A Semantic Web-Based Digital Library Infrastructure to Facilitate Computational Epidemiology

S.M.Shamimul Hasan

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Madhav V. Marathe, Co-Chair
Edward A. Fox, Co-Chair
Sandeep Gupta
Eli Tilevich
Jonathan P. Leidig

August 24, 2017
Blacksburg, Virginia, 24061

Keywords: Epidemiology, Information Retrieval, Schema, Semantic Web

A Semantic Web-Based Digital Library Infrastructure to Facilitate Computational Epidemiology

(Abstract)

Computational epidemiology generates and utilizes massive amounts of data. There are two primary categories of datasets: reported and synthetic. Reported data include epidemic data published by organizations (e.g., WHO, CDC, other national ministries and departments of health) during and following actual outbreaks, while synthetic datasets are comprised of spatially explicit synthetic populations, labeled social contact networks, multi-cell statistical experiments, and output data generated from the execution of computer simulation experiments. The discipline of computational epidemiology encounters numerous challenges because of the size, volume, and dynamic nature of both types of these datasets.

In this dissertation, we present semantic web-based schemas to organize diverse reported and synthetic computational epidemiology datasets. There are three layers of these schemas: conceptual, logical, and physical. The conceptual layer provides data abstraction by exposing common entities and properties to the end user. The logical layer captures data fragmentation and linking aspects of the datasets. The physical layer covers storage aspects of the datasets. We can create mapping files from the schemas. The schemas are flexible and can grow.

The schemas presented include data linking approaches that can connect large-scale and widely varying epidemic datasets. This linked data leads to an integrated knowledge-base, enabling an epidemiologist to ask complex queries that employ multiple datasets. We demonstrate the utility of our knowledge-base by developing a query bank, which represents typical analyses carried out by an epidemiologist during the course of planning for or responding to an epidemic. By running queries with different data mapping techniques, we demonstrate the performance of various tools. The empirical results show that leveraging semantic web technology is an effective strategy for: reasoning over multiple datasets simultaneously, developing network queries pertinent in an epidemic analysis, and conducting realistic studies undertaken in an epidemic investigation. The performance of queries varies according to the choice of hardware, underlying database, and resource description framework (RDF) engine. We provide application programming interfaces (APIs) on top of our linked datasets, which an epidemiologist can use for information retrieval, without knowing much about underlying datasets. The proposed semantic web-based digital library infrastructure can be highly beneficial for epidemiologists as they work to comprehend disease propagation for timely outbreak detection and efficient disease control activities.

A Semantic Web-Based Digital Library Infrastructure to Facilitate Computational Epidemiology

(General Audience Abstract)

Computational epidemiology generates and utilizes massive amounts of data, and the field faces numerous challenges because of the volume and dynamic nature of the datasets utilized. There are two primary categories of datasets. The first contains epidemic datasets tracking actual outbreaks of disease, which are reported by governments, private companies, and associated parties. The second category is synthetic data created through computer simulation. We present semantic web-based schemas to organize diverse reported and synthetic computational epidemiology datasets. The schemas are flexible in use and scale, and utilize data linking approaches that can connect large-scale and widely varying epidemic datasets. This linked data leads to an integrated knowledge-base, enabling an epidemiologist to ask complex queries that employ multiple datasets. This ability helps epidemiologists better understand disease propagation, for efficient outbreak detection and disease control activities.

Dedication

*First of all, I want to dedicate this dissertation to Almighty **Allah** Subhanahu Wa Ta'ala (Glorious is He, and He is Exalted) for giving me the opportunity to prove myself in this world. Next, I dedicate this dissertation to my lovely and caring parents–**Hasina Sardar** and **Shahjahan Sardar**. Thank you for showing me the most beautiful form of love and teaching me how to overcome all obstacles and challenges.*

Acknowledgments

Alhamdulillah, Allah has given me the opportunity, the strength, the blessing, and the endurance to finish this dissertation. Hence, I praise and thank Allah Subhanahu Wa Ta'ala.

First and foremost, my advisor, Dr. Madhav V. Marathe has supervised me with exceptional patience and care, as well as provided me excellent professional advice during my Ph.D. study. Hence, my deepest gratitude goes to him. Without his depth of technical knowledge, ideas, time, and resources, it would not have been possible for me to shape this dissertation into its final form. Dr. Marathe's constant guidance made my Ph.D. journey a very exciting one. I feel blessed to be his student.

My sincere thanks also goes to another advisor, Dr. Edward A. Fox, for his encouragement, motivation, and guidance. He inspired me to continue learning with an open and positive mind. His comments regarding writing this dissertation and numerous research papers were invaluable and helped me to refine my writing skills.

I am also indebted to Dr. Sandeep Gupta, my Ph.D. committee member. His active participation, constructive criticism, and insightful comments have contributed tremendously to the development and sharpening of our research ideas. He has spent a substantial amount of time guiding my research.

I am thankful to my other Ph.D. committee members–Dr. Eli Tilevich and Dr. Jonathan P. Leidig–for reviewing my dissertation and providing valuable feedback.

Also, I would like to acknowledge Mandy Wilson for her help and encouragement.

None of this would have been possible without the devotion, prayers, and endless love of my family that has always given me strength and courage to endure. No words can express my deepest gratitude to my parents, Hasina Sardar and Shahjahan Sardar, for their patience, trust, and unconditional love. My mother has had a dream of me having a Ph.D.

degree since I was a little boy. It was not only her greatest wish, but she has also sacrificed so much to prepare me for this role. I could never imagine having better parents and role models. My wife, Sabrina Humayun Upoma, has always been there with the right gentle words during the most difficult moments of my Ph.D. study. I can never thank her enough for being such a patient and loving spouse. Also, I am grateful to my beloved brothers, Dr. S.M.Shajedul Hasan and Dr. S.M.Fahim Hasan, who gave me moral support when I needed it the most.

I thank the Network Dynamics and Simulation Science Laboratory (NDSSL) and Department of Computer Science (CS) at Virginia Tech (VT) for the opportunity to study here. My thanks to all faculty members from whom I have taken courses. Their classes have inspired me.

Last but not least, my friends from the Bangladeshi community at Virginia Tech have a special place in my heart. They were my inspiration to work hard and never give up, and I cannot thank them enough for that, as the third Bangladeshi Hokie in my family. Also, my friends from so many countries all over the world made my stay at Virginia Tech a truly rewarding experience.

Contents

1 Introduction **1**

1.1 Computational Epidemiology . . . 1

1.1.1 Datasets Studied in Computational Epidemiology . . . 2

1.1.2 Sample Queries . . . 3

1.1.3 Digital Library Infrastructure Services . . . 3

1.2 Reported Data . . . 4

1.2.1 Generators . . . 4

1.2.2 Current Storage Mechanism . . . 4

1.2.3 Uses . . . 5

1.3 Synthetic Data . . . 6

1.3.1 Generators . . . 6

1.3.2 Current Storage Mechanism . . . 7

1.3.3 Uses . . . 8

1.4 Hypothesis and Research Questions . . . 9

1.4.1 Hypothesis . . . 9

1.4.2 Research Questions . . . 9

1.5 Contribution and Organization of the Dissertation . . . 10

2 Literature Review **12**

2.1 Related Work . . . 12

2.2 Preliminaries . . . 15

2.2.1 Relational Data Model . . . 15

2.2.2 Resource Description Framework (RDF) . . . 16

2.2.3 Web Ontology Language (OWL) . . . 17

2.2.4 D2RQ Mapping Language . . . 18

2.2.5 SPARQL Protocol and RDF Query Language (SPARQL) . . . 20

2.3 CNEW Terminology . . . 20

3 A Framework for Unified Simulation Epidemiological Datasets (FUSED) 22
3.1 FUSED Schema . . . 22
3.1.1 Conceptual Schema . . . 23
3.1.2 Logical Schema . . . 30
3.1.3 Physical Schema . . . 44
3.2 Mapping Across FUSED Schemas . . . 61
3.3 Experimental Study . . . 63
3.3.1 Software Used . . . 64
3.3.2 RDF Graph Creation . . . 64
3.3.3 Complex Queries . . . 64
3.4 Limitations and Scope . . . 72
3.5 Summary . . . 73

4 A Semantic Web-Based Schema for Reported Infectious Disease Epidemic Datasets 74
4.1 Reported Epidemic Dataset Schema . . . 75
4.1.1 Conceptual Schema . . . 75
4.1.2 Logical Schema . . . 77
4.1.3 Physical Schema . . . 82
4.1.4 Macros . . . 83
4.1.5 Instance Creation . . . 91
4.2 Linking Approach . . . 92
4.2.1 Preliminaries . . . 92
4.2.2 Literal Node Creation . . . 93
4.2.3 Link Node Creation . . . 93
4.2.4 Data Organization Hierarchy . . . 94
4.3 Limitations and Scope . . . 97
4.4 Summary . . . 98

5 Computational Epidemiology Data Analytics 99
5.1 The EpiK: Data Federation and Homogeneous Query Execution Framework for Synthetic Data . . . 99
5.1.1 Data Layer . . . 101
5.1.2 Mapping Layer . . . 109
5.1.3 RDF Engine and Services Layer . . . 114
5.2 Query Bank . . . 114
5.2.1 Native Queries Based on 5W1H Approach . . . 114
5.2.2 Benchmark Queries . . . 116

5.3 Empirical Analysis of EpiK . . . 116
5.3.1 Experimental Design . . . 116
5.3.2 Datasets . . . 119
5.3.3 Software and Hardware Used . . . 120
5.3.4 Creating the RDF graphs . . . 120
5.3.5 Results . . . 121
5.4 Analytics with EpiK . . . 122
5.5 Query Performance Evaluation for Reported Data . . . 130
5.5.1 Experimental Setup . . . 130
5.5.2 Development of Mapping and RDF Graph . . . 131
5.5.3 Example Query . . . 136
5.6 Application Programming Interface (API) . . . 148
5.6.1 Schema Construction APIs . . . 148
5.6.2 Query APIs . . . 158
5.6.3 Synthetic and Reported Data Schemas Linking Architecture . . . 171
5.7 Summary . . . 172

6 Conclusions and Future Work **174**
6.1 Conclusions . . . 174
6.2 Future Work . . . 175

Bibliography **177**

List of Figures

3.1 High-level architecture of FUSED . . . 23
3.2 Interaction between three layers of the FUSED schema . . . 25
3.3 Instance-level example of the FUSED schema . . . 26
3.4 FUSED conceptual schema in a graph format . . . 27
3.5 SPARQL graph pattern of the example query . . . 30
3.6 Partial pictorial view of the logical schema. . . . 32
3.7 Physical level data sources . . . 44
3.8 Part 1: Partial pictorial view of the synthetic population component of the physical schema . . . 57
3.9 Part 2: Partial pictorial view of the study related table part of the physical schema . . . 58
3.10 Part 3: Partial pictorial view of the study related files (contact network, configuration, and dendrogram) portion of the physical schema . . . 59

4.1 Worldwide Ebola and Zika affected regions . . . 75
4.2 Sample RDF graph. . . . 92
4.3 Literal node creation example. . . . 93
4.4 Link node creation example. . . . 94
4.5 Data organization hierarchy. . . . 95
4.6 Hierarchy edge creation. . . . 96
4.7 Transitivity edge creation. . . . 97

5.1 EpiK proposed for agent-based epidemic modeling and analytics . . . 100
5.2 Datasets used as a part of agent-based epidemic modeling and analytics framework pipeline . . . 101
5.3 Tuple-based mapping example . . . 112
5.4 Value-based mapping example . . . 112
5.5 Native query performance over a virtual RDF graph . . . 121
5.6 Native query performance over a materialized RDF graph . . . 122

5.7 BSBM-like epidemiology query performance over virtual and materialized RDF graphs. . . . 123
5.8 D2RQ benchmark query performance over virtual and materialized RDF graphs. . . . 124
5.9 Degree distribution information of a ten replicates one cell experiment . . . 126
5.10 Path distribution information . . . 126
5.11 Distribution of the star patterns in the dendrograms . . . 127
5.12 Percentage of the paths (over total) where root infector is a school child for various path lengths . . . 128
5.13 Percentage of star patterns (over total) where the root node is a school-infected child for various star sizes . . . 128
5.14 WHO queries (Q1, Q2, Q3) performance. . . . 139
5.15 WHAT queries (Q4, Q5, Q6) performance. . . . 140
5.16 WHERE queries (Q7, Q8) performance. . . . 142
5.17 WHY and HOW queries (Q9, Q10) performance. . . . 143
5.18 HOW queries (Q11, Q12, Q13, and Q14) performance. . . . 144
5.19 HOW and WHERE queries (Q15, Q16, and Q17) performances. . . . 146
5.20 WHEN, WHERE, HOW queries (Q18, Q19, and Q20) performance. . . . 147
5.21 API level structures to answer complex queries (find how many Ebola deaths occurred in a country region for a particular time interval). . . . 171
5.22 Synthetic and reported data schemas linking architecture. . . . 172

List of Tables

3.1 Chicago mid-flu experiment datasets. . . . 63
3.2 Chicago mid-flu experiment RDF graph information. . . . 64
3.3 Query 1 Result. Replicate number, and, starting from school, the number of people infected (maximum path length) before the end of the epidemic. . . . 68
3.4 Query 2 (find disease transmission between various demographic groups) results. . . . 71

5.1 Computational networked epidemiology datasets available at NDSSL. . . . 102
5.2 Household related attributes. . . . 103
5.3 Person related attributes. . . . 104
5.4 Activity related attributes. . . . 104
5.5 Activity location-related attributes. . . . 105
5.6 Home location-related attributes. . . . 106
5.7 Experiment related attributes. . . . 106
5.8 Analysis related attributes. . . . 107
5.9 Disease model related attributes. . . . 107
5.10 Interventions related attributes. . . . 108
5.11 Experiment configuration and output file related attributes. . . . 108
5.12 Example of the RDF triples generated by tuple-based and value-based mappings, a sample query, its SPARQL implementations, and results. . . . 110
5.13 Partial D2RQ mapping implementation of the value-based mapping (join part only). . . . 111
5.14 5W1H questions mapped to RDF data property vocabularies. . . . 115
5.15 SPARQL implementation of a sample 5W1H query. . . . 115
5.16 Queries collected interviewing various epidemiologists (Native Queries). . . . 117
5.17 BSBM benchmark queries, and their corresponding epidemiology queries, 5W1H question categories, and benchmark query properties. . . . 118
5.18 D2RQ benchmark queries used in the experimentation. . . . 119
5.19 Tuple-based and value-based mapping files, creation times, sizes, and triple counts. . . . 119

5.20 RDF graphs created using tuple-based and value-based approaches. 119
5.21 Epidemic experiments carried out using Seattle synthetic data. 120
5.22 SPARQL implementation of query "How many preschool children's information is used in Seattle strong flu experiment?". 122
5.23 Degree count information. 127
5.24 Path count statistics. 129
5.25 Star pattern count information. 129
5.26 Summary of degree, star, and path counts. 129
5.27 Sample Ebola reported datasets used in prototype development. 132
5.28 Default and literal node mapping files creation times, their sizes, and triple counts. 133
5.29 RDF graph information created using our approach. Here we present RDF graph sizes, triple counts, RDF graph generation time, link generation time, and Virtuoso triplestores loading times. 134
5.30 Example queries collected by interviewing various epidemiologists. 135
5.31 5W1H questions mapped to the RDF data property vocabularies. 137
5.32 SPARQL implementation of a sample 5W1H query. 137
5.33 Q1 Results (partial): Relief material supplier at Zwedru, Liberia. 138
5.34 Q2 Results (partial): Ebola parameter information reporter. 138
5.35 Q3 Results: ETU constructor in Macenta, Guinea. 138
5.36 Q4 Results: Ebola care facility names (ECF) at Sierra Leone. 139
5.37 Q5 Results: Time series data sources. 140
5.38 Q6 Results: Ebola categories of infection. 140
5.39 Q7 Results (partial): Country names, where Ebola outbreak occurred before 2014. 141
5.40 Q8 Results (partial): Burial team locations in Sierra Leone. 141
5.41 Q9 Results: ETU constructed in Liberia on December 2, 2014. 142
5.42 Q10 Results: Counties in Liberia with the highest reported number of cases. 143
5.43 Q11 Result: ETUs with an onsite lab in Sierra Leone. 143
5.44 Q12 Result: Number of individuals who responded in Senegal. 143
5.45 Q13 Result: Number of distinct datasets available in the RDF graph related to Liberia. 144
5.46 Q14 Results (partial): Possible line list representation regarding Montserrado County of Liberia. 144
5.47 Q15 Results (partial): Number of healthcare facilities and their locations in Liberia. Here Admin1 means first-level administrative country subdivisions. 145
5.48 Q16 Results (partial): Community care centers location in Liberia with no beds in use. 145

5.49 Q17 Results (partial): Dates with no case count information and their location. 145
5.50 Q18 Results: First operational ETU in Liberia. 146
5.51 Q19 Results (partial): Ebola cases in various counties of Liberia, the location where the most relief items were sent, relief item descriptions, and shipment dates. 147
5.52 Query 20 Results (partial): Ebola deaths in various counties of Liberia and availability of healthcare facility. 147
5.53 Reported and synthetic data API levels. 170

Chapter 1

Introduction

1.1 Computational Epidemiology

A primary objective of computational epidemiology is the development of computer models and informatics tools for analyzing the spatiotemporal spread of diseases. It is a data-driven life science discipline that consumes and produces a massive amount of data. We focus in this work on computational epidemiology using a networked representation of the underlying population, as opposed to mean field [1] or compartmental models [2]. Computer simulations play an important role in understanding the properties of diseases, especially because, unlike in the physical sciences, real world experiments are usually not possible. Studies are conducted through simulation, and therefore require information on population structure, agent behavior, and infectious disease transmission, and a model of the given disease. Here, we mainly focus on infectious diseases.

The computational epidemiology domain faces many challenges regarding the size, dynamic nature, and volume of data. This heterogeneous content includes metadata, text, tables, spreadsheets, experimental descriptions, and large result files. There is no accepted framework that allows unified access to such content. Services such as curation, provenance management, and querying multiple domain-specific datasets are unavailable. This makes collaborative research and policy analysis challenging and often distracts from the important goal of planning and responding to pandemics. The diversity of models, data sources, data representations, and modalities collected, used, and modified motivate the development of a digital library infrastructure to support computational epidemiology.

1.1.1 Datasets Studied in Computational Epidemiology

Briefly, the types of datasets used in computational epidemiology are:

Reported Data: This describes continuously gathered health data for a given population. Epidemic reported data can come from the Centers for Disease Control and Prevention, the World Health Organization, enteric virus surveillance systems, and systems like Google Flu Trends. Also, individuals frequently interact, share, and exchange health information through social media, which can help researchers to identify and analyze epidemics.

Synthetic Population Data: This describes synthetic individuals and their activities.

Synthetic Contact Data: This describes activities performed by synthetic individuals, including travel to locations of work, home, and daily chores. As part of the activities, people come in close contact with other individuals. The contact data captures this information and is central to the study of epidemics.

Model Data: This describes how a given virus affects a population and is characterized by attributes like susceptibility, infectivity, symptom severity, prodrome severity, and incapacitation [3]. For some diseases weather is correlated with the spread of diseases.

Experiment Setup Data: Computational epidemiology experiments are divided into multiple cells. Each cell characterizes one set of quantified simulation conditions, which may have many replicates. Numerous intervention actions are applied to epidemic simulations for the purpose of determining the most effective mitigation strategy. Social distance, school closure, work closure, vaccinations, and antivirals are some of the possible intervention types. The goal of the intervention is to interrupt disease transmission from one person to another person. These may be implemented in the model at a predetermined time (e.g., on day 1), or when a certain condition is met (e.g., when 1% of school-aged children are diagnosed). Interventions may be targeted to specific demographic groups, or subpopulations, such as school-aged children or workers in critical jobs (e.g., first responders).

Experiment Output Data: This describes the spread of disease through a population. It contains three parts [4]. First is a list of infected people with information about the time and duration of incubation and infection, and results of interactions with health care providers, such as whether they were diagnosed. Second is a set of trees with the initially infected persons as the roots showing the path of infection through the population. This is

referred to as a dendrogram. Third is a list of interventions and the time at which they were implemented. Multiple dendrograms across replicates and cells are used to produce tabular data (epidemic curves, intervention rankings, interaction effects, etc.).

We can divide computational epidemiology data into two categories: first, **reported data** published by governments, private firms, and other organizations which focuses on the infection and spread of diseases; and second, **synthetic data** generated from computer simulation.

1.1.2 Sample Queries

Epidemiologists want to execute complex queries on computational epidemiology datasets. We interviewed numerous epidemiologists and asked them about the types of queries they desire to run on heterogeneous and fragmented computational epidemiology datasets. To give the reader a quick sense, in the following we provide two sample queries. However, many more possibilities exist. Currently it is a laborious manual or semi-automated process to conduct these queries because of data heterogeneity, fragmentation, and lack of linking among the datasets.

- **Query on Reported Data:** Find the number of Ebola cases and deaths that occurred in Monrovia, Liberia, and nearby healthcare facilities. Also, find healthcare facilities' opening dates and cases or death counts before and after the healthcare facilities' opening dates.
- **Query on Synthetic Data:** Find an infected person in a school at the beginning of the epidemic. Starting from that person, discover the transitive disease transmission path length before the epidemic dies out.

1.1.3 Digital Library Infrastructure Services

Epidemiologists need a data organization scheme for reported and synthetic computational epidemiology datasets. The scheme should provide services as follows:

- a graph-based extensible schema service capable of organizing heterogeneous and fragmented datasets and handling data growth and change,
- a data linking service for heterogeneous and fragmented computational networked epidemiology workflow (CNEW) datasets,
- a data abstraction service for the end user that hides redundancy existing in the physical level storage without changing it,

- a mapping file creation service from the schema that links conceptual and physical data, and supports virtual and materialized graph database creation, and
- a complex query execution service over numerous datasets.

1.2 Reported Data

1.2.1 Generators

The world has been shaken by several devastating infectious disease epidemics in recent years. The most serious ones have been Ebola and Zika. The Ebola outbreak in 2014 killed over 10,000 people and was one of the most violent Ebola outbreaks in the history of humanity [5, 6, 7]. The Ebola outbreak has currently tapered off to a large extent, but the Zika outbreak is still progressing. The World Health Organization has announced the Zika virus epidemic as an international public health crisis [8]. Many experts believe these epidemics are wake-up calls, and outbreaks even more intense than these will appear soon [9]. At the moment of this writing, there is no comprehensive commercial anti-viral treatment or vaccine available for epidemics like Ebola or Zika. The only way to stop an outbreak is to break the chains of infection [10]. Government officials, international health organizations, and the public are publishing a huge volume of epidemic data on the web (see The Humanitarian Data Exchange website: https://data.hdx.rwlabs.org/ebola). We call these reported epidemic datasets.

1.2.2 Current Storage Mechanism

The problem is that most of the online datasets are in disparate forms, such as comma separated value (CSV), Excel sheets, text, etc. Although humans can understand these datasets through manual processing, it requires customized software development for a computer program to use and analyze these datasets. Researchers are overwhelmed with the diverse epidemic datasets, which are in separate locations, and in using different formats and schemas. The problem is exacerbated by the fact that most of the datasets are not linked. This is somewhat understandable. Linking is complicated because of redundancy, heterogeneity, and lack of a common standard data storing scheme. However, linking is required because complicated epidemiological queries cannot be answered based on data from one domain.

1.2.3 Uses

There is a tremendous push among researchers to analyze these datasets and understand disease propagation. Combining these datasets, gathered by front line outbreak responders, has the potential to yield valuable insights regarding the epidemics and how to handle them.

An illustrative scenario

Timely delivery of medical supplies is vital for treating Ebola patients and protecting health care workers. It would follow that more medical supplies should be routed to areas with a higher disease burden than others. An epidemiologist is curious to see if the routing of medical supplies reached areas that most needed them. To start, she collects datasets for the populations of the outbreak countries with case count time series and a dataset on the routing of medical supplies. She matches the cumulative case counts to the dates that medical supplies were shipped. She finds a few instances in which medical supplies did not go to the areas that most needed them, and she is curious to find out what might be the reason for the disparities. She adds additional datasets on Ebola treatment unit (ETU) and clinic location to her analysis. These fail to account for the discrepancy in medical supply distribution, because the routing of medical supplies does not correlate with the number of ETUs or clinics nearby. She then hypothesizes that, perhaps, there were logistical difficulties in shipping the medical supplies that would account for their distribution. To estimate shipping conditions during the outbreak, she looks at the number of major inbound roads per county, and weather forecasts and the United Nations' monthly collected food price data for good measure. All of these datasets she matches by date and location to her original datasets. She finds that the accessibility of the counties correlates with the amount of medical supplies shipped to them.

One approach to answer these scenarios would be to browse documents manually, create ad-hoc links, and derive the requisite dataset. An alternate approach would be to develop domain-specific software that performs these tasks automatically. The manual approach is both cumbersome and time-consuming. It is also highly likely that researchers may miss some important information. The software development, however, would limit data sharing and reuse, a key requirement for this work. In both cases, the epidemiologist must collect data from multiple sources and determine which content is available in what collection. It is a tedious process and makes information retrieval difficult. Hence, a lack of formal semantics and query limitations prohibits a deep dive into these datasets. There is a great need to organize and link epidemic reported datasets to satisfy epidemiologists' information needs.

1.3 Synthetic Data

1.3.1 Generators

Recent advances in data science and high-performance computing allow epidemiologists to study the spread of infectious diseases over realistic social contact networks using computer simulations. These simulations are used as a part of increasingly complex workflows for situation assessment, ranking and evaluating pharmaceutical and non-pharmaceutical control strategies, evaluating the economic and social impact of specific response methods, etc. Epidemiologists must work through diverse datasets that vary in size, frequency, logical format, and temporal and spatial granularity to derive impactful insights about a given outbreak. Here, we will focus on workflows developed for networked epidemiology (referred to as CNEW workflows) wherein social interactions are explicitly represented within the framework to capture the geographical and social variability inherent in populations and disease dynamics. We refer to datasets that are generated and used as a part of such workflows as CNEW datasets.

Broadly, a typical CNEW workflow is comprised of: (*i*) generating a synthetic population for a region of interest, (*ii*) describing a counterfactual experiment that captures a specific planning or response scenario as a statistical experiment comprising several cells and replicates, (*iii*) executing the design on a parallel computer, and finally (*iv*) analyzing the results of the computer simulations. See [11, 12] for more detail.

An analyst creates a CNEW workflow typically using a computational tool. Here, we use a Network Dynamics and Simulation Science Laboratory (NDSSL) web application, called the Synthetic Information Based Epidemiological Laboratory (SIBEL - formerly called ISIS) [13, 14], to illustrate typical workflows; similar tools are now being designed by other research groups [15, 16, 17, 18, 19]. We are using SIBEL because it is one of the pre-eminent tools that manages CNEW datasets and is more advanced than others. For example, VecNet, EpiC, and Gleamviz are incapable of modeling individual level detail, FRED has a small set of interventions, and the FRED mobile app only supports modeling to a county level [13]. SIBEL overcomes these limitations. Using SIBEL, an analyst can set up an individual level experimental design with many interventions and access the results of the simulations with a set of basic analyses for each of the runs. The plots and epidemic curves provide detailed information about the epidemic. A key aspect of SIBEL is its simplicity – public health analysts can be trained to use SIBEL in a few hours [13]. A typical workflow consists of choosing: (*i*) *a region of interest, an associated synthetic population, and a dynamic relational social contact network;* (*ii*) *a within-host disease progression model;* (*iii*) *a set of initial conditions; and* (*iv*) *a set of interventions.* Each intervention requires additional detail, such as compliance level, sub-populations to which interventions are applied, and under what conditions the

interventions take place. The experiment contains one or more sweeping parameter(s) across a user-specified range of values. Experiments in a typical study would be divided into one or more cell(s), each of which may have multiple interventions. Vaccination, social distancing, and closing schools are a few possibilities. To arrive at a stochastically robust conclusion, each experiment is run multiple times – these are called replicates, following statistical science literature. The information associated with each of the above components is complex and diverse. For example, the social contact network is best stored as RDF triples, the synthetic population is stored in a relational form, and interventions can be stored as flat files [20]. The output data generated is equally diverse. A basic data structure is a dendrogram: it tells when a person was infected, how long the person stays infected, and who infected the person. It's best seen as a directed acyclic graph (DAG) [21].

The system can carry out studies for any region in the US and several regions in Africa, India, and Europe. We can also study multiple diseases using SIBEL. These include influenza-like illnesses, smallpox, Ebola, etc. The set of interventions available to the end users has similarly increased. This growth presents new challenges. First, the data reside on various machines and in differing formats (e.g., configuration files, relational databases, Excel sheets, graph databases) and does not follow common data storage and representation standards. Second, datasets are often generated hastily to facilitate the processing need of a newly appearing epidemic (e.g., Ebola outbreak 2014-2015, ongoing Zika outbreak, etc.) simulation study. Many model parameters of a new epidemic are unknown initially. Also, epidemiologists are continuously gathering a vast amount of new data from the field that can be used for modeling and forecasting [22]. This means datasets are unstable; there are redundancies, and fragmention, which lead to normalization problems. Datasets often suffer from the referential integrity problem. Further, data dictionaries (describing content, format, structure, and relation of the datasets), and codebooks (documents to store the codes) are not available.

1.3.2 Current Storage Mechanism

CNEW data (in SIBEL) is distributed across multiple storage locations. Synthetic population, contact network, study related files, and a study database are four major components of a CNEW dataset. The synthetic population is stored in two high-performance computing (HPC) clusters, in files, and in two relational databases as tables. Contact networks, disease models, and study configuration information are stored in an HPC cluster as files. Study-related metadata are stored in the study database.

1.3.3 Uses

Synthetic datasets are used in agent-based discrete time step simulation. Each individual in the population is treated as an agent, and the duration is divided into a fixed number of time steps. In each time step, the simulation computes the health of each individual, based on the disease model and the activities recorded by the synthetic generator.

An illustrative scenario

An epidemiologist prepares for a likely influenza epidemic. Using SIBEL, the epidemiologist can choose the desired geographical region (for example, the city of Boston in Massachusetts, USA). The epidemiologist sets the values of disease parameters, such as susceptibility, infectivity, etc. based upon the information available for the current disease. Since information about all the parameters may not be exactly known at the time (if the disease is in its early stages), she may choose a range of values for each parameter as discussed earlier. This would generate a set of experiments (cells and replicates), which would then run on various high-performance computing resources. The results of the experiments are displayed to the user through a set of plots and aggregate statistics. Suppose, upon completion of the study, the epidemiologist wants to drill down further to understand the space-time distribution of infected individuals. The epidemiologist also wants to understand the locations where most infections seem to occur, and to study the household structure of the infected individuals. The epidemiologist believes young children in certain locations are spreaders; if true, then this would help the epidemiologist design better-targeted interventions. The epidemiologist is seeking to develop an understanding of the super-spreaders in this scenario, but unfortunately SIBEL does not provide this ability. The information exists, but it is fragmented across various databases. It requires linking social network data, output data, and spatial data relevant to the study.

To retrieve the results for the queries mentioned above, a user with access to the servers must determine the cell information of the experiment from the SIBEL user interface. Then a software developer must read the cell directory from the high-performance computing (HPC) resource. The cell directory contains the configuration file, which provides the output file location in the HPC resource. The output file provides infector and infectee information and their identification (ID) numbers. The configuration file contains information about the contact network used in the study. The contact network is derived from the synthetic population data stored in a relational database. By querying that database with the sick person IDs, the software developer can discover the activities performed by each sick person in the days prior to infection, the disease transmission between various demographic groups, and their location information. In the current SIBEL system, answering queries such as this is a manual, tedious, and laborious process.

1.4 Hypothesis and Research Questions

The primary objective of this research is to develop a digital library infrastructure. This infrastructure can be developed by using the 5S digital library framework with semantic web-based concepts [23]. We perform our research based on the following innovative hypothesis and research questions, which are novel in the area of computational epidemiology.

1.4.1 Hypothesis

A flexible, efficient, and effective infrastructure for computational epidemiology can be constructed by integrating semantic web and digital library frameworks.

- Heterogeneous, fragmented, and dynamic datasets found in computational epidemiology can be better managed through semantic web-based linked schema for both synthetic and reported data.
- Semantic web-based schemas are highly beneficial to epidemiologists, because these permit both the data abstraction that hides redundancy existing in the physical level storage, and mapping file creation that links conceptual data to physical data (which is helpful in creating a virtual and materialized graph database).
- Semantic web-based schemas support efficient complex query execution, spanning various mixes of heterogeneous and fragmented computational epidemiology datasets.

1.4.2 Research Questions

The above hypothesis leads to the following research questions:

- How can we better develop a semantic web-based schema linking heterogeneous and fragmented synthetic computational epidemiology datasets?
- How can we better develop a semantic web-based schema linking heterogeneous and fragmented reported computational epidemiology datasets?
- How can we perform data abstraction and management effectively with semantic web-based schemas?
- How can we create accurate mapping files from these schemas?
- How do we accomplish efficient complex query execution over various computational epidemiology datasets?

1.5 Contribution and Organization of the Dissertation

We present the organization of this dissertation, with contributions, below.

In **Chapter 2**, we present a review of the literature and introduce several preliminary semantic web concepts. Also, it explains the precise interpretation utilized in this work.

In **Chapter 3**, we propose a schema, called FUSED, to organize logically the diverse datasets that make up typical epidemic analysis workflows. FUSED focuses specifically on workflows developed for networked epidemiology (referred to as computational networked epidemiology workflow (CNEW)) wherein social interactions are explicitly represented within the framework to capture the geographical and social variability inherent in populations and disease dynamics. We refer to datasets used and generated as a part of workflows, as CNEW datasets. FUSED represents CNEW datasets and their relationships using RDF/OWL-based syntax. It has a three-tier architecture, comprised of a conceptual layer, a logical layer, and a physical layer. The conceptual layer schema presents a high level view of CNEW datasets and hides lower level details about the data, including representation, layout, and storage. This exposes common data properties that fit the mental models of the users. The logical layer schema captures data fragmentation and linking aspects of the CNEW datasets. The physical layer schema covers the storage aspects of the CNEW datasets. FUSED provides extensible, easy to use, linked, logical access to CNEW datasets. This includes: (*i*) spatially explicit synthetic populations; (*ii*) spatially explicit dynamic, labeled social contact networks; (*iii*) experimental designs used to generate multi-cell statistical experiments; and (*iv*) output data generated by executing computer experiments. FUSED provides data abstraction, mapping file creation, and advanced query execution facilities. It links the output data that differs in the storage technology involved and is distributed physically across different machines in the network.

In **Chapter 4**, we present a schema that formally illustrates epidemic reported dataset entities and their semantic relations. The schema is comprehensive enough to represent various heterogeneous reported epidemic datasets. It can provide a one-stop data service for reported epidemic datasets, and is designed to handle current and future outbreaks. The proposed schema comes with a novel data linking approach which can connect the large-scale and broad variety of epidemic datasets reported by government officials, epidemiologists, responding organizations, and the public. This linking facility combines data into a single global data space, allowing the discovery and integration of new data sources at runtime. The proposed schema has sophisticated query execution ability to help epidemiologists understand disease propagation for efficient outbreak detection and

control activities. Our proposed schema provides flexibility, as it is developed using the RDF data model, i.e., a graph-based data model that can grow. As proof of the concept, we implemented a prototype of our schema with several Ebola and Zika datasets. The corpus is published as Linked Open Data (LOD), an openly available web of data in machine readable form. Our dataset is accessible on the web for browsing, navigation, and faceted search.

In **Chapter 5**, we propose a synthetic and reported data analytics approach. The proposed approach offers various advantages: (*i*) linking large amounts of data sources, (*ii*) linking a variety of data sources, (*iii*) access to programmatic data, and (*iv*) query efficiency. For synthetic data, we study two classes of RDF mappings in this context: value-based and tuple-based. The difference in these two mappings lies in how relationships among entities are expressed and materialized. Value-based mapping preserves child-parent entities relationships, while tuple-based mapping ignores them. We investigate the trade-offs of the mappings regarding schema clarity (including ease in understanding the data organization and ability to frame queries), storage cost, and efficiency of answering queries. We develop a benchmark to evaluate different implementations for a homogeneous query execution framework, which builds a federation of the disparate datasets and exposes them (using the mappings) as a unified whole. The benchmark is a collection of SPARQL queries over the federated SIBEL data. Collectively, they capture access patterns and workloads that appear frequently in the domain. The queries represent real-time epidemiology queries. We use this benchmark to evaluate two kinds of query execution frameworks: one relational engine-based and the second RDF engine-based. Relational engines are space efficient and can scale to very large datasets while RDF engines are query efficient and useful when a real-time response is mandatory. The choice regarding style of representation (value-vs. tuple-based) combined with the implementation of the execution platform (relational engine-based vs. RDF engine-based) provides us with four possible design points for the proposed platform. Each design point has different trade-offs. This work evaluates all the four possible implementations regarding the benchmark.

In **Chapter 6**, we discuss conclusions and possible future extensions of our research.

Chapter 2

Literature Review

2.1 Related Work

Our research builds on prior work published on scientific data management, digital libraries (DLs), and semantic web tools. Scientific data management is a long-standing research field, driven by the fact that scientific software often produces a variety of specialized data. Early work in this field introduced the concept of a process-oriented scientific database model (POSDBM) [24], which includes two data objects, and relation types. This work is extended with key semantic elements of scientific experimentation by Pratt et al. [25]. Examples of scientific digital libraries include earthquake simulation repositories [26], embedded sensor network DLs [27], and D4Science II [28]. MARIAN is a digital library that supports networks of digital information objects [29]. Barrett et al. describe a data management tool to study infectious diseases [30]. Schriml et al. provide the GeMIna system, which uses epidemiology metadata to identify infectious pathogens and their representative genomic sequences [31]. The VecNet digital library maintains curated data, tagged citations, and articles related to epidemiology [19]. Leidig describes a scientific digital library to manage epidemiology experiments and simulations, that served as a starting point for our current work [32].

Shi et al. present a web-based epidemiology reporting system using Google Maps data [33]. Allon et al. describe a leukemia epidemiology study where the data model arises from the experiment based on a relational modeling approach [34]. In the domain of the Semantic Web, tools that map data from the relational model to RDF (and vice versa) have been an active area of research [35, 36, 37, 38, 39]. Bertails et al. demonstrate the power of mapping to treat all important features of SQL tables, like cardinality and NULLs, and to yield an RDF graph preserving the relational information [36]. Hert et al. illustrate ontology-based access to relational databases as discussed in [40]. It describes a mapping language, the translation algorithms, and a prototype implementation. Robert et al.

developed a schema ontology for healthcare using a reference information model (RIM). This modeling helps better manage healthcare effectiveness data and information set (HEDIS) measures by providing a rule ontology matched to the language of the specification [41]. Horrocks et al. propose an ontology-based data access (OBDA) solution for diverse varieties of data management. They describe their efforts in managing disparate Siemens Energy Services datasets; see [42] for further discussion.

The Ebola epidemic of 2014 and 2015 has been addressed by several papers. Van Kerkhove, Bento, Mills, Ferguson, and Donnelly created a detailed database containing estimates of epidemiological parameters on the basis of delay distributions, the incubation period distribution, effective reproduction number, basic reproduction number, and case fatality rate [43]. Lu et al. developed an innovative visual analytics framework for sentiment visualization of geolocated Twitter data [44]. In order to enable the quick virtual screening of molecular libraries for candidate inhibitors of Ebola virus infection, Veljkovic et al. designed efficient theoretical criteria [45]. Fallah et al. examined transmission factors and heterogeneity of Ebola incidence in over 300 communities in Montserrado County, Liberia. The sample was categorized by socioeconomic status (SES) [46]. Richards et al. collected cross-sectional data about the epidemic in Sierra Leona, focusing on rural areas in which the spread of Ebola was undocumented to a great extent [47]. To understand human mobility during the Ebola outbreak, Wesolowski et al. developed the utility of call data records (CDRs). They underlined the pressing need to have protocols for prompt sharing of operator data to respond to public health emergencies adequately [48]. Liu et al. aimed at identifying early stage host factors related to acute illness and focused on the factor responsible for survival or fatality. In their research, the authors analyzed transcriptome data for peripheral blood taken from convalescent recovering patients [49]. Drake et al. explored modeling directions, suggesting that the minimal epidemiologic dataset for an Ebola model should include compliance with intervention recommendations, distribution of secondary cases by infection, and the duration of the incubation period [50]. Caballero et al. examined the effect of interferon-signaling in response to Ebola virus infection in the body [51]. Li and Chen proposed a hypothesis of the evolutionary history of Ebola virus to examine its molecular evolution [52]. Stadler et al. applied phylogenetic trees to discover the role of the population structure in regards to the Ebola virus [53]. Prabhat Jha rated new introductions, case fatality ratio, and the potential to spread from person to person [54]. Giovanetti et al. conducted analysis and homology modeling on the basis of the G Glycoprotein (GP) sequences obtained from public databases. The authors explored the genetic diversity and modification of antibody response to the outbreak of the Ebola virus [55]. Tran and Lee completed a detailed study of understanding and mining the spread of information related to Ebola on social media [56]. Odlum and Yoon examined the use of real-time Ebola outbreak surveillance to follow the spread of information, detect an

epidemic early, and explore the content of public knowledge and attitudes [57]. Hunt et al. used laboratory and clinical data from patients with the Ebola virus to improve clinical management algorithms, enhance understanding of the main variables related to outcome, and gain insight into the pathophysiology of the Ebola virus [58]. Rubins et al. enhanced comprehension of the Ebola virus (EBOV) pathogenesis and EBOV-host interactions by exploring the molecular features of EBOV infection in vivo [59].

Multiple studies have addressed processing queries using mapping [60, 61, 62]. Bornea et al. describe novel query translation techniques as well [60]. Experiments show the approach provides good results compared with current state-of-the-art RDF stores. Groppe et al. describe a SPARQL query optimization technique that uses seven indices to retrieve RDF data quickly [61]. This approach computes joins by dynamically restricting triple patterns, and provides good efficiency. Arenasa et al. introduce a theoretical foundation for faceted search explicitly designed for RDF-based knowledge graphs improved with OWL 2 ontologies. Authors also investigate convenient faceted interfaces [63].

Unification of heterogeneous data, using a semantic approach, has been actively studied for some time. BioPortal provides biomedical ontologies [64]. Federation (or mediation) of physically distributed data sources using RDF (and other semantic constructs) in DartGrid is presented in [65], which provides RDF/OWL to define the mediated ontologies for integration, and automatic conversion rules from the relational schema to RDF/OWL descriptions. Kamdar et al. propose an approach for Ebola virus knowledge-based creation using semantic web technologies [66]. An affinity-based unification method is presented in [67]. The Epidemic Marketplace framework [68, 69] is part of the EPIWORK project [70]. Authors of the Epidemic Marketplace literature discuss integrating multiple heterogeneous epidemic data sources but provide no detailed description of how they achieve it. The project ended in 2013 and has not been actively maintained, making it harder to obtain specific details. Semantic integration of multiple heterogeneous semi-structured and structured data sources is discussed in [71]. Zhu et al. proposed a data integration approach for large-scale spatially detailed synthetic populations [72]. Many research papers about biological data integration using semantic web technologies are available [73, 74, 75, 76, 77]. Biozon combines many biological databases including a variety of data types (e.g., DNA sequences, proteins, interactions, and cellular pathways) [78]. Amann et al. discussed a modular, component-based method to build metadata schema by integrating ontologies and thesaurus hierarchies [79].

Ontologies are an important part of the semantic web and an active research area. Epidemiology is an interdisciplinary area and several disease ontologies fit in this domain, for example, Disease Ontology [80], Infectious Disease Ontology (IDO) [81], Symptom Ontology [31], Vaccine Ontology [82], Pathogen Transmission Ontology (TRANS) [31] etc. Pesquita et al. developed an Epidemiology Ontology (EPO) [83] that considered the

ontologies mentioned above and the Network of Relevant Ontologies for Epidemiology (NERO) [84], Open Biomedical Ontologies (OBO) Foundry guidelines [85], Basic Formal Ontology (BFO) [86], and Information Artifact Ontology (IAO) [87]. Recently Hogan et al. proposed the Apollo Structured Vocabulary (Apollo-SV) for epidemic simulations [88].

Ontology is a vital part of semantic web technology. We have developed a lightweight ontology as part of the proposed schema. The concepts, relations, or vocabulary terms we employ in our schema may be partially available in other ontologies. Our schema is flexible enough to incorporate other ontologies. *We omit further discussion in this dissertation because our focus is not on developing a comprehensive ontology for the computational networked epidemiology domain but on showcasing the benefit and suitability of a semantic web-technology based schema for heterogeneous and fragmented computational networked epidemiology data integration.*

2.2 Preliminaries

The schemas are expressed using OWL and RDF concepts and terminologies. We describe below these concepts along with the specific interpretation we have adhered to in this work. Also, we explain some relational database notions used in the dissertation.

2.2.1 Relational Data Model

This is a data model that contains a collection of relations where each relation is expressed by a "table" with rows and columns [89].

Primary Key: A column (or a combination of columns) that uniquely identifies table records [90].

Foreign Key: A column (or a combination of columns) in one table that uniquely identifies records in another table. It expresses a link between two tables [91].

Super Key: A super key expresses any collection of columns that are combinatorically unique. There may be several super keys of a table, and multiple super keys can use the same column [92].

Composite Key: A composite key has more than one column that uniquely identifies a row. A composite key is the minimal super key [92].

2.2.2 Resource Description Framework (RDF)

In the following, we present an illustration of RDF concepts.

Resource: Anything can be a resource, for example, physical thing, metaphorical idea, web content, number, string, etc. The term resource is identical to the term "entity" [93].

RDF data model: RDF is the core of semantic web technology. It is a graph based data model that stores data in a directed labeled graph. RDF stores data in triples. A triple is composed of three parts called subject, predicate, and object. It is designed to be used in the global scale. It uses unique Hypertext Transfer Protocol (HTTP) Uniform Resource Identifiers (URIs) and standard or domain specific vocabulary terms. RDF serialization supports storing data in multiple formats. They include RDF/XML, NTriple, Turtle, and N3 [94].

RDF Schema (RDFS): A data-modelling vocabulary for RDF data. At their core, RDFS vocabularies provide class and property type descriptions. The RDFS has similarities to the object-oriented programming languages (e.g., Java) because both contain the concepts of class, subclass, property, inheritance, and instance. However, it is worthwhile to state that the definition of some concepts mentioned above is different in RDFS than in object-oriented programming languages (e.g., property) [95, 96].

- Class: A class represents a collection of resources [95].
- SubClass: A subclass is a subordinate of another class [95].
- Property: An RDF property represents relations between resources [95].
- SubProperty: SubProperty correlates a property to one of its super properties [95].
- Instance: Instances are physical objects (or things). The predicate "rdf:type" is used for instance creation [95].
- Literal Class: Represents the class of literal values (e.g., integer, string, etc.) [95].
- Blank Node: An RDF blank node represents an anonymous resource for that URI, or that a literal value is not present. The RDF standard permits subject and object portions of the RDF triple to be a blank node [97].
- Named Graph: Named graph is an RDF concept used to assign a name to any subset of triples. In this dissertation, we use URIs to label named graphs. A named graph is a powerful triple store feature. It enables a user to identify a set of triples. Sometimes a quad type structure is used instead of the triple to represent a named graph [98].

A quad contains an RDF triple plus an additional fourth element (e.g., graph name). Selection of the named graph data model depends on the context. Named graphs can be queried. SPARQL uses the "GRAPH" keyword with the graph URI to find a particular named graph, allows named graphs to be variables, and allows for simultaneously querying over multiple named graphs [99, 100].

2.2.3 Web Ontology Language (OWL)

The OWL is a semantic web language. The purpose of the language is for publishing and distributing ontologies on the World Wide Web [101].

Thing: "owl:Thing" describes what someone is attempting to represent and model. It's equivalent to "rdfs:Resource" [102, 103].

Object property: Represents a link between individuals (instances). Suppose "AlgorithmBook" and "Knuth" are two individuals of the classes "Book" and "Author", then by using the "hasAuthor" object property, we can represent the relationship between them as follows: "AlgorithmBook hasAuthor Knuth." "owl:Thing" serves as both the domain and the range of the object property.

Data property: Data properties represent a link from individuals (instances) to literal value (e.g., strings, numbers, etc.). Suppose individual "AlgorithmBook" has an ISBN value "978-0262033848," then by using the "hasISBN" data property, we can represent the relationship as follows: "AlgorithmBook hasISBN 978-0262033848." "owl:Thing" and "rdfs:Literal" are the domain and range of the data property.

Axiom: OWL axiom denotes the class of annotated axioms. It aids various RDF serializations in annotating subject, predicate, and object [104].

Annotated Source: Expresses the property that describes the subject of an annotated axiom [104].

Annotated Property: Expresses the property that describes the predicate of an annotated axiom [104].

Annotated Target: Expresses the property that describes the object of an annotated axiom [104].

Note 1: In our schema, properties are explicit, i.e., instead of using generic relations (e.g., has) we use more domain specific relations (e.g., hasHousehold, hasLocation, hasActivity, etc.) [105]. This design is preferred as the resulting RDF dataset is semantically richer and easier to comprehend by a larger group of users.

Note 2: In this dissertation, subclass means "rdfs:subClassof." This dissertation considers "A rdfs:subClassOf B" to mean all instances of class A are also an instance of class B and all the properties of B are also properties of A [106]. We know the semantic web community has different opinions about subclass property inheritance [107], but those are not under consideration for this dissertation.

2.2.4 D2RQ Mapping Language

The D2RQ Mapping Language maps a relational database schema to RDF vocabularies and OWL ontologies [108]. The D2RQ mapping file allows conversion of a relational database to an RDF graph. In Listing 2.1, we present a sample D2RQ mapping file. Below the sample mapping we provide a description of the D2RQ mapping language terminology.

Listing 2.1: D2RQ mapping example.

```
.............................................. 1
map:database a d2rq:Database; 2
        d2rq:jdbcDriver "oracle.jdbc.driver.OracleDriver"; 3
        d2rq:jdbcDSN "jdbc:oracle:thin:@//ndssl.bi.vt.edu:xxxx/"; 4
        d2rq:username "xxxx"; 5
        d2rq:password "*****"; 6
        . 7
map:student a d2rq:ClassMap; 8
        d2rq:dataStorage map:database; 9
        d2rq:uriPattern "http://ndssl.bi.vt.edu/student/pid#@@student.id@@"; 10
        d2rq:class vocab:student; 11
        d2rq:classDefinitionLabel "student"; 12
        . 13
map:student__label a d2rq:PropertyBridge; 14
        d2rq:belongsToClassMap map:student; 15
        d2rq:property rdfs:label; 16
        d2rq:pattern "student id#@@student.id@@"; 17
        . 18
map:student_id a d2rq:PropertyBridge; 19
        d2rq:belongsToClassMap map:student; 20
        d2rq:property vocab:hasID; 21
        d2rq:propertyDefinitionLabel "student id"; 22
        d2rq:column "student.id"; 23
        d2rq:datatype xsd:decimal; 24
        . 25
.............................................. 26
```

ClassMap: D2RQ *ClassMap* expresses a class or a collection of identical classes from an RDFS vocabulary or OWL ontology [108]. *ClassMap* is a mechanism to define a URI or blank node for instances of a class. *ClassMap* contains a set of property bridges. The property value can be literal, URI, or blank node.

PropertyBridges: PropertyBridges defines *ClassMap* RDF resource properties. It associates a database column with an RDF property [108].

Database: In D2RQ, *Database* expresses the JDBC connection to native and remote relational databases [108].

dataStorage: In D2RQ, *dataStorage* refers to a *Database* [108].

class: In D2RQ, *class* means RDFS or OWL class [108].

uriPattern: D2RQ uses *uriPattern* to recognize instances of the *ClassMap* [108].

classDefinitionLabel: defines a class label. It acts as an *rdfs:label* for an affiliated class. [108].

belongsToClassMap: shows that a property bridge refers to a *ClassMap*. It is required for all *PropertyBridges* [108].

property: In D2RQ, *property* indicates the RDF property that associates *ClassMap* with the object or literal [108].

pattern: specifies properties with a literal value [108].

propertyDefinitionLabel: represents a label as *rdfs:label* for the corresponding property definition [108].

column: specifies the database column that holds literal values [108].

datatype: define literal's RDF datatype [108].

2.2.5 SPARQL Protocol and RDF Query Language (SPARQL)

A SPARQL query finds graph patterns. A SPARQL *WHERE* clause describes the graph, and pattern conditions. A pattern may include a variable (e.g., ?household) and serve as a wildcard which means any value in that position is fine. The variable also stores the value that allows a user to utilize it in various parts of the query. The SPARQL *WHERE* clause extracts data from the dataset and *SELECT* shows the part of the extracted data the user wants to see [100, 109, 110, 111].

2.3 CNEW Terminology

We provide focused definitions for terminologies used in this dissertation to describe different aspects of CNEW datasets.

Cell: A cell is a configuration for a simulation. A cell denotes one collection of defined simulation conditions [14].

Configuration: The configuration file contains EpiStudy simulation input parameters and data file locations [112].

ContactNetwork: The contact network is an edge-weighted graph where nodes represent individuals in the population; an edge represents a contact between two nodes, and edge weights represent contact duration in seconds [112].

Dendrogram: EpiStudy produces a dendrogram as an output, "A dendrogram is a labeled graph used to represent the stochastic output of an epidemic process on a graph" [21].

DiseaseModel: Disease model deals with the spread of diseases through a population [14].

EpiStudy: An EpiStudy is a study of an epidemic that uses computer models and simulation tools to understand the spatio-temporal spread of diseases [113, 14]. An EpiStudy comprises a contact network (person-to-person and person-to-location) developed from a synthetic population of a specific geographic region, disease transmission model (e.g., Catastrophic Flu: A flu that infects over 50% of population), initial condition (e.g., 20 seeds from school age), number of replicates (e.g., 10), number of simulation days (e.g., 120), and intervention (e.g., vaccination).

InitialCondition: At the start of the EpiStudy, a user must specify whether random

individuals are selected to be infected at: the beginning of an experimental period, during the experimental period, or at specific days of the experimental period [14].

Region: The region represents a geographical territory where the EpiStudy is performed [14].

Replicate: The replicate defines the number of times EpiStudy execution is repeated to reduce changeability in the results. In replicates, an individual run uses a distinctive random seed (initially infected people) [14, 114].

SimulationEngine: The algorithm used to conduct simulations (e.g., EpiFast, EpiSimdemics, etc.) [14].

Synthetic Population: The synthetic population contains a set of synthetic people, households, locations, and activities information for a particular geographic region [115].

Note 3: In this dissertation, we are not considering "intervention" and "Epicuve." These will be addressed in future work.

Chapter 3

A Framework for Unified Simulation Epidemiological Datasets (FUSED)

In this chapter, we propose a schema called FUSED to logically organize the diverse datasets that comprise typical epidemic analysis workflows. FUSED focuses specifically on workflows developed for networked epidemiology (referred to as computational networked epidemiology workflows (CNEW)) – wherein social interactions are explicitly represented within the framework to capture the geographical and social variability inherent in populations and disease dynamics. We refer to datasets that are used and generated as a part of such workflows as CNEW datasets. FUSED represents CNEW datasets and their relationships using RDF/OWL-based syntax, utilizing a three-tier architecture consisting of a conceptual layer, a logical layer, and a physical layer. The conceptual layer schema presents a high-level view of CNEW datasets and hides lower level details about the data, including representation, layout, and storage. This exposes common data properties that fit the mental models of users. The logical layer schema captures data fragmentation and linking aspects of the CNEW datasets. The physical layer schema covers the storage aspects of the CNEW datasets.

3.1 FUSED Schema

We now describe FUSED schemas designed using the semantic web and relational concepts. The schemas collectively describe the system and the semantic aspects of the datasets in CNEW. We show the three layers of schemas and their mappings that together present a unified view of this fragmented landscape of CNEW datasets. *We describe our schema by using several listings. In the listings, the separator line dots (.) imply we are omitting schema details. The separator line dashes (-) help identify different listing sections.* A pictorial view of a part of our schema and schema instance example are shown in Figs. 3.1, 3.2, and 3.3.

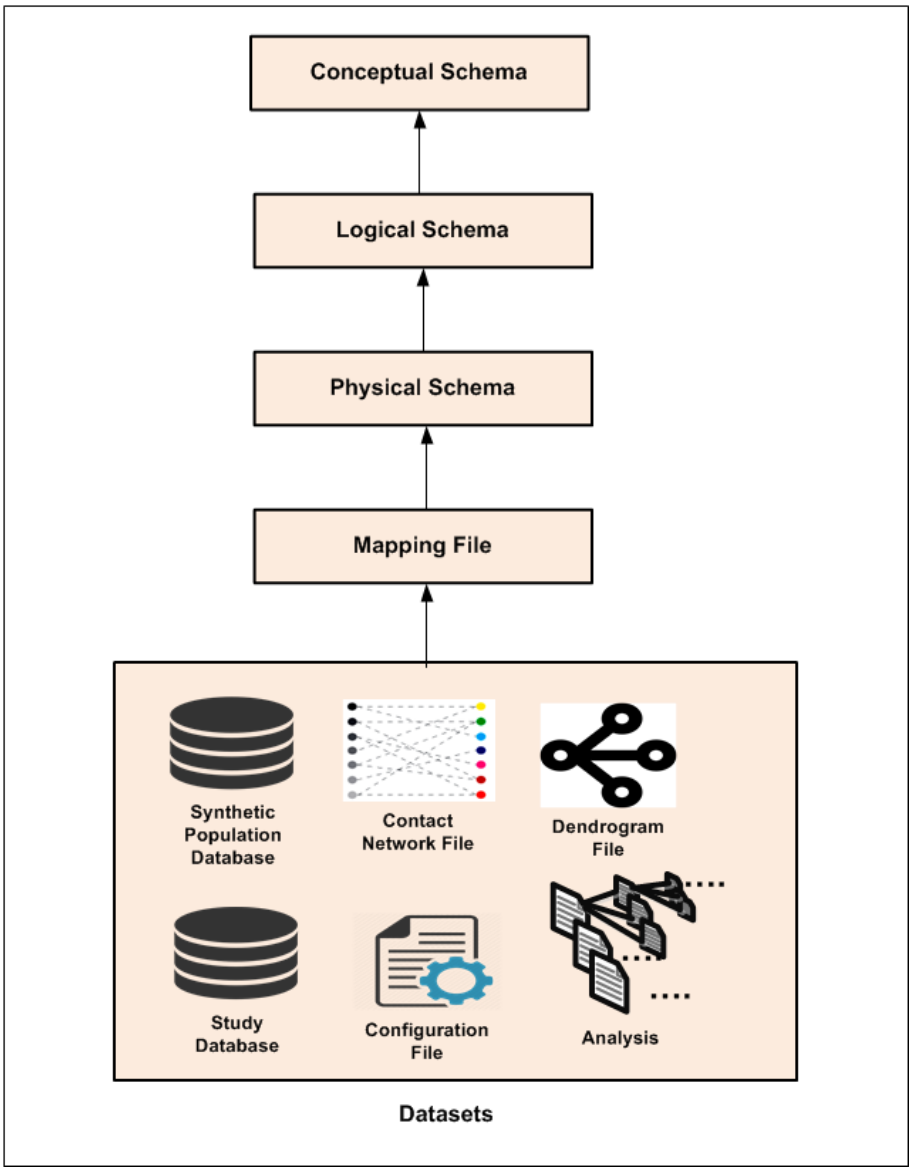


Figure 3.1: High-level architecture of FUSED. Conceptual, logical, and physical are three layers of the FUSED schema. The conceptual schema presents a simplistic portrayal of the complex CNEW datasets. It is connected to the logical schema that captures data fragmentation and linking aspects of the CNEW datasets. The logical schema is attached to the physical schema that covers the storage features of the CNEW datasets. These schema layers are connected with underlying heterogeneous and fragmented datasets through a mapping file. The bottom rectangular box contains a high-level overview of the heterogeneous and fragmented CNEW datasets.

3.1.1 Conceptual Schema

The conceptual schema aims to describe the CNEW concepts and relationships that match the external user's mental model, i.e., only the semantic aspects of the CNEW datasets. We describe the synthetic population conceptual schema first and follow by defining the

conceptual schema for an epidemiological study consisting of experiment setup and output. In Fig. 3.4, we present a pictorial view of our conceptual schema.

Synthetic population classes are as follows: *Person* models a synthetic person, *Household* models a set of persons that reside in the same household, *Activity* models how each person spends his time during the day, and *Locations* captures the physical space where an activity takes place. Technically, *Household* is a type of location that hosts *home* activity. However, this is not used in CNEW datasets in a manner that fits most users' mental models, and therefore has been omitted from the conceptual schema.

Listing 3.1: Synthetic population classes.

```
:Person, :Household, :Activity, :Location
```

We now turn to properties for these classes. Properties are of two kinds: the data property that defines literal attributes of the classes, and the object property that defines relationships (or linking between) classes. We represent the data property as *hasDataPropertyAttribute* and object property as *hasObjectPropertyAttribute*. Property names are unique in our schemas.

Our primary focus, in this dissertation, is on linking. Hence, we mention only those data and object properties required to define various links across datasets.

We define the following object properties for a synthetic population:
hasHousehold implies a Person lives in a Household; *hasLocation* indicates a Household has a Location; *hasActivity* implies a Person performs an Activity; *hasActivityLocation* links an Activity to a Location.

In addition, each class has a number of data properties. For example, the *hasAge* data property represents the age property of the *Person* class (Listing 3.2). In our schema, all object properties are sub-properties of *hasObjectPropertyAttribute* and all data properties are sub-properties of *hasDataPropertyAttribute*.

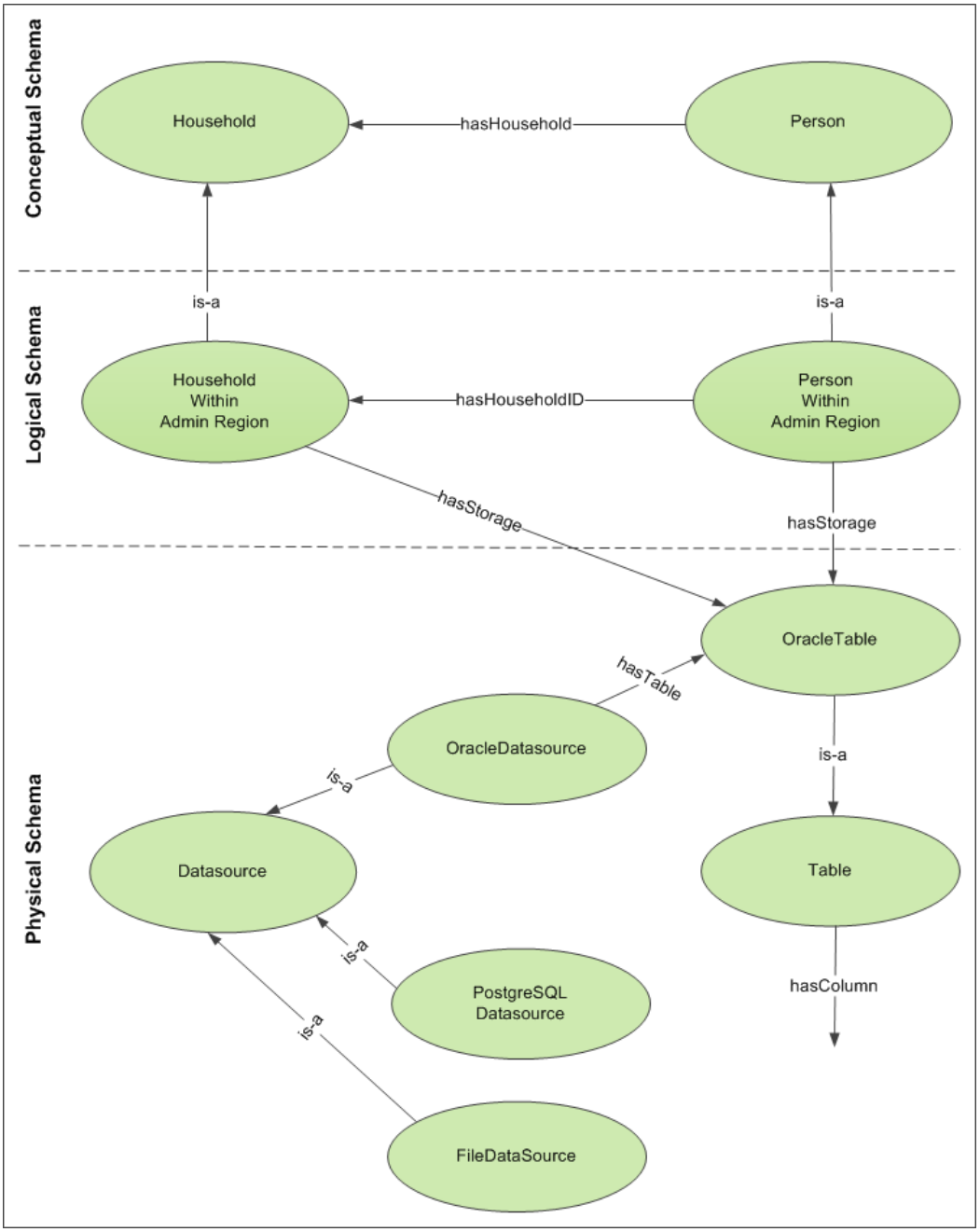


Figure 3.2: Interaction between three layers of the FUSED schema. For clarity, only a partial view of the entire schema is shown. The conceptual layer shows top-level classes (simple and intuitive) to the end user. Typically, these classes are fragmented (for example, the synthetic population is fragmented across the parts of a country: state, province, area, etc.). The logical level captures that fragmentation. Finally, the physical schema captures the storage and formatting aspects of the CNEW datasets. Here, green ovals represent classes, arrows demonstrate relationships (object property) among classes, and "is-a" means "rdfs:subClassOf".

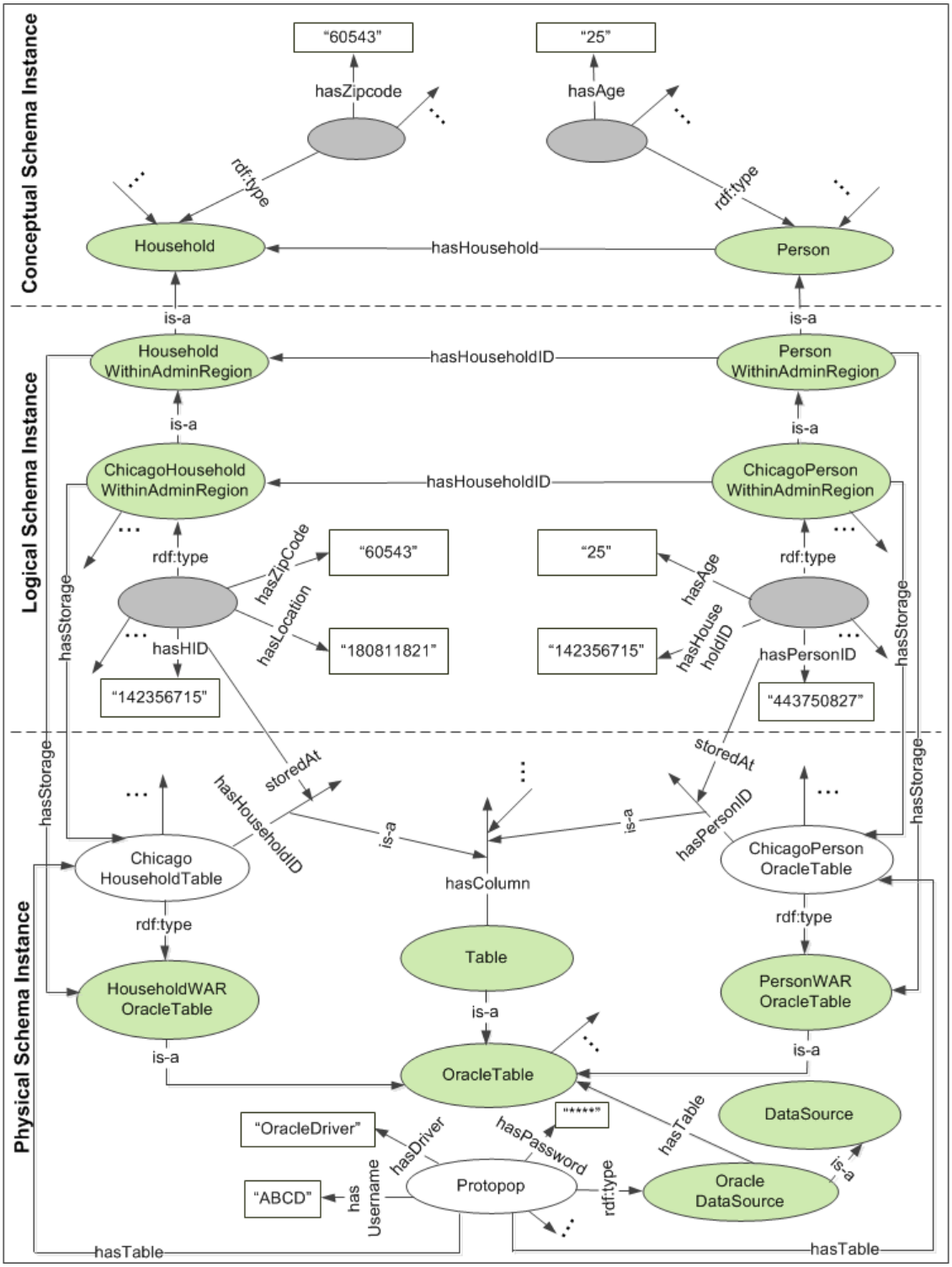


Figure 3.3: Instance level example of the FUSED schema, as mentioned in Figure 3.2. This example is based on the Chicago administrative region. Conceptual, logical, and physical schema and their instances are presented in the top, middle, and bottom segments of the figure. We show the relationships in both schema and instance level to display a clear view of our approach. Here, green ovals represent classes, white ovals mean class instances, gray ovals mean blank nodes, the rectangular boxes represent literal values, and the arrows represent object and data properties. This is a partial view of the complete schema.

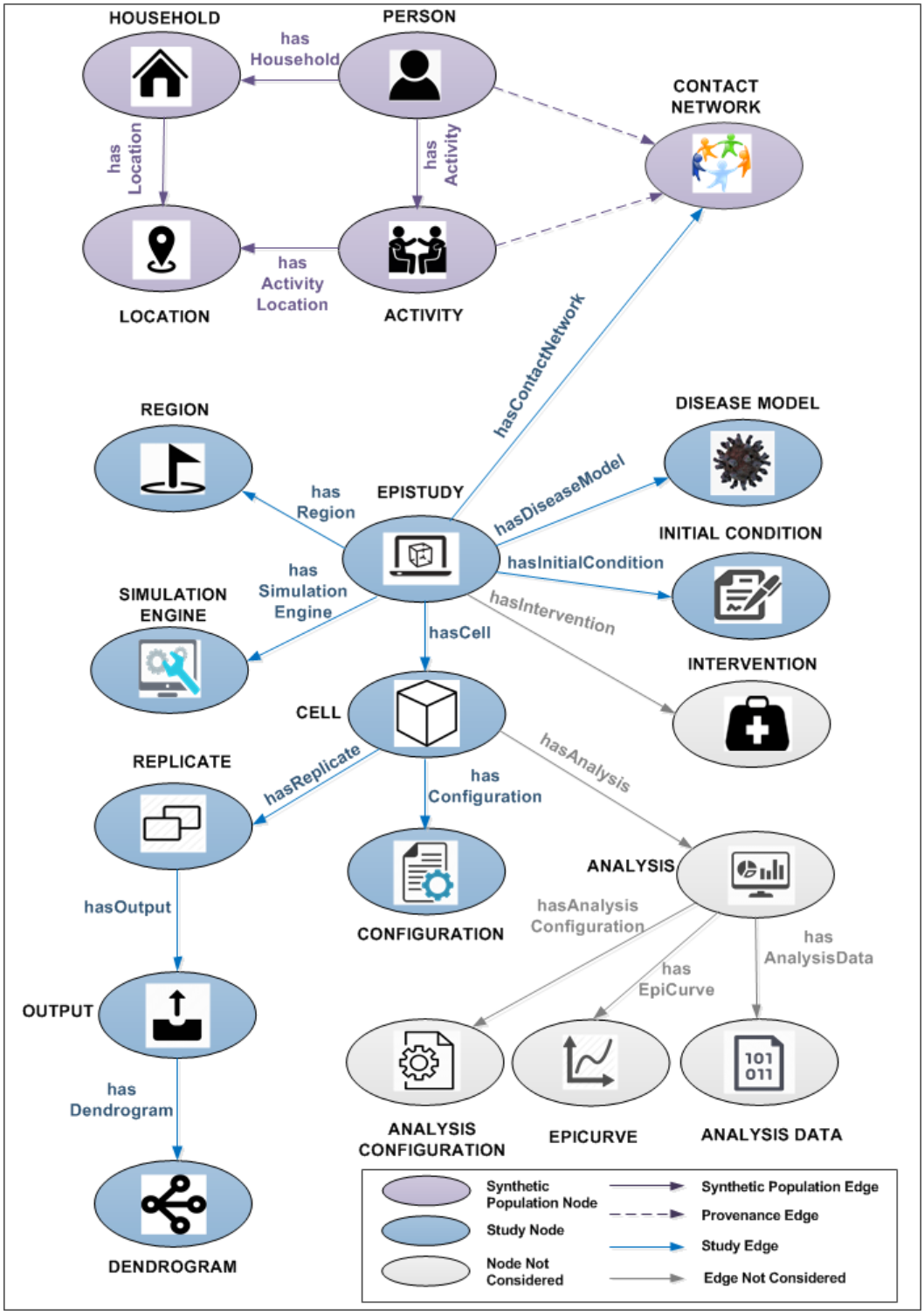


Figure 3.4: FUSED conceptual schema in a graph format. The conceptual schema displays a simplistic portrayal of the complex CNEW datasets. In the figure, ovals and arrows with a light purple color represent synthetic population classes and their relationships. Blue ovals and arrows symbolize study related classes and their connections. Light gray ovals and arrows represent classes and relationships not considered in this dissertation (left for our future work).

Listing 3.2: Relationship among synthetic population classes, a property domain and range example, a class property example.

```
:Person     rdf:type    owl:Class
.............................................
:Person     :hasActivity            :Activity
:Person     :hasHousehold           :Household
:Household  :hasLocation            :Location
:Activity   :hasActivityLocation    :Location
.............................................
:hasObjectPropertyAttribute
               rdf:type owl:ObjectProperty

:hasActivity
  rdfs:subPropertyOf :hasObjectPropertyAttribute
             rdfs:domain    :Person
             rdfs:range     :Activity
.............................................
:hasDataPropertyAttribute
        rdf:type owl:DatatypeProperty

:hasAge
    rdfs:subPropertyOf :hasDataPropertyAttribute
:hasGender
    rdfs:subPropertyOf :hasDataPropertyAttribute


:hasAge         rdfs:domain     :Person
                rdfs:range rdfs:Literal

:hasGender      rdfs:domain     :Person
                rdfs:range rdfs:Literal
.............................................
```

So far we have defined atomic classes and relationships to model a synthetic population. The social contact network, which is a central data type for the CNEW dataset, can be derived from the basic synthetic population constructs (derivation discussion is available in the logical schema). The social contact network consists of *social contact*, i.e., a representation of the event where two people are present at the same location during an overlapping time period. We refer to this concept as *ContactNetwork* in our model.

We now turn our attention to the specific (epidemiological) study part of the model which, along with other classes (disease models, etc.) takes a contact network and builds several stochastic instantiations of contagion percolation through it. First, the classes required for modeling are given in Listing 3.3.

Listing 3.3: Study related classes.

```
:EpiStudy,          :Region,        :DiseaseModel,
:SimulationEngine,  :InitialCondition,
:Cell,              :Configuration, :Replicate,
:Output,            :Dendrogram
```

Class *EpiStudy* captures the design of a study setup; *DiseaseModel* classifies the contagious disease being studied, such as influenza, etc.; *SimulationEngine* is a tool that propagates an epidemic on a contact network; *InitialCondition* captures various contagion parameters that control the spread of the disease. Each experiment is divided into multiple *Cells* where each cell represents a set of specific simulation conditions. A *Cell* contains a *Configuration* that models the study arrangement, while *Output* contains output data from a simulation run. The relationships among these and with *ContactNetwork* are presented in Listing 3.4.

Listing 3.4: Relationship among study classes and contact network.

```
:EpiStudy    :hasContactNetwork    :ContactNetwork
             :hasRegion            :Region
             :hasDiseaseModel      :DiseaseModel
             :hasSimulationEngine  :SimulationEngine
             :hasInitialCondition  :InitialCondition
             :hasCell              :Cell

:Cell        :hasConfiguration     :Configuration
             :hasReplicate         :Replicate

:Replicate   :hasOutput            :Output

:Output      :hasDendrogram        :Dendrogram
```

Output contains information about epistudy, cell, replicate, infector, infectee, and exposure day information. The key output of interest to an end user is the infection tree (also called the dendrogram).

Queries Over Conceptual Schema

The conceptual schema exports the "storage agnostic" linked and unified view of the CNEW datasets. This view can be used to formulate queries yielding meaningful insights about the synthetic population structure and disease dynamics. The main advantage of expressing queries over the conceptual view compared to the current state-of-the-art (specifying queries over a heterogeneous mix of storage media) is simplicity and maintenance. Querying over the conceptual schema is simple because: (*a*) it hides the lower level details about the data, including representation, layout, and storage; (*b*) it provides a linked view of the entire set of CNEW datasets; and (*c*) it only exposes common attributes to the epidemiologist.

Currently, queries in FUSED are expressed in SPARQL (see Section 2.2.5 for a brief overview). Consider as an example the following query: find pairs of individuals who live and work at the same location (Listing 3.5). The structural representation of the query is depicted in Fig. 3.5. The query simply enumerates the edge constraints as triples.

Example Query: Find two people who live in the same large household and also work in the same location (Fig. 3.5).

Listing 3.5: Example query on conceptual level.

```
SELECT                                                          1
?person1 ?person2                                               2
?work_location                                                  3
WHERE                                                           4
{                                                               5
?person1    :hasHousehold ?household.                           6
?person2    :hasHousehold ?household.                           7
?household  :hasSize      "Large".                              8
?person1    :hasActivity ?activity1.                            9
?person2    :hasActivity ?activity2.                            10
?activity1  :hasActivityType "work activity".                   11
?activity2  :hasActivityType "work activity".                   12
?activity1  :hasActivityLocation ?work_location.                13
?activity2  :hasActivityLocation ?work_location                 14
}                                                               15
```

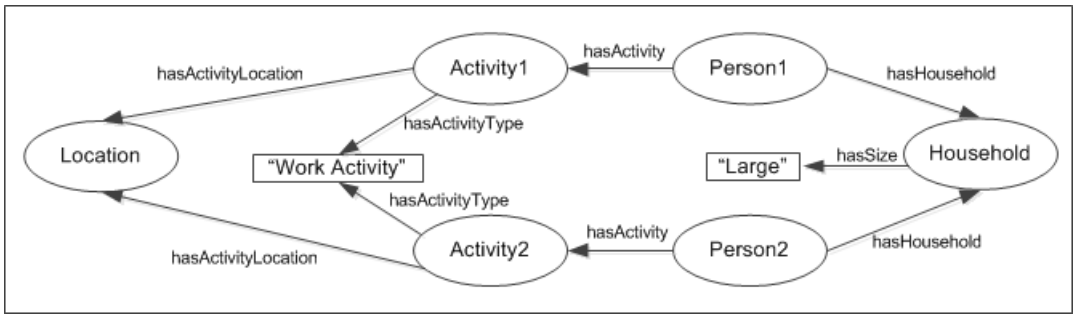


Figure 3.5: SPARQL graph pattern of the example query: (find two individuals who are part of a household and also work at the same location). Here, white ovals represent classes, rectangular boxes represent literal values, and the arrows represent object and data properties.

We argue that the above query expression is more straightforward and maps directly to the constructs one would intuitively craft to express this query. Furthermore, the query hides the organizational and storage details – making it far less cumbersome compared to an expression in SQL or using logical or physical schemas (described next).

3.1.2 Logical Schema

We now describe the logical schema that captures the fragmentation aspect of CNEW datasets. It also defines the linking necessary to "defrag" the dataset and provide a unified view. The mappings that tie this unified logical view to the conceptual schema are presented in the mappings section (see Section 3.2 for details).

Typically, the synthetic population data are fragmented along two dimensions: partitioning along administrative regions (part of a country, e.g., state, province, area, etc.) dimension; and is also along the type of data, i.e., the person dataset is siloed from the household dataset and so on. To capture fragmentation along administrative regions we introduce the *AdminRegion* class which is part of the *Region* class. For example, the United States can be considered as a region and the state of Florida as a subregion. We use *WAR* as an abbreviation of the WithinAdminRegion and define *PersonWAR*, *HouseholdWAR*, *ActivityWAR*, and *LocationWAR* which are subclasses (see Note 2 in preliminaries) of the classes *Person*, *HouseHold*, *Activity*, and *Locations*.

A partial view of the logical schema is available in Fig. 3.6. It shows the logical organization of the person, household, location, and activity datasets of the CNEW pipeline. The respective fragmented class represents each dataset. For example, the household dataset is represented using *HouseholdWAR*. Each class has a set of data properties. Data properties represent the attributes of the datasets. There are two kinds of data properties. One maps directly to conceptual level properties. For example, age is an attribute of the person dataset and is represented by the *hasAge* data property of the corresponding class *PersonWAR*. This property maps directly to the conceptual level data property *hasAge* of class *Person*. The second kind of data property is one which encodes the linking between fragmented datasets. In the relational framework, such attributes are referred to as primary key/foreign key attributes. In FUSED, these attributes are annotated to assert that they express a foreign key to a target class. For example, *householdID* is an attribute of the person dataset and is used to refer to the household of the person. In the logical schema, this attribute is expressed using the *hasHID* data property. Furthermore, it is linked to *HouseholdWAR* class's *hasHouseholdID* data property through the *linksviaPrimaryKeyForeignKey* data property. Later in the *CNEW Logical Schema* subsection, we describe our mechanism to express this within the RDF framework.

The following logical schema represents the fragmentation (by regions) of the synthetic population:

Listing 3.6: Synthetic population subclasses.

```
:PersonWAR      rdfs:subClassOf     :Person                                    1
:HouseholdWAR   rdfs:subClassOf     :HouseHold                                 2
:ActivityWAR    rdfs:subClassOf     :Activity                                  3
:LocationWAR    rdfs:subClassOf     :Location                                  4
```

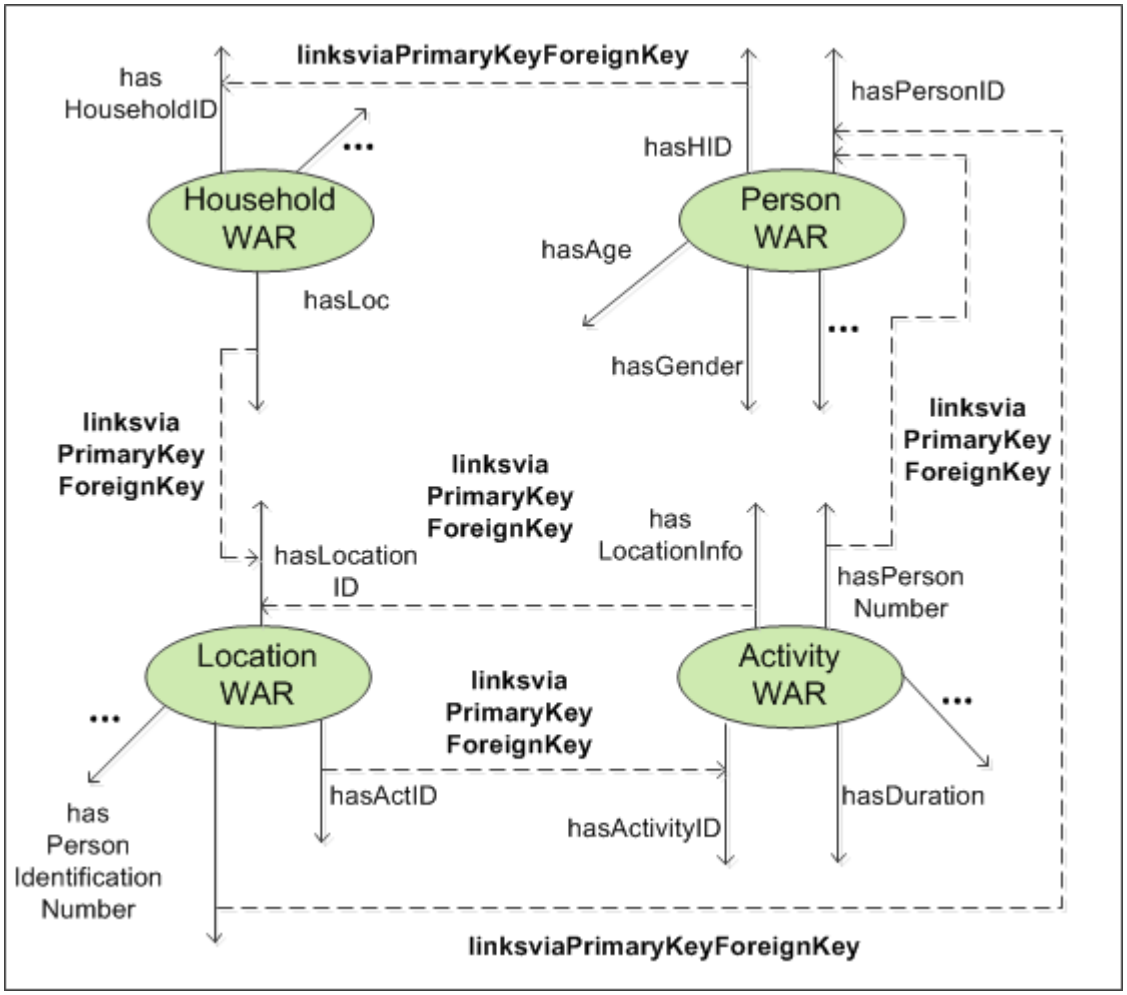


Figure 3.6: A partial pictorial view of the logical schema, which links heterogeneous and fragmented datasets. Here, green ovals represent classes, straight arrows show data properties, and dotted arrows indicate a connection between two data properties from two classes.

Logical Schema Macros

We define a set of macros in listings 3.7, 3.8, 3.9, 3.10, and 3.11. The macros (or RDF patterns) enable us to present a succinct representation of the logical schema, thereby improving readability.

Listing 3.7: Macro 1: Class and its properties.

```
#define                                                                 1
CLASS_AND_PROPERTY(CLASS_NAME, PROPERTY_LIST)                           2
:CLASS_NAME     rdf:type    owl:Class                                   3
                                                                        4
:PROPERTY_LIST  rdfs:subPropertyOf                                      5
                    :hasDataPropertyAttribute                           6
                rdfs:domain owl:Thing                                   7
                rdfs:range  rdfs:Literal                                8
-----------------------------------------------                         9
Where:                                                                  10
CLASS_NAME = Name of the class.                                         11
PROPERTY_LIST = List of the class properties.                           12
-----------------------------------------------                         13
Example:                                                                14
personPropertyList=                                                     15
[hasPersonID, hasAge, hasGender,...]                                    16
                                                                        17
CLASS_AND_PROPERTY(Person, personPropertyList)                          18
                                                                        19
Means:                                                                  20
                                                                        21
:Person         rdf:type    owl:Class                                   22
                                                                        23
:hasPersonID    rdfs:subPropertyOf                                      24
                    :hasDataPropertyAttribute                           25
                rdfs:domain :Person                                     26
                rdfs:range  rdfs:Literal                                27
:hasAge         rdfs:subPropertyOf                                      28
                    :hasDataPropertyAttribute                           29
                rdfs:domain :Person                                     30
                rdfs:range  rdfs:Literal                                31
:hasGender      rdfs:subPropertyOf                                      32
                    :hasDataPropertyAttribute                           33
                rdfs:domain :Person                                     34
                rdfs:range  rdfs:Literal                                35
.........................................                               36
```

Listing 3.8: Macro 2: Data property.

```
#define DATA_PROPERTY(PROPERTY_NAME)

:PROPERTY_NAME
  rdfs:subPropertyOf :hasDataPropertyAttribute
-----------------------------------------------
Where:
PROPERTY_NAME = Name of the property.
-----------------------------------------------
Example:

DATA_PROPERTY(hasAge)

Means:

:hasAge
   rdfs:subPropertyOf :hasDataPropertyAttribute
```

Listing 3.9: Macro 3: Primary key data property.

```
#define
PRIMARY_KEY_PROPERTY(PRIMARY_KEY_PROPERTY_NAME)

:PRIMARY_KEY_PROPERTY_NAME
 rdfs:subPropertyOf :hasPrimaryKeyAttribute
-------------------------------------------
Where:
PRIMARY_KEY_PROPERTY_NAME = Name of the
primary key property.
-------------------------------------------
Example:

PRIMARY_KEY_PROPERTY(hasPersonID)

Means:

:hasPersonID rdfs:subPropertyOf
                :hasPrimaryKeyAttribute
```

Listing 3.10: Macro 4: Foreign key data property.

```
#define
FOREIGN_KEY_PROPERTY
(FOREIGN_CLASS_NAME, FOREIGN_KEY_PROPERTY_NAME,
SOURCE_CLASS_NAME, PRIMARY_KEY_PROPERTY_NAME)

:FOREIGN_CLASS_NAME
  :linksviaPrimaryKeyForeignKey :SOURCE_CLASS_NAME
:FOREIGN_CLASS_NAME
  :linkForeignKey :FOREIGN_KEY_PROPERTY_NAME
:SOURCE_CLASS_NAME
  :linkPrimaryKey :PRIMARY_KEY_PROPERTY_NAME
-----------------------------------------------------
:FOREIGN_CLASS_NAME  = Foreign class name of
the foreign key property.
:FOREIGN_KEY_PROPERTY_NAME = Name of
```

```
the foreign key property.
:SOURCE_CLASS_NAME = Parent class name of
the foreign key property.
:PRIMARY_KEY_PROPERTY_NAME = Name of the primary key
property of the source class that maps
foreign key property.
-------------------------------------------------------
Suppose we have a class called "PersonWAR."
In "PersonWAR" class, "hasPersonID" is
primary key data property. We have another
class called "ActivityWAR" and "hasPersonNumber"
is a foreign key data property in the "ActivityWAR"
class (ActivityWAR class's "hasPersonNumber"
corresponds to "hasPersonID" in the "PersonWAR" class).
Hence in "ActivityWAR" class, we can use the above
mentioned macro to express foreign key relationship

Example:

FOREIGN_KEY_PROPERTY(ActivityWAR, hasPersonNumber,
PersonWAR, hasPersonID)

Means:

:ActivityWAR
    :linksviaPrimaryKeyForeignKey   :PersonWAR
:ActivityWAR
    :linkForeignKey                 :hasPersonNumber
:PersonWAR
    :linkPrimaryKey                 :hasPersonID

Implementation:

In implementation level we are achieving
"linksviaPrimaryKeyForeignKey" relationship
like following:

Let "_:x", and  "_:y" are blank nodes.

_:x     rdf:type                rdf:Statement
_:x     rdf:subject             :ActivityWAR
_:x     rdf:predicate
                    :linksviaPrimaryKeyForeignKey
_:x     rdf:object              :PersonWAR
_:x     :linkForeignKey         :hasPersonNumber
_:y     rdf:type                rdf:Statement
_:y     rdf:subject             _:x
_:y     rdf:predicate           :linkForeignKey
_:y     rdf:object              :hasPersonNumber
_:y     :linkPrimaryKey         :hasPersonID
```

Note 4: Our *linksviaPrimaryKeyForeignKey* connects primary and foreign key data properties. To the best of our knowledge, there is no explicit property available in RDF/OWL to create a relationship between two data properties. Therefore, we achieve this by using the RDF reification technique. In Listing 3.10, we show a detailed example. In

Section 3.4 we provide a sample SPARQL implementation used for mapping file creation.

Often, a property of a class is a constituent of the composite key and also maps into a primary key of another class. For example, the property *hasPersonID* acts as a primary key data property to the *Person* class, and *hasPersonNumber* of the *Activity* class maps to *hasPersonID* as a foreign key data property. However, *hasPersonNumber* is a part of the composite key in the *Activity* class.

Listing 3.11: Macro 5: Foreign key data property used as a part of the composite key.

```
#define
PRIMARY_FOREIGN_KEY_PROPERTY(PROPERTY_NAME)

:PROPERTY_NAME rdfs:subPropertyOf
        :hasPrimaryForeignKeyAttribute
-------------------------------------------
Where:
PROPERTY_NAME = Name of the foreign key attribute
use as a part of composite key of a class.
-------------------------------------------
Example:

Suppose "hasPersonNumber" is a foreign key
data property in "Activity" class.
The property "hasPersonNumber" is a part
of the composite key in "Activity" class.
Hence in "Activity" class we can use
above macro like following to represent it.


PRIMARY_FOREIGN_KEY_PROPERTY(hasPersonNumber)

Means:

:hasPersonNumber    rdfs:subPropertyOf
            :hasPrimaryForeignKeyAttribute
```

CNEW Logical Schema

We now describe the CNEW logical level schema by using logical level classes, macros, and the necessary data properties. The logical level schema explains how we link various classes mentioned in the conceptual schema. To do so, we borrow the primary key, foreign key, and composite key concepts from the field of relational databases.

We have already introduced logical level classes and macros. Now we are introducing *hasPrimaryKeyAttribute*, *linksviaPrimaryKeyForeignKey*, *hasPrimaryForeignKeyAttribute*, *linkPrimaryKey*, and *linkForeignKey* data properties to the logical schema. The *hasPrimaryKeyAttribute* data property helps us to uniquely identify a class instance. We use the *linksviaPrimaryKeyForeignKey* to represent primary key and foreign key relationships

between two classes. In some cases, we need a composite key to identify a class instance uniquely. In our situation, a composite key can be defined in the following two ways: i) by combining a collection of class properties that solely belong to the class and other classes (foreign key), or ii) by combining a collection of class properties coming from other classes (foreign key). The *hasPrimaryForeignKeyAttribute* data property assists us in determining which foreign key data property to use as a part of the composite key (all foreign keys are not used in a composite key). For example, *hasActivityID* is the primary key in the *ActivityWAR* class, but *hasPersonNumber* is the foreign key in the same class, and maps to *hasPersonID* of the *PersonWAR* class. We need both *hasActivityID* and *hasPersonNumber* to identify *ActivityWAR* class instances uniquely. In this case, we need both *hasPrimaryKeyAttribute* and *hasPrimaryForeignKeyAttribute* data properties. However, to define the *Configuration* class composite key, we need *hasStudyNumber*, *hasCellNumber*, and *hasRegionNumber* data properties, and they are foreign keys in the *Configuration* class. They map to *EpiStudy*, *Cell*, and *Region* classes primary key data properties. Hence, in this case, we need only the *hasPrimaryForeignKeyAttribute* data property to determine the composite key. Therefore, based on the requirements both the *hasPrimaryKeyAttribute* and *hasPrimaryForeignKeyAttribute* data properties or only *hasPrimaryForeignKeyAttribute* are used to define the composite key.

We use two special data properties only for primary and foreign key link implementation, and they are: i) *linkPrimaryKey* – which represents the primary key data property of a class, and ii) *linkForeignKey* – which represents the foreign key data property of a class (see Listing 3.10).

Listing 3.12: Primary key, foreign key, and composite key relationship model using the macro expression.

```
DATA_PROPERTY   (hasPrimaryKeyAttribute)                    1
DATA_PROPERTY   (linksviaPrimaryKeyForeignKey)              2
DATA_PROPERTY   (hasPrimaryForeignKeyAttribute)             3
DATA_PROPERTY   (linkPrimaryKey)                            4
DATA_PROPERTY   (linkForeignKey)                            5
```

Note 5: In our schema, the terminology *Attribute* refers to the Class property. Therefore, *hasPrimaryKeyAttribute*, *linksviaPrimaryKeyForeignKey*, *hasPrimaryForeignKeyAttribute*, *linkPrimaryKey*, and *linkForeignKey* indicate various type of class properties.

In Listing 3.13 we present synthetic population classes in the logical schema.

Listing 3.13: Synthetic population classes in the logical schema.

```
CLASS_AND_PROPERTY (HouseholdWAR,
[hasHouseholdID,...])
PRIMARY_KEY_PROPERTY (hasHouseholdID)
.................................................
-------------------------------------------------
CLASS_AND_PROPERTY (PersonWAR,
[hasPersonID, hasAge,...])
PRIMARY_KEY_PROPERTY (hasPersonID)
DATA_PROPERTY(hasAge)
.................................................
-------------------------------------------------
CLASS_AND_PROPERTY (ActivityWAR,
[hasActivityID, hasDuration, hasPersonNumber...])
PRIMARY_KEY_PROPERTY (hasActivityID)
PRIMARY_KEY_PROPERTY (hasDuration)
FOREIGN_KEY_PROPERTY (ActivityWAR, hasPersonNumber,
PersonWAR, hasPersonID)
PRIMARY_FOREIGN_KEY_PROPERTY(hasPersonNumber)
.................................................
-------------------------------------------------
CLASS_AND_PROPERTY (LocationWAR,
[hasLocationID, hasPersonIdentificationNumber,
hasActID, ...]
PRIMARY_KEY_PROPERTY(hasLocationID)
FOREIGN_KEY_PROPERTY(
LocationWAR, hasPersonIdentificationNumber,
PersonWAR, hasPersonID)
FOREIGN_KEY_PROPERTY(LocationWAR, hasActID,
ActivityWAR, hasActivityID)
PRIMARY_FOREIGN_KEY_PROPERTY(
hasPersonIdentificationNumber)
PRIMARY_FOREIGN_KEY_PROPERTY(hasActID)
.................................................
```

In Listing 3.14 we present study related classes in the logical schema.

Listing 3.14: Study related classes in the logical schema.

```
CLASS_AND_PROPERTY (Region, [hasRegionID,...])
PRIMARY_KEY_PROPERTY (hasRegionID)
.....
-----------------------------------------
CLASS_AND_PROPERTY (DiseaseModel,
[hasDiseaseModelID,...])
PRIMARY_KEY_PROPERTY (hasDiseaseModelID)
.....
-----------------------------------------
CLASS_AND_PROPERTY (SimulationEngine,
[hasSimulationEngineID,...])
PRIMARY_KEY_PROPERTY (hasSimulationEngineID)
.....
-----------------------------------------
CLASS_AND_PROPERTY (InitialCondition,
[hasInitialConditionID,...])
PRIMARY_KEY_PROPERTY (hasInitialConditionID)
.....
-----------------------------------------
CLASS_AND_PROPERTY
(EpiStudy, [hasStudyID,
hasRegionIdentificationNumber,
hasDiseaseModelIdentificationNumber,
hasSimulationEngineIdentificationNumber,
hasInitialConditionIdentificationNumber, ...])
PRIMARY_KEY_PROPERTY (hasStudyID)
FOREIGN_KEY_PROPERTY (EpiStudy,
hasRegionIdentificationNumber,
Region, hasRegionID)
FOREIGN_KEY_PROPERTY (EpiStudy,
hasDiseaseModelIdentificationNumber,
DiseaseModel, hasDiseaseModelID)
FOREIGN_KEY_PROPERTY (EpiStudy,
hasSimulationEngineIdentificationNumber
SimulationEngine, hasSimulationEngineID)
FOREIGN_KEY_PROPERTY (EpiStudy,
hasInitialConditionIdentificationNumber,
InitialCondition, hasInitialConditionID,)
.....
-----------------------------------------
CLASS_AND_PROPERTY (Cell, [hasCellID,
hasEpiStudyIdentificationNumber, ...])
PRIMARY_KEY_PROPERTY (hasCellID)
FOREIGN_KEY_PROPERTY (Cell,
hasEpiStudyIdentificationNumber,
EpiStudy, hasEpiStudyID)
PRIMARY_FOREIGN_KEY_PROPERTY(
hasEpiStudyIdentificationNumber)
.....
```

In Listing 3.15 we present the logical schema for the study configuration class.

Listing 3.15: Configuration class in the logical schema.

```
CLASS_AND_PROPERTY                                                    1
(Configuration, [hasStudyNumber, hasCellNumber,                       2
hasRegionNumber, hasContactNetworkFileLocation, ...])                 3
FOREIGN_KEY_PROPERTY (Configuration,                                  4
hasStudyNumber, EpiStudy, hasStudyID)                                 5
FOREIGN_KEY_PROPERTY (Configuration,                                  6
hasCellNumber, Cell, hasCellID)                                       7
FOREIGN_KEY_PROPERTY (Configuration,                                  8
hasRegionNumber, Region, hasRegionID)                                 9
DATA_PROPERTY(hasContactNetworkFileLocation)                          10
PRIMARY_FOREIGN_KEY_PROPERTY(hasStudyNumber)                          11
PRIMARY_FOREIGN_KEY_PROPERTY(hasCellNumber)                           12
.............................................                         13
```

As mentioned in the conceptual schema, a contact network represents an event where two persons are present at corresponding locations throughout an overlapping period. We can derive a contact network from synthetic population classes. In Listing 3.16, we show the derivation of a contact network using SPARQL-like syntax. It has *hasPersonID, hasFirstPerson, hasSecondPerson, hasFirstPersonActivityType, hasSecondPersonActivityType, hasActivityLocation*, and *duration* relationships. *hasPersonID* represents a person's identification number. *hasFirstPerson* and *hasSecondPerson* represent two different individual identification numbers who are in contact. *hasFirstPersonActivityType* and *hasSecondPersonActivityType* represent the contacted individuals' activity type (home, work, shopping, school, others). The *duration* relation expresses contact duration for a particular activity. The exact time range of contact is also explicitly maintained (not shown here) in the model.

Listing 3.16: Contact network derivation.

```
CONSTRUCT                                                                    1
 {                                                                           2
    :ContactNetwork                                                          3
        :hasFirstPerson                 ?pid                                 4
        :hasFirstPersonActivityType     ?pa                                  5
        :hasSecondPerson                ?qid                                 6
        :hasSecondPersonActivityType    ?qa                                  7
        :duration                       ?duration_time                       8
 }                                                                           9
FROM                                                                         10
 {                                                                           11
    :ActivityWAR                                                             12
 }                                                                           13
WHERE                                                                        14
 {                                                                           15
    ?p      :memberOf               :Activity.                               16
    ?q      :memberOf               :Activity.                               17
    ?p      :hasPersonID            ?pid.                                    18
    ?q      :hasPersonID            ?qid.                                    19
    ?pid    :hasActivityType        ?pa.                                     20
    ?qid    :hasActivityType        ?qa.                                     21
    ?pa     :hasActivityLocation    ?l.                                      22
    ?qa     :hasActivityLocation    ?l.                                      23
    ?pa     :duration               ?ps.                                     24
    ?qa     :duration               ?qs.                                     25
    BIND(abs(?ps-?qs) as ?duration_time)                                     26
                                                                             27
 }                                                                           28
```

Even though we can express a contact network within the FUSED framework, we choose to compute and materialize the contact network and store it externally. This materialized network not only helps the performance of the query but also reduces the computational cost.

Note 6: After derivation, we store the contact network in a file. In this dissertation, we only consider a "person-to-person" contact network. An additional network such as "person-to-location" or others are out of the scope of this dissertation.

In Listing 3.17 we present a contact network in the logical schema.

Note 7: *ContactEdge* stores instance level triples. *hasEdge* relationship connects class and blank node (see *Blank Node* in Section 2.2.2)

Listing 3.17: Contact network class in the logical schema.

```
:ContactNetwork :hasEdge :ContactEdge

CLASS_AND_PROPERTY
(ContactEdge, [hasRegionIDNumber, hasFirstPerson,
hasFirstPersonActivityType, hasSecondPerson,
hasSecondPersonActivityType...])
PRIMARY_KEY_PROPERTY (hasFirstPersonActivityType)
PRIMARY_KEY_PROPERTY (hasSecondPersonActivityType)
FOREIGN_KEY_PROPERTY (ContactEdge,
hasRegionIDNumber, Region, hasRegionID)
FOREIGN_KEY_PROPERTY (ContactEdge,
hasFirstPerson, PersonWAR, hasPersonID)
FOREIGN_KEY_PROPERTY (ContactEdge,
hasSecondPerson, PersonWAR, hasPersonID)
PRIMARY_FOREIGN_KEY_PROPERTY(hasRegionIDNumber)
PRIMARY_FOREIGN_KEY_PROPERTY(hasFirstPerson)
PRIMARY_FOREIGN_KEY_PROPERTY(hasSecondPerson)
.............................................
```

Note 8: In the contact network *hasFirstPersonActivityType* and *hasSecondPersonActivityType* are properties used with other properties to uniquely identify an element. The above mentioned properties are not foreign keys of the *ActivityWAR* class.

The EpiStudy simulation produces a dendrogram, a directed graph, where nodes represent individuals and edges represent disease transmission between two individuals. The edge direction goes from the infector individual to the infectee individual. Basically, the dendrogram contains who infected whom information. The dendrogram contains exposure day and replicate number information as well. To link dendrogram data with CNEW datasets, we add EpiStudy Unique Number, and Cell Unique Number, to the dendrogram dataset. In Listing 3.18, we present a dendrogram schema.

Listing 3.18: Dendrogram class in the logical schema.

```
:Dendrogram :hasEdge :DendrogramContactEdge

CLASS_AND_PROPERTY
(DendrogramContactEdge, [hasEpiStudyUniqueNumber,
hasCellUniqueNumber, hasInfecteePID,
hasInfectorPID, hasReplicateID,
hasExposureDay, ...])
PRIMARY_KEY_PROPERTY (hasReplicateID)
FOREIGN_KEY_PROPERTY (DendrogramContactEdge,
hasEpiStudyUniqueNumber, EpiStudy, hasEpiStudyID)
FOREIGN_KEY_PROPERTY (DendrogramContactEdge,
hasCellUniqueNumber, Cell, hasCellID)
FOREIGN_KEY_PROPERTY (DendrogramContactEdge,
hasInfecteePID, PersonWAR, hasPersonID)
DATA_PROPERTY(hasExposureDay)
PRIMARY_FOREIGN_KEY_PROPERTY(hasEpiStudyUniqueNumber)
PRIMARY_FOREIGN_KEY_PROPERTY(hasCellUniqueNumber)
PRIMARY_FOREIGN_KEY_PROPERTY(hasInfecteePID)
.............................................
```

We derived a named graph from *Dendrogram* for each replicate. Derivation is available in Listing 3.19. This named graph is useful to find the disease transmission chain through the population (see Section 3.3 for detail). The *getInfectedBy* relation shows who infected whom. The Infectee-Infector graph schema is available in Listing 3.20.

Listing 3.19: Infectee-Infector graph drivation.

```
CONSTRUCT
{
   ?infectee :getInfectedBy ?infector
}

FROM
{
   :Dendrogram
}

WHERE
{
   ?s :hasInfecteePID ?infectee.
   ?s :hasInfectorPID ?infector
}
```

Listing 3.20: Infectee-Infector named graph schema.

```
:infecteePersonID :getInfectedBy :infectorPersonID
```

In Listing 3.21, we present a logical level implementation of the example query (*find two people who live in the same large household and also work in the same location*) mentioned regarding the conceptual schema. In a synthetic population, a person is recognized by an ID number. Hence we need to use the *hasPersonID* attribute to find the person ID. In our schema the *hasPersonID* attribute is a primary key attribute for the *PersonWAR* class.

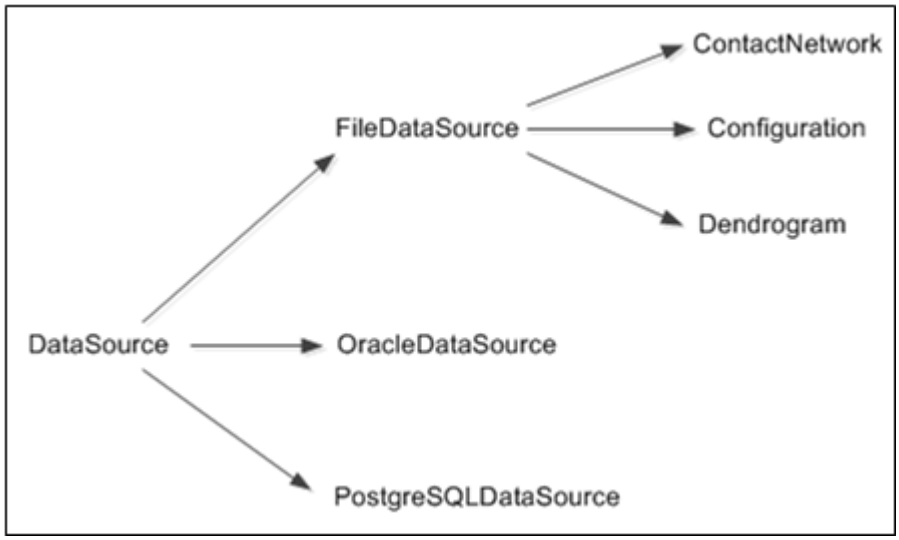


Figure 3.7: Physical level data sources. CNEW datasets are distributed across multiple data sources such as OracleDataSource, PostgreSQLDataSource, and FileDataSource. The contact network, configuration, and dendrogram are stored in FileDataSource, while others are stored in OracleDataSource and PostgreSQLDataSource.

Listing 3.21: Example query (*find two people who live in the same household and also work in the same location*) at the logical level.

```
SELECT                                                                  1
                                                                        2
?pid1 ?pid2                                                             3
                                                                        4
WHERE                                                                   5
{                                                                       6
?person1    :hasPersonID      ?pid1.                                    7
?person1    :hasHouseholdID   ?hid.                                     8
?person2    :hasPersonID      ?pid2.                                    9
?person2    :hasHouseholdID   ?hid.                                     10
?hid        :hasSize          "Large".                                  11
?person1    :hasActivity    ?activity1.                                 12
?person2    :hasActivity    ?activity2.                                 13
?activity1  :hasActivityType     "work activity".                       14
?activity2  :hasActivityType     "work activity".                       15
?activity1  :hasActivityLocation ?work_location.                        16
?activity2  :hasActivityLocation ?work_location                         17
}                                                                       18
```

3.1.3 Physical Schema

We now turn our attention to physical schema, which capture the storage and formatting aspects of CNEW datasets.

Physical Schema Macros

In Listings 3.22, 3.23, 3.24, 3.25, 3.26, 3.27, and 3.28, we present subclass, column properties, class, and property storage macros. We use these macros to define the schema pattern of subclass creation and different column properties creation. Macros also show class and property storage.

Listing 3.22: Macro 6: Subclass relation.

```
#define SUBCLASSOF(SUBCLASS, SUPERCLASS)

:SUBCLASS
    rdfs:subClassOf :SUPERCLASS
-------------------------------------------
Where:
SUBCLASS = Subclass name.
SUPERCLASS = Superclass name.
-------------------------------------------
Example:

SUBCLASSOF(OracleDataSource, DataSource)

Means:

:OracleDataSource
    rdfs:subClassOf :DataSource
```

Listing 3.23: Macro 7: Table column property.

```
#define COLUMN_PROPERTY(COLUMN_NAME)

:COLUMN_NAME
    rdfs:subPropertyOf :hasColumn
-------------------------------------------
Where:
COLUMN_NAME = Name of the table column.
-------------------------------------------
Example:

COLUMN_PROPERTY(hasYear)

Means:

:hasYear
    rdfs:subPropertyOf :hasColumn
```

Listing 3.24: Macro 8: File column property.

```
#define FILE_COLUMN_PROPERTY(FILE_COLUMN_NAME)

:FILE_COLUMN_NAME
    rdfs:subPropertyOf :hasFileColumn
-------------------------------------------
Where:
COLUMN_NAME = Name of the file column.
-------------------------------------------
Example:

FILE_COLUMN_PROPERTY(hasColumnFirstPerson)

Means:

:hasColumnFirstPerson
    rdfs:subPropertyOf :hasFileColumn
```

Listing 3.25: Macro 9: Key value file column property.

```
#define KEY_VALUE_FILE_COLUMN_PROPERTY(KEY_NAME)

:KEY_NAME
    rdfs:subPropertyOf :hasKeyValueFileColumn
-------------------------------------------
Where:
KEY_NAME = Name of the file keys.
-------------------------------------------
Example:

KEY_VALUE_FILE_COLUMN_PROPERTY
(hasColumnContactNetworkFileLocation)

Means:

:hasColumnContactNetworkFileLocation
    rdfs:subPropertyOf :hasKeyValueFileColumn
```

Listing 3.26: Macro 10: All column property.

```
#define All_COLUMN_PROPERTY(COLUMN_PROPERTY)

:COLUMN_PROPERTY
    rdfs:subPropertyOf :hasAllColumn
-----------------------------------------------
Where:
COLUMN_PROPERTY = Name of the columns property.
-----------------------------------------------
Example:

All_COLUMN_PROPERTY(hasColumn)

Means:

:hasColumn
    rdfs:subPropertyOf :hasAllColumn
```

Listing 3.27: Macro 11: Class physical storage.

```
#define CLASS_STORAGE (SCHEMA_CLASS_NAME,
PHYSICAL_STORAGE_CLASS_NAME)

:SCHEMA_CLASS_NAME
    :hasStorage :PHYSICAL_STORAGE_CLASS_NAME
-------------------------------------------
Where:
SCHEMA_CLASS_NAME = Name of the FUSED schema class.
PHYSICAL_STORAGE_CLASS_NAME =
Name of the FUSED storage class.
-------------------------------------------
Example:

CLASS_STORAGE(student, studentTable)

Means:

:student :hasStorage :studentTable
```

Listing 3.28: Macro 12: Property storage.

```
#define PROPERTY_STORAGE(CLASS_PROPERTY_NAME,
PHYSICAL_STORAGE_PROPERTY_NAME)

:CLASS_PROPERTY_NAME
    :storedAt :PHYSICAL_STORAGE_PROPERTY_NAME
-------------------------------------------
Where:
CLASS_PROPERTY_NAME =
Name of the schema class property.
PHYSICAL_STORAGE_PROPERTY_NAME =
Name of the physical storage class property.
-------------------------------------------
Example:

PROPERTY_STORAGE (hasID, hasIdentificationNumber)

Means:

:hasID :storedAt :hasIdentificationNumber
```

CNEW Physical Schema

DataSource

In FUSED, we model storage with classes for various storage media types (database schema, relational tables, CSV and configuration files, etc.) and *storedAt* relation.

Listing 3.29: Physical schema classes.

```
:DataSource, :Table, :FileColumn, :KeyValueFileColumn,
:AllColumn
```

DataSource asserts various datasets. Its subclasses *OracleDataSource*, *PostgreSQLDataSource*, and *FileDataSource* define the respective media where the dataset is stored (Listing 4.27). We also present *DataSource* class property examples in Listing 4.28 and Fig. 3.7.

Listing 3.30: Datasource class and subclasses.

```
SUBCLASSOF (OracleDataSource, DataSource)
SUBCLASSOF (PostgreSQLDataSource, DataSource)
SUBCLASSOF (FileDataSource, DataSource)
```

Listing 3.31: Datasource class with property example.

```
CLASS_AND_PROPERTY
(OracleDataSource, [hasUsername, hasPassword, ...])
.....................................
```

In FUSED, relational and file data sources store values in different formats. For example *Table* and some *Files* store data in column format. Another *File* stores data in key-value pair

format. To model this aspect of the storage we introduce necessary classes and properties in Listing 4.29.

Listing 3.32: Column classes with property example.

```
CLASS_AND_PROPERTY (Table, [hasColumn])
CLASS_AND_PROPERTY
(FileColumn, [hasFileColumn])
CLASS_AND_PROPERTY
(KeyValueFileColumn, [hasKeyValueFileColumn])
CLASS_AND_PROPERTY (AllColumn, [hasAllColumn])
```

We introduce a property called *hasAllColumn*. Various types of physical data storage columns are sub-properties of *hasAllColumn* (Listing 4.30). The purpose of the *hasAllColumn* property is to express the attributes of the datasets. We link these attributes to the corresponding data property attribute. Hence, property *hasDataPropertyAttribute* maintains *storedAt* relationship with *hasAllColumn* (Listing 3.34). An example is "hasAge storedAt hasColumnAge." *hasAge* and *hasColumnAge* are sub-properties of *hasDataPropertyAttribute* and *hasAllColumn*.

Listing 3.33: Various column properties.

```
All_COLUMN_PROPERTY (hasColumn)
All_COLUMN_PROPERTY (hasFileColumn)
All_COLUMN_PROPERTY (hasKeyValueFileColumn)
```

Listing 3.34: Attribute *storedAt* relation.

```
:hasDataPropertyAttribute :storedAt :hasAllColumn
:hasAllColumn rdf:type owl:DatatypeProperty
```

The *Table* class represents datasets stored in a relational database table format, and the *hasColumn* property captures table column information. We are not considering relational database primary key and foreign key constraints in the *Table* class (because these are maintained at the logical layer).

Listing 3.35: Table class, subclasses, and relations.

```
SUBCLASSOF (OracleTable, Table)
SUBCLASSOF (PostgreSQLTable, Table)

:OracleDataSource   :hasTable   :OracleTable
.........................................
```

ContactNetworkFile, *ConfigurationFile*, and *DendrogramFile* are subclasses of the *FileDataSource* and capture contact network, configuration, and dendograph parts of the CNEW file datasets.

Listing 3.36: File class and subclasses.

```
SUBCLASSOF(ContactNetworkFile, FileDataSource)
SUBCLASSOF(ConfigurationFile, FileDataSource)
SUBCLASSOF(DendrogramFile, FileDataSource)
```

In SIBEL, synthetic populations are stored in Oracle database tables. We model a synthetic population in Listing 3.37.

Listing 3.37: Synthetic population table subclasses.

```
SUBCLASSOF(HouseholdWAROracleTable, OracleTable)
.....
```

In the following we model study related table subclasses (Listing 3.38).

Listing 3.38: Study related table subclass.

```
SUBCLASSOF(EpiStudyOracleTable, OracleTable)
.....
```

Broadly, a study can be classified into the following two subclasses: i) EpiFast [112], and ii) EpiSimdemics [116]. The EpiFast study examines person-to-person contact disease progression whereas EpiSimdemic examines person-to-location disease propagation. As mentioned in ***Note 6***, we are not considering "EpiSimdemics" in our schema design.

Listing 3.39: Study subclasses.

```
SUBCLASSOF(EpiFastEpiStudy, EpiStudy)
SUBCLASSOF(EpiSimdemicsEpiStudy, EpiStudy)
```

Synthetic Population

In Listings 3.40, 3.41, 3.42, 3.43, and 3.44 we present synthetic population physical schema. At the physical level two tables store location data: *HomeLocationWAROracleTable* and *OutsideLocationWAROracleTable*. If activity ID "1" means "home activity" then data is stored in the *HomeLocationWAROracleTable*, otherwise data is stored in the *OutsideLocationWAROracleTable*. This concept is depicted in Listings 3.43 and 3.44.

Household

Listing 3.40: Synthetic population physical schema: HouseholdWAR class.

```
CLASS_AND_PROPERTY (HouseholdWAR,
[hasHouseholdID, hasZipcode, ...])
DATA_PROPERTY(hasGender)
.....
-----
CLASS_AND_PROPERTY (HouseholdWAROracleTable,
[hasColumnHouseholdID, hasColumnZipcode, ...])
COLUMN_PROPERTY(hasColumnHouseholdID)
COLUMN_PROPERTY(hasColumnZipcode)
.....
-----
CLASS_STORAGE(HouseholdWAR,
HouseholdWAROracleTable)
.....
-----
PROPERTY_STORAGE (hasHouseholdID,
hasColumnHouseholdID)
PROPERTY_STORAGE (hasZipcode,
hasColumnZipcode)
.....
```

Person

Listing 3.41: Synthetic population physical schema: PersonWAR class.

```
CLASS_AND_PROPERTY (PersonWAR, [hasPersonID, ...])
PRIMARY_KEY_PROPERTY (hasPersonID)
.....
-----
CLASS_AND_PROPERTY (PersonWAROracleTable,
[hasColumnPersonID, ...])
COLUMN_PROPERTY(hasColumnPersonID)
.....
-----
CLASS_STORAGE(PersonWAR, PersonWAROracleTable)
.....
-----
PROPERTY_STORAGE (hasPersonID, hasColumnPersonID)
.....
```

Activity

Listing 3.42: Synthetic population physical schema: ActivityWAR class.

```
CLASS_AND_PROPERTY (ActivityWAR, [hasActivityID, ...])
PRIMARY_KEY_PROPERTY (hasActivityID)
.................................................
-------------------------------------------------
CLASS_AND_PROPERTY (ActivityWAROracleTable,
[hasColumnActivityID, ...])
COLUMN_PROPERTY(hasColumnActivityID)
.................................................
-------------------------------------------------
CLASS_STORAGE(ActivityWAR, ActivityWAROracleTable)
.................................................
-------------------------------------------------
PROPERTY_STORAGE (hasActivityID, hasColumnActivityID)
.................................................
```

Location

Listing 3.43: Synthetic population physical schema: HomeLocationWAR class.

```
CLASS_AND_PROPERTY (LocationWAR,
[hasLocationID, ...])
PRIMARY_KEY_PROPERTY(hasLocationID)
.................................................
-------------------------------------------------
CLASS_AND_PROPERTY (HomeLocationWAROracleTable,
[hasColumnLocationID, ...])
COLUMN_PROPERTY(hasColumnLocationID)
.................................................
-------------------------------------------------
CLASS_STORAGE(LocationWAR, HomeLocationWAROracleTable)
.................................................
-------------------------------------------------
PROPERTY_STORAGE (hasLocationID, hasColumnLocationID)
.................................................
```

Listing 3.44: Synthetic population physical schema: OutsideLocationWAR class.

```
.................................................
PROPERTY_STORAGE (LocationWAR,
OutsideLocationWAROracleTable)
.................................................
```

EpiStudy

We now turn our attention to experiment study datasets. Listing 3.45 shows the *EpiStudy* class model. Other class models (Region, DiseaseModel, SimulationEngine, InitialCondition,

Cell) are similar, and therefore omitted.

Listing 3.45: Study related classes in physical schema.

```
CLASS_AND_PROPERTY (EpiStudy, [hasStudyID, ...])
PRIMARY_KEY_PROPERTY (hasStudyID)
.................................................
-------------------------------------------------
CLASS_AND_PROPERTY (EpiStudyOracleTable,
[hasColumnStudyID, ...])
COLUMN_PROPERTY(hasColumnStudyID)
.................................................
-------------------------------------------------
CLASS_STORAGE(EpiStudy, EpiStudyOracleTable)
.................................................
-------------------------------------------------
PROPERTY_STORAGE (hasStudyID, hasColumnStudyID)
.................................................
```

Study Configuration

In Listing 3.46 we present the study configuration class physical schema model.

Listing 3.46: Study configuration class in physical schema.

```
CLASS_AND_PROPERTY (Configuration,
[hasContactNetworkFileLocation, ...])
DATA_PROPERTY (hasContactNetworkFileLocation)
.................................................
-------------------------------------------------
CLASS_AND_PROPERTY (ConfigurationFile,
[hasColumnContactNetworkFileLocation, ...])
KEY_VALUE_FILE_COLUMN_PROPERTY
(hasColumnContactNetworkFileLocation)
.................................................
-------------------------------------------------
CLASS_STORAGE(Configuration, ConfigurationFile)
.................................................
-------------------------------------------------
PROPERTY_STORAGE (hasContactNetworkFileLocation,
hasColumnContactNetworkFileLocation)
.................................................
```

Contact Network

In Listing 3.47 we present the contact network class physical schema model.

Listing 3.47: Contact network class in physical schema.

```
:ContactNetwork :hasEdge :ContactEdge

CLASS_AND_PROPERTY (ContactEdge,
[hasFirstPerson, hasFirstPersonActivityType, ...])
PRIMARY_KEY_PROPERTY (hasFirstPersonActivityType)
.............................................
---------------------------------------------
CLASS_AND_PROPERTY (ContactNetworkFile,
[hasColumnFirstPerson, ...])
FILE_COLUMN_PROPERTY(hasColumnFirstPerson)
.............................................
---------------------------------------------
CLASS_STORAGE(ContactEdge, ContactNetworkFile)
.............................................
---------------------------------------------
PROPERTY_STORAGE (hasFirstPerson,
hasColumnFirstPerson)
.............................................
```

Dendrogram

In Listing 3.48 we present the dendrogram class physical schema model.

Listing 3.48: Dendrogram class in physical schema.

```
:Dendrogram :hasEdge :DendrogramContactEdge

CLASS_AND_PROPERTY (DendrogramContactEdge,
[hasReplicateID, ...])
PRIMARY_KEY_PROPERTY (hasReplicateID)
.............................................
---------------------------------------------
CLASS_AND_PROPERTY (DendrogramFile,
[hasColumnReplicateID, ...])
FILE_COLUMN_PROPERTY(hasColumnReplicateID)
.............................................
---------------------------------------------
CLASS_STORAGE(DendrogramContactEdge, DendrogramFile)
.............................................
---------------------------------------------
PROPERTY_STORAGE (hasReplicateID,
hasColumnReplicateID)
.............................................
```

The physical level exhibits the underlying representation of the CNEW datasets. We are treating all of our physical level file datasources as tables. We present the physical level of

our example query (*find two people who live in the same household and also work in the same location*) in Listing 3.49. At the physical level, "work activity" is represented by the numeric value of "2." *Activity* class stores *Person* information. Also, location data are stored in two classes (*home_location, outside_location*).

Listing 3.49: Example query (*find two people who live in the same large household and also work in the same location*) on the physical level.

```
SELECT

?pid1 ?pid2

WHERE
{
?person1    :person_pid ?pid1.
?person1    :person_hid ?hid.
?person2    :person_pid ?pid2.
?person2    :person_hid ?hid.
?hid        :hasSize
                '1'^^xsd:decimal.
?activity1 :activities_pid ?person1.
?activity2 :activities_pid ?person2.
?activity1 :activities_purpose
                '2'^^xsd:decimal.
?activity2 :activities_purpose
                '2'^^xsd:decimal.
?activity1 :activities_location
                ?person1_activity_location.
?activity2 :activities_location
                ?person2_activity_location.
?person1_activity_location
           :outside_location_id
                ?work_loc_id.
?person2_activity_location
           :outside_location_id
                ?work_loc_id
}
```

Representation of CNEW datasets using the physical schema

We now illustrate how we use the proposed physical schema to represent datasets stored across dataservers and files. As an example, we will consider the storage subsystem and datasets for CNEW. Pictorial views of the schema are presented in Figs. 3.8, 3.9, and 3.10.

Note 9: We are considering *rdfs:Class* (*owl:Class* is a subclass of *rdfs:Class*) as a set [117, 118]. Fig. 3.8 shows the relationship between this collection of classes and instances. In RDF it is achievable in various ways. Here we omit implementation detail for the sake of brevity.

RI_Pop and *RI_Study* are instances of *OracleDataSource*. *ContactNetworkFile*, *ConfigurationFile*, and *DendrogramFile* are subclasses of *FileDataSource* (in the Listing *RI* means *Region Instance*).

Listing 3.50: Data source instances.

```
:RI_Pop     rdf:type    :OracleDataSource
:RI_Study   rdf:type    :OracleDataSource
:RI_ContactNetwork_File
            rdf:type    :ContactNetworkFile
:RI_Configuration_File
            rdf:type    :ConfigurationFile
:RI_Dendrogram_File
            rdf:type    :DendrogramFile
.............................................
```

In the following, we provide attribute instances for the example datasource.

Listing 3.51: *Datasource* attributes instance example.

```
CLASS_AND_PROPERTY (RI_Pop, [hasUsername, hasPassword,
hasDsn, hasdriver...])
DATA_PROPERTY(hasUsername)
DATA_PROPERTY(hasPassword)
DATA_PROPERTY(hasDsn)
DATA_PROPERTY(hasdriver)
.....................................
```

In Listing 3.52, we provide an *AdminRegion* synthetic population example.

Listing 3.52: *AdminRegion* synthetic population subclasses.

```
:RI_Person
.........................................
-----------------------------------------
:RI_Household
.........................................
-----------------------------------------
:RI_Activity
.........................................
-----------------------------------------
:RI_Location
.........................................
```

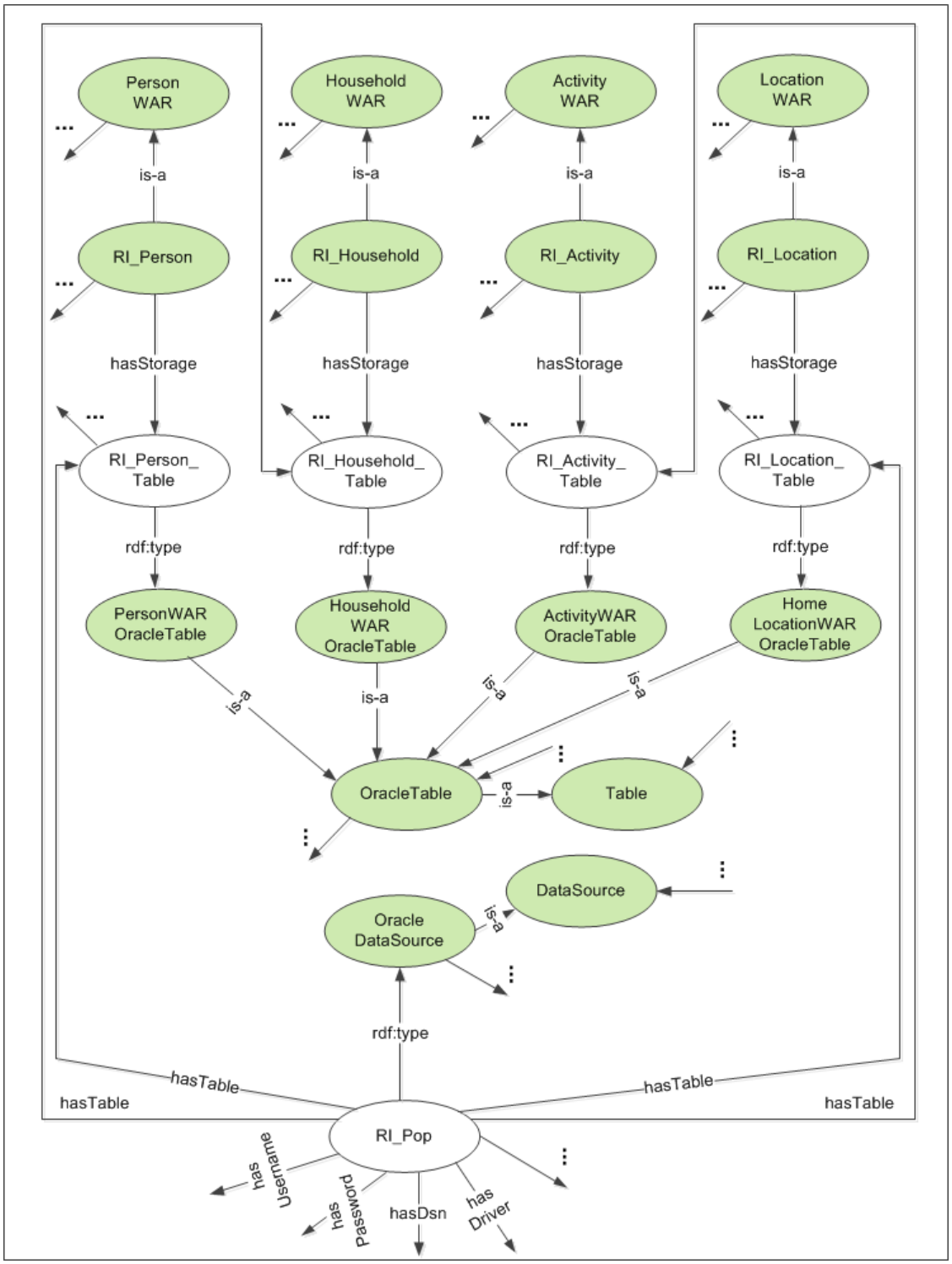


Figure 3.8: Part 1: Partial pictorial view of the synthetic population component of the physical schema. This figure shows how we utilize the proposed physical schema to connect to datasets that are stored in the database. Here, we present physical schema classes with corresponding instance examples. Green ovals represent classes, white ovals denote class instances, and arrows represent object and data properties.

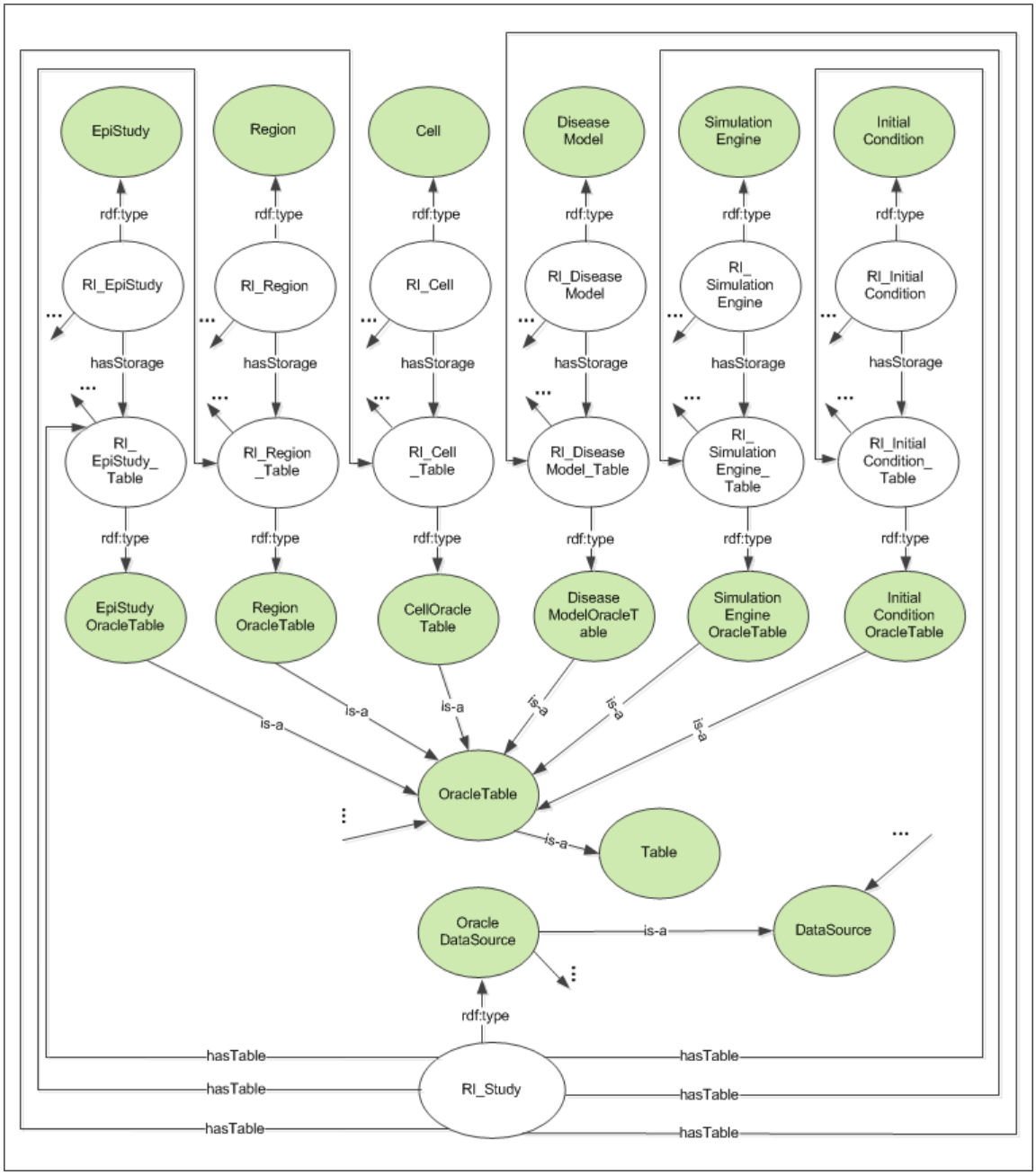


Figure 3.9: Part 2: Partial pictorial view of the study related table part of the physical schema. This figure highlights how we utilize the proposed physical schema to connect to datasets that are stored in the database. Here, we present physical schema classes with corresponding instances. Green ovals represent classes, white ovals denote class instances, and arrows represent object and data properties.

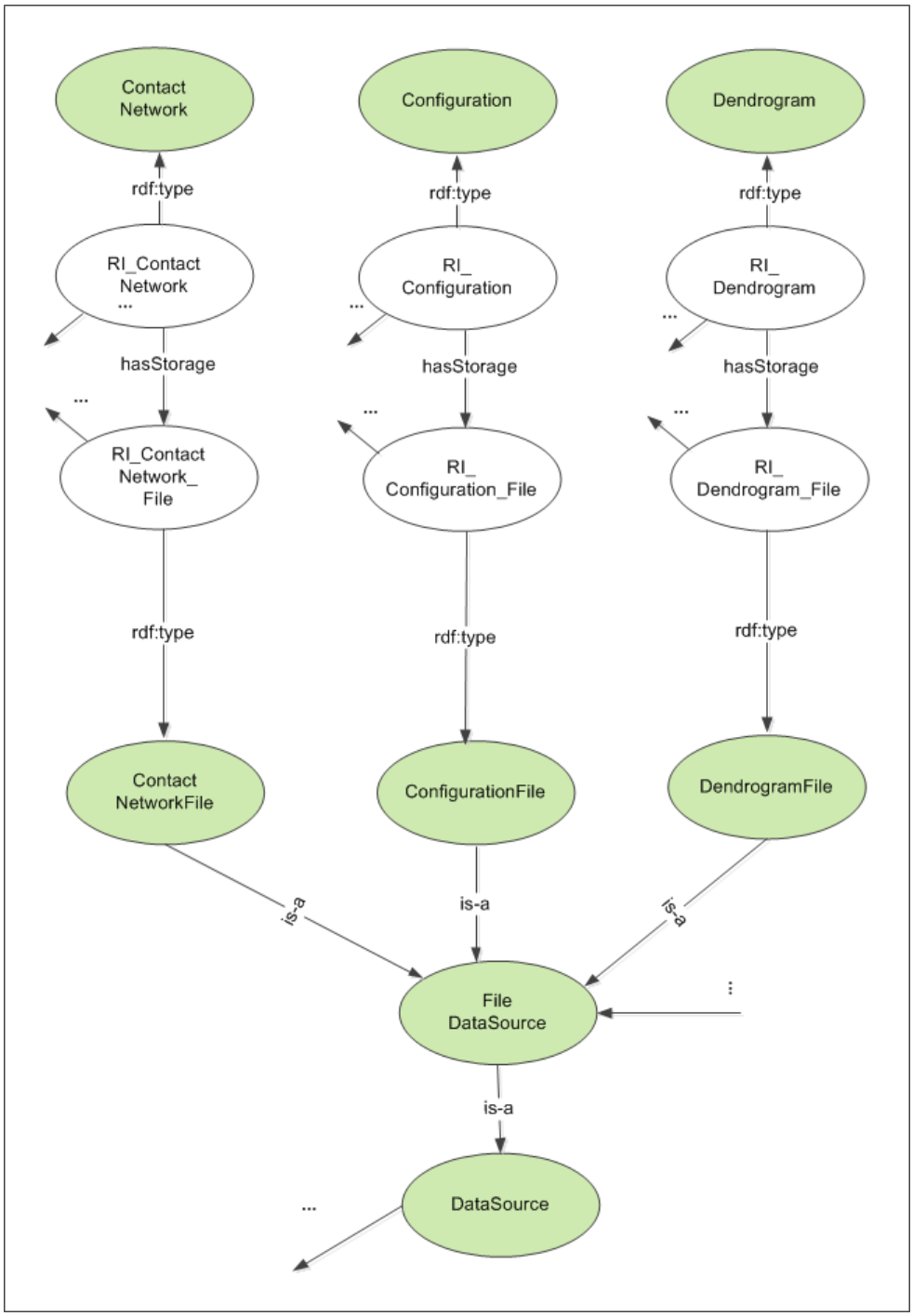


Figure 3.10: Part 3: Partial pictorial view of the study related files (contact network, configuration, and dendrogram) portion of the physical schema. This figure presents how we utilize the proposed physical schema to connect to datasets stored in various files. Here, we present physical schema classes with corresponding instance examples. Green ovals represent classes, white ovals denote class instances, and arrows represents object and data properties.

Synthetic population *Table* classes instances are provided below.

Listing 3.53: Synthetic population *Table* classes instances.

```
:RI_Person_Table
.........................................
-----------------------------------------
:RI_Household_Table
.........................................
-----------------------------------------
:RI_Activity_Table
.........................................
-----------------------------------------
:RI_HomeLocation_Table
.........................................
```

In the following, we show the link between synthetic population classes and their physical storage classes instances.

Listing 3.54: Synthetic population classes and their corresponding table classes relationship (instance level).

```
:RI_Person
        :hasStorage :RI_PersonTable
:RI_Household
        :hasStorage :RI_HouseholdTable
:RI_Activity
        :hasStorage :RI_ActivityTable
:RI_Location
        :hasStorage :RI_HomeLocationTable
.............................................
```

In the following, we provide a computational networked epidemiology study, contact network, configuration, and dendrogram related classes instances example (Listing 3.55). Also, we provide their storage instances example in Listing 3.56.

Listing 3.55: EpiStudy, contact network, configuration, and dendrogram related classes instances.

```
:RI_EpiStudy
.........................................
-----------------------------------------
:RI_ContactNetwork
.........................................
-----------------------------------------
:RI_Configuration
.........................................
-----------------------------------------
:RI_Dendrogram
.........................................
```

Listing 3.56: EpiStudy *Table* , contact network, configuration, and dendrogram *FileDataSource* classes instances.

```
:RI_EpiStudy_Table
.........................................
-----------------------------------------
:RI_ContactNetwork_File
.........................................
-----------------------------------------
:RI_Configuration_File
.........................................
-----------------------------------------
:RI_Dendrogram_File
.........................................
```

In the following, we show the link between study, contact network, configuration, and dendrogram related classes and their physical storage classes (instance level).

Listing 3.57: EpiStudy, contact network, configuration, and dendrogram related classes and their corresponding physical storage classes relationship (instance level).

```
:RI_EpiStudy
        :hasStorage  :RI_EpiStudy_Table
.............................................
:RI_ContactNetwork
            :hasStorage
            :RI_ContactNetwork_File
:RI_Configuration
            :hasStorage
            :RI_Configuration_File
:RI_Dendrogram
            :hasStorage
            :RI_Dendrogram_File
.............................................
```

Various Application Programming Interfaces (APIs) can be developed for FUSED schema instance creation.

3.2 Mapping Across FUSED Schemas

The FUSED schema represent different aspects of CNEW datasets. They progressively encapsulate system and organization level details to present an uncluttered and unified view of the CNEW datasets. However, these schema as presented are themselves disconnected, i.e., there is no mechanism to translate (rewrite) physical level representation to the logical level and from the logical level to the conceptual level.

We now present a mapping framework enabling this translation. It is based upon "D2RQ", a mapping language that allows the creation of an RDF graph from relational datasets. Briefly, D2RQ mapping has the following core parts: i) *Database* represents a database with necessary credential information, ii) *ClassMap* maps an RDFS or OWL class to its database representation (normally mapped to a database table), iii) *uriPattern*

creates URIs by loading values from database columns, and iv) *PropertyBridge* maps an RDF property to one or more database columns [119]. However, it requires a mapping file to create the RDF graph. A D2RQ mapping file stores its mapping in RDF format. The FUSED schema is an RDF graph. Hence, we can create a D2RQ mapping file by executing a SPARQL query on top of the FUSED schema instances. We present example *DataBase*, *ClassMap*, *PropertyBridge* components of a mapping file in Listings 3.58, 3.59, and 3.60.

Listing 3.58: Sample D2RQ mapping file for *DataBase*.

```
.................................................
map:database a d2rq:Database;
        d2rq:jdbcDriver
        "oracle.jdbc.driver.OracleDriver";
        d2rq:jdbcDSN
        "jdbc:oracle:thin:
        @//noldor-db.vbi.vt.edu:1521/
        ndssl.bioinformatics.vt.edu";
        d2rq:username "ABCD";
        d2rq:password "****";
        .
.................................................
```

Listing 3.59: Sample D2RQ mapping file for *ClassMap*.

```
.................................................
map:chicago_person_WAR a d2rq:ClassMap;
        d2rq:dataStorage map:database;
        d2rq:uriPattern "http://ndssl.bi.vt.edu/
            chicago/person/pid#@@person.pid@@";
        d2rq:class vocab:person;
        d2rq:classDefinitionLabel "person";
        .
.................................................
```

Listing 3.60: Sample D2RQ mapping file for *PropertyBridge*.

```
.................................................
map:chicago_person_WAR_age a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:chicago_person_WAR;
        d2rq:property vocab:person_age;
        d2rq:propertyDefinitionLabel "person age";
        d2rq:column "person.age";
        d2rq:datatype xsd:decimal;
        .
.................................................
```

Table 3.1: Chicago mid-flu experiment datasets.

Component	Contained Data	Number of Rows	Representation	Size in MB
Synthetic Population	Person, Household, Activity, Location	110,766,180	Relational Database	11,370
Contact Network	Person-to-Person contact graph	273,526,625	File	8,100
Study File	Study Configuration Information (e.g., contact network location, disease model location, simulation engine, log file location, etc.)	21	File	0.032
Study Database	Contain data and metadata about followings: region, cells, disease model, experiments, initial conditions, simulation engine	86	Relational Database	0.0043
Dendrogram	Infector-to-Infectee graph. Contain disease exposure day and replicate information as well.	8,638	File	0.31

3.3 Experimental Study

To demonstrate the usefulness of our schema we conduct an experiment on a comprehensive CNEW dataset. We collect computational networked epidemiology simulation data from a real-time study performed in the Chicago, USA geographical region. The study simulates influenza transmission in populations via person-person contact, using the EpiFast engine for simulation. Given a contact network, a disease model, and initial conditions, EpiFast rapidly computes many realizations of a stochastic process of disease propagation. For each realization of the propagation, EpiFast computes which individuals are infected and when. We use "AL_25 Mild flu" as a disease model. This disease model infects 10% of the population, and its transmissibility (probability of transmission per minute of contact with an infectious person) is 0.00006 [13]. The initial condition of the study is "20 seeds from school age." The study is replicated 10 times, with each run using different random seeds. Our study data reside in various heterogeneous places, including a relational database and a file system. Experiment dataset detail is available in Table 3.1. We convert all the data into the relational format and store them in Oracle and Postgres databases.

Table 3.2: Chicago mid-flu experiment RDF graph information.

Component	Number of Triples	RDF Graph Generation Time (in Minute)	Virtuoso Loading Time (in Minute)	RDF Graph Size in MB
Synthetic Population	1,680,392,618	258	381	289,000
Contact Network	2,188,213,012	406	418	459,000
Study File	74	0.05	0.01	0.01
Study Database	1,084	0.09	0.07	0.02
Dendrogram	69,115	0.05	0.23	15

3.3.1 Software Used

We use D2RQ version 0.8.1 as a mapping language, Virtuoso Open-Source Edition 7.1.0, SPARQL 1.1 for querying the RDF graph, Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64-bit Production, and Postgres version 8.3.14.

3.3.2 RDF Graph Creation

D2RQ mapping files are created from our schema through the SPARQL language. A mapping file is also an RDF graph. Next, we apply mapping files to our data sources to produce RDF graphs. After generating the RDF graph, it is loaded into Virtuoso to execute queries. RDF graph details are available in Table 3.2.

3.3.3 Complex Queries

Epidemiologists need to execute complex queries over CNEW datasets. We interviewed a number of epidemiologists and asked them about the types of queries they want to run on fragmented SIBEL simulation datasets. In the following, we provide two queries as examples. However, many other queries are possible. Our schema links the various components of the CNEW dataset. Without this linking, it would be a laborious manual/semi-automated process to answer the same queries because of data heterogeneity and formats.

Query 1: Find an infected person in a school at the beginning of the epidemic. Starting from that person discover the transitive disease transmission path length before the epidemic dies out.

We can divide *Query 1* into the following sub-queries:

- *Subquery 1:* Find an infected person in a school at the beginning of the epidemic. (To answer this query we need a contact network and dendrogram sources.)
- *Subquery 2:* How does the disease spread into the population originating from that infected person? (To answer this query we need dendrogram sources.)

We can use the following SPARQL query (Listing 3.61) to find an infected person in a school at the beginning of the epidemic.

Listing 3.61: Query 1 Solution: Find an infected person in a school at the beginning of the epidemic.

```
PREFIX
fused:<http://ndssl.bi.vt.edu/chicago/vocab/>

SELECT  ?o2

FROM <http://ndssl.bi.vt.edu/chicago/>

where{
    ?s fused:dendrogram_infector_pid ?o1.
    ?s fused:dendrogram_infectee_pid ?o2.
    ?s fused:dendrogram_iteration
        '0'^^xsd:decimal.
    ?s fused:dendrogram_exposureday ?e.
    ?s1 fused:contactnetwork_pid1 ?o1.
    ?s1 fused:contactnetwork_pid2 ?o2.
    ?s1 fused:contactnetwork_acttype1
        '5'^^xsd:decimal.
   ?s1 fused:contactnetwork_acttype1
        '5'^^xsd:decimal
}ORDER BY ASC(?e) LIMIT 1
```

Beginning with that infected person, we must discover the transitive disease propagation path length leading eventually to the end of the epidemic. SPARQL allows finding a transitive relationship between triples. However, triples should organize with a common predicate where the subject value of one triple will be used as an object for the next one and so on, as in Listing 3.62.

Listing 3.62: Find property paths in SPARQL.

```
{ ?infector :getInfectedBy* ?infectee }
```

In our RDF graph, we have "infector" and "infectee" information available in the "object" part of the triple where "subject" is the instance of the dendrogram and "predicate" is a "fused" vocabulary like Listing 3.63.

Listing 3.63: Partial example of dendrogram data in default RDF graph.

```
PREFIX fused: <http://ndssl.bi.vt.edu/chicago/vocab/>

<http://ndssl.bi.vt.edu/chicago/dendrogram/
experiment_id#7385/cell_id#86304/infectee_pid#442061435/
iteration#0>
 fused:dendrogram_infector_pid
  <http://ndssl.bi.vt.edu/chicago/person/
  pid#445683978> .
```

To make our RDF data useable for SPARQL transitivity, we constructed "Named Graph" from our default RDF graph as in Listing 3.64. The above listing is showing "Named Graph" creation for the first (0) replicate. Similarly, we created "Named Graph" for ten (0-9) replicates. We wrote a shell script that creates and uploades "Named Graph" from the Virtuoso command line for all ten replicates.

Listing 3.64: Query 1 Solution: Named graph creation (for first replicate).

```
PREFIX
rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX
fused:<http://ndssl.bi.vt.edu/chicago/vocab/>

CONSTRUCT
  {
  ?infectee ?getInfectedBy ?infector
  }

FROM <http://ndssl.bi.vt.edu/chicago/>

WHERE
   {
     ?s rdf:type fused:dendrogram.
     ?s fused:dendrogram_infectee_pid ?infectee.
     ?s fused:dendrogram_infector_pid ?infector.
     ?s fused:dendrogram_iteration
        '0'^^xsd:decimal.
     BIND (iri
     ('http://ndssl.bi.vt.edu/chicago/
     vocab/getInfectedBy') as ?getInfectedBy)
   }
```

Next, we use the infected person found in a school at the beginning of the epidemic query (Listing 3.61) and SPARQL transitivity on "Named Graph" to find the path length. Our

SPARQL solution for the first replicate is presented in Listing 3.65. It contains subqueries. http://ndssl.bi.vt.edu/chicago/ is our default graph and http://ndssl.bi.vt.edu/chicago/dendrogram/replicate0/ is our named graph. Similarly, we compute the path length for other replicates through our shell script mentioned earlier. Our query results are available in Table 3.3.

Listing 3.65: Query 1 Solution: Starting from school infected person finds transitive disease transmission path length before the epidemic dies out.

```
PREFIX
fused:<http://ndssl.bi.vt.edu/chicago/vocab/>

SELECT (count(?mid) as ?path_length)

WHERE{

{
SELECT ?o1  from <http://ndssl.bi.vt.edu/chicago/>
WHERE {
    ?s  fused:dendrogram_infector_pid ?o1.
    ?s  fused:dendrogram_infectee_pid ?o2.
    ?s  fused:dendrogram_iteration '9'^^xsd:decimal.
    ?s  fused:dendrogram_exposureday ?e.
    ?s1 fused:contactnetwork_pid1 ?o1.
    ?s1 fused:contactnetwork_pid2 ?o2.
    ?s1 fused:contactnetwork_acttype1
        '5'^^xsd:decimal.
    ?s1 fused:contactnetwork_acttype2
        '5'^^xsd:decimal
    }ORDER BY ASC(?e) LIMIT 1
}

GRAPH
<http://ndssl.bi.vt.edu/chicago/dendrogram/replicate0/>{
?begin fused:getInfectedBy ?o2.
?mid   fused:getInfectedBy* ?begin.
?end   fused:getInfectedBy* ?mid.
 }
}
```

Table 3.3: Query 1 Result. Replicate number, and, starting from school, the number of people infected (maximum path length) before the end of the epidemic.

Replicate	**Maximum path length**
First	26
Second	24
Third	13
Fourth	25
Fifth	27
Sixth	25
Seventh	27
Eights	22
Ninth	23
Tenth	27

Query 2: Find disease transmission between various demographic groups (school age, young age, middle age, senior age).

We can divide *Query 2* into the following sub-queries:

- *Subquery 1:* Who infected whom? (To answer we need dendrogram sources.)
- *Subquery 2:* What are the ages of an infector and an infectee? (To answer we need person age information from synthetic population sources.)

We divide the population into the following four demographic groups: i) school age (age range: 3-18), ii) young age (age range: 19-39), iii) middle age (age range: 40-64), and iv) senior age (age range: 65+). At first, we find out how many school-age people infected school age, young age, middle age, and senior age people. We then do the same for young age, middle age, and senior age infectors. We implement this query in the SPARQL language. In the following, we provide the query solution in steps.

In Listing 3.66 we present the select part of the query.

Listing 3.66: Query 2 Solution: Select Part. It counts infector-infectee groups.

```
PREFIX
fused:<http://ndssl.bi.vt.edu/chicago/vocab/>

SELECT

(count(?infectee_school_age_school_age)
as ?infectee_school_age_school_age),
(count(?infectee_school_age_young_age)
as ?infectee_school_age_young_age),
(count(?infectee_school_age_middle_age)
as ?infectee_school_age_middle_age),
(count(?infectee_school_age_senior_age)
as ?infectee_school_age_senior_age),
.........................................

FROM <http://ndssl.bi.vt.edu/chicago/>
```

In the following (Listing 3.67) we show a part of the "WHERE" clause of the query. Listing 3.67 is used to count the number of infectees where both infector and infectee belong to the school age group.

Listing 3.67: Query 2 Solution: Infector and infectee belong to school age group.

```
WHERE {

{
 ?dendrogram fused:dendrogram_infectee_pid
    ?infectee.
 ?dendrogram fused:dendrogram_infector_pid
    ?infector.
 ?infector fused:person_age
    ?infector_school_age_school_age.
    FILTER (?infector_school_age_school_age >= 3
    && ?infector_school_age_school_age <=18).
 ?infectee fused:person_age
    ?infectee_school_age_school_age.
 FILTER (?infectee_school_age_school_age >= 3
    && ?infectee_school_age_school_age <=18)
}
```

Listing 3.68 is used to count the number of young age people infected by school age people.

Listing 3.68: Query 2 Solution: Find the number of young age people infected by school age people.

```
UNION
{
?dendrogram fused:dendrogram_infectee_pid
    ?infectee.
?dendrogram fused:dendrogram_infector_pid
    ?infector.
?infector fused:person_age
    ?infector_school_age_young_age.
    FILTER (?infector_school_age_young_age >= 3
    && ?infector_school_age_young_age <=18).
?infectee fused:person_age
    ?infectee_school_age_young_age.
    FILTER (?infectee_school_age_young_age >= 19
    && ?infectee_school_age_young_age <=39)
}
```

Listing 3.69 is used to count the number of middle age people infected by school age people.

Listing 3.69: Query 2 Solution: Find the number of middle age people infected by school age people.

```
UNION
{
?dendrogram fused:dendrogram_infectee_pid
    ?infectee.
?dendrogram fused:dendrogram_infector_pid
    ?infector.
?infector fused:person_age
    ?infector_school_age_middle_age.
    FILTER (?infector_school_age_middle_age >= 3
    && ?infector_school_age_middle_age <=18).
?infectee fused:person_age
    ?infectee_school_age_middle_age.
    FILTER (?infectee_school_age_middle_age >= 40
    && ?infectee_school_age_middle_age <=64)
}
```

Listing 3.70 is used to count the number of senior age people infected by school age people.

Listing 3.70: Query 2 Solution: Find the number of senior age people infected by school age people.

```
UNION
{
?dendrogram fused:dendrogram_infectee_pid
    ?infectee.
?dendrogram fused:dendrogram_infector_pid
    ?infector.
?infector fused:person_age
    ?infector_school_age_senior_age.
    FILTER (?infector_school_age_senior_age >= 3
    && ?infector_school_age_senior_age <=18).
?infectee fused:person_age
    ?infectee_school_age_senior_age.
    FILTER (?infectee_school_age_senior_age >= 65)
}
.............................................
```

In this dissertation, we present the *Query 2* solution for "school age" infector group only. Similarly, we construct a SPARQL query for the young age, middle age, and senior age infector groups. As those solutions are similar to "school age" we omit those solutions from the dissertation. We present the *Query 2* result in Table 3.4.

Table 3.4: Query 2 (find disease transmission between various demographic groups) results.

Infector	**Infectee**	**Number of Persons**
School age	School age	3,877
	Young age	436
	Middle age	486
	Senior age	100
Young age	School age	174
	Young age	588
	Middle age	543
	Senior age	130
Middle age	School age	169
	Young age	554
	Middle age	591
	Senior age	142
Senior age	School age	30
	Young age	108
	Middle age	138
	Senior age	34

3.4 Limitations and Scope

The primary focus of this dissertation is to showcase the advantages of utilizing RDF/OWL-based schema for heterogeneous and fragmented CNEW data management. We have presented a flexible, ever-evolving schema, but the present work is not meant to be exhaustive. Our work has limitations. First, we have not discussed analysis and interventions in this dissertation. Second, disease models are discussed but in a limited fashion. For example, vector-borne diseases will require more complicated within-host disease progression automata. Third, our FUSED schema discussion is limited to class level. We have not discussed, in detail, the properties that can be exposed as a part of the conceptual, logical, and physical schemas. Fourth, although we discussed examples of complex high-level queries that can be written to query data stored using the FUSED schema, a more comprehensive development is needed. This would involve the development of a domain-specific language that can allow users to express complex queries in an intuitive fashion [120]. Finally, we have omitted a detailed discussion on schema instance creation, and the comprehensive SPARQL query that generates the mapping file from the FUSED schema. However, we provide a partial sample implementation of the D2RQ mapping file from the FUSED schema below. We use a Virtuoso SPARQL endpoint.

Listing 3.71: D2RQ Mapping: SPARQL query over FUSED schema.

```
prefix map: <#>
prefix rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix fuse: <http://www.ndssl.bi.vt.edu/fuse#>
prefix owl:<http://www.w3.org/2002/07/owl#>
prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/
D2RQ/0.1#>

CONSTRUCT {
    `iri(map:database)` a d2rq:Database.
    `iri(map:database)` d2rq:jdbcDriver ?driverName.
    `iri(map:database)` d2rq:jdbcDSN ?dsnName.
    `iri(map:database)` d2rq:username ?userName.
    `iri(map:database)` d2rq:password ?password.
}
FROM <http://www.ndssl.bi.vt.edu/fuse>
WHERE
{
    {
     fuse:DataSource rdf:type owl:Class.
     fuse:OracleDataSource  rdfs:subClassOf
                            fuse:DataSource.
     ?DB rdf:type fuse:OracleDataSource.
     ?DB fuse:driver ?driver.
            BIND (STR(?driver) as ?driverName).
     ?DB fuse:dsn ?dsn.
            BIND (STR(?dsn) as ?dsnName).
```

```
    ?DB fuse:username ?user.
           BIND (STR(?user) as ?userName).
    ?DB fuse:password ?pass.
           BIND (STR(?pass) as ?password)
   }
}
```

3.5 Summary

In this chapter, we described a schema for heterogeneous CNEW datasets. We demonstrated that it facilitates large-scale epidemiology simulation input and output data linking and management. First, we described three levels of the schema. Next, we provided a discussion of mapping file creation. Finally, we provided a discussion about experimentation. We implemented a prototype linking complete CNEW study datasets, and show the strength of the linked datasets by executing two complex queries. To the best of our knowledge this is the first attempt to develop a schema for computational networked epidemiology datasets. The empirical results show that our proposed schema is capable of extracting complex query results from heterogeneous data sources. Using linked data principles, we proposed a framework that empowers epidemiologists to examine the granularity of data hidden inside simulation contents, while saving time spent searching for information from various machines. Our application could also help government officials to understand the hidden nature of a given epidemic and take necessary steps to control it.

Chapter 4

A Semantic Web-Based Schema for Reported Infectious Disease Epidemic Datasets

Numerous devastating epidemics have shaken the world in recent times. In 2014, an outbreak of the Ebola virus claimed at least 10,000 lives. At the moment, Ebola is under control but the Zika virus is still on the upward trajectory. Figure 4.1 presents worldwide Ebola and Zika affected regions. Researchers have pointed out that the two diseases could be signs of a coming outbreak of an even larger scale and severity. As of now, no comprehensive commercial vaccine or treatment medication exists to cure Zika or Ebola viruses. The current viable way to end the spread of the diseases is breaking the chains of infection. Many parties, such as governments, private firms, and others are increasingly releasing epidemic datasets to focus on the infection and spread of the diseases. Collecting and studying these datasets will allow experts to find patterns in the way a given disease infects and spreads from one person to the next. The problem facing researchers is that the datasets are in various places and varying forms. Although these forms of data can be read and comprehended by human beings, the data that is published should be read by a machine to discover underlying patterns. The fact that the datasets have been published using different formats and schema, and exist in various locations, makes it difficult to establish links among them. In this dissertation, we propose a semantic web-based schema that enables the organization of epidemic reported datasets from diverse sources. We propose a novel linking scheme to connect various epidemic datasets. This linked data creates an integrated knowledge-base that allows epidemiologists to ask complex queries employing multiple datasets. This helps epidemiologists to understand disease propagation to aid efficient outbreak detection and control activities.

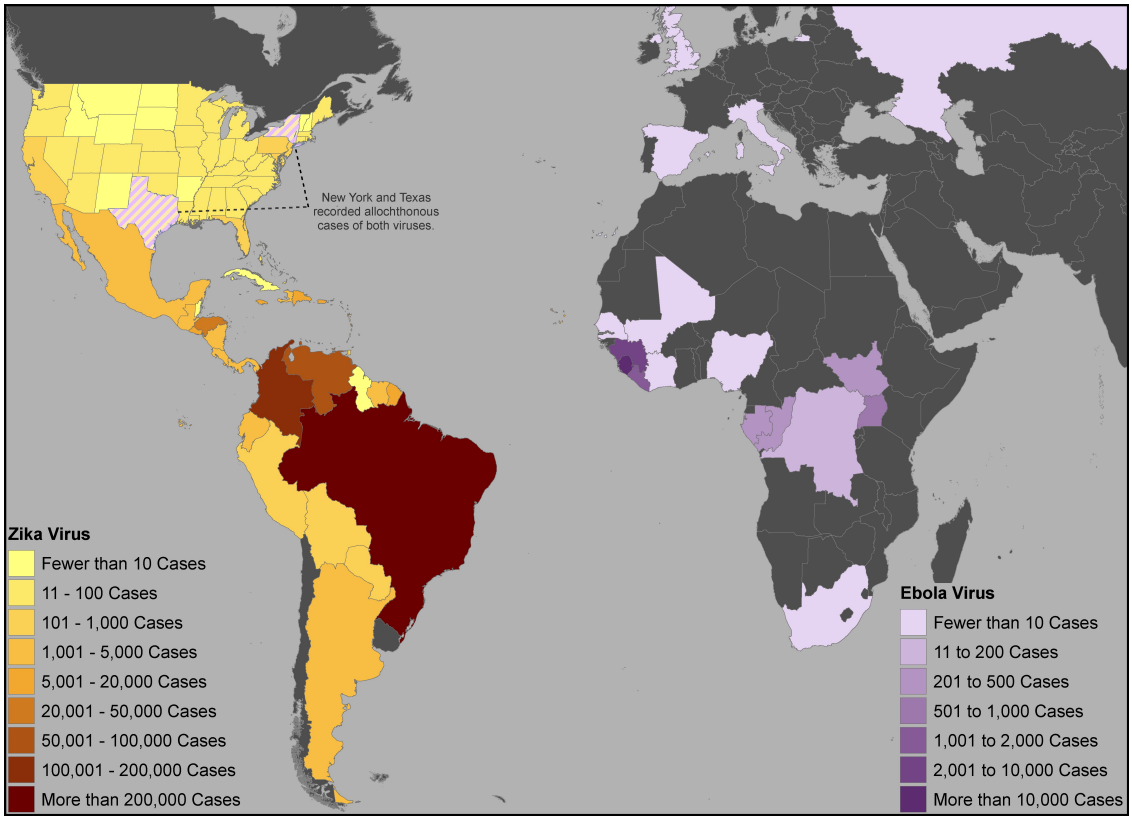


Figure 4.1: Worldwide Ebola and Zika affected regions. The case count information in the figure includes confirmed and suspected cases for both allocthonous and autocthonous. The US and countries in Europe do not have any autocthonous Ebola. However, Florida and Utah, states in the US, have authocthonous Zika (Image courtesy: Pyrros A. Telionis, Virginia Tech).

4.1 Reported Epidemic Dataset Schema

In this section, we illustrate the proposed reported epidemic dataset schema. The schema is designed by considering the semantic web concepts discussed in Chapter 2. It is comprised of three layers: conceptual, logical, and physical. We present our schema with numerous listings. In the listings, a separator line with dots (.) means details are omitted, and a separator line with dashes (-) identifies a different listing section.

4.1.1 Conceptual Schema

The conceptual schema represents reported epidemic datasets and relationships among them in the simplest terminology that fits an end user's mental model. Epidemic dataset classes are as follows: *Epidemic* contains a *Disease*; occurred in a *Country*; has *TimeSeries* information that contains case and death counts; various organizations responding to an epidemic are a *RespondedOrganization*; numerous *Logistics* are sent to the *Epidemic* infected

regions; a *HealthCareWorker* provides service to the infected regions; many *Publications* come out to describe the epidemic; people upload *Demography* datasets related to the Epidemic; *Environment* and *Transportation* influence an *Epidemic*; and it has an impact on a country's *Economy*. Epidemiology reported dataset classes and their relationships are presented in Listing 4.1.

In our schema, we have two types of properties, data property (defines literal values of a class) and object property (defines the relationship among classes). We denote the data property as *hasDataProperty* and object property as *hasObjectProperty*. Property names are unique in our schemas, and properties are explicit, which means instead of the generic relation *has* we use the domain specific relation *hasDisease*. Our properties are self-explanatory. In Listing 4.2, we present Epidemic datasets, data, and object properties.

Listing 4.1: Epidemiology reported dataset classes.

```
:Epidemic       :Country            :Disease                  1
:TimeSeries     :LineList           :RespondedOrganization    2
:Logistics      :MedicalFacilities  :HealthCareWorker         3
:Publication    :Map                :Demography               4
:Environment    :Transportation     :Economics                5
```

Listing 4.2: Relationships among epidemic dataset classes with object and data property example.

```
:Epidemic   rdf:type    owl:Class                              1
.....................................................          2
:Epidemic   :hasDisease                 :Disease               3
            :occuredIn                  :Country               4
            :hasTimeSerie               :TimeSeries            5
            :hasRespondedOrganization                          6
                                    :RespondedOrganization     7
            :hasLogistics               :Logistics             8
            :hasMedicalFacilities       :MedicalFacilities     9
            :hasHealthCareWorker        :HealthCareWorker      10
            :hasPublication             :Publication           11
            :hasDemography              :Demography            12
            :hasImpactedBy              :Environment           13
            :hasImpactedBy              :Transportation        14
            :hasImapctOn                :Economy               15
.....................................................          16
:hasObjectProperty  rdf:type    owl:ObjectProperty             17
                                                               18
:hasTimeSeries  rdfs:subPropertyOf  :hasObjectProperty         19
                rdfs:domain         :Epidemic                  20
                rdfs:range          :TimeSeries                21
.....................................................          22
:hasDataProperty    rdf:type    owl:DatatypeProperty           23
                                                               24
:hasNumberOfCases   rdfs:subPropertyOf  :hasDataProperty       25
                                                               26
:hasNumberOfCases   rdfs:domain     :TimeSeries                27
                    rdfs:range      rdfs:Literal               28
.....................................................          29
```

4.1.2 Logical Schema

The classes that we describe in the conceptual schema are fragmented, which therefore requires incorporation into our schema. Also, we need a mechanism to identify class instances uniquely. Logical schema model these aspects. We define a set of macros in Listings 4.3, 4.4, 4.5, and 4.6. The macros (or RDF patterns) enable us to present a succinct illustration of the logical schema, thereby increasing readability.

Macros

In the following we present logical schema macros.

Listing 4.3: Class and its properties macro.

```
#define CLASS_AND_PROPERTY(CLASS_NAME, PROPERTY_LIST)

:CLASS_NAME     rdf:type    owl:Class

:PROPERTY_LIST  rdfs:subPropertyOf
                    :hasDataProperty
                rdfs:domain :CLASS_NAME
                rdfs:range  rdfs:Literal
--------------------------------------------------
Where:

CLASS_NAME = Name of the class.
PROPERTY_LIST = List of the class properties.
--------------------------------------------------
Example:

personPropertyList = [hasAge,...]

CLASS_AND_PROPERTY(Person, personPropertyList)

Means:

:Person         rdf:type            owl:Class

:hasAge         rdfs:subPropertyOf  :hasDataProperty
                rdfs:domain         :Person
                rdfs:range          rdfs:Literal
..................................................
```

Listing 4.4: Subclass macro.

```
#define SUBCLASSOF(SUBCLASS, SUPERCLASS)

:SUBCLASS
    :subClassOf :SUPERCLASS
-------------------------------------------
Where:

SUBCLASS = Name of the subclass.
SUPERCLASS = Name of the superclass.
-------------------------------------------
Example:

SUBCLASSOF(ResearchPublication, Publication)

Means:

:ResearchPublication :subClassOf :Publication
```

Listing 4.5: Data property.

```
#define DATA_PROPERTY(PROPERTY_NAME)

:PROPERTY_NAME
  rdfs:subPropertyOf :hasDataPropertyAttribute
-----------------------------------------------
Where:
PROPERTY_NAME = Name of the property.
-----------------------------------------------
Example:

DATA_PROPERTY(hasAge)

Means:

:hasAge
   rdfs:subPropertyOf :hasDataPropertyAttribute
```

Listing 4.6: Primary key data property.

```
#define
PRIMARY_KEY_PROPERTY(PRIMARY_KEY_PROPERTY_NAME)

:PRIMARY_KEY_PROPERTY_NAME
 rdfs:subPropertyOf :hasPrimaryKeyProperty
-------------------------------------------
Where:
PRIMARY_KEY_PROPERTY_NAME = Name of the
primary key property.
-------------------------------------------
Example:

PRIMARY_KEY_PROPERTY(hasPersonID)

Means:
```

```
:hasPersonID rdfs:subPropertyOf                                    17
                :hasPrimaryKeyProperty                             18
```

Fragmentation

In the following we present the fragmentation aspect of the logical schema.

Country class fragmentations are presented in Listing 4.7.

Listing 4.7: Country subclasses.

```
SUBCLASSOF (FirstLevelAdministrativeRegion, Country)               1
SUBCLASSOF (SecondLevelAdministrativeRegion,                       2
    FirstLevelAdministrativeRegion)                                3
```

Timeseries class fragmentations are presented in Listing 4.8.

Listing 4.8: Timeseries subclasses.

```
SUBCLASSOF (HistoricalTimeseries, Timeseries)                      1
SUBCLASSOF (CurrentTimeseries, Timeseries)                         2
SUBCLASSOF (ForecastedTimeseries, Timeseries)                      3
```

Logistics class fragmentations are presented in Listing 4.9.

Listing 4.9: Logistics subclasses.

```
SUBCLASSOF (ReliefMaterial, Logistics)                             1
SUBCLASSOF (Laboratories, Logistics)                               2
```

MedicalFacilities class fragmentations are presented in Listing 4.10.

Listing 4.10: MedicalFacilities subclasses.

```
SUBCLASSOF (Hospital, MedicalFacilities)                           1
SUBCLASSOF (CommunityCareCenter, MedicalFacilities)                2
```

HealthCareWorker class fragmentations are presented in Listing 4.11.

Listing 4.11: HealthCareWorker subclasses.

```
SUBCLASSOF (HealthCareWorkerInfected, HealthCareWorker)            1
SUBCLASSOF (HealthCareWorkerDeath, HealthCareWorker)               2
SUBCLASSOF (BurialTeam, HealthCareWorker)                          3
```

Disease class fragmentation is presented in Listing 4.12.

Listing 4.12: Disease subclass.

```
SUBCLASSOF (DiseaseParameter, Disease)                             1
```

Publication class fragmentations are presented in Listing 4.13.

Listing 4.13: Publication subclasses.

```
SUBCLASSOF (ResearchPublication, Publication)
SUBCLASSOF (SituationReport, Publication)
SUBCLASSOF (NewsArticle, Publication)
SUBCLASSOF (Survey, Publication)
```

Demography class fragmentations are presented in Listing 4.14.

Listing 4.14: Demography subclasses.

```
SUBCLASSOF (Birth, Demography)
SUBCLASSOF (Death, Demography)
SUBCLASSOF (Pregnancy, Demography)
```

Environment class fragmentations are presented in Listing 4.15.

Listing 4.15: Environment subclasses.

```
SUBCLASSOF (Temperature, Environment)
SUBCLASSOF (Rainfall, Environment)
SUBCLASSOF (Wind, Environment)
SUBCLASSOF (Humidity, Environment)
```

Transportation class fragmentations are presented in Listing 4.16.

Listing 4.16: Transportation subclasses.

```
SUBCLASSOF (Road, Transportation)
SUBCLASSOF (Air, Transportation)
SUBCLASSOF (Railway, Transportation)
```

Economy class fragmentations are presented in Listing 4.17.

Listing 4.17: Economy subclasses.

```
SUBCLASSOF (Trade, Economy)
SUBCLASSOF (Food, Economy)
```

Schema

We now turn our attention to the logical schema. The logical schema exposes two type of data properties. One maps directly to the conceptual property. For example *hasNumberOfCases* is a property of the *TimeSeries* class. This property maps directly to the conceptual level data property. The second type of data property uniquely identifies instances of a class. We refer to it as *hasPrimaryKeyProperty*. Also, we are proposing a class *EpidemicCountry* in the logical schema. The purpose of this class is to represent the epidemic name with corresponding country information. We need this class because some datasets do not belong to a country. For example, an Ebola disease parameter dataset can contain Ebola information, but it does not belong to a country. *EpidemicCountry* class is a more normalized version of the *Epidemic* and *Country* classes, so we introduce it in the logical schema. We present the logical schema

in Listings 4.18, 4.19, 4.20, 4.21, and 4.22, but are not presenting the entire schema for the sake of brevity.

Listing 4.18: Epidemic classes in the logical schema.

```
CLASS_AND_PROPERTY(Epidemic, [hasEpidemicIndex,
hasEpidemicName])
PRIMARY_KEY_PROPERTY(hasEpidemicIndex)
DATA_PROPERTY(hasEpidemicName)
.....................................................
CLASS_AND_PROPERTY(EpidemicCountry,
[hasEpidemicCountryIndex, hasEpidemicName, hasCountryName,
hasCountryCode])
PRIMARY_KEY_PROPERTY(hasEpidemicCountryIndex)
.....................................................
CLASS_AND_PROPERTY(Country,
[hasCountryIndex, hasCountryName, hasCountryCode])
PRIMARY_KEY_PROPERTY(hasCountryIndex)
.....................................................
CLASS_AND_PROPERTY(TimeSeries, [hasTimeSeriesIndex,
hasNumberOfCases, ...])
PRIMARY_KEY_PROPERTY(hasTimeSeriesIndex)
.....................................................
CLASS_AND_PROPERTY(LineList, [hasLineListIndex,
hasExposureDate, ...])
PRIMARY_KEY_PROPERTY(hasLineListIndex)
.....................................................
CLASS_AND_PROPERTY
(Logistics, [hasLogisticsIndex, hasLogisricName,...])
PRIMARY_KEY_PROPERTY(hasLogisticsIndex)
.....................................................
CLASS_AND_PROPERTY
(ReliefMaterial, [hasReliefMaterialIndex, hasShippingType,
hasDate...])
PRIMARY_KEY_PROPERTY(hasReliefMaterialIndex)
.....................................................
CLASS_AND_PROPERTY
(Laboratories, [hasLaboratoriesIndex, hasName,
hasLatitude, Longitude, ...])
PRIMARY_KEY_PROPERTY(hasLaboratoriesIndex)
.....................................................
CLASS_AND_PROPERTY
(CommunityCareCenter, [hasCommunityCareCenterIndex,
hasCommunityCareCenterName, ...])
PRIMARY_KEY_PROPERTY(hasCommunityCareCenterIndex)
.....................................................
CLASS_AND_PROPERTY
(HealthCareWorkerInfected,
[hasHealthCareWorkerInfectedIndex,
hasTotalCases, ...])
PRIMARY_KEY_PROPERTY(hasHealthCareWorkerInfectedIndex)
.....................................................
CLASS_AND_PROPERTY
(HealthCareWorkerDeath, [hasHealthCareWorkerDeathIndex,
hasTotalDeaths, ...])
PRIMARY_KEY_PROPERTY(hasHealthCareWorkerDeathIndex)
.....................................................
```

```
CLASS_AND_PROPERTY
(RespondedOrganization, [hasRespondedOrganizationIndex,
hasOrganizationName, ...])
PRIMARY_KEY_PROPERTY(hasRespondedOrganizationIndex)
.....................................................
CLASS_AND_PROPERTY
(Disease, [hasDiseaseIndex, hasParameterName, ...])
PRIMARY_KEY_PROPERTY(hasDiseaseIndex)
.....................................................
```

Listing 4.19: Demography classes in the logical schema.

```
CLASS_AND_PROPERTY (Demography, [hasDemographyIndex,
hasTotalPopulation, ...])
PRIMARY_KEY_PROPERTY(hasDemographyIndex)
.....................................................
```

Listing 4.20: Environment classes in the logical schema.

```
CLASS_AND_PROPERTY (Temperature, [hasTemperatureIndex,
    hasTemperature, ...])
PRIMARY_KEY_PROPERTY(hasTemperatureIndex)
.....................................................
CLASS_AND_PROPERTY (Rainfall, [hasRainfallIndex,
hasAverageRainfall, ...])
PRIMARY_KEY_PROPERTY(hasRainfallIndex)
.....................................................
CLASS_AND_PROPERTY (Humidity, [hasHumidityIndex,
hasDewPoint, ...])
PRIMARY_KEY_PROPERTY(hasHumidityIndex)
.....................................................
```

Listing 4.21: Transportation classes in the logical schema.

```
CLASS_AND_PROPERTY (Road, [hasRoadIndex,
hasDrivingDistnace, ...])
PRIMARY_KEY_PROPERTY(hasRoadIndex)
.....................................................
CLASS_AND_PROPERTY (Air, [hasAirIndex,
hasAirPortName, ...])
PRIMARY_KEY_PROPERTY(hasAirIndex)
.....................................................
```

Listing 4.22: Economy classes in the logical schema.

```
CLASS_AND_PROPERTY (Food, [hasFoodIndex,
hasPrice, ...])
PRIMARY_KEY_PROPERTY(hasFoodIndex)
.....................................................
```

4.1.3 Physical Schema

Physical schema model the storage aspect of the reported datasets. In the following, we present physical schema macros, subclasses, and schema description.

4.1.4 Macros

We present subproperty, class storage, and property storage macros in Listings 4.23, 4.24, and 4.25.

Listing 4.23: Subproperty relation macro.

```
#define SUBPROPERTYOF(SUBPROPERTY, SUPERPROPERTY)

:SUBPROPERTY
    :subPropertyOf :SUPERPROPERTY
-------------------------------------------
Where:

SUBCLASS = Name of the subproperty.
SUPERCLASS = Name of the superproperty.
-------------------------------------------
Example:

SUBPROPERTYOF(hasAge, hasColumn)

Means:

:hasAge :subPropertyOf :hasColumn
```

Listing 4.24: Class physical storage.

```
#define CLASS_STORAGE (SCHEMA_CLASS_NAME,
PHYSICAL_STORAGE_CLASS_NAME)

:SCHEMA_CLASS_NAME
    :hasStorage :PHYSICAL_STORAGE_CLASS_NAME
-------------------------------------------
Where:
SCHEMA_CLASS_NAME = Name of the schema class.
PHYSICAL_STORAGE_CLASS_NAME =
Name of the storage class.
-------------------------------------------
Example:

CLASS_STORAGE(student, studentTable)

Means:

:student :hasStorage :studentTable
```

Listing 4.25: Property storage.

```
#define PROPERTY_STORAGE(CLASS_PROPERTY_NAME,
PHYSICAL_STORAGE_PROPERTY_NAME)

:CLASS_PROPERTY_NAME
    :storedAt :PHYSICAL_STORAGE_PROPERTY_NAME
-----------------------------------------------
Where:
CLASS_PROPERTY_NAME =
Name of the schema class property.
PHYSICAL_STORAGE_PROPERTY_NAME =
Name of the physical storage class property.
-----------------------------------------------
Example:

PROPERTY_STORAGE (hasID, hasIdentificationNumber)

Means:

:hasID :storedAt :hasIdentificationNumber
```

We model storage with various media (relational database, CSV file, etc.) type classes, and the *storeAt* relation. Physical schema classes are available in Listing 4.26.

Listing 4.26: Physical schema classes.

```
:DataSource, :Table, :FileColumn,
:KeyValueFileColumn
```

Fragmentation

DataSource represents various types of data storage formats. We model it in Listing 4.27. We provide a *DataSource* class property example in Listing 4.28.

Listing 4.27: Datasource class and subclasses.

```
SUBCLASSOF (OracleDataSource, DataSource)
SUBCLASSOF (PostgreSQLDataSource, DataSource)
SUBCLASSOF (FileDataSource, DataSource)
```

Listing 4.28: Datasource class with property example.

```
CLASS_AND_PROPERTY
(OracleDataSource, [hasUsername, hasPassword, ...])
```

Our schema supports various types of data storage. For example a relational database stores data in a Table in column format. Some files store data in column format, while others store data in key-value pair format. We model this aspect of the storage in Listing 4.29.

Listing 4.29: Column classes with property example.

```
CLASS_AND_PROPERTY (Table, [hasColumn])
CLASS_AND_PROPERTY
(FileColumn, [hasFileColumn])
CLASS_AND_PROPERTY
(KeyValueFileColumn, [hasKeyValueFileColumn])
CLASS_AND_PROPERTY (AllColumn, [hasAllColumn])
```

Listing 4.30: Various column properties.

```
SUBPROPERTYOF (hasColumn, hasAllColumn)
SUBPROPERTYOF (hasFileColumn, hasAllColumn)
SUBPROPERTYOF (hasKeyValueFileColumn, hasAllColumn)
```

Schema

We propose a property called *hasAllColumn*. All sorts of physical data storage columns are sub-properties of *hasAllColumn* (Listing 4.30). It expresses attributes of the dataset. We link an attribute to the equivalent data property attribute. Therefore, *hasDataProperty* maintains the *storedAt* relationship with *hasAllColumn* 4.31. An example is "*hasEpidemicName storedAt hasColumnEpidemicName*."

Listing 4.31: Attribute *storedAt* relation

```
:hasDataProperty :storedAt :hasAllColumn
:hasAllColumn rdf:type owl:DatatypeProperty
```

Here the Table class represents datasets that are stored in a relational database table format and the *hasTable* property models the relationship between Data Source and Table classes 4.32.

Listing 4.32: Table class, subclasses, and relations.

```
SUBCLASSOF (OracleTable, Table)
SUBCLASSOF (PostgreSQLTable, Table)

:OracleDataSource   :hasTable   :OracleTable
.........................................
```

TimeSeriesFile is a subclass of the FileDataSource and models the timeseries part of the reported datasets (Listing 4.33).

Listing 4.33: File class and subclasses.

```
SUBCLASSOF(TimeSeriesFile, FileDataSource)
.........................................
```

In Listings 4.34, 4.35, 4.36, and 4.37 we present physical level epidemic classes. We are not presenting the whole schema for the sake of brevity.

Listing 4.34: Part 1: Epidemic classes in physical schema.

```
CLASS_AND_PROPERTY(Epidemic, [hasEpidemicIndex,
hasEpidemicName])
PRIMARY_KEY_PROPERTY(hasEpidemicIndex)
DATA_PROPERTY(hasEpidemicName)
.....................................................
CLASS_AND_PROPERTY(EpidemicFile,
[hasEpidemicFileRowIndex,
hasColumnEpidemicName])
.....................................................
SUBPROPERTYOF(hasColumnEpidemicName, hasFileColumn)
.....................................................
CLASS_STORAGE (Epidemic, EpidemicFile)
.....................................................
PROPERTY_STORAGE (hasEpidemicIndex ,
hasEpidemicFileRowIndex)
.....................................................
-----------------------------------------------------
CLASS_AND_PROPERTY(EpidemicCountry,
[hasEpidemicCountryIndex, hasEpidemicName,
hasCountryName, hasCountryCode])
.....................................................
CLASS_AND_PROPERTY(EpidemicCountryFile,
[hasEpidemicCountryFileRowIndex, hasColumnEpidemicName,
hasColumnCountryName, hasColumnCountryCode])
.....................................................
CLASS_STORAGE (EpidemicCountry, EpidemicCountryFile)
.....................................................
PROPERTY_STORAGE (hasEpidemicCountryIndex,
hasEpidemicCountryFileRowIndex)
.....................................................
-----------------------------------------------------
CLASS_AND_PROPERTY(Country,
[hasCountryIndex, hasCountryName, hasCountryCode])
.....................................................
CLASS_AND_PROPERTY(CountryFile,
[hasCountryFileRowIndex, hasColumnCountryName,
hasColumnCountryCode])
.....................................................
CLASS_STORAGE (Country, CountryFile)
.....................................................
PROPERTY_STORAGE (hasCountryIndex,
hasCountryFileRowIndex)
.....................................................
```

Listing 4.35: Part 2: Epidemic classes in physical schema.

```
CLASS_AND_PROPERTY(TimeSeries, [hasTimeSeriesIndex,
hasNumberOfCases, ...])
.....................................................
CLASS_AND_PROPERTY
(TimeSeriesFile, [hasTimeSeriesFileRowIndex,
hasColumnNumberOfCases, ...])
.....................................................
CLASS_STORAGE (TimeSeries, TimeSeriesFile)
.....................................................
PROPERTY_STORAGE (hasTimeSeriesIndex,
```

```
hasTimeSeriesFileRowIndex)
.....................................................
-----------------------------------------------------
.....................................................
CLASS_AND_PROPERTY(LineList, [hasLineListIndex,
hasExposureDate, ...])
.....................................................
CLASS_AND_PROPERTY
(LineListFile, [hasLineListFileRowIndex,
hasColumnExposureDate, ...])
.....................................................
CLASS_STORAGE (LineList, LineListFile)
.....................................................
PROPERTY_STORAGE (hasLineListIndex,
hasLineListFileRowIndex)
.....................................................
-----------------------------------------------------
.....................................................
CLASS_AND_PROPERTY
(Logistics, [hasLogisticsIndex, hasLogisricName,...])
.....................................................
CLASS_AND_PROPERTY
(LogisticsFile, [hasLogisticsFileRowIndex,
hasColumnLogisricName,...])
.....................................................
CLASS_STORAGE (Logistics, LogisticsFile)
.....................................................
PROPERTY_STORAGE (hasLogisticsIndex,
hasLogisticsFileRowIndex)
.....................................................
```

Listing 4.36: Part 3: Epidemic classes in physical schema.

```
CLASS_AND_PROPERTY
(ReliefMaterial, [hasReliefMaterialIndex,
hasShippingType, hasDate...])
.....................................................
CLASS_AND_PROPERTY
(ReliefMaterialFile, [hasReliefMaterialFileRowIndex,
hasColumnShippingType, hasColumnDate...])
.....................................................
CLASS_STORAGE (ReliefMaterial, ReliefMaterialFile)
.....................................................
PROPERTY_STORAGE (hasReliefMaterialIndex,
hasReliefMaterialFileRowIndex)
.....................................................
-----------------------------------------------------
CLASS_AND_PROPERTY
(Laboratories, [hasLaboratoriesIndex, hasName, hasLatitude,
hasLongitude, ...])
.....................................................
CLASS_AND_PROPERTY
(LaboratoriesFile, [hasLaboratoriesFileRowIndex,
hasColumnName, hasColumnLatitude,
hasColumnLongitude, ...])
.....................................................
CLASS_STORAGE (Laboratories, LaboratoriesFile)
```

```
.....................................................
PROPERTY_STORAGE (hasLaboratoriesIndex,
hasLaboratoriesFileRowIndex)
.....................................................
-----------------------------------------------------------
CLASS_AND_PROPERTY
(CommunityCareCenter, [hasCommunityCareCenterIndex,
hasCommunityCareCenterName, ...])
.....................................................
CLASS_AND_PROPERTY
(CommunityCareCenterFile,
[hasCommunityCareCenterFileRowIndex,
hasColumnCommunityCareCenterName, ...])
.....................................................
CLASS_STORAGE (CommunityCareCenter,
CommunityCareCenterFile)
.....................................................
PROPERTY_STORAGE (hasCommunityCareCenterIndex,
hasCommunityCareCenterFileRowIndex)
.....................................................
```

Listing 4.37: Part 4: Epidemic classes in physical schema.

```
CLASS_AND_PROPERTY
(HealthCareWorkerInfected,
[hasHealthCareWorkerInfectedIndex, hasTotalCases, ...])
.....................................................
CLASS_AND_PROPERTY
(HealthCareWorkerInfectedFile,
[hasHealthCareWorkerInfectedFileRowIndex,
hasColumnTotalCases, ...])
.....................................................
CLASS_STORAGE (HealthCareWorkerInfected,
HealthCareWorkerInfectedFile)
.....................................................
PROPERTY_STORAGE (hasHealthCareWorkerInfectedIndex,
hasHealthCareWorkerInfectedFileRowIndex)
.....................................................
-----------------------------------------------------------
.....................................................
CLASS_AND_PROPERTY
(HealthCareWorkerDeath, [hasHealthCareWorkerDeathIndex,
hasTotalDeaths, ...])
.....................................................
CLASS_AND_PROPERTY
(HealthCareWorkerDeathFile,
[hasHealthCareWorkerDeathFileRowIndex,
hasColumnTotalDeaths, ...])
.....................................................
CLASS_STORAGE (HealthCareWorkerDeath,
HealthCareWorkerDeathFile)
.....................................................
PROPERTY_STORAGE (hasHealthCareWorkerDeathIndex,
hasHealthCareWorkerDeathFileRowIndex)
.....................................................
-----------------------------------------------------------
.....................................................
```

```
CLASS_AND_PROPERTY
(RespondedOrganization, [hasRespondedOrganizationIndex,
hasOrganizationName, ...])
.....................................................
CLASS_AND_PROPERTY
(RespondedOrganizationFile,
[hasRespondedOrganizationFileRowIndex,
hasColumnOrganizationName, ...])
.....................................................
CLASS_STORAGE (RespondedOrganization,
RespondedOrganizationFile)
.....................................................
PROPERTY_STORAGE (hasRespondedOrganizationIndex,
hasRespondedOrganizationFileRowIndex)
.....................................................
-----------------------------------------------------------
.....................................................
CLASS_AND_PROPERTY
(Disease, [hasDiseaseIndex, hasParameterName, ...])
.....................................................
CLASS_AND_PROPERTY
(DiseaseFile, [hasDiseaseFileRowIndex,
hasColumnParameterName, ...])
.....................................................
CLASS_STORAGE (Disease, DiseaseFile)
.....................................................
PROPERTY_STORAGE (hasDiseaseIndex,
hasColumnParameterName)
.....................................................
```

The *Demography* physical level model is presented in Listing 4.38.

Listing 4.38: Demography classes in physical schema.

```
CLASS_AND_PROPERTY (Demography, [hasDemographyIndex,
hasTotalPopulation, ...])
.....................................................
CLASS_AND_PROPERTY
(DemographyFile, [hasDemographyFileRowIndex,
hasColumnTotalPopulation, ...])
.....................................................
CLASS_STORAGE (Demography, DemographyFile)
.....................................................
PROPERTY_STORAGE (hasDemographyIndex,
hasDemographyFileRowIndex)
.....................................................
```

The *Environment* physical level model is presented in Listing 4.39.

Listing 4.39: Environment classes in physical schema.

```
CLASS_AND_PROPERTY (Temperature, [hasTemperatureIndex,
hasTemperature, ...])
.....................................................
CLASS_AND_PROPERTY (TemperatureFile,
[hasTemperatureFileRowIndex,
hasColumnTemperature, ...])
.....................................................
```

```
CLASS_STORAGE (Temperature, TemperatureFile)
.....................................................
PROPERTY_STORAGE (hasTemperatureIndex,
hasTemperatureFileRowIndex)
.....................................................
-----------------------------------------------------------
.....................................................
CLASS_AND_PROPERTY (Rainfall, [hasRainfallIndex,
hasAverageRainfall, ...])
.....................................................
CLASS_AND_PROPERTY (RainfallFile, [hasRainfallFileRowIndex,
hasAverageRainfall, ...])
.....................................................
CLASS_STORAGE (Rainfall, RainfallFile)
.....................................................
PROPERTY_STORAGE (hasRainfallIndex,
hasRainfallFileRowIndex)
.....................................................
-----------------------------------------------------------
.....................................................
CLASS_AND_PROPERTY (Humidity, [hasHumidityIndex,
hasDewPoint, ...])
.....................................................
CLASS_AND_PROPERTY (HumidityFile, [hasHumidityFileRowIndex,
hasColumnDewPoint, ...])
.....................................................
CLASS_STORAGE (Humidity, HumidityFile)
.....................................................
PROPERTY_STORAGE (hasHumidityIndex,
hasHumidityFileRowIndex)
.....................................................
```

The *Transportation* physical level model is presented in Listing 4.40.

Listing 4.40: Transportation classes in physical schema.

```
CLASS_AND_PROPERTY (Road, [hasRoadIndex,
hasDrivingDistnace, ...])
.....................................................
CLASS_AND_PROPERTY (RoadFile, [hasRoadFileRowIndex,
hasColumnDrivingDistnace, ...])
.....................................................
CLASS_STORAGE (Road, RoadFile)
.....................................................
PROPERTY_STORAGE (hasRoadIndex, hasRoadFileRowIndex)
.....................................................
-----------------------------------------------------
.....................................................
CLASS_AND_PROPERTY (Air, [hasAirIndex,
hasAirPortName, ...])
.....................................................
CLASS_AND_PROPERTY (AirFile, [hasAirFileIndex,
hasColumnAirPortName, ...])
.....................................................
CLASS_STORAGE (Air, AirFile)
.....................................................
PROPERTY_STORAGE (hasAirIndex , hasAirFileIndex)
.....................................................
```

The *Economic* physical level model is presented in Listing 4.41.

Listing 4.41: Economic classes in physical schema.

```
CLASS_AND_PROPERTY (Food, [hasFoodIndex, hasPrice, ...])
.....................................................
CLASS_AND_PROPERTY (FoodFile, [hasFoodFileRowIndex,
hasColumnPrice, ...])
.....................................................
CLASS_STORAGE (Food, FoodFile)
.....................................................
PROPERTY_STORAGE (hasFoodIndex, hasFoodFileRowIndex)
.....................................................
```

4.1.5 Instance Creation

We now demonstrate the usage of the proposed physical schema to represent datasets stored across data servers and files in Listings 4.42, 4.43, and 4.44. For example *I_T* is an instance of the *TimeSeries* class and *I_Timeseries_File* is an instance of the *TimeSeriesFile*. We are not presenting the instances of the whole schema for brevity.

Listing 4.42: Epidemic class instances.

```
.....................................
:I_Timeseries
.....................................
-----------------------------------------
:I_Logistics
.....................................
-----------------------------------------
:I_ReliefMaterial
.....................................
-----------------------------------------
```

Listing 4.43: Epidemic file class instances.

```
.....................................
:I_Timeseries_File
.....................................
-----------------------------------------
:I_Logistics_File
.....................................
-----------------------------------------
:I_ReliefMaterial_File
.....................................
-----------------------------------------
```

Listing 4.44: Epidemic classes and their corresponding file class relationship in instance level.

```
:I_Timeseries       :hasStorage :I_Timeseries_File                    1
:I_Logistics        :hasStorage :I_Logistics_File                     2
:I_ReliefMaterial   :hasStorage :I_ReliefMaterial_File                3
.....................................................                 4
```

4.2 Linking Approach

4.2.1 Preliminaries

RDF stores data in triple (subject, predicate, object) format. A triple can be expressed as a graph where subject and object are represented by nodes, and the predicate as an edge. Let all graphs $G = \bigcup G_i$ and $G_i = (V_i, E_i)$. Nodes $V_i = \bigcup_j X_{ij}$. Edges $E_i = \{(u, v) : u \in X_{ij}, v \in Xik$, where $k > j\}$. Let l be a label of a node. Let $\overrightarrow{p}$ be a label of an incoming edge in a node. Let u' and v' be leaf nodes. Let the parent node of u' be $F(u')$ and the parent node of v' be $F(v')$. Here $u' \in V_m$ and $v' \in V_n$ where $m \neq n$.

Example: We present a sample RDF dataset G_1 in Figure 4.2. Here numbers 1, 2, and 3 in a circular box represent nodes of the graph G_1. Node 3 is an example of a leaf node u', and node 2 is an example of u''s parent $F(u')$. Here the leaf node's incoming edge label ($\overrightarrow{p}(u')$) is "hasCountryName."

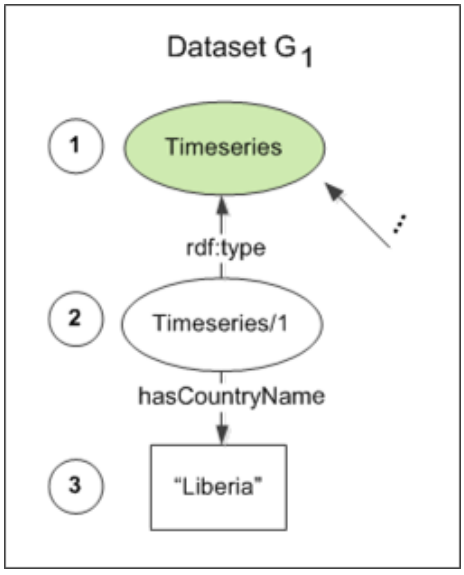


Figure 4.2: Sample RDF graph.

4.2.2 Literal Node Creation

Definition 4.2.1 *Literal node creation is an operation that creates new node b_t, where $V_i = V_i \cup \{b_t\}$, edge $E_i = E_i \cup \{F(u'), b_t\}$, and $E_i = E_i \cup \{b_t \cup u'\}$. The label of edge $(F(u'), b_t)$ is $\overrightarrow{p}(u')$ and the label of (b_t, u') is "hasValue".*

Example: We perform the literal node creation operation on Figure 4.2 and present the result in Figure 4.3. In Figure 4.3 node 3 is the new node b_t. The label of b_t is created by concatenation of class name (e.g., Timeseries), instance id (1), and part of the instance outgoing edge label (country_name). The edge label of $(F(u'), b_t)$ is "hasCountryName" and $(b_t \cup u')$ is "hasValue."

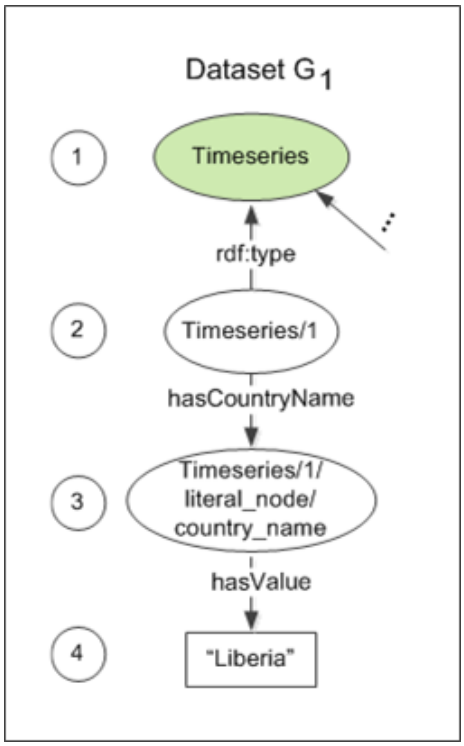


Figure 4.3: Literal node creation example.

4.2.3 Link Node Creation

Definition 4.2.2 *Link node creation is an operation. This operation is performed after the literal node creation operation. The link node creation operation is performed* ***if*** *label of leaf node $l(u') = l(v')$, label of incoming edge in a leaf node $\overrightarrow{p}(u') = \overrightarrow{p}(v') =$ "$hasValue$", label of parent node $l(F(u'))\neq l(F(v'))$, and label of incoming edge in parent node $\overrightarrow{p}(F(u')) = \overrightarrow{p}(F(v'))$. The link*

node creation operation creates new nodes c_s, where $V_i = V_i \cup \{c_s\}$, $E_i = \{(F(u'), c_s), (F(v'), c_s)$. The Label of $(F(u'), c_s) = (F(v'), c_s) =$ "$rdfs:seeAlso$.". We create the label of c_s by combining the term "link_node", part of the literal node label, and leaf node value.

Example: In Figure 4.4, we have two graphs G_1 and G_2 where leaf node label $l(u') = l(v') =$ "$Liberia$" (node numbers 4 and 9), leaf node incoming edge label $\overrightarrow{p}(u') = \overrightarrow{p}(v') =$ "$hasValue$", $F(u')$ and $F(v')$ are literal nodes created earlier (node numbers 3 and 8). $\overrightarrow{p}(F(u')) = \overrightarrow{p}(F(v')) =$ "$hasCountryName$". In the Figure 4.4, node 5 is link node c_s. Link node label contains the term "link_node", part of $\overrightarrow{p}(F(u')) = \overrightarrow{p}(F(v'))$ label, and label of $u' = v'$.

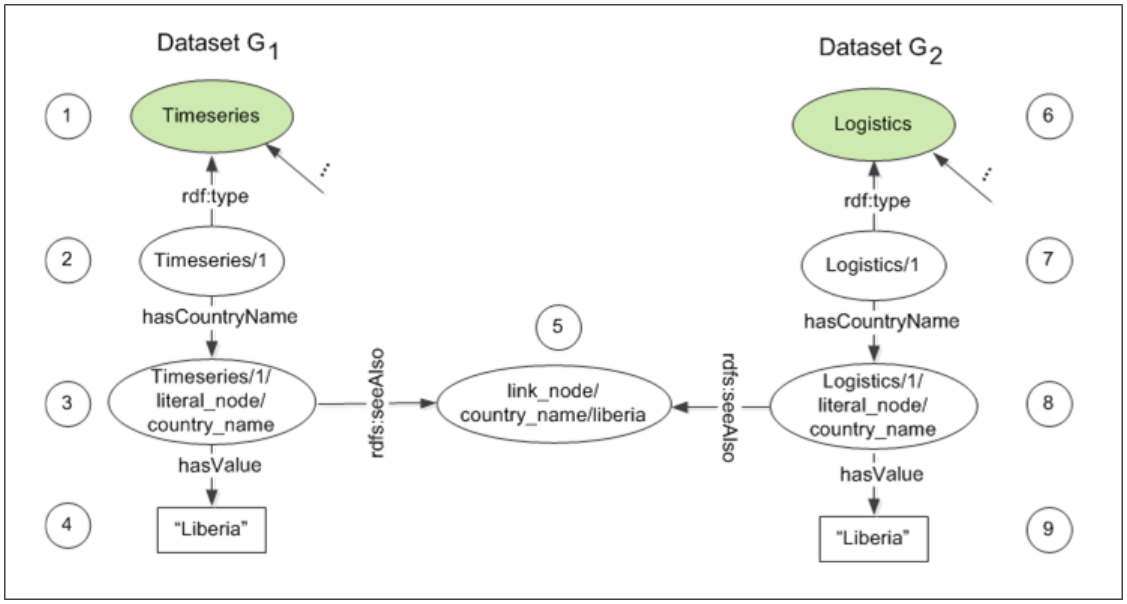


Figure 4.4: Link node creation example.

4.2.4 Data Organization Hierarchy

We examined numerous epidemic datasets and noticed that most appear with some country administrative division data. We can use this knowledge to organize our epidemic datasets. We create a data organization hierarchy where the epidemic is the root node. The level 1 node is country, level 2 node is country administrative division 1 (for the USA it is State), and level 3 node is country administrative division 2 (for the USA it is County). We do not go beyond country administrative division 2 because our epidemic dataset observation points out that most of the epidemic datasets do not contain more country subdivision information deeper than administrative division 2. Some datasets do not contain any country administrative division information, and so must be directly connected to the root node (Epidemic). We present our data organization hierarchy in Figure 4.5.

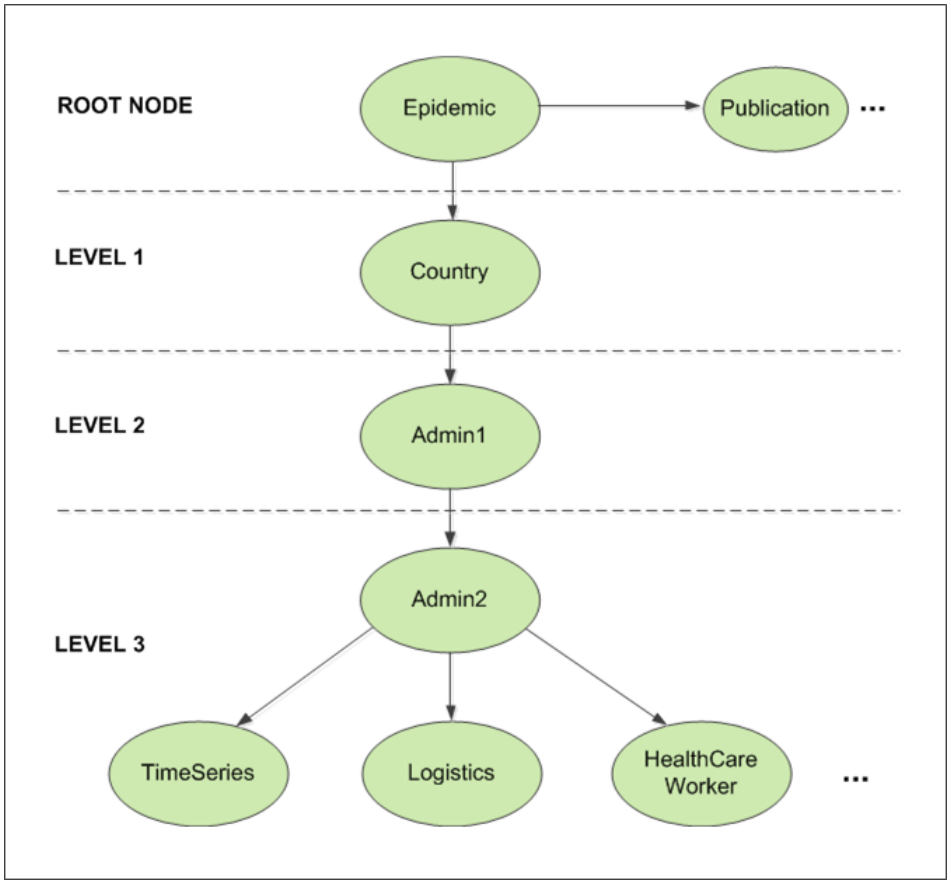


Figure 4.5: Data organization hierarchy.

Definition 4.2.3 (Hierarchy edge creation) *The hierarchy edge creation operation creates an edge between link nodes such that* $E_i = E_i \cup \{c_s, c_{s+1}\}$, $E_i = E_i \cup \{c_{s+1}, c_{s+2}\}$, $E_i = E_i \cup \{c_{s+2}, c_{s+3}\}$.

Example: In Figure 4.6, we present an example of hierarchy edge creation. In the figure c_s is node 1, c_{s+1} is node 2, c_{s+2} is node 3, and c_{s+3} is node 4. The label of $\{c_s, c_{s+1}\}$ is "hasCountryName", $\{c_{s+1}, c_{s+2}\}$ is "hasAdmin1Name", and $\{c_{s+2}, c_{s+3}\}$ is "hasAdmin2name".

Note 10: We have a class called *CountryEpidemic*. We need this class to create a link between *Epidemic* and *Country* classes.

We are organizing data in a hierarchical structure. This structure permits us to create a transitive connection between link nodes. A transitivity link is useful. Primarily, it will provide efficiency for some queries.

Definition 4.2.4 (Transitivity edge creation) *The Transitivity edge preserves a transitive relation among link nodes. It creates an edge between* $E_i = E_i \cup \{c_s, c_{s+2}\}$, $E_i = E_i \cup \{c_s, c_{s+3}\}$, *and* $E_i = E_i \cup \{c_{s+1}, c_{s+3}\}$.

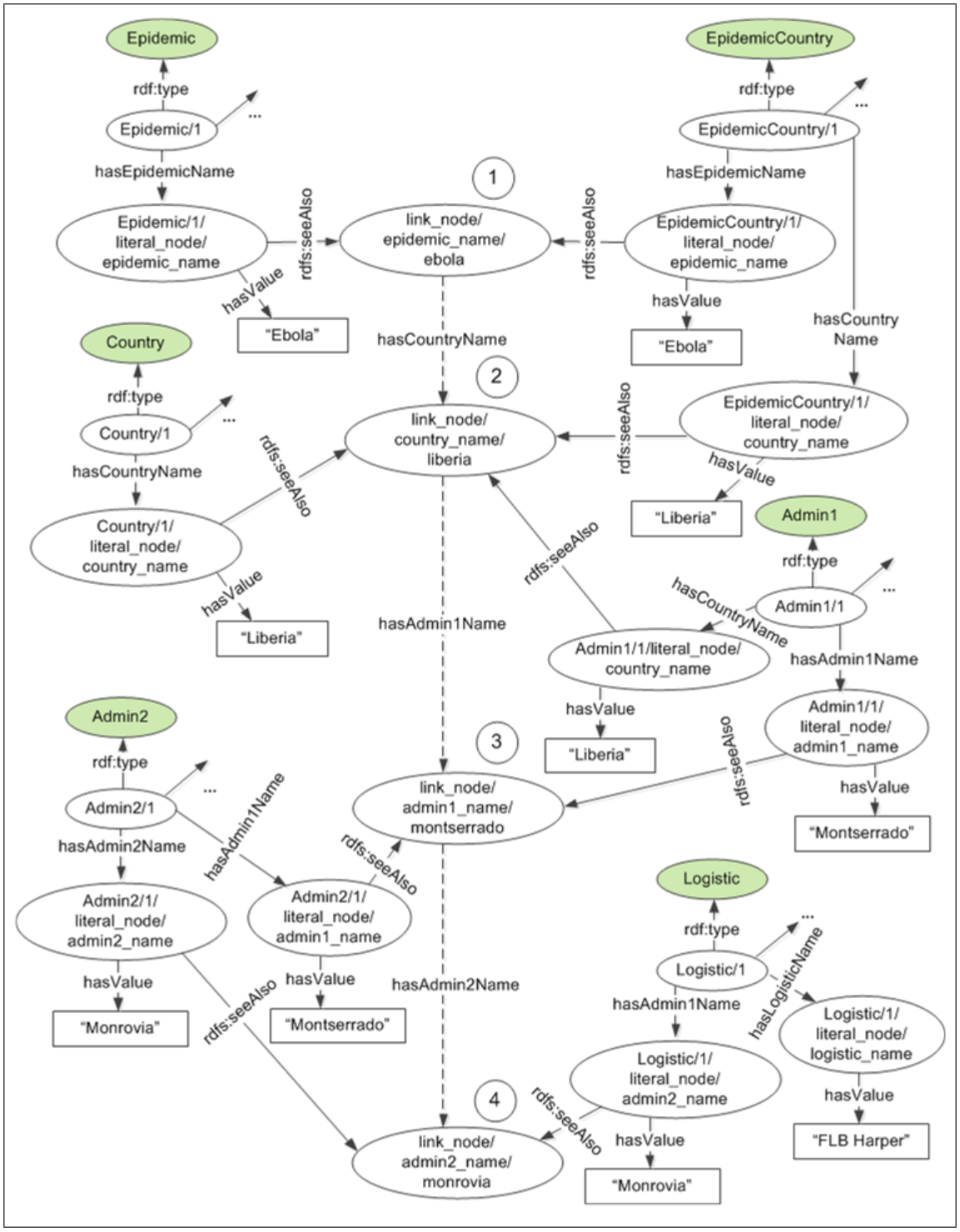


Figure 4.6: Hierarchy edge creation.

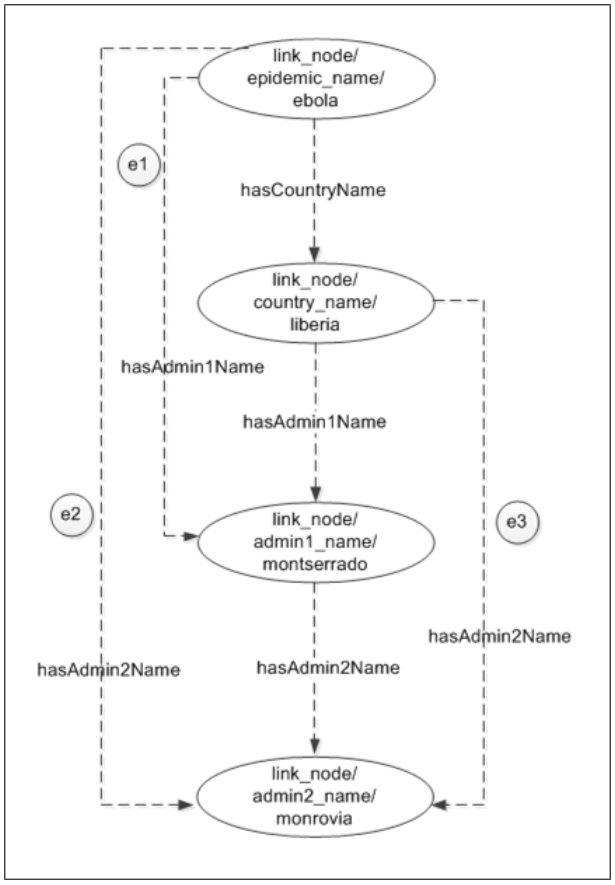


Figure 4.7: Transitivity edge creation.

Example: In Figure 4.7, we present a transitivity edge example. In the figure $\{c_s, c_{s+2}\}$, $\{c_s, c_{s+3}\}$, $\{c_{s+1}, c_{s+3}\}$ are represented by edges e_1, e_2, and e_3.

4.3 Limitations and Scope

The main purpose of this study is to present the advantages of using schema based on RDF/OWL for heterogeneous reported epidemic datasets. The study does not aim to be exhaustive. The presented schema is flexible and evolving. There are certain limitations of the present work that we have not discussed. First, it is possible to include many additional reported data classes into the schema. For instance, there are numerous different types of geographical map datasets published online that can be added to the schema. Second, it is possible to incorporate additional fragmentations of a conceptual class. Third, there may be duplicate data in the RDF graph. To illustrate, at the moment of outbreak progression, people add data in stages, such as on a monthly basis, and these data consist of both previous and new data. Different versions of the identical datasets can be incorporated into the RDF graph, which is likely to result in duplication of data. Fourth, a significant number

of datasets are not accompanied by documentation, which means that we cannot know the meaning of certain data fields resulting in data ambiguity. Fifth, there are different possible variants of data cleaning. We are not entirely addressing that in this dissertation. However, to illustrate, different literal values from different datasets whose meaning is the same can be linked through the *owl:sameAS* predicate.

4.4 Summary

Creating a schema for reported epidemic datasets is challenging due to the disorganized, vagueness, and heterogeneity of the datasets. Nevertheless, there is a pressing need to arrange datasets to analyze and understand an epidemic outbreak. Semantic web technology is exceptionally flexible, and it enables the rapid development of new solutions. In this chapter, a semantic web-based schema for heterogeneous reported epidemic datasets was described. It shows that preservation, organization, and linking of a large number of epidemic datasets is of great help to epidemiologists. The data are further enriched by classifying data classes and determining their relationship. The schema represents an excellent information source, as it elaborates on the content of a collection and the type of information that can be found in it.

Chapter 5

Computational Epidemiology Data Analytics

In this chapter, we discuss analytical environments to support epidemic science. An important goal is to analyze linked input as well as output datasets to facilitate effective spatio-temporal and social reasoning critical in planning and intervention analysis before and during an epidemic.

5.1 The EpiK: Data Federation and Homogeneous Query Execution Framework for Synthetic Data

EpiK is designed to support agent-based modeling and analytics frameworks – aggregate models can be seen as special cases and are thus supported. We use semantic web technologies to create a representation of the datasets that encapsulates both location and schema heterogeneity. The choice of RDF as a representation language is motivated by the diversity and growth of the datasets that need to be integrated. A query bank is developed – the queries capture a broad range of questions that can be posed and answered during a typical case study pertaining to disease outbreaks. The queries are constructed using SPARQL over the EpiK.

EpiK can hide schema and location heterogeneity while efficiently supporting queries that span the computational epidemiology modeling pipeline: from model construction to simulation output. We show that the performance of benchmark queries varies significantly with respect to the choice of hardware underlying the database and RDF engine.

The proposed EpiK consists of three layers: (*i*) the data layer: datasets (Section 5.1.1), (*ii*) the mapping layer (Section 5.1.2) and (*iii*) RDF engine and services layer (Section 5.1.3). See Figure 5.1. We discuss each of the layers below.

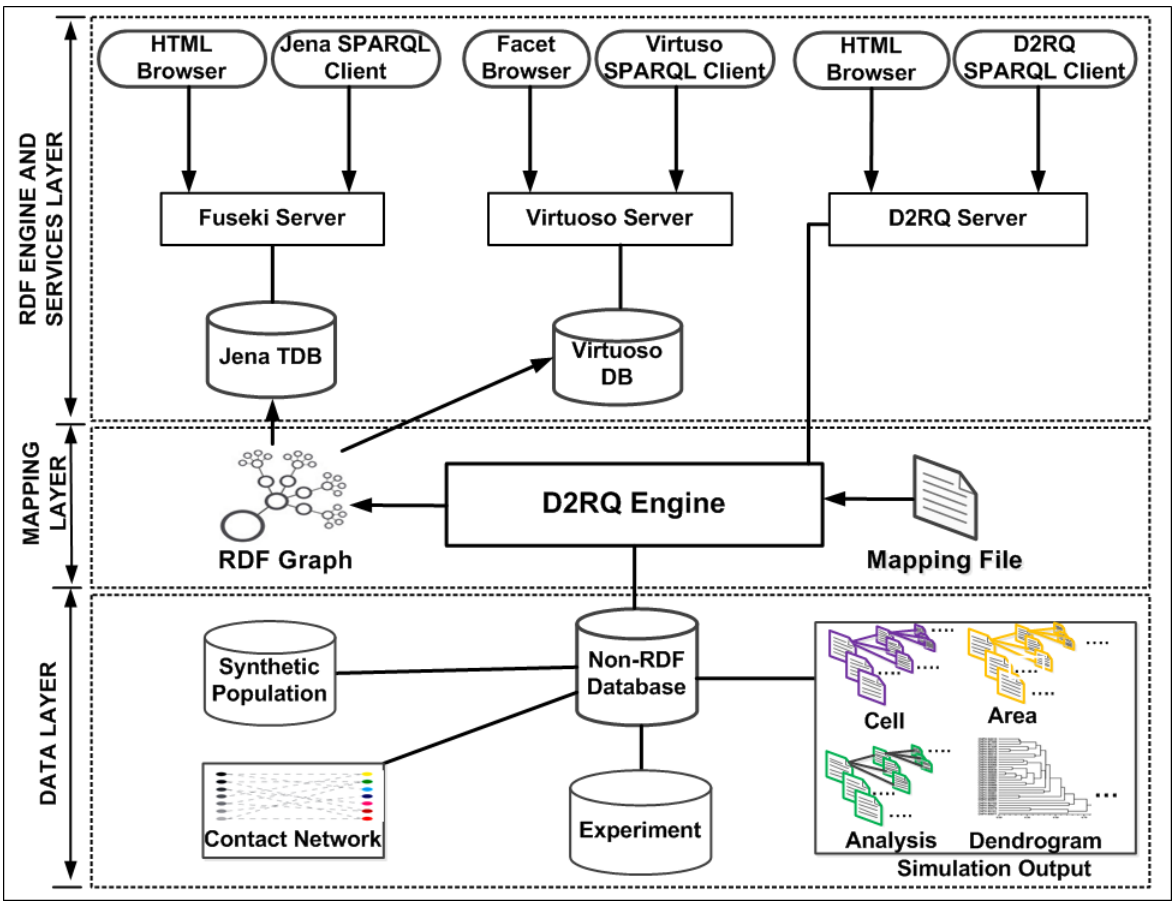


Figure 5.1: The EpiK proposed for agent-based epidemic modeling and analytics. It consists of three components/layers. The bottom layer (data layer) presents various computational epidemiology datasets in numerous formats. The middle layer (mapping layer) converts all the relevant data into materialized and virtual RDF graphs through the mapping file and D2RQ engine. The top layer (RDF engine and services layer) exposes the materialized RDF graph through Jena TDB [121] & Virtuoso servers [122], and the virtual RDF graph through the D2RQ server [119].

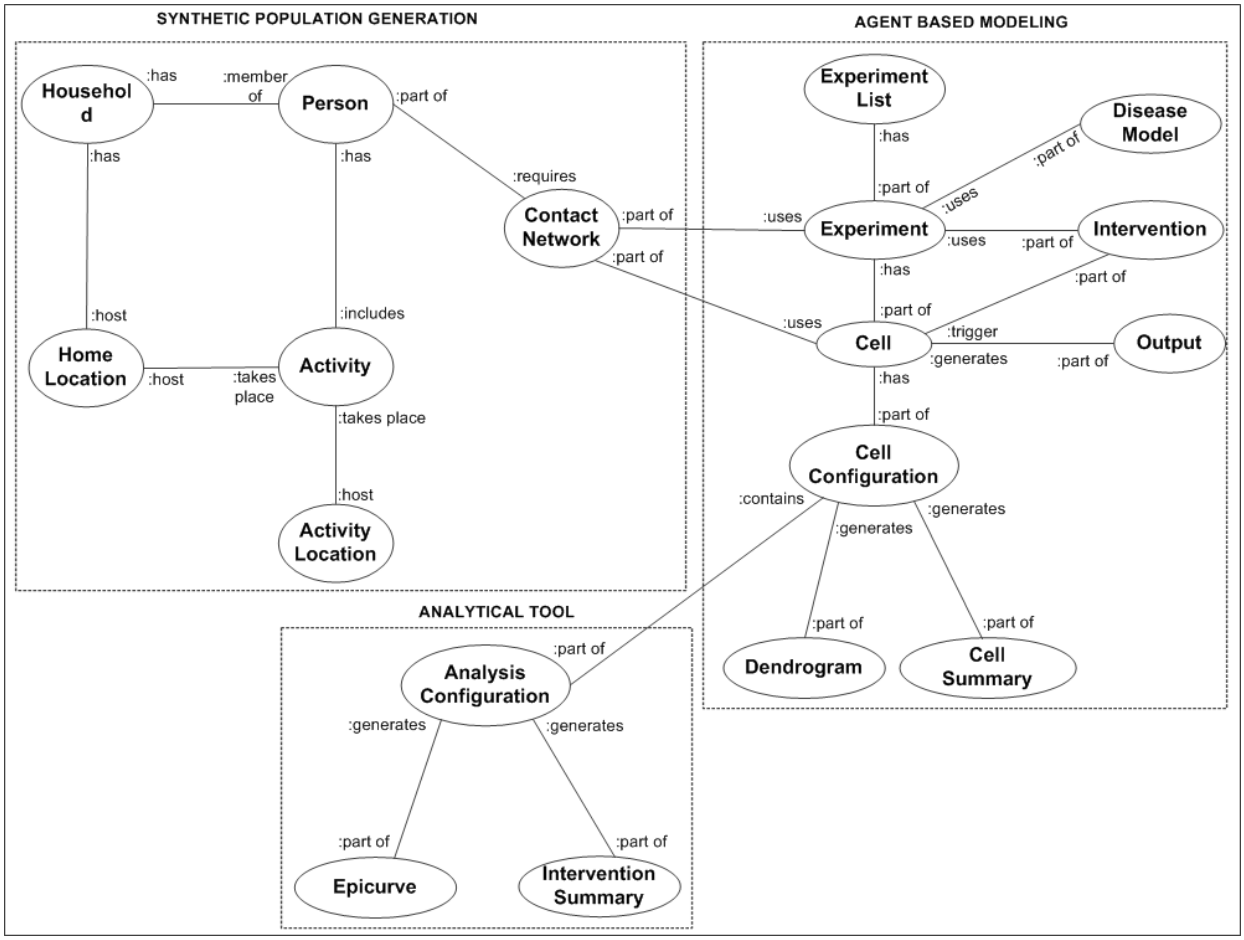


Figure 5.2: Datasets used as a part of agent-based epidemic modeling and analytics framework pipeline. The oval shape represents dataset, and the ontological relationships between various datasets are available just next to the oval. The synthetic population generation section covers person, household, location, activity, and contact network datasets. Agent-based modeling uses the contact network, disease model, and intervention to produce output datasets. The analytical tool covers experiment output analysis related datasets.

5.1.1 Data Layer

We start by describing EpiK's dataset layer; see Figure 5.1. The datasets are summarized in Figure 5.2 and are comprised of the (*i*) synthetic population, synthetic people-location network, and synthetic social contact network; (*ii*) experimental setup data consisting of disease model parameters and intervention specification; and (*iii*) experimental output data comprised of epidemic curves, dendrograms and various analyses. Table 5.1 shows the type, size, and format for datasets present in SIBEL [113].

Table 5.1: Computational networked epidemiology datasets available at NDSSL. Here we present an estimation for three broad categories: Synthetic Population, Contact Network and Output, and Experiment.

Category	**Data**	**Size**	**Representation**
Synthetic Population	Household, Person, Activity	~8 TB	Relational
Contact Network and Output	Contact Network, Simulation, Output	~2 TB	File
Experiment	Experiment	~500 TB	Relational

Synthetic input data

Synthetic population and activity data describe individuals and their activities. Synthetic activity data describe activities performed by individuals, which include travel and visits to locations of work, home, and daily errands. As part of these activities, people come in close contact with other individuals. The contact data capture this information and are central to the study of epidemics. The synthetic population database contains households, persons, activities, and location information, stored in a relational format. Schema descriptions of the synthetic population tables for household, person, and activity, activity location and home locations are given in Tables 5.2, 5.3, 5.4, 5.5, and 5.6. Descriptions of U.S. Census 2000 Public Use Microdata Sample (PUMS) demographic variables are available online at [123].

Table 5.2: Household related attributes.

Column	Description
state	FIPS state ID
county	FIPS county ID
tract	Census tract
blkgrp	Census blockgroup
hid	Unique household ID
persons	Number of persons in the household
vehicl	Census household demographic
hloc	Home location ID
serialno	Serial number of the PUMS household used to generate this synthetic household
hinc	Household income
bldgsz	Units in structure
busines	Business on property
fuel	Heating Fuel
hhl	Household language
hht	Household family type
p18	Number of people under 18 years in household
p65	Number of people 65 years and over in household
value	Property Value
workers	Number of workers in the household
p_gt_18	Number of adults in the household (age >18)
p_lt_19	Number of children in the household (age <19)
subloc_1	The sub-location assignment for this household within the home location

Disease manifestation and intervention data

Experiments are divided into multiple cells. Each cell characterizes one set of quantified simulation conditions and may have many replicates (10 is typical). Numerous intervention actions are applied to the epidemic simulation for decision making processes. Social distance, school closure, work closure, vaccinations, and antivirals are some of the intervention types. Interventions may be implemented at a predetermined time (e.g., on day 1), or when

Table 5.3: Person related attributes.

Column	Description
hid	Household ID
pid	Person ID
age	Age of person
relate	Relationship
sex	Sex
esr	Employment Status Recode
occcen5	Occupation (Census) for 5% File
abgo	Able to Go Out Disability
enroll	School Enrollment; Attended since February 1, 2000
grade	School Enrollment: Grade Level Attending

Table 5.4: Activity related attributes.

Column	Description
hid	Household ID of the person
pid	Person ID
patternid	ID assigned to this set of activity sequences for this person
anum	The number of this activity in the activity sequence. First activity has the lowest number, etc.
purpose	The purpose of this activity. Valid values for the US population are: 1 - home activities 2 - work activities 3 - shopping activities 4 - other activities 5 - school activities
starttime	The start time of the activity in seconds past midnight
duration	The duration of the activity in seconds
location	The location ID where the activity occurs
subloc_1	The sublocation ID within the location where the activity occurs
mode	The transportation mode used to arrive at the activity. Valid values are: 1 - walk, 2 - automobile, 3 - transit, 4 - airplane, 5 - other
firstathome	1 if the first activity in the sequence starts at home, 0 otherwise
surveyhh	The ID of the survey household that was matched to this person's household

a certain condition is met (e.g., when 1% of school-aged children are diagnosed). The goal of the intervention is to interrupt disease transmission from one person to another. Interventions may be targeted to specific demographic groups, or subpopulations, such as school-aged children or workers in critical jobs (e.g., first responders). SIBEL experiment

Table 5.5: Activity location-related attributes.

Column	Description
id	The location ID
x	Longitude of the location
y	Latitude of the location
z	Altitude of the location
zoneid	Assigned zone for the location. The census tract can be used for the zone. The ID for a census tract zone has the following format: SSCCCTTTTTT where SS = FIPS State CC = FIPS County TTTTTT = Census Tract
state	FIPS state
county	FIPS county
tract	Six characters designated census tract
blockgroup	Census block group
linkid	NAVTEQ link ID closest to the location
work	Attractor value for work activities
school	Attractor value for school activities
college	Attractor value for college activities
shopping	Attractor value for retail activities
hospital	Is location a hospital designator
other	Attractor value for other activities

setup data are stored in both the relational database and the file system. The relational database mainly stores experiment, analysis, disease model, and intervention information (Tables 5.7, 5.8, 5.9, and 5.10) [3]. Configuration information is stored in the file system (Table 5.11).

Table 5.6: Home location-related attributes.

Column	Description
id	The location ID
x	Longitude of the location
y	Latitude of the location
z	Altitude of the location
state	FIPS state
county	FIPS county
tract	FIPS census tract
blockgroup	FIPS block group
hloctype	Type of residential unit. Values are from census type designators for housing units
ctb_id	County, tract, blockgroup designator for this home location.

Table 5.7: Experiment related attributes.

Column	Description
Cells	Number of cells in the experiment.
Id	Each experiment is assigned a unique sequential ID number, generated automatically by the system
Name	The user may provide a Name for each experiment.
Region	This is a unique name for the geographic region within which the experiment is performed. The name of the region is specified at the time the region database is generated by NDSSL
Status	The current status of the experiment. May be one of the following six values: New – just created, not yet executed experiment, Starting – initializing execution environment and preparing to run, Queued – ready to run as soon as computing resources are available, Running – currently executing on the IDAC cluster, Completed – simulations completed, and data ready for analysis on the IDAC cluster, Failed – failed to achieve normal termination.

Table 5.8: Analysis related attributes.

Column	Description
id	Unique ID number assigned by the system.
name	Each analysis may be assigned a unique name by the analyst for reference purposes. The system does not use this Name field – all pointers in the software are specified by the ID number.
category	Analysis may be done on raw epi curves, or on Reproduction Number (actual and estimated). The category is set by the Owner.
owner	The Username associated with the login account from which this analysis was specified.
status	Operational condition of the analysis. Values are: New, Starting, Running, or Completed.
type	Type of analysis employed. Examples of Plot Types are: plots all of the epicurves or cumulative epicurves (replicates) for each cell in the combined experiments, the mean of the replicates for each experimental cell and displays them on the same plot, the attack rate for the cells in the combined experiment.

Table 5.9: Disease model related attributes.

Column	**Description**
id	The diseases model ID. Disease models are prepared by NDSSL.
name	Disease model name
transmissibility	The rate of transmission
incubation period	From the time a node becomes exposed until the time it becomes infectious.
infectious period	From the time a node becomes infectious until the time it becomes removed.

Table 5.10: Interventions related attributes.

Column	Description
id	The intervention ID.
name	Intervention examples are: vaccinate, antiviral, social distance, close work, close school.
description	Description of the intervention.

Table 5.11: Experiment configuration and output file related attributes.

File	Stored Information
Configuration	contact graph file, contact graph file format, simulation duration, transmissibility, incubation period format, incubation period file, infectious period format, infectious period file, epidemic seed type, epidemic seed number, iteration number, output file, output level, log file, intervention file, simulation random seed, config version.
Output	incubation duration, symptomatic duration, iteration, exposure day, infector.

Output data

Output data describe the spread of disease through a population. This generally contains three parts [4]. First is a list of infected people along with information about the time and duration of incubation and infection, and results of interactions with health care providers such as whether or not they were diagnosed. Second, it contains a set of trees with the initially infected persons as the roots, showing the path of infection through the population. This is referred to as a dendrogram. Third is a list of interventions and the time at which they were implemented. Multiple dendrograms across replicates and cells are used to produce tabular data (e.g., epidemic curves, intervention rankings, interaction effects, etc.).

The datasets are distributed across several different databases, schemas, file systems, and machines. The scale of the data, and use of fragmented storage, makes tracking experiment setups and scientific workflows cumbersome and error prone, and therefore

limits verification and reproducibility of results (Table 5.11).

5.1.2 Mapping Layer

Next, we discuss the second layer in EpiK as depicted in Figure 5.1. This layer maps all the datasets to RDF. We briefly discuss our rationale for doing this.

Why RDF?

Our choice to represent all data using RDF comes from similar considerations by other researchers using RDF in the field of bio-medical and health sciences. We briefly discuss the rationale below.

First, epidemiologically relevant datasets come in various forms, are large, and are growing rapidly as SIBEL continues to be used by policy analysts. Often new kinds of datasets must be accommodated. This requires frequent schema changes when using a relational database management system (RDMS), but is easily handled with RDF. Commercial relational databases can be useful in some cases but their cost and licensing rules limit their use and increase the cost of using SIBEL.

Second, the RDF data model is a good choice for epidemiologists as it satisfies the prerequisites listed earlier. It is a standard data model recognized by the World Wide Web Consortium (W3C) [20]. RDF permits the consolidation of various data models (tree, relational, graph, and so on) and vocabularies [124]. The data model offers a metadata structure more dynamic than relational databases. RDF is much more suitable for handling growing graph-oriented dynamic data [124].

Third, RDF-based storage allows us to identify data on the internet with URIs. This allows the creation of globally unique names and is useful, as multiple stakeholders across the world currently use SIBEL with little direct coordination. Using RDF makes it possible to distribute our data as linked open data (LOD) on the internet using the RDF data model. This makes it easy to link our datasets to pertinent internal and external datasets from the LOD cloud (for example, BioModels, BioPortal, BioGateway, BioSamples, and so on.) [125, 126]. Linking these and other social media datasets is needed (or will be in the near future) to understand complex questions arising in anti-microbial resistance and phylodynamic analysis.

Fourth, SPARQL is a natural language for studying questions arising in epidemic modeling, given that a number of questions in the field pertain to traversing social contact networks or disease dendrograms. Complex queries that need multiple combinations in SQL are relatively easy to create in SPARQL [127]. Moreover, transitive closure style queries (e.g., looking for paths in a network) are easier in SPARQL than SQL.

Finally, using RDF allows utilization of a various publicly accessible tools for data browsing, link discovering, and visualizing purposes. Very little effort is needed to implement these tools. Rather than creating a personalized graphical user interface (GUI) for data browsing, epidemic experts can utilize current faceted programs to analyze data with the help of basic ontologies [128, 129, 130, 131]. Any change in the data or ontology will automatically reflect in the GUI. Most of these tools are freely available on the web. Although these tools have some limitations, they provide a quick view on top of the data with no cost.

Table 5.12: Example of the RDF triples generated by tuple-based and value-based mappings, a sample query, its SPARQL implementations, and results.

<table>
<tr><th>Tuple-Based Mapping</th><th>Value-Based Mapping</th></tr>
<tr><td>@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix vocab: <http://ndssl.vbi.vt.edu.dl/vocab/>.

<http://ndssl.vbi.vt.edu.dl.TupleBasedMapping/HOUSEHOLD/55274861>
a vocab:HOUSEHOLD ;
rdfs:label "HOUSEHOLD #55274861" ;
vocab:HOUSEHOLD_HID "55274861"^^xsd:decimal ;
vocab:HOUSEHOLD_STATE "ALABAMA" ;
vocab:HOUSEHOLD_ZIPCODE "35633" .

<http://ndssl.vbi.vt.edu.dl.TupleBasedMapping/PERSON/215016716>
a vocab:PERSON ;
rdfs:label "PERSON #215016716" ;
vocab:PERSON_AGE "33"^^xsd:decimal ;
vocab:PERSON_HID "55274861"^^xsd:decimal ;
vocab:PERSON_PID "215016716"^^xsd:decimal ;
vocab:PERSON_SEX "MALE" .</td><td>@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix vocab: <http://ndssl.vbi.vt.edu.dl/vocab/>.

<http://ndssl.vbi.vt.edu.dl.ValueBasedMapping/HOUSEHOLD/55274861>
a vocab:HOUSEHOLD ;
rdfs:label "HOUSEHOLD #55274861" ;
vocab:HOUSEHOLD_HID "55274861"^^xsd:decimal ;
vocab:HOUSEHOLD_STATE "ALABAMA" ;
vocab:HOUSEHOLD_ZIPCODE "35633" .

<http://ndssl.vbi.vt.edu.dl.ValueBasedMapping/PERSON/215016716>
a vocab:PERSON ;
rdfs:label "PERSON #215016716" ;
vocab:PERSON_AGE "33"^^xsd:decimal ;
vocab:PERSON_HID
<http://ndssl.vbi.vt.edu.dl.ValueBasedMapping/HOUSEHOLD/55274861>;
vocab:PERSON_PID "215016716"^^xsd:decimal ;
vocab:PERSON_SEX "MALE" .</td></tr>
<tr><td>Query: Get all information related to person 215016716.

SPARQL: SELECT ?property ?hasValue
WHERE {
{ <http://ndssl.vbi.vt.edu.dl.TupleBasedMapping/PERSON/215016716>
?property ?hasValue
}</td><td>Query: Get all information related to person 215016716.

SPARQL: SELECT ?property ?hasValue ?isValueOf
WHERE {
{ <http://ndssl.vbi.vt.edu.dl.ValueBasedMapping/PERSON/215016716>
?property ?hasValue
UNION
{?hasValue vocab:HOUSEHOLD_HID ?isValueOf }
}</td></tr>
<tr><td>Result:
property hasValue
vocab:PERSON_AGE 33
vocab:PERSON_HID 55274861
vocab:PERSON_PID 215016716
vocab:PERSON_SEX "MALE"
rdf:type vocab:PERSON
rdfs:label "PERSON #215016716"</td><td>Result:
property hasValue isValueOf
vocab:PERSON_SEX "MALE" -
vocab:PERSON_AGE "33" -
vocab:PERSON_PID "215016716" -
rdfs:label "PERSON #215016716" -
rdf:type vocab:PERSON -
vocab:PERSON_HID <PERSON/215016716> -
<PERSON/215016716> "55274861"</td></tr>
</table>

Table 5.13: Partial D2RQ mapping implementation of the value-based mapping (join part only).

```
map:person_hid a d2rq:propertybridge;
  d2rq:belongstoclassmap map:person;
  d2rq:property vocab:person_hid;
  d2rq:propertydefinitionlabel "person hid";
  d2rq:referstoclassmap map:household;
  d2rq:join "person.hid =>household.hid";
  ..........................................................................
```

Mapping to RDF

We would like to unify heterogeneous and fragmented computational networked epidemiology pipeline (CNEP) datasets into RDF format. (We interchangeably use the words CNEW and CNEP in this dissertation.) That requires relational to RDF mapping. Several mapping strategies exist in the literature along with tools that use mapping to build an RDF dataset [35, 36, 37, 38, 39]. In this work we investigate tuple-based and value-based mapping techniques [132], using a D2RQ tool to generate both.

Tuple-based and Value-based mapping

In tuple-based mapping, the RDB table name is considered as an RDF class name, and a column name as a property name. For each instance of the table a unique blank node is created. Naming of the blank node can be done in various ways but incremental number assignment is the simplest way. A tuple of an instance contains a blank node, a label edge, and a literal. The property name is used as an edge label and each of the property values is used as a literal of the tuple (Figure 5.3). Tuple-based mapping uses relational database tables as input and produces an RDF graph as output, ignoring the primary key foreign key relationship.

Primary key and foreign key join exist in the database and are preserved in value-based mapping. The input of value-based mapping is relational tables and output is an RDF graph with join information. Primary key attribute values are presented by URIs. In value-based mapping the primary key attribute value of the parent object points to the child object (Figure 5.4).

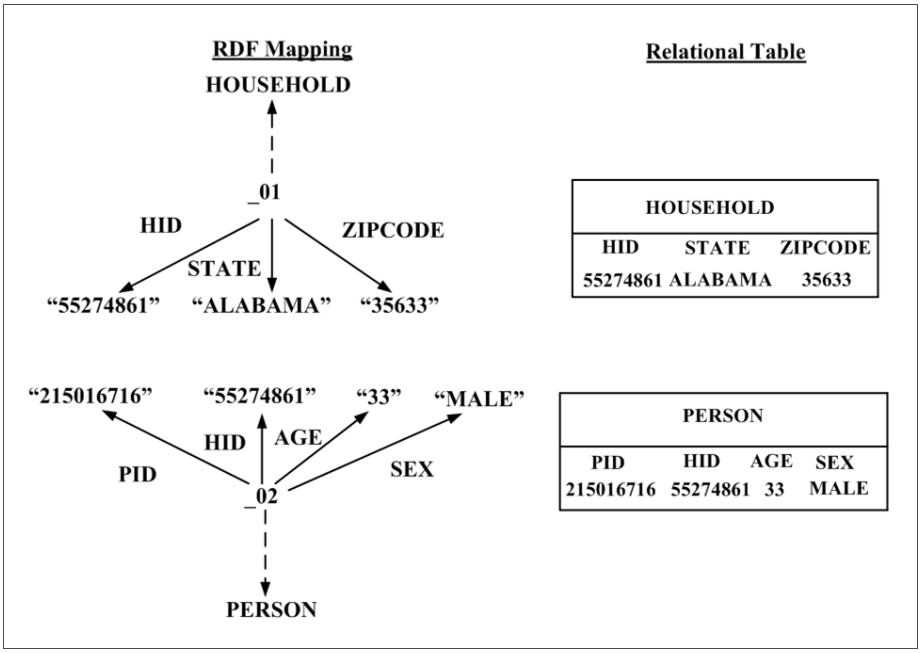


Figure 5.3: Tuple-based mapping example. Here, we have two datasets (Person and Household). The tuple-based mapping disregards the primary key and foreign key relationship in RDF mapping. Here in RDF Mapping PERSON (_01 is a class instance) and HOUSEHOLD (_02 is a class instance) are two classes. We represent class instances by using a dotted arrow.

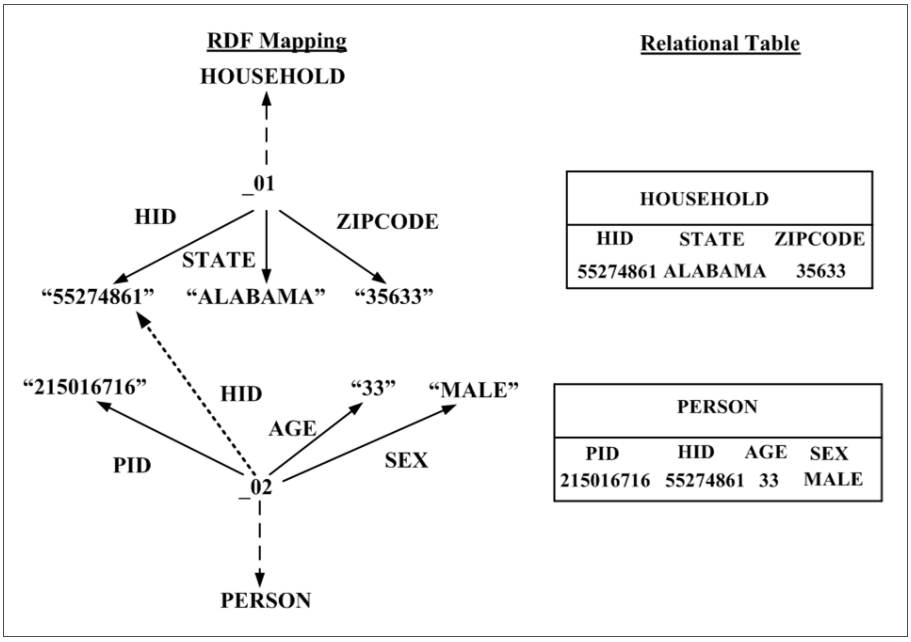


Figure 5.4: Value-based mapping example. The value-based mapping considers the primary key and foreign key relationship in RDF mapping (dotted arrow in the figure). Here, "HID" is the primary key in the Household table and foreign key in the Person table.

Tuple-based and value-based mappings are formally defined below.

Preliminaries. Let S be a relational schema and $A(S) = \{A_1, ..., A_k\}$ indicate the attributes of S. I denotes an instance of S and $T = (a_1, ..., a_k)$ represents every tuple of I. We use C_S as a class over S with properties $P(C_S) = \{P_{S,A_1}, ..., P_{S,A_k}\}$. We use F_C to denote the finite set of classes. Mapping creates an RDF graph G such that for every tuple T in I one creates a node n_T and an edge $(n_T, rdf{:}type, C_S)$. Tuple-based and value-based mappings have differences in edge creation.

Tuple-based mapping. Let $T.A$ be the value of an attribute A. For every non-null value $T.A$ of T, $A \in A(S)$, tuple-based mapping creates an edge $(n_T, P_{S,A}, (T.A)_{n_T,P_{S,A}})$. This type of mapping ignores primary key and foreign key relationships.

Value-based mapping. We use C_S to denote a child class and $C'_{S'}$ for parent class. Let $C_S, C'_{S'} \in F_C$. We write $T.A$ to represent a non-null value of an attribute A over C_S and $T'.A'$ for a non-null value of an attribute A' over $C'_{S'}$. $P_{S,A}$ represents the C_S properties and $P'_{S',A'}$ represents $C'_{S'}$ properties. Value-based mapping contains primary key and foreign key relationships information. For that, if $T.A$ of T contains a foreign key value and $T'.A'$ of T' contains a primary key value then an edge $(n_T, P_{S,A}, (T'.A')_{n'_{T'},P'_{S',A'}})$ is created in value-based mapping to represent the relationship. For every other non-null value of $T.A$ of T introduce an edge $(n_T, P_{S,A}, (T.A)_{n_T,P_{S,A}})$.

Mapping implementation

We use the D2RQ mapping language to convert relational data to RDF graphs. It is a declarative language that generates mapping files from the table structure of a database. The mapping file is used to generate an RDF graph through resource identification and property value generation techniques. D2RQ maps database schemas to RDFS/OWL schemas. D2RQ *ClassMap* maps database records to RDF classes of resources. *ClassMap* has a number of *PropertyBridges* that specify how resource descriptions are created. Resources are identified by using URI patterns. For example:
uriPattern:seattle/@@person.pid@@ creates URI like *seattle/person/215016716*. As a property value D2RQ supports literals, URI, or blank nodes. A property value can be created directly from a database or by using patterns. The D2RQ *join* facility enables us to perform primary key foreign key joining. The following propertyBridge definition (Table 5.13) creates property **person:hid** and also shows the *D2RQ* join facility.

In Table 5.12 we present corresponding RDF triples of Figures 5.3 and 5.4. To demonstrate the difference between the two types of mapping, we present the results of a query. Results show that value-based mapping captures data link information unlike tuple-based mapping because it is capable of storing primary and foreign key relationships.

5.1.3 RDF Engine and Services Layer

We now discuss the third layer of EpiK (Figure 5.1), focusing on the RDF engine, the query language, and various services. Querying over the RDF generated by using mapping requires an RDF data engine and a querying interface (web frontend for the query service). We evaluate Virtuoso DB and Jena TDB as the RDF data engine. Virtuoso is an open source framework for developing semantic web and linked data applications, and Jena is a useful tool for processing RDF data. The querying interface is granted through the Fuseki server and Virtuoso server. Both of these servers provide SPARQL endpoints.

In addition to querying, the framework allows faceted browsing using a Virtuoso Facet Browser. This capability offers browsing over billions of triples, full-text search, structured querying and result ranking capabilities. This facility allows epidemiologists to navigate through complete epidemiology workflow, exploring experiment, synthetic population, disease model, interventions, and analysis information.

5.2 Query Bank

We studied two classes of queries; they are summarized in Tables 5.16, 5.17, and 5.18. The queries in Table 5.16 are created from our discussions with epidemiologists, while queries in Table 5.17 are created by following BSBM SPARQL benchmark queries. The queries in Table 5.18 are standard D2RQ benchmark queries.

5.2.1 Native Queries Based on 5W1H Approach

We employ a **5W1H** interrogative approach to classify the queries [133, 134, 135]. This approach draws on rhetorical theory and structuring and better captures the social and epidemiological context [136]. The 5W1H approach classifies a query in six dimensions, namely: (*i*) WHO: (human/participants); (*ii*) WHAT: (object/content); (*iii*) WHEN: (time); (*iv*) WHERE: (space/location), (*v*) WHY: (purpose/behavior); and (*vi*) HOW: (behavior/form). See [135, 136, 137, 138] for additional discussion of structuring queries and knowledge-bases using the 5W1H approach.

In Table 5.31, we present mapping between 5W1H questions and RDF property vocabularies. The table shows that it is possible to construct a large corpus of queries using the 5W1H approach. The first column exhibits 5W1H question types, the second column provides question types description, and finally the last column shows an example RDF data property vocabulary list. The wildcard (*) in the vocabulary means syntax variation is possible. This list can be used in the predicate part of RDF triples (for example, "has_age", "has_state", etc.). We provide an example SPARQL query in Table 5.32.

Table 5.14: Example of 5W1H questions mapped to RDF data property vocabularies. A partial example of the 5W1H questions' vocabulary domain is shown for brevity. This vocabulary list is expandable based on the properties we present in Tables 5.2–5.11. Here, the asterisk symbol represents the wildcard (a character or sequence of characters).

5W1H Question	**Definition**	**Example Property Vocabulary**
WHO	property used to refer to people	**Person Class:** *pid
WHAT	property refers to specific information	**Person Class:** *age, *sex, *esr, ... **Household Class:** *hid, *hinc, *fuel, ... **Activity Class:** *anum, ...
WHERE	property refers to place or location	**Household Class:** *state, *county, *blkgrp, ... **Location Class:** *latitude, *longitude, ...
WHEN	property refers to time or occasion	**Activity Class:** *starttime, *duration, ...
WHY	property refers to reason or cause	**Activity Class:** *purpose, ...
HOW	property refers to the manner something is done	Queries that need SPARQL aggregate function like COUNT, SUM. etc.

Table 5.15: SPARQL implementation of a sample 5W1H query. Here, we are showing a WHAT query. In this query, we assume the infected person ID is given. The vocabulary "has sex" fits the WHAT type property vocabulary (*sex).

Query: What is the gender of an infected person? **Sample SPARQL Implementation:**

@prefix ndssl:<http://ndssl.bi.vt.edu/>
@prefix vocab:<http://ndssl.bi.vt.edu/vocab/>
SELECT ?infected_person_gender
WHERE {
ndssl:person#1 vocab:has_sex ?infected_person_gender
}

This query bank was created after a number of discussions with epidemiologists who used SIBEL and other similar tools over the years. Table 5.16 summarizes the queries generated through this effort, chosen to highlight the need to synthesize varied datasets to answer them. For example, to answer a query such as: *What is the household location and the daily activity of an infected individual?* requires one to access *Experiment, Dendrogram, Person, Household, Location, and Activity* datasets. Some parts of the experimental data exists in a relational database, and others are stored in files. As discussed earlier, we map all datasets into RDF. SPARQL can then be used to answer the query.

5.2.2 Benchmark Queries

It is important to compare the performance of our framework with current state-of-the-art research. However, there are no SPARQL benchmark queries in existence for the epidemiology domain. Therefore, we created a set of benchmark queries based on Berlin SPARQL Benchmark (BSBM) version 3.1 [139]. BSBM provides a suite of benchmark queries to compare the performance of SPARQL queries. It is developed on an e-commerce use case. We investigated BSBM and found that each query follows some standard properties. We used those to create a corresponding epidemiology benchmark query (BQ) list (Table 5.17), and implemented BQ to demonstrate the effectiveness of our mapping techniques with different query patterns.

D2RQ provides a set of benchmark queries to assess performance analysis [140]. We selected some D2RQ benchmark queries (DBQ) and executed them in our framework, using Benchmarking D2RQ v0.2.(Table 5.18).

5.3 Empirical Analysis of EpiK

As a proof of concept, we built a prototype implementation of EpiK. A set of computational experiments were undertaken to demonstrate the linked datasets and to compare the performance of the system as a function of various mappings and views.

5.3.1 Experimental Design

The overall design is comprised of the following variables: (*i*) a set of queries summarized in Tables 5.16, 5.17, and 5.18; (*ii*) two sets of RDF graphs, created by using value-based and tuple-based mappings summarized in Table 5.20, and (*iii*) virtual and materialized views of the underlying data. The choice with respect to style of representation (value-based vs. tuple-based) combined with the implementation of the execution platform (relational engine based (virtual) vs. RDF engine based (materialized)) provides us with four possible

Table 5.16: Queries collected from interviewing various epidemiologists (Native Queries). These are examples of queries frequently asked in the computational networked epidemiology domain. We present queries in plain English, their 5W1H question category, and datasets needed to answer the queries.

No.	**English Query**	**5W1H Question**	**Data Needed to Answer the Query**
Q1	What is the gender of an infected individual?	WHAT	Experiment, Dendrogram, Person
Q2	What is the household location of an infected person, where he has been in past few days, and why?	WHAT, WHERE	Experiment, Dendrogram, Person, Household, Location, Activity
Q3	How many preschool children's information is used in the Seattle strong flu experiment?	HOW	Experiment, Area, Subpopulations
Q4	What are the interventions used in the Seattle strong flu experiment?	WHAT	Experiment, Interventions
Q5	How many people become infected after the intervention?	HOW	Experiment, Dendrogram, Interventions
Q6	What is the demographic information of a given infected person?	WHAT	Experiment, Dendrogram, Person
Q7	How many people of a particular demographic are sick on the first day of the simulation?	HOW	Experiment, Dendrogram
Q8	Who infected whom of a particular demographic?	WHO	Experiment, Dendrogram
Q9	How long does a person remain in an infected state?	HOW	Experiment, Dendrogram
Q10	Given a demographic location and disease type, find an infected person in that location and find the activities he/she performed in the last few days?	WHO, WHAT, WHERE	Experiment, Dendrogram, Person, Activities

Table 5.17: BSBM benchmark queries, and their corresponding epidemiology queries, 5W1H question categories, and benchmark query properties. We employ experiment and intervention datasets to answer the queries.

No.	**BSBM Query**	**Corresponding Epidemiology Query**	**5W1H Question**	**Properties**
BQ1	Find products for a given set of generic features.	Find experiment information for a given set of generic properties.	WHAT	Touches a large amount of data, uses ORDER BY and LIMIT.
BQ2	Retrieve basic information about a specific product for display purposes	Retrieve basic information about a specific experiment for display purposes.	WHAT	Query touches only a small amount of data, Larger set of triple patterns, uses OPTIONAL
BQ3	Find products having a label that contains a specific string.	Find experiment having a name that contains a specific string.	WHAT	Query uses REGEX
BQ4	Retrieve in-depth information about a specific product including offers and reviews.	Retrieve in-depth information about a specific experiment including cell and intervention information.	WHAT	Touches lot of data, uses OPTIONAL
BQ5	Get information about a reviewer.	Get information about an intervention.	WHAT	Use DESCRIBE

Table 5.18: D2RQ benchmark queries used in the experimentation.

No.	D2RQ Benchmark Queries
DBQ1	find(s ? ?) on large table
DBQ2	find(s ? ?) with non-existing subject
DBQ3	find(? p o)
DBQ4	find(? ? o) with o being a resource that matches pattern
DBQ5	find(? ? o) with o being a resource that doesn't match pattern

Table 5.19: Tuple-based and value-based mapping files, creation times, sizes, and triple counts.

Mapping	Size (KB)	Number of Triples	Creation Time (min)
Tuple-Based	92	2 273	<1
Value-Based	92	2 276	<1

Table 5.20: Information related to RDF graphs created using tuple-based and value-based approaches. Here, we present RDF graph sizes, triple counts, RDF graph generation times, Jena TDB, and Virtuoso triplestores loading times.

Type	RDF Graph Size (GB)	Number of Triples	RDF Graph Generation Time (hr.)	Jena TDB Loading Time (hr.)	Virtuoso Loading Time (hr.)
Tuple-Based RDF Graph Synthetic Population, Experiment Setup, Output	85	308990904	2.0	2.32	0.82
Value-Based RDF Graph Synthetic Population, Experiment Setup, Output	102	339977981	2.35	2.75	1.11

design points for the proposed platform. Each design point has different trade-offs. Our experiments evaluate these decision choices using the set of queries discussed above.

5.3.2 Datasets

All of our experiments are conducted using a case study created for the Seattle region. The basic goal of the study was to understand the efficacy of various pharmaceutical and non-pharmaceutical interventions before and during an influenza outbreak. SIBEL is used to carry out the computational counter-factual experiments. The data used in the study and produced as a result of the study is stored in various places. As a first step, we convert all the data into a relational format and store them in Oracle and Postgres databases (Table 5.21). The dataset is briefly summarized in Table 5.21.

Table 5.21: Information related to the epidemic experiments carried out using Seattle synthetic data. We present their size and number of rows.

Databases	Size (GB)	Number Rows
Seattle Synthetic Population	2.33	49726461
Output	0.04	3237375
Experiment Database	0.004	4785

5.3.3 Software and Hardware Used

We use D2RQ (relational database to RDF mapping) version 0.8.1 as a mapping language [119], Virtuoso Open-Source Edition 7.1.0 (RDF triplestore and SPARQL engine) [122], Apache Jena 2.11.2 (RDF triplestore and SPARQL engine) [121], SPARQL [141] for querying the RDF graph, Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64-bit Production (relational database) [142], and Postgres version 8.3.14 (relational database) [143]. SIBEL simulations were run on ***Shadowfax*** [144]. We performed the Postgres experiment on ***Shadowfax*** and others on the ***Taos*** machine because of the software facilities available at NDSSL. ***Shadowfax*** contains 2500 CPU cores, 4 quad-socket large memory nodes, 28 TB of RAM, 1200 TB of high-performance GPFS storage for both scratch and home directories. ***Taos*** contains 16 CPU cores, 192 GB of RAM, and 5 TB of storage.

5.3.4 Creating the RDF graphs

We created tuple and value-based mapping files using the D2RQ language. A mapping file is also an RDF graph. D2RQ maps the table name to the RDFS class name, and column name to the property bridge. Most of the mapping generations take less than a minute. File sizes are relatively small, and therefore easy to maintain. The mappings create linked and storage-agnostic access to the heterogeneous data. Table 5.19 reports our mapping file size, number of triples in the mapping files, and mapping generation time. Next we apply mapping files to our data sources to produce RDF graphs (Table 5.20). The value-based RDF graph size is larger than the tuple-based RDF graph size because of the primary key and foreign key link information. Hence, a 10% increase in the number of triples causes a 20% increase in the RDF graph size. After generating the RDF graph it is loaded into the Virtuoso and Jena TDBs to publish data and execute queries. Mapping files create a link between various heterogeneous data sources, thus solving our data linking problems.

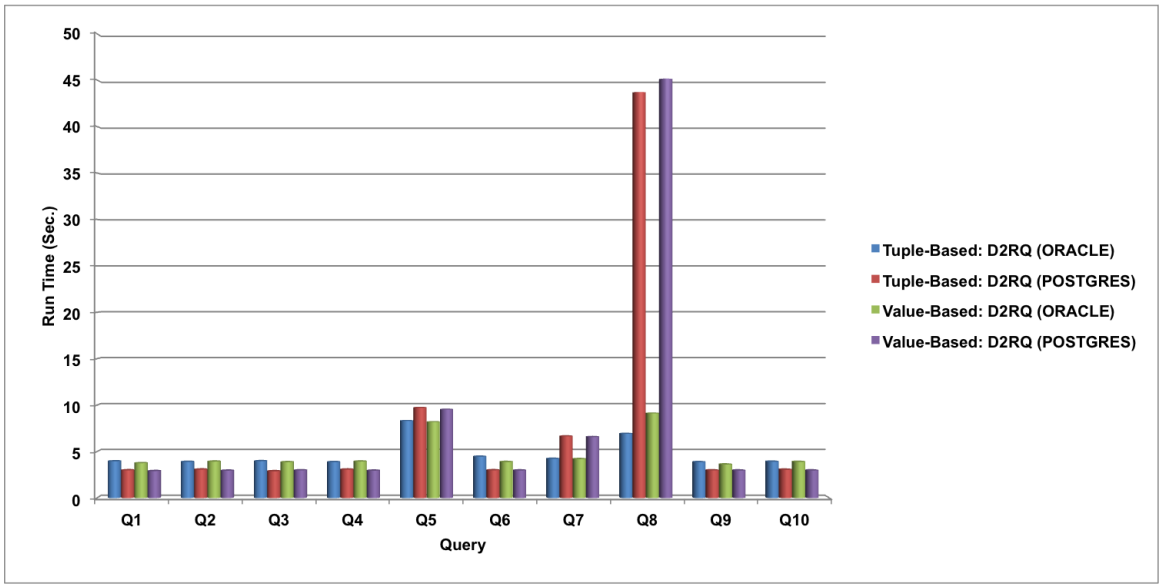


Figure 5.5: Native query performance over a virtual RDF graph. Here, we present the performance for both tuple and value-based mappings by employing Oracle and PostgreSQL databases.

5.3.5 Results

We used SPARQL to implement all queries listed in Tables 5.16, 5.17, and 5.18. For example, the SPARQL query shown in Table 5.22 indicates retrieval of information on the number of preschool children, which is a data type used in the Seattle case study. We then executed queries to measure the strength of mapping approaches over various types of RDF graphs. Figures 5.5 and 5.6 show query execution performance of our mapping approaches over D2RQ with the Oracle database, D2RQ with Postgres database, Jena TDB, and Virtuoso tools, respectively.

Observation 1. For both virtual and materialized RDF graphs, value-based mapping performs better than tuple-based mapping with native queries (except Q8). Figure 5.5 shows that "Find infector and infectee information on a particular demographic" query (Q8) on PostgreSQL takes longer to run because it outputs many result triples and depends on the system RAM.

Observation 2. Performance results for benchmark queries are shown in Figures 5.7(a), 5.7(b), 5.8(a), and 5.8(b). It can be seen that tuple-based mapping with the Virtuoso tool performs better in this case.

Observation 3. Our experiment results show that a SPARQL query that contains a regular expression (BQ3) performs faster with tuple-based mapping and an Oracle tool for a virtual RDF graph. On the other hand, for a materialized RDF graph, the value-based mapping is faster (for BQ3) and provides the same performance for both the Jena TDB and Virtuoso tools.

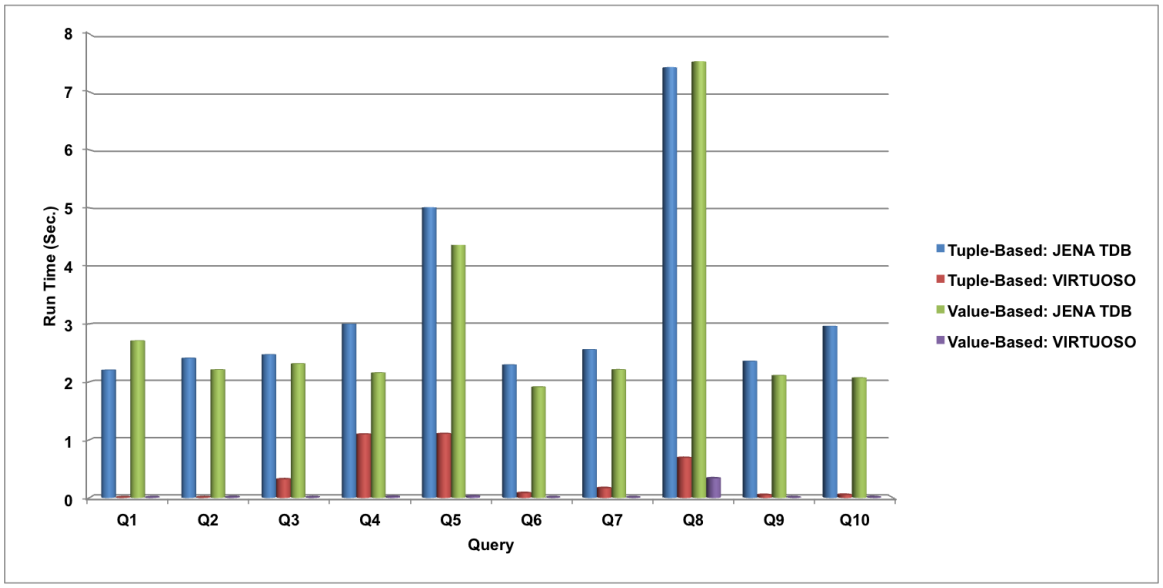


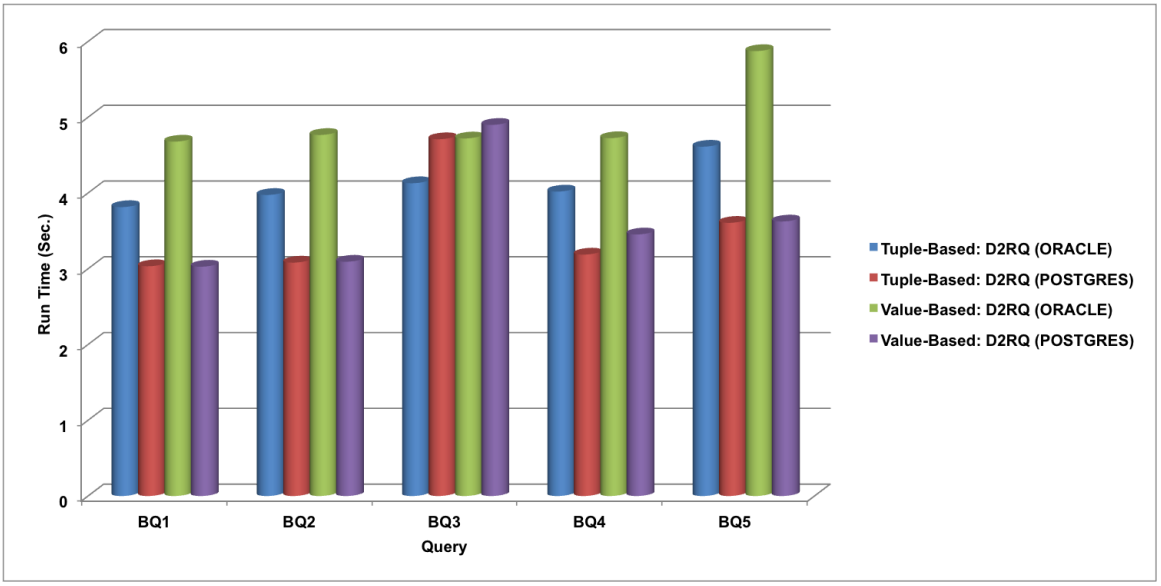
Figure 5.6: Native query performance over a materialized RDF graph. Here, we present the performance for both tuple and value-based mappings by using Jena TDB and Virtuoso triplestores.

Table 5.22: SPARQL implementation of the query "How many preschool children's information is used in Seattle strong flu (flu that infects 30% of the population) experiment?"

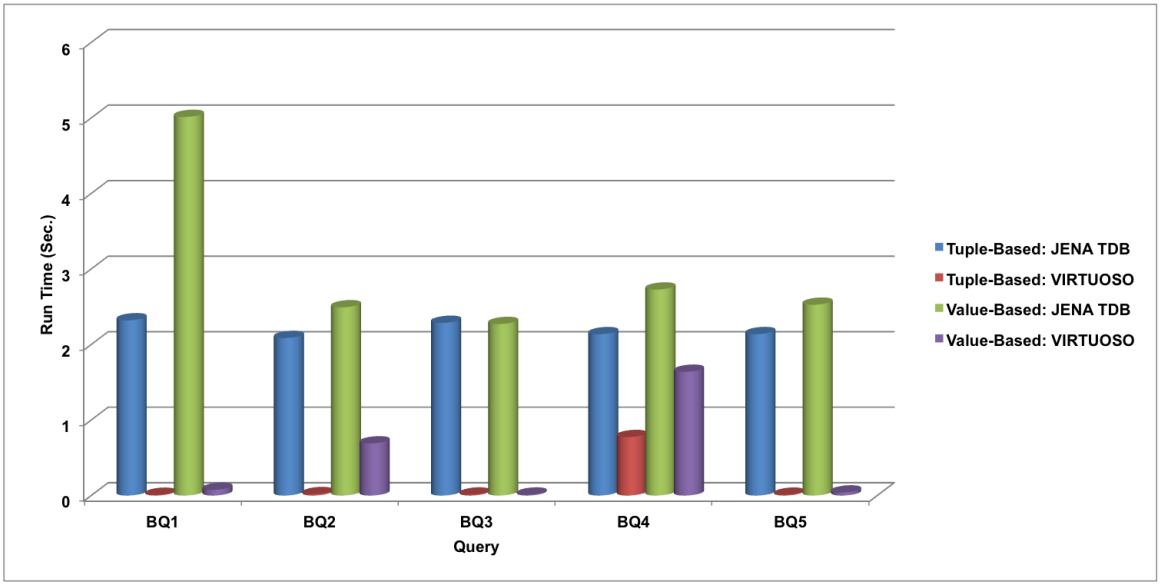
```
@prefix vocab: <http://ndssl.vbi.vt.edu.dl/vocab/> .
@prefix rdf:
<http://www.w3.org/1999/ 02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

select ?s ?p ?o where {
?s <http://ndssl.vbi.vt.edu.dl/vocab/AREA_ID >
'5'^^ <http://www.w3.org/2001/XMLSchema#decimal>.
?s <http://ndssl.vbi.vt.edu.dl/vocab/DESCRIPTION>
"Seattle children with age less than 5 years".
?s <http://ndssl.vbi.vt.edu.dl/vocab/POPULATION_SIZE>?o
}
```

5.4 Analytics with EpiK

We briefly describe the kinds of analytics that can be done using EpiK. The goal is not to be exhaustive but to convey the utility of developing an environment such as EpiK. We describe three simple examples, chosen to: (*i*) illustrate the value of developing a federated data representation so that insights can be obtained by reasoning over multiple datasets

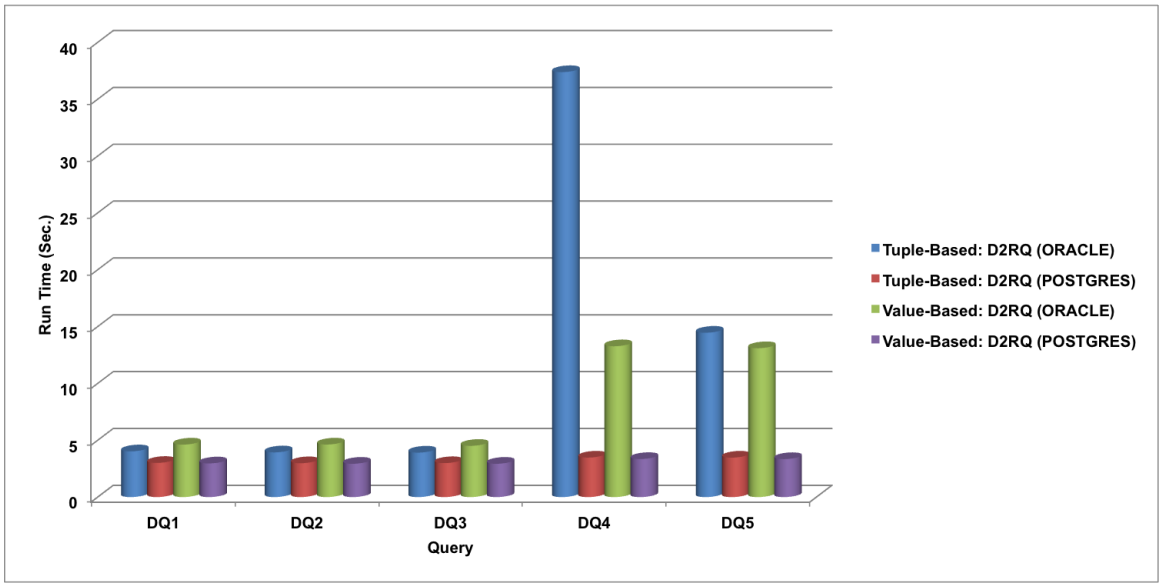


(a) BSBM-like epidemiology query performance over the virtual RDF graph. Here, we present the performance for both tuple-based and value-based mappings by employing Oracle and PostgreSQL databases.

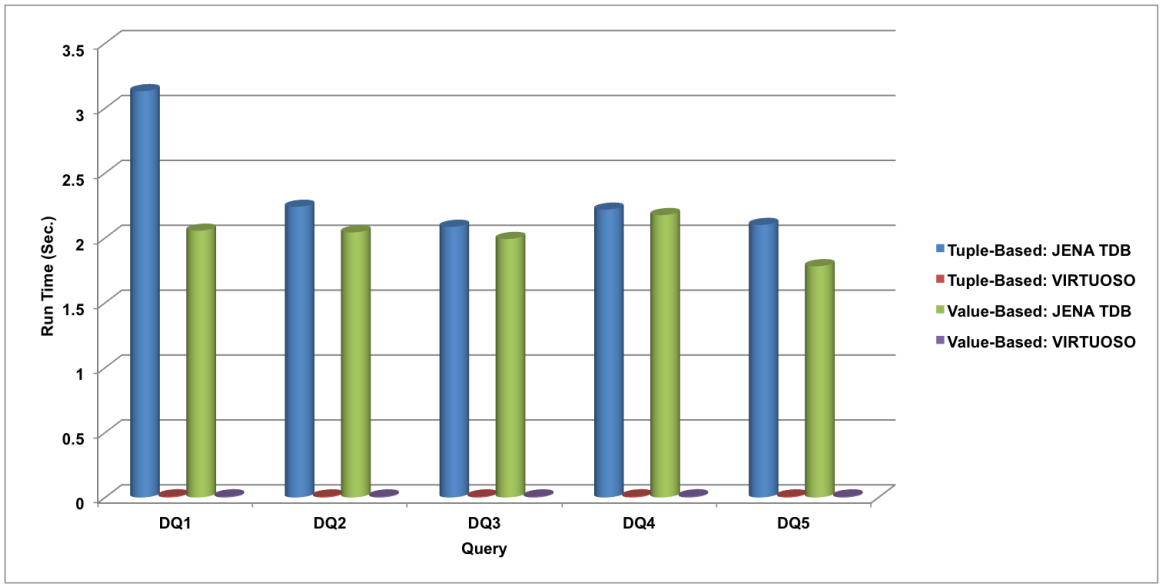


(b) BSBM-like epidemiology query performance over a materialized RDF graph. Here, we present the performance for both tuple and value-based mappings by using the Jena TDB and Virtuoso triplestores.

Figure 5.7: BSBM-like epidemiology query performance over virtual and materialized RDF graphs.



(a) D2RQ benchmark query performance over a virtual RDF graph. Here, we present the performance for both tuple and value-based mappings by employing Oracle and PostgreSQL databases.



(b) D2RQ benchmark query performance over a materialized RDF graph. Here, we present the performance for both tuple and value-based mappings by using Jena TDB and Virtuoso triplestores.

Figure 5.8: D2RQ benchmark query performance over virtual and materialized RDF graphs.

simultaneously; (*ii*) illustrate the use of SPARQL and RDF in terms of query language and data representation; this allows us to develop network queries that are pertinent in epidemic analysis; and (*iii*) illustrate "realistic" studies that can be undertaken in an agent-based epidemic analysis. All the queries that arose in the examples below are done using EpiK. In our experimentation, we use a materialized RDF graph generated through the value-based mapping approach. The materialized RDF graph is constructed from various datasets used and generated by a one cell ten replicate influenza simulation study.

Example 1: Understanding the structure of the dendrograms. A basic task in an epidemic analysis is to understand the structure of the dendrograms. The dendrogram is a directed acyclic graph (DAG), upon which one can do basic analysis that involves the demographics of the infected individuals. We analyze ten dendrograms coming from ten replicates. We compute basic properties of a dendrogram: degree, path, and star distributions. We count distinct star patterns only and discard duplicates. For example, a five nodes star is not included in a six nodes star. However, duplication is considered in path counts. In our experimentation the minimum star length is four nodes. The degree, star, and path distributions of the ten replicates are shown in Figures 5.9, 5.10, and 5.11. Figure 5.9 shows that we have few high degree nodes and many low degree nodes. Similarly, Figures 5.10 and 5.11 show that we have a small number of long path (or large star) and many small paths (or stars). Figures 5.9 and 5.11 show that replicates are relatively similar for degree and star distributions. Furthermore, the distributions contain long tails. However, that is not the case for path distribution. Figure 5.10 shows that replicates have variations and do not have a long tail like the degree and star distributions. In Tables 5.23, 5.24, and 5.25 we provide detailed information of degree, path, and star counts. A summary of statistics across all the replicates is presented in Table 5.26.

Example 2: Understanding chains of transmissions in epidemics. During the course of H5N1 planning efforts, several groups including ours were interested in understanding long chains of epidemic transmission. The basic idea builds on Example 1 and tries to understand super-spreaders. In the epidemiology discipline, it is important to identify super-spreaders because many disease distributions indicate that 20% of the population cause 80% of a given disease's spread [145]. Hence super-spreader information can help policymakers to design precise interventions to effectively diminish an epidemic.

Intuitively a node (individual) is a super-spreader if the node infects many other nodes. The basic issue of interest is direct infections versus indirect infections. A *direct super-spreader* is a node that directly infects a large number of individuals. An *indirect super-spreader* is a node that is the root of a long chain of infections. Intuitively, these two transmission pathways are different. The first resembles a star while the latter is a long path in the dendrogram. Understanding the distribution of paths and stars in a dendrogram thus

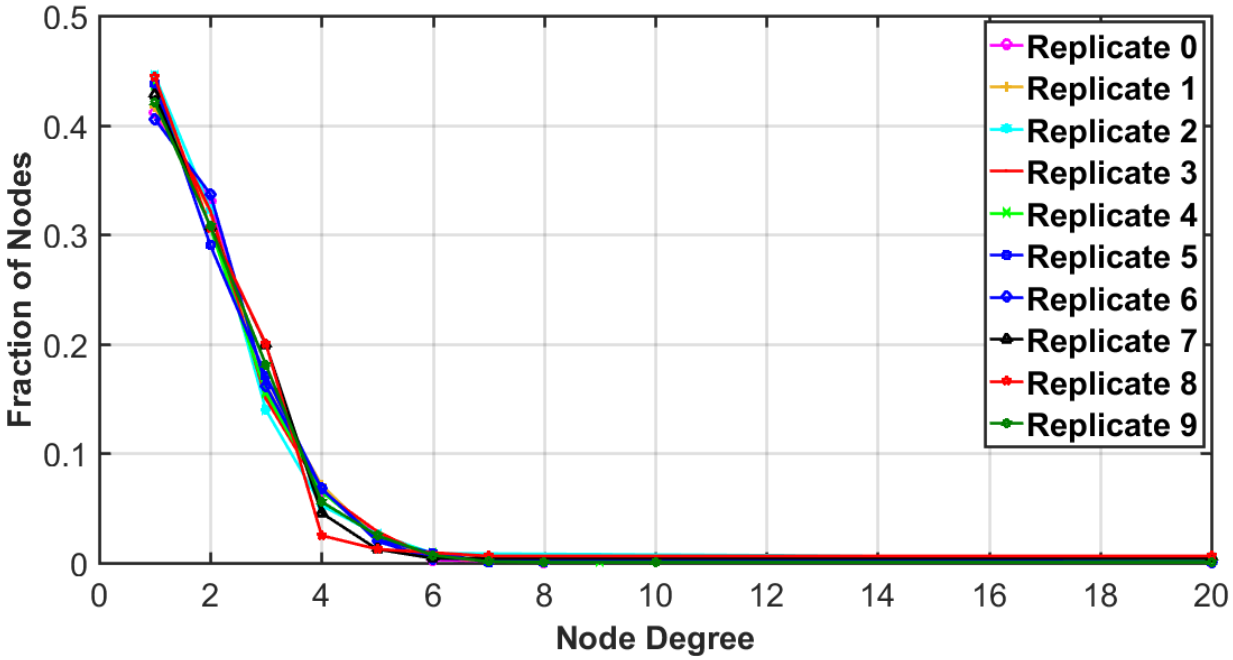


Figure 5.9: Degree distribution information of a ten replicates one cell experiment. For each replicate, we present node degree in the x-axis and fraction of nodes in the y-axis. Degree distributions for ten replicates are similar. We have a lower number of higher degree nodes and a higher number of lower degree nodes.

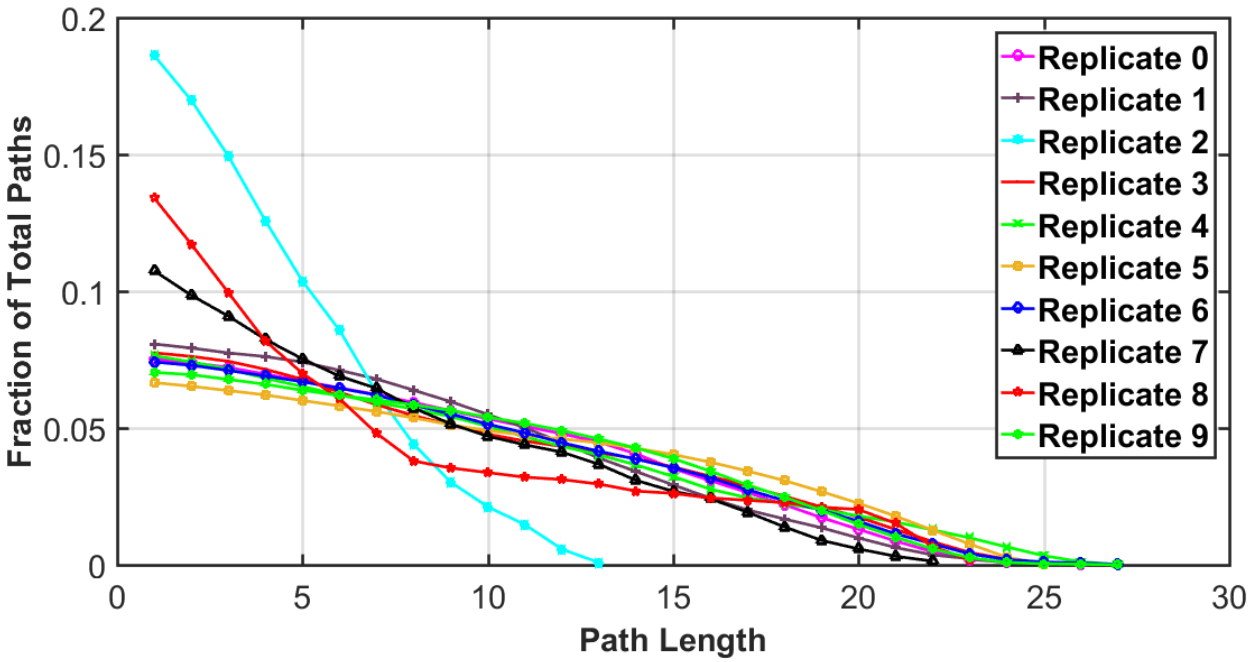


Figure 5.10: Path distribution information. We count the number of occurrences of the various length of paths in the dendrograms. Plots show the path length distribution across replicates is similar.

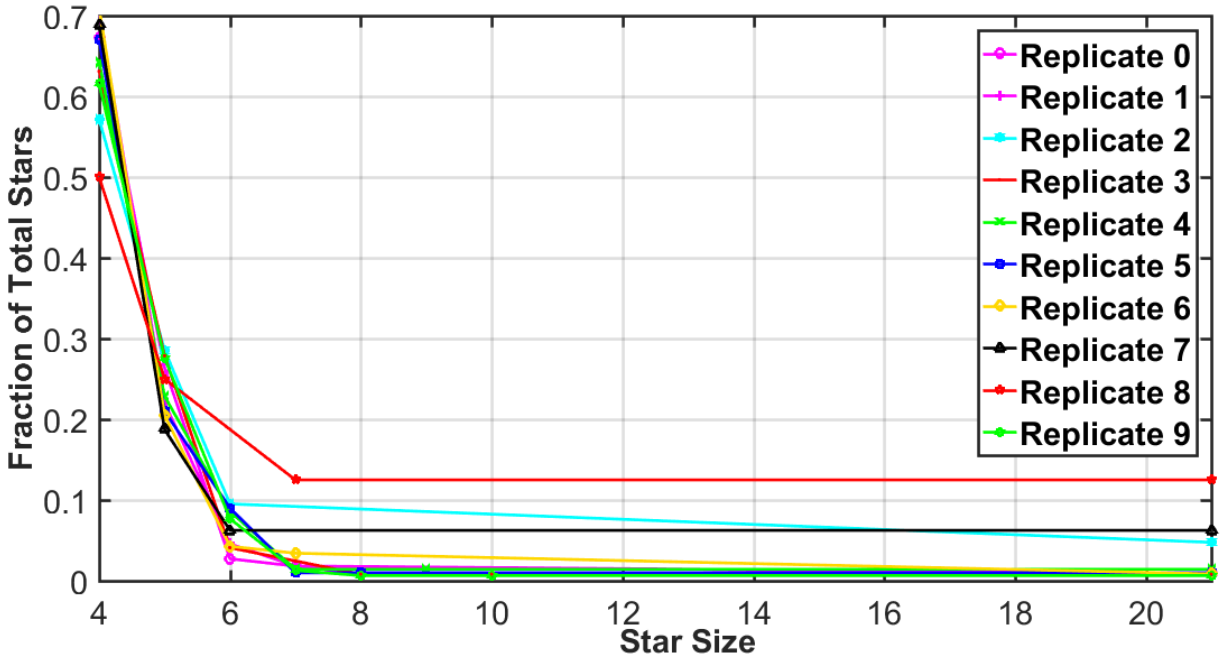


Figure 5.11: Distribution of the star patterns in the dendrograms. We present star size (number of nodes in a star) in the x-axis and fraction of total stars in the y-axis.

Table 5.23: Degree count information. The table presents maximum degree, minimum degree, mean degree, and standard deviation of degree for various replicates.

Degree Count						
	Maximum		Minimum		Mean Degree	Standard Deviation of Degree
	Degree	Number of Nodes	Degree	Number of Nodes		
Replicate 0	20	1	1	466	6.2	5.67
Replicate 1	20	1	1	452	6	6
Replicate 2	20	1	1	102	5.86	6.47
Replicate 3	20	1	1	502	6.6	5.46
Replicate 4	20	1	1	301	6.3	5.7
Replicate 5	20	1	1	433	6.2	5.65
Replicate 6	20	1	1	495	6	6
Replicate 7	20	1	1	103	5.86	6.47
Replicate 8	20	1	1	71	6	6.48
Replicate 9	20	1	1	724	6.6	5.46

provides us with an understanding of the transmission structure. An analysis across multiple dendrograms tells us how robust the conclusions are within a single cell of the experiment (and thus provides a basis for sensitivity analysis). We present path and star distributions in Figures 5.10 and 5.11 (count information in Tables 5.24 and 5.25). The figures clearly show a small number of occurrences of super-spreaders across different replicates.

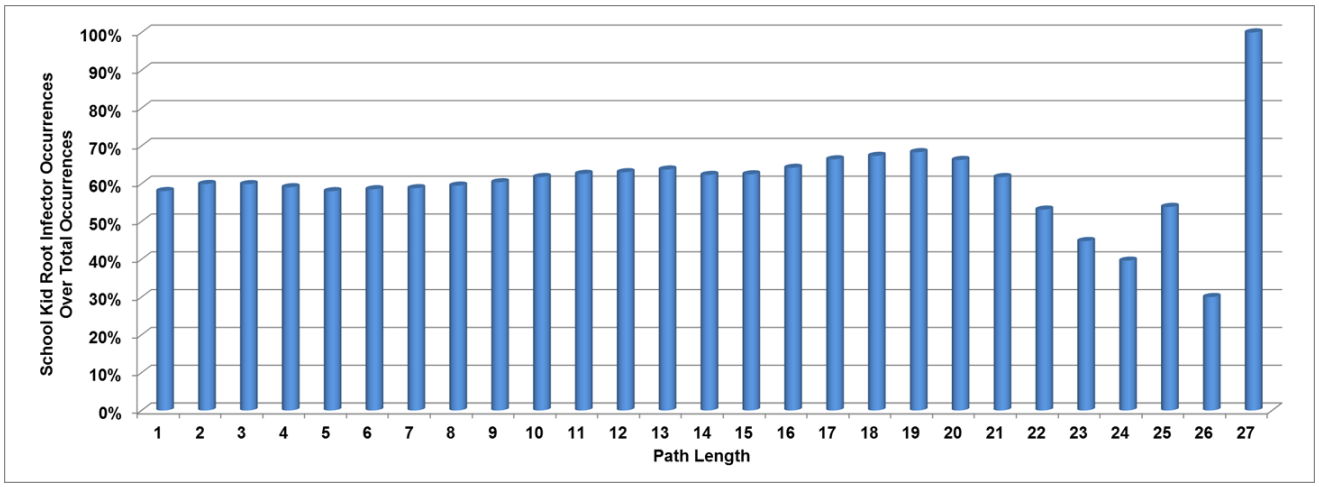


Figure 5.12: Percentage of the paths (over total) where root infector is a school child for various path lengths. We present path length in the x-axis and school child root infector paths percentage on the y-axis.

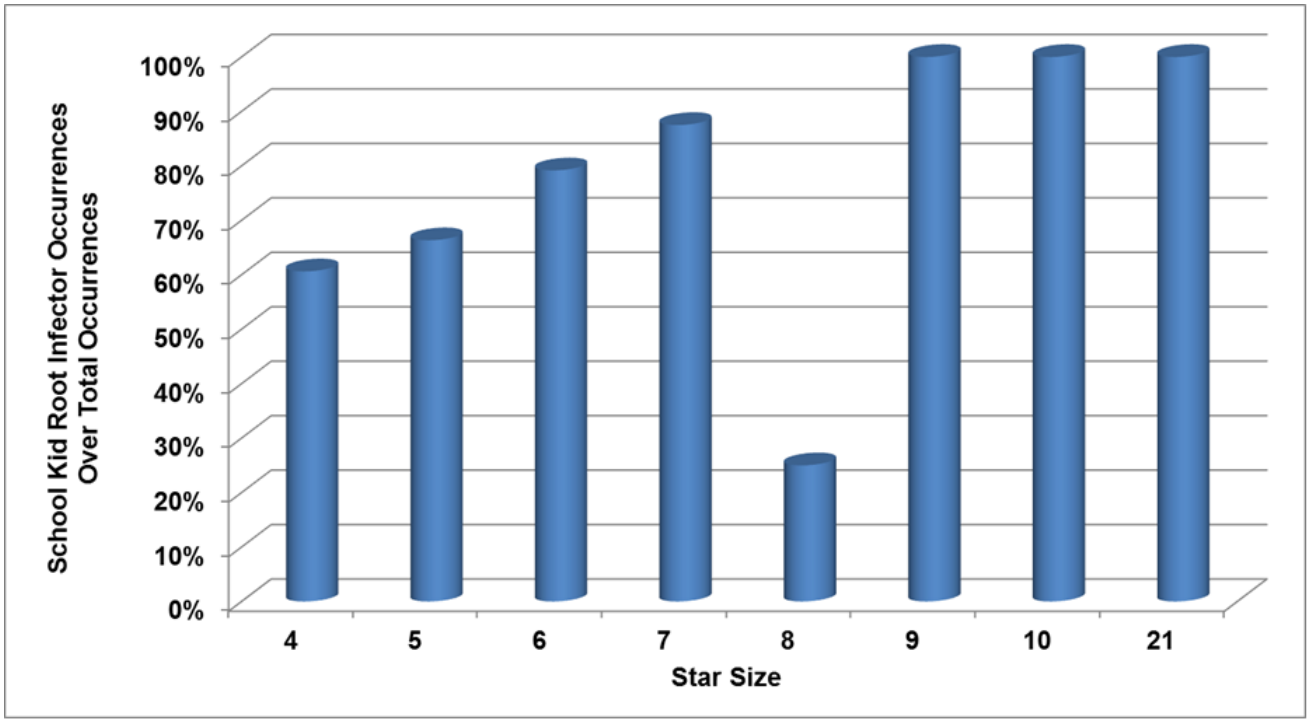


Figure 5.13: Percentage of star patterns (over total) where the root node is a school-infected child for various star sizes. We present star size in the x-axis and school child root infector stars percentage on the y-axis. In the experiment, we consider four nodes as a minimum length of a star pattern.

Table 5.24: Path count statistics. That includes maximum, minimum, mean, and standard deviation of path lengths for all replicates.

Path Count						
	Maximum		Minimum		Mean Path Length	Standard Deviation of Path Length
	Path Length	Number of Occurrences	Path Length	Number of Occurrences		
Replicate 0	26	3	1	1133	13.5	7.65
Replicate 1	24	11	1	1084	12.5	7.07
Replicate 2	13	1	1	228	7	3.89
Replicate 3	25	7	1	1180	13	7.36
Replicate 4	27	1	1	688	14	7.94
Replicate 5	25	6	1	988	13	7.36
Replicate 6	27	4	1	1219	14	7.94
Replicate 7	22	33	1	240	11.5	6.49
Replicate 8	23	2	1	159	12	6.78
Replicate 9	27	1	1	1719	14	7.94

Table 5.25: Star pattern count information. We provide the maximum, minimum, mean, and standard deviation of star sizes for all replicates.

Star Count						
	Maximum		Minimum		Mean Star Size	Standard Deviation of Star Size
	Star Size	Number of Occurrences	Star Size	Number of Occurrences		
Replicate 0	21	1	4	74	8.5	6.78
Replicate 1	21	1	4	77	8.6	7.02
Replicate 2	21	1	4	12	9	8.04
Replicate 3	21	1	4	77	8.71	5.77
Replicate 4	21	1	4	45	8.67	6.28
Replicate 5	21	1	4	67	8.5	6.28
Replicate 6	21	1	4	83	8.6	7.02
Replicate 7	21	1	4	11	9	8.04
Replicate 8	21	1	4	4	9.25	7.93
Replicate 9	21	1	4	96	8.71	5.77

Table 5.26: Summary of degree, star, and path counts.

	Maximum	Minimum	Mean	Standard Deviation
Degree	20	1	6.2	5.57
Path	27	1	12.78	7.34
Star	21	4	8.72	6.09

Policymakers can use this super-spreader information to design targeted interventions (e.g., vaccinations) to stop the disease propagation.

Example 3: Investigating the role of school children in disease transmission. As a final example, we study the role of school children in an epidemic. Intuitively, it is well accepted that school children play an important role in epidemics [146, 147]. EpiK can be used for a simulation-based analytical experiment to understand this. We look at dendrograms produced by simulations as discussed in the earlier experiments, examining two kinds of motifs. We look at paths of various lengths in which the starting node is a school aged child (age range: 3-18). We also look at stars of various sizes where the root of the star motif is a school aged child. Our results are summarized in Figures 5.12 and 5.13. In both the plots, for a given size S, the Y-axis plots the percentage of the fraction $\frac{A}{B}$ where A is the total number of paths (stars) of size S where the root is a school-aged child and B is the total number of paths (stars) of size S. The higher the fraction, the more prominent the role of a school aged child. The figures clearly show that school-aged children play an important role in epidemic spread. For example, by examining Figure 5.12 we see that for most of the path lengths, over 50% of the paths have a school-aged child as a root. The same conclusion is true for stars. Furthermore, for 100% of the large stars (size: 9, 10, 21) and path (size: 27), root nodes (acting as a super-spreader) are school-aged children.

The choice of paths and stars is important. Stars represent a particular school-aged child infecting many other individuals in a small time frame; a path represents a school-aged child being the originator of a sustained transmission chain. Each motif points to a different role played by a school-aged child. The high percentage of such motifs with school-aged children as root provides evidence that school children play a major role in disease transmission.

5.5 Query Performance Evaluation for Reported Data

We implement a prototype of our schema and linking approach that is mentioned in Chapter 4, using various datasets related to the Ebola epidemic 2014–2015. We provide detail about our experimentation below.

5.5.1 Experimental Setup

Datasets

We began by identifying datasets that provide information about the Ebola outbreak 2014–2015. We interviewed several epidemiologists and asked them ***what are the most useful datasets you need for analysis?*** Based on their input we identified 19 Ebola datasets for our prototype development. Dataset names, conceptual schema category, source format, number of records, contributor, and date of the dataset are available in Table 5.27. We provide dataset references in the tables as well. Dataset license information and metadata

information are accessible from the cited references. Please visit corresponding references for details. We constructed additional "Epidemic", "Country", "Admin1", and "Admin2" datasets from the "FSC: Matrix 4Ws for Activities MASTER" dataset.

Data Cleaning

We found a number of inconsistencies in the input files. For example, in Guinea there is a sub-location called Nzerekore. However, in some files, it is misspelled as Mzerekore. We corrected it with proper spelling. We also fixed several underscore issues. For example, Rivercess County of Liberia is represented as "River_Cess_County" in some files and "Rivercess_County" in others. Some value fields contain unnecessary commas. In the input files, null values presented in various ways (e.g., space, NA, NR) are all set to the database null value. In some datasets, country name is represented in abbreviated format. We converted it into a full name. In some datasets, administrative region names contain the word "county" along with the name (e.g., Montserrado County). We removed the word county and only keep the region name. We represented dates in the "month, day, and year" order. To uniquely identify file records we added a column in the file with the unique value for each row of data.

We have several types of datasets, and different datasets need different kinds of cleaning. Hence our data cleaning process is manual. It takes around 5-30 minutes to clean a dataset based on the number of records. After the cleaning, the datasets were loaded into a PostgreSQL database. The CSV file names became the table names. The column header in the CSV file was used as the column name in the table. We defined the column datatype by examining the raw data.

5.5.2 Development of Mapping and RDF Graph

Most of the reported datasets exist in a tabular format. Hence, we can quickly inject these datasets into relational database (RDB) tables. Various tools are available to convert RDB to RDF, e.g., D2RQ [119]. The conversion tools can create mapping files from database tables, known as default mapping. We present D2RQ default mapping file creation pseudocode in Algorithm 2. In Chapter 4, we described the literal node creation approach. In Algorithm 3, we provide pseudocode for the literal node creation in D2RQ. We use the mapping file with Algorithm 3 to create the RDF graph. We provide linking pseudocode in Algorithm 4, and also create hierarchy edges and transitivity edges by following definitions 4.2.3 and 4.2.4. We implement the linking program in Python 2.7.11 with RDFLib 4.2.1. The entire RDF graph creation process is available in Algorithm 1. We use the Shadowfax high-performance

Table 5.27: Sample Ebola reported datasets used in prototype development.

Dataset Name	Category	Format	Number of Records	Contributor	Date
Data for Ebola Recovery [148]	Publication	CSV, XLSX	1,573	MIT Governance Lab	December 04, 2014 - January 07, 2015
Direct Relief Ebola Materials Shipped [149]	Logistics	XLSX	897	Direct Relief	July 22, 2015 - December 31, 2015
Ebola Community Care Centers [150]	Medical Facilities	CSV, SHAPEFILE	119	UN Mission for Ebola Emergency Response (UNMEER)	December 29, 2014
Ebola Treatment Centers or Units (ETCs or ETUs) [151]	Medical Facilities	CSV, SHAPEFILE	64	UN Mission for Ebola Emergency Response (UNMEER)	December 30, 2014
Ebola outbreaks before 2014 [152]	Time Series	XLSX	34	The Humanitarian Data Exchange	December 31, 2013
Global Food Prices Database (WFP) [153]	Food	CSV, TXT, XLSX	474,760	World Food Programme (WFP)	August 31, 2016
Health Facilities Liberia oct 2014 [154]	Medical Facilities	XLSX	791	Standby Task Force	November 01, 2014
Human Outbreaks of Ebola Zaire, Ebola Sudan and Ebola Bundibugyo from 1976 [155]	Time Series	XLSX	24	Van Kerkhove, M., Bento, A. I., Mills, H. L., Ferguson, N. M. & Donnelly, C. A	May 26, 2015
Lat/Long/Names of ETUs in Liberia updated as of 21 Oct [156]	Medical Facilities	XLSX	31	United States Department of Defense	Not Available
Liberia ETU Constructions [157]	Medical Facilities	CSV, XLSX	30	United States Department of Defense	October 01, 2014 - October 09, 2014
Liberia FTS indicators [158]	Economics	CSV, TXT, XLSX	20	The Humanitarian Data Exchange	Not Available
Number of Ebola health-care worker deaths [159]	Time Series	CSV, XLSX	289	The Humanitarian Data Exchange	September 07, 2014 - November 04, 2015
FSC: Matrix 4Ws for Activities MASTER [?]	Food	XLSX	1488	Food Security Cluster	February 29, 2016 - December 31, 2016
Number of health-care workers infected with Ebola [160]	Time Series	CSV, XLSX	525	The Humanitarian Data Exchange	September 07, 2014 - November 04, 2015
Parameter estimates for the ongoing EVD outbreak in West Africa [161]	Disease	XLSX	87	Van Kerkhove, M., Bento, A. I., Mills, H. L., Ferguson, N. M. & Donnelly, C. A	May 26, 2015
Parameter Estimates by Outbreak [162]	Epidemic	CSV, XLSX	113	Van Kerkhove, M., Bento, A. I., Mills, H. L., Ferguson, N. M. & Donnelly, C. A	May 27, 2015
Safe and Dignified Burial Teams [163]	Medical Facilities	CSV, SHAPEFILE	61	World Health Organization (WHO)	November 28, 2014
Sub-national Indicators Ebola Countries[164]	Demographic	XLSX	240	Global Data Lab (GDL)	October 22, 2014
Time Series data on Ebola [165]	Time Series	XLSX	58,636	Regional Office for West and Central Africa (ROWCA), Office for the Coordination of Humanitarian Affairs (OCHA)	April 02, 2015

computing (HPC) resource available at the Biocomplexity Institute of Virginia Tech for RDF graph creation.

We create an RDF graph based on the datasets mentioned in Table 5.27, and then we need mapping files. We present default and literal node mapping creation information in Table 5.28. In Table 5.29 we present our RDF graph size, number of triples, graph generation time, link generation time, and Virtuoso loading time. The RDF graph is accessible through the following SPARQL endpoint: **http://taos.vbi.vt.edu:8890/sparql**. The Graph URI is **http://ndssl.bi.vt.edu/infectious_disease/**.

Note 11: We use Python 2.7.11 with RDFLib 4.2.1 to implement our linking Algorithm 4. We have found that the performance of RDFLib is very slow for large datasets. Hence, we use selected datasets for the link creation program. The datasets are Ebola Community Care Centers, Parameter Estimates by Outbreak, Ebola Treatment Centers or Units (ETCs or ETUs), Liberia ETU Constructions, Liberia FTS indicators, Lat/Long/Names of ETUs in Liberia updated as of 21 Oct, Number of health-care workers infected with Ebola, Number of Ebola health-care worker deaths, Ebola outbreaks before 2014, Safe and Dignified Burial Teams, Sub-national Indicators Ebola Countries, epidemic, Human Outbreaks of Ebola Zaire, Ebola Sudan and Ebola Bundibugyo from 1976, and Parameter estimates for the ongoing EVD outbreak in West Africa.

Algorithm 1 Pseudo-code for D2RQ mapping file creation.

INPUT: database user name u, database password p, database driver d, database table list T, D2RQ mapping file name with format n, Database JDBC connection URL c.
OUTPUT: D2RQ default mapping file map, D2RQ linked node mapping file $lmap$, and RDF graph G.

1: $map \leftarrow$ D2RQ mapping file generation program $generate\text{–}mapping\ (u, p, d, T, n, c)$
2: $lmap \leftarrow literal\text{–}node\text{–}mapping\ (map)$
3: $G \leftarrow$ D2RQ tool $dump\text{–}rdf\ (lmap)$

Table 5.28: Default and literal node mapping files creation times, their sizes, and triple counts.

Mapping	Size (KB)	Number of Triples	Creation Time (min.)
Default	256	4775	<1
Literal Node	576	10517	<1

Algorithm 2 $generate–mapping$ (u, p, d, T, n, c):
Pseudo-code for D2RQ default mapping file creation.

1: create an RDFS class $Database$.
2: create $Database$ class properties $username$, $password$, $jdbcDriver$, $jdbcDSN$ based on value of u, p, d, and c.
3: **for** each table t in T **do**
4: create an RDFS class called $ClassMap$.
5: create an RDF property called $uriPattern$ from table's primary key value that uniquely identifies instances of the $ClassMap$.
6: create an RDF property called $class$ to use establish vocabulary term with prefix $vocab$
7: create an RDF property called $classDefinitionLabel$.
8: **for** each column col of the corresponding table t **do**
9: create an RDFS class $PropertyBridge$ that maps RDF property to one or more database columns.
10: create an RDF property called $belongsToClassMap$ that indicates a property belongs to which $ClassMap$.
11: create an RDF property called $property$ by using standard or domain specific vocabulary terms with prefix $vocab$
12: create an RDF property called $propertyDefinitionLabel$ that provides a property label
13: create an RDF property called $column$ that indicates the table column where property is linked.
14: create an RDF property called $datatype$ to indicate literal value data types.
15: **end for**
16: **end for**

Table 5.29: RDF graph information created using our approach. Here we present RDF graph sizes, triple counts, RDF graph generation time, link generation time, and Virtuoso triplestores loading times.

Number of Triples	26313381
RDF Graph Size (GB)	5
RDF Graph Generation Time (min.)	4.6
Link Generation Time (min)	59.29
Virtuoso Loading Time (min.)	16

Table 5.30: Example queries we collected by interviewing various epidemiologists. We present queries in plain English, their 5W1H question category, and datasets needed to answer the queries.

Query Number	English Query	5W1H Question	Data Needed to Answer the Query
Q1	Who supplied relief material at Zwedru, Liberia?	WHO	Epidemic, Direct Relief Ebola Materials Shipped
Q2	Who reported Ebola parameter information?	WHO	Epidemic, Parameter Estimates by Outbreak
Q3	Who constructed Ebola Treatment Units (ETUs) in Macenta, Guinea?	WHO	Epidemic, Country, Admin1, Admin2, Ebola Treatment Centers or Units (ETCs or ETUs)
Q4	What were the Ebola care facility names (ECF) at Sierra Leone?	WHAT	Epidemic, Country, Ebola Community Care Centers
Q5	What were the sources of the time series data?	WHAT	Epidemic, Time Series
Q6	What were the categories of infection?	WHAT	Epidemic, Time Series
Q7	Where did the past Ebola outbreaks occur?	WHERE	Epidemic, Ebola outbreaks before 2014
Q8	Where were the burial teams in Sierra Leone?	WHERE	Epidemic, Safe and Dignified Burial Teams
Q9	Why did the number of new cases drop in Liberia after December 2, 2014?	WHY, HOW	Epidemic, Country, Ebola Treatment Centers or Units (ETCs or ETUs), Time Series,
Q10	Why were many ETUs constructed in Montserrado, Liberia?	WHY, HOW	Epidemic, Country, Admin1, Time Series
Q11	How many ETUs had a lab onsite in Sierra Leone?	HOW	Epidemic, Ebola Treatment Centers or Units (ETCs or ETUs)
Q12	How many individuals responded in Senegal?	HOW	Epidemic, Country Sub-national IndicatorsEbola Countries
Q13	How many datasets about Liberia are available?	HOW	Epidemic, Country, Admin1, Direct Relief Ebola Materials Shipped, Global Food Prices Database (WFP), Health Facilities Liberia oct 2014, Number of Ebola health-care worker deaths, Number of health-care workers infected with Ebola, Safe and Dignified Burial Teams, Sub-national Indicators Ebola Countries, Time Series
Q14	How to create a line list related to Montserrado county of Liberia.	HOW	Epidemic, Country, Admin1, Admin2, Ebola Treatment Centers or Units (ETCs or ETUs), Global Food Prices Database (WFP), Health Facilities Liberia oct 2014, Liberia ETU Constructions, Safe and Dignified Burial Teams, Sub-national Indicators Ebola Countries, Time Series
Q15	How many healthcare facilities were opened in Liberia, and where were they?	HOW, WHERE	Epidemic, Country, Health FacilitiesLiberia oct 2014
Q16	Where were the community care centers in Liberia, and how many have no beds in use?	HOW, WHERE	Epidemic, Country, Ebola Community Care Centers
Q17	How many dates have no case count information available and tell me their country and county names?	HOW, WHERE	Epidemic, Country, Admin1, Time Series
Q18	Which ETU at Liberia was operational first, where is it, and how many new cases were reported in that location three months before and after the ETU opening date?	WHEN, WHERE, HOW	Epidemic, Country, Ebola Treatment Centers or Units (ETCs or ETUs), Time Series
Q19	How many Ebola cases occurred in various counties of Liberia, how many households are available in those counties, where were most relief items sent, when and what relief items were sent?	WHEN, WHERE, HOW	Epidemic, Country, Admin1, Time Series, Sub-national Indicators Ebola Countries, Direct Relief Ebola Materials Shipped
Q20	How many Ebola deaths occurred in various counties of Liberia, were there any healthcare facilities available in that location, what type of facilities were they, when were they opened, how many doctors were available, how many healthcare workers were infected with Ebola, and how many died?	WHEN, WHERE, HOW	Epidemic, Country, Admin1, Health Facilities, Time Series, Liberia oct 2014, Number of health-careworkers infected with Ebola, Number of Ebola health-care worker deaths

Algorithm 3 $literal–node–mapping$ (map):
Pseudo-code for D2RQ literal node creation mapping file.

```
for each line l in map do
    if l contains prefix vocab then
        clean it by removing underscore, capitalizing first letter of each word, concatenating
        words, and adding has before concatenated word.
    end if
    if l contains column then
        create a new ClassMap with Database and uriPattern mentioned in Algorithm 2
        create a new PropertyBridge with belongsToClassMap and vocabulary
        hasValue
        add column under new PropertyBridge.
        remove l and add a line that uses the RDF property refersToClassMap to refer to
        new ClassMap
    end if
end for
```

Algorithm 4 Pseudo-code for link node creation.

INPUT: RDF graph G
OUTPUT: D2RQ linked node mapping file $LINKMAP$.

```
G contains RDF triples with subject S, predicate P, and object O
s1, p1 o1 are one triple, where s1 ∈ S, p1 ∈ P, and o1 ∈ O
s2, p2 o2 are another triple, where s2 ∈ S, p2 ∈ P, and o2 ∈ O
for all triples T in G do
    if s1 ≠ s2, p1 = p2 = hasValue, o1 = o2 then
        create a triple with subject p1, predicate rdfs:SeeAlso, and object link node lv
        create triple with subject p2, predicate rdfs:SeeAlso, and object link node lv.
    end if
end for
```

5.5.3 Example Query

We interviewed epidemiologists and based on their requirements prepared a query list. We present the query list in Table 5.30. We implemented all the queries listed in Table 5.30 in the SPARQL language.

We employ a **5W1H** interrogative approach to classify the queries [133, 134, 135]. In Table 5.31, we present a mapping between 5W1H questions and RDF property vocabularies. The table shows that it is possible to construct a large corpus of queries using the 5W1H

approach. The first column contains 5W1H question types, the second column provides question types description, and finally the last column shows an example RDF data property vocabulary list. This list can be used in the predicate part of the RDF triples. The wildcard (*) in the vocabulary means syntax variation is possible, for example, "hasSource", "hasSupplier", etc. We provide an example SPARQL query in Table 5.32.

Table 5.31: 5W1H questions can be mapped to the RDF data property vocabularies. A partial sample of the 5W1H questions' vocabulary domain is shown for the brevity of space. Here, an asterisk symbol represents the wildcard (a character or sequence of characters).

5W1H Question	**Definition**	**Example Property Vocabulary**
WHO	property used to refer people	*Supplier, *Partner, ...
WHAT	property refers to specific information	*Source, *Category,.....
WHERE	property refers to place or location	*County, *Latitude, *Longitude, ...
WHEN	the property refers to time or occasion	*OpenDate, *CloseDate, ...
WHY	the property use to refer reason or cause	*Purpose, ...
HOW	property refers the manner that something is done	Queries that need data arrangement and SPARQL aggregate function like COUNT, SUM. etc.

Table 5.32: SPARQL implementation of a sample 5W1H query. Here, we are showing a WHAT query. The vocabulary "hasSource" fits the WHAT type property vocabulary (*Source).

Query: What were the sources of the Ebola time series data? **Sample SPARQL Implementation:**

SELECT distinct ?timeSeriesSources WHERE {
?timeSeries rdf:type :ebola_time_series.
?timeSeries :hasSources ?timeSeriesSources.
}

Query Results

In the following, we discuss query results and their performance.

WHO Queries:

We present WHO query (Q1, Q2, and Q3) results in Tables 5.33, 5.34, and 5.35. We present partial results for Q1 and Q2 for the sake of brevity . We show running time of the WHO queries (Q1, Q2, and Q3) in Figure 5.14, which illustrates that Q1 takes more time than Q2 and Q3 because the ReliefMaterial class used in Q1 has more instances.

Table 5.33: Q1 Results (partial): This shows relief material supplier at Zwedru, Liberia.

Relief Material Supplier
Teva Pharmaceuticals
Actavis Pharma, Incs
Mylan Laboratories Inc
Omron Healthcare, Inc
Johnson & Johnson Consumer Companies
.....

Table 5.34: Q2 Results (partial): Ebola parameter information reporter.

Ebola Parameter Reporters
International Commission, 1978
Muyembe & Kipasa, 1995
Lekone & Finkenst, 2006
WHO Ebola Response Team, 2014
Nishiura & Chowell, 2014
.....

Table 5.35: Q3 Results: ETU constructor in Macenta, Guinea.

ETU Constructor in Macenta, Guinea
French Red Cross

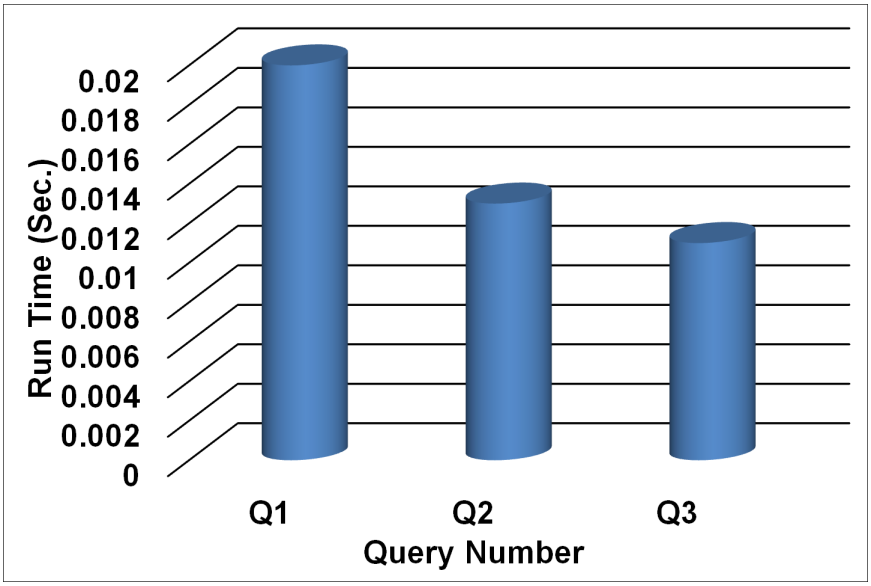


Figure 5.14: WHO queries (Q1, Q2, Q3) performance.

WHAT Queries:

We show WHAT query (Q4, Q5, and Q6) results in Tables 5.36, 5.37, and 5.38. We present partial results for queries Q4 and Q5 because of limited space. We show running time of the WHAT queries (Q4, Q5, and Q6) in Figure 5.15, which shows that queries Q5 and Q6 take more time than Q1 because the TimeSeries class used in Q5 and Q6 has more instances.

Table 5.36: Q4 Results: Ebola care facility names (ECF) at Sierra Leone.

Ebola care facility names (ECF) at Sierra Leone
Port Loko Gov. Hospital EHC
Lokomasama (Petifu Junction)
Gbendembu
Kenema Gov. Hospital
Catholic Hospital (Mabesseneh)
.....

Table 5.37: Q5 Results: Time series data sources.

Sources
WHO
UNICEF
ECHO
Ministere de la Sante
GVT
.....

Table 5.38: Q6 Results: Ebola categories of infection.

Categories of Infection
Suspected cases
Confirmed cases
Deaths
Probable cases
New cases
Cases
Suspected Cases
New Cases

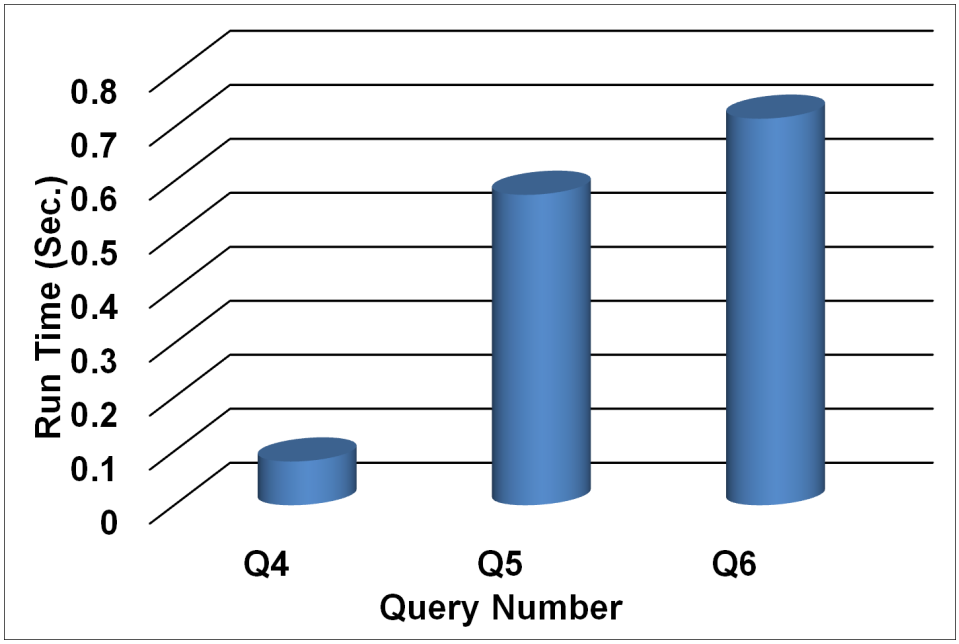


Figure 5.15: WHAT queries (Q4, Q5, Q6) performance.

WHERE Queries:

We present WHERE query (Q7 and Q8) results in Tables 5.39 and 5.40, with partial results displayed for queries Q7 and Q8 because of limited space. We show running time of the WHERE queries (Q7 and Q8) in Figure 5.16, which shows that query Q8 takes more time than Q7 because the BurialTeam class used in Q8 has more instances.

Table 5.39: Q7 Results (partial): Country names, where Ebola outbreak occurred before 2014.

Ebola Outbreak Before 2014 Country
Democratic Republic of the Congo
Gabon
Sudan
South Africa
Uganda
.....

Table 5.40: Q8 Results (partial): Burial team location in Sierra Leone. Admin1 means first-level administrative country subdivisions and Admin2 means second-level administrative country subdivisions.

Admin1	*Admin2*	*Burial Team*
Northern	Kambia	4
Eastern	Kailahun	9
Eastern	Kenema	5
Western Area	Freetown	20
Southern	Bonthe	9
.....		

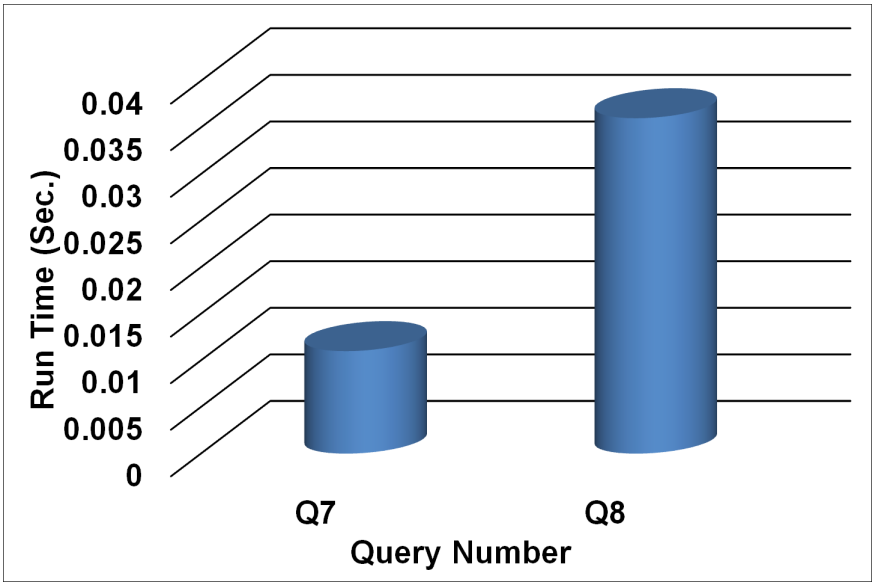


Figure 5.16: WHERE queries (Q7, Q8) performance.

WHY AND HOW Queries:

We show a combination of WHY and HOW queries (Q9 and Q10) results in Tables 5.41 and 5.42. Table 5.41 shows that an ETU was constructed in Liberia on December 2, 2014. The table represents new Ebola cases reported before and after the ETU construction date, and that the latter number is reduced after the ETU construction. That answers the query: why the number of new cases dropped in Liberia after December 2, 2014. Table 5.42 presents the top three counties in Liberia regarding the total number of reported cases, showing that Montserrado County is heavily infected with Ebola disease. Hence, it justifies the many ETUs constructed in Montserrado, Liberia. In queries Q9 and Q10, besides WHERE properties, we use SPARQL SUM, GROUP BY, and ORDER BY operations. Hence, queries Q9 and Q10 fall in both WHY and HOW categories. We show the query execution performance in Figure 5.17. Query Q9 aggregates time series data twice (before and after ETU opening date) and therefore it takes longer than query Q10.

Table 5.41: Q9 Results: ETU constructed in Liberia on December 2, 2014. The number of new Ebola cases before and after the ETU construction date is displayed, showing a drop in new cases after the ETU construction date.

New Cases Before December 12, 2014	*New Cases After December 12, 2014*	*ETU Open Date*
7791	2512	12/02/2014

Table 5.42: Q10 Results: Counties in Liberia with the highest reported number of cases.

Liberia Counties	*Total Cases*
Montserrado	445077
Margibi	135229
Lofa	96338

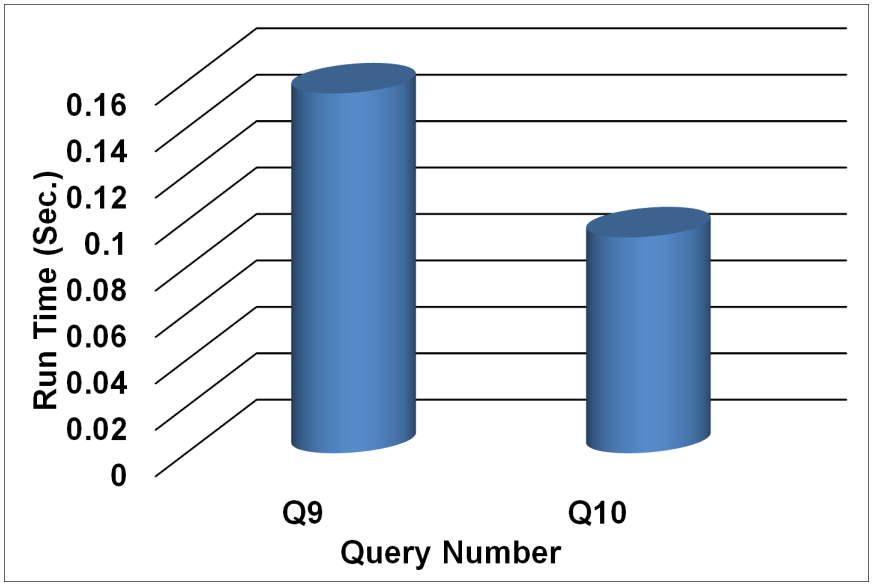


Figure 5.17: WHY and HOW queries (Q9, Q10) performance.

HOW Queries:

We show HOW queries (Q11, Q12, Q13, and Q14) results in Tables 5.43, 5.44, 5.45, and 5.46. We present partial results for query Q14 because of limited space. We show running time of the HOW queries (Q11, Q12, Q13, and Q14) in Figure 5.18, which shows queries Q13 and Q14 take longer because they need to traverse more triples (dataset names mentioned in Table 5.30) to fetch the results.

Table 5.43: Q11 Result: Number of ETUs that had an onsite lab in Sierra Leone.

ETUs had onsite lab in Sierra Leone
11

Table 5.44: Q12 Result: Number of individuals who responded in Senegal.

Individuals Responding in Senegal
76906

Table 5.45: Q13 Result: Number of distinct datasets available in the RDF graph related to the country Liberia.

Datasets about Liberia
12

Table 5.46: Q14 Results (partial): Possible line list representation regarding Montserrado County of Liberia.

Dataset Name	*Instance*	*Attribute*	*Value*
Health Facilities Liberia oct 2014	458	center	Fiamah Community Clinic
Health Facilities Liberia oct 2014	458	district	Greater Monrovia
Health Facilities Liberia oct 2014	458	org	Private
Health Facilities Liberia oct 2014	458	source of info	List from: Medical Teams International
.....			

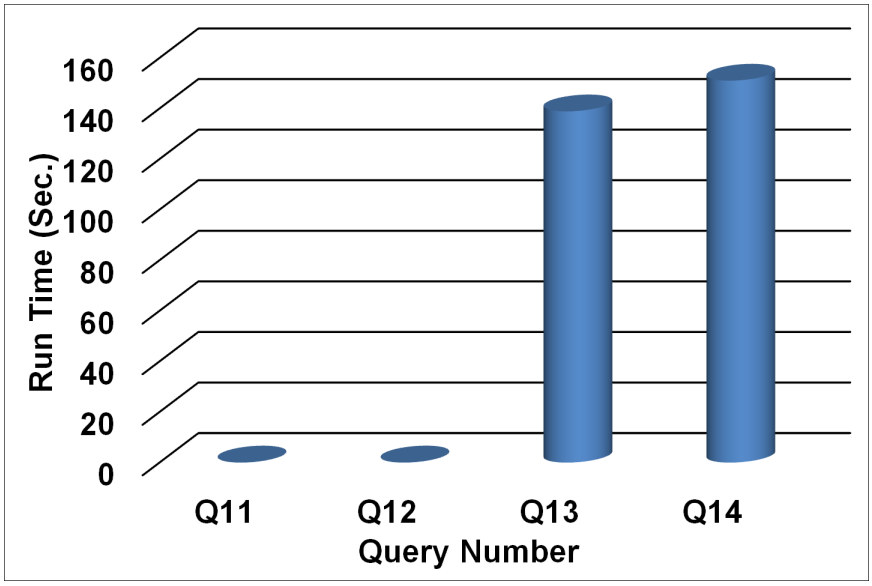


Figure 5.18: HOW queries (Q11, Q12, Q13, and Q14) performance. Queries Q13 and Q14 need more datasets to answer and therefore take longer.

HOW and WHERE Queries:

We present a combination of HOW and WHERE query (Q15, Q16, and Q17) results in Tables 5.47, 5.48, and 5.49. We present partial results because of a lack of space. Table 5.47 presents the number of healthcare facilities opened in Liberia and their location information. Table 5.48 displays community care centers locations in Liberia, where no beds were in use. Table 5.49 shows the total number of dates with no case count information and their locations. If the data value is missing in the epidemic peak, then it raises a question about

the accuracy of the time series dataset. Time series is an important component of the epidemic datasets. Hence, results in Tables 5.49 can validate the accuracy of the time series dataset. We show running time of the combination of HOW and WHERE queries (Q15, Q16, and Q17) in Figure 5.19, illustrating that query Q17 takes more time than Q15 and Q16 because the TimeSeries class used in query Q17 has more instances.

Table 5.47: Q15 Results (partial): Number of healthcare facilities and their locations in Liberia. Here Admin1 means first-level administrative country subdivisions.

Admin1	***Healthcare Facilities***
Bomi	34
Bong	42
Lofa	82
Nimba	70
.....	

Table 5.48: Q16 Results (partial): Community care centers in Liberia with no beds in use.

Community care centers in Liberia	***No beds in use***
Gbarpolu	4
Grand Bassa	2
Montserrado	5
Nimba	2
Sinoe	2
.....	

Table 5.49: Q17 Results (partial): Time series dataset of the total number of dates with no case count information and their location.

Country	***Admin1***	***Total Dates (no case count)***
Guinea	Koundara	13
Guinea	Mamou	108
Guinea	Labe	47
Liberia	Grand Kru	3
Liberia	Grand Gedeh	1
.....		

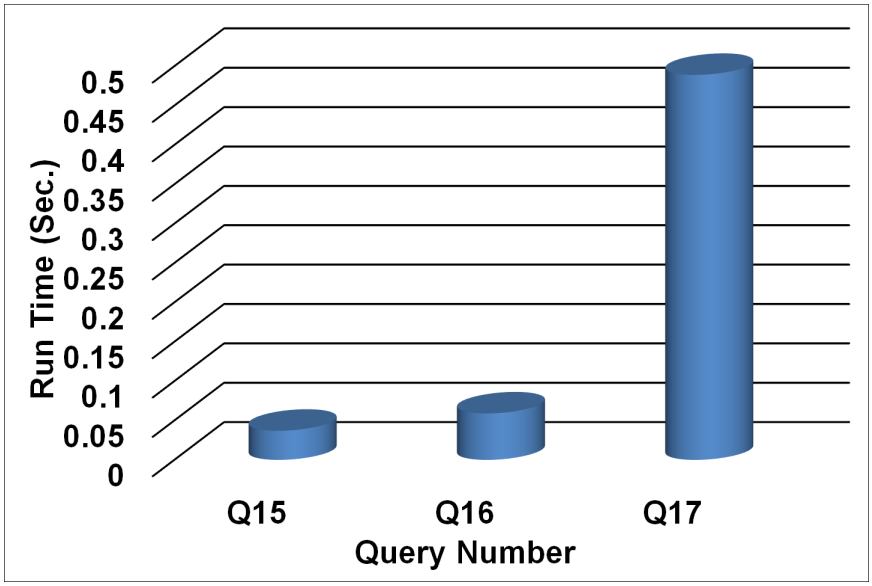


Figure 5.19: HOW and WHERE queries (Q15, Q16, and Q17) performances.

WHEN, WHERE, and HOW Queries:

We show the combination of WHEN, WHERE, and HOW query (Q18, Q19, and Q20) results in Tables 5.50, 5.51, and 5.52, with partial results shown for queries Q19 and Q20 because of limited space. Tables 5.50 shows the first operational ETU in Liberia, opening date, county (Admin1), and the total number of new cases three months before and after the ETU opening date. Table 5.51 presents Ebola cases in various counties in Liberia, the number of households in those counties, the location with the most relief materials sent to Liberia, relief material descriptions, and shipment date. Table 5.52 displays Ebola deaths in various counties in Liberia, availability of the healthcare facility on that location, type of facility, facility open date, number of doctors available in those facilities, Ebola infected healthcare worker total, and Ebola infected healthcare worker death total. We show running times of the combination of the WHEN, WHERE, and HOW queries (Q18, Q19, and Q20) in Figure 5.20, which indicates that query Q20 took longer than others, because query Q20 uses multiple SPARQL subqueries, the aggregation function, and different dataset joins.

Table 5.50: Q18 Results: First operational ETU in Liberia, ETU location, opening date, and the total of new cases three months before and after the ETU opening date. Here Admin1 means first-level administrative country subdivisions.

ETU	*Date*	*Admin1*	*Total New Cases (three months before ETU opening date)*	*Total New Cases (three months after ETU opening date)*
ELWA 2	12/02/2014	Montserrado	1799	745

Table 5.51: Q19 Results (partial): Ebola cases in various counties of Liberia, the location where the most relief items were sent, relief item descriptions, and shipment dates.

County (Admin1)	*Case Total*	*Number of Household*	*Relief Location*	*Relief Materials Shipped Date*	*Relief Materials Shipped Description*
Bomi	29327	288	Montserrado	09/15/2014	Cover_shoe_medium_100paa
				09/16/2014	Mask_procedure_50ct
				09/25/2014	Hydrocortisone-Aceticacid_1%-2%_10ml_sol
				12/03/2014	Sheet_bed_assorted_cv
				12/05/2014	Set_extension_6"_1.14ml_strle_25ct
.....					

Table 5.52: Query 20 Results (partial): Ebola deaths in various counties of Liberia, availability of healthcare facility at that location, type of the facility, facility open date, the number of doctors available, number of healthcare workers infected with Ebola, and the number of healthcare workers who died of the Ebola disease.

County (Admin1)	*Death Total*	*Health Care Facility Available*	*Health Care Facility Type*	*Facility Open Date*	*Number of Doctors*	*Ebola Infected Healthcare WorkerTotal*	*Ebola Infected HealthCare Worker Death Total*
Montserrado	195492	Yes	Hospital	08/19/2014	6	21671	10297
Grand Cape Mount	9305	Yes	Clinic	08/19/2014	1		
Maryland	3135	Yes	Clinic	08/11/2014	1		
Grand Bassa	7485	Yes	Hospital	08/19/2014	2		
Bomi	16301	Yes	Clinic	08/19/2014	1		
.....							

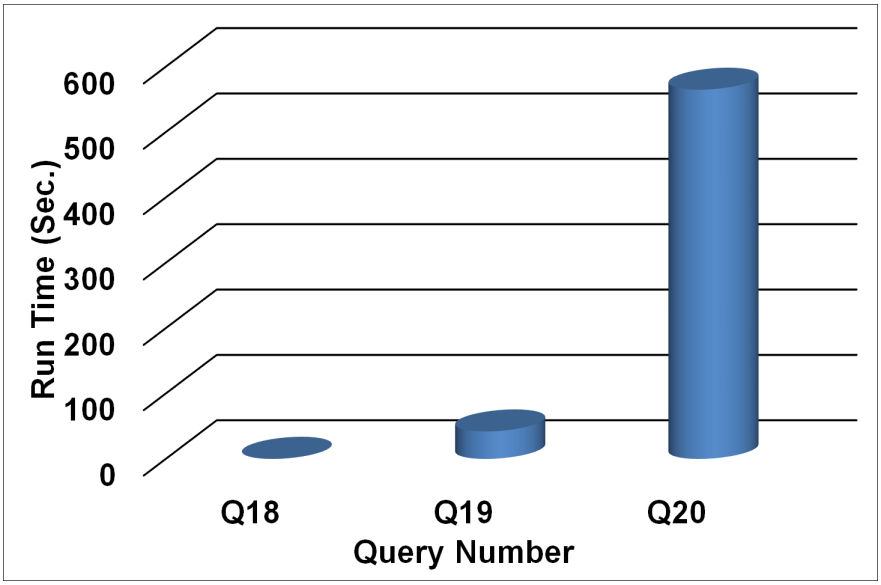


Figure 5.20: WHEN, WHERE, HOW queries (Q18, Q19, and Q20) performance.

5.6 Application Programming Interface (API)

We propose to implement a number of REST APIs to facilitate schema construction and querying, by using Python 2.7.0, and RDFLib 4.2.1.

5.6.1 Schema Construction APIs

Schema construction APIs will create schema components, instances, and links. Target users of these APIs are those with knowledge of CNEW and computational epidemiology reported datasets, and those responsible for maintaining and operating these datasets such as a database administrator (DBA). In the following, we present some example API descriptions.

- **addClass (className):** Allows a user to create a class, taking the class name as an input parameter.

Listing 5.1: *addClass (className)* example API call.

```
API:
---
addClass (className)

INPUT:
------
className =  Class name.

In this API 'className' is a mandatory parameters.

Example:
className='PersonWithinAdminRegion'


OUTPUT:
------
<http://www.ndssl.bi.vt.edu/fuse#PersonWithinAdminRegion>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#Class> .
```

Preliminaries:
We can represent RDF triples in graph format by considering subjects and objects as nodes and predicates as edges. An RDF graph $G = (V, E)$, where V is the set of nodes, and E is the set of directed edges such that $E = \{(u, v) : u \in V, v \in V, \text{and } u \neq v\}$. Let $l(u)$ be a label of node u.

Formal Definition:
Let class node c, and OWL class node oc, be elements of V. The labels of c and oc are respectively $l(c)$ and $l(oc)$ ="owl:Class." We assume node oc already exists in G.

addClass is an operation that takes a class name $l(c)$ from the user. This operation adds a node c with label $l(c)$ in G, i.e., $V = V \cup \{c\}$. It also adds an edge (c, oc), i.e., $E = E \cup \{(c, oc)\}$. Label of the edge (c, oc) is "rdf:type".

- **addClassProperty (className, propertyList, primaryKeyPropertyName):** Allows creating a class property and defining a primary key property.

Listing 5.2: *addClassProperty (className, propertyList, primaryKeyPropertyName)* example API call.

```
API:
---
addClassProperty (className, propertyList, primaryKeyPropertyName)

INPUT:
-----
className = Name of the class.
propertyList = List of the class property.
primaryKeyPropertyName = Name of the primary key property.


Example:
className='PersonWithinAdminRegion'
attributeList=['hasPersonID','hasAge' , 'hasGender']
primaryKeyPropertyName = 'hasPersonID'

In this API all of the parameters are mandatory.

OUTPUT:
------
<http://www.ndssl.bi.vt.edu/fuse#hasPersonID>
<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>
<http://www.ndssl.bi.vt.edu/fuse#hasDataPropertyAttribute> .

<http://www.ndssl.bi.vt.edu/fuse#hasPersonID>
<http://www.w3.org/2000/01/rdf-schema#domain>
<http://www.ndssl.bi.vt.edu/fuse#PersonWithinAdminRegion> .

<http://www.ndssl.bi.vt.edu/fuse#hasPersonID>
<http://www.w3.org/2000/01/rdf-schema#range>
<http://www.w3.org/2000/01/rdf-schema#Literal> .
.........................................
<http://www.ndssl.bi.vt.edu/fuse#hasPersonID>
<http://www.w3.org/2000/01/rdf-schema#range>
<http://www.w3.org/2000/01/rdf-schema#hasPrimaryKeyAttribute> .
```

Formal Definition:

Let data property attribute node da, literal node ln, primary key node pk, and primary key attribute node hpk be the elements of V. Labels of da, ln, hpk are respectively "fused:hasDataPropertyAttribute", "rdfs:Literal", and "fused:hasPrimaryKeyAttribute". Let $P = \{p_1, p_2, ..., p_n\}$ be a list of property nodes and $l(p_i)$ be the label of p_i. Primary key node $pk \in P$ and the label of pk is $l(pk)$. Let class node $c \in V$ and the label of c be $l(c)$. Assume that nodes da, c, ln, and hpk

already exist in G.

addClassProperty is a operation that takes $l(c)$, $l(p_i)$, and $l(pk)$. The operation adds a list of new nodes P with labels $l(p_i)$, i.e., $V = V \cup \{P\}$. It also adds the following new edges: (p_i, da) such that $E = E \cup \{(p_i, da)\}$ and the label of (p_i, da) is "rdfs:subPropertyOf", (p_i, c) such that $E = E \bigcup \{(p_i, c)\}$ and the label of (p_i, c) is "rdfs:domain", (p_i, ln) such that $E = E \cup \{(p_i, ln)\}$ and the label of (p_i, ln) is "rdfs:range", and (pk, hpk) such that $E = E \cup \{(pk, hpk)\}$ and label of (pk, hpk) is "rdfs:subPropertyOf".

- **addClassInstance (className, classInstanceName):** Allows a user to create an instance of a class, taking the class name and class instance name as input parameters. Both of the parameters are required for this API.

Listing 5.3: *addClassInstance (className, classInstanceName)* example API call.

```
API:
---
addClassInstance (className, classInstanceName)

INPUT:
------
className =  Class name.
classInstanceName= Class instance name.

In this API both of the parameters are mandatory.

Example:
className='PersonWithinAdminRegion'
classInstanceName=
    'ChicagoPersonWithinAdminRegion'

OUTPUT:
------

<http://www.ndssl.bi.vt.edu/fuse#ChicagoPersonWithinAdminRegion>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#NamedIndividual> .
<http://www.ndssl.bi.vt.edu/fuse#ChicagoPersonWithinAdminRegion>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.ndssl.bi.vt.edu/fuse#PersonWithinAdminRegion> .
```

Formal Definition:
Let class node c, class instance node ci, and OWL named individual node oni be the elements of V. We consider $l(c)$ as label of c, $l(ci)$ as label of ci, and the label of oni is $l(oni)$ ="owl:NamedIndividual". Assume that c and oni already exist in G.

addClassInstance is an operation that takes $l(c)$ and $l(ci)$ as inputs. The operation adds a new node ci with label $l(ci)$ in G, i.e., $V = V \cup \{ci\}$. It adds a new edge (ci, oni), i.e., $E = E \cup \{(ci, oni)\}$. The label of the edge (ci, oni) is "rdf:type". The operation

also adds a new edge (ci, c), i.e., $E = E \cup E\{(ci, c)\}$ where node c's label is $l(c)$. The label of (ci, c) is "rdf:type."

- **addForeignKeyProperty (foreignClassName, foreignKeyPropertyName, sourceClassName, primaryKeyPropertyName):**

Listing 5.4: *addForeignKeyProperty (foreignClassName, foreignKeyPropertyName, sourceClassName, primaryKeyPropertyName)* example API call.

```
API:
---
addForeignKeyProperty (foreignClassName, foreignKeyPropertyName,
sourceClassName, primaryKeyPropertyName)

INPUT:
-----
foreignClassName = Foreign class name of the foreign key property.
foreignKeyPropertyName = Name of the foreign key property.
sourceClassName = Parent class name of the foreign key property.
primaryKeyPropertyName = Name of the primary key property
of the source class that maps foreign key property.

In this API all of the parameters are mandatory.

Example:
foreignClassName = 'ActivityWithinAdminRegion'
foreignKeyPropertyName = 'hasPersonNumber'
sourceClassName = 'PersonWithinAdminRegion'
primaryKeyPropertyName = 'hasPersonID'

OUTPUT:
------
<http://www.ndssl.bi.vt.edu/fuse#ActivityWithinAdminRegion>
<http://www.ndssl.bi.vt.edu/fuse#linksviaPrimaryKeyForeignKey>
<http://www.ndssl.bi.vt.edu/fuse#PersonWithinAdminRegion> .

<http://www.ndssl.bi.vt.edu/fuse#ActivityWithinAdminRegion>
<http://www.ndssl.bi.vt.edu/fuse#linkForeignKey>
<http://www.ndssl.bi.vt.edu/fuse#hasPersonNumber> .

<http://www.ndssl.bi.vt.edu/fuse#PersonWithinAdminRegion>
<http://www.ndssl.bi.vt.edu/fuse#linkPrimaryKey>
<http://www.ndssl.bi.vt.edu/fuse#hasPersonID> .
```

Formal Definition:

Let foreign class node fc, foreign key property node fk, source class node sc, and primary key property node pk be elements of V. We consider $l(fc)$ as label of fc, $l(fk)$ as label of fk, $l(sc)$ as label of sc, and $l(pk)$ as label of pk. Assume fc, fk, sc, and pk exist in G.

addForeignKeyProperty is an operation that takes $l(fc)$, $l(fk)$, $l(sc)$, and $l(pk)$ as inputs. The operation create new edges (fc, sc), i.e., $E = E \cup \{(fc, sc)\}$ where node fc's label is $l(fc)$ and node sc's label is $l(sc)$. The label of the edge (fc, sc)

is "fused:linksviaPrimaryKeyForeignKey". It creates a new edge (fc, fk), i.e., $E = E \cup \{(fc, fk)\}$ and node fk's label is $l(fk)$. The label of the edge (fc, fk) is "fused:linkForeignKey". This operation also adds a new edge (sc, pk), i.e., $E = E \cup \{(sc, pk)\}$ and node pk's label is $l(pk)$. The label of the edge (sc, pk) is "fused:linkPrimaryKey."

- **addSubclass (className, subClassName):** Allows creation of a subclass of a class.

Listing 5.5: *addSubclass (className, subClassName)* example API call.

```
API:
---
addSubclass(className, subClassName)

INPUT:
-----
className = Name of the class.
subClassName = Name of the subclass.

Example:
className = 'DataSource'
subClassName = 'OracleDataSource'

OUTPUT:
-------
<http://www.ndssl.bi.vt.edu/fuse#OracleDataSource>
<http://www.w3.org/2000/01/rdf-schema#subClassOf>
<http://www.ndssl.bi.vt.edu/fuse#DataSource> .
```

Formal Definition:

Let class node c and subclass node suc be elements of V. Here $l(c)$ and $l(suc)$ are labels of c and suc. Assume c already exists in G.

addSubclass is an operation that takes $l(c)$ and $l(suc)$ as inputs. The operation adds a new node c with label $l(c)$ in G, i.e., $V = V \cup \{c\}$. It also adds a new edge (suc, c), i.e., $E = E \cup \{(suc, c)\}$ where node suc's label is $l(suc)$. The label of the edge (suc, c) is "rdfs:subClassOf".

- **addTableColumnProperty (columnName):** Allows creation of a table column property.

Listing 5.6: *addTableColumnProperty (columnName)* example API call.

```
API:
---
addTableColumnProperty (columnName)

INPUT:
-----
columnName = Name of the table column.

Example:
columnName = 'hasYear'

OUTPUT:
-------
<http://www.ndssl.bi.vt.edu/fuse#hasYear>
<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>
<http://www.ndssl.bi.vt.edu/fuse#hasColumn> .
```

Formal Definition:
Let column node $acol$ and column node col be elements of V. The label of $acol$ is $l(acol)$ ="fused:hasColumn" and label of col is $l(col)$. Assume $acol$ and col already exist in G.

addTableColumnProperty is an operation that takes $l(col)$ as an input. The operation adds a new edge $(col, acol)$ in G, i.e., $E = E \cup \{(col, acol)\}$ where node col's label is $l(col)$. The label of the edge $(col, acol)$ is "rdfs:subPropertyOf."

- **addFileColumnProperty (fileColumnName):** Allows creation of a file column property.

Listing 5.7: *addFileColumnProperty (fileColumnName)* example API call.

```
API:
---
addFileColumnProperty (fileColumnName)

INPUT:
-----
fileColumnName = Name of the file column.

Example:
columnName = 'hasColumnFirstPerson'

OUTPUT:
-------
<http://www.ndssl.bi.vt.edu/fuse#hasColumnFirstPerson>
<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>
<http://www.ndssl.bi.vt.edu/fuse#hasFileColumn> .
```

Formal Definition:
Let master file column node $mfcol$ and file column node $fcol$ be elements of V. The label of $mfcol$ is $l(mfcol)$ ="fuse:hasFileColumn" and label of $fcol$ is $l(fcol)$. Assume $mfcol$ and $fcol$ already exist in G.

addFileColumnProperty is an operation that takes $l(fcol)$ as input. The operation adds a new edge $(fcol, mfcol)$ in G, i.e., $E = E \cup \{(fcol, mfcol)\}$ where node $fcol$'s label is $l(fcol)$. The label of the edge $(fcol, mfcol)$ is "rdfs:subPropertyOf."

- **keyValueFileColumn (keyName):** Allows creation of a key-value file column property.

Listing 5.8: *keyValueFileColumn (keyName)* example API call.

```
API:
---
keyValueFileColumn (keyName)

INPUT:
-----
keyName = Name of the file key.

Example:
keyName = 'hasColumnContactNetworkFileLocation'

OUTPUT:
-------
<http://www.ndssl.bi.vt.edu/fuse#hasColumnContactNetworkFileLocation>
<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>
<http://www.ndssl.bi.vt.edu/fuse#hasKeyValueFileColumn> .
```

Formal Definition:
Let master key value column node $mkcol$ and key column node $kcol$ be elements of V. The label of $mkcol$ is $l(mkcol)$ ="fused:hasKeyValueFileColumn" and label of $kcol$ is $l(kcol)$. Assume $mkcol$ and $kcol$ already exist in G.

keyValueFileColumn is an operation that takes $l(kcol)$ as input. The operation adds a new edge $(kcol, mkcol)$ in G, i.e., $E = E \cup \{(kcol, mkcol)\}$ where node $kcol$'s label is $l(kcol)$. The label of the edge $(kcol, mkcol)$ is "rdfs:subPropertyOf."

- **classStorage (schemaClassName, physicalStorageClassName):** Links a schema class to a physical storage class.

Listing 5.9: *classStorage (schemaClassName, physicalStorageClassName)* example API call.

```
API:
---
classStorage (schemaClassName, physicalStorageClassName)

INPUT:
-----
schemaClassName = Name of the schema class.
physicalStorageClassName = Name of the physical level storage class.

Example:
schemaClassName = 'ChicagoPersonWithinAdminRegion'
physicalStorageClassName = 'ChicagoPersonOracleTable'

OUTPUT:
-------
<http://www.ndssl.bi.vt.edu/fuse#ChicagoPersonWithinAdminRegion>
<http://www.ndssl.bi.vt.edu/fuse#hasStorage>
<http://www.ndssl.bi.vt.edu/fuse#ChicagoPersonOracleTable> .
```

Formal Definition:
Let schema class node shc and physical storage class node psc be elements of V. The labels of shc and psc are respectively $l(shc)$ and $l(psc)$. Assume shc and psc already exist in G.

classStorage is an operation that takes $l(shc)$ and $l(psc)$ as inputs. The operation adds a new edge (shc, psc) in G, i.e., $E = E \cup \{(shc, psc)\}$, where nodes shc and psc's labels are $l(shc)$ and $l(psc)$. The label of the edge (shc, psc) is "fused:hasStorage."

- **classInstanceStorage (schemaClassInstanceName, physicalStorageClassInstanceName):** Links a schema class instance to a physical storage class instance.

Listing 5.10: *classInstanceStorage (schemaClassInstanceName, physicalStorageClassInstanceName)* example API call.

```
API:
---
classInstanceStorage (schemaClassInstanceName, physicalStorageClassInstanceName)

INPUT:
-----
schemaClassInstanceName = Name of the schema class instance.
physicalStorageClassInstanceName = Name of the physical level storage class instance.

Example:
schemaClassInstanceName = 'RI_EpiStudy'
physicalStorageClassInstanceName = 'RI_Epistudy_Table'

OUTPUT:
```

```
-------                                                                    15
<http://www.ndssl.bi.vt.edu/fuse#RI_EpiStudy>                              16
<http://www.ndssl.bi.vt.edu/fuse#hasStorage>                               17
<http://www.ndssl.bi.vt.edu/fuse#RI_Epistudy_Table> .                      18
```

Formal Definition:
Let schema class instance node $shci$ and physical storage class instance node $psci$ be elements of V. The labels of $shci$ and $psci$ are respectively $l(shci)$ and $l(psci)$. Assume $shci$ and $psci$ already exist in G.

classInstanceStorage is an operation that takes $l(shci)$ and $l(psci)$ as inputs. The operation creates a new edge $(shci, psci)$, i.e., $E = E \cup \{(shci, psci)\}$ in G, where nodes $shci$ and $psci$'s labels are $l(shci)$ and $l(psci)$. The label of the edge $(shci, psci)$ is "fused:hasStorage."

- **classPropertyStorage (classPropertyName, physicalStoragePropertyName):** Links a schema class property to a physical storage class property.

Listing 5.11: *classPropertyStorage (classPropertyName, physicalStoragePropertyName)* example API call.

```
API:                                                                       1
---                                                                        2
classPropertyStorage (classPropertyName, physicalStoragePropertyName)     3
                                                                           4
INPUT:                                                                     5
-----                                                                      6
classPropertyName = Name of the schema class property.                     7
physicalStoragePropertyName = Name of the physical storage class property. 8
                                                                           9
Example:                                                                   10
classPropertyName = 'hasID'                                                11
physicalStoragePropertyName = 'hasIdentificationNumber'                    12
                                                                           13
OUTPUT:                                                                    14
-------                                                                    15
<http://www.ndssl.bi.vt.edu/fuse#hasID>                                    16
<http://www.ndssl.bi.vt.edu/fuse#storedAt>                                 17
<http://www.ndssl.bi.vt.edu/fuse#hasIdentificationNumber> .                18
```

Formal Definition:
Let class property node cpn and physical storage property node psn be elements of V. The labels of cpn and psn are respectively $l(cpn)$ and $l(psn)$. Assume cpn and psn already exist in G.

classPropertyStorage is an operation that takes $l(cpn)$ and $l(psn)$ as inputs. The operation adds a new edge (cpn, psn) in G, i.e., $E = E \cup \{(cpn, psn)\}$ where nodes cpn and psn's labels are $l(cpn)$ and $l(psn)$. The label of the edge (cpn, psn) is "fuse:storedAt."

- **mapping (FUSEDSchema):** Creates mapping files from FUSED schema.

Listing 5.12: *physicalMapping (FUSEDSchema)* example API call.

```
API:
---
mapping (FUSEDSchema)

INPUT:
-----
FUSEDSchema = FUSED schema triples.

Example:
.....................................................
<http://www.ndssl.bi.vt.edu/fuse#hasAge>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#NamedIndividual> .
.....................................................

OUTPUT:
-------

Conceptual Level Mapping:
-------------------------------------------------
.........................................
map:person a d2rq:PropertyBridge;
        d2rq:property vocab:hasAge;
.................................................
-------------------------------------------------

Logical Level Mapping:
-------------------------------------------------
.................................................
map:chicago_person_WAR a d2rq:PropertyBridge;
        d2rq:property vocab:hasPersonID;
.........................................
map:chicago_person_WAR a d2rq:PropertyBridge;
        d2rq:property vocab:hasAge;
.................................................
-------------------------------------------------

Physical Level Mapping:
-------------------------------------------------
.................................................
map:chicago_person_2009_1 a d2rq:PropertyBridge;
        d2rq:property vocab:person_pid;
.........................................
map:chicago_person_2009_1 d2rq:PropertyBridge;
        d2rq:property vocab:person_age;
.................................................
-------------------------------------------------
```

Formal Definition:
The FUSED schema is a graph $FG = (FGV, FGE)$, where FGV is the set of FUSED schema nodes and FGE is the set of directed edges such that $FGE = \{(fgu, fgv) : fgu \in FGV, fgv \in FGV, \text{and } fgu \neq fgv\}$. Let $l(fgu)$ be a label of node fgu.

mapping is an operation that finds all the OWL class instances oci and their corresponding properties $ocip$ from FG. This operation creates mapping class $mcls$ for each oci and mapping property $mpro$ for each $ocip$ and their corresponding nodes and edges, with primary key and foreign key relationships. This operation creates a physical level mapping graph $PMG = (PMV, PME)$, where PMV is the set of physical level mapping nodes and PME is the set of directed edges. Next, this operation uses PMG and creates a logical level mapping graph $LMG = (LMV, LME)$ (where LMV is the set of logical level mapping nodes and LME is the set of directed edges) by modifying PMV and PME with logical level concepts. Finally, this operation uses LMG and creates a conceptual level mapping graph $CMG = (CMV, CME)$ (where CMV is the set of conceptual level mapping nodes and CME is the set of directed edges) by considering domain-centric common CMV (classes) and CME (properties).

5.6.2 Query APIs

We are proposing to develop various query APIs on top of reported and synthetic data schemas for computational epidemiologists to use. In the following, we provide examples.

Proposed APIs for Reported Data

- **getCountry ():** Allows an end-user to retrieve all the country names affected by various epidemics. This API does not take any input parameter.

Listing 5.13: *getCountry ()* example API call.

```
API:
---
getCountry ()

INPUT:
-----
No input.

OUTPUT:
------
[{"country": "Liberia", "epidemicName": "Ebola" },
.........................................]
```

Preliminaries:
A reported RDF graph $RG = (RV, RE)$, where RV is the set of nodes, and RE is the set of directed edges, where $RE = \{(ru, rv) : ru \in V, rv \in V, \text{and } ru \neq rv\}$. Let $l(ru)$ be the label of a node ru.

Formal Definition:
Let country class node $ctrn$ be an element of RV.

getCountry is an operation. It finds the instances of $ctrn$ and for each instance this operation retrieves a lower level child node's label information. It does not take any input parameter.

- **getAdminLevel1 (country):** Allows end-user to retrieve first level administrative division names of countries affected by various epidemics.

Listing 5.14: *getAdminLevel1 (country)* example API call.

```
API:
---
getAdminLevel1 (country)

INPUT:
-----
country = Country name.

Parameter Option: 'country' is a mandatory parameter.

Example: Country = 'Liberia'

OUTPUT:
------
[{ "adminlevel1" : "Bomi"},
.........................
]
```

Formal Definition:
Let country administrative region level one class node $cadmin1$, a country literal value node ctr, and admin1 literal value node $adm1$ be the elements of RV. $l(cadmin1)$ is "Admin1" class, $l(ctr)$ is a country name and $l(adm1)$ is a country's administrative region one name.

getAdminLevel1 is an operation. It takes $l(ctr)$ as input and retrieves the corresponding admin1 name from $cadmin1$.

- **getAdminLevel2 (country, adminLevel1):** Allows an end-user to retrieve second level administrative division names of countries affected by various epidemics.

Listing 5.15: *getAdminLevel2 (country, adminLevel1)* example API call.

```
API:
---
getAdminLevel2 (country, adminLevel1)

INPUT:
-----
country = Country name.
adminLevel1 = First level administrative division of a country.

Parameter Option: 'country' and 'adminLevel1' both of them are mandatory parameters.

Example:
country = 'Liberia'
adminLevel1 = 'Bomi'

OUTPUT:
------
[{"adminlevel2": "Dowein"},
.........................
]
```

Formal Definition:
Let country administrative region level two class node $cadmin2$ and admin2 literal node $adm2$ be the elements of RV. The Label of $adm2$ is $l(adm2)$.

getAdminLevel2 is an operation. It takes $l(ctr)$ and $l(adm1)$ as inputs and retrieves $l(adm2)$ by joining $cadmin1$ and $cadmin2$.

- **getDataset (country, adminLevel1, adminLevel2):** Provides dataset names based on a country name and its administrative division information.

Listing 5.16: *getDataset (country, adminLevel1, adminLevel2)* example API call.

```
API:
---
getDataset (country, adminLevel1, adminLevel2)

INPUT:
-----
country = Country name.
adminLevel1 = First level administrative division of the country.
adminLevel2 = Second level administrative division of the country.

Parameter Option: 'country' is a
mandatory parameter. 'adminLevel1'
and 'adminLevel2' are optional parameters.

Example:
country = 'Liberia'
```

```
adminLevel1 = 'Bomi'
adminLevel2 = 'Dowein'

OUTPUT:
------
[{"datasetName" : "Ebola TimeSeries 2015"},
.....................................
]
```

Formal Definition:
getDataset is an operation. It takes $l(ctr)$, $l(adm1)$ and $l(adm2)$ as inputs and retrieves dataset information by joining $ctrn$, $cadmin1$, $cadmin2$.

- **getDatasetByEpidemic (country, adminLevel1, adminLevel2, epidemicName):** This API provides dataset names based on country, administrative division information, and epidemic name.

Listing 5.17: *getDatasetByEpidemic (country, adminLevel1, adminLevel2, epidemicName)* example API call.

```
API:
---
getDatasetByEpidemic (country, adminLevel1, adminLevel2, epidemicName)

INPUT:
-----
country = Country name.
adminLevel1 = First level administrative division of the country.
adminLevel2 = Second level administrative division of the country.
epidemicName = Epidemic name.

Parameter Option: 'country' and
'epidemicName' are mandatory parameters.
'adminLevel1' and 'adminLevel2'
are optional parameters.

Example:
country = 'Liberia'
adminLevel1 = 'Bomi'
adminLevel2 = 'Dowein'
epidemicName = 'Ebola'

OUTPUT:
------
[{"datasetName": ""Liberia ETU Constructions"},
.....................................
]
```

Formal Definition:
Let epidemic class node $cepidemic$ and epidemic literal node epi be elements of RV. $l(cepidemic)$ is "Epidemic" and $l(epi)$ represents an infectious disease epidemic name.

getDatasetByEpidemic is an operation. It takes $l(ctr)$, $l(adm1)$, $l(adm2)$, and $l(epi)$ as inputs and retrieves epidemic dataset names by joining $ctrn$, $cadmin1$, $cadmin2$, and $cepidemic$.

- **getProperty (epidemicName, datasetName):** Permits a user to retrieve a dataset property.

Listing 5.18: *getProperty (epidemicName, datasetName)* example API call.

```
API:
---
getProperty (epidemicName, datasetName)

INPUT:
-----
epidemicName = Epidemic name.
datasetName = Dataset name.

Parameter Option: Both ‘epidemicName’ and ‘datasetName’ are mandatory parameters.

Example:
epidemicName = 'Ebola'
datasetName = 'EbolaTimeSeries2015'

OUTPUT:
------
[{"property ":"Cases" }
.............................
]
```

Formal Definition:
Let dataset class node $cdat$ and dataset name value node $dset$ be elements of RV. Here $l(dset)$ represents a dataset name.

getProperty is an operation. It takes $l(epi)$ and $l(dset)$ as inputs and retrieves outgoing edges of $dset$ by joining $cepidemic$ and $cdat$.

- **getCount (epidemicName, country, adminLevel1, adminLevel2, startDate, endDate, type):** Allows retrieving count (e.g., case, death, etc.) information from time series dataset based on a range of dates.

Listing 5.19: *getCount (epidemicName, country, adminLevel1, startDate, endDate, type)* example API call.

```

API:
---
getCount (epidemicName, country, adminLevel1, startDate, endDate, type)

INPUT:
-----
epidemicName = Epidemic name.
country = Country name.
```

```
adminLevel1 = First level administrative division of the country.
adminLevel1 = Second level administrative division of the country.
startDate = Start date of an epidemic.
endDate =  End date of an epidemic.
type =  Count category (Possible categories are Case and Death.)

Parameter Option: ‘epidemicName’ ,
‘country’ and ‘startDate’ are
mandatory parameters. Others are
optional parameter.

Example:
epidemicName = ’Ebola’
datasetName = ’EbolaTimeSeries2015’
OUTPUT:
------
[{"cases" : 5000},
.................
]
```

Formal Definition:

Let time series class node cts, start date literal node sd, end date literal node nd, and type literal node ty be elements of RV.

getCount is an operation. It takes $l(epi)$, $l(ctr)$, $l(adm1)$, $l(adm2)$, sd, nd, ty as inputs. It retrieves count information from cts, $cepidemic$, $ctrn$, $cadmin1$, $cadmin2$.

- **getMedicalFacilities (epidemicName, country, adminLevel1):** Retrieves available medical facilities information for a location.

Listing 5.20: *getMedicalFacilities (epidemicName, country, adminLevel1)* example API call.

```

API:
---
getMedicalFacilities (epidemicName, country, adminLevel1)

INPUT:
-----
epidemicName = Epidemic name.
country = Country name.
adminLevel1 = First level administrative
division of the country.

Parameter Option: ‘epidemicName’ , and ‘country’
are mandatory parameters. ‘adminLevel1’ is
optional parameter.

Example:
epidemicName = ‘Ebola’
country = ‘Liberia’
adminLevel1 = ‘Bomi’

OUTPUT:
------
```

```
[{"name": " Mount Barclay Clinic", "occupancy ": 100,
"latitude": 6.3486, "longitude ":  -10.654} ,
.....................................................
]
```

- **getReliefMaterial (epidemicName, country, adminLevel1):** Retrieves information about relief material distributed in a location.

Listing 5.21: *getReliefMaterial (epidemicName, country, adminLevel1)* example API call.

```

API:
---
getReliefMaterial (epidemicName, country, adminLevel1)

INPUT:
-----
epidemicName = Epidemic name.
country = Country name.
adminLevel1 = First level
administrative division of the country.

Parameter Option: 'epidemicName' ,
and 'country' are mandatory parameters.
'adminLevel1' is optional parameter.

Example:
epidemicName = 'Ebola'
country = 'Liberia'
adminLevel1 = 'Bomi'

OUTPUT:
------
[{"name": "Shipment Item Mask_Respirator_Industrial",
"shipmentQuantity ": 10,"supplierName": "Uline"},
.....................................................
]
```

Formal Definition:

Let medical facilities class node $mf \in V$. $l(mf)$ is "MedicalFacilities".

getMedicalFacilities is an operation. It takes $l(epi)$, $l(ctr)$, and $l(adm1)$ as inputs. It retrieves mf instances literal values by joining $cepidemic$, $ctrn$, $cadmin1$, and mf.

- **getParameter (epidemicName):** Retrieves epidemic disease parameter information.

Listing 5.22: *getParameter (epidemicName)* example API call.

```
API:
---
getParameter (epidemicName)

INPUT:
-----
epidemicName = Epidemic name.

Parameter Option:
'epidemicName' is mandatory parameter.

Example:
epidemicName = 'Ebola'

OUTPUT:
------
[{"parameterName" : "Incubation Period Distribution (days)"},
.....................................................
]
```

Formal Definition:
Let disease parameter class node $dp \in V$. $l(dp)$ is "DiseaseParameter".

getParameter is an operation. It takes $l(epi)$ as input. It retrieves disease parameter information by joining $cepidemic$ and dp.

- **getPublication (epidemicName, publicationType):** Retrieves epidemic-related publication information.

Listing 5.23: *getPublication (epidemicName, publicationType)* example API call.

```
API:
---
getPublication (epidemicName, publicationType)

INPUT:
-----
epidemicName = Epidemic name.
publicationType = Type of the publication.
For example News Article, Situation Report,
Research publication, etc.).

Parameter Option: Both 'epidemicName'
and 'publicationType' are mandatory
parameters.

Example:
epidemicName = 'Ebola'
publicationType = 'Research Publication'
```

```
OUTPUT:
------
[{"tile": "A review of epidemiological parameters
from Ebola outbreaks to inform early public health
decision-making.", "author": "Van Kerkhove,
Maria D and Bento, Ana I and Mills,
Harriet L and Ferguson, Neil M and Donnelly,
Christl A", "year": 2015,
"publisher": "Nature Publishing Group",....},
.................................................
]
```

Formal Definition:
Let publication class node pub and $pubt$ be elements of RV. Here the label of pub is $l(pub)$ ="Publication" and label of $pubt$ is $l(pubt)$.

getPublication is an operation that takes $l(epi)$ and $l(pubt)$ as inputs. It retrieves publication information by joining $cepidemic$, pub, and $pubt$.

Proposed APIs for Synthetic Data

- **getAllRegion ():** Retrieves all region information.

Listing 5.24: *getAllRegion ()* example API call.

```
API:
---
getAllRegion ()

INPUT:
-----
No input

OUTPUT:
------
[{"name":"Chicago",
"id":81,
"description":"Chicago, IL region",
"population":8972437,
"region":"Chicago",
"state":"IL",
"country":"USA"},
................................]
```

Preliminaries:
A linked synthetic data RDF graph $SG = (SV, SE)$, where SV is the set of nodes, and SE is the set of directed edges such that $SE = SE\{(su, sv) : su \in SV, sv \in SV, and\ su \neq sv\}$. Let $l(su)$ be the label of a node su.

Formal Definition:
Let region class node reg be an element of SV. Here $l(reg)$ ="Region"

getAllRegion is an operation. It finds the instances of reg and for each instance this operation retrieves lower level child nodes label information. It does not take any input parameter.

- **getAllEpiStudy ():** Retrieves all experiment information.

Listing 5.25: *getAllEpiStudy ()* example API call.

```
API:
---
getAllEpiStudy ()

INPUT:
-----
No input

OUTPUT:
------
[{"name":"Mid Flu Study",
"id":41,"owner":"sample",
"modifiedDate":"Apr 29 2016,15:45",
"description":"Description",
"areaId":81,
"initialCondition":157,
"diseaseId":5,
"replicates":10,
"exptLength":100,
"status":"Complete",
"versionId":9782,
"modelId":1},
.................................]
```

Formal Definition:
Let epistudy class node $epis$ be an element of SV. Here $l(epis)$ is "EpiStudy".

getAllEpiStudy is an operation. It finds the instances of $epis$ and for each instance this operation retrieves property information. It does not take any input parameter.

- **getInfectionInformation (regionName, experimentName):** Retrieves infector and infectee of a experiment conducted in a particular region.

Listing 5.26: *getInfectionInformation (regionName, experimentName)* example API call.

```
API:
---
getInfectionInformation (regionName, studyName)

INPUT:
-----
regionName = Name of the region.
studyName = Name of the study.

Both of the parameters are mandatory for this API.

Example:

regionName= 'Chicago'
studyName = 'Mid Flu Study'


OUTPUT:
------
[{"infectee":442061435,
"infector":445683978},
.....................
]
```

Formal Definition:

Let dendrogram class node den, region literal value node $regl$, and epistudy literal value node $epil$ be elements of SV. Here the label of $regl$ is $l(regl)$ and label of $epil$ is $l(epil)$.

getInfectionInformation is an operation. It takes $l(regl)$ and $l(epil)$ as inputs. It retrieves infection information from den by using $l(regl)$ and $l(epil)$.

- **getAge (infectee, infector):** Retrieves infector and infectee age information.

Listing 5.27: *getAge (infectee, Infector)* example API call.

```
API:
---
getAge (infectee, infector)

INPUT:
-----
infectee = Infectee ID.
infector = Infector ID.
Both of the parameters are mandatory for this API.

Example:
infectee = 442061435,
infector = 445683978

OUTPUT:
------
[{"infectee":142356715,
"infecteeage":65,
"infector":445683978,
"infectorage":10},
.....................]
```

Formal Definition:
Let person class node per, infectee literal node $infec$, and infector literal node $intor$ be elements of SV. Here the label of $infec$ is $l(infec)$ and label of $intor$ is $l(intor)$.

getAge is an operation that takes $l(infec)$ and $l(intor)$ as inputs. It retrieves infectee and infector ages from per class instances.

- **countDemographicGroups (infectee, infecteeage, infector, infectorage):** Counts infector and infectee between various demographic groups.

Listing 5.28: *countDemographicGroups (infectee, infecteeage, infector, infectorage)* example API call.

```
API:
---
countDemographicGroups (infectee, infecteeage, infector, infectorage)
INPUT:
-----
infectee = Infectee ID.
infecteeage = Infectee Age.
infector = Infector ID.
infectorage = Infector Age.

All the parameters are mandatory for this API.

Example:
infectee = 142356715,
infecteeage = 65,
infector = 445683978,
infectorage = 10
```

```
OUTPUT:
------
[{"infectee": "School Age",
"infector": "Middle Age","count":486},
.................................]
```

Formal Definition:

Let infectee age literal node $infecage$ and infector age node $intorage$ be elements of SV. Here label of $infecage$ is $l(infecage)$ and label of $intorage$ is $l(intorage)$.

countDemographicGroups is an operation that takes $l(infec)$, $l(infecage)$, $l(intor)$, and $l(intorage)$. It calculate various demographic group infection counts from den and per instances.

We can divide APIs mentioned above into various levels (Table 5.53). Multiple APIs can be employed to answer a complex query (Figure 5.21).

Table 5.53: Reported and synthetic data API levels.

API Level	Reported Data	Synthetic Data
Level 1	getCountry(), getProperty (epidemicName, datasetName), getParameter(epidemicName), getPublication (epidemicName , publicationType)	getAllRegion(), getAllEpiStudy()
Level 2	getAdminLevel1 (country)	getInfectionInformation (regionName,studyName)
Level 3	getAdminLevel2 (country , adminLevel1), getMedicalFacilities (epidemicName , country , adminLevel1), getReliefMaterial (epidemicName , country , adminLevel1)	getAge(infectee, Infector)
Level 4	getCount(epidemicName, country, adminLevel1, adminLevel2, startDate, endDate, type), getDatasetByEpidemic (country, adminLevel1, adminLevel2, epidemicName), getDataset (country, adminLevel1, adminLevel2)	countDemographicGroups (infectee, infecteeage, infector, infectorage)

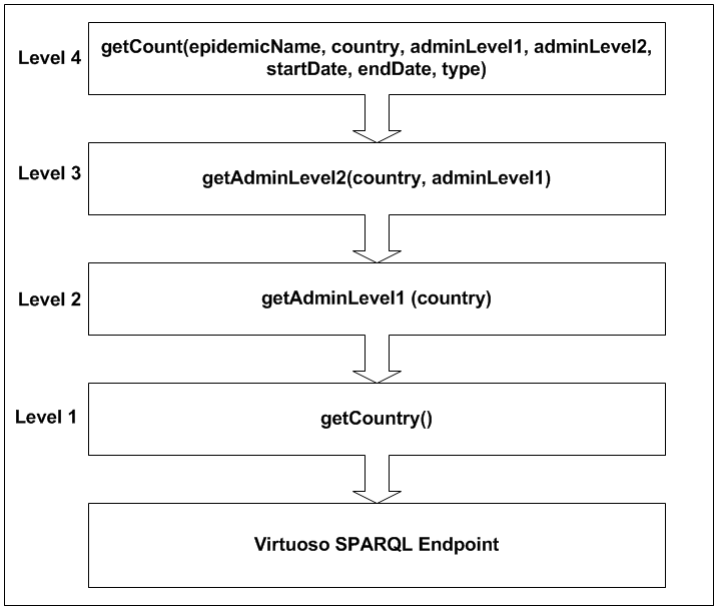


Figure 5.21: API level structures to answer complex queries (find how many Ebola deaths occurred in a country region for a particular time interval).

5.6.3 Synthetic and Reported Data Schemas Linking Architecture

A synthetic population is fragmented based on regions (e.g., country, state, county, etc.). We capture this aspect in the FUSED schema (see Section 3.1.2 for details). Also, we organize reported data schema by using country hierarchical (e.g., country, admin1, admin2, etc.) information (see Section 4.2.4 for details). Hence, it is possible to link synthetic and reported data schemas by using region information. *EpiStudy* is conducted in a region. The final output of an *EpiStudy* is a *Dendrogram* (which contains who infected whom information). Researchers collect epidemic-related reported data from various regions. One key type of reported datasets are *TimeSeries* datasets (mainly contains case count information). We can compare the *Dendrogram* with *TimeSeries* dataset based on a region to help in computational epidemiology simulation model validation and calibration. We present our synthetic and reported schemas linking architecture in Figure 5.22.

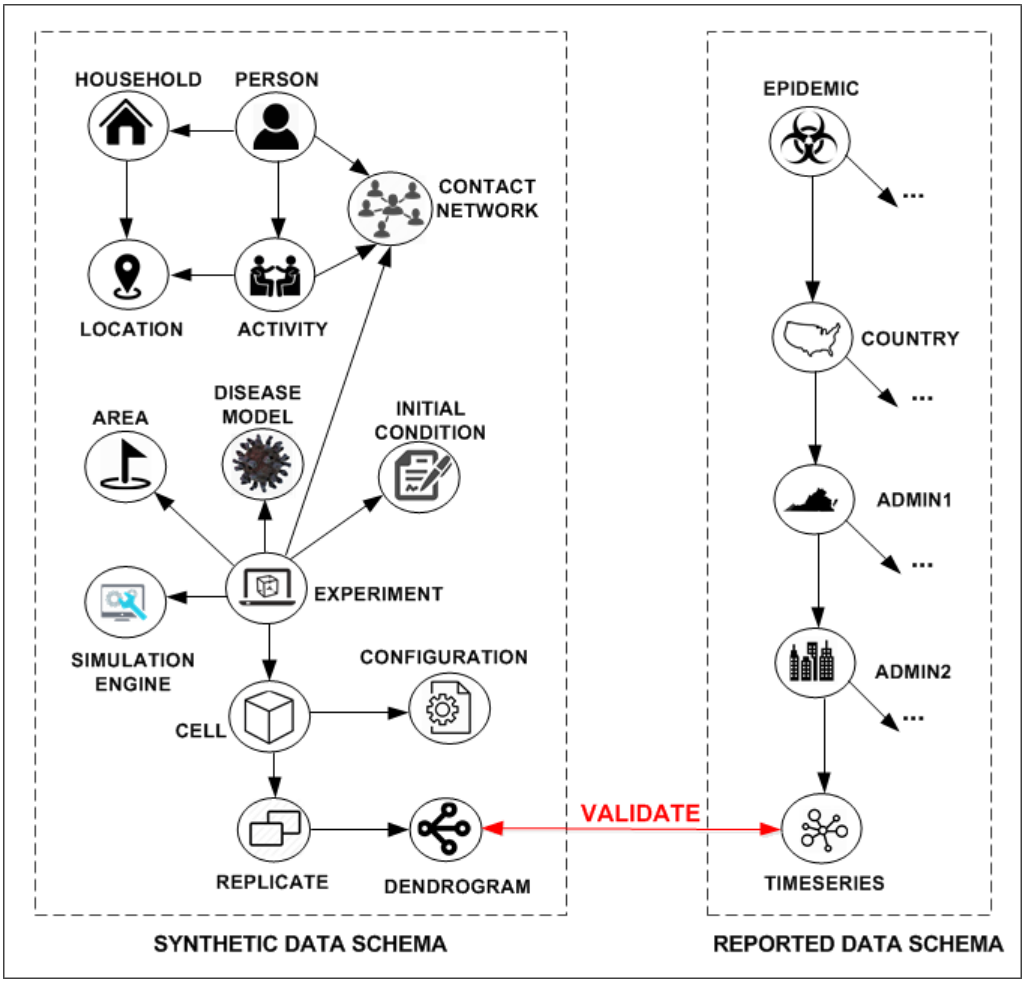


Figure 5.22: Synthetic and reported data schemas linking architecture.

5.7 Summary

We provide a solution for how to create a linked database for heterogeneous and federated computational epidemiology data sources. Preliminary evaluation demonstrates that leveraging semantic web technology (in particular RDF concepts) is an effective strategy for handling data-centric challenges faced in such distributed, multi-user large-scale computational science. Our approach demonstrates that fast unified access across fragmented datasets is feasible. The query bank provides examples of the kinds of analysis that can be done using our approach. As users add new types of queries to the system, the task of an analyst can become progressively easier. To the best of our knowledge this is the first attempt to develop a query bank and a benchmark suite in this area. Finally, by running various epidemiologically relevant queries with two types of data mapping techniques, we demonstrate the performance of various tools. The empirical results show that our proposed framework is capable of extracting complex query results

from heterogeneous data sources, with performance comparable to the state-of-the-art technologies. Different strategies offer trade-offs with respect to storage space or query time. Query execution in a pure RDF engine is faster, but has a large storage cost. In a the hybrid relational-RDF engine we have zero RDF graph storage cost but efficiency is reduced.

Our efforts have focused primarily on infectious disease epidemiology – extensions to chronic disease and environmental epidemiology would be interesting next steps. Extensions will also be needed when studying vector-borne and water-borne diseases to represent vectors, climatic, hydrological, environmental, and ecological datasets. Extension of our approach to study infectious diseases such as Malaria and Zika require data pertaining to vectors, weather, and the habitat of the vector.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Massive numbers of datasets, some of which are very large, are used and generated in computational epidemiology. We present a semantic web-based scientific digital library infrastructure to facilitate and improve the efficient use and analysis of these data sources.

We propose schemas that formally illustrate epidemics reported, synthetic dataset entities, and the semantic relations between the two. Our schemas serve as a data structure and are comprehensive enough to represent heterogeneous and fragmented datasets accurately: a one-stop linked data service. Our schemas are also extensible, employing a graph-based data model (RDF). A graph is a flexible data structure capable of handling continuous data growth. This feature is essential in computational epidemiology considering the frequent schema changes that take place in this domain (see Chapters 3 and 4 for details). The abovementioned features have supported our hypothesis: "Heterogeneous, fragmented, and dynamic datasets found in computational epidemiology can be better managed through semantic web-based linked schemas for both synthetic and reported data."

Our proposed schemas follow a three-tier architecture: conceptual, logical, and physical. The conceptual schema provides data abstraction by exposing common data properties that fit an end user's (epidemiologist's) mental model. It is easy for end users to write queries in a conceptual schema. A conceptual schema does not encompass lower level details, such as storage, layout, and representation, but is related to the logical schema that includes linking aspects of the datasets and data fragmentation. Further, the logical schema is connected to the physical schema that includes the dataset's storage aspects. We can create mapping files from these three layers (conceptual, logical, and physical) of a schema. The mapping files support virtual and materialized graph database creation (see Chapters 3 and 4 for details). Therefore, the above characteristics have supported our hypothesis: "Semantic

web-based schemas are highly beneficial to epidemiologists because they permit both the data abstraction that hides redundancy existing in the physical level storage, and mapping file creation that links conceptual data to physical data, and are helpful in creating a virtual and materialized graph database."

The proposed schemas include innovative data linking approaches, which can connect large-scale computational epidemiology reported and synthetic datasets of many differing types. Such linking facilities allow for combining data into a global data space. A new data source can be integrated promptly upon discovery. Data connected in this way form an integrated knowledge-base and allow epidemiologists to execute complex queries that use multiple datasets. We developed a query bank to provides examples of the kinds of analysis that can be done using our schemas. By running queries from the query bank, we demonstrate the performance of various tools. The empirical results are proof of the capacity of the proposed schemas to extract highly complex query results efficiently from various data sources. Without our linked schema, answering such complex queries is a manual, tedious, and laborious process (see Chapters 3, 4, and 5 for details). Therefore we have provided support for our hypothesis: "Semantic web-based schemas support efficient complex query execution spanning across various mixes of heterogeneous and fragmented computational epidemiology datasets."

We implemented an Ebola data portal based on semantic web technologies. The complete corpus is launched as Linked Open Data (LOD). Hence, it is an openly available web of data in a form that is machine readable. It is available online for faceted search, navigation, and browsing.

Last, we also provide different APIs to allow epidemiological information retrieval, without a need for an end-user to be thoroughly informed about underlying datasets. This is highly beneficial for epidemiologists to comprehend disease propagation for timely outbreak detection and efficient control activities.

6.2 Future Work

A domain-specific language (DSL) can be developed on top of the semantic web-based schemas to support a user base that includes epidemiologists, policymakers, and government officials. The DSL will allow domain scientists to easily generate complex queries, and can be advanced enough to decouple users from lower level query complexity and physical level storage implementation. The idea is to allow experts in various disciplines to focus on obtaining domain-specific insights from a linked knowledge-base instead of worrying about lower level programming syntax. This will be a better way to foster computational epidemiology interdisciplinary research and critically improve the productivity of domain

scientists.

The computational epidemiology domain encompasses a large volume of datasets available for intervention and analysis. These datasets can be used to extend our schemas and knowledge-base development work. It is possible to do further research on the automatic incorporation of a massive amount of datasets related to disease models into our knowledge-base. We can incorporate vector-borne diseases requiring complex within-host disease progression automata.

Our research can be extended to develop a vast, machine-understandable LOD knowledge-base with synthetic and reported computational epidemiology data for various diseases, including smallpox and human monkeypox, and coronaviruses (SARS, MERS-CoV), etc. Subsequently, our knowledge-base can be connected to datasets such as BioGateway [166], BioPortal [64], and BioModels [167]. To gain even deeper knowledge about factors of disease transmission, our knowledge-base can be linked to environmental, behavioral, demographic, pollution, meteorological, and medical datasets. This would allow scientists to mine healthcare datasets to discover more fine-grained information, improving research by a great margin. Moreover, this research can further extend toward developing a link discovery engine by employing machine learning and natural language processing algorithms. The engine can predict all possible links between diverse and continuously changing data sources. It is possible to create a self-sustainable and self-manageable system on top of our knowledge-base. Hence, a graphical user interface (GUI) can be developed to provide an automated capability for end users to contribute and link datasets into our knowledge-base. This GUI can be further extended to provide searching, browsing, crawling, curating, provenance tracking, reasoning, visualizing, and query recommendation services. Moreover, our knowledge-base can be used for dynamic data mining and to send automatic notifications about disease spread information to healthcare professionals and the general public. Hence, our knowledge-base will support the concept of an Internet of Things (IoT). In this way, the system will be beneficial to government officials, policymakers, epidemiologists, and the public.

Bibliography

[1] M. Kretzschmar and M. Morris, "Measures of concurrency in networks and the spread of infectious disease," *Mathematical biosciences*, vol. 133, no. 2, pp. 165–195, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0025556495000933

[2] F. Brauer, "Compartmental models in epidemiology," in *Mathematical epidemiology*. Springer, 2008, pp. 19–79. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-78911-6_2

[3] S. Shaikh, G. Mehta, and R. Garg, "SIBEL 3.1.1 User Manual," https://ndssl.vbi.vt.edu/cms/files/4/35f8fcbd-3c14-d9f4-01ad-724a3bdd1e8f/1831/SIBEL3.1.1_User%20Manual.pdf, 2016, [Online; accessed 2017-07-27].

[4] K. R. Bisset, J. Chen, X. Feng, Y. Ma, and M. V. Marathe, "Indemics: an interactive data intensive framework for high performance epidemic simulation," in *Proceedings of the 24th ACM International Conference on Supercomputing*. ACM, 2010, pp. 233–242. [Online]. Available: http://dl.acm.org/citation.cfm?id=1810118

[5] P. Ross, "Ebola Death Toll 2014: How Many People Have Really Died From Virus?" http://www.ibtimes.com/ebola-death-toll-2014-how-many-people-have-really-died-virus-1711477, [Online; accessed 2016-09-04].

[6] C. M. Rivers, E. T. Lofgren, M. Marathe, S. Eubank, and B. L. Lewis, "Modeling the impact of interventions on an epidemic of Ebola in Sierra Leone and Liberia," *arXiv preprint arXiv:1409.4607*, 2014. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4325479/

[7] J. Zwiebel, L. Tozzi, and T. Griffin, "Heres How Many People Have Died From Ebola So Far," http://www.buzzfeed.com/justinezwiebel/heres-how-many-people-have-died-from-ebola-so-far#.pr1n79JMQ, BuzzFeedNEWS, 2015, [Online; accessed 2016-09-04].

[8] BBC News Staff, "Zika outbreak: What you need to know?" http://www.bbc.com/news/health-35370848, BBC NEWS, 2016, [Online; accessed 2016-09-04].

[9] M. Morin, "Bill Gates: World must prepare for outbreaks deadlier than Ebola," http://www.latimes.com/science/sciencenow/la-sci-sn-gates-ebola-20150319-story.html, Los Angeles Times, 2015, [Online; accessed 2016-09-04].

[10] D. Addington, "Ebola: Dallas and New York City Experiences Drive Governments to Change Practices," http://www.heritage.org/research/reports/2014/10/ebola-dallas-and-new-york-city-experiences-drive-governments-to-change-practices, The Heritage Foundation, 2014, [Online; accessed 2016-09-29].

[11] M. V. Marathe and V. S. A. Kumar, "Computational Epidemiology," *Communications of the ACM (CACM)*, vol. 56, no. 7, pp. 88–96, 2013. [Online]. Available: http://doi.acm.org/10.1145/2483852.2483871

[12] E. T. Lofgren, M. E. Halloran, C. M. Rivers, J. M. Drake, T. C. Porco, B. L. Lewis, W. Yang, A. Vespignani, J. Shaman, J. Eisenberg, M. C. Eisenberg, M. V. Marathe, S. V. Scarpino, K. A. Alexander, R. Meza, M. J. Ferrari, J. M. Hyman, L. A. Meyers, and S. G. Eubank, "Opinion: Mathematical models: A key tool for outbreak response," *Proceedings of the National Academy of Sciences (PNAS)*, vol. 111, no. 51, p. 18095–18096, 2014. [Online]. Available: http://www.pnas.org/content/111/51/18095.short

[13] R. Beckman, K. R. Bisset, J. Chen, B. Lewis, M. Marathe, and P. Stretz, "ISIS: A networked-epidemiology based pervasive Web app for infectious disease pandemic planning and response," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 1847–1856. [Online]. Available: http://dl.acm.org/citation.cfm?id=2623375

[14] Biocomplexity Institute, "Synthetic Information Based Epidemiological Laboratory (SIBEL) App," http://ndssl.vbi.vt.edu/apps/sibel/, Biocomplexity Institute at Virginia Tech, [Online; accessed 2016-07-08].

[15] S. Hutcheson, "The EpiC Marketplace," URL: http://cns.iu.edu/data_tools.html, Indiana University, [Online; accessed 2016-09-11].

[16] W. Van den Broeck, C. Gioannini, B. Gonçalves, M. Quaggiotto, V. Colizza, and A. Vespignani, "The GLEaMviz computational tool, a publicly available software to explore realistic epidemic spreading scenarios at the global scale," *BMC infectious diseases*, vol. 11, no. 1, p. 1, 2011. [Online]. Available: https://bmcinfectdis.biomedcentral.com/articles/10.1186/1471-2334-11-37

[17] J. J. Grefenstette, S. T. Brown, R. Rosenfeld, J. DePasse, N. T. Stone, P. C. Cooley, W. D. Wheaton, A. Fyshe, D. D. Galloway, A. Sriram *et al.*, "FRED (A Framework for Reconstructing Epidemic Dynamics): an open-source software system for modeling infectious diseases and control strategies using census-based populations," *BMC public health*, vol. 13, no. 1, p. 1, 2013. [Online]. Available: https://bmcpublichealth.biomedcentral.com/articles/10.1186/1471-2458-13-940

[18] S. Brown, "The geospatial area and information analyzer (GAIA)." URL: http://gaia.psc.edu/, Pittsburgh Supercomputing Center, [Online; accessed 2016-09-11].

[19] Greg Madey, "Vector-Borne Disease Network (VecNet)," URL: https://www.vecnet.org/, 2013, Vector-borne disease network., [Online; accessed 2016-09-11].

[20] "Resource Description Framework," URL: https://en.wikipedia.org/wiki/Resource_Description_Framework, Wikipedia, [Online; accessed 2015-04-10].

[21] T. Dutta, M. Khan, V. S. A. Kumar, M. V. Marathe, and Z. Zhao, "Structural and Relational Properties of Social Contact Networks with Applications to Public Health Informatics," 2009. [Online]. Available: http://people.cs.vt.edu/zhaozhao/papers/GraphMining_TR.pdf

[22] S. Pyne, M. V. Marathe, and V. S. A. Kumar, *Big Data Applications in Health Sciences and Epidemiology, Handbook of Statistics, Volume 33: Big Data Analytics*. Elsevier. Edited by V. Govindaraju, V.V. Raghavan, C.R. Rao, 2015. [Online]. Available: https://ndssl.vbi.vt.edu/cms/files/2/36ae5989-2674-05e4-c5fa-67cb286165f8/1784/978-0-444-63492-4.pdf

[23] E. A. Fox, M. A. Gonçalves, and R. Shen, "Theoretical foundations for digital libraries: The 5S (societies, scenarios, spaces, structures, streams) approach," *Synthesis Lectures on Information Concepts, Retrieval, and Services*, vol. 4, no. 2, pp. 1–180, 2012. [Online]. Available: http://www.morganclaypool.com/doi/abs/10.2200/S00434ED1V01Y201207ICR022

[24] J. M. Pratt and M. Cohen, "A process-oriented scientific database model," *ACM SIGMOD Record*, vol. 21, no. 3, pp. 17–25, 1992. [Online]. Available: http://dl.acm.org/citation.cfm?id=140987

[25] J. M. Pratt, "Data modeling of scientific experimentation," in *Proceedings of the 1995 ACM symposium on Applied computing*. ACM, 1995, pp. 86–90. [Online]. Available: http://dl.acm.org/citation.cfm?id=315913

[26] T. H. Jordan, "SCEC 2009 Annual Report," *Southern California Earthquake Center*, 2009. [Online]. Available: http://files.scec.org/s3fs-public/2009SCECAnnualMeetingVolume.pdf

[27] Borgman, Christine L. and Wallis, Jillian C. and Mayernik, Matthew S. and Pepe, Alberto, "Drowning in data: digital library architecture to support scientific use of embedded sensor networks," in *Proc. JCDL 2007*, 2007, pp. 269–277. [Online]. Available: http://doi.acm.org/10.1145/1255175.1255228

[28] Leonardo Candela, Donatella Castelli, Pasquale Pagano, "D4Science: an e-infrastructure for supporting virtual research," in *Proceedings of IRCDL 2009 - 5th Italian Research Conference on Digital Libraries*, 2009, pp. 166–169. [Online]. Available: https://pdfs.semanticscholar.org/0c18/ea62ce71bf63fa6a7ffe7ad55bca56b558f9.pdf#page=176

[29] R. K. France, "Effective, Efficient Retrieval in a Network of Digital Information Objects," Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2001. [Online]. Available: http://hdl.handle.net/10919/29754

[30] C. L. Barrett, K. Bisset, S. Eubank, E. A. Fox, Y. Ma, M. Marathe, and X. Zhang, "A scalable data management tool to support epidemiological modeling of large urban regions," in *Research and Advanced Technology for Digital Libraries*, 2007, pp. 546–548. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-74851-9_65

[31] L. M. Schriml, C. Arze, S. Nadendla, A. Ganapathy, V. Felix, A. Mahurkar, K. Phillippy, A. Gussman, S. Angiuoli, E. Ghedin *et al.*, "GeMInA, Genomic Metadata for Infectious Agents, a geospatial surveillance pathogen database," *Nucleic acids research*, vol. 38, no. suppl 1, pp. D754–D764, 2010. [Online]. Available: https://academic.oup.com/nar/article/38/suppl_1/D754/3112211/GeMInA-Genomic-Metadata-for-Infectious-Agents-a

[32] J. P. Leidig, "Epidemiology Experimentation and Simulation Management through Scientific Digital Libraries," Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2012. [Online]. Available: http://hdl.handle.net/10919/28759

[33] H. Shi, Y. Zhang, J. Zhang, P. Wan, and K. Shaw, "Development of Web-Based Epidemiological Reporting System for Tasmania Utilizing a Google Maps Add-On," in *Digital Image Computing Techniques and Applications, 9th Biennial Conference of the Australian Pattern Recognition Society on*. IEEE, 2007, pp. 118–123. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/4426785/

[34] D. Allon and P. Nicholson, "Data Modelling for an Epidemiological Database," URL: http://www.sascommunity.org/seugi/SEUGI1997/ALLON_POSTERS.PDF, 1997, [Online; accessed 2015-03-25].

[35] J. F. Sequeda, M. Arenas, and D. P. Miranker, "On directly mapping relational databases to RDF and OWL," in *Proceedings of the 21st international conference on World Wide Web*. ACM, 2012, pp. 649–658. [Online]. Available: http://dl.acm.org/citation.cfm?id=2187924

[36] A. Bertails and E. G. Prud'hommeaux, "Interpreting relational databases in the RDF domain," in *Proceedings of the sixth international conference on Knowledge capture*. ACM, 2011, pp. 129–136. [Online]. Available: http://doi.acm.org/10.1145/1999676.1999699

[37] P. E. Salas, E. Marx, A. Mera, and J. Viterbo, "RDB2RDF plugin: relational databases to RDF plugin for Eclipse," in *Proceedings of the 1st Workshop on Developing Tools as Plug-ins*. ACM, 2011, pp. 28–31. [Online]. Available: http://dl.acm.org/citation.cfm?id=1984717

[38] A. Zappa, A. Splendiani, and P. Romano, "Towards linked open gene mutations data," *BMC bioinformatics*, vol. 13, no. Suppl 4, p. S7, 2012. [Online]. Available: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-13-S4-S7

[39] T. Dalamagas, N. Bikakis, G. Papastefanatos, Y. Stavrakas, and A. G. Hatzigeorgiou, "Publishing life science data as linked open data: the case study of miRBase," in *Proceedings of the First International Workshop on Open Data*. ACM, 2012, pp. 70–77. [Online]. Available: http://dl.acm.org/citation.cfm?id=2422615

[40] M. Hert, G. Reif, and H. C. Gall, "Updating relational data via SPARQL/update," in *Proceedings of the 2010 EDBT/ICDT Workshops*. ACM, 2010, p. 24. [Online]. Available: http://dl.acm.org/citation.cfm?id=1754266

[41] R. Piro, Y. Nenov, B. Motik, I. Horrocks, P. Hendler, S. Kimberly, and M. Rossman, "Semantic Technologies for Data Analysis in Health Care," in *International Semantic Web Conference*. Springer, 2016, pp. 400–417. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-46547-0_34

[42] I. Horrocks, M. Giese, E. Kharlamov, and A. Waaler, "Using Semantic Technology to Tame the Data Variety Challenge," *IEEE Internet Computing*, vol. 20, no. 6, pp. 62–66, 2016. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7781544/

[43] M. D. Van Kerkhove, A. I. Bento, H. L. Mills, N. M. Ferguson, and C. A. Donnelly, "A review of epidemiological parameters from ebola outbreaks to inform early public

health decision-making," *Scientific data*, vol. 2, p. 150019, 2015. [Online]. Available: https://www.nature.com/articles/sdata201519?report=reader

[44] Y. Lu, X. Hu, F. Wang, S. Kumar, H. Liu, and R. Maciejewski, "Visualizing social media sentiment in disaster scenarios," in *Proceedings of the 24th International Conference on World Wide Web*. ACM, 2015, pp. 1211–1215. [Online]. Available: http://dl.acm.org/citation.cfm?id=2741720

[45] V. Veljkovic, P. M. Loiseau, B. Figadere, S. Glisic, N. Veljkovic, V. R. Perovic, D. P. Cavanaugh, and D. R. Branch, "Virtual screen for repurposing approved and experimental drugs for candidate inhibitors of EBOLA virus infection," *F1000Research*, vol. 4, 2015. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4329668.1/

[46] M. P. Fallah, L. A. Skrip, S. Gertler, D. Yamin, and A. P. Galvani, "Quantifying poverty as a driver of Ebola transmission," *PLoS neglected tropical diseases*, vol. 9, no. 12, p. e0004260, 2015. [Online]. Available: http://journals.plos.org/plosntds/article?id=10.1371/journal.pntd.0004260

[47] P. Richards, J. Amara, M. C. Ferme, P. Kamara, E. Mokuwa, A. I. Sheriff, R. Suluku, and M. Voors, "Social pathways for Ebola virus disease in rural Sierra Leone, and some implications for containment," *PLoS neglected tropical diseases*, vol. 9, no. 4, p. e0003567, 2015. [Online]. Available: http://journals.plos.org/plosntds/article?id=10.1371/journal.pntd.0003567

[48] A. Wesolowski, C. O. Buckee, L. Bengtsson, E. Wetter, X. Lu, and A. J. Tatem, "Commentary: containing the Ebola outbreak-the potential and challenge of mobile network data," *PLoS currents*, vol. 6, 2014. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4205120/

[49] X. Liu, E. Speranza, C. Muñoz-Fontela, S. Haldenby, N. Y. Rickett, I. Garcia-Dorival, Y. Fang, Y. Hall, E.-G. Zekeng, A. Lüdtke *et al.*, "Transcriptomic signatures differentiate survival from fatal outcomes in humans infected with Ebola virus," *Genome biology*, vol. 18, no. 1, p. 4, 2017. [Online]. Available: https://genomebiology.biomedcentral.com/articles/10.1186/s13059-016-1137-3

[50] J. M. Drake, I. Bakach, M. R. Just, S. M. O'Regan, M. Gambhir, and I. C.-H. Fung, "Transmission models of historical Ebola outbreaks," *Emerging infectious diseases*, vol. 21, no. 8, p. 1447, 2015. [Online]. Available: TransmissionmodelsofhistoricalEbolaoutbreaks

[51] I. S. Caballero, A. N. Honko, S. K. Gire, S. M. Winnicki, M. Melé, C. Gerhardinger, A. E. Lin, J. L. Rinn, P. C. Sabeti, L. E. Hensley *et al.*, "In vivo Ebola virus infection leads to a strong innate response in circulating immune cells," *BMC genomics*, vol. 17, no. 1, p. 707, 2016. [Online]. Available: https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-016-3060-0

[52] Y. Li and S. Chen, "Evolutionary history of Ebola virus," *Epidemiology & Infection*, vol. 142, no. 6, pp. 1138–1145, 2014. [Online]. Available: https://www.cambridge.org/core/journals/epidemiology-and-infection/article/evolutionary-history-of-ebola-virus/8594C381318B4E265B898A30EE665A49

[53] T. Stadler, D. Kühnert, D. A. Rasmussen, and L. du Plessis, "Insights into the early epidemic spread of Ebola in Sierra Leone provided by viral sequence data," *PLoS currents*, vol. 6, 2014. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4205153/

[54] T. House, "Epidemiological dynamics of Ebola outbreaks," *Elife*, vol. 3, p. e03908, 2014. [Online]. Available: EpidemiologicaldynamicsofEbolaoutbreaks

[55] M. Giovanetti, A. Grifoni, A. Lo Presti, E. Cella, C. Montesano, G. Zehender, V. Colizzi, M. Amicosante, and M. Ciccozzi, "Amino acid mutations in Ebola virus glycoprotein of the 2014 epidemic," *Journal of medical virology*, vol. 87, no. 6, pp. 893–898, 2015. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/jmv.24133/full

[56] T. Tran and K. Lee, "Understanding citizen reactions and Ebola-related information propagation on social media," in *Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on*. IEEE, 2016, pp. 106–111. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7752221/

[57] M. Odlum and S. Yoon, "What can we learn about the Ebola outbreak from tweets?" *American journal of infection control*, vol. 43, no. 6, pp. 563–571, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196655315001376

[58] L. Hunt, A. Gupta-Wright, V. Simms, F. Tamba, V. Knott, K. Tamba, S. Heisenberg-Mansaray, E. Tamba, A. Sheriff, S. Conteh *et al.*, "Clinical presentation, biochemical, and haematological parameters and their association with outcome in patients with Ebola virus disease: an observational cohort study," *The Lancet infectious diseases*, vol. 15, no. 11, pp. 1292–1299, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1473309915001449

[59] K. H. Rubins, L. E. Hensley, V. Wahl-Jensen, K. M. D. DiCaprio, H. A. Young, D. S. Reed, P. B. Jahrling, P. O. Brown, D. A. Relman, and T. W. Geisbert, "The temporal program

of peripheral blood gene expression in the response of nonhuman primates to Ebola hemorrhagic fever," *Genome biology*, vol. 8, no. 8, p. R174, 2007. [Online]. Available: https://genomebiology.biomedcentral.com/articles/10.1186/gb-2007-8-8-r174

[60] M. A. Bornea, J. Dolby, A. Kementsietsidis, K. Srinivas, P. Dantressangle, O. Udrea, and B. Bhattacharjee, "Building an efficient RDF store over a relational database," in *Proceedings of the 2013 international conference on Management of data*. ACM, 2013, pp. 121–132. [Online]. Available: http://dl.acm.org/citation.cfm?id=2463718

[61] J. Groppe, S. Groppe, S. Ebers, and V. Linnemann, "Efficient processing of SPARQL joins in memory by dynamically restricting triple patterns," in *Proceedings of the 2009 ACM symposium on Applied Computing*. ACM, 2009, pp. 1231–1238. [Online]. Available: http://dl.acm.org/citation.cfm?id=1529560

[62] M. Hert, G. Reif, and H. C. Gall, "A comparison of RDB-to-RDF mapping languages," in *Proceedings of the 7th International Conference on Semantic Systems*. ACM, 2011, pp. 25–32. [Online]. Available: http://dl.acm.org/citation.cfm?id=2063522

[63] M. Arenas, B. C. Grau, E. Kharlamov, Š. Marciuška, and D. Zheleznyakov, "Faceted search over RDF-based knowledge graphs," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 37, pp. 55–74, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570826815001432

[64] N. F. Noy, N. H. Shah, P. L. Whetzel, B. Dai, M. Dorf, N. Griffith, C. Jonquet, D. L. Rubin, M.-A. Storey, C. G. Chute *et al.*, "BioPortal: ontologies and integrated data resources at the click of a mouse," *Nucleic acids research*, vol. 37, no. suppl 2, pp. W170–W173, 2009. [Online]. Available: https://academic.oup.com/nar/article/37/suppl_2/W170/1154845/BioPortal-ontologies-and-integrated-data-resources

[65] H. Chen, Z. Wu, G. Zheng, and Y. Mao, "RDF-based schema mediation for database grid," in *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*. IEEE, 2004, pp. 456–460. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/1382866/

[66] M. R. Kamdar and M. Dumontier, "An Ebola virus-centered knowledge base," *Database: the journal of biological databases and curation*, vol. 2015, 2015. [Online]. Available: https://academic.oup.com/database/article/doi/10.1093/database/bav049/2433182/An-Ebola-virus-centered-knowledge-base

[67] S. Castano and V. De Antonellis, "Global viewing of heterogeneous data sources," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 2, pp. 277–297, 2001. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/917566/

[68] L. F. Lopes, F. A. Silva, F. Couto, J. Zamite, H. Ferreira, C. Sousa, and M. J. Silva, "Epidemic marketplace: an information management system for epidemiological data," in *Information Technology in Bio-and Medical Informatics, ITBAM 2010*. Springer, 2010, pp. 31–44. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-642-15020-3.pdf#page=41

[69] M. J. Silva, F. A. B. D. Silva, L. F. Lopes, and F. M. Couto, "Building a digital library for epidemic modelling," in *Proceedings of ICDL 2010 - The International Conference on Digital Libraries*. TERI Press, 2010, pp. 23–27. [Online]. Available: http://xldb.fc.ul.pt/xldb/publications/Silva.etal:BuildingADigital:2010_document.pdf

[70] M. J. Silva, "EPIWork," http://xldb.di.fc.ul.pt/wiki/EPIWork, XLDB, 2013, [Online; accessed 2016-09-19].

[71] S. Bergamaschi, S. Castano, and M. Vincini, "Semantic integration of semistructured and structured data sources," *ACM Sigmod Record*, vol. 28, no. 1, pp. 54–59, 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=309897

[72] Y. Zhu and J. Ferreira Jr, "Data Integration to Create Large-Scale Spatially Detailed Synthetic Populations," in *Planning Support Systems and Smart Cities*. Springer, 2015, pp. 121–141. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-18368-8_7

[73] C. Pasquier, "Biological data integration using Semantic Web technologies," *Biochimie*, vol. 90, no. 4, pp. 584–594, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0300908408000412

[74] S. Gupta, "A unified data model and declarative query language for heterogenous life sciences data," *San Diego Super Computing Center, UCSD, Tech. Rep. SDSC TR-2011-3*, 2011. [Online]. Available: https://scholar.google.com/citations?user=Fuu-Ov4AAAAJ&hl=en

[75] C. Goble and R. Stevens, "State of the nation in data integration for bioinformatics," *Journal of biomedical informatics*, vol. 41, no. 5, pp. 687–693, 2008. [Online]. Available: Stateofthenationindataintegrationforbioinformatics

[76] K.-H. Cheung, K. Y. Yip, A. Smith, A. Masiar, M. Gerstein *et al.*, "YeastHub: a semantic web use case for integrating data in the life sciences domain," *Bioinformatics*, vol. 21, no. suppl 1, pp. i85–i96, 2005. [Online]. Available: https://academic.oup.com/bioinformatics/article/21/suppl_1/i85/203201/YeastHub-a-semantic-web-use-case-for-integrating

[77] D. Caragea, J. Pathak, J. Bao, A. Silvescu, C. Andorf, D. Dobbs, and V. Honavar, "Information integration and knowledge acquisition from semantically heterogeneous biological data sources," in *International Workshop on Data Integration in the Life Sciences*. Springer, 2005, pp. 175–190. [Online]. Available: https://link.springer.com/content/pdf/10.1007/11530084.pdf#page=186

[78] A. Birkland and G. Yona, "BIOZON: a system for unification, management and analysis of heterogeneous biological data," *BMC bioinformatics*, vol. 7, no. 1, p. 70, 2006. [Online]. Available: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-7-70

[79] B. Amann, I. Fundulaki, and M. Scholl, "Integrating ontologies and thesauri for RDF schema creation and metadata querying," *International Journal on Digital Libraries*, vol. 3, no. 3, pp. 221–236, 2000. [Online]. Available: https://link.springer.com/article/10.1007%2Fs007990000037?LI=true

[80] L. M. Schriml, C. Arze, S. Nadendla, Y.-W. W. Chang, M. Mazaitis, V. Felix, G. Feng, and W. A. Kibbe, "Disease Ontology: a backbone for disease semantic integration," *Nucleic acids research*, vol. 40, no. D1, pp. D940–D946, 2012. [Online]. Available: https://academic.oup.com/nar/article/40/D1/D940/2903651/Disease-Ontology-a-backbone-for-disease-semantic

[81] L. G. Cowell and B. Smith, "Infectious disease ontology," in *Infectious disease informatics*. Springer, 2010, pp. 373–395. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4419-1327-2_19

[82] B. Yang, S. Sayers, Z. Xiang, and Y. He, "Protegen: a web-based protective antigen database and analysis system," *Nucleic acids research*, p. gkq944, 2010. [Online]. Available: https://academic.oup.com/nar/article/39/suppl_1/D1073/2508846/Protegen-a-web-based-protective-antigen-database

[83] C. Pesquita, J. D. Ferreira, F. M. Couto, and M. J. Silva, "The epidemiology ontology: an ontology for the semantic annotation of epidemiological resources," *Journal of biomedical semantics*, vol. 5, no. 1, p. 1, 2014. [Online]. Available: https://jbiomedsem.biomedcentral.com/articles/10.1186/2041-1480-5-4

[84] J. D. Ferreira, C. Pesquita, F. M. Couto, and M. J. Silva, "Bringing epidemiology into the semantic web." in *ICBO*, 2012. [Online]. Available: http://webpages.fc.ul.pt/~fjcouto/files/conference%20jferreira-icbo2012.pdf

[85] B. Smith, M. Ashburner, C. Rosse, J. Bard, W. Bug, W. Ceusters, L. J. Goldberg, K. Eilbeck, A. Ireland, C. J. Mungall *et al.*, "The OBO Foundry:

coordinated evolution of ontologies to support biomedical data integration," *Nature biotechnology*, vol. 25, no. 11, pp. 1251–1255, 2007. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2814061/

[86] P. Grenon, B. Smith, and L. Goldberg, "Biodynamic ontology: applying BFO in the biomedical domain," *Studies in health technology and informatics*, pp. 20–38, 2004. [Online]. Available: https://books.google.com/books?hl=en&lr=&id=k0X9gljExgQC&oi=fnd&pg=PA20&dq=Biodynamic+ontology:+applying+BFO+in+the+biomedical+domain&ots=LNFAUo2TTF&sig=cYn5L55DidETY7U9SP8gACIBS_s#v=onepage&q=Biodynamic%20ontology%3A%20applying%20BFO%20in%20the%20biomedical%20domain&f=false

[87] C. Stoeckert, "The information artifact ontology." URL: https://code.google.com/archive/p/information-artifact-ontology/, Google Code Archive, [Online; accessed 2016-09-20].

[88] W. R. Hogan, M. M. Wagner, M. Brochhausen, J. Levander, S. T. Brown, N. Millett, J. DePasse, and J. Hanna, "The Apollo Structured Vocabulary: an OWL2 ontology of phenomena in infectious disease epidemiology and population biology for use in epidemic simulation," *Journal of Biomedical Semantics*, vol. 7, no. 1, p. 50, 2016. [Online]. Available: https://jbiomedsem.biomedcentral.com/articles/10.1186/s13326-016-0092-y

[89] B. A. Prakash, "CS 4604: Introduction to Database Management Systems," URL: http://courses.cs.vt.edu/~cs4604/Spring16/lectures/lecture-2.pdf, 2016, Virginia Tech, [Online; accessed 2016-06-17].

[90] Techopedia contributors, "Primary Key," URL: https://www.techopedia.com/definition/5547/primary-key, Techopedia Inc., [Online; accessed 2016-06-17].

[91] Wikipedia contributors, "Foreign Key," URL: https://en.wikipedia.org/wiki/Foreign_key, Wikipedia, [Online; accessed 2016-06-17].

[92] Branko Dimitrijevic , "Difference between super key and composite key," URL: http://stackoverflow.com/questions/24080915/difference-between-super-key-and-composite-key, Stack Overflow, 2014, Stack Overflow, [Online; accessed 2016-08-12].

[93] R. Cyganiak, "RDF 1.1 Concepts and Abstract Syntax," URL: https://www.w3.org/TR/rdf11-concepts/, World Wide Web Consortium (W3C), 2014, [Online; accessed 2016-05-27].

[94] T. Heath and C. Bizer, "Linked data: Evolving the web into a global data space," *Synthesis lectures on the semantic web: theory and technology*, vol. 1, no. 1, pp. 1–136, 2011. [Online]. Available: http://www.morganclaypool.com/doi/abs/10.2200/s00334ed1v01y201102wbe001

[95] D. Brickley, "RDF Schema 1.1," URL: https://www.w3.org/TR/rdf-schema/, 2014, World Wide Web Consortium (W3C), [Online; accessed 2016-05-27].

[96] H. Knublauch, "A Semantic Web Primer for Object-Oriented Software Developers," URL: https://www.w3.org/TR/sw-oosd-primer/, 2006, World Wide Web Consortium (W3C), [Online; accessed 2016-06-15].

[97] "Blank node," https://en.wikipedia.org/wiki/Blank_node, Wikipedia, [Online; accessed 2016-08-14].

[98] Chris Bizer, "Quality driven information filtering: in the context of web based information systems," URL: http://www.diss.fu-berlin.de/diss/servlets/MCRFileNodeServlet/FUDISS_derivate_000000002736/05_Chapter5-Named-Graphs.pdf?hosts=, 2010, [Online; accessed 2016-06-23].

[99] Chris Bizer, "Overview of Named Graphs," URL: http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/Bizer-NamedGraphs-ProvXG.pdf, 2010, [Online; accessed 2016-06-19].

[100] B. DuCharme, *Learning SPARQL*. O'Reilly Media, Inc., 2011. [Online]. Available: https://books.google.com/books?hl=en&lr=&id=j2kXeNeZ00YC&oi=fnd&pg=PR7&dq=Learning+SPARQL.+O%E2%80%99Reilly+Media,+Inc.,+2011&ots=-HWDcgmgPv&sig=YY-ulw53jurTNPjAv8WrZzOtrhA#v=onepage&q&f=false

[101] W3C contributors, "OWL Web Ontology Language Current Status," URL: https://www.w3.org/standards/techs/owl#w3c_all, World Wide Web Consortium (W3C), [Online; accessed 2016-06-17].

[102] Jerome Diaz, "what is the difference between owl:Class and owl:Thing?" URL: http://stackoverflow.com/questions/16362926/what-is-the-difference-between-owlclass-and-owlthing, Stack Overflow, 2013, [Online; accessed 2016-09-15].

[103] S. Bechhofer, "OWL Web Ontology Language Reference," URL: https://www.w3.org/TR/2004/REC-owl-ref-20040210/#Thing-def, World Wide Web Consortium (W3C), [Online; accessed 2016-09-15].

[104] D. Galway, "The OWL 2 Schema vocabulary (OWL 2)," URL: https://prefix.cc/owl, World Wide Web Consortium (W3C), 2004, [Online; accessed 2016-08-05].

[105] S. Hasan, "How to define multiple domain and range connection in Protégé?" URL: http://stackoverflow.com/questions/34817048/how-to-define-multiple-domain-and-range-connection-in-prot%C3%A9g%C3%A9, Stack Overflow, 2015, [Online; accessed 2016-01-15].

[106] Cambridge Semantics contributors, "RDFS Introduction," URL: https://www.cambridgesemantics.com/semantic-university/rdfs-introduction, Cambridge Semantics, [Online; accessed 2016-06-19].

[107] S. Hasan, "OWL Class and Subclass Property Inheritance," URL: http://stackoverflow.com/questions/37854557/owl-class-and-subclass-property-inheritance/37860677#37860677, Stack Overflow, 2016, [Online; accessed 2016-06-19].

[108] R. Cyganiak, "The D2RQ Mapping Language," http://d2rq.org/d2rq-language#deprecated-class-map-property-bridge, Freie Universität Berlin, [Online; accessed 2016-06-28].

[109] R. Angles and C. Gutierrez, "The expressive power of SPARQL," in *International Semantic Web Conference*. Springer, 2008, pp. 114–129. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-540-88564-1_8?LI=true

[110] E. V. Kostylev, J. L. Reutter, and M. Ugarte, "Expressiveness of CONSTRUCT Queries in SPARQL," Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015. [Online]. Available: https://doi.org/10.1007/978-3-540-88564-1_8

[111] J. F. Sequeda and D. P. Miranker, "Ultrawrap: SPARQL execution on relational data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 22, pp. 19–39, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570826813000383

[112] K. R. Bisset, J. Chen, X. Feng, V. Kumar, and M. V. Marathe, "EpiFast: a fast algorithm for large scale realistic epidemic simulations on distributed memory systems," in *Proceedings of the 23rd international conference on Supercomputing*. ACM, 2009, pp. 430–439. [Online]. Available: http://dl.acm.org/citation.cfm?id=1542336

[113] S. Hasan, S. Gupta, E. Fox, K. Bisset, M. V. Marathe *et al.*, "Data mapping framework in a digital library with computational epidemiology datasets," in *Digital Libraries (JCDL), 2014 IEEE/ACM Joint Conference on*. IEEE, 2014, pp. 449–450. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/6970219/

[114] S. Deodhar, "Data Integration Methodologies and Services for Evaluation and Forecasting of Epidemics," Ph.D. dissertation, 2016. [Online]. Available: http://hdl.handle.net/10919/71303

[115] A. Adiga, A. Agashe, S. Arifuzzaman, C. L. Barrett, R. J. Beckman, K. R. Bisset, J. Chen, Y. Chungbaek, S. G. Eubank, S. Gupta, M. Khan, C. J. Kuhlman, E. Lofgren, B. L. Lewis, A. Marathe, M. V. Marathe, H. S. Mortveit, E. Nordberg, C. Rivers, P. Stretz, S. Swarup, A. Wilson, and D. Xie, "Generating a Synthetic Population of the United States," NDSSL, Tech. Rep., 2015. [Online]. Available: https://ndssl.vbi.vt.edu/cms/files/4/b2f197d4-d2e6-5994-d194-cef438939034/1647/US-pop-generation.pdf

[116] C. L. Barrett, K. R. Bisset, S. G. Eubank, X. Feng, and M. V. Marathe, "EpiSimdemics: an efficient algorithm for simulating the spread of infectious disease over large realistic social networks," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 37. [Online]. Available: http://dl.acm.org/citation.cfm?id=1413408

[117] N. Sandsmark, "rdfs:Class," URL: http://www.infowebml.ws/rdf-owl/Class-rdfs.htm, Semantic Web Adoption and Applications, 2015, [Online; accessed 2016-09-19].

[118] N. Sandsmark, "owl:Class," URL: http://www.infowebml.ws/rdf-owl/Class-owl.htm, Semantic Web Adoption and Applications, 2015, [Online; accessed 2016-09-19].

[119] C. Bizer, "D2RQ: Accessing Relational Databases as Virtual RDF Graphs," URL: http://d2rq.org/, [Online; accessed 2016-09-19].

[120] S. C. Betrabet, E. A. Fox, and Q.-F. Chen, "A query language for information graphs," *Technical Report, Department of Computer Science, Virginia Tech, Blacksburg, VA 24061*, 1993. [Online]. Available: http://hdl.handle.net/10919/19783

[121] Apache Software Foundation, "Apache Jena: TDB," https://jena.apache.org/documentation/tdb/, [Online; accessed 2015-10-04].

[122] OpenLink Software Documentation Team, "Virtuoso Open-Source Edition, 2014," URL: http://www.openlinksw.com/dataspace/doc/dav/wiki/Main/, [Online; accessed 2015-03-25].

[123] U.S. Census Bureau, "U.S. Census 2000. 5-Percent Public Use Microdata Sample Files." https://www.census.gov/census2000/PUMS5.html, [Online; accessed 2015-03-25].

[124] D. Booth, "Why RDF for Healthcare Interoperability – Part 2 of Yosemite Series," URL: http://yosemiteproject.org/recorded-webinars/2015-2/why-rdf-for-healthcare-interoperability-part-2-of-yosemite-series/, [Online; accessed 2015-04-10].

[125] S. Jupp, J. Malone, J. Bolleman, M. Brandizi, M. Davies, L. Garcia, A. Gaulton, S. Gehant, C. Laibe, N. Redaschi *et al.*, "The EBI RDF platform: linked open data for the life sciences," *Bioinformatics*, vol. 30, no. 9, pp. 1338–1339, 2014. [Online]. Available: https://academic.oup.com/bioinformatics/article/30/9/1338/234645/The-EBI-RDF-platform-linked-open-data-for-the-life

[126] T. Heath, "Linked data - connect distributed data across the web," URL: http://linkeddata.org/, [Online; accessed 2015-04-10].

[127] S. Harris, "Triple Stores vs Relational Databases," URL: http://stackoverflow.com/questions/9159168/triple-stores-vs-relational-databases, [Online; accessed 2015-04-10].

[128] Apache Software Foundation, "Fuseki: serving RDF data over HTTP," https://jena.apache.org/documentation/serving_data/, [Online; accessed 2015-10-04].

[129] OpenLink Software Documentation Team, "Installation and Configuration of the Virtuoso Faceted Browser," http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtFacetBrowserInstallConfig, [Online; accessed 2015-10-04].

[130] R. Isele, A. Jentzsch, C. Bizer, J. Volz, and P. Petrovski, "Silk - The Linked Data Integration Framework," URL: http://silkframework.org/, [Online; accessed 2015-04-10].

[131] S. Mazzocchi and P. Ciccarese, "WELKIN," http://www.visualcomplexity.com/vc/project.cfm?id=395, Massachusetts Institute of Technology, [Online; accessed 2015-03-25].

[132] G. Lausen, *Relational Databases in RDF: Keys and Foreign Keys*, ser. Lecture Notes in Computer Science, V. Christophides, M. Collard, and C. Gutierrez, Eds. Springer Berlin Heidelberg, 2008, vol. 5005. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-70960-2_3

[133] Wikipedia contributors, "Five Ws," URL: https://en.wikipedia.org/wiki/Five_Ws, Wikipedia, [Online; accessed 2016-10-20].

[134] P. Dolog, F. A. Durão, K. Jahn, Y. Lin, and D. K. Peitersen, "Recommending Open Linked Data in Creativity Sessions using Web Portals with Collaborative

Real Time Environment." *J. UCS*, vol. 17, no. 12, pp. 1690–1709, 2011. [Online]. Available: http://www.jucs.org/jucs_17_12/recommending_open_linked_data/jucs_17_12_1690_1709_dolog.pdf

[135] H. S. Kim, J. H. Son, G. H. Lim, and I. H. Suh, "Semantic Robot Memory Store using 5W1H for Service Tasks," in *International Conference on Advanced Mechatronics*, 2010, pp. 579–584. [Online]. Available: https://www.researchgate.net/profile/Gi_Hyun_Lim/publication/268516006_Semantic_Robot_Memory_Store_using_5W1H_for_Service_Tasks/links/546e7db60cf2b5fc1760782a.pdf

[136] T. Yoshioka, G. Herman, J. Yates, and W. Orlikowski, "Genre taxonomy: A knowledge repository of communicative actions," *ACM Transactions on Information Systems (TOIS)*, vol. 19, no. 4, pp. 431–456, 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=502798

[137] M. R. Johannessen and A. Følstad, "Political Social Media Sites as Public Sphere: A Case Study of the Norwegian Labour Party," *Communications of the Association for Information Systems*, vol. 34, no. 56, pp. 1067–1096, 2014. [Online]. Available: http://aisel.aisnet.org/cais/vol34/iss1/56/

[138] T. Ikeda, A. Okumura, and K. Muraki, "Information classification and navigation based on 5W1H of the target information," in *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*. Association for Computational Linguistics, 1998, pp. 571–577. [Online]. Available: http://dl.acm.org/citation.cfm?id=980940

[139] C. Bizer and A. Schultz, "The Berlin SPARQL Benchmark," http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/, 2009, [Online; accessed 2015-03-25].

[140] R. Cyganiak, "Benchmarking D2RQ v0.2," http://wifo5-03.informatik.uni-mannheim.de/bizer/d2rq/benchmarks/, Freie Universität Berlin, [Online; accessed 2015-03-25].

[141] E. Prud and A. Seaborne, "SPARQL Query Language for RDF," URL: http://www.w3.org/TR/rdf-sparql-query/, World Wide Web Consortium (W3C), 2014, [Online; accessed 2015-03-25].

[142] Oracle Corporation Staff, "Oracle Corporation," URL: https://www.oracle.com/index.html, [Online; accessed 2015-04-10].

[143] PostgreSQL Global Development Group, "PostgreSQL: The world's most advanced open source database," URL: https://www.postgresql.org/, [Online; accessed 2015-04-10].

[144] K. Shinpaugh, "High-Performance Computing and Information Technology," https://www.bi.vt.edu/services/computational-core, [Online; accessed 2016-09-07].

[145] A. P. Galvani and R. M. May, "Epidemiology: dimensions of superspreading," *Nature*, vol. 438, no. 7066, pp. 293–295, 2005. [Online]. Available: http://www.nature.com/nature/journal/v438/n7066/full/438293a.html?foxtrotcallback=true

[146] L. Fumanelli, M. Ajelli, S. Merler, N. M. Ferguson, and S. Cauchemez, "Model-based comprehensive analysis of school closure policies for mitigating influenza epidemics and pandemics," *PLoS Comput Biol*, vol. 12, no. 1, pp. 1–15, 2016. [Online]. Available: http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004681

[147] K. Glass and B. Barnes, "How much would closing schools reduce transmission during an influenza pandemic?" *Epidemiology*, vol. 18, no. 5, pp. 623–628, 2007. [Online]. Available: http://journals.lww.com/epidem/Abstract/2007/09000/How_Much_Would_Closing_Schools_Reduce_Transmission.17.aspx

[148] MIT Governance Lab, "Data for Ebola Recovery," https://data.humdata.org/dataset/data-for-ebola-recovery, Humanitarian Data Exchange, 2015, [Online; accessed 2016-09-06].

[149] Direct Relief, "Direct Relief Ebola Materials Shipped," https://data.humdata.org/dataset/direct-relief-ebola-materials-shipped, Humanitarian Data Exchange, 2015, [Online; accessed 2016-09-06].

[150] UN Mission for Ebola Emergency Response, "Ebola Community Care Centers," https://data.humdata.org/dataset/ebola-community-care-centers, Humanitarian Data Exchange, 2014, [Online; accessed 2016-09-06].

[151] UN Mission for Ebola Emergency Respons, "Ebola Treatment Centers or Units (ETCs or ETUs)," https://data.humdata.org/dataset/ebola-treatment-centers, Humanitarian Data Exchange, 2014, [Online; accessed 2016-09-06].

[152] Humanitarian Data Exchange, "Ebola outbreaks before 2014," https://data.humdata.org/dataset/ebola-outbreaks-before-2014, Humanitarian Data Exchange, 2013, [Online; accessed 2016-09-06].

[153] World Food Programme, "Global Food Prices Database (WFP)," https://data.humdata.org/dataset/wfp-food-prices, Humanitarian Data Exchange, 2016, [Online; accessed 2016-09-06].

[154] Standby Task Force, "Health Facilities Liberia oct 2014," https://data.humdata.org/dataset/health-facilities-liberia-oct-2014, Humanitarian Data Exchange, 2014, [Online; accessed 2016-09-06].

[155] M. V. Kerkhove, A. I. Bento, H. L. Mills, N. M. Ferguson, and C. A. Donnelly, "Table S1 Human Outbreaks of Ebola Zaire, Ebola Sudan and Ebola Bundibugyo from 1976 to present," https://figshare.com/articles/Table_S1_Human_Outbreaks_of_Ebola_Zaire_Ebola_Sudan_and_Ebola_Bundibugyo_from_1976_to_present/1381874, [Online; accessed 2016-09-06].

[156] United States Department of Defense, "Lat/Long/Names of ETUs in Liberia updated as of 21 Oct," https://data.humdata.org/dataset/lat-long-names-of-etus-updated-as-of-21-oct, [Online; accessed 2016-09-06].

[157] United States Department of Defense, "Liberia ETU Constructions," https://data.humdata.org/dataset/etu-constructions, [Online; accessed 2016-09-06].

[158] United Nations Office for the Coordination of Humanitarian Affairs, "Liberia FTS indicators," https://data.humdata.org/dataset/lbr_fts_indicators, [Online; accessed 2016-09-06].

[159] World Health Organization, "Number of Ebola health-care worker deaths," https://data.humdata.org/dataset/number-of-health-care-workers-deaths-by-edv, [Online; accessed 2016-09-06].

[160] World Health Organization, "Number of health-care workers infected with Ebola," https://data.humdata.org/dataset/number-of-health-care-workers-infected-with-edv, [Online; accessed 2016-09-06].

[161] M. V. Kerkhove, A. I. Bento, H. L. Mills, N. M. Ferguson, and C. A. Donnelly, "Table S3 Parameter estimates for the ongoing EVD outbreak in West Africa," https://figshare.com/articles/Table_S3_Parameter_estimates_for_the_ongoing_EVD_outbreak_in_West_Africa/1381877, [Online; accessed 2016-09-06].

[162] M. V. Kerkhove, A. I. Bento, H. L. Mills, N. M. Ferguson, and C. A. Donnelly, "Table S2 Epidemiologic parameters for EVD and Marburg by Study," https://figshare.com/articles/Table_S2_Epidemiologic_parameters_for_EVD_and_Marburg_by_Study/1381876, [Online; accessed 2016-09-06].

[163] World Health Organization, "Safe and Dignified Burial Teams," https://data.humdata.org/dataset/safe-and-dignified-burial-teams, [Online; accessed 2016-09-06].

[164] Global Data Lab, "Sub-national Indicators Ebola Countries," https://data.humdata.org/dataset/sub-national-indicators-ebola-countries, [Online; accessed 2016-09-06].

[165] Office for the Coordination of Humanitarian Affairs, "Sub-national time series data on Ebola cases and deaths in Guinea, Liberia, Sierra Leone, Nigeria, Senegal and Mali since March 2014," https://data.humdata.org/dataset/rowca-ebola-cases, [Online; accessed 2016-09-06].

[166] E. Antezana, W. Blondé, M. Egaña, A. Rutherford, R. Stevens, B. De Baets, V. Mironov, and M. Kuiper, "Biogateway: a semantic systems biology tool for the life sciences," *BMC bioinformatics*, vol. 10, no. 10, p. S11, 2009. [Online]. Available: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-10-S10-S11

[167] N. Le Novere, B. Bornstein, A. Broicher, M. Courtot, M. Donizelli, H. Dharuri, L. Li, H. Sauro, M. Schilstra, B. Shapiro *et al.*, "Biomodels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems," *Nucleic acids research*, vol. 34, no. suppl_1, pp. D689–D691, 2006. [Online]. Available: https://academic.oup.com/nar/article/34/suppl_1/D689/1133255