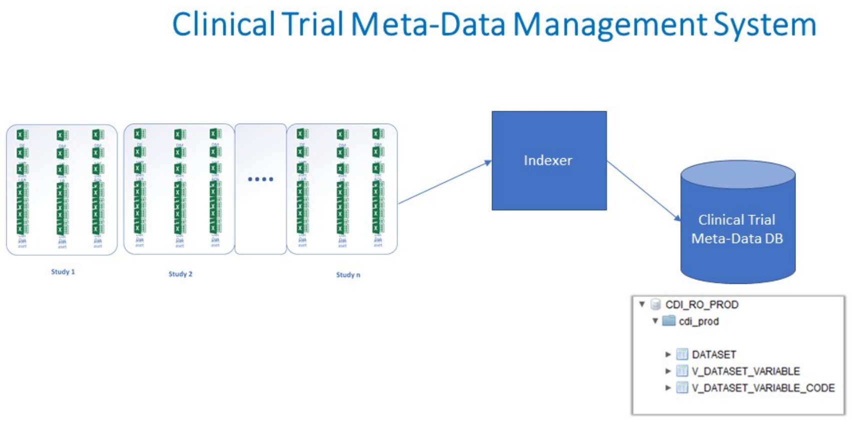


Figure 2.3: This figure shows the transformation process from SAS dataset files into meta-data level RDB for CDI project. There are three level meta-data; dataset, variable, and code

between clinical trials to be able to prepare a catalogue of available trials for further analysis. In another initiative called Strategic Health IT Advanced Research Projects (SHARP) program [9], semantic web technologies are being used on medical data for high-throughput extraction, representation, integration, and querying [10].

2.2 NoSQL Database

For 40 years relational databases have dominated the entire database market, until the 2000's when NoSQL databases started to enter the market. The term NoSQL was first introduced by Carlo Strozzi in 1998 for a relational database that does not use SQL (structured query language) as a query language, and it stood for "no SQL". It is stated on his website that his use of term has nothing to do with the current NoSQL movement[2]. The term was re-introduced in 2009 in an event for "open source distributed, non relational databases" where Google and Amazon talked about their new non-relational databases, Bigtable [11] and Dynamo [12]. Currently, the term NoSQL stands for "not only SQL", to highlight that NoSQL database model does not replace relational database model rather provides an

[2]http://www.strozzi.it/cgi-bin/CSA/tw7/I/en$_US/NoSQL/Home$

alternative when relational database does not scale.
One of the main reasons for NoSQL databases gaining popularity was that big companies, such as Amazon and Google, required databases that can handle massive amounts of data. In addition, NoSQL databases support flexible schema, resulting in easier modification for database model compared to relational database. Furthermore, NoSQL offers four categories of data models depending on application purpose and use cases; key-value databases, document databases, column-oriented databases, and graph databases.
Graph databases will be used in this project, and the following section will explain more about them. The other three types are beyond this project scope.

2.2.1 Graph Databases

Graph databases are a NoSQL database type where data are represented as nodes, the relation between them as edges, and nodes can have properties, resulting in graphs of interconnected key-value pairings. Figure 2.4 is an example of a graph database.

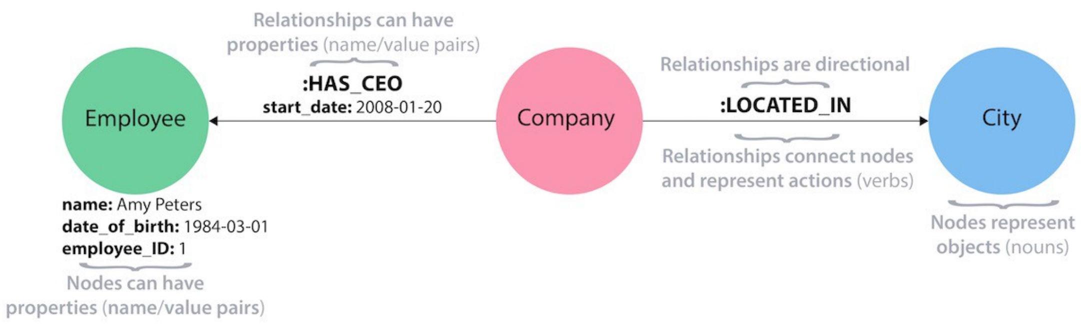

Relationships can have properties (name/value pairs)
:HAS_CEO
start_date: 2008-01-20
Employee
Company
Relationships are directional
:LOCATED_IN
Relationships connect nodes and represent actions (verbs)
City
name: Amy Peters
date_of_birth: 1984-03-01
employee_ID: 1
Nodes can have properties (name/value pairs)
Nodes represent objects (nouns)

Figure 2.4: This figure shows an example of a property graph type. Image source; https://medium.com/@npsinghmrj/what-are-graph-databases-and-different-types-of-graph-databases-369e5040a9d0

Graph databases are specialized in efficient management of heavily linked data, and focus on visual representation of information, which makes them more human-friendly than other NoSQL databases [13]. This makes graph databases a good fit for use cases when we are interested in relationships between data more than the data itself, just like in social networks and recommendation systems. In our use case when querying "which clinical trials investigated product X, have Y dataset

and collected Z variable" in relational databases, it would require multiple joins between different tables, that becomes heavily performance demanding, while in graph databases the traversal between relationships (edges) is efficient. Furthermore, graph databases have the advantage of keeping all information about an entity[3] in a single node, and showing the entity relationships with others by edges, making it more natural for users to understand data.
As mentioned in 2.2.1, NoSQL databases support flexible schema, and in graph databases case no pre-defined schema is needed to start creating the database and storing data, and new information and relations can be continuously added and updated without much effort. This leads to easier adaptation to schema evolution and ability to capture ad-hoc relations continuously [14]. All of the previous mentioned characteristics make graph database a good choice for our use case. One thing to highlight, not all graph databases models support the previous mentioned characteristics.
There are three types of graph databases based on the data model:

- Hypergraphs.
- Property graphs.
- Resource description framework (RDF) stores.

In the following sections, more details will be provided about these types.

2.2.1.1 Property Graphs

A property graph model is widely used in most graph database systems because it is an intuitive and easy to understand. All of the graph database characteristics that were mentioned in 2.2.1 apply to property graph models. It is made up of nodes, relationships, properties, and labels [15]. Figure 2.4 shows an example of a property graph.

- Nodes: nodes represent the information entities in the graph. They can hold any number of attributes called properties in the form of key-value pairs. Nodes can be tagged with one or more labels to group nodes together, and indicate the roles they play within the dataset.
- Relationships: a relationship connects two nodes together to indicate a connection between them (one-to-one relationship). Relationships are represented as directed named arcs, with a start node and an end node. Just like nodes, relationships can have properties, which can provide an additional meta-data for graph algorithms, e.g., relationships can have weight as a property, which can be used in finding short path in a graph.

[3]An information entity is any object in a use case that we want to model and store information about.

2.2.1.2 Hypergraphs

A hypergraph is a graph data model, which contains nodes and edges, and can handle many-to-many relationships between information entities, unlike the other graph databases. This means an edge can have any number of nodes at either end of it, and it is called hyperedge. Figure 2.5 shows an example of a hypergraph.

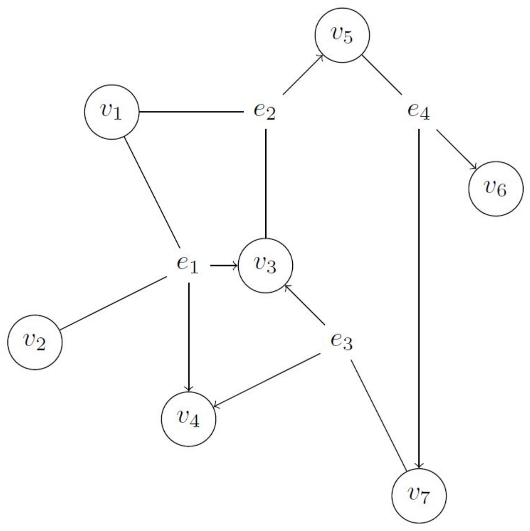


Figure 2.5: This figure shows an example of a hypergraph. It has four hyperedges and seven nodes. Image credits from the author of [16]

This model may require a description for understanding the relationships between nodes, but still it is a simple, generic model. Hypergraphs can be translated into a property graph by transferring the multidimensional hyperedges into more relationships (edges) between nodes to make them one-to-one relationships, but this will increase the cost of storage [15, 16].

2.2.1.3 RDF Stores

Resource description framework (RDF) is a standard data model for supporting resource description, or meta-data, for the Web, and it is part of the Semantic Web movement [17]. RDF stores or triple stores are databases for the storage and retrieval of any type of data expressed in RDF format [18]. RDF express data in a format known as a triple of subject-predicate-object data structure, more will be explained in section 2.3. RDF graph notation is represented by a node for the subject, a node for the object, and an arc for the predicate. Figure 2.6 shows an example of an RDF graph, where *Amsterdam* is a subject, *city* is an object, and the directed arc with a label *Is_a* is a predicate.

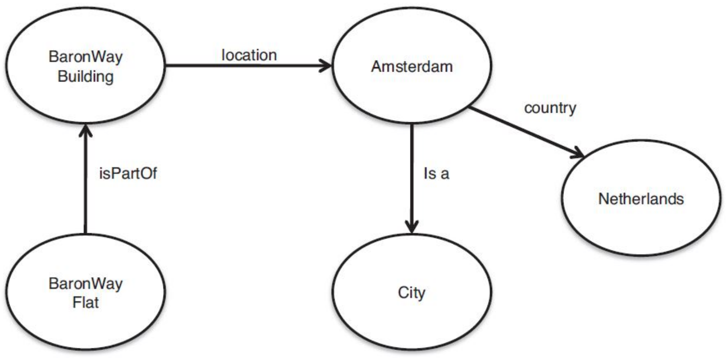


Figure 2.6: This figure shows an example of an RDF graph. Image credits from the authors of [19]

A set of triples expressing information about a use case provide a rich dataset from which to harvest knowledge and infer connections [15]. However, Triple stores are not "native" graph databases, they fall under a general category of graph databases because they do not support all their characteristics. RDF stores do not support index free adjacency, which allows for relationship traversal, resulting in fast querying. On the other hand RDF stores store triples as independent artifacts, which allows them to scale horizontally for storage, but prevents them from rapidly traversing relationships. To perform queries, RDF stores must create connected structures from independent facts, which adds latency for each query [15].

2.3 RDF Data Model

To understand the RDF data model, we need to define four fundamental concepts of RDF data model; resources, properties, statements, and graphs.

2.3.1 Resources

A resource is an object of interest that we want to model. It can be considered as the equivalent of an information entity. In an RDF graph, a resource is the subject node. Every resource has a URI (universal resource identifier), which is unique, to distinguish them from each other. Using the URIs mechanism allows for a global, worldwide unique naming scheme, which means we can identify if any two resources are referring to the same thing or not, and easily merge different RDF data models [19]. RDF data model uses some of the world wide web consortium

(W3C) standards for URI namespace[4], and there are some collaborative community groups, like schema.org[5], who support providing a common set of semantic vocabularies for resources, which can be used as URIs.

2.3.2 Properties

Properties describe relationships between resources, and they are considered as a special kind of resources, which means they are also identified by URIs.

2.3.3 Statements

Statements assert the properties of resources, and in RDF it is formed as subject-predicate-object, or entity-attribute-value [19]. Subject is a resource, or an entity of interest, while object can be either another resource, or a value referred to it as literal[6]. Predicate is object's attribute that describes the relationship with the object.
RDF statements can be expressed in many different formats, called *serializations*. The four common RDF serialisation formats are:

- RDF/XML
- N-Triples
- Turtle
- JSON

Listing 2.1 shows an example of RDF statements written in OWL2 language using RDF/XML syntax. In this example both the subject and object are resources, where *MedicalTrial* is the subject, *Dataset* is the object, and *hasDataset* is the predicate. We used schema.org to identify the resource *MedicalTrial* using "http://schema.org/MedicalTrial" URI.

Listing 2.1: An example of an RDF statements written in OWL2 using RDF/XML syntax

```
<owl:ObjectProperty rdf:about="http://www.semanticweb.org/
    ontologies/2019/AZ-thesis-ontology#hasDataset">
    <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/
        PharmaClinicalTrial#Dataset"/>
</owl:ObjectProperty>
```

[4]A namespace is a set of symbols that are used to organize objects of various kinds, so that these objects may be referred to by name. Namespace ensures that all the identifiers within it must have unique names so that they can be easily identified.

[5]schema.org

[6]Literals are atomic values, for example, numbers, strings, or dates.

2.3.4 Graphs

RDF statements can be represented as a graph. Figure 2.7 shows a graph representation for former RDF statements, where URIs were dropped for readability, which is a common practice. As mentioned in 2.2.1.3 subject and object are represented as nodes, and predicate as arc. The arc is directed from the subject of the statement to the object of the statement, with the label on the arc to the statement's property [19].

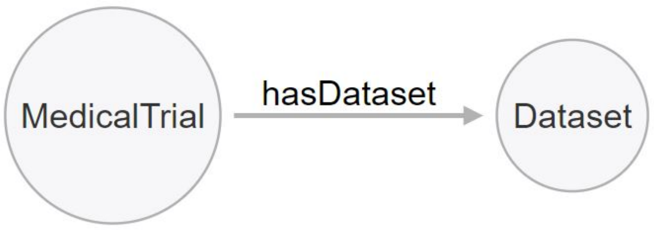


Figure 2.7: This figure shows a graph representation for RDF statements.

2.4 RDF Schema

RDF schema (RDFS) is an ontology language that defines class relations, property relations, and domain and range restrictions for properties. Individual objects that belong to a class are referred to as instances of that class. Authors of [19] book describe RDFS as a mechanism for describing specific domains, since RDF is domain-independent. Figure 2.8 illustrate how RDF is domain-independent, but having its RDFS shed lights of the specific domain of interest. This RDFS describes classes such as *person* and *residential unit*, subclasses like *apartment*, and properties like *residesAt*. It specifies the domain and range for each property to provide some kind of constrains on which entities can be connected together, or what type of value an entity's instance can be assigned to.

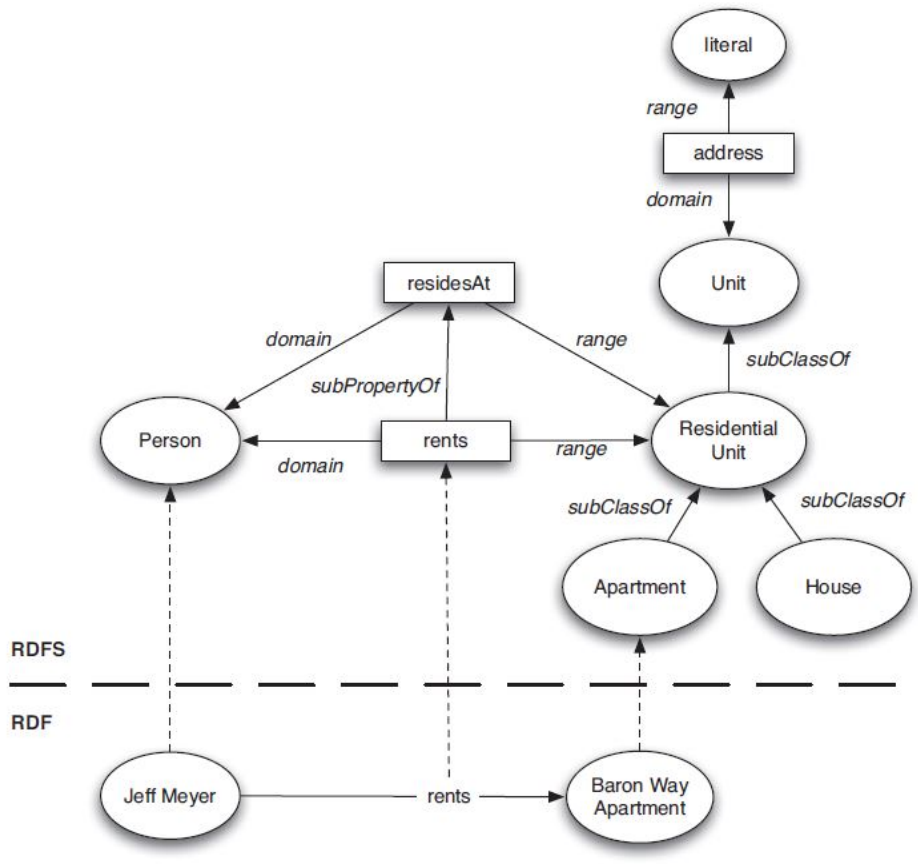


Figure 2.8: This figure shows RDFS and RDF layers. Image credits from the authors of [19].

2.5 OWL2

Web Ontology Language OWL2 is a Semantic Web language designed to represent rich and complex knowledge about resources, and relations between them. OWL2 is mainly used to build ontology[7]. It extends RDF and RDF schema with a number of very expressive language features, but every OWL2 document is a legal RDF document, while RDF document needs some extensions and restrictions before being considered OWL2 document [19].
Just like RDF, OWL2 defines class relations, property relations, and domain and range restrictions for properties, however OWL2 distinguishes two types of properties:

[7]An ontology is an explicit formal specification of the concepts in a domain.

- Object Properties: These properties relate individuals (resources) to other individuals (resources).
- Datatype Properties: These properties relate individuals to literal values of a certain data type.

Listing 2.2 shows examples of an object property and a datatype property taken from [19] book. The object property *rents* connect the class *person* to class *apartment*. The datatype property specify that *age* value should be a positive integer.

Listing 2.2: Examples of OWL2 properties

```
     <!--
     ///////////////////////////////////////////////////////////
     // Object Properties example
     ///////////////////////////////////////////////////////////
      -->
 :rents rdf:type owl:ObjectProperty
     rdfs:domain :Person
     rdfs:range :Apartment
     <!--
     ///////////////////////////////////////////////////////////
     // Datatype Properties example
     ///////////////////////////////////////////////////////////
      -->
 :age rdf:type owl:DatatypeProperty
     rdfs:range xsd:nonNegativeInteger
```

OWL2 language provide a feature called automatic reasoning. Automatic reasoning allows us to check the correctness of the ontology, which can help verify our ontology[19]. Some of the use cases a reasoner checks are:

- Check the consistency of the ontology.
- Check for unintended relations between classes.
- Check if imported instances match defined types.

OWL2 has four standard syntaxes:

- RDF/XML.
- Manchester Syntax.
- OWL/XMLs.
- The Functional Style syntax.

In this project, we will extend our RDF and RDF schema by using OWL2, to make use of the automatic reasoner. RDF/XML format will be used as a syntax format.

2.6 Protégé

Protégé[8] was developed by Stanford center for biomedical informatics research (BMIR), and is described as "a free, open source ontology editor and a knowledge management system, which provides a graphic user interface to define ontologies, and infer new information based on the analysis of an ontology". It supports RDF, RDFS, and OWL2 languages, and can be used to edit schemas. It has a feature for importing instances into classes and properties, and it supports multiple reasoners for automatic reasoning. It is going to be used in this project for creating our schema and importing data into it.

2.7 Clustering

Clustering is an unsupervised machine learning problem, where the goal is to assign a set of objects $X = \{x_1, x_2, ..., x_n\}$ to groups, called clusters $C = \{c_1, c_2, ..., c_n\}$, where objects in a cluster are more similar to each other than to those of other clusters according to a specific similarity measurement. In chapter 3, we are going to use hierarchical clustering to find similar clinical trials according to their collected variables. Thus, in the following sections, we are going to introduce a basic description of used algorithm and similarity measurement.

2.7.1 Hierarchical Clustering

Hierarchical clustering algorithms build a hierarchy of clusters. The output of hierarchical clustering is a dendrogram, which is a tree showing a sequence of nested clusterings [20]. There are two types of hierarchical clustering:

- Agglomerative: agglomerative is a bottom-up approach, where each object in $X = \{x_1, x_2, ..., x_n\}$ is considered as singleton cluster, then pairs of clusters are merged according to similarity measurement, until having at the end one large cluster of all objects. This approach is the most common one in hierarchical clustering.
- Divisive: divisive is a top-down approach, and it is the reverse of agglomerative. It starts by assigning all objects in $X = \{x_1, x_2, ..., x_n\}$ to one big cluster, then recursively splitting it to the most appropriate clusters. The process continues until a stopping criterion is achieved.

Figure 2.9 shows an example of hierarchical clustering output, dendrogram, where the difference between agglomerative and divisive hierarchical approaches is highlighted by arrows. One thing to highlight is through the thesis we will use the term

[8]https://en.wikipedia.org/wiki/Prot%C3%A9g%C3%A9_(software)/

hierarchical clustering without specifying which approach. However, we mean agglomerative hierarchical clustering. Everything we explain and talk about can be projected on divisive approach by doing some slight changes.

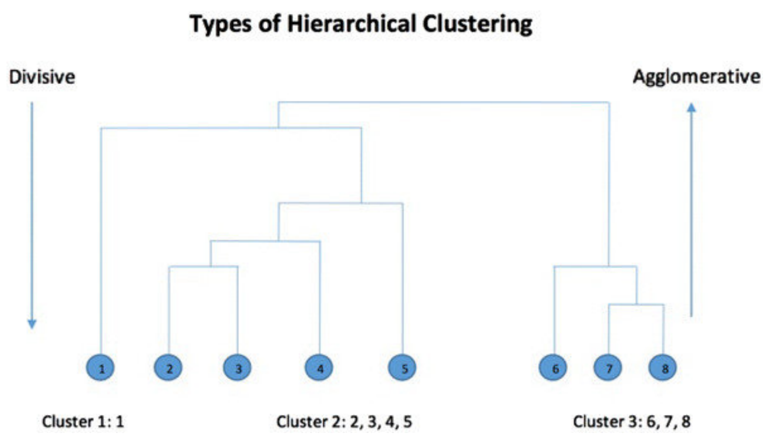


Figure 2.9: Overview of the difference between agglomerative and divisive hierarchical clustering. Image credits from the authors of [21].

To explain hierarchical clustering, we assume we have n objects $X = \{x_1, x_2, ..., x_n\}$, and $m+1$ clusters $C = \{c_0, c_2, ..., c_3\}$, where c_0 is the weak clustering of n objects with distance value 0, and c_m is the strong clustering (the final cluster with all objects and/or clusters). d is a similarity function, e.g., the *Euclidean distance.*

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

We can divide hierarchical clustering algorithm into four steps according to authors of [22]:

Step 1. Start with calculating the distance matrix. Distance matrix values are calculated as follow; given two objects x_i and x_j, their distance matrix value in row i and column j of the matrix is

$$d(x_i, x_j) = a_{i,j}$$

Step 2 Given the clustering c_{i-1} with the distance matrix between each cluster or object and every other. Let a_i be the minimum nonzero entry in the matrix. Merge the pair of objects and/or clusters with distance a_i, to create a new cluster c_i, of value a_i.

Step 3. Recalculate a new similarity function (distance matrix) for new cluster c_i as follow;

– if x_i and x_j are clustered in c_i, and not in c_{i-1}, the distance from the cluster $\{x_i, x_j\}$ to any third object or cluster x_z is

$$d([x_i, x_j], x_z) = min[d(x_i, x_z), d(x_j, x_z)]$$

– if x_i and x_j are objects and/or clusters in c_{i-1}, and not clustered in c_i, their distance $d(x_i, x_j)$ remains the same.

Step 4. Steps 2 and 3 is repeated until finally obtaining the strong clustering c_m, then the clustering is finished.

In the third step of the above explained algorithm, a minimum method was used– a method where the minimum distance is chosen. If we want to use a maximum method, we adjust the distance matrix calculation in step 3 to

$$d([x_i, x_j], x_z) = max[d(x_i, x_z), d(x_j, x_z)]$$

Figure 2.10 illustrates hierarchical clustering algorithm in a simpler way.

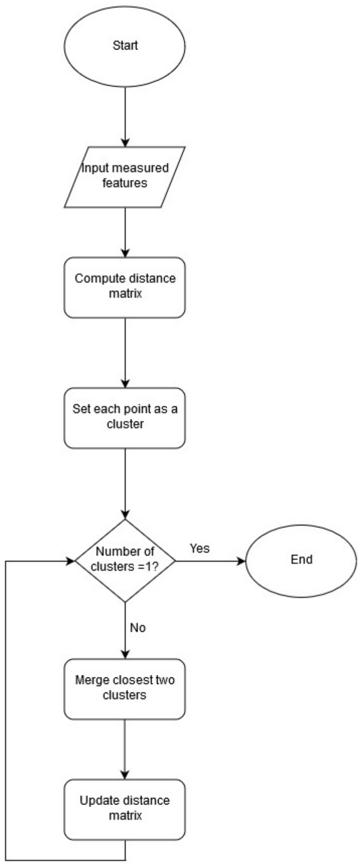


Figure 2.10: This flow chart illustrates agglomerative hierarchical clustering algorithm.

To be able to say how close two clusters are a similarity measurement d need to be calculated. As mentioned in step 2, the decision of merging pairs of clusters is taken on the basis of finding the minimum nonzero entry in the matrix. But this is not the only approach. There are multiple ones, such as:

- **Ward clustering**: minimize the sum of squared error or variance.
- **Single-link clustering**: minimize distance between clusters.
- **Complete-link clustering**: maximize distance between clusters.

2.7.1.1 Ward Variance Minimization Method

Ward's method considers the distance between two clusters as how much the sum of squares (or variance) will increase when we merge them. In hierarchical clustering, the sum of squares starts out at zero, since every point is a cluster, and then grows as we merge clusters. The goal of ward variance minimization method is finding the pair of clusters that leads to minimum increase in total within-cluster sum of squares (variance) after merging [23, 24].
To achieve this goal, ward's method calculates a *merging cost* function, which measures the change in total sum of squares resulting from merging clusters, and chooses the clusters that minimize this cost function. The *merging cost* equation is:

$$\Delta(s,t) = \sum_{i \in s \cup t} \|\vec{x}_i - \vec{\mu}_{s \cup t}\|^2 - \sum_{i \in s} \|\vec{x}_i - \vec{\mu}_s\|^2 - \sum_{i \in t} \|\vec{x}_i - \vec{\mu}_t\|^2 \tag{2.1}$$

$$= \frac{n_s n_t}{n_s + n_t} \|\vec{\mu}_s - \vec{\mu}_t\|^2 \tag{2.2}$$

where s and t are two clusters, μ represents the mean of a cluster (center of cluster), and n represents the number of objects in a cluster. From the equation we can notice that not only the distance between clusters' centers affects the merging decision, but also the number of points in the clusters. Thus, when two pairs of clusters have equal means (same distance away), ward's method prefers to merge the pair with small number of points.
After deciding which clusters to merge, the distance matrix is updated to reflect the distance of the newly formed cluster with the remaining clusters in the forest. In this project, the following updating formula is used:

$$d(U,V) = \sqrt{\frac{|V|+|s|}{T} d(V,s)^2 + \frac{|V|+|t|}{T} d(V,t)^2 - \frac{|V|}{T} d(s,t)^2} \tag{2.3}$$

where U is the newly formed cluster consisting of clusters s and t, and V is one of the remaining clusters in the forest. T is the sum of each cluster cardinality $T = |V| + |s| + |t|$, and d is a distance function, e.g., *Euclidean distance.*

3 Implementation

As mentioned in section 1.2, we want to make pooling studies easier by providing data-driven pooling suggestions. To achieve the former, we will start by visualizing studies' datasets and variables using a directed graph to see how variables are connected to different datasets.
After understanding the data better, we will build a graph database to migrate the studies' meta-data from the current CDI relational database. We start by modeling the database, then implementing the model using Protégé. Finally, we group similar studies according to their variables by applying hierarchical clustering.

3.1 Extracting Studies

Within AstraZeneca there is data available from around 20,000 clinical trials. Using expert knowledge, we identified around 377 out of them that have complete data in the CDI RDB, and are of interest to scientists. Each clinical trial has a unique ID to identify it in the CDI RDB, called AstraZeneca Trial ID (AZT_ID) [6], which was used to query the RDB and extract the data. Python script was written to query the RDB, and the *records* library[1] was used to perform the query. According to its developers, "records is a very simple, but powerful, library for making raw SQL queries to most relational databases"[2]. In addition, it is easy to transform the query output into a `pandas DataFrame`[3] and Comma Separated Values (CSV) file format, which was the main reason we chose it instead of other SQL libraries.
Since we were comparing 377 clinical trials *AZT_ID* against 20,000 studies in the query, and extracting data from millions of records, it was not possible to perform one query to extract the 377 studies' data. We ran into a memory error, due to shortage of memory. After a couple of trial and error attempts, we found that dividing the 377 *AZT_ID* into multiple lists, each one containing ten *AZT_ID*,

[1]`https://github.com/kennethreitz/records/`
[2]`https://github.com/kennethreitz/records/`
[3]https://pandas.pydata.org/

and looping over the lists to pass each list to a query, was the best practice for the available hardware we had. Afterward, the query results were saved into CSV files for each clinical trial. By the end of this step, we had 377 CSV files for desired clinical trial meta-data.

3.2 Data Visualization

Before we build a graph database, we need to understand our data to overcome problems already existing in current RDB, like redundant data. We learned from AstraZeneca data scientists that there are variables that occur in multiple datasets. These are the ones that identify the study and participant subject. However, we were not sure if there are other variables nor their percentage. To answer questions about the data, such as "were there variables reported in multiple datasets?", three representative studies were chosen according to their size to plot.

Directed graph structure was chosen to plot the studies' datasets and variables because it is more intuitive to understand and reflects the natural representation of relation between datasets and variables. The *NetworkX* library was used to plot graphs [25], which is "a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks and graphs"[4]. A Python script was written to create the directed graph (DiGraph) by adding datasets and variables names as nodes, and the relation between them as edges. Directed edges were added from the datasets to their variables, and from the *AZT_ID* for a study to its datasets. Figure 3.1 shows one of the clinical trials data in terms of DiGraph. The blue node represents the clinical trial, and the red nodes represent its datasets. The green nodes are the clinical trial variables. Figure 3.2 shows another clinical trial in terms of DiGraph but has a bigger size than the previous one.

[4]https://networkx.github.io/documentation/stable/index.html/

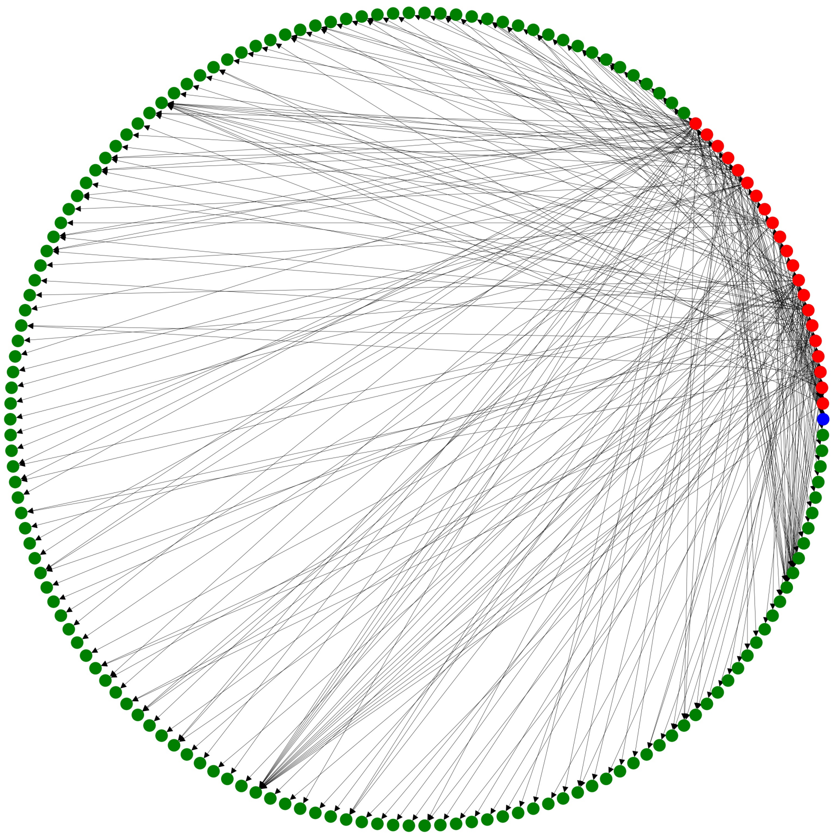


Figure 3.1: This DiGraph shows the meta-data of a clinical trial in form of a directed graph. The blue node represents the clinical trial, the red nodes represent its datasets, and the green nodes are the clinical trial variables. There are two types of edges; from a clinical trial represented by its *AZT_ID* to its datasets, and from a dataset to its variables. The number of edges in this graph is 280, and the number of nodes is 162.

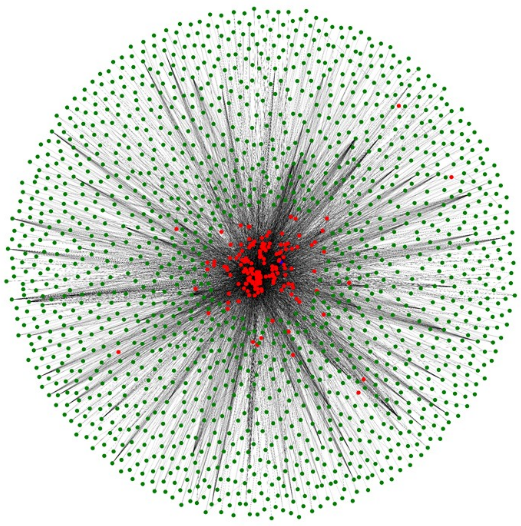


Figure 3.2: This DiGraph shows the meta-data of a clinical trial in form of a directed graph. The blue node represents the clinical trial, the red nodes represent its datasets, and the green nodes are the clinical trial variables. There are two types of edges; from a clinical trial represented by its *AZT_ID* to its datasets, and from a dataset to its variables. The number of edges in this graph is 5426, and the number of nodes is 1730.

3.3 Database Modeling

Currently there is not a well defined methodology for graph database design. It is mostly based on best practices and guidelines related to specific use cases. Some of these guidelines start directly with creation of a graph database without sketching its schema. Even though NOSQL databases are famous for being schemaless, it does not imply the absence of business requirements or the modeling of these requirements. After reviewing multiple guidelines, we decided to choose a combination of two methods [26, 27] since both of them stress the importance of creating a graph database schema. It is proposed to start with a traditional relational database design method, construction of conceptual representation by creating entity–relationship diagram (ERD), then mapping the ERD into a graph database schema.
We performed two steps to create our database model; requirements gathering and creating conceptual model. In the following sections we describe them.

3.3.1 Database Requirements Gathering

In this project, we are migrating data from multiple sources; clinical trial meta-data from the CDI RDB, and other clinical trials information, such as clinical

trial phase, countries it is being conducted in, etc. This information is stored in different Excel files in multiple data systems. With this clinical trial data scattered in different RDB and systems, we need to identify which data to migrate into our new graph database model. We started defining our graph database model by extracting the business requirements that the current data management systems have failed to meet. We decided to use *Jim Gray "20 queries"* proposed approach for tackling data engineering challenges related to large-scale scientific datasets [28].
As mentioned in [28], *Jim Gray* came up with the heuristic rule of *"20 queries"*. On each project he was involved with, he asked for the 20 most important questions the researchers wanted the data management system to answer. He said that five questions are not enough to see a broader pattern, and a hundred questions would result in a shortage of focus.
After discussions with different stakeholders at AstraZeneca, we identified only ten questions that we want our graph database to be able to answer. We decided to start with ten questions instead of 20 as this graph database is a proof of concept. The ten questions are:

- Which studies (for product X) have investigated (LabCode Y)? (This will help identifying potential studies to reuse their datasets to better understand BioMarker related to LabCode)
- Which studies across products (X,Y) were run in the time frame (Date1-Date2)?
- Have we measured (X level) for studies that have been conducted in (X, Y) countries?
- Have studies conducted in (X,Y,Z) countries collected same datasets? (This and the previous question will help in finding if different countries followed same standards)
- How many studies between (Date1-Date2) have collected (X,Y,Z) datasets? (This will help in mapping studies with standards issued at specified duration)
- How many of (the product X) studies have investigated (Y) compound?
- Which studies have (X Datasets) and have investigated (Y levels)?
- How many studies in (X,Y) focus area measured (X level)?
- In which countries has (X,Y,Z) indications been conducted?
- Which studies investigated (X) compound have reported (Y level)?

3.3.2 Conceptual Model

After formulating the above questions and analysing them, we were able to identify the main information entities to sketch preliminary entity–relationship diagram (ERD) representing the conceptual view. The preliminary ERD was discussed with multiple stakeholders and iteratively modified until reaching a final version. It was decided to make the center of the design is the *clinical trial*.

Figure 3.3 shows the final version of ERD, containing three information entities. These entities are:

- **Clinical_trial entity**: it represents clinical trial study and its properties:
  - **AZT_ID**: AstraZeneca unique study ID, and it is going to be used as unique identifier for clinical trial node (URI).
  - **Study_title**.
  - **Acronym**: some studies have a short popular name in the company.
  - **Study_focus_area**: the scientific focus area of the study, e.g., respiratory and inflammation, cardiovascular, oncology, gastrointestinal.
  - **Chemical_compound**: is an official generic and non-proprietary name given to a pharmaceutical drug or an active ingredient, and it provides a unique standard name for each active ingredient.
  - **AZ_product_ID**: the commercial name for the drug investigated or developed in the study.
  - **Countries**: name of countries that the study has been conducted in.
  - **FSI_Date**: first subject in, which indicates the beginning of collecting the study's data.
  - **LSLV_Date**: last subject last visit, which indicates the end of collecting the study's data.
  - **Study_phase**: a stage in the conduct of a clinical trial, which are generally categorized into four (sometimes five) phases.
  - **URI**: study URI where original clinical trial data is stored in one of AstraZeneca's database systems.
  - **Indications**: a sign, symptom, or medical condition that leads to the study, e.g., breast cancer, asthma, depression and anxiety disorders.
- **Dataset entity**: it represents the clinical trial dataset and its properties:
  - **Name**: dataset name, e.g. DM dataset.
  - **File_Path**: the file path where the dataset is stored in one of AstraZeneca's database systems.
- **Variable entity**: it represents the dataset collected variable and its properties:
  - **Name**: variable name, e.g., USUBJID.
  - **Label**: the full format of the abbreviated variable name, e.g., USUBJID label is unique subject identifier.
  - **File_Path**: the file path where the dataset of this variable is stored in one of AstraZeneca's database systems.

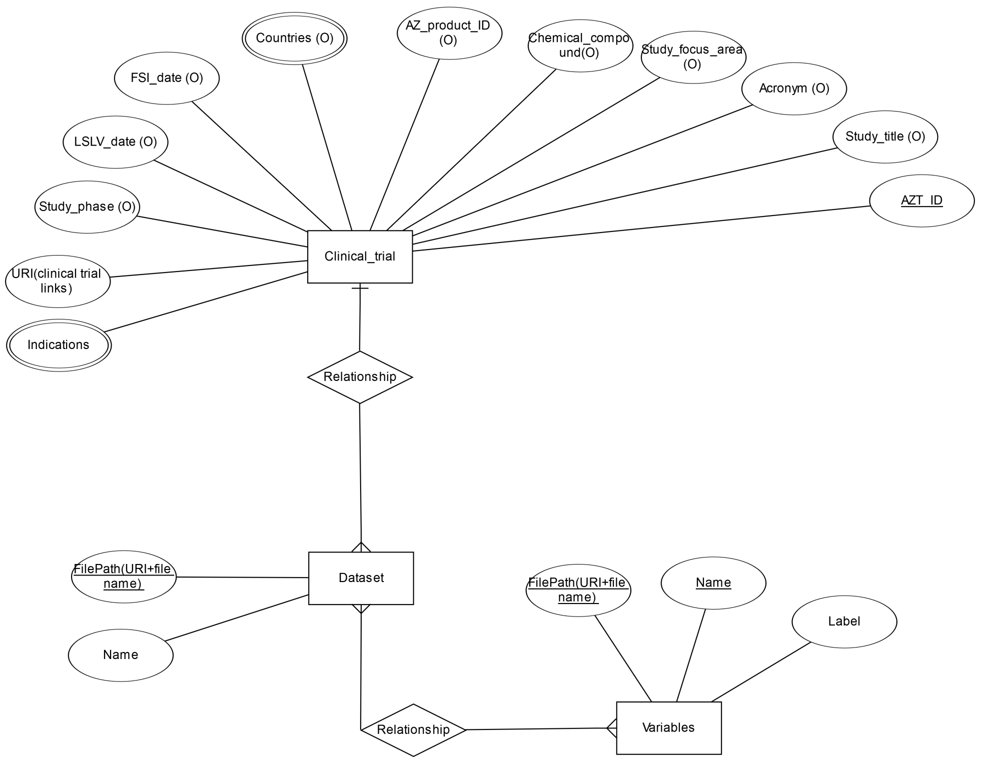


Figure 3.3: This figure shows the Entity–relationship diagram (ERD) that represents the information entities and their properties which answers the proposed modeling questions. Underline properties mean unique values, while properties with (O) means optional, meaning not all clinical trials might have this property. Double circle around a property means it is multi-valued.

3.4 Implementing Graph Database

To implement a graph database, two steps were taken. We started by mapping the former ERD into a graph database schema, then loading the extracted meta-data into the graph database. Resource description framework (RDF) or triple stores was chosen as a graph database type. In the following sections the process of creating graph database will be explained.

3.4.1 Creating Graph Database Schema

Graph databases are schema-less. Nevertheless, providing schema information as additional descriptions of resources in the database is important to describes the relation between the different entities and their properties.
Based on the work of the authors in paper [26], mapping the ERD into RDF schema (RDFS) was done in two steps.

Step 1. Each information entity in the ERD is mapped to a node and the entity's properties become the node's properties.

Step 2. Each relationship between entities is mapped to an edge connecting the respective nodes.

Protégé was used to create the RDFS using OWL2 language, and we chose an XML/RDF format as syntax. Each entity was translated into a class, and their properties were mapped as data properties. Relationships between entities were mapped as object properties. Although there is no strict naming convention for properties, we used the recommended schema by the authors of the Protégé guide [29]. It is recommended that properties' names are prefixed with the word 'has', or the word 'is', and the properties' names start with a lower case letter, have no spaces and have the remaining words capitalized.

Figure 3.4 shows the resulted RDFS. There are three classes; *MedicalTrial*, *Dataset*, and *Variable*. There are two object properties (predicates); *hasDataset*, and *hasVariable*. Predicate *hasDataset* subject is *MedicalTrial* and its object is *Dataset*. Predicate *hasVariable* subject is *Dataset* and its object is *Variable*. For each datatype property, its range type was defined using W3 XML standards[5] e.g., string, date, etc. The whole script for creating the RDFS can be found in appendix A in XML/RDF format.

To standardize used class vocabulary, we used *schema.org* definitions for *MedicalTrial* as this class URI[6]. For now there is no official definition for class *Dataset* and *Variable* on *schema.org*, so we created our own URIs.

[5]https://www.w3.org/TR/xmlschema11-2/
[6]http://schema.org/MedicalTrial/

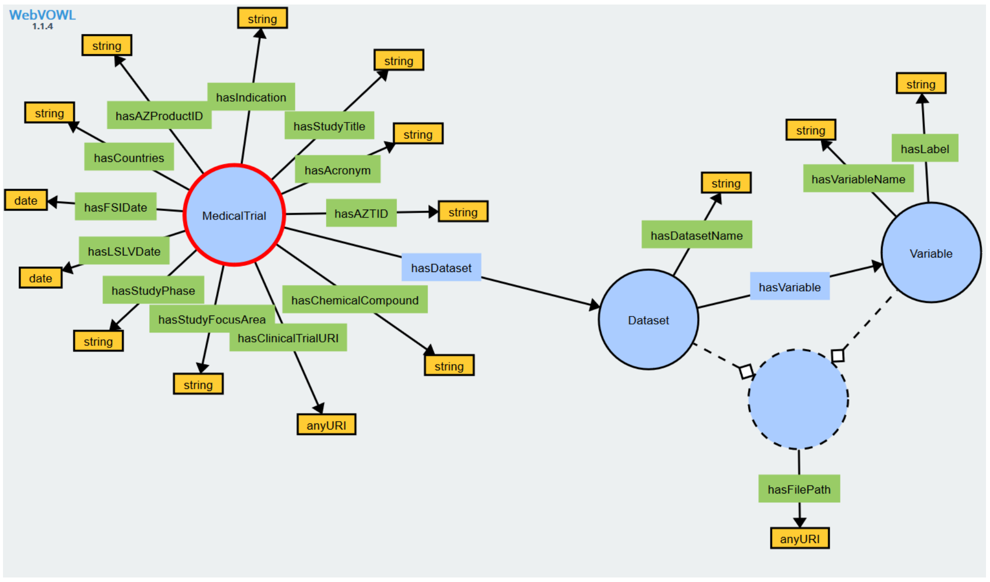


Figure 3.4: This figure shows the result of mapping the ERD into RDFS. Classes are represented as blue circles. Datatype properties are represented as green rectangles. Datatype properties' ranges are yellow rectangles. Object properties are represented as directed arrows from a node to another.

3.4.2 Loading Data into The Graph Database

The *Cellfie* feature[7] in Protégé was used to load the data from CSV file into created RDFS. This feature requires identifying rules to load data from CSV file, by mapping the spreadsheet columns to their respective classes, and datatype properties.

Listing 3.1: Transformation rules to load spreadsheet data into RDFS

```
Individual: @B*
Types: schema:MedicalTrial
Facts: hasAZTID @B*,
       hasClinicalTrialURI @C* ,
       hasAcronym @D* (mm:ProcessIfEmptyLocation ),
       hasTitle @E* (mm:ProcessIfEmptyLocation),
       hasIndications @F* (mm:ProcessIfEmptyLocation),
       hasAZProductID @G* (mm:ProcessIfEmptyLocation),
       hasChemicalCompound @H* (mm:ProcessIfEmptyLocation),
       hasStudyFocusArea @I* (mm:ProcessIfEmptyLocation),
       hasPhase @J* (mm:ProcessIfEmptyLocation),
       hasFSIDate @K* (mm:ProcessIfEmptyLocation xsd:date),
       hasLSLVDate @L* (mm:ProcessIfEmptyLocation xsd:date),
```

[7]https://github.com/protegeproject/cellfie-plugin/wiki/

```
        hasCountries @BH* (mm:ProcessIfEmptyLocation)
```

Listing 3.1 shows an example of extraction rules for loading data into the *MedicalTrial* class from the CSV file. Each data property was mapped to its corresponding spreadsheet column, e.g., *hasTitle* data property is mapped to column E that holds studies' titles. *AZT_ID* was used to identify each instance in the *MedicalTrial* class.

3.5 Clustering Clinical Trial

As mentioned in 1.2, we want to find different groups of clinical trials to make it faster to judge how easy or difficult it is to pool clinical trials. Having clinical trials with similar study structure (same datasets and collected variables) can make it easier to pool studies. In this section, we will cluster clinical trials according to their collected variables.

3.5.1 Creating Feature Vectors

To find similar studies and cluster them, Python script was written where clinical trials' *AZT_ID* and their variables were read using `pandas DataFrame` from the previous extracted CSV files. A dictionary for all studies was created where clinical trials were represented by *AZT_ID* as keys. The list of variables for each clinical trial was considered as a corpus, and saved as a value in the dictionary for its respective *AZT_ID*. Data pre-processing was done by changing all variables' text to lower case.
A vector for each study was created using *CountVectorizer* from *scikit-learn* library[8]. Each vector contained all variables in the clinical trials represented as 1 or 0, depending on the trial. 1 indicates that a variable is present in a dataset, and 0 indicates that a variable is not present in a dataset
To understand this vectorization, we need to define the following process:

- V is the set of all clinical trials variables, where $V = \{v_1, v_2, ..., v_n\}$, and n is the total number of variables in the selected clinical trials (the 377 trials).
- V_A is the set of trial A variables, where $V_A = \{v_1, v_2, ..., v_m\}$.
- F_A is clinical trial A feature vector, where $F_A = \{f_1, f_2, ..., f_n\}$, and n is the total number of variables in the selected clinical trials.
- In this vectorization, the value of f_i is defined as follow:
  - If $v_i \subset V_A$, then its corresponding entry is $f_i = 1$.
  - If $v_i \not\subset V_A$, then its corresponding entry is$f_i = 0$.

[8]https://scikit-learn.org/stable/modules/generated/sklearn.feature$_e$*xtraction.text.CountVectorizer.html/*

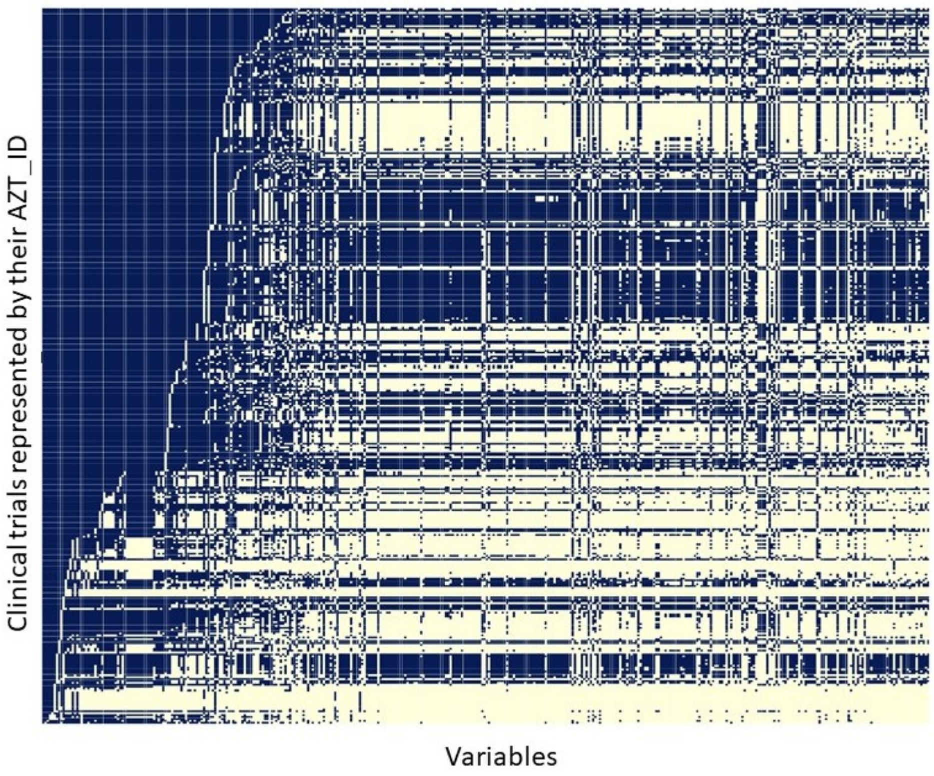


Figure 3.5: This figure shows a heat map for CountVectorizer generated array. Each row in the heat map is a clinical trial's feature vector. Blue represents a variable existence in a clinical trial and yellow represents it does not. Only a percentage of variables are shown in this plot.

After the above process was applied for all clinical trials, an array was generated with variables as columns and clinical trials' feature vectors as rows. The resulting array had 377 rows, corresponding to the number of clinical trials, and 153,288 columns corresponding to the total number of variables in the 377 trials.

The resulting array was represented as a heat map, and only a percentage of the variables were selected to include in the plot. This percentage was chosen as follow; if a variable occurred in $X\%$ of the trials or more, it was chosen to include in the plot, otherwise it was dropped. After a number of iterations to make the heat map informative, we chose that a variable has to occur in 40% of the clinical trials or more. Figure 3.5 shows the resulting heat map, where blue represents 1 and yellow represents 0. Each row in this array can be considered as a feature vector for a clinical trial.

3.5.2 Hierarchical Clustering

To cluster clinical trials, hierarchical clustering with the ward variance minimization algorithm was applied, using the *Linkage* library [9].
The generated feature vectors array was given as an input to the hierarchical clustering algorithm explained in 2.7.1. The output of the algorithm is presented in figure 3.6 as dendrogram. We can see how similar clinical trials are grouped together, and they are illustrated as blocks in the heat map. The *Clustermap* library[10] was used to produce this plot.
To have a better view of clinical trial clusters, the clinical trials dendrogram was extracted and condensed to have a more informative plot. Figure 3.7 shows a condense clinical trials dendrogram. The height of each branch represents the distance at which the clusters merged. The further apart branches represent clusters that are well defined.

[9] https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html/
[10] https://seaborn.pydata.org/generated/seaborn.clustermap.html

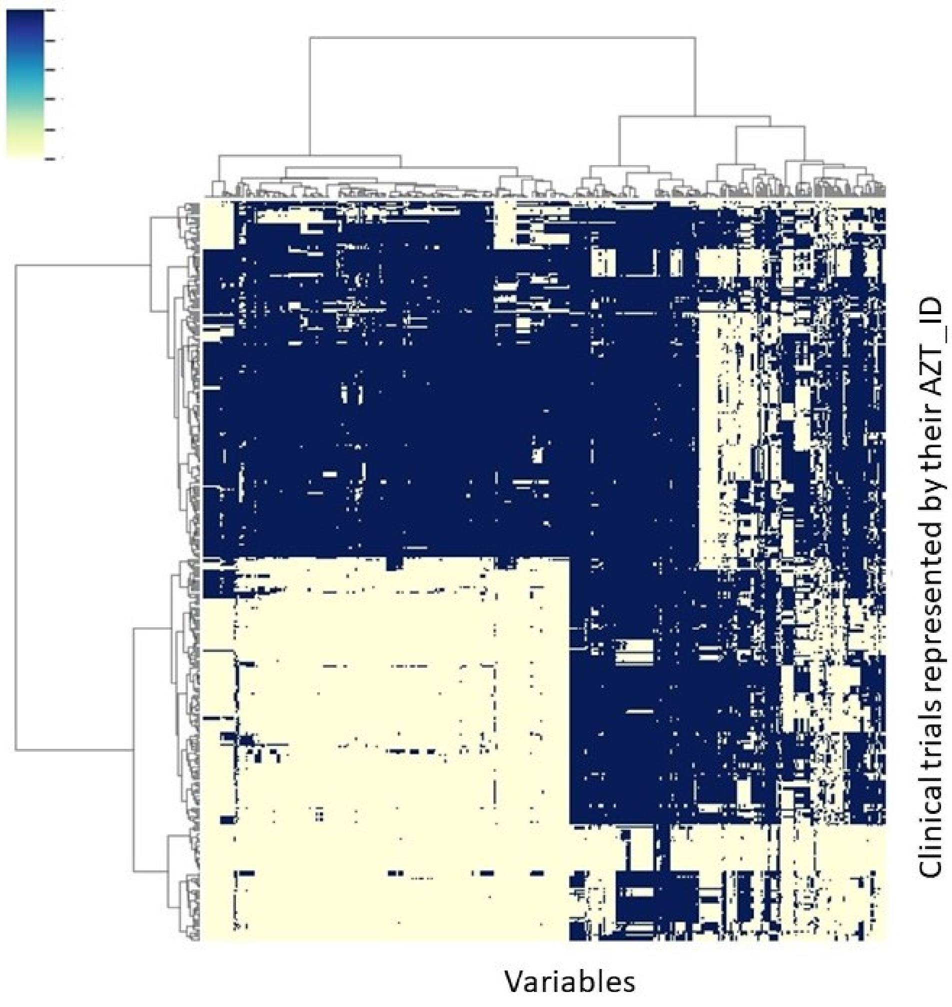


Figure 3.6: .
This figure shows a heat map with double dendrograms for clinical trials' clusters according to their variables.

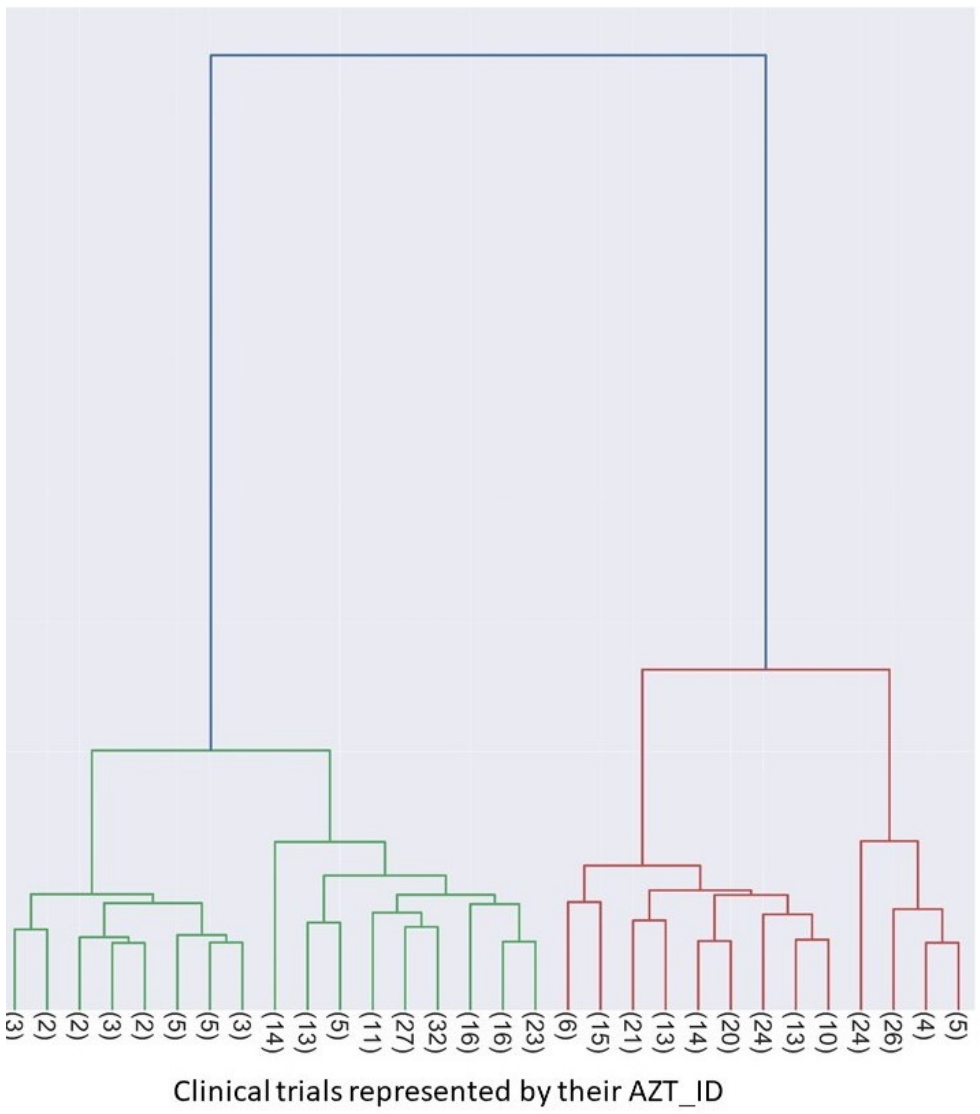


Figure 3.7: This figure shows a condense dendrogram for clinical trials' clusters. Numbers at the end of dendrogram leaves are number of objects in a cluster.

4
Results

The results from chapter 3 will be discussed in the following sections.

4.1 Data Visualization

As mentioned in 3.2, we learned from data experts that there are two variables occurring in almost every dataset. Figure 3.1 shows that more than two variables occur in multiple datasets. This is illustrated as a green node (variable) having multiple incoming edges, represented by multiple arrows, from multiple red nodes (datasets).
To investigate further, we extracted these variables from each study and calculated for each study the percentage of these variables against the total number of variables in a study. By taking the average of all clinical trials, 53% of variables were reported in multiple datasets in a clinical trial.
The most common variables across the 377 clinical trials were extracted. Table 4.1 shows these common variables that appear in multiple datasets across all selected studies. Variables identifying the study and participating subjects can be seen in the table (STUDY_CODE, PATIENT), in addition to other general variables, like HEIGHT and WEIGHT. In general, we can summarize that the variables, occurring in multiple datasets across one study, are either identifying a record in a dataset as unique (STUDY_CODE, PATIENT, VISIT, SUBJECT), or important information about participating subjects and trial (HEIGHT, WEIGHT, SEX, DRUGNAME, DRUG, TYPE).
Having the same variables recorded in multiple datasets is a consequence of the data management systems that is used currently in AstraZeneca. Clinical trial data is stored in disparate datasets' files, resulting in the same variable being reported in multiple datasets' files. Then, when clinical trials meta-data were extracted from SAS datasets' files, the same happened due to relational database structure of tables. As mentioned in section 2.2.1, graph databases have the advantage of keeping all information and relationships about an entity in a single node. Thus, using a graph database for clinical trial data is a good choice, that solves the previous problem of having redundant variables in multiple datasets.

Variable	Variable label
STUDY_CODE	Study code
PATIENT	Randomization Code
SUBJECT	Enrolment Code
VISIT	Visit Number
HEIGHT	Height
WEIGHT	Weight
SEX	Sex
DRUGNAME	Medication CRF Text
DRUG	Name of drug
TYPE	Type of format

Table 4.1: Common variables between 377 selected clinical trials that appears in multiple datasets.

The data visualizing in section 3.2 was informative. It helped in communicating AstraZeneca data scientists' intuitions about the data. However, we noticed that the bigger the study's size, the harder it became to find insights, which can be seen in figure 3.2. Furthermore, we could not extract any additional information, since extracting new meaning from visualizations of clinical trial data would require extensive expert knowledge about the domain.

4.2 Implementing Graph Database

In section 3.4.2, we loaded extracted data into the created RDFS. Protégé offers a querying feature called *DL Query*[1] to query created ontology. Before querying, an automatic reasoner needs to verify the imported instances into classes and properties. We ran the automatic reasoner, which returned in an error of matching data types for dates loaded into *hasFSIDate* and *hasLSLVDate*. The error stated that *hasFSIDate* and *hasLSLVDate* datatype range is date, while the imported instances are string.
After investigating this error, we found that the CSV file, used to store instances, uses UNIX timestamp to represent the raw date/time data, which *Cellfie* detects as a string format. Unfortunately, there was not an intuitive fix for this problem, and this error did not allow us to query the data in Protégé. We started working on Neo4j[2] graph database, but due to shortage of time we decided to skip.
One of the lessons learned is to distinguish between an ontology editor software and

[1]https://protegewiki.stanford.edu/wiki/DLQueryTab
[2]https://neo4j.com/

a graph database software. Eventhough Protégé provides multiple features that resemble features in a database software, it is an ontology editor with limitations. Since the main goal of creating a graph database was finding similar clinical trials to make pooling data faster, and have a higher success rate, we decided to choose a different approach to accomplish this goal. We decided to cluster clinical trials using their collected variables, which is explained in section 3.5.

4.3 Clustering Clinical Trials

In section 3.5, the output of the hierarchical clustering was presented as a dendrogram. To find the number of clusters, an exploratory approach was implemented, where a horizontal line is drawn on the dendrogram. To define the number of clusters, this line needs to transverse the maximum distance vertically without intersecting any cluster. The number of clusters is the number of vertical lines in this dendrogram cut. Figure 4.1 shows the hierarchical clustering dendrogram with the horizontal line to identify the number of clusters. By observing the dendrogram, we were able to identify four clusters. This result can be observed as well in the heat map 3.6, where four blocks can be observed.
To evaluate the resulting clusters, the *cophenetic correlation coefficient* was calculated. This measurement is usually used to evaluate how well the hierarchical structure from the dendrogram represents the actual distances. It correlates the actual pairwise distances of all our objects to those implied by the hierarchical clustering. The closer the value is to 1, the better the clustering preserves the original distances. For our clustering the Cophenetic correlation coefficient is 0.92, which we considered a satisfying result.

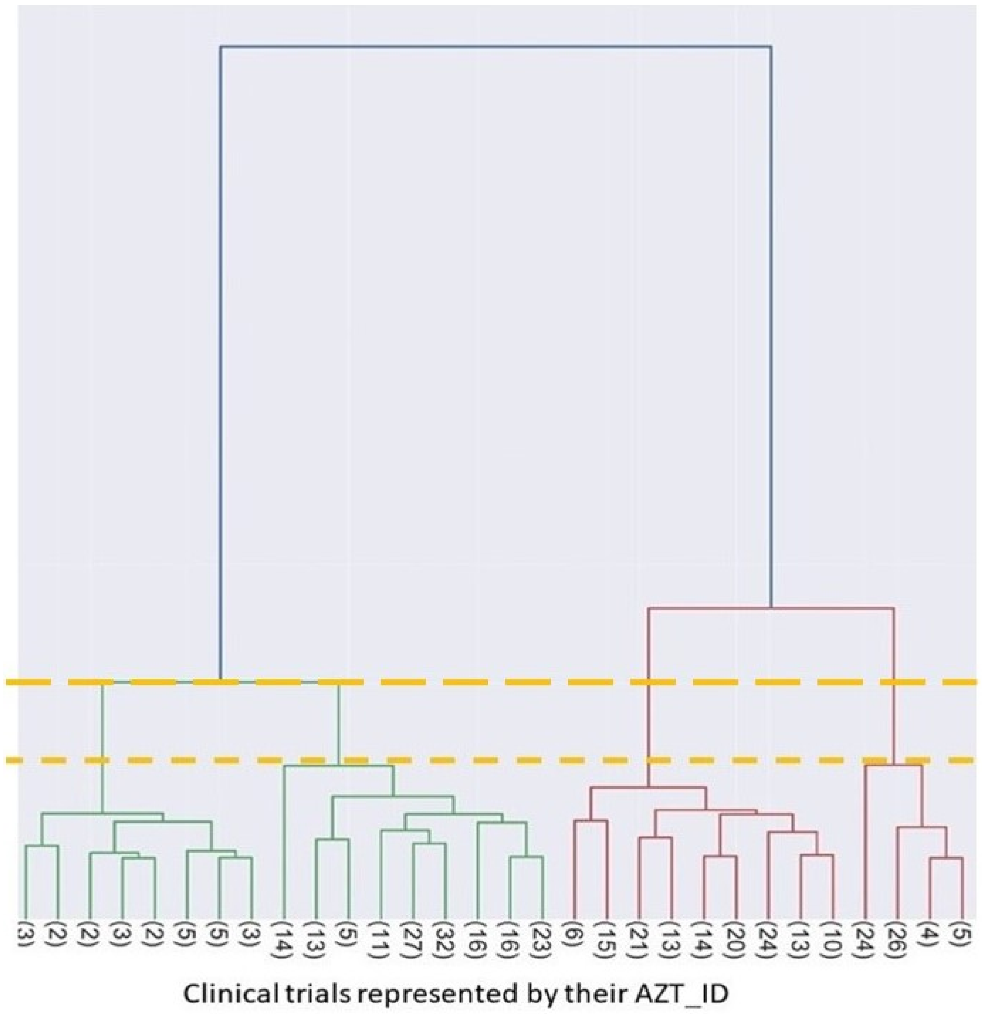


Figure 4.1: This figure shows a condense dendrogram for clinical trials' clusters. The numbers at the end of dendrogram leaves are the numbers of objects in a cluster. The yellow line is the drawn horizontal line to find the number of clusters.

5
Conclusion

The aim of this thesis is to find similar clinical trials. This work can be considered as one of the steps needed to make clinical trials pooling faster and more efficient. To achieve the former, two methods were investigated; migrating clinical trials data into graph database, and clustering clinical trials according to their collected variables.

Our work started by visualizing the clinical trials meta-data using directed graphs. The results of visualization gave a quick inspection of the data. Visualization helped in understanding the data structure and the connection between datasets and variables.

Next, we began the creation of the graph database, using a combination of [26, 27] suggested approaches. Ten database design questions were identified that we wanted our database to be able to answer. These questions were mapped into information entities to create an ERD. The resulting ERD represented the conceptual model of the database. Using the suggested approach by the authors of [26], the resulted ERD was mapped into an RDF schema. The RDFS was created using OWL2 language. The choice of the language resulted in creating an ontology. Afterward, data were loaded into the ontology as instances of classes and properties. To evaluate the created ontology, we wanted to query the ten proposed questions in the design step. Because of the software used to create the ontology (Protégé), an automatic reasoner needed to verify if the imported instances matches the created schema. However, an error was reported by the reasoner. The imported instances into datatype properties did not match the properties' ranges. This error was raised because of the file format storing the instances. Thus, we could not query the ontology using Protégé. Since we could not verify if our created ontology can help in finding similar trials, and due to shortage of time, another method was investigated to find similar clinical trials.

The other method was clustering clinical trials using their collected variables With a hierarchical clustering algorithm. The output of the clustering was represented as a dendrogram, and an exploratory approach was applied to find the number of clusters. By observing the dendrogram, we identified four clusters. Each one of the four clusters contains a group of clinical trials, that are similar in terms of

their collected variables.
As a conclusion we can say, that we succeeded in our goal to find similar clinical trials from data structure aspect, using a clustering method. Unfortunately, we can not say the same about the graph database method. However, the created graph schema and the lessons learned can be used in a future work.

Bibliography

Bibliography

[1] U. Palm. *"The evolving role of clinical trial data sharing"*. https://pharmaphorum.com/views-and-analysis/clinical-trial-data-sharing/. Blog. June 2017.

[2] C. Parry. "Stop Copying CDISC Standards". In: *Proceedings of PhUSE EU Connect 2018 - The Clinical Data Science Conference.* (Frankfurt, Germany). Nov. 2018.

[3] R. L. Richesson and J. Krischer. "Data standards in clinical research: gaps, overlaps, challenges and future directions". In: *Journal of the American Medical Informatics Association* 14.6 (2007), pp. 687–696.

[4] A. Ndikom and L. Wang. "Clinical Metadata – Metadata management with a CDISC mindset". In: *Proceedings of PhUSE Edinburgh 2017 - Digital Innovation in Healthcare.* (Edinburgh, Scotland). Oct. 2017.

[5] E. Nyman, K. Forsberg, and M. Doulis. "The MetaDataHub - Integrating, searching and presenting study descriptive information". In: *Proceedings of PhUSE EU Connect 2018 - The Clinical Data Science Conference.* (Frankfurt, Germany). Nov. 2018.

[6] K. Forsberg and D. Goude. "Study URI". In: *Proceedings of PhUSE EU Connect 2018 - The Clinical Data Science Conference.* (Frankfurt, Germany). Nov. 2018.

[7] J. Shi, M. Zheng, L. Yao, and Y. Ge. "Developing a healthcare dataset information resource (DIR) based on Semantic Web". In: *BMC medical genomics* 11.5 (2018), p. 102.

[8] J. S. Ross, J. Waldstreicher, S. Bamford, J. A. Berlin, K. Childers, N. R. Desai, G. Gamble, C. P. Gross, R. Kuntz, R. Lehman, et al. "Overview and experience of the YODA Project with clinical trial data sharing after 5 years". In: *Scientific data* 5 (2018), p. 180268.

[9] U. D. of Health, H. Services, et al. *"Strategic Health IT Advanced Research Projects (SHARP) Program"*. https://dashboard.healthit.gov/datadashboard/documentation/sharp-project-outputs-documentation.php. (2010).

[10] J. Pathak, R. C. Kiefer, and C. G. Chute. "The linked clinical data project: applying Semantic Web technologies for clinical and translational research using electronic medical records". In: *Proceedings of the 4th International Workshop on Semantic Web Applications and Tools for the Life Sciences.* ACM. (2011), pp. 94–95.

[11] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. "Bigtable: A distributed storage system for structured data". In: *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008), p. 4.

[12] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. “Dynamo: Amazon’s highly available key-value store”. In: *ACM SIGOPS operating systems review*. Vol. 41. 6. ACM. (2007), pp. 205–220.

[13] A. Moniruzzaman and S. A. Hossain. “Nosql database: New era of databases for big data analytics-classification, characteristics and comparison”. In: *arXiv preprint arXiv:1307.0191* (2013).

[14] K. Kaur and R. Rani. “Modeling and querying data in NoSQL databases”. In: *2013 IEEE International Conference on Big Data*. IEEE. (2013), pp. 1–7.

[15] I. Robinson, J. Webber, and E. Eifrem. *"Graph databases"*. O’Reilly Media, Inc., (2013), pp. 5–10, 26–27, 193–210.

[16] Y. Chen et al. “Comparison of Graph Databases and Relational Databases When Handling Large-Scale Social Data”. PhD thesis. University of Saskatchewan, (2016).

[17] S. Powers. *"Practical RDF: solving problems with the resource description framework"*. " O’Reilly Media, Inc.", (2003).

[18] G. E. Modoni, M. Sacco, and W. Terkaj. “A survey of RDF store solutions”. In: *2014 International Conference on Engineering, Technology and Innovation (ICE)*. IEEE. (2014), pp. 1–7.

[19] G. Antoniou, P. Groth, F. v. v. Harmelen, and R. Hoekstra. *"A Semantic Web Primer"*. The MIT Press, (2012), pp. 25–64, 91–125. ISBN: 0262018284, 9780262018289.

[20] R. Dubes and A. K. Jain. “Clustering techniques: the user’s dilemma”. In: *Pattern Recognition* 8.4 (1976), pp. 247–260.

[21] M. Deliu, M. Sperrin, D. Belgrave, and A. Custovic. “Identification of asthma subtypes using clustering methodologies”. In: *Pulmonary therapy* 2.1 (2016), pp. 19–41.

[22] S. C. Johnson. “Hierarchical clustering schemes”. In: *Psychometrika* 32.3 (1967), pp. 241–254.

[23] F. Murtagh and P. Legendre. “Ward’s hierarchical agglomerative clustering method: which algorithms implement Ward’s criterion?” In: *Journal of classification* 31.3 (2014), pp. 274–295.

[24] C. Shalizi. “"Distances between Clustering, Hierarchical Clustering"”. In: *Lecture eight notes from Statistics 36-350: Data Mining course* (Sept. 2009). URL: https://www.stat.cmu.edu/~cshalizi/350/lectures/08/lecture-08.pdf.

[25] A. Hagberg, P. Swart, and D. S Chult. *"Exploring network structure, dynamics, and function using NetworkX"*. Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), (2008).

[26] N. Roy-Hubara, L. Rokach, B. Shapira, and P. Shoval. “Modeling graph database schema”. In: *IT Professional* 19.6 (2017), pp. 34–43.

[27] R. De Virgilio, A. Maccioni, and R. Torlone. “Model-driven design of graph databases”. In: *International Conference on Conceptual Modeling*. Springer. (2014), pp. 172–185.

[28] A. J. Hey, S. Tansley, K. M. Tolle, et al. *"The fourth paradigm: data-intensive scientific discovery"*. Vol. 1. Microsoft research Redmond, WA, (2009).

[29] M. Horridge, S. Brandt, S. Jupp, G. Moulton, A. Rector, R. Stevens, and C. Wroe. “A Practical Guide To Building OWL Ontologies Using The Protégé 4 and CO-ODE Tools Edition 1.3”. In: *The university of Manchester* (2011), p. 26.

A

Appendix 1

Listing A.1: RDFS XML script

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.semanticweb.org/kggt458/ontologies
    /2019/3/untitled-ontology-7"
      xml:base="http://www.semanticweb.org/kggt458/ontologies
          /2019/3/untitled-ontology-7"
      xmlns:owl="http://www.w3.org/2002/07/owl#"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:xml="http://www.w3.org/XML/1998/namespace"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:pharma="http://www.semanticweb.org/pharma/">
    <owl:Ontology rdf:about="http://www.semanticweb.org/ontologies
        /2019/AZ-thesis-ontology"/>

    <!--
    ///////////////////////////////////////////////////////////////
    // Datatypes

    ///////////////////////////////////////////////////////////////
     -->

    <!-- http://www.w3.org/2001/XMLSchema#date -->

    <rdfs:Datatype rdf:about="http://www.w3.org/2001/XMLSchema#
        date"/>


    <!--
    ///////////////////////////////////////////////////////////////
    // Object Properties
    ///////////////////////////////////////////////////////////////
     -->

    <!-- http://www.semanticweb.org/kggt458/ontologies/2019/3/
        untitled-ontology-7#hasDataset -->

    <owl:ObjectProperty rdf:about="http://www.semanticweb.org/
        ontologies/2019/AZ-thesis-ontology#hasDataset">
        <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"
            />
        <rdfs:range rdf:resource="http://www.semanticweb.org/
            PharmaClinicalTrial#Dataset"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/kggt458/ontologies/2019/3/
        untitled-ontology-7#hasVariable -->

    <owl:ObjectProperty rdf:about="http://www.semanticweb.org/
        ontologies/2019/AZ-thesis-ontology#hasVariable">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/
            PharmaClinicalTrial#Dataset"/>
        <rdfs:range rdf:resource="http://www.semanticweb.org/
            PharmaClinicalTrial#Variable"/>
    </owl:ObjectProperty>



    <!--
    ///////////////////////////////////////////////////////////////
    // Data properties

    ///////////////////////////////////////////////////////////////
     -->

    <!-- http://www.semanticweb.org/PharmaClinicalTrial#
        hasAZProductID -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#hasAZProductID">
        <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"
            />
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
           #string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#hasAZTID
        -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#hasAZTID">
        <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"
            />
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#hasAcronym -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/PharmaClinicalTrial#hasAcronym">
        <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#hasChemicalCompound -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/PharmaClinicalTrial#hasChemicalCompound">
        <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#hasClinicalTrialURI -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/PharmaClinicalTrial#hasClinicalTrialURI">
        <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#hasCountries -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/PharmaClinicalTrial#hasCountries">
        <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
            #string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#
        hasDatasetName -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#hasDatasetName">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/
            PharmaClinicalTrial#Dataset"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
            #string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#hasFSIDate
         -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#hasFSIDate">
        <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"
            />
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
            #date"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#
        hasFilePath -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#hasFilePath">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/
            PharmaClinicalTrial#Dataset"/>
        <rdfs:domain rdf:resource="http://www.semanticweb.org/
            PharmaClinicalTrial#Variable"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
            #anyURI"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#
        hasIndication -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#hasIndication">
        <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"
            />
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
            #string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#
        hasLSLVDate -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#hasLSLVDate">
        <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"
            />
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
            #date"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#hasLabel
        -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#hasLabel">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/
            PharmaClinicalTrial#Variable"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
            #string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#
        hasStudyFocusArea -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#hasStudyFocusArea">
        <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"
            />
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
            #string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#
        hasStudyPhase -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#hasStudyPhase">
        <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"
            />
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
            #string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#
        hasStudyTitle -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#hasStudyTitle">
        <rdfs:domain rdf:resource="http://schema.org/MedicalTrial"
            />
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
            #string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#
        hasVariableName -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#hasVariableName">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/
            PharmaClinicalTrial#Variable"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
            #string"/>
    </owl:DatatypeProperty>



    <!--
    ///////////////////////////////////////////////////////
    // Classes
    /////////////////////////////////////////////////////////
     -->

    <!-- http://schema.org/MedicalTrial -->

    <owl:Class rdf:about="http://schema.org/MedicalTrial"/>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#Dataset
        -->

    <owl:Class rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#Dataset">
        <owl:disjointWith rdf:resource="http://www.semanticweb.org
            /PharmaClinicalTrial#Variable"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/PharmaClinicalTrial#Variable
        -->

    <owl:Class rdf:about="http://www.semanticweb.org/
        PharmaClinicalTrial#Variable"/>



    <!--
    ///////////////////////////////////////////////////////////////
    // General axioms
    ///////////////////////////////////////////////////////////////
     -->

    <rdf:Description>
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#
            AllDisjointClasses"/>
        <owl:members rdf:parseType="Collection">
            <rdf:Description rdf:about="http://schema.org/
                MedicalTrial"/>
            <rdf:Description rdf:about="http://www.semanticweb.org
                /PharmaClinicalTrial#Dataset"/>
            <rdf:Description rdf:about="http://www.semanticweb.org
                /PharmaClinicalTrial#Variable"/>
        </owl:members>
    </rdf:Description>
</rdf:RDF>



<!-- Generated by the OWL API (version 4.5.9.2019-02-01T07:24:44Z)
      https://github.com/owlcs/owlapi -->
```