I, Abhishek Pandey, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.

It is entitled:
Relational Schema Integration Using Ontologies

Student's name: Abhishek Pandey

This work and its defense approved by:

Committee chair: Karen Davis, Ph.D.
Committee member: Raj Bhatnagar, Ph.D.
Committee member: Carla Purdy, Ph.D.
Relational Schema Integration Using Ontologies

A thesis submitted to the
Division of Research and Advanced Studies
of the University of Cincinnati

in the partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in the Department of Electrical Engineering and Computing Systems
of the College of Engineering and Applied Science

July 18, 2014

by
Abhishek Pandey

B.E., Rajeev Gandhi Technical University, Bhopal, 2010

Thesis Advisor and Committee Chair:
Dr. Karen C. Davis
Abstract

With increasing data volume, automatically sharing data becomes more valuable because it decreases storage requirements by reducing data replication. Database integration is an important problem to address in current real world applications. The schema integration problem is as follows: given a group of schemas, the goal is to create a merged schema that represents all of the underlying schemas. The construction of the unified schema should not result in loss of information, and it should grant the capability to query the underlying databases either individually or collectively.

Integration of multiple heterogeneous data sources usually includes an initial step to combine the schemas of the sources so that they form a unified view, which can be used to give users the illusion (and simplicity) of interacting with one single target combined from all the sources. Users are presented with a uniform logical representation of data that is physically spread over heterogeneous data sources. For this, an integrated schema) has to be created that incorporates all the data contents of the sources. Although there have been some projects on integration of the relational schemas but there is no universal tool for achieving the schema integration using ontologies.

In this work, we propose an approach and develop a tool to achieve relational database schema integration using ontologies. Our focus is on improving the automation for integrating relational database schemas by using ontologies. An ontology represents a shared, explicit specification of a conceptualization of the domain of knowledge [G93]. Our schema integration approach is a combination of three different approaches that are adapted from previous research efforts. First we convert relations in the schema into their respective ontologies written in the ontology language OWL. After creating individual ontologies, we then merge them using ontology merging techniques to obtain a unified single ontology. Finally, the unified ontology is converted into a relational database schema. We develop a semi-automated tool that takes two or more relational schemas as input and converts them into their respective ontologies using pre-defined mapping rules. These ontologies are merged into a single unified ontology and we convert this merged ontology into a relational schema. Our approach is demonstrated using two example data sources and our integration tool RIO. Finally, we evaluate the quality of the merged schemas using the approach proposed by Pavlic et al. [PK+08].
Table of Contents

List of Figures ........................................................................................................... v
List of Tables .............................................................................................................. vi
Chapter 1  Introduction ............................................................................................. 1
  1.1 General Research Objective .............................................................................. 2
  1.2 Specific Objectives and Research Methodology .................................................. 2
  1.3 Contributions ...................................................................................................... 3
  1.4 Overview ............................................................................................................. 3
Chapter 2  Relational Schema Integration ................................................................. 4
  2.1 Overview of Relational Schema Integration ......................................................... 4
  2.2 Approaches to Schema Integration .................................................................... 5
  2.3 Challenges in Relational Schema Integration ...................................................... 6
  2.4 Ontology and Ontology Languages ................................................................... 7
  2.5 Integration of Relational Schema Using Ontologies .............................................. 8
  2.6 Approaches for Schema Integration Using Ontologies ......................................... 9
  2.7 Conclusion ......................................................................................................... 11
Chapter 3  Relational Database Schema to Ontology Transformation ....................... 12
  3.1 Approaches for Generating an Ontology from a Relational Schema .................... 13
  3.2 Our Approach to Map a Relational Schema to an Ontology ............................... 15
  3.3 Conclusion ......................................................................................................... 21
Chapter 4  Ontology Merging .................................................................................. 22
  4.1 Ontology Merging Process ................................................................................. 23
  4.2 Ontology Merging Tools/Algorithms .................................................................. 23
  4.3 Alignment and OPTIMA APIs ......................................................................... 25
  4.4 Conclusion ......................................................................................................... 26
Chapter 5  Ontology to Relational Schema ............................................................... 28
  5.1 Approaches for Generating a Relational Schema from an Ontology ..................... 28
  5.2 Transformation of an Ontology to a Relational Schema ...................................... 30
  5.3 Mapping Rules ................................................................................................. 32
  5.4 Data Type Conversion ...................................................................................... 33
  5.5 Conclusion ......................................................................................................... 34
Chapter 6  Schema Integration Methodology with a Case Study .................................. 35
Chapter 7  Contributions and Future Work ............................................................... 60
  7.1 Contributions ..................................................................................................... 60
  7.2 Future Work ...................................................................................................... 60
  7.3 Conclusion ......................................................................................................... 61
References .................................................................................................................. 62
Appendix A Relational Schema and OWL Ontology Generation Datasets ................. 67
Appendix B Merged Schemas Generated using RIO tool .......................................... 97
List of Figures

Figure 2-1: Overview of Relational Schema Integration ................................................................. 4
Figure 2-2: Ontology Based Schema Integration Approaches [WV+01] ............................................... 9
Figure 3-1: Classification of Database-to-Ontology Mapping Approaches [GC+07] .......................... 13
Figure 3-2: Employee Relational Schema [NS+07] .......................................................................... 15
Figure 3-3: Mapping Attributes in OWL ......................................................................................... 18
Figure 3-4: Constraint UNIQUE Is Mapped to an Inverse Functional Property ....................... 18
Figure 3-5: Constraint NOT NULL Is Mapped to Minimum Cardinality of 1 ............................... 19
Figure 3-6: Mapping Primary Key to Ontology ............................................................................ 19
Figure 3-7: Foreign Key Is Mapped to Object Property ................................................................. 20
Figure 3-8: Foreign Key Is Mapped to Class Inheritance ............................................................. 21
Figure 3-9: Foreign Key Is Mapped to Symmetric Property ........................................................... 21
Figure 4-1: Ontology Merging Process .......................................................................................... 23
Figure 5-1: Mapping Between a Relational Schema and an Ontology [AN+07] ............................ 29
Figure 5-2: Transformation of a Relational Schema to an Ontology [AN+07] ............................... 29
Figure 5-3: A Basic Ontological Model [AK+07] ......................................................................... 30
Figure 5-4: Basic Relational Model [AI+07] .................................................................................. 31
Figure 5-5: Transformation of Ontologies to Relational Schemas ................................................. 32
Figure 6-1: RIO tool (main window) ............................................................................................... 37
Figure 6-2: RIO tool (database configuration window) ................................................................. 37
Figure 6-3: RIO tool (Logger window) ........................................................................................... 38
Figure 6-4: Ontology Created ......................................................................................................... 39
Figure 6-5: Opening OWL Ontology in Protégé 4.1 ..................................................................... 40
Figure 6-6: COMPANY Schema [RS+99] ....................................................................................... 41
Figure 6-7: EMPLOYEE Schema [SD+09] ..................................................................................... 41
Figure 6-8: OWL Classes in Protégé 4.1 ......................................................................................... 43
Figure 6-9: Mapped Attributes ..................................................................................................... 43
Figure 6-10: Mapped Primary Key ................................................................................................. 44
Figure 6-11: Foreign Key is not part of Primary Key ...................................................................... 45
Figure 6-12: Foreign Key is Part of Primary Key .......................................................................... 46
Figure 6-13: Cardinality Created ................................................................................................... 46
Figure 6-14: Foreign Key Mapped to Inverse Functional and Symmetric Property .................. 47
Figure 6-15: Company Ontology .................................................................................................. 48
Figure 6-16: Employee Ontology .................................................................................................. 48
Figure 6-17: Alignment Methods ................................................................................................... 49
Figure 6-18: Trimming Threshold .................................................................................................. 50
Figure 6-19: Optima suggestions (COMPANY-EMPLOYEE) ...................................................... 51
Figure 6-20: Optima suggestions (EMPLOYEE- COMPANY) ...................................................... 51
Figure 6-21: SMOANaNameAlignment suggestions (COMPANY-EMPLOYEE) ......................... 52
Figure 6-22: SMOANaNameAlignment suggestions (EMPLOYEE- COMPANY) ......................... 52
Figure 6-23: Resultant Classes in Merged Ontology .................................................................... 53
Figure 6-24: Resultant Data Properties in Merged Ontology ....................................................... 53
Figure 6-25: Resultant Object Properties in Merged Ontology.................................................................54
Figure 6-26: Merged Ontology ......................................................................................................................54
Figure 6-27: Merged Schema Created Window.............................................................................................55
Figure 6-28: Resultant Merged Schema.......................................................................................................57

List of Tables

Table 2-1: Comparison Table........................................................................................................................11
Table 3-1: Common Mapping Strategies.........................................................................................................14
Table 3-2 Correspondence between Relational Model and OWL Features [JS+07] ..................................16
Table 3-3: Correspondence between Relational Model and OWL Datatypes [JS+07] .........................17
Table 5-1: Data Type Conversion [AI+07] .......................................................................................................34
Table 6-1: Mapping Relational Schema to Ontology......................................................................................42
Table 6-2: Mapping Ontology to Relational Schema......................................................................................56
Table 6-3: Complexity of Resultant Schemas.................................................................................................58
Chapter 1  Introduction

Combining data in distinct databases is known as database integration; it is the problem of merging data and providing the user with a unified view in a single database. This process becomes significant when two similar enterprises must merge their databases. With increasing data volume, automatically sharing data becomes more valuable because it decreases storage requirements by reducing data replication. Database integration is an important problem to address in current real world applications, however, defining accurate mappings between the sources and the unified database can be tedious and error-prone.

Integration of multiple data sources usually includes an initial step to combine the schemas of the sources so that they form a unified view, which can be used to give users the illusion (and simplicity) of interacting with one single target combined from all the sources. Users are presented with a uniform logical representation of data that is physically spread over distinct data sources. For this, a description of the data contents (i.e., a schema) has to be created that incorporates all the data contents of the sources. This task includes detection and resolution of structural and semantic conflicts in the source schemas and data. A unified schema can also be used as the structural description of a merged, physically stored repository or warehouse.

One issue encountered in data integration is the semantic integration problem [S03]. The primary concern of the semantic integration problem is finding the best way to resolve semantic inconsistencies between data from heterogeneous sources. For example, various databases may define a similar concept or term in distinct ways. There is no guarantee that a given term will have the same meaning in each database. This causes an inconsistency to arise in the definition of particular terms within a given schema. For example, the word “orange” can refer to a color, a class of fruit, a county in southern California, a county in New York, or a title of nobility from the Netherlands. However, each usage is correct depending upon the context in which it is used. One way of solving such a problem, known as ontology-based data integration (OBDI), uses ontologies to define and map terms in a schema to clarify inconsistent semantic meanings between different uses of the same term. This approach uses ontologies to combine and integrate information from various sources that are heterogeneous in nature.

One source of heterogeneity arises because data can be stored in different ways, including relational databases, spreadsheets, text files, and HTML/XML files. Each data storage format has
particular methods of accessing and cataloguing data unique to a particular format. Because relational databases are among the most widely used sources of data, our research focuses on integrating/mapping relational database schemas using ontologies; we address semantic heterogeneity within relational databases.

We use a step-by-step approach for semi-automated integration of relational database schemas. First we convert schemas into their respective ontologies written in the ontology language OWL. After creating individual ontologies, we then merge them using ontology merging techniques to obtain a unified single ontology. Finally, the unified ontology is converted into a relational database schema.

1.1 General Research Objective

The general research objective of this thesis is to create a semi-automated approach for integrating relational schemas using ontologies and representing the integrated components with a unified relational schema extracted from the unified ontology.

1.2 Specific Objectives and Research Methodology

Developing a semi-automated approach for integrating relational database schemas using ontologies requires several specific research objectives to be addressed:

A. Investigate challenges in the general area of relational schema integration: we review the literature for relational schema integration, focusing on issues related to using ontologies for database schema integration.

B. Investigate reverse engineering approaches to transform a relational database schema into an ontology: we study the literature and select an appropriate methodology to provide the basis for our approach. We leverage the mapping approach proposed by Juric et al. [JS+01]. It uses a set of rules as mappings to create ontologies using OWL and is implemented in the tool we developed for this thesis.

C. Decide on ontology merging techniques to merge resultant ontologies into a single ontology: we survey the literature on ontology merging and select Alignment API [DJ+11] and the Optima API [KD+08] for our ontology merging operation.
D. Investigate techniques to efficiently convert an ontology into a relational database schema: we adapt the approach used by Astrova et al. [AI+07] for the transformation of a unified ontology to a unified relational schema.

E. Demonstrate the functionality of our implementation through a case study and a software tool.

1.3 Contributions

This research implements (1) a methodology to transform relational schemas into respective ontologies using mapping rules, (2) a technique to merge all the resultant ontologies into a single ontology, and (3) an approach to transform an ontology to a relational schema. The techniques are implemented using available software platforms and demonstrated with a case study.

1.4 Overview

Chapter 2 discusses background on relational schema integration research, gives an overview of semantic web technologies such as ontologies, and describes previous work on using ontologies for schema integration. In Chapter 3, a methodology is discussed that converts a relational database schema into an ontology. In Chapter 4, techniques used to merge two or more ontologies are discussed. Chapter 5 illustrates how the merged ontology is converted back into a relational database schema. Chapter 6 includes a detailed demonstration of the proposed methodology for schema integration with a case study. Chapter 7 focuses on the contribution of this research and ideas for future work.
Chapter 2  Relational Schema Integration

This chapter presents related research on relational schema integration and an overview of semantic web technologies such as ontologies and ontology languages. It concludes with a discussion of the use of ontologies in relational schema integration.

2.1 Overview of Relational Schema Integration

The schema integration problem is as follows: given a group of schemas, the goal is to create a merged schema that represents all of the underlying schemas. The construction of the unified schema should not result in loss of information, and it should grant the capability to query the underlying databases either individually or collectively. Relational schema integration merges relations from different relational schema sources into one integrated relational schema as shown in Figure 2-1. The user’s query is answered using the integrated schema. Therefore, he or she need not worry about the organization or data described by individual schemas.

Figure 2-1: Overview of Relational Schema Integration
The integrated schema can be materialized; that is, a new database can be created to store data from the different source databases. In contrast, it can also be a virtual view of the integrated schema and the data may remain in the respective sources. In a materialized schema integration methodology, a replica of the data is made. It is suitable for stable data. The virtual data integration approach provides an interface to autonomous data sources; it can be used for a large amount of data with frequently changing content, so it is suitable for web data, for example. There are possibilities to merge both of the above methods. For example, an integration system can provide a virtual integrated view as well as materialize some data in the cache for frequently accessed data.

According to Batini et al. [BC+86], a schema integration methodology comprises four main steps:

1. **Pre-integration**: an examination of schemas is carried out prior to integration to determine the integration strategy. This choice of schemas that are to be integrated is made and the order of integration is decided.
2. **Schema comparison**: schemas are examined and compared to determine the similarities among concepts and identify possible differences. Inter-schema properties may be determined while comparing schemas.
3. **Conforming schemas**: after conflicts are identified, an attempt is made to fix them to get the intermediate integrated schema.
4. **Merging and restructuring**: the intermediary integrated schema(s) that got generated are analyzed and restructured in order to get desirable results.

We use this methodology as a basis to derive our approach towards relational schema integration using ontologies.

### 2.2 Approaches to Schema Integration

According to Rahm et al. [RB+01] schema integration approaches can be broadly classified according to whether they exploit either schema information or both schema and instance level information.

1) **Schema-level matchers** consider schema information and not instance data. The schema information includes the properties of a schema such as its name, description, data type, relationship types (e.g., part-of or is-a), constraints, and schema structure. Language-
based matchers use names and text to find semantically similar schema elements. Constraint based matchers exploit constraints such as data types and their ranges, uniqueness, optionality, relationship types, and cardinalities of a schema. The results of language based matchers and constraint based matchers are used to determine the similarity between the schema elements of two separate schemas.

2) **Instance-level matchers** use data from instance-level to collect significant insights into the details and meaning of the elements in the schema. These are most commonly used in enhancing the match results as they also take care of instance details. They are used when the information available at the schema level is insufficient. Matchers at this level use linguistic and constraint-based characterizations of instances.

3) **Hybrid matchers** combine different matching approaches to find match results that are based on various criteria and information sources.

We utilize the schema-level matchers approach as part of this thesis to solve the problem of relational schema integration as it is less complex to realize using a semi-automated tool. In the next section, common challenges for relational schema integration are discussed.

### 2.3 Challenges in Relational Schema Integration

Kim et al. [KM+91] mention common challenges in relational schema integration, especially during the phase of matching and mapping in the schema integration process. The authors provide a detailed list of types of heterogeneity that recognizes schematic vs. semantic heterogeneity. Many of these heterogeneities exist because relational schemas use different definitions to describe the same information. The main focus of research efforts is to provide an automated way in finding semantic matches between two schemas [KM+91]. This can be challenging due to heterogeneities at the following levels [S98]:

1. **syntactic heterogeneity** concerns differences between data models (such as entity relationship, relational, and object-oriented),

2. **structural heterogeneity** (schematic heterogeneity) means that different information systems store their data in different structures or arrangements [SK+91,KS+96], and
3. **semantic heterogeneity** considers the content of an information and its intended meaning. To accomplish semantic interoperability in a heterogeneous system, the intent of the information that is interchanged must be understood across the systems [SK+91].

Researchers have proposed the use of ontologies for schema integration to overcome these heterogeneity problems because ontologies are recognized as a key to interoperability [BC+01, WV+01]. We utilize ontologies to facilitate schema integration.

### 2.4 Ontology and Ontology Languages

Ontologies were developed by the artificial intelligence community to promote knowledge sharing and reuse [G98]. Ontologies are mainly used for representing domain knowledge. A typical use of ontologies is data standardization and conceptualization using a formal machine-understandable ontology language. Technically an ontology is a formal, explicit specification of a shared conceptualization [G93]. Here, **conceptualization** means an abstract model of similar domain knowledge that identifies the domain's relevant concepts. **Shared** means that the ontology captures knowledge that is accepted by a group. **Explicit** means that the concepts in the ontology and their limitations should be defined explicitly. Lastly, **formal** means that the ontology should be machine readable.

Examples of typical ontologies include web taxonomies (e.g., Yahoo! categories), catalogs for online shopping (e.g., Amazon's product catalog), and domain specific standard terminology (e.g., Gene ontology [A+00]). An ontology can be realized using several languages and some of them are listed as follows.

1. **RDF** and **RDFS** [MM+04]: Resource Description Framework is a data model developed by W3C for describing web resources. RDF allows for the specification of the semantics of data in a standardized, interoperable manner. In RDF, a pair of resources is connected by a property that forms a statement: resource, property, value. RDF Schema (RDFS) [BG+04] is a language for defining vocabularies of RDF in terms of rdfs:class, rdfs:property, rdfs:domain, and rdfs:range. RDFS defines the semantic relationships among resources and their respective properties.

2. **DAML+OIL** [CH+01]: DARPA Agent Markup Language-Ontology Interface Language, is a web-based ontology language developed over RDFS. It has a layered architecture
with XML like syntax.

3. *OWL* [SF+04]: Ontology Web Language, is a semantic markup language for publishing and sharing ontologies on the Web. It is extended version of RDF and is lightly influenced from DAML+OIL.

Among all these ontology languages, we are interested in OWL for its particular role in schema integration and the semantic web [DM+00]. The reason for picking OWL over these other languages is that in OWL, the mapping between relational schemas and ontologies can be realized more effectively using available APIs such as OWL API and Jena API. We used these APIs to parse an ontology created in OWL for the purpose of this thesis.

### 2.5 Integration of Relational Schema Using Ontologies

Ontologies have been used in relational database schema integration methods because they provide an explicit and machine understandable conceptualization of a domain. Wache et al. [WV+01] proposes a classification of ontology-based integration methodologies as shown in Figure 2-22-2. We will discuss these approaches briefly as follows.

1. *Global ontology approach*: a unified ontology represents data from all the sources, and a query is executed over this global ontology as shown in Figure 2-22-2(a). This approach requires a domain expert who knows the semantics of all the underlying data sources to define the global ontology.

2. *Multiple ontology approach*: every data source is represented by its own local ontology. Mapping between the ontologies has to be established as shown in Figure 2-22-2(b). A query on integrated data is expressed over a local ontology, and the mappings are used to perform local queries on other local ontologies. There is no need to integrate data in a global ontology. Changes to local ontologies may or may not affect the mapping.

3. *Hybrid ontology approach*: a local ontology describes data in each source, and a shared vocabulary is built for communicating among local ontologies as shown in Figure 2-22-2(c). This approach takes the advantages of the first two approaches: ease of determining ontologies locally and querying using a shared vocabulary.
Of the three approaches discussed above, we select the multiple ontology approach, as it is convenient to create individual ontologies from respective schemas using direct mappings from a schema to ontological concepts and vice versa. We discuss more about these mappings in the coming chapters.

In the next section we examine major methods that use ontologies as the basis for relational schema integration, and we compare our approach with them based on different factors.

### 2.6 Approaches for Schema Integration Using Ontologies

There are a number of approaches for schema integration based on ontologies. Ontologies can be used in various phases of the integration process. They can be used for identification and association of semantically corresponding information concepts, which is crucial in mapping discoveries. We discuss and compare four different approaches that use ontologies for schema integration and compare our work with them.

Linkova et al. [L07] propose an integration system based on ontologies of the data sources.
They use an ontology for mapping relational schema descriptions that creates not only the opportunity to capture different kinds of conceptual correspondences, but it also creates an opportunity to reuse the ontologies in other tasks or situations. In this approach local ontologies are merged to obtain a unified ontology, similar to our approach. The expressions are built by merging global and local ontologies and all relationships inside them. The ontology merging technique used in this thesis is motivated from Linkova’s approach. However they used instance level matchers for relational schema integration but they did not provided a software tool to realize their approach.

Gagnon et al. [G07] describe an ontology-based data integration system that consists of building a global ontology from the local ontologies corresponding to the data sources. Their data integration system architecture constitutes a virtual database as opposed to a materialized data store. Their mediator maps the requests and answers between the global ontology/schema and the local ontologies with their associated source schemas. The contribution of this method is that it does not require committing to a global ontology that makes the ontology matching less time-consuming. But there is no implementation for this approach.

Duo et al. [DQ+10] propose a heterogeneous database integration tool OntoGrate that queries multiple relational databases spread behind the ontology framework as if they are integrated. OntoGrate architecture combines ontology-based schema representation, first order logic inference, and SQL wrappers to integrate two sample relational databases. They have used inferential data integration as the theoretical framework for realizing their approach. To model database schemas, concepts, and the relationships (mappings) among them, they use the Web-PDDL ontology language, a first order logic language for the OntoEngine inference engine. The use of SQL wrappers for accessing real database systems makes their approach very practical. While OntoGrate system is promising for logical integration of relational databases, it does not provide a mechanism for physically integrating the relational database into a single database.

Table 2-11 compares these approaches with our approach on the following features: schema integration approach, ontology approach, ontology language, and whether a software tool is provided to implement the approach. Our approach is the only schema-level matching approach that we are aware of and it is one of two that provides a software implementation of the approach.
<table>
<thead>
<tr>
<th>Features/Papers</th>
<th>[L07]</th>
<th>[G07]</th>
<th>[DQ+10]</th>
<th>Our Research</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema integration approach</td>
<td>Instance</td>
<td>Hybrid</td>
<td>Hybrid</td>
<td>Schema</td>
</tr>
<tr>
<td>ontology approach</td>
<td>Multiple</td>
<td>Hybrid</td>
<td>Multiple</td>
<td>Multiple</td>
</tr>
<tr>
<td>ontology language</td>
<td>OWL</td>
<td>OWL</td>
<td>Web-PDDL</td>
<td>OWL</td>
</tr>
<tr>
<td>automated tool</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>type of integration</td>
<td>Physical</td>
<td>Logical</td>
<td>Logical</td>
<td>Physical</td>
</tr>
</tbody>
</table>

Table 2-1: Comparison Table

2.7 Conclusion

This chapter gives a brief summary of relational schema integration and an overview of four approaches that perform schema integration using ontologies. A feature summary contrasting approaches that perform schema integration using ontologies is given. The next chapter introduces our ontology generation approach that provides general mappings for describing relational schemas and illustrates the step-by-step mappings needed to transform a relational schema to its respective ontology.
Chapter 3  Relational Database Schema to Ontology Transformation

This chapter outlines our procedure for transforming a relational database schema into an ontology that is used as an initial step for our schema integration methodology. Ontologies are created through a set of transformations (or mappings) on a relational database schema, and are written in the Ontology Web Language (OWL).

Currently, there are many tools and techniques to perform a relational schema to ontology mapping. They can be broadly categorized into two main categories: approaches for creating a new ontology from a relational database schema and approaches for mapping a relational database schema to an already existing ontology as seen in Figure 3-13-1. Both the approaches are discussed in more detail below.

1. Creating ontology from the relational schema: this type of approach creates an ontology from a relational schema and transfers database contents to the newly created ontology. The ontology is created using the pre-defined mappings between ontological elements (e.g., class or property) and its corresponding database elements (e.g., table or column). In this type of approach, the relational schema and the generated ontology are analogous. The creation of an ontology structure is straightforward, involving direct transformations of relational schema tables to an ontology class and columns into properties. The additional semantic relations between database components such as referential constraints or foreign keys are mapped as unique properties while creating a corresponding ontology.

2. Mapping a relational schema to an existing ontology: this approach assumes that an ontology already exists. The goal is to create mappings between them and/or populate the ontology with the database contents. Mappings in this case can be more complex than those in the first approach discussed above because of different levels of overlap between the database domain and the ontology. These domains do not always coincide because the modeling criteria used for creating databases is different from those used for designing ontology models [BC+04] as one of them is richer, more generic or specific, or better structured than the other.
Both mapping approaches include two processes: (1) mapping definition, i.e., the transformation of a database schema into an ontology structure, and (2) data migration, i.e., the migration of database contents into ontology instances. The migration of database instances into ontological instances is not in the scope of this thesis.

We have selected a methodology to create an ontology from a relational schema using a mappings for definition approach; i.e., the transformation of the database schema into an ontology structure. In the following section, we review approaches used for generating an ontology from a relational schema.

### 3.1 Approaches for Generating an Ontology from a Relational Schema

Several methods have been proposed for generating ontologies from relational schemas. The method described by Stojanovic et al. [SS+02, SV+02] uses Frame-Logics [KL+95] to create an ontology (semi-automatically) from relational databases. The main disadvantage of this method is that it requires additional inputs from user about the schema while obtaining foreign key relationships. These inputs may vary from determining the foreign keys, the table and column they are pointing to, and their datatypes. In general, a user has to decide when to apply mapping rules and on which table and foreign keys. Astrova [A04] considers a similar approach where the focus is to examine key, data, and correlations-combinations of attributes. Fraternali [F99] considers fully automatic approaches and a simple reverse engineering approach where relations are mapped as classes, attributes in the relations are mapped to attributes in classes, and tuples in a relational database are mapped as instances in the ontology. Benslimane et al. [BS+06] considers a different
approach, where an OWL ontology is created corresponding to the content of a relational database based on analysis of their HTML forms. Semantics are extracted by analyzing HTML forms, and then these semantics are used to restructure and enrich the relational database schema. OWL ontologies are constructed through a set of transformation rules from the enriched schema. Barrasa et al. [BC+03, BC+04] map an existing database to an appropriate existing ontology implemented in RDFS or OWL. This approach uses the database and the ontology as they are and defines a declarative specification of the mappings between their modeling components.

Most of the approaches are based on direct mapping using mapping rules. Table 3-1 shows the common mapping strategies used in the existing approaches.

<table>
<thead>
<tr>
<th>RDB Schema</th>
<th>OWL Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>named class</td>
</tr>
<tr>
<td>foreign key column</td>
<td>object property</td>
</tr>
<tr>
<td>non-foreign key column</td>
<td>data property and a XSD data type</td>
</tr>
</tbody>
</table>

Table 3-1: Common Mapping Strategies

These approaches have one or more of the following drawbacks:

1. They do not discover inheritance, thus producing an ontology that is flat and does not capture richer semantics; a generated ontology has the same flat structure as the original relational database.

2. They do not discover restrictions or symmetric and transitive properties.

3. They ignore constraints that capture additional semantics.

4. They are not implemented.

5. They require extensive user interaction.

We overcome these drawbacks by using deep mappings between the relational schema to an ontology and realizing these mapping using a semi-automated tool.
3.2 Our Approach to Map a Relational Schema to an Ontology

This section describes a semi-automated, step-by-step approach for relational database schema-to-ontology mapping proposed by Juric et al. [JS+07]. An Employee schema shown in Figure 3-23-2 is used as an example to demonstrate the mapping technique.

According to Juric et al. [JS+07], primary keys and attributes in a relational schema are vital information for mapping a relational schema into an ontology. The mappings in Table 3-2 [JS+07] show a correspondence of elements from a relational database schema and an OWL ontology. There might be some scenarios, such as inheritance, where it is hard to find efficient mappings between a schema and an ontology. In those scenarios, Juric et al. suggest using a data analysis approach to find mappings. For example, if we analyze the dnumber attribute in the DEPT_LOCATIONS and DEPARTMENT relations, we find that both the attributes have similar data and hence we can combine those two relations into one, or we can mark them as equal classes in an ontology. The data analysis is out of scope for the research done in this thesis as we are only focusing on schema.
Table 3-2 Correspondence between Relational Model and OWL Features [JS+07]

The mapping rules suggested by Juric et al. [JS+07] are discussed below. We adopt this approach to generate an ontology from a relational schema for this thesis.

1. **Mapping relations**: according to this rule, relations in a relational database schema are similar to the concept of a class in an ontology, or owl:Class in OWL. Relations are mapped as a class in an OWL ontology. Following this step, an OWL ontology can be populated with basic class structures that contain all the classes, i.e., every relation from the relational schema will have a counterpart in the ontology as a class.

2. **Mapping attributes**: all attributes from the relation R in the relational schema are mapped into an equivalent OWL datatype property (owl:Datatype Property). The datatype property here will be attribute x from relation R. Datatype properties (owl:Datatype Property) are a subclass of the property class (owl:Property) in OWL. Datatype properties are used to identify the property whose value is associated with a data type. Values of owl:Datatype Property are literals. A datatype property defines a relation between instances in a class and values that belong to the datatype. A datatype can be a simple XML datatype or it can be a string. A subset of these predefined XML datatypes is what OWL uses. The domain and range of any datatype property must be defined inside the owl:datatype property. The domain corresponds to pre-defined classes that were derived by Rule 1. That implies that classes can be mentioned in the domain for which they are applicable. Range can be determined from Table 3-3.
<table>
<thead>
<tr>
<th>SQL data type</th>
<th>OWL Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALLINT</td>
<td>Short</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>unsignedShort</td>
</tr>
<tr>
<td>INT(11)</td>
<td>Integer</td>
</tr>
<tr>
<td>INT(11)</td>
<td>positiveInteger</td>
</tr>
<tr>
<td>INT(11)</td>
<td>negativeInteger</td>
</tr>
<tr>
<td>INT(11)</td>
<td>nonPositiveInteger</td>
</tr>
<tr>
<td>INT(11)</td>
<td>nonNegativeInteger</td>
</tr>
<tr>
<td>INT(11)</td>
<td>Int</td>
</tr>
<tr>
<td>INT(11)</td>
<td>unsignedInt</td>
</tr>
<tr>
<td>INT(20)</td>
<td>Long</td>
</tr>
<tr>
<td>INT(20)</td>
<td>unsignedLong</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>Decimal</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Float</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>Double</td>
</tr>
<tr>
<td>VARCHAR(45)</td>
<td>String</td>
</tr>
<tr>
<td>VARCHAR(45)</td>
<td>Language</td>
</tr>
<tr>
<td>VARCHAR(45)</td>
<td>Name</td>
</tr>
<tr>
<td>TIME</td>
<td>Time</td>
</tr>
<tr>
<td>DATE</td>
<td>Date</td>
</tr>
<tr>
<td>DATE</td>
<td>Datetime</td>
</tr>
<tr>
<td>DATE</td>
<td>gYearMonth</td>
</tr>
<tr>
<td>DATE</td>
<td>gMonthDay</td>
</tr>
<tr>
<td>DATE</td>
<td>gDay</td>
</tr>
<tr>
<td>DATE</td>
<td>gMonth</td>
</tr>
<tr>
<td>BIT</td>
<td>Boolean</td>
</tr>
<tr>
<td>BIT VARYING</td>
<td>unsignedByte</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>hexBinary</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>anyURI</td>
</tr>
</tbody>
</table>

Table 3-3: Correspondence between Relational Model and OWL Datatypes [JS+07]

Next, the ontology is populated with datatype properties. These properties represent attributes of relations in a relational schema. Each attribute converts to a datatype property having the same name. For example, an Fname attribute from the Employee relation is mapped as a datatype property having a domain of EMPLOYEE and a range as string, as shown in Figure 3-3.
Mapping constraints: constraints are the conditions applied to the attributes of a relation. The different possible constraints are UNIQUE, NOT NULL, PRIMARY KEY, REFERENCES, and FOREIGN KEY. The way each of them is handled when mapping into the OWL ontology is explained below.

a) Constraint UNIQUE: UNIQUE is a column constraint. It is mapped to an inverse functional property in OWL ontology. For example, in Figure 3-4, a UNIQUE constraint on column $Ssn$ implies that no two rows in table $EMPLOYEE$ can have the same value for column $Ssn$. This ensures that the $Ssn$ uniquely identifies an employee. Therefore, this constraint is mapped to an inverse functional property.

b) Constraint NOT NULL: constraint NOT NULL is a column constraint. It is mapped to a minimum cardinality of 1 in an OWL ontology. For example, in Figure 3-5, a NOT NULL constraint on column $EMP\_ID$ in table $EMPLOYEE$ specifies that the value of $EMP\_ID$ in each row cannot be null. This ensures that all employees are assigned an employee ID. Therefore, this constraint is mapped to a minimum cardinality of 1.
c) Constraint PRIMARY KEY: in a relational database, a primary key is an attribute or set of attributes that uniquely define the contents of each row. Because of its uniqueness property, a primary key is a constraint that disallows any two entities from having the same value for the key attributes at the same time. A primary key is also an attribute or a set of attributes and the entire mapping rules that apply to attributes also apply to primary keys, except for the fact that the datatype property that applies to a primary key should be declared as functional. Functional properties are single-valued properties.

For example, in Figure 3-6, a constraint PRIMARY KEY on column Ssn in relation EMPLOYEE specifies that Ssn is the primary key for this table. Hence, it is mapped as functional property in the OWL ontology.

```xml
<!-- Ssn -->
<owl:DatatypeProperty rdf:about="Ssn">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Employee</rdfs:comment>
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>
```

**Figure 3-6: Mapping Primary Key to Ontology**
d) **Constraints FOREIGN KEY:** relationships between tables in a relational schema are formed using foreign keys. A foreign key is an attribute (or a set of attributes) in one table that is a primary key in another table. The existence of a foreign key between two relations makes it evident that there is a relationship between them. A foreign key can be mapped to three different constructs in the ontology: an object property, class inheritance, and a symmetric property as part of this thesis. These different mappings are explained via examples below.

i. **FOREIGN KEY mapped to an object property:** In Figure 3-7, column Dno in relation EMPLOYEE is a foreign key to another relation DEPARTMENT, which indicates that a binary relationship (one-to-one or many-to-one) exists between both these relations. Since the foreign key is neither the primary key and nor is it a part of the primary key, it is mapped to an object property Dno that has OWL classes Employee and Department as its domain and range, respectively.

```xml
<!-- Dno -->
<owl:ObjectProperty rdf:about="Dno">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Dnumber</rdfs:comment>
  <rdfs:range rdf:resource="Department"/>
  <rdfs:domain rdf:resource="Employee"/>
</owl:ObjectProperty>
```

**Figure 3-7: Foreign Key Is Mapped to Object Property**

ii. **FOREIGN KEY mapped to class inheritance:** when a foreign key is part of the primary key as column EmSsn in relation Dependent is a foreign key to another relation Employee, indicating a binary relationship between these two relations. However, since the foreign key is a part of the primary key, this relationship is more significant than the one explained in case 1 above. Therefore, we create an object property EmSsn that has OWL classes Dependent and Employee as its domain and range respectively as similar to case 1 above and class Dependent will be added as subclass of Employee as shown in Figure 3-8.
iii. FOREIGN KEY mapped to symmetric property: in Figure 3-9, column Super_ssn in relation Employee is a foreign key to the same relation, which indicates a unary relationship. Therefore, the foreign key is mapped to a symmetric property Super_ssn that has OWL class Employee as both its domain and range (i.e., one employee is a supervisor of another employee).

Figure 3-9: Foreign Key Is Mapped to Symmetric Property

3.3 Conclusion

In this chapter, we describe rules for generating an OWL ontology from a relational schema for the purpose of relational schema integration. OWL features including concepts, properties, restrictions, and instances can be generated from a relational schema using these mappings. The mapping rules for concepts, properties, and restrictions illustrate the resemblance at a metadata level. We develop a tool that implements these mapping rules to convert a relational schema into an OWL ontology. The examples shown in this chapter are generated using the tool. In the next chapter, we discuss ontology merging techniques to transform individual ontologies into a merged single ontology that can be later converted to a relational schema.
Chapter 4  Ontology Merging

This chapter discusses ontology merging algorithms and also describes the approach that is used for our research. Ontology merging in general is a method of building a new ontology from two or more ontologies that have common features. In our approach, two ontologies are united together to obtain a union ontology that contains all the elements from both ontologies and then both the ontologies are aligned to get common terms. Ontology Alignment is the procedure of finding correspondences among the concepts of both ontologies. These common terms and intermediate union ontology together make the final merged ontology. We select the Alignment API [DJ+11] and the Optima API [KD+08] for finding the alignments between the two ontologies. Both APIs provide a semi-automatic approach to align two ontologies. The ontologies used here are created from the relational schemas that are generated using our approach described in Chapter 3.

According to Ghidini et al. [GG+04], ontology merging creates a new single and coherent ontology from two or more existing source ontologies. Various forms of mismatches may occur as discussed by Klein et al. [KM+01]:

1. Language level mismatches or syntactic mismatches caused by using different ontology representation languages may result.
2. Ontology level mismatches or semantic mismatches such as synonyms, homonyms, and hyponyms may result.

Ontology merging systems have to identify these types of mismatches in order to resolve them when finding the similar elements in two ontologies. Doing this at a later stage is difficult and costly to achieve manually, especially in the case of large, complicated ontologies. Algorithms and tools for ontology merging based on various criteria, such as CHIMAERA [MF+00], PROMPT [NM+00], ONION [MW+02], FCA-Merge [SM+01], and GLUE [DM+03], have been developed. All of these tools are semi-automatic and require human intervention to determine the different types of mismatches because their algorithms do not use the semantics embedded within ontologies.

In the next section, we explain the general concepts behind the ontology merging process that have been used by many researchers as the basis of their approaches for ontology merging.
4.1 Ontology Merging Process

Ontology merging is a process that creates a new ontology representing the union of two source ontologies; it is intended to gather the knowledge contained in the two ontologies into a single ontology. In other words, the resulted merged ontology reflects the similarities and dissimilarities in the two source ontologies. According to Amrouch et al. [AS+12], the ontology merging process has three basic stages as shown in Figure 4-1.

1. importing the source ontologies (assumed sufficiently homogenous),
2. identifying related concepts, and
3. merging related concepts and copying dissimilar ones into the resulting ontology.

Figure 4-1: Ontology Merging Process

In the next section we discuss ontology merging approaches and tools that are published in literature.

4.2 Ontology Merging Tools/Algorithms

Several algorithms and tools for ontology merging, mapping, and alignment are proposed in the literature. Most of these tools are semi-automatic and designing fully automatic tools is typically a complex issue. In this section, we briefly describe some of the most prominent tools used for ontology merging.

1. PROMPT [NM+00] proposed by Noy et al. is an interactive ontology merging tool and a plugin under Protégé. It suggests a list of all possible merging actions (a to-do list). Then user selects the appropriate proposals out of these suggestions. Subsequently, PROMPT automatically merges the chosen pairs of concepts and if any conflicts occur after the merging PROMPT suggests their appropriate solutions. Finally, the user selects the most
suitable solution. The major demerit of this algorithm is that it requires intensive human intervention as each step of the algorithm is based on the user’s decisions. Hence, in cases of huge and complex ontologies, the merging becomes cumbersome.

2. *FCA-Merge* [SM+01] proposed by Stemme et al. is a method for semi-automatic ontology merging. The process is summarized as follows.

First, ontologies populated with instances are derived from a set of input documents. Once the instances are derived, and the concept lattice is created, FCA-techniques are utilized to create the context of ontologies. FCA-techniques use lexical analysis to retrieve precise information that merges a word or expression to the concept if it has a related concept in another ontology. Then the two formal contexts are merged to produce the cropped concept lattice. Herein, the user may finally interfere to resolve disputes and eliminate duplications using his background about the domain. The major demerit of FCA-Merge is that it uses instances to classify same concepts; however, in other applications, no objects exist that are instances in both source ontologies at the same time.

3. *Chimaera* [MF+00] proposed by McGuiness et al. is an interactive ontology merging tool. Chimaera examines the source ontologies, performs automatic merging if it finds linguistic matches; otherwise, the user is asked for further action. Like PROMPT, Chimaera is a plugin under Protégé, namely Ontolingua, but they differ in the recommendations they make to their users with about the merging steps. However, the major drawback of this algorithm is the requirement of human intervention at every step, similar to PROMPT.

4. *Glue* [DM+03] is proposed by Doan et al. to find mappings between two source ontologies, say O1 and O2. For each concept of ontology O1, Glue finds its most similar concept in ontology O2 based on various practical similarity criteria and several machine learning strategies. The authors also use a method called relaxation labeling in order to map the hierarchies from the two ontologies. This method assigns a label to every node of a graph and uses a collection of domain independent constraints. The major drawback of this algorithm is that it is based on instances to determine similar concepts, where in most applications there are no objects that are instances in both ontologies.
5. **ONION** [MW+02] proposed by Mitra et al. is an ontology composition system that presents a mechanism for solving mismatches between distinct ontologies. The rules in the articulation generator define the relationship among two or more concepts relating to the ontologies. Manual establishment of these rules is an expensive and laborious task. Full automation is not feasible due to inadequacies of natural language processing technology. The authors also elaborate on a match between related terms in ontology that give a coarse relatedness measure, and it is the responsibility of the human expert to refine it. In their system, after a human domain expert validates the suggested matches, a learning component relies on the user’s feedback to generate better matches in the future when matching similar ontologies. The major drawbacks of this algorithm are that the method of identifying similar concepts is not specific, and the manual generation of articulation rules between them is tedious.

6. **Alignment API** [DJ+11] is a JAVA API for finding alignments between the ontologies. The Alignment API is a set of abstractions that helps in accessing, expressing, and sharing ontology alignments. It assists in the development of tools for finding alignments using various matchers.

7. **Optima** [KD+08] is a general purpose tool, which aims at performing ontology alignment to automatically detects and matches related concepts between ontologies. Optima uses WordNet [MF+93] to find synonyms between the terms.

After reviewing major approaches for ontology merging, we conclude that semi-automated ontology merging approaches can be tedious to use but are easier to implement, whereas automated ontology merging approaches are complex to implement but are easier to use. We use the Alignment API and Optima for the purpose of aligning ontologies in this thesis because they are straightforward to use and can be easily integrated with the tool that we have created. Furthermore, they provide better accuracy on merging ontologies as a user can decide between the alignments offered by these APIs to get the refined result.

### 4.3 Alignment and OPTIMA APIs

Ontology alignment means finding the relationships between the concepts from different ontologies. That is, given two ontologies, each consisting of discrete entities (classes, properties, rules, predicates, etc.) find the relationships (e.g., equivalence) between these entities. The
alignments can be generated manually or by using ontology matchers and can be used for merging ontologies.

Alignment API uses several algorithms to find the alignments between the two ontologies:

1. *NameEqAlignment* simply compares the equality of class and property names and aligns those objects with the same name.
2. *EditDistNameAlignment* uses an edit distance between entity names. It thus has to build a matrix of distances and chooses the alignment from the distance values.
3. *SubsDistNameAlignment* computes a substring distance on the entity name.
4. *StrucSubsDistNameAlignment* computes a substring distance on the entity names and uses and aggregates this distance with the symmetric difference of properties in classes.
5. *SMOANameAlignment*, the String Matching for Ontology Alignment (SMOA) method utilizes a specialized string metric *smoaDistance* [SS+05] for ontology alignment.
6. *StringDistAlignment* computes the levenshtein edit distance or pairwise alignment score matrix for a set of strings.

Optima models the ontologies as graphs and focuses on the graph matching problem. Optima use both structural similarity between the ontology graphs and lexical similarity between the concept labels and instances to select the alignments. The WordNet software uses a string based similarity distance called *basicSynonymySimilarity* that measures the similarity of two terms based on their synonymic similarity.

The Alignment API is a tool for determining alignments between the ontologies. It can be easily integrated with other software tools that are based on ontology merging. Due to its simplistic design and rich API it is gaining popularity in the ontology merging domain [E12].

### 4.4 Conclusion

In this chapter, a general approach to ontology merging is given. We use the Alignment and Optima API tools, both semi-automatic algorithms for finding ontology alignments. We discuss strategies and methods that both APIs use to determine suitable alignments between two ontologies. The strategies and methods described for Alignment API are based on a general ontology knowledge model. The Optima API uses WordNet as a source of finding the synonym relations between the concepts of two ontologies. The Optima API is used as a default alignment method for the software developed as a part of this thesis. We have used alignment methods of Alignment API as an
alternative for the purpose of ontology merging in the tool we developed. In the next chapter, we discuss a methodology used to convert a merged ontology into a relational schema.
Chapter 5  Ontology to Relational Schema

This chapter outlines the procedure for transforming an ontology into a relational schema. Mapping rules are applied to create an equivalent relational schema from a single (merged) ontology. An ontology written in OWL is used to create a relational schema represented in SQL. A relational schema is created through a series of transformations on an ontology.

According to Astrova et al. [AN+07], when ontologies are translated to relational schemas, the translation mechanism should resolve the following issues:

1. *Data loss:* resulting transformations should adequately describe the original data.

2. *Loss of structure:* in some cases, the transformation from an ontology to a relational schema is not lossless. That means in transformation of an ontology to the relational schema not every construct in an ontology can be mapped to a relational schema.

3. *Focus on structures:* mechanisms should be provided for the mapping of structures, for example, classes in an ontology to relations in a relational schema.

4. *Focus on data:* data from the ontology should be mapped to a relational schema within the context of data types.

5.1 Approaches for Generating a Relational Schema from an Ontology

Mappings between a relational schema and an ontology [BC+04, KS+06, XZ+06, GL+04] have been proposed by many researchers. A general mapping generation strategy is shown in Figure 5-1. Defining a mapping between an ontology and a relational schema is different from transforming ontologies to relational databases. The difference is that mapping assumes the existence of both a relational schema and an ontology, and it produces a set of correspondences between the two.
A relational schema is generated from an ontology as shown in Figure 5-2. That is, the input to the transformation technique is an ontology and the output is a relational schema.

There are several approaches for transformation of ontologies to relational databases [VN+06, AK+07]. However, some of these approaches suffer from at least one of the following problems:

1. most of them are not implemented,
2. they are not fully automated (i.e., they require user interaction),
3. they ignore value restrictions that capture additional semantics, and
4. they do not analyze loss of semantics induced by the transformation. Instead, they assume that every construct from an ontology can be mapped to a relational schema.
To solve most of these problems, Astrova et al. [AI+07] propose a novel approach for transformation of an ontology to a relational schema. The mapping strategy that we used as part of this thesis is motivated from the mapping approach proposed by Astrova et al. The pre-requisite of these mapping rules is that an ontology is written in OWL [OWL04] and a relational schema is generated using SQL.

In the next section, we discuss basic strategies using a generic ontology model and a generic relational model.

5.2 Transformation of an Ontology to a Relational Schema

An ontology represents an implementation of an ontological model. This model specifies ontology constructs as classes, properties, data types, inheritance, restrictions, and other semantics, as shown in Figure 5-3. The ontological model shown in Figure 5-3 is a generalized model for an ontology because it shows that an ontology comprises of concepts that have names and definitions. A concept is made up of classes and properties (object or datatype). A class can have subclasses and superclasses whereas a property has a scope limited to a class for which it is defined. However, it is not required that the ontology includes all the constructs of an ontological model.

Similarly, a relational database schema represents an implementation of the relational model. This model specifies relational schema constructs such as tables, columns, data types, and
constraints, as shown in Figure 5-45-4. The relational model shown in Figure 5-45-4 is a generalized model for a relational schema, where it is shown that a schema is comprised of a table having a name. A table is made up of columns and constraints. A constraint exists in a table and can be identified as a primary key, foreign key, or both. Similar to an ontology, the relational schema also does not need to include all constructs of the relational model.

Figure 5-4: Basic Relational Model [AI+07]

Astrova et al. state that almost all ontologies are defined on an ontological model and a relational schema is defined on the relation model. The basic idea behind their approach for the conversion of an ontology to a relational schema is shown in Figure 5-5. Conversion of an ontology to a relational schema is based on the set of mapping rules. These rules specify how the constructs of the ontological model are to be mapped to the relational model. Mapping rules are then applied to an ontology (the source) to produce a relational database (the target). The mapping rules are defined over the ontology model level hence they are applicable to any ontology that conforms to the ontological model.
The mapping rules of Astrova et al. to transform an ontology into a relational schema are given below.

### 5.3 Mapping Rules

There are two types of properties in an ontology that are needed to be considered for transformation of an ontology into a relational schema: data type properties and object properties. The constructs of an ontology are mapped to a relational schema by applying the following rules.

**Rule 1**: A named class (including subclasses and association classes) maps to a table and is named after the class. The functional property of this class is converted as a primary key of this table. The primary key is specialized as follows:

1. A table that relates to an association class (i.e., a class that relates to other classes directly) in an ontology then its primary key is specialized as a combination of foreign keys to all of its related tables.

2. A table that relates to a subclass in an ontology specialized its primary key as a foreign key to its superclass table.

---

**Figure 5-5: Transformation of Ontologies to Relational Schemas**

The mapping rules of Astrova et al. to transform an ontology into a relational schema are given below.
Rule 2: If a property is data type property, then it is mapped to a column of the table. This column relates to the class, which is specified in the domain of the data type property. This column receives its name from the corresponding data type property’s name. The range of the corresponding data type property determines the column’s type. This range is converted from XSD to SQL as shown in Table 5-1.

Rule 3: If a property is an object property then this property corresponds to a foreign key in the table. This foreign key corresponds to the table determined as the object property’s domain. This key references the primary key in the table that relates to the class determined as the object property’s range. The foreign key and the object property have the same name.

Rule 4: If an object property’s domain class is subclass of any other class then a foreign key created for this object property and this foreign key will be the part of primary key of the table corresponding to the domain of this property. That means this foreign key will also be the primary key of the table.

Rule 5: If an object property is also a symmetric property then the foreign key corresponding to this object property will refer to the same relation.

Rule 6: An inverse functional property maps to a UNIQUE constraint on the corresponding column.

Rule 7: A required property maps to a NOT NULL constraint on the corresponding column.

5.4 Data Type Conversion

Most of the transformations of data type properties convert data types from XSD to SQL. Unlike SQL, OWL does not have any built-in data types. Instead, it uses XSD data types such as string, integer, float, boolean, time, and date. Table 5-1 shows how to convert data types from XSD to SQL. This conversion is simple for the XSD data types that directly correspond to SQL data types. For example, if an XSD data type is a string, then the corresponding SQL data type is CHARACTER VARYING. However, the conversion becomes a challenge for unsupported data types.
### Table 5-1: Data Type Conversion [AI+07]

<table>
<thead>
<tr>
<th>XSD data type</th>
<th>SQL data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>unsignedShort</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>Integer</td>
<td>INT(11)</td>
</tr>
<tr>
<td>positiveInteger</td>
<td>INT(11)</td>
</tr>
<tr>
<td>negativeInteger</td>
<td>INT(11)</td>
</tr>
<tr>
<td>nonPositiveInteger</td>
<td>INT(11)</td>
</tr>
<tr>
<td>nonNegativeInteger</td>
<td>INT(11)</td>
</tr>
<tr>
<td>Int</td>
<td>INT(11)</td>
</tr>
<tr>
<td>unsignedInt</td>
<td>INT(11)</td>
</tr>
<tr>
<td>Long</td>
<td>INT(20)</td>
</tr>
<tr>
<td>unsignedLong</td>
<td>INT(20)</td>
</tr>
<tr>
<td>Decimal</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>Float</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Double</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>String</td>
<td>VARCHAR(45)</td>
</tr>
<tr>
<td>Language</td>
<td>VARCHAR(45)</td>
</tr>
<tr>
<td>Name</td>
<td>VARCHAR(45)</td>
</tr>
<tr>
<td>Time</td>
<td>TIME</td>
</tr>
<tr>
<td>Date</td>
<td>DATE</td>
</tr>
<tr>
<td>Datetime</td>
<td>DATE</td>
</tr>
<tr>
<td>gYearMonth</td>
<td>DATE</td>
</tr>
<tr>
<td>gMonthDay</td>
<td>DATE</td>
</tr>
<tr>
<td>gDay</td>
<td>DATE</td>
</tr>
<tr>
<td>gMonth</td>
<td>DATE</td>
</tr>
<tr>
<td>Boolean</td>
<td>BIT</td>
</tr>
<tr>
<td>unsignedByte</td>
<td>BIT VARYING</td>
</tr>
<tr>
<td>hexBinary</td>
<td>CHARACTER VARYING</td>
</tr>
<tr>
<td>anyURI</td>
<td>CHARACTER VARYING</td>
</tr>
</tbody>
</table>

### 5.5 Conclusion

There are many existing methodologies for mapping or transforming an ontology to a relational schema. In this chapter, we discuss rules for generating a relational schema from an OWL ontology to achieve relational schema integration. The next chapter provides a case study to demonstrate transformation of a relational schema to an ontology, ontology merging, and relational schema extraction from the merged ontology using the techniques outlined in Chapters 3, 4, and 5. Together, the steps provide a novel approach to achieving relational schema integration.
Chapter 6  Schema Integration Methodology with a Case Study

In order to provide better understanding of the functionality of our schema integration methodology, we apply it to relational schemas that belong to similar domains. The objective of this case study is to demonstrate the software based on our methodology that can integrate relational schemas using ontologies in a semi-automatic fashion. Section 6.1 outlines the features of the software components of our tool. Section 6.2 explains the functionality of our system with two example schemas. Section 6.3 analyses different merged schemas that are generated by the tool and Section 6.4 summarizes the contribution. Appendix A contains a more detailed illustration of all the inputs and results of each phase.

6.1 Development Tools

The RIO (Relational schema Integration using Ontology) tool is developed using the Java development framework and uses OWL API, Jena API, Alignment API and Optima in the background for creating, reading, and matching ontologies. The mappings between schemas and ontologies are achieved using RIO. The tool uses the OWL API to create OWL ontologies, the Jena API to read the contents of an ontology, and the Alignment API and Optima to realize the matching between two or more ontologies. We have also used MySQL server and MySQL Workbench to create and visualize the schemas. The advantages and key features of these tools and APIs are:

1. MySQL server is one of the most used and widely available open-source relational database management systems. We use MySQL server to create our input schemas for the case study.

2. MySQL Workbench is a visual database design tool that integrates database design, SQL development, maintenance, and administration into a unified integrated development environment for the MySQL database system. We have used MySQL workbench to visualize the relational schema.

3. The OWL API is a Java API and reference implementation for creating, manipulating and serializing OWL Ontologies. We use the OWL API to create OWL ontologies from schema using the mappings discussed in Chapter 3.
4. Jena is a semantic web framework for Java that provides an API to read data from and write to RDF graphs. Jena provides support for OWL (Web Ontology Language). We use the Jena API to read OWL ontologies and use that information to create schemas.

5. The Alignment API is an implementation for expressing and sharing ontology alignments. Alignment API is a java API that comprises of several alignment methods to find the alignment between the two inputted ontologies.

6. Optima is a state of the art general-purpose tool that performs ontology alignment by automatically identifying and matching relevant concepts between ontologies. Optima uses WordNet to find semantics between concepts in an ontology.

We use both Alignment API and Optima API in RIO to create the alignments between the two ontologies. These alignments are used by RIO to create the merged ontology.

6.2 Overview of RIO

RIO is a Java based tool that has a user friendly UI developed using Java swing. Figure 6-16-1 shows the startup window of RIO where buttons initiate our schema integration approach. RIO integrates with MySQL DB and imports all the databases from the MySQL server. A user can create a relational schema in MySQL Workbench and can load that schema into RIO using the configuration window shown in Figure 6-26-2. To start the integration operation, a SQL server has to be connected to RIO using the database configuration window shown in Figure 6-26-2. A user needs to configure the database connection settings for the first time using RIO and these settings will be saved for future use. The “add connection” button adds the database server properties to RIO and these properties can be saved using the “save settings” button. A user can reload the saved database server settings using the “reload” button and RIO automatically connects to the database server that was previously configured. Deleting a server setting from the tool is also made possible using the “delete connection” button provided in the window.

Once the database server is configured the left panel of the tool shows all the imported schemas. A user can select multiple schemas from this panel and RIO uses the SQL DDL of the selected schemas for the conversion. A user can select a maximum of five schemas to convert to an OWL ontology. The selected schemas are then converted to ontologies using the “convert schema” button. If there is no schema selected then an appropriate error message is shown. A user has to
choose at least one schema to start the conversion. Once two or more ontologies are created the “merge” button is enabled. A user can merge the loaded ontologies using this button. Merging ontologies is combination of two operations: uniting and aligning. Initially, a union ontology is created using the loaded ontologies and then RIO assists a user to find the alignments to be added into the union ontology to create a merged ontology. Once ontologies are merged successfully, the resulting ontology can be converted to a relational schema using the “convert ontology” button. A resultant schema is created into the MySQL server and the user is notified by a success message.

Figure 6-1: RIO tool (main window)

Figure 6-2: RIO tool (database configuration window)
RIO also provides a logger window as shown in Figure 6-3-6-3 that displays the entire mapping and merging operations for the user to monitor. A user can review the ongoing operations from this window. If an error occurs while using the tool then it can be traced using this window. It helps the user to use and visualize the processing.

![Figure 6-3: RIO tool (Logger window)](image)

RIO is seamlessly integrated with ontology generating and parsing APIs and so it can also be used as a multipurpose tool that can do individual operations of creating an ontology from a relational schema and vice versa. The created ontologies and schema DDLS are shown in Appendix A. The next section illustrates the procedure to create an OWL ontology in Protégé 3.1.

6.3 Creating OWL Ontologies using RIO and Protégé

Creating an OWL ontology is done by selecting an appropriate schema from the schema panel of RIO and clicking the “convert schema” button. RIO then parses the SQL DDL of that particular schema and creates Java objects related to the tables and columns of the selected schema.
Once the objects are created, then RIO uses mapping rules (Chapter 3) and creates an OWL ontology using the OWL API. After an ontology is created successfully, RIO gives a success message as shown in Figure 6-46-4, else an error message is displayed. A user can also monitor the conversion using the logger window of the tool.

![Image](image.png)

**Figure 6-4: Ontology Created**

To visualize the created ontologies, a user can use the Protégé 4.1 platform. Clicking on the Protégé icon launches the screen shown in Figure 6-56-5. The created ontology can be opened using the options shown in Figure 6-56-5.
One of the objectives of this thesis is to map relational schemas to OWL ontologies. Two relational schemas, a *COMPANY* schema [RS+99] (Figure 6-66-6) and an *EMPLOYEE* schema [SD+09] (Figure 6-76-7) have been selected to illustrate our framework.

The *Company* schema shown in Figure 6-66-6 has six relations shown as 6 rectangles: *Employee*, *Department*, *Dept_Locations*, *Project*, *Works_On*, and *Dependent*. The attributes for each relation are shown inside the rectangles. Similarly, the *EMPLOYEE* schema shown in Figure 6-76-7 also has six relations: *employees*, *salaries*, *dept_manager*, *dept_emp*, *titles*, and *departments*. Primary keys can be identified by a key symbol shown in front of them while foreign keys are identified by a red diamond symbol in the schema. All other attributes in the table are shown by blue diamond symbol. The solid blue diamond means the given attribute is not null. The lines connecting the two relations show the foreign key link between them. A solid lines means when a primary key itself is a foreign key for example, *Pno* attribute of relation *Works_On* is primary key that is also a foreign key referencing to *Pnumber* attribute of relation *Project*. And, a dotted lines means any other attribute of a relation is a foreign key, for example, *Mgr_ssn* attribute of relation *Department* is a foreign key referencing to *Ssn* attribute of relation *Employee*. 

**Figure 6-5: Opening OWL Ontology in Protégé 4.1**
Figure 6-6: COMPANY Schema [RS+99]

Figure 6-7: EMPLOYEE Schema [SD+09]
Table 6-1 summarizes the mapping rules described in Chapter 3. In both of these schemas, each relation is converted to an OWL class by following Rule 1. Figure 6-86-8 illustrates the creation of an OWL class in Protégé 4.1.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Relational Model</th>
<th>OWL Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>table</td>
<td>OWL: class</td>
</tr>
<tr>
<td>2</td>
<td>attribute</td>
<td>OWL: DatatypeProperty</td>
</tr>
<tr>
<td>3</td>
<td>primary key</td>
<td>OWL: DatatypeProperty, OWL: Functional Property</td>
</tr>
<tr>
<td>4</td>
<td>foreign key is not part of primary key</td>
<td>OWL: ObjectProperty</td>
</tr>
<tr>
<td>5</td>
<td>foreign key is part of primary key</td>
<td>OWL: ObjectProperty, sub-class/super-class hierarchy is created rdf:subClassOf</td>
</tr>
<tr>
<td>6</td>
<td>foreign key referring to same relation</td>
<td>OWL: ObjectProperty, OWL: SymmetricProperty</td>
</tr>
<tr>
<td>7</td>
<td>foreign key is unique</td>
<td>OWL: ObjectProperty, OWL: InverseFunctionalProperty</td>
</tr>
<tr>
<td>8</td>
<td>attribute is not null</td>
<td>OWL: DatatypeProperty, OWL: Restriction</td>
</tr>
</tbody>
</table>

**Table 6-1: Mapping Relational Schema to Ontology**

Figure 6-86-8 shows that classes *Employee, Department, Dept_Locations, Project, Works_On, and Dependent* OWL are created for corresponding relations in the *COMPANY* schema. Similarly, classes *employees, salaries, dept_manager, dept_emp, titles*, and *departments* are created for corresponding relations in the *EMPLOYEE* schema. The *Employee* class is a subclass of *owl:Thing*, which is the superclass of all classes. Initially, each newly created OWL class is a subclass of *owl:Thing* and is a sibling to every other OWL class. Next, a hierarchy is built among these classes depending on the relations between their attributes. RIO tools build this hierarchy using the mappings from Table 6-1.
Attributes of relations correspond to an *owl:DatatypeProperty* in OWL ontology, as shown in Rule 2 in Table 6-1. Figure 6-9 shows the mapping of attribute *Pname* from relation *Project* to *owl:DatatypeProperty* that has domain *Project* and range *xsd:string*.

Using Rule 3 from Table 6-1, an OWL property marked as *Functional* corresponds to a primary key in a relational schema. A constraint primary key on column *Ssn* in relation *Employee*
specifies that \textit{Ssn} is the primary key for this table. Hence, it is mapped as a functional property in the OWL ontology shown in Figure 6-106-10. \textit{Ssn} is a \textit{owl:DatatypeProperty} that has domain \textit{Employee} and range \textit{xsd:integer}.

![Figure 6-10: Mapped Primary Key](image)

Relations are linked to each other using foreign keys. A foreign key in one relation refers to a primary key in another relation. This can be explained using an example. Relations \textit{Project} and \textit{Department} of the \textit{COMPANY} schema are shown in Figure 6-66-6. The \textit{Dnum} attribute in the \textit{Project} relation refers to the \textit{Dnumber} attribute in the \textit{Department} relation. The \textit{Dnumber} attribute is the primary key of the \textit{Department} relation. The \textit{Dnum} attribute in the \textit{Project} relation is a foreign key that is not part of primary key. As shown in Rule 4 from Table 6-1, foreign key is not a part of the primary key, the \textit{Dnum} attribute is mapped to an object property \textit{Dnum} that has OWL classes \textit{Project} and \textit{Department} as its domain and range, respectively, as shown in Figure 6-116-11.
Figure 6-11: Foreign Key is not part of Primary Key

There can be a scenario where an attribute in a relation is a primary key but also refers to some other relation as a foreign key. For example, the attribute Essn in relation Works_On is a primary key attribute but also refers to another relation Employee as a foreign key. Using Rule 5 of Table 6-1, foreign key is part of the primary key, the Essn attribute of relation Works_On is mapped as an owl:ObjectProperty with Works_On and Employee as its domain and range as shown in Figure 6-12. Also, it gives the scope of having inheritance within the OWL ontology.

Inheritance is defined using the subclass/superclass relationship in an OWL ontology. Foreign keys play a significant role to define the hierarchical relationship between classes. When an attribute in a relation is a primary key and refers to the primary key of another relation, then the former relation is treated as a subclass of the latter one. For example, the relation Works_On in Employee schema has composite primary key, i.e. Essn and Pno. Essn is a foreign key to Employee whereas Pno is a foreign key to Project. Referring to Rule 5 of Table 6-1, Works_On is a subclass of Employee and Project as shown in Figure 6-8. Similarly, Dept_Locations is subclass of Department and Dependent is subclass of Employee.
OWL property restrictions consist of two groups, value constraints and cardinality constraints. Value constraints are further subdivided into three specific restrictions, which are owl:allValuesFrom, owl:someValuesFrom, owl:hasValue. Cardinality constraints consist of owl:cardinality, owl:minCardinality, owl:maxCardinality. Since attribute Fname from relation Employee is not null then according to Rule 6 of Table 6-1, the restriction owl:minCardinality that has a value of 1 is created for the Fname datatype property and is given domain as Employee and range as xsd:string.

There can be a scenario where an attribute refers to its own relation/table, i.e., an attribute is a foreign key to its own relation. For example, the attribute Super_ssn in the relation Employee of the
COMPANY schema is a foreign key to the same relation Employee, which indicates a unary relationship. Therefore, according to Rule 7 of Table 6-1, the attribute Super_ssn is mapped as an owl:ObjectProperty and owl:SymmetricProperty that has OWL class Employee as both its domain and range as shown in Figure 6-14-14.

![Figure 6-14: Foreign Key Mapped to Inverse Functional and Symmetric Property](image)

When an attribute in a relation is foreign key and is also marked with a constraint unique key then according to Rule 8 of Table 6-1, it maps to an owl:ObjectProperty and owl:InverseFunctionalProperty. Hence, attribute Super_ssn in the relation Employee is also marked as owl:InverseFunctionalProperty in Figure 6-14-14.

Each relation and their attributes in the COMPANY and EMPLOYEE schemas are mapped into their respective OWL ontologies using the set of rules shown in Table 6-1 derived from Chapter 3. Relations are mapped as an owl:class whereas attributes are mapped to an owl:property. After being mapped as owl:class, classes were arranged in a hierarchy of superclass and subclass relationships. After mapping all the relations and their attributes, the respective OWL ontologies were created. As a result, the COMPANY schema is converted into the COMPANY ontology as shown in Figure 6-156-15 and the EMPLOYEE schema is converted into the EMPLOYEE ontology as shown in Figure 6-166-16. Form the figures it can be inferred that the entities that are shown as rectangles are all OWL classes where solid purple lines shows inheritance between these classes and dotted lines show foreign key relationships. The detailed OWL ontologies i.e. COMPANY.owl and EMPLOYEE.owl and their respective SQL DDLs are given in Appendix A.
Now that these ontologies have been created from their respective schemas, the next section describes how these two ontologies can be unified and aligned into a merged ontology.
6.3 Merging Ontologies Using RIO and Alignment APIs

Merging ontologies using RIO includes two operations: unification and alignment. First, RIO loads the individual ontologies into one, which means all the classes and properties of the ontologies are inserted into a larger intermediate ontology. Once this ontology is created then RIO adds the alignments into this ontology to create the final merged ontology. If there are more than two ontologies to be merged, RIO merges the first two ontologies and then it merges the result with the next individual ontology. RIO uses the OPTIMA API to suggest potential alignments based on the ontologies. The two ontologies that are created in the previous section for the COMPANY and EMPLOYEE schemas are used as an example to illustrate the merging process.

To start the merging operation in RIO, the user clicks the “merge” button shown in Figure 6-17-17. RIO then unifies the loaded ontologies into an intermediate ontology and subsequently pops up the window that displays the available alignment methods. The OptimaAlignment method is the default and it uses the OPTIMA API in the background. Alignment API offers the other alignment methods shown in Figure 6-17-17. RIO always gives OptimaAlignment suggestions irrespective of the chosen alignment method as it includes WordNet to give semantic alignments. In this case study, we have chosen the SMOANameAlignment method as another example alignment technique.
After selecting an alignment method, RIO asks for the trimming threshold as shown in Figure 6-18 that is used for the alignment operation. The default value for the trimming threshold is set to 0.5 and a user can modify it in the range of 0-1. RIO then initiates a background job that finds the OptimaAlignment using the OPTIMA API and SMOANameAlignment using the alignment API between the two ontologies. The original ontologies are left untouched.

The first set of alignments is discovered using the COMPANY ontology as a source and the EMPLOYEE ontology as a target using the default OptimaAlignment method as shown in Figure 6-19. The alignment Employee=employees:1 is one of the suggestions made by RIO; these two axioms are equivalent with similarity match of 1. “Axiom” is the generalized term for ontology content such as classes or properties. A user can choose multiple appropriate alignments from the window and RIO filters them accordingly. If the source and target ontologies are swapped, it is often possible to get a different set of alignments. A second set of alignments are discovered using the EMPLOYEE ontology as a source and the COMPANY ontology as a target using the OptimaAlignment method as shown in Figure 6-20. Similarly, third and fourth set of alignments are discovered using the SMOANameAlignment method shown in Figure 6-21 and Figure 6-22.

A user can experiment with different alignment methods and threshold to find the best possible set of alignments. Once a good set of alignments are found RIO modifies the intermediate union ontology with these alignments and a merged ontology is created.
Figure 6-19: Optima suggestions (COMPANY-EMPLOYEE)

Figure 6-20: Optima suggestions (EMPLOYEE- COMPANY)
As a result of the merge operation using the alignments we have chosen in this case study, the classes of the merged ontology are shown in Figure 6-23. Alignments are shown with an equal sign, for example “Department = departments.” Similarly, the properties are also aligned as shown in Figure 6-24 and Figure 6-25.

Figure 6-21: SMOANameAlignment suggestions (COMPANY-EMPLOYEE)

Figure 6-22: SMOANameAlignment suggestions (EMPLOYEE- COMPANY)
Figure 6-23: Resultant Classes in Merged Ontology

Figure 6-24: Resultant Data Properties in Merged Ontology
Finally, a merged ontology is created as shown in Figure 6-26. The detailed OWL format of the merged ontology is given in Appendix A.

Figure 6-25: Resultant Object Properties in Merged Ontology

Now that a merged ontology has been created from each of the individual source ontologies, the next section will describe how this merged ontology is transformed to a relational schema.

Figure 6-26: Merged Ontology
6.4 Creating Relational Schema from an Ontology

In the last phase of our approach, we use RIO to transform the merged OWL ontology (created in the previous section) to a relational schema. Figure 6-27 shows the success message displayed after the creation of a schema using the merged COMPANY and EMPLOYEE ontology (called MERGED) as an input. Table 6-2 shows the list of mappings that RIO uses to create a relational schema from the merged ontology.

Figure 6-27: Merged Schema Created Window

After merging the ontologies, a user clicks the “convert ontology” button to start the conversion operation. RIO then parses the MERGED ontology using Jena API and creates java objects related to the tables and columns of the resultant schema. Once the objects are created, RIO then uses the mappings defined in Table 6-2 to create the final relational schema.
An OWL ontology is converted to a relational schema by following the rules discussed in Chapter 5 and shown in Table 6-2. According to Rule 1, each class in the OWL ontology maps to a relation/table in the relational schema. For example, the Employee class in the MERGED ontology is mapped to a relation in the relational schema as shown in Figure 6-28. Following Rule 3, the owl:property marked as functional in an owl:class is mapped as a primary key of the corresponding relation. For example, Ssn and eno are owl:DatatypeProperty and are marked as functional in MERGED ontology. These two properties are mapped as a primary key for relation EMPLOYEE as shown in Figure 6-28.

Properties in OWL are divided into datatype and object property. A property always maps to an attribute in a relation. A constraint on that attribute is determined by the type of owl:property. Mapping an owl:DatatypeProperty is easier than mapping an owl:ObjectProperty as it directly maps to an attribute of the relation. This is in accordance with Rule 2 of Table 6-2. The data type of this attribute is determined using the Table 5.1. For example, Fname is an owl:DatatypeProperty that has domain as Employee class and hence, it is mapped to an attribute in the relation Employee as shown in Figure 6-28. Also, as Fname property has restriction on it then according to Rule 8 it is marked as not null shown with solid blue diamond in Figure 6-28.

To map an owl:objectProperty to a relational schema, Rules 4-7 of Table 6-2 are followed. For example, Dno is an object property and is mapped as a foreign key in the relation Employee and refers to the relation Department because Employee and Department are its domain and range,
respectively. Similarly, all object properties can be mapped to a relational schema using the rules in Table 6-2.

Once all classes and their properties have been mapped into a relational schema, the final schema obtained is shown in Figure 6-28-28.

![Figure 6-28: Resultant Merged Schema](image)

6.5 Evaluation of RIO

In this section we evaluate the complexity of the merged schemas generated by RIO using the alignment methods discussed in Chapter 4. Pavlic et al. [PK+08] propose a method to compute the complexity of a relational schema. They compute the complexity of a schema using basic elements of a schema such as attributes, primary keys, foreign keys, and indexes. The complexity $C$ of a given schema is expressed as the summation of the weights $\sum W$ of all the tables/relations exist in a given
Each table’s weight $W$ is computed as the summation of the number of attributes in a table ($A$), number of primary keys of a table ($K$), number of foreign keys in a table ($F$) and the number of indexes in a table ($I$), which is shown using the formula $W = A + K + F + I$. Using this complexity formula we have found complexity for both input schemas and our final merged schema generated by RIO using different alignment methods as shown in Table 6-33. The merged schemas are given in Appendix B.

The complexity of COMPANY schema shown in Figure 6-66-6 is calculated as follows. There are total 6 relations in COMPANY schema. In relation Employee, the number of attributes $A = 10$, the number of keys $K = 1$, the number of indexes $I = 4$, and the number of foreign keys $F = 2$; therefore the weight $W_{Employee} = 16$. Similarly, we can compute the weight of other relations as $W_{Department} = 8$, $W_{Project} = 8$, $W_{Dependent} = 10$, $W_{Dept Locations} = 6$, $W_{Works On} = 10$. Therefore, the complexity of COMPANY schema is 58. Similarly, the complexity of EMPLOYEE schema can be computed as 53.

<table>
<thead>
<tr>
<th>Alignment approach</th>
<th>Complexity of resultant schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>OptimaAlignment</td>
<td>103</td>
</tr>
<tr>
<td>EditDistNameAlignment</td>
<td>101</td>
</tr>
<tr>
<td>NameEqAlignment</td>
<td>103</td>
</tr>
<tr>
<td>SMOANameAlignment</td>
<td>101</td>
</tr>
<tr>
<td>StringDistAlignment</td>
<td>103</td>
</tr>
<tr>
<td>StrucSubsDistAlignment</td>
<td>103</td>
</tr>
<tr>
<td>SubsDistNameAlignment</td>
<td>103</td>
</tr>
</tbody>
</table>

Table 6-3: Complexity of Resultant Schemas

It can be observed in Table 6-33 that the complexity of the merged schema using different alignment methods is in the range of 101-103 for the EMPLOYEE and COMPANY schemas. The individual schemas have complexity values of 58 and 53, respectively, in which case the union of these schemas gives a complexity of 111. RIO reduces the complexity to 101 with EditDistNameAlignment and SMOANameAlignment as alignment methods. Combining the results of alignment methods and changing with the threshold value can reduce the complexity of the merged
schema even more. RIO does not support the combination of alignment methods at this time but can be extended as a future work.

6.6 Conclusion

In this chapter we discuss the features and functionality of RIO as a relational schema integration platform. We illustrate our relational schema integration approach on a small case study using RIO, and discuss the complexity of the generated merged schema. In the next chapter, contributions of this research and ideas for future work are discussed.
Chapter 7  Contributions and Future Work

In this chapter we give an overview of our research contributions and suggest future work that can be done to extend this research.

7.1 Contributions

In Chapter 2, we give an outline of relational schema integration and an overview of semantic web technologies such as ontologies and ontology languages. We survey approaches for relational schema integration using ontologies and compare them with our approach. In Chapter 3, we survey approaches for transforming a relational database schema into an ontology and select an approach for the purpose of this thesis. In Chapter 4, we discuss the ontology merging problem and review approaches to solve the problem. After surveying the literature, we have propose a unique approach that unifies the ontologies and then combines the unified ontology with the alignments found between them. We use the well-known Alignment API and Optima API to realize the alignments between ontologies. In Chapter 5, we survey approaches for transforming an OWL ontology to a relational database schema and select suitable mappings from the reviewed approaches.

In Chapter 6, we provide the implementation of our approach with the RIO (Relational schema Integration using Ontology) tool. We discuss and illustrate the functioning of RIO with a test case of relational schemas that belong to same domain. In summary, we have created a novel approach and software tool for integrating relational schemas using ontologies and representing the integrated components with a merged relational schema extracted from the merged ontology.

7.2 Future Work

The proposed methodology for integrating relational schemas using ontology merging techniques can be extended further as follows.

A. Our schema integration approach can be extended to include data from the schema as well. Our approach integrates the basic structure of the relational model, i.e., the schema, but to realize the integration of whole relational databases we need to include the data as well.

B. More semantics of a relational schema can be addressed for improving the integration methodology. Column attributes such as binary, unsigned, zerofill, and auto-increment can be addressed to create more robust ontology.
C. The schema integration approach we have proposed in this thesis can be scaled for merging n number of schemas related to the same or different domains.

D. Alignment API offers more alignment methods when using it through the Alignment server. Including more alignment methods may help to find more alignments between the ontologies and hence the final merged schema may be more refined.

E. Combining the output of different alignment methods of Alignment API and filter the results on each stage may produce a more refined merged ontology. This way all the alignment methods offered by Alignment API could be utilized.

The software tool RIO can be extended in the following areas.

A. RIO is a semi-automated schema integration tool. It can be extended to function as an automated tool by using machine learning approaches to automate alignments. The algorithm for finding the alignments between the ontologies could be improved to find the best possible alignments with less or no user input.

B. Currently, RIO only supports relational schemas from the MySQL database but it can be extended to include support for other relational database platform. Adding support for more databases in RIO facilitates integrating schemas from different databases such as MySQL and Oracle DB. For example, RIO could take schemas from both MySQL and Oracle DB and integrates them into either MySQL or Oracle DB.

C. Extending RIO to put more visualization of the intermediate ontologies using the graphical frameworks. Users can visualize the intermediate ontologies in the tool itself rather than switching to Protégé or other ontology editors.

7.3 Conclusion

In this chapter we discuss the contributions of our work as part of research done in this thesis. We give ideas for improvements and future work on our methodology and the tool developed for this thesis. We achieve the objective behind this research work by proposing a new methodology for integrating relational schema using ontology merging techniques and providing a software tool to realize it.
References


Appendix A  Relational Schema and OWL Ontology Generation Datasets

A.1  COMPANY Schema DDL

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

CREATE SCHEMA IF NOT EXISTS `COMPANY` DEFAULT CHARACTER
SET latin1 ;
USE `COMPANY` ;

--  Table `COMPANY`.`Employee`
CREATE TABLE IF NOT EXISTS `COMPANY`.`Employee` (
`Ssn` INT(11) NOT NULL,
`Fname` VARCHAR(45) NOT NULL,
`Minit` VARCHAR(45) NULL DEFAULT NULL,
`Lname` VARCHAR(45) NULL DEFAULT NULL,
`Bdate` DATE NULL DEFAULT NULL,
`Address` VARCHAR(45) NULL DEFAULT NULL,
`Sex` VARCHAR(45) NULL DEFAULT NULL,
`Salary` INT(11) NULL DEFAULT NULL,
`Super_ssn` INT(11) NULL DEFAULT NULL,
`Dno` INT(11) NULL DEFAULT NULL,
PRIMARY KEY (`Ssn`),
UNIQUE INDEX `Super_ssn_UNIQUE` (`Super_ssn` ASC),
INDEX `Dno_idx` (`Dno` ASC),
CONSTRAINT `Dno`
FOREIGN KEY (`Dno`)
REFERENCES `COMPANY`.`Department` (`Dnumber`)
ON DELETE NO ACTION
ON UPDATE NO ACTION,
CONSTRAINT `Super_ssn`
FOREIGN KEY (`Super_ssn`)
REFERENCES `COMPANY`.`Employee` (`Ssn`)
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

--  Table `COMPANY`.`Department`
CREATE TABLE IF NOT EXISTS `COMPANY`.`Department` (
'Dnumber' INT(11) NOT NULL,
'Dname' VARCHAR(45) NULL DEFAULT NULL,
'Mgr_ssn' INT(11) NULL DEFAULT NULL,
'Mgr_start_date' DATE NULL DEFAULT NULL,
PRIMARY KEY ('Dnumber'),
INDEX 'Mgr_ssn_idx' ('Mgr_ssn' ASC),
CONSTRAINT 'Mgr_ssn'
FOREIGN KEY ('Mgr_ssn')
REFERENCES `COMPANY`.'Employee' ('Ssn')
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- -- Table `COMPANY`.'Dependent'
CREATE TABLE IF NOT EXISTS `COMPANY`.'Dependent' (  
'EmSsn' INT(11) NOT NULL,
'Dependent_name' VARCHAR(45) NOT NULL,
'Dsex' VARCHAR(45) NULL DEFAULT NULL,
'DBdate' DATE NULL DEFAULT NULL,
'Relationship' VARCHAR(45) NULL DEFAULT NULL,
PRIMARY KEY ('EmSsn', 'Dependent_name'),
INDEX 'EmSsn_idx' ('EmSsn' ASC),
CONSTRAINT 'EmSsn'
FOREIGN KEY ('EmSsn')
REFERENCES `COMPANY`.'Employee' ('Ssn')
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- -- Table `COMPANY`.'Dept_Locations'
CREATE TABLE IF NOT EXISTS `COMPANY`.'Dept_Locations' (  
'Dnum' INT(11) NOT NULL,
'Dlocation' VARCHAR(45) NOT NULL,
PRIMARY KEY ('Dnum', 'Dlocation'),
CONSTRAINT 'Dnumber'
FOREIGN KEY ('Dnum')
REFERENCES `COMPANY`.'Department' ('Dnumber')
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;
CREATE TABLE IF NOT EXISTS `COMPANY`.`Project` (
    `Pnumber` INT(11) NOT NULL,
    `Pname` VARCHAR(45) NULL DEFAULT NULL,
    `Plocation` VARCHAR(45) NULL DEFAULT NULL,
    `Dnum` INT(11) NULL DEFAULT NULL,
    PRIMARY KEY (`Pnumber`),
    INDEX `Dnum_idx` (`Dnum` ASC),
    CONSTRAINT `Dnum`
        FOREIGN KEY (`Dnum`)
        REFERENCES `COMPANY`.`Department` (`Dnumber`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

CREATE TABLE IF NOT EXISTS `COMPANY`.`Works_On` (
    `Essn` INT(11) NOT NULL,
    `Pno` INT(11) NOT NULL,
    `Hours` VARCHAR(45) NULL DEFAULT NULL,
    PRIMARY KEY (`Essn`, `Pno`),
    INDEX `Essn_idx` (`Essn` ASC),
    INDEX `Pno_idx` (`Pno` ASC),
    CONSTRAINT `Essn`
        FOREIGN KEY (`Essn`)
        REFERENCES `COMPANY`.`Employee` (`Ssn`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
    CONSTRAINT `Pno`
        FOREIGN KEY (`Pno`)
        REFERENCES `COMPANY`.`Project` (`Pnumber`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
A.2 EMPLOYEE Schema DDL

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

CREATE SCHEMA IF NOT EXISTS `EMPLOYEE` DEFAULT CHARACTER SET latin1 ;
USE `EMPLOYEE` ;

-- Table `EMPLOYEE`.'departments'

CREATE TABLE IF NOT EXISTS `EMPLOYEE`.`departments` (
  `dno` INT(11) NOT NULL,
  `dept_name` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`dno`)) ENGINE = InnoDB DEFAULT CHARACTER SET = latin1;

-- Table `EMPLOYEE`.'employees'

CREATE TABLE IF NOT EXISTS `EMPLOYEE`.`employees` (
  `eno` INT(11) NOT NULL,
  `birth_date` DATE NULL DEFAULT NULL,
  `first_name` VARCHAR(45) NULL DEFAULT NULL,
  `last_name` VARCHAR(45) NULL DEFAULT NULL,
  `gender` VARCHAR(45) NULL DEFAULT NULL,
  `hire_date` DATE NULL DEFAULT NULL,
  PRIMARY KEY (`eno`)) ENGINE = InnoDB DEFAULT CHARACTER SET = latin1;

-- Table `EMPLOYEE`.'dept_emp'

CREATE TABLE IF NOT EXISTS `EMPLOYEE`.'dept_emp` (
  `emp_no` INT(11) NOT NULL,
  `dept_no` INT(11) NOT NULL,
  `from_date` DATE NULL DEFAULT NULL,
  `to_date` DATE NULL DEFAULT NULL,
  PRIMARY KEY (`emp_no`, `dept_no`),
  INDEX `FK3_idx` (`dept_no` ASC),
INDEX 'FK4_idx' ('emp_no' ASC),
CONSTRAINT 'FK3'
FOREIGN KEY ('dept_no')
REFERENCES 'EMPLOYEE'. 'departments' ('dno')
ON DELETE NO ACTION
ON UPDATE NO ACTION,
CONSTRAINT 'FK4'
FOREIGN KEY ('emp_no')
REFERENCES 'EMPLOYEE'. 'employees' ('eno')
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- Table 'EMPLOYEE'. 'dept_manager'

CREATE TABLE IF NOT EXISTS 'EMPLOYEE'. 'dept_manager' (  'dept_no' INT(11) NOT NULL,
 'emp_no' INT(11) NOT NULL,
 'from_date' DATE NULL DEFAULT NULL,
 'to_date' DATE NULL DEFAULT NULL,
 PRIMARY KEY ('dept_no', 'emp_no'),
 INDEX 'FK1_idx' ('dept_no' ASC),
 INDEX 'FK2_idx' ('emp_no' ASC),
 CONSTRAINT 'FK1'
 FOREIGN KEY ('dept_no')
 REFERENCES 'EMPLOYEE'. 'departments' ('dno')
 ON DELETE NO ACTION
 ON UPDATE NO ACTION,
 CONSTRAINT 'FK2'
 FOREIGN KEY ('emp_no')
 REFERENCES 'EMPLOYEE'. 'employees' ('eno')
 ON DELETE NO ACTION
 ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- Table 'EMPLOYEE'. 'salaries'

CREATE TABLE IF NOT EXISTS 'EMPLOYEE'. 'salaries' (  'emp_no' INT(11) NOT NULL,
 'salary' INT(11) NULL DEFAULT NULL,
'from_date' DATE NOT NULL,
'to_date' DATE NULL DEFAULT NULL,
PRIMARY KEY ('emp_no', 'from_date'),
INDEX `FK6_idx` ('emp_no' ASC),
CONSTRAINT 'FK6'
FOREIGN KEY ('emp_no')
REFERENCES `EMPLOYEE`.employees ('eno')
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;
-- Table 'EMPLOYEE`.`titles`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS 'EMPLOYEE`.titles' (  
'emp_no' INT(11) NOT NULL,
'title' VARCHAR(45) NOT NULL,
'from_date' DATE NOT NULL,
'to_date' DATE NULL DEFAULT NULL,
PRIMARY KEY ('emp_no', 'title', 'from_date'),
INDEX `emp_no_idx` ('emp_no' ASC),
CONSTRAINT 'FK5'
FOREIGN KEY ('emp_no')
REFERENCES `EMPLOYEE`.employees ('eno')
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

A.3 COMPANY OWL Ontology

<?xml version="1.0"?>
<rdf:RDF xmlns="COMPANY#"
   xml:base="COMPANY"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <owl:Ontology rdf:about="COMPANY"/>
</!--
// Object Properties

<!-- Dno -->
<owl:ObjectProperty rdf:about="Dno">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Dnumber</rdfs:comment>
  <rdfs:range rdf:resource="Department"/>
  <rdfs:domain rdf:resource="Employee"/>
</owl:ObjectProperty>

<!-- Dnum -->
<owl:ObjectProperty rdf:about="Dnum">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Dnumber</rdfs:comment>
  <rdfs:range rdf:resource="Department"/>
  <rdfs:domain rdf:resource="Dept_Locations"/>
  <rdfs:domain rdf:resource="Project"/>
</owl:ObjectProperty>

<!-- EmSsn -->
<owl:ObjectProperty rdf:about="EmSsn">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ssn</rdfs:comment>
  <rdfs:range rdf:resource="Employee"/>
  <rdfs:domain rdf:resource="Dependent"/>
</owl:ObjectProperty>

<!-- Essn -->
<owl:ObjectProperty rdf:about="Essn">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ssn</rdfs:comment>
  <rdfs:range rdf:resource="Employee"/>
  <rdfs:domain rdf:resource="Works_On"/>
</owl:ObjectProperty>

<!-- Mgr_ssn -->
<owl:ObjectProperty rdf:about="Mgr_ssn">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ssn</rdfs:comment>
</owl:ObjectProperty>
<rdfs:domain rdf:resource="Department"/>
<rdfs:range rdf:resource="Employee"/>
</owl:ObjectProperty>

<!-- Pno -->
<owl:ObjectProperty rdf:about="Pno">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Pnumber</rdfs:comment>
  <rdfs:range rdf:resource="Project"/>
  <rdfs:domain rdf:resource="Works_On"/>
</owl:ObjectProperty>

<!-- Super_ssn -->
<owl:ObjectProperty rdf:about="Super_ssn">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#SymmetricProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ssn</rdfs:comment>
  <rdfs:range rdf:resource="Employee"/>
  <rdfs:domain rdf:resource="Employee"/>
</owl:ObjectProperty>

<!-- Data properties -->
<owl:DatatypeProperty rdf:about="Address">
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<!-- Bdate -->
<owl:DatatypeProperty rdf:about="Bdate">
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>

<!-- DBdate -->
<owl:DatatypeProperty rdf:about="DBdate">
  <rdfs:domain rdf:resource="Dependent"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>
</!-- Dependent_name -->
<owl:DatatypeProperty rdf:about="Dependent_name">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Dependent</rdfs:comment>
  <rdfs:domain rdf:resource="Dependent"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

</!-- Dlocation -->
<owl:DatatypeProperty rdf:about="Dlocation">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Dept_Locations</rdfs:comment>
  <rdfs:domain rdf:resource="Dept_Locations"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

</!-- Dname -->
<owl:DatatypeProperty rdf:about="Dname">
  <rdfs:domain rdf:resource="Department"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

</!-- Dnumber -->
<owl:DatatypeProperty rdf:about="Dnumber">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Department</rdfs:comment>
  <rdfs:domain rdf:resource="Department"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>

</!-- Dsex -->
<owl:DatatypeProperty rdf:about="Dsex">
  <rdfs:domain rdf:resource="Dependent"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

</!-- Fname -->
<owl:DatatypeProperty rdf:about="Fname">
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

</!-- Hours -->
<owl:DatatypeProperty rdf:about="Hours">
<rdfs:domain rdf:resource="Works_On"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

!-- Lname --
<owl:DatatypeProperty rdf:about="Lname">
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

!-- Mgr_start_date --
<owl:DatatypeProperty rdf:about="Mgr_start_date">
  <rdfs:domain rdf:resource="Department"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>

!-- Minit --
<owl:DatatypeProperty rdf:about="Minit">
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

!-- Plocation --
<owl:DatatypeProperty rdf:about="Plocation">
  <rdfs:domain rdf:resource="Project"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

!-- Pname --
<owl:DatatypeProperty rdf:about="Pname">
  <rdfs:domain rdf:resource="Project"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

!-- Pnumber --
<owl:DatatypeProperty rdf:about="Pnumber">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Project</rdfs:comment>
  <rdfs:domain rdf:resource="Project"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>

!-- Relationship --
<owl:DatatypeProperty rdf:about="Relationship">
  <rdfs:domain rdf:resource="Dependent"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

!-- Salary --
<owl:DatatypeProperty rdf:about="Salary">
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="Sex">
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="Ssn">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Employee</rdfs:comment>
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>

<owl:Class rdf:about="Department"/>

<owl:Class rdf:about="Dependent">
  <rdfs:subClassOf rdf:resource="Employee"/>
</owl:Class>

<owl:Class rdf:about="Dept_Locations">
  <rdfs:subClassOf rdf:resource="Department"/>
</owl:Class>

<owl:Class rdf:about="Employee">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="Fname"/>
      <owl:minCardinality
        rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
A.4 EMPLOYEE OWL Ontology

<?xml version="1.0"?>
<rdf:RDF xmlns="EMPLOYEE#"
  xmlns:base="EMPLOYEE"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <owl:Ontology rdf:about="EMPLOYEE"/>
  <!-- Object Properties -->
  <!----
  // Object Properties
  //
  /////////////////////////////////////////////////////////////////////
  -->
  <!-- dept_no -->
  <owl:ObjectProperty rdf:about="dept_no">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">dno</rdfs:comment>
    <rdfs:domain rdf:resource="dept_emp"/>
    <rdfs:range rdf:resource="departments"/>
  </owl:ObjectProperty>
  <!-- emp_no -->
  <owl:ObjectProperty rdf:about="emp_no">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">eno</rdfs:comment>
    <rdfs:domain rdf:resource="dept_emp"/>
  </owl:ObjectProperty>
</rdf:RDF>
<rdfs:domain rdf:resource="dept_manager"/>
<rdfs:range rdf:resource="employees"/>
<rdfs:domain rdf:resource="salaries"/>
<rdfs:domain rdf:resource="titles"/>
</owl:ObjectProperty>

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

<!-- birth_date -->
<owl:DatatypeProperty rdf:about="birth_date">
  <rdfs:domain rdf:resource="employees"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>

<!-- dept_name -->
<owl:DatatypeProperty rdf:about="dept_name">
  <rdfs:domain rdf:resource="departments"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<!-- dno -->
<owl:DatatypeProperty rdf:about="dno">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">departments</rdfs:comment>
  <rdfs:domain rdf:resource="departments"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>

<!-- eno -->
<owl:DatatypeProperty rdf:about="eno">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">employees</rdfs:comment>
  <rdfs:domain rdf:resource="employees"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>

<!-- first_name -->
<owl:DatatypeProperty rdf:about="first_name">
  <rdfs:domain rdf:resource="employees"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<!-- from_date -->
<owl:DatatypeProperty rdf:about="from_date"
 .rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">salaries</rdfs:comment>
<rdfs:domain rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>
<!-- gender -->
<owl:DatatypeProperty rdf:about="gender"
  rdf:domain rdf:resource="employees"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<!-- hire_date -->
<owl:DatatypeProperty rdf:about="hire_date"
  rdf:domain rdf:resource="employees"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>
<!-- last_name -->
<owl:DatatypeProperty rdf:about="last_name"
  rdf:domain rdf:resource="employees"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<!-- salary -->
<owl:DatatypeProperty rdf:about="salary"
  rdf:domain rdf:resource="salaries"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>
<!-- title -->
<owl:DatatypeProperty rdf:about="title"
  rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">titles</rdfs:comment>
<rdfs:domain rdf:resource="titles"/>
</owl:DatatypeProperty>
<!-- to_date -->
<owl:DatatypeProperty rdf:about="to_date">
  <rdfs:domain rdf:resource="dept_emp"/>
  <rdfs:domain rdf:resource="dept_manager"/>
  <rdfs:domain rdf:resource="salaries"/>
  <rdfs:domain rdf:resource="titles"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>

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

<!-- departments -->
<owl:Class rdf:about="departments"/>

<!-- dept_emp -->
<owl:Class rdf:about="dept_emp">
  <rdfs:subClassOf rdf:resource="departments"/>
  <rdfs:subClassOf rdf:resource="employees"/>
</owl:Class>

<!-- dept_manager -->
<owl:Class rdf:about="dept_manager">
  <rdfs:subClassOf rdf:resource="departments"/>
  <rdfs:subClassOf rdf:resource="employees"/>
</owl:Class>

<!-- employees -->
<owl:Class rdf:about="employees"/>

<!-- salaries -->
<owl:Class rdf:about="salaries">
  <rdfs:subClassOf rdf:resource="employees"/>
</owl:Class>

<!-- titles -->
<owl:Class rdf:about="titles">
  <rdfs:subClassOf rdf:resource="employees"/>
</owl:Class>

</rdf:RDF>
A.5 MERGED OWL Ontology

<?xml version="1.0"?>
<rdf:RDF xmlns="Union#
  xml:base="Union"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdf="http://www.w3.org/1999/02/rdf-syntax-ns#">
  <owl:Ontology rdf:about="Union"/>
</rdf:RDF>

<!--  
// Object Properties
-->
<!--  Dno -->
<owl:ObjectProperty rdf:about="Dno">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Dnumber</rdfs:comment>
  <rdfs:range rdf:resource="Department"/>
  <rdfs:domain rdf:resource="Employee"/>
</owl:ObjectProperty>

<!--  Dnum -->
<owl:ObjectProperty rdf:about="Dnum">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Dnumber</rdfs:comment>
  <rdfs:range rdf:resource="Department"/>
  <rdfs:domain rdf:resource="Dept_Locations"/>
  <rdfs:domain rdf:resource="Project"/>
</owl:ObjectProperty>

<!--  EmSsn -->
<owl:ObjectProperty rdf:about="EmSsn">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ssn</rdfs:comment>
  <rdfs:domain rdf:resource="Dependent"/>
  <rdfs:range rdf:resource="Employee"/>
</owl:ObjectProperty>

<!--  Essn -->
<owl:ObjectProperty rdf:about="Essn">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ssn</rdfs:comment>
    <rdfs:range rdf:resource="Employee"/>
    <rdfs:domain rdf:resource="Works_On"/>
</owl:ObjectProperty>

<!-- Mgr_ssn -->
<owl:ObjectProperty rdf:about="Mgr_ssn">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ssn</rdfs:comment>
    <rdfs:domain rdf:resource="Department"/>
    <rdfs:range rdf:resource="Employee"/>
</owl:ObjectProperty>

<!-- Pno -->
<owl:ObjectProperty rdf:about="Pno">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Pnumber</rdfs:comment>
    <rdfs:range rdf:resource="Project"/>
    <rdfs:domain rdf:resource="Works_On"/>
</owl:ObjectProperty>

<!-- Super_ssn -->
<owl:ObjectProperty rdf:about="Super_ssn">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#SymmetricProperty"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ssn</rdfs:comment>
    <rdfs:range rdf:resource="Employee"/>
    <rdfs:domain rdf:resource="Employee"/>
</owl:ObjectProperty>

<!-- dept_no -->
<owl:ObjectProperty rdf:about="dept_no">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">dno</rdfs:comment>
    <rdfs:range rdf:resource="departments"/>
    <rdfs:domain rdf:resource="dept_emp"/>
    <rdfs:domain rdf:resource="dept_manager"/>
</owl:ObjectProperty>

<!-- emp_no -->
<owl:ObjectProperty rdf:about="emp_no"
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">eno</rdfs:comment>
<rdfs:domain rdf:resource="dept_emp"/>
<rdfs:domain rdf:resource="dept_manager"/>
<rdfs:range rdf:resource="employees"/>
<rdfs:domain rdf:resource="salaries"/>
<rdfs:domain rdf:resource="titles"/>
</owl:ObjectProperty>

<!--
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
-->
<!-- Address -->
<owl:DatatypeProperty rdf:about="Address">
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<!-- Bdate -->
<owl:DatatypeProperty rdf:about="Bdate">
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>

<!-- DBdate -->
<owl:DatatypeProperty rdf:about="DBdate">
  <rdfs:domain rdf:resource="Dependent"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>

<!-- Dependent_name -->
<owl:DatatypeProperty rdf:about="Dependent_name">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Dependent</rdfs:comment>
  <rdfs:domain rdf:resource="Dependent"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<!-- Dlocation -->
<owl:DatatypeProperty rdf:about="Dlocation">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Dept_Locations</rdfs:comment>

<owl:DatatypeProperty>
<!-- Dname -->
<owl:DatatypeProperty rdf:about="Dname">
  <rdfs:domain rdf:resource="Department"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<!-- Dnumber -->
<owl:DatatypeProperty rdf:about="Dnumber">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Department</rdfs:comment>
  <rdfs:domain rdf:resource="Department"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>

<!-- Dsex -->
<owl:DatatypeProperty rdf:about="Dsex">
  <rdfs:domain rdf:resource="Dependent"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<!-- Fname -->
<owl:DatatypeProperty rdf:about="Fname">
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<!-- Hours -->
<owl:DatatypeProperty rdf:about="Hours">
  <rdfs:domain rdf:resource="Works_On"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<!-- Lname -->
<owl:DatatypeProperty rdf:about="Lname">
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<!-- Mgr_start_date -->
<owl:DatatypeProperty rdf:about="Mgr_start_date">
  <rdfs:domain rdf:resource="Department"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <!-- Minit -->
    <owl:DatatypeProperty rdf:about="Minit">
        <rdfs:domain rdf:resource="Employee"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>

    <!-- Plocation -->
    <owl:DatatypeProperty rdf:about="Plocation">
        <rdfs:domain rdf:resource="Project"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>

    <!-- Pname -->
    <owl:DatatypeProperty rdf:about="Pname">
        <rdfs:domain rdf:resource="Project"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>

    <!-- Pnumber -->
    <owl:DatatypeProperty rdf:about="Pnumber">
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
        <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Project</rdfs:comment>
        <rdfs:domain rdf:resource="Project"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    </owl:DatatypeProperty>

    <!-- Relationship -->
    <owl:DatatypeProperty rdf:about="Relationship">
        <rdfs:domain rdf:resource="Dependent"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>

    <!-- Salary -->
    <owl:DatatypeProperty rdf:about="Salary">
        <rdfs:domain rdf:resource="Employee"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    </owl:DatatypeProperty>

    <!-- Sex -->
    <owl:DatatypeProperty rdf:about="Sex">
        <rdfs:domain rdf:resource="Employee"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>

    <!-- Ssn -->
    <owl:DatatypeProperty rdf:about="Ssn">
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    </owl:DatatypeProperty>
</body>
</html>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Employee</rdfs:comment>
  <rdfs:domain rdf:resource="Employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>
 <!-- birth_date -->
<owl:DatatypeProperty rdf:about="birth_date">
  <rdfs:domain rdf:resource="employees"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>
 <!-- dept_name -->
<owl:DatatypeProperty rdf:about="dept_name">
  <rdfs:domain rdf:resource="departments"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
 <!-- dno -->
<owl:DatatypeProperty rdf:about="dno">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">departments</rdfs:comment>
  <rdfs:domain rdf:resource="departments"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>
 <!-- eno -->
<owl:DatatypeProperty rdf:about="eno">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">employees</rdfs:comment>
  <rdfs:domain rdf:resource="employees"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</owl:DatatypeProperty>
 <!-- first_name -->
<owl:DatatypeProperty rdf:about="first_name">
  <rdfs:domain rdf:resource="employees"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
 <!-- from_date -->
<owl:DatatypeProperty rdf:about="from_date">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">salaries</rdfs:comment>
</owl:DatatypeProperty>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">titles</rdfs:comment>
   <rdfs:domain rdf:resource="dept_emp"/>
   <rdfs:domain rdf:resource="dept_manager"/>
   <rdfs:domain rdf:resource="salaries"/>
   <rdfs:domain rdf:resource="titles"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
   </owl:DatatypeProperty>
</!-- gender -->
<owl:DatatypeProperty rdf:about="gender">
   <rdfs:domain rdf:resource="employees"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
   </owl:DatatypeProperty>
</!-- hire_date -->
<owl:DatatypeProperty rdf:about="hire_date">
   <rdfs:domain rdf:resource="employees"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
   </owl:DatatypeProperty>
</!-- last_name -->
<owl:DatatypeProperty rdf:about="last_name">
   <rdfs:domain rdf:resource="employees"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
   </owl:DatatypeProperty>
</!-- salary -->
<owl:DatatypeProperty rdf:about="salary">
   <rdfs:domain rdf:resource="salaries"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
   </owl:DatatypeProperty>
</!-- title -->
<owl:DatatypeProperty rdf:about="title">
   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">titles</rdfs:comment>
   <rdfs:domain rdf:resource="titles"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
   </owl:DatatypeProperty>
</!-- to_date -->
<owl:DatatypeProperty rdf:about="to_date">
   <rdfs:domain rdf:resource="dept_emp"/>
   <rdfs:domain rdf:resource="dept_manager"/>
   <rdfs:domain rdf:resource="salaries"/>
   <rdfs:domain rdf:resource="titles"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>

<!--
 /////////////////////////////////////////////////
 //
 // Classes
 //
 /////////////////////////////////////////////////
 -->
<!-- Department -->
<owl:Class rdf:about="Department"/>
<!-- Dependent -->
<owl:Class rdf:about="Dependent">
  <rdfs:subClassOf rdf:resource="Employee"/>
</owl:Class>
<!-- Dept_Locations -->
<owl:Class rdf:about="Dept_Locations">
  <rdfs:subClassOf rdf:resource="Department"/>
</owl:Class>
<!-- Employee -->
<owl:Class rdf:about="Employee">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="Fname"/>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </owl:subClassOf>
</owl:Class>
<!-- Project -->
<owl:Class rdf:about="Project"/>
<!-- Works_On -->
<owl:Class rdf:about="Works_On">
  <rdfs:subClassOf rdf:resource="Employee"/>
  <rdfs:subClassOf rdf:resource="Project"/>
</owl:Class>
<!-- departments -->
<owl:Class rdf:about="departments"/>
<!-- dept_emp -->
<owl:Class rdf:about="dept_emp">
  <rdfs:subClassOf rdf:resource="departments"/>
  <rdfs:subClassOf rdf:resource="employees"/>
<!-- dept_manager -->
<owl:Class rdf:about="dept_manager">
  <rdfs:subClassOf rdf:resource="departments"/>
  <rdfs:subClassOf rdf:resource="employees"/>
</owl:Class>
<!-- employees -->
<owl:Class rdf:about="employees"/>
<!-- salaries -->
<owl:Class rdf:about="salaries">
  <rdfs:subClassOf rdf:resource="employees"/>
</owl:Class>
<!-- titles -->
<owl:Class rdf:about="titles">
  <rdfs:subClassOf rdf:resource="employees"/>
</owl:Class>
<!-- Alignments -->
<owl:DatatypeProperty rdf:about="Sex">
  <owl:equivalentProperty rdf:resource="gender"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="Bdate">
  <owl:equivalentProperty rdf:resource="birth_date"/>
</owl:DatatypeProperty>
<owl:Class rdf:about="Employee">
  <owl:equivalentClass rdf:resource="employees"/>
</owl:Class>
<owl:DatatypeProperty rdf:about="Dname">
  <owl:equivalentProperty rdf:resource="dept_name"/>
</owl:DatatypeProperty>
<owl:Class rdf:about="Department">
  <owl:equivalentClass rdf:resource="departments"/>
</owl:Class>
<owl:DatatypeProperty rdf:about="birth_date">
  <owl:equivalentProperty rdf:resource="Bdate"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="employees">
  <owl:equivalentClass rdf:resource="Employee"/>
</owl:Class>
<owl:DatatypeProperty rdf:about="gender">
  <owl:equivalentProperty rdf:resource="Sex"/>
A.6 Merged Schema DDL

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
CREATE SCHEMA IF NOT EXISTS `OptimaAlignment_Merged` DEFAULT CHARACTER SET utf8 ;
USE `OptimaAlignment_Merged` ;

-- Table `OptimaAlignment_Merged`.`departments`

CREATE TABLE IF NOT EXISTS `OptimaAlignment_Merged`.`departments` ( 
  `dno` INT(11) NOT NULL,
  `dept_name` VARCHAR(45) NULL DEFAULT NULL,
  `Mgr_ssn` INT(11) NULL DEFAULT NULL,
  `Dnumber` INT(11) NOT NULL,
  `Mgr_start_date` DATE NULL DEFAULT NULL,
PRIMARY KEY (`dno`, `Dnumber`),
INDEX `Dnumber` (`Dnumber` ASC),
INDEX `Mgr_ssn_FK3` (`Mgr_ssn` ASC),
CONSTRAINT `Mgr_ssn_FK3` FOREIGN KEY (`Mgr_ssn`) REFERENCES `OptimaAlignment_Merged`.`Employee` (`Ssn`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

-- Table `OptimaAlignment_Merged`.`Employee`

CREATE TABLE IF NOT EXISTS `OptimaAlignment_Merged`.`Employee` ( 
  `Super_ssn` INT(11) NULL DEFAULT NULL,
  `Dno` INT(11) NULL DEFAULT NULL,
  `Address` VARCHAR(45) NULL DEFAULT NULL,
  `Bdate` DATE NULL DEFAULT NULL,
  `Minit` VARCHAR(45) NULL DEFAULT NULL,
  `Sex` VARCHAR(45) NULL DEFAULT NULL,
  `Ssn` INT(11) NOT NULL,
  `Fname` VARCHAR(45) NOT NULL,
  `Lname` VARCHAR(45) NOT NULL,
  `Salary` INT(11) NULL DEFAULT NULL,
  `first_name` VARCHAR(45) NULL DEFAULT NULL,
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null?</th>
<th>Default</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eno</code></td>
<td>INT(11) NOT NULL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>last_name</code></td>
<td>VARCHAR(45) NULL DEFAULT NULL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>hire_date</code></td>
<td>DATE NULL DEFAULT NULL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRIMARY KEY (<code>Ssn</code>, <code>eno</code>)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDEX <code>eno</code> (<code>eno</code> ASC)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDEX <code>Super_ssn_FK5</code> (<code>Super_ssn</code> ASC)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDEX <code>Dno_FK6</code> (<code>Dno</code> ASC)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONSTRAINT <code>Dno_FK6</code> FOREIGN KEY (<code>Dno</code>) REFERENCES <code>OptimaAlignment_Merged</code>.departments (<code>Dnumber</code>)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONSTRAINT <code>Super_ssn_FK5</code> FOREIGN KEY (<code>Super_ssn</code>) REFERENCES <code>OptimaAlignment_Merged</code>.Employee (<code>Ssn</code>)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENGINE = InnoDB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEFAULT CHARACTER SET = utf8;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CREATE TABLE IF NOT EXISTS `OptimaAlignment_Merged`. `Dependent` (  
 `EmSsn` INT(11) NOT NULL,  
 `DBdate` DATE NULL DEFAULT NULL,  
 `Relationship` VARCHAR(45) NULL DEFAULT NULL,  
 `Dsex` VARCHAR(45) NULL DEFAULT NULL,  
 `Dependent_name` VARCHAR(45) NOT NULL,  
 PRIMARY KEY (`EmSsn`, `Dependent_name`),  
 CONSTRAINT `EmSsn_FK4` FOREIGN KEY (`EmSsn`) REFERENCES `OptimaAlignment_Merged`.Employee (`Ssn`)  
 ENGINE = InnoDB  
 DEFAULT CHARACTER SET = utf8;  

CREATE TABLE IF NOT EXISTS `OptimaAlignment_Merged`. `Dept_Locations` (  
 `Dnum` INT(11) NOT NULL,  
 `Dlocation` VARCHAR(45) NOT NULL,  
 PRIMARY KEY (`Dnum`, `Dlocation`),  
 CONSTRAINT `Dnum_FK1` FOREIGN KEY (`Dnum`) REFERENCES `OptimaAlignment_Merged`.departments (`Dnumber`)  
 ENGINE = InnoDB  
 DEFAULT CHARACTER SET = utf8;
-- Table `OptimaAlignment_Merged`.`Project`
CREATE TABLE IF NOT EXISTS `OptimaAlignment_Merged`.`Project` (  
  `Dnum` INT(11) NOT NULL,  
  `Plocation` VARCHAR(45) NULL DEFAULT NULL,  
  `Pname` VARCHAR(45) NULL DEFAULT NULL,  
  `Pnumber` INT(11) NOT NULL,  
  PRIMARY KEY (`Dnum`, `Pnumber`),  
  INDEX `Pnumber` (`Pnumber` ASC),  
  CONSTRAINT `Dnum_FK7`  
    FOREIGN KEY (`Dnum`)  
    REFERENCES `OptimaAlignment_Merged`.`departments` (Dnumber))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

-- Table `OptimaAlignment_Merged`.`Works_On`
CREATE TABLE IF NOT EXISTS `OptimaAlignment_Merged`.`Works_On` (  
  `Pno` INT(11) NOT NULL,  
  `Essn` INT(11) NOT NULL,  
  `Hours` VARCHAR(45) NULL DEFAULT NULL,  
  PRIMARY KEY (`Pno`, `Essn`),  
  INDEX `Essn` (`Essn` ASC),  
  CONSTRAINT `Essn_FK12`  
    FOREIGN KEY (`Essn`)  
    REFERENCES `OptimaAlignment_Merged`.`Employee` (Ssn),  
  CONSTRAINT `Pno_FK11`  
    FOREIGN KEY (`Pno`)  
    REFERENCES `OptimaAlignment_Merged`.`Project` (Pnumber))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

-- Table `OptimaAlignment_Merged`.`dept_emp`
CREATE TABLE IF NOT EXISTS `OptimaAlignment_Merged`.`dept_emp` (  
  `emp_no` INT(11) NOT NULL,  
  `dept_no` INT(11) NOT NULL,  
  `to_date` DATE NULL DEFAULT NULL,  
  `from_date` DATE NULL DEFAULT NULL,  
  PRIMARY KEY (`emp_no`, `dept_no`),  
  INDEX `dept_no` (`dept_no` ASC),
CONSTRAINT `dept_no_FK9`  
FOREIGN KEY (`dept_no`)  
REFERENCES `OptimaAlignment_Merged`.`departments` (`dno`),  
CONSTRAINT `emp_no_FK8`  
FOREIGN KEY (`emp_no`)  
REFERENCES `OptimaAlignment_Merged`.`Employee` (`eno`))  
ENGINE = InnoDB  
DEFAULT CHARACTER SET = utf8;

-- Table `OptimaAlignment_Merged`.`dept_manager`  
-- -----------------------------------------------  
CREATE TABLE IF NOT EXISTS `OptimaAlignment_Merged`.`dept_manager` (  
`emp_no` INT(11) NOT NULL,  
`dept_no` INT(11) NOT NULL,  
`to_date` DATE NULL DEFAULT NULL,  
`from_date` DATE NULL DEFAULT NULL,  
PRIMARY KEY (`emp_no`, `dept_no`),  
INDEX `dept_no` (`dept_no` ASC),  
CONSTRAINT `dept_no_FK14`  
FOREIGN KEY (`dept_no`)  
REFERENCES `OptimaAlignment_Merged`.`departments` (`dno`),  
CONSTRAINT `emp_no_FK13`  
FOREIGN KEY (`emp_no`)  
REFERENCES `OptimaAlignment_Merged`.`Employee` (`eno`))  
ENGINE = InnoDB  
DEFAULT CHARACTER SET = utf8;

-- Table `OptimaAlignment_Merged`.`salaries`  
-- -----------------------------------------------------  
CREATE TABLE IF NOT EXISTS `OptimaAlignment_Merged`.`salaries` (  
`emp_no` INT(11) NOT NULL,  
`salary` INT(11) NULL DEFAULT NULL,  
`to_date` DATE NULL DEFAULT NULL,  
`from_date` DATE NOT NULL,  
PRIMARY KEY (`emp_no`, `from_date`),  
CONSTRAINT `emp_no_FK2`  
FOREIGN KEY (`emp_no`)  
REFERENCES `OptimaAlignment_Merged`.`Employee` (`eno`))  
ENGINE = InnoDB  
DEFAULT CHARACTER SET = utf8;
CREATE TABLE IF NOT EXISTS `OptimaAlignment_Merged`.`titles`(
    `emp_no` INT(11) NOT NULL,
    `to_date` DATE NULL DEFAULT NULL,
    `title` VARCHAR(45) NOT NULL,
    `from_date` DATE NOT NULL,
    PRIMARY KEY (`emp_no`, `title`, `from_date`),
    CONSTRAINT `emp_no_FK10`
    FOREIGN KEY (`emp_no`)
    REFERENCES `OptimaAlignment_Merged`.`Employee` (`eno`)
) ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
Appendix B   Merged Schemas Generated using RIO tool

B.1 OptimaAlignment Merged Schema
B.2 StucSubsDistanceAlignment Merged Schema
B.3 StringDistAlignment Merged Schema
B.4 SubsDistNameAlignment Merged Schema
B.5 NameEqAlignment Merged Schema
B.6 EditDistNameAlignment Merged Schema