Georg-August-Universität
Göttingen
Zentrum für Informatik

ISSN 1612-6793
Nr. ZFI-BM-201x-xx

Master's Thesis
submitted in partial fulfilment of the
requirements of the course "Applied Computer Science"

Transformation-based Ontology Mapping

Sebastian Schrage

Institute of Computer Science

Bachelor's and Master's Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

30.November 2016

Georg-August-Universität Göttingen
Institute of Computer Science
Goldschmidtstraße 7
37077 Göttingen
Germany
☎ +49 (551) 39-172000
FAX +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
🌍 www.informatik.uni-goettingen.de

First Supervisor:	Prof. Dr. Wolfgang May
Second Supervisor:	Dr. Lena Wiese

I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Göttingen, 30.November 2016

Abstract

This thesis is about matching similar SQL [SQL] databases and transforming SPARQL [Apab] queries in the ontology of the one database into equivalent queries onto the other. To do so previous works [TH, RSa, RSb, Run] are used to further develop the storage of RDF [RDF] data in an SQL databases to an schema matching tool. As a result we got transformed ontological metadata which allows the QueryConverter to create SQL queries from the ontology of the other database. This approach delivers a new method of automated database matching with relative good results, but nevertheless automated database matching remains a great challenge for computer science.

Contents

List of Abbreviations

RDF	Resource Description Framework	43
SQL	Structured Query Language	31
XML	Extensible Markup Language	1
SPARQL	SPARQL Protocol And RDF Query Language	31
QC	QueryConverter	71
DC	DatabaseConverter	51
MD	MappingDictionary	72
SM	SchemaMatcher	72
RMB	RelationalModelBuilder	51
CV	ClassVariable	64
URI	Unique Resource Identifier	63
SMM	SchemaMatchingMatrix	99
MS	MatchingStep	65
UDT	User Defined Type	45
ODB	Ontology Data Base	71
FDB	Focus Data Base	71
GUI	Graphical User Interface	65

Chapter 1

Introduction

On of the biggest problems in data processing is no longer the existence nor the accessibility of data but that such data is stored in many formats, medias and locations. This leads to the predicament, that on the one hand, the sources are too distributed and too many for human users and on the other hand the sources are often not interpretable for machines to efficiently combine those. Therefore an attempt was made by introducing meta languages like Resource Description Framework (RDF) [W3Cc] and query languages like the SPARQL Protocol And RDF Query Language (SPARQL) to help machines to interpret that data, but in real world cases this is rarely used and many databases are just plain Structured Query Language (SQL) [SQL], Extensible Markup Language (XML), or another simpler database language, sometimes even insufficiently detailed RDF - based. To solve this problem, another even older problem with still no real sufficient automated solution must be solved; the matching of two similar databases. If this can be done, an ontology with sufficient meta information can be matched with a database which does not have this meta information. By matching these two the database, one can automatically get the necessary metadata from the ontology for the database.

In "RODI: Benchmarking Relational-to-Ontology Mapping Generation Quality" [PBJR$^+$15] such tools were evaluated. "Overall, systems demonstrate that they can cope well with relatively simple mapping challenges. However, all tested tools perform poorly on most of the more advanced challenges that come close to actual real-world problems. Thus, further research is needed to address these challenges" [PBJR$^+$15]. Since most of these tools were already finished and further development and improvements were rather unlikely, the topic for [Run] and this thesis is to develop a similar tool by adding the necessary changes to the previously develop RDF2SQL [TH,RSa], looking into the problems and evaluate the solution on the Mondial scenario of the RODI Benchmark. With the advantage of already having this data sets and benchmark queries even essential improvements could be made on the Mondial [May99] benchmark part.

The matchers mentioned in [PBJR$^+$15] do all more or less the same task with slightly different methods. Since most of these papers do not reveal too much information about the inner methods of each tool, it is quite hard to compare the structures. As far as it could be seen from the outside, those tools are based on pattern matching and graph theoretic methods as one would expect. The technique to use the ontological metadata structure developed for RDF2SQL is not too different from those other methods, but provides additional opportunities and is a new structural format to work with, which makes this approach very adaptable to different databases. Further, it fits into the other tools developed in previous works. The RDF2SQL consists of four parts which perform the following task:

Name	Abbrev.	Function	Reference
Relational Model Builder	RMB	Create ontological metadata from a arbitrary SQL database	[Run]
DatabaseConverter	DC	Create ontological metadata and an SQL database from an RDF ontology	[RSa]
SchemaMatcher	SM	Generate a mapping from on MappingDictionary (MD) to another MD	(this thesis)
QueryConverter	QC	Convert SPARQL queries into fitting SQL queries	[RSa]

This thesis is specifically about creating a matching from an ontology to a relational database and then transforming the ontological metadata in a way that SPARQL [Apab] queries based on one ontology can be used on the other database. This is based on mainly three components, the theoretical foundations of [TH], the RDF2SQL - Converter [RSa] and RelationalModelBuilder (RMB) [Run]. With this foundation, a matrix can be created of the two ontologies.With various methods the similarity of the different classes of both ontologies calculated and used for computing a matching those class pairs. This matching can then be used to transform the MD in a way that SPARQL queries against the one ontology can be transformed into equivalent SQL queries for the other database.

In this thesis, an adaptation of the QueryConverter was done and the development of the SchemaMatcher were made. The adaptation of the QC is mainly about its ability to use multi-column key values which was not possible before. Therefore additional meta information tables were introduced . The SchemaMatcher was a completely new developed program and uses the information of the DC or the RMB [Run] to match and translate databases.

1.1 SchemaMatcher overview - ontology mapping concept

The basic idea of this alignment tool is, that two ontologies with nearly the same content should have similarities in their structures. Therefore the SchemaMatcher uses so called *MatchingStep (MS)* to find those similarities. Each *MatchingStep* focus on one kind of hints which these ontologies provide and tries to find out which classes and attributes can be matched. The results are stored in similarity matrices and all calculations are done in two layers. The class layer, consisting of the classes from both ontology and the attribute layer which is specific for each class combination, consisting of the attributes of those two classes. For most steps, every single cell of the matrix in the class layer is influenced by its own matrix in the attribute layer (see Figure 1.1) .

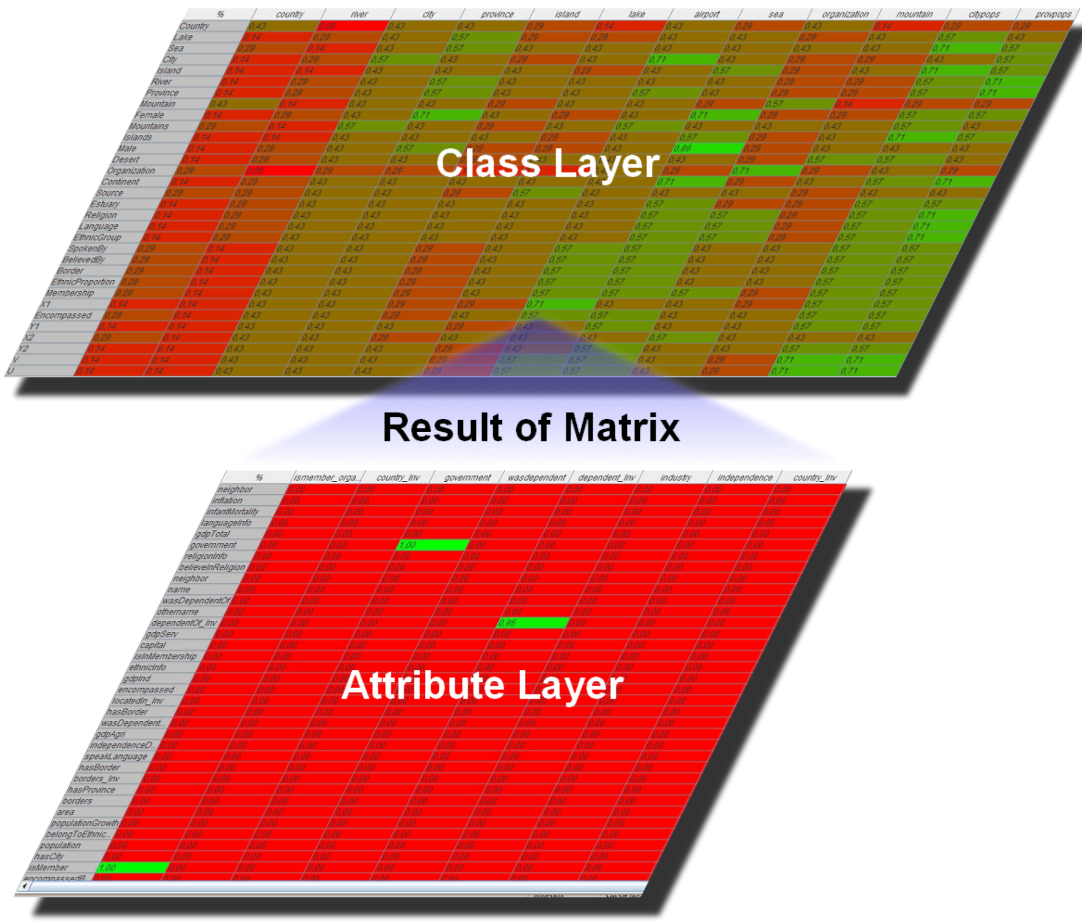


Figure 1.1: Layering of matrices - Every cell in the class layer is computed of a whole matrix in the attribute layer

The general execution order of those steps is determined by their phase they belong to (see Figure 1.2). After the class matching is done, the attributes were computed again, because with the class matching additional information are available.

After every step is done with its calculations, the result is automatically evaluated and depending on its significance it gets a weighting. The contribution of each MatchingStep for the finale results depends on those weights. This strict separation of the MatchingSteps come in handy, if a MatchingStep does not work due to the nature of one ontology, then it does not destroy everything, but gets just a low weight and does not have much impact anymore. When all steps are done, there is a finale decision matrix which gives the alignment of those ontologies.

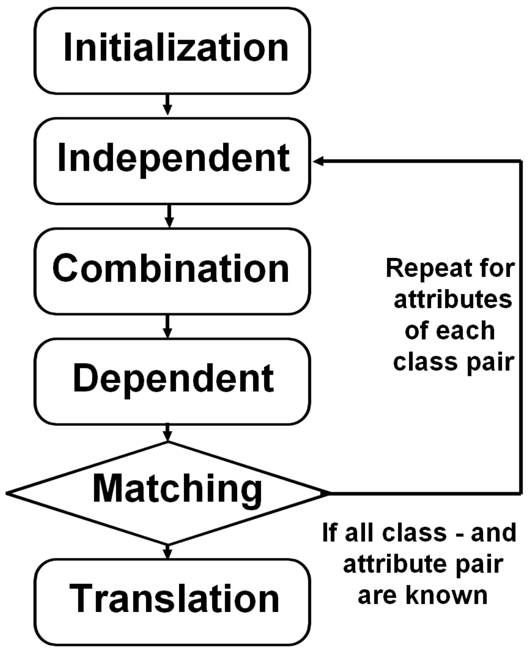

Initialization
Independent
Combination
Dependent
Matching
Translation
Repeat for attributes of each class pair
If all class - and attribute pair are known

Figure 1.2: Execution order of the calculation phases

1.2 Mondial overview

The programs of this thesis were tested and developed using the Mondial database. Mondial is a geographical database which is available in several formats. Mondial stores different information on cities, countries, organizations, continents and geographical landmarks like rivers, lakes, seas, islands and mountains (see Figure 1.3). Its ER model uses many different ways to design the relationships, therefore it is a good test approach for database related problems. It was created form different web sources in 1998. The major parts are from the CIA Factbook [Age], a part of the Karlsruher TERRA Database and the international atlas of Kümmerly & Frey, Rand McNally and Westermann. Originally Mondial was developed for testing the F-Logic system FLORID [FLO].

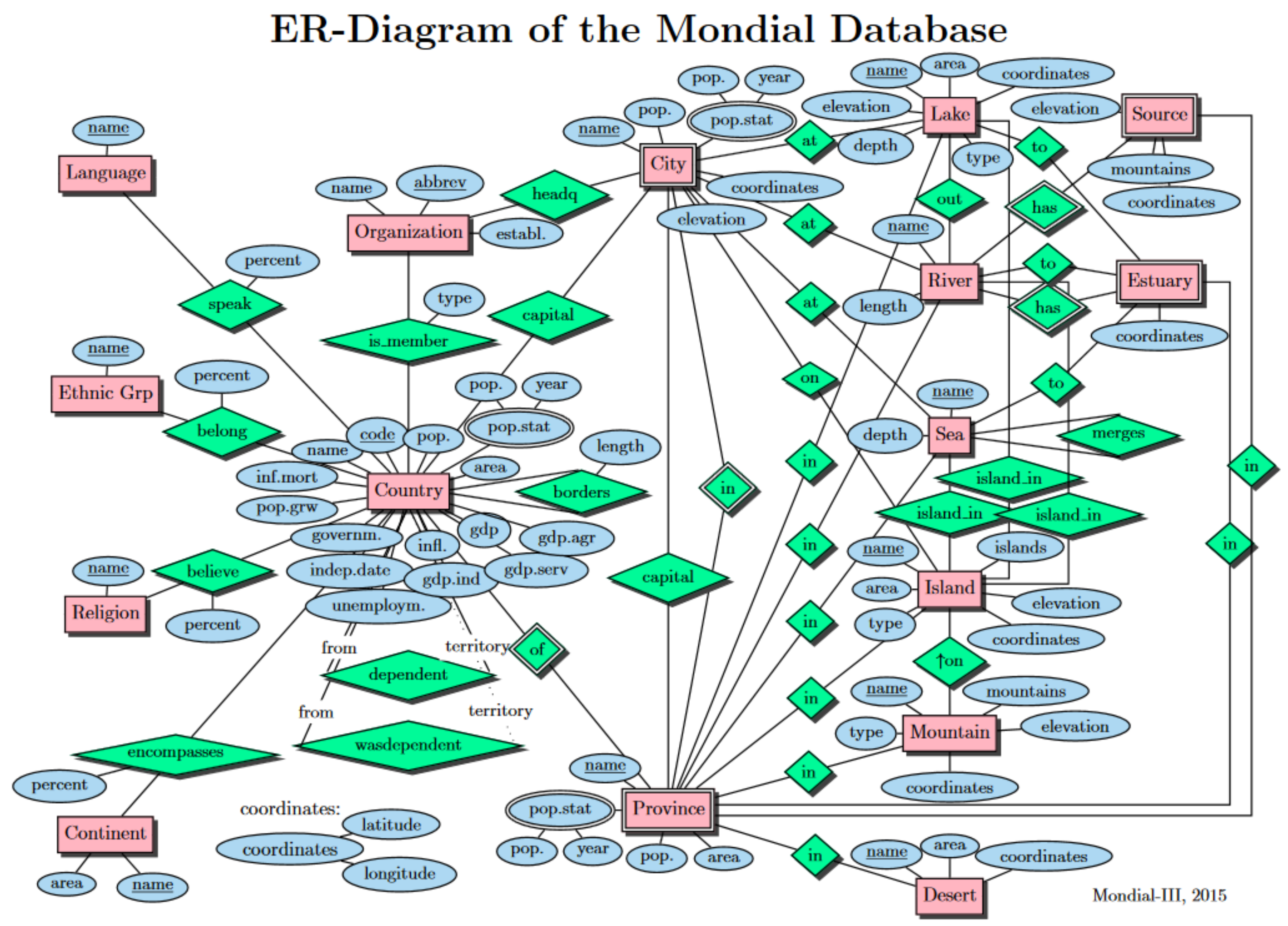


Figure 1.3: Mondial_rel - ER Diagram [May]

1.3 Related Work

On one hand, as an ontology to ontology mapper, the SchemaMatcher (SM) can be compared to LogMap [CGJR11](see Sub Section 1.3.1) which is a mapper for detailed ontologies and provides high scalability. In opposite to the SM, LogMap was not created for derived but for well designed, hand made and detailed ontologies. Never the less BootOx uses this program for the class matching.

On the other hand, as an OBDA tool, the whole program RDF2SQL must be compared to other OBDA tools, like BootOX [JRKZ+15], Karma [ea12], COMA++ [ea05], MIRROR [dMPC15], IncMap [PBKH13] and OnTop [ea14] (see sub Section 1.3.2 & 1.3.3). These tools split into two scopes of application: the semi-automated supportive tools and the fully automated stand alone tools. Like the other tools, the SchemaMatcher is able to be modified by various settings and its performance might be improved by that, but it can be considered a fully automated approach, since it does not require the evaluation nor the decision making of a human.

1.3.1 LogMap [CGJR11]

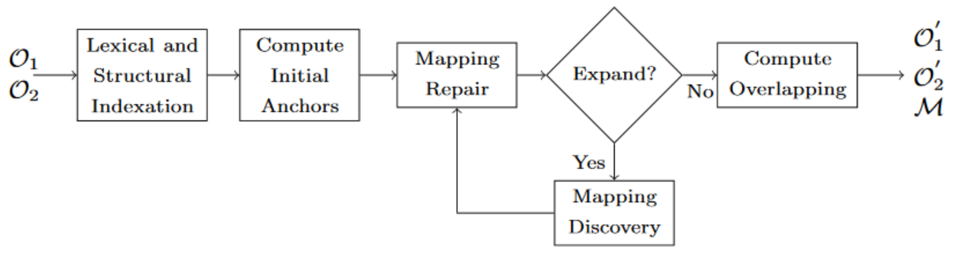


Figure 1.4: LogMap in a Nutshell [CGJR11]

LogMap is made for calculating a matching between two well designed ontologies. The results of LogMap should be much better than the results of the SM, as long as one deals with hand made ontologies. When it comes to ontologies, derived from relational databases, tables turn. "LogMap uses an interval labeling schema to represent the extended class hierarchy of each input ontology" [CGJR11] where it can unfold the power of reasoning (see Fig.1.4). While SM is based on similarities and method evaluation and do not need any hierarchies (but can use them as well). The following five points describe the work flow of LogMap and have been taken from [CGJR11]:

1. *Lexical indexation*. The first step after parsing the input ontologies is their lexical indexation. LogMap indexes the labels of the classes in each ontology as well as their lexical variations, and allows for the possibility of enriching the indexes by using an external lexicon (e.g., WordNet or UMLS-lexicon).

2. *Structural indexation*. LogMap uses an interval labeling schema to represent the extended class hierarchy of each input ontology. Each extended hierarchy can be computed using either simple structural heuristics, or an in-the-shelf DL reasoner.

3. *Computation of initial 'anchor mappings'*. LogMap computes an initial set of equivalence anchor mappings by intersecting the lexical indexes of each input ontology. These mappings can be considered 'exact' and will later serve as starting point for the further discovery of additional mappings.

4. *Mapping repair and discovery.* The core of LogMap is an iterative process that alternates repair and discovery steps.

   (a) In the repair step , LogMap uses a sound and highly scalable (but possibly incomplete) reasoning algorithm to detect classes that are unsatisfiable w.r.t. (the merge of) both input ontologies and the mappings computed thus far. Then, each of these undesirable logical consequences is automatically repaired using a 'greedy' diagnosis algorithm.

   (b) To discover new mappings , LogMap maintains two contexts (sets of 'semantically related' classes) for each anchor. Contexts for the same anchor are expanded in parallel using the class hierarchies of the input ontologies. New mappings are then computed by matching the classes in the relevant contexts using ISUB [SSK05] a flexible tool that computes a similarity score for any pair of input strings. This mapping discovery strategy is based on a principle of locality : if classes C_1 and C_2 are Inverted correctly mapped, then the classes semantically related to C_1 in $\mathcal{O}_1$ are likely to be mapped to those semantically related to C_2 in $\mathcal{O}_2$.

   LogMap continues the iteration of repair and discovery steps until no context is expanded in the discovery step. The output of this process is a set of mappings that are likely to be 'clean' that is, it will not lead to logical errors when merged with the input ontologies.

5. *Ontology overlapping estimation*. In addition to the final set of mappings, LogMap computes a fragment of each input ontology, which intuitively represent the 'overlapping' between both ontologies. When manually looking for additional mappings that LogMap might have missed, curators can restrict themselves to these fragments since 'correct' mappings between classes not mentioned in these fragments are likely to be rare.

The *Lexical indexation* and Name rating s.(3.2.3.1), and the *Structural indexation* and inheritance rating (see Section 3.2.3.2) are very similar to each other.

During step 1, as well as during the name rating, first the synonyms were added to each name, like it is done in 3.2.3.1, then the distance or the similarity of the words is calculated. Major difference in the name mapping is that LogMap vector based approach should be much faster, but relies on the usually very large and precise ontology names, while SM distinguishes between exact matches and similar words. If the ontology is derived from a relational database, it is very likely that the names are shorter than from well designed ontologies and even slight differences can matter (e.g. Mondial_RDF2SQL_Standard.mountain $\neq$Mondial_RDF2SQL_Standard.mountains).

In the *Structural indexation* "LogMap exploits the information in the (extended) class hierarchy of the input ontologies in different steps of the matching process." [CGJR11] which is comparable with the inheritance rating (see Section 3.2.5.2) of the SM, but much more sophisticated. In context of ontology to ontology mapping, which LogMap originally was made for, this is a powerful method. However in context of ontologies derived from relational database there will be no hierarchy of classes, since a common ER model just does not have a class hierarchy. Without such a proper class structure the whole reasoning aspect of LogMap, which made it so powerful, will be eliminated. If the reasoning does not work, all further steps degenerate from very sophisticated and effective to very basic steps, since the anchor points are only trivial matches now and the repair and discover phases have no base to work with.

1.3.2 Automated Approaches

Those approaches are very similar to RDF2SQL, but the internal matching methods and the meta data storage vary from RDF2SQL. The following characterizations are taken from [PBJR$^+$15].

BootOX [JRKZ$^+$15] is based on the approach called direct mapping by the W3C every table in the database (except for those representing n:m relationships) is mapped to one class in the ontology; every data attribute is mapped to one data property; and every foreign key to one object property. Explicit and implicit database constraints from the schema are also used to enrich the bootstrapped ontology with axioms about the classes and properties from these direct mappings. Afterwards, BootOX performs an alignment with the target ontology using the LogMap system.

MIRROR [dMPC15] is a tool for generating an ontology and R2RML direct mappings automatically from an RDB schema. MIRROR has been implemented as a module of the RDB2RDF engine morph-RDB.

IncMap [PBKH13] maps an available ontology directly to the relational schema. IncMap represents both the ontology and schema uniformly, using a structure-preserving meta-graph for both. It runs in two phases, using lexical and structural matching.

COMA++ [ea05] has been a contender in the field of schema matching for several years already; it is still widely considered state of the art. In contrast to other systems from the same era, COMA++ is built explicitly also for inter-model matching.

OnTop [ea14] the Ontop Protege Plugin is a mapping generator developed for OnTop. OnTop is a full-fledged query rewriting system with limited ontology and mapping bootstrapping capabilities.

1.3.3 Semi-automated approach

Karma [ea12] does need a human supervisor and OnTop [ea14] as well as BootOX [DHEM, JRKZ$^+$15] have modes which requires human [DHEM] help with decisions or evaluating results.

Even though this leads to better results, this always means that the program relies on a human to make critical decisions. It is not fair to compare these approaches to the automated ones, since they skip the most critical and error prone parts and can be considered rather as supportive tools than as replacement by their nature. However, those tools might be the most useful ones and since "all tested tools perform poorly on most of the more advanced challenges that come close to actual real-world problems" [PBJR$^+$15] none of these tools can be used without being controlled by a human. [PBJR$^+$15] states that:

Karma [ea12] "is one of the most prominent modern relational-to-ontology mapping generation systems. It is strictly semi-automatic, i.e., there is no fully automatic baseline that we could use for non-interactive evaluation. In addition, Karma's mode of iterations is designed to take advantage mostly from integrating a series of data sources to the same target ontology. Karma is therefore not well suited for single-scenario evaluations." [PBJR$^+$15]

1.4 Contributions

Can handle weak ontologies This thesis is about strategies to match sparse detailed ontologies. It does not rely on class hierarchies nor class constraints which could not be expected from derived ontologies. Normally a ontology to ontology mapper would assume to have rich super and sub class trees for every class, like LogMap [CGJR11] did. However, with derived ontologies there will be no to very little class hierarchies or constrains therefore this work offers alternatives to this inheritance dependency.

Fully automated ontology to ontology mapping SM does not rely on completing the resulting ontologies and mappings manually other than COMA++ [ea05], IncMap [PBKH13], Karma [ea12] or OnTop [ea14].

Improvement performance on Mondial While working towards [PBJR$^+$15] Mondial was also evaluated as an benchmark scenario. However, since nearly none of the queries could be answered by any tool, it did not made it into the published version. This result was the motivation to try an own approach and find out what the reason is and if there is a solution for at least some of those problems. The performance of the SchemaMatcher was found to be better than that of the other tools (see chapter 6.1). It is able to handle about 40% of the queries of the RODI - Mondial - Benchmark [PBJR$^+$15] (see chapter 6.1). This tool is only tested on the Mondial queries, since they are considered very well designed and nearly cover all common database problems. However, the RDF2SQL tool was developed on this database, so one can assume some kind of an advantage for the SM.

Analyzing the problems Some of the biggest problems with the RODI benchmark for matching tools (see Section 6.2) are pointed out. The Graphical User Interface (GUI) (see Section 4.3) and the MD make an analysis of these problems very comfortable, shows where improvements can be made and problems exist, to support further development.

Chapter 2

Basics

2.1 Tools and Test data sets

2.1.1 Tools

- All coding was done with Java 7
- For programming the IDE Eclipse Kepler Service Release 2 [Fou] was used.
- All database operations where performed on a postgres [Grob, Groa] database with pgAdmin III [pDT].

2.1.2 Test data set - Mondial and RODI

The Mondial database has the great advantage of having a reasonable size, using many different design techniques in Structured Query Language (SQL) and a data set with real data, which makes it much easier to realize if something is wrong and what might be the problem, in comparison with generic data. Further it is available in F-Logic, SQL , XML, Resource Description Framework (RDF) /OWL and Datalog at [May99] [RSa]. Due to the different language, the different Mondials do not have the same structure in every approach. With the DatabaseConverter (DC) it is possible to transform the OWL-Mondial into a SQL version named Mondial_RDF2SQL. This version differs from the original SQL version of Mondial, named Mondial_rel, in many ways. The Mondial_RDF2SQL does always have a Unique Resource Identifier (URI) column for every class and it is the only primary key, while the Mondial_rel has many multi-column key values. This also leads to weak entities in Mondial_rel like City or Source. Additional Mondial_rel has no class hierarchy at all, due to its relational model nature. This means the relations between class can be much more complicate, for example, the loactedAt relation is on Mondial_RDF2SQL a reference to Water while in Mondial_rel there is a single relation to each subclass of water, means to sea, lake and river. Further some attributes might not exists in both ontologies, have different names (especially the inverses from Mondial_rel do not have own names, while the ones form Mondial_RDF2SQL do) and some of them have a complete different relation to their classes. For example, the population of a city is just a literal in Mondial_RDF2SQL, while it is a weak entity with a 1:n connection in Mondial_rel, since it stores the population of a specific date and not just only the current population. Other classes like *Estuary* and *Source* are normal classes in Mondial_RDF2SQL, but in Mondial_rel these are only attributes of the *River* class. All these smaller differences in the ontologies, which have a big impact on the database structure, result in two different ontology, modeling the same situation, with the same data and the same query results, which makes them perfect for an close-to real world use case for ontology alignment.

Additional the RODI - Mondial - benchmark - scenario - queries [PBJR^{+}15] make testing of prototypes much easier. To evaluate the results of database converter the RODI test [PBJR^{+}15] queries where developed. These queries aim to test a broad variety of complications, which might appear on databases and have always an equivalent query in SQL to compare the results with the correct SQL result. These benchmark queries were tested on the QueryConverter and the SchemaMatcher in different stages of the development. The great advantage of such a broad variety of queries is, that not just the special case which was under development can be tested, but the whole program with minimal efforts to get easily ride of side effects.

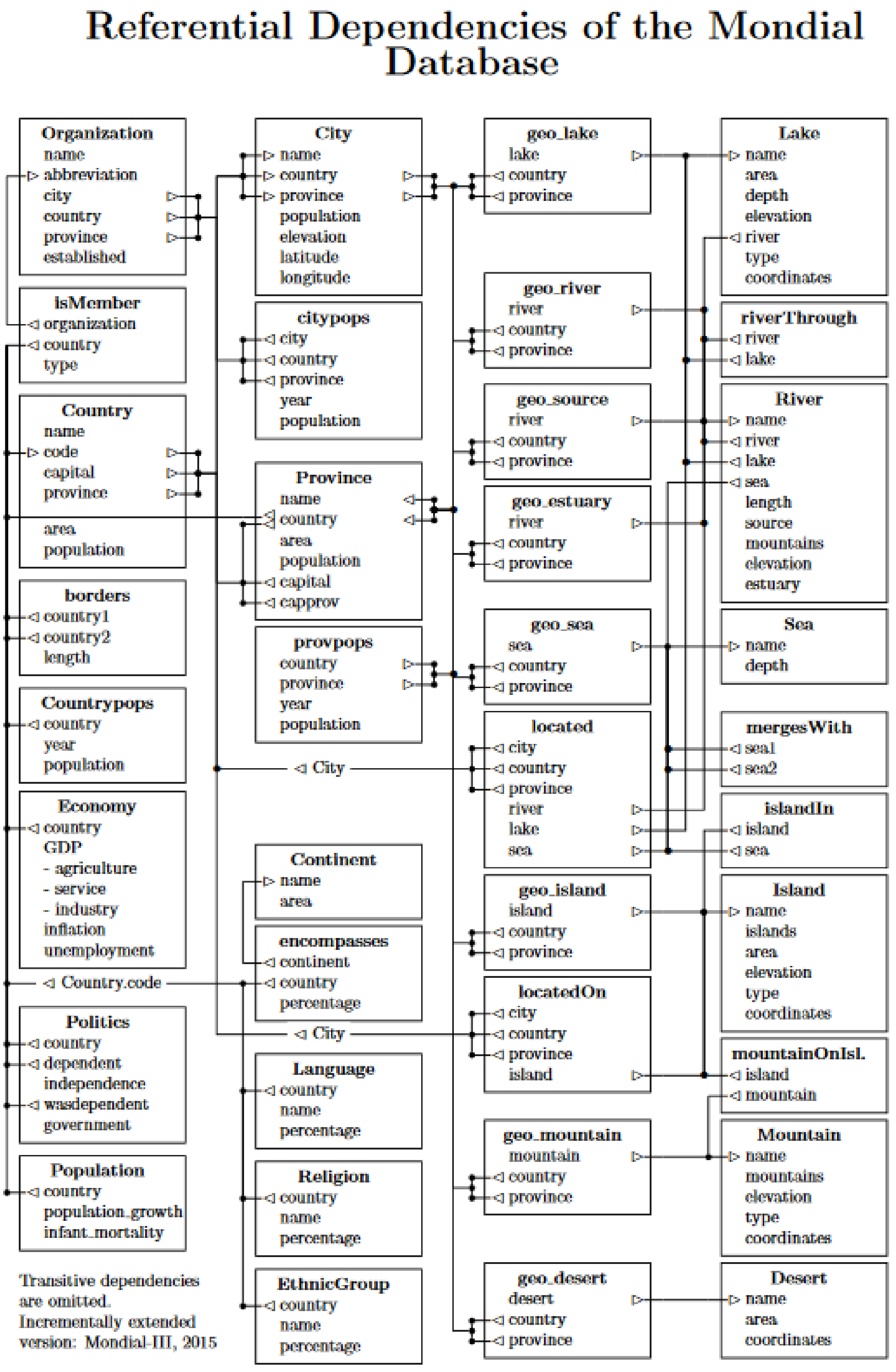


Figure 2.1: Mondial_rel referential dependencies - Diagram [May]

2.2 Metadata structures

When designing a database there is always a so called metadata structure involved. This structure is made of the conceptional ideas behind the attempt to model a real world problem in a database structure. A metadata structure is not about the actual data set, which should be stored or accessed, but about the way how that data is represented. Therefore a classification of the environment happens. This classification can be expressed by creating ontology classes or tables.

2.2.1 Relational Databases

Relational databases are the most common type of databases and are used in business application as well as in scientific context. They are based on a relational model which was introduced by Codd [Cod70]. This model consists of *tuples* to represent data and group these *tuples* into *relations*. Those *relations* are usually entities of the context, which should be represented in the model, for example a country, a city, or a person. Those *relations* consists of *tuples*, which are a combination of labels, data types and context, e.g. name, size, or a reference to another *relation* like isCityIn or isMarriedWith. *Tuples* of a single *relation* always have to have the same structure or in other words *relations* are fully structured, which means, they take a fixed amount of memory each, therefore iterations through the *tuples* of one *relations* can be done efficiently.

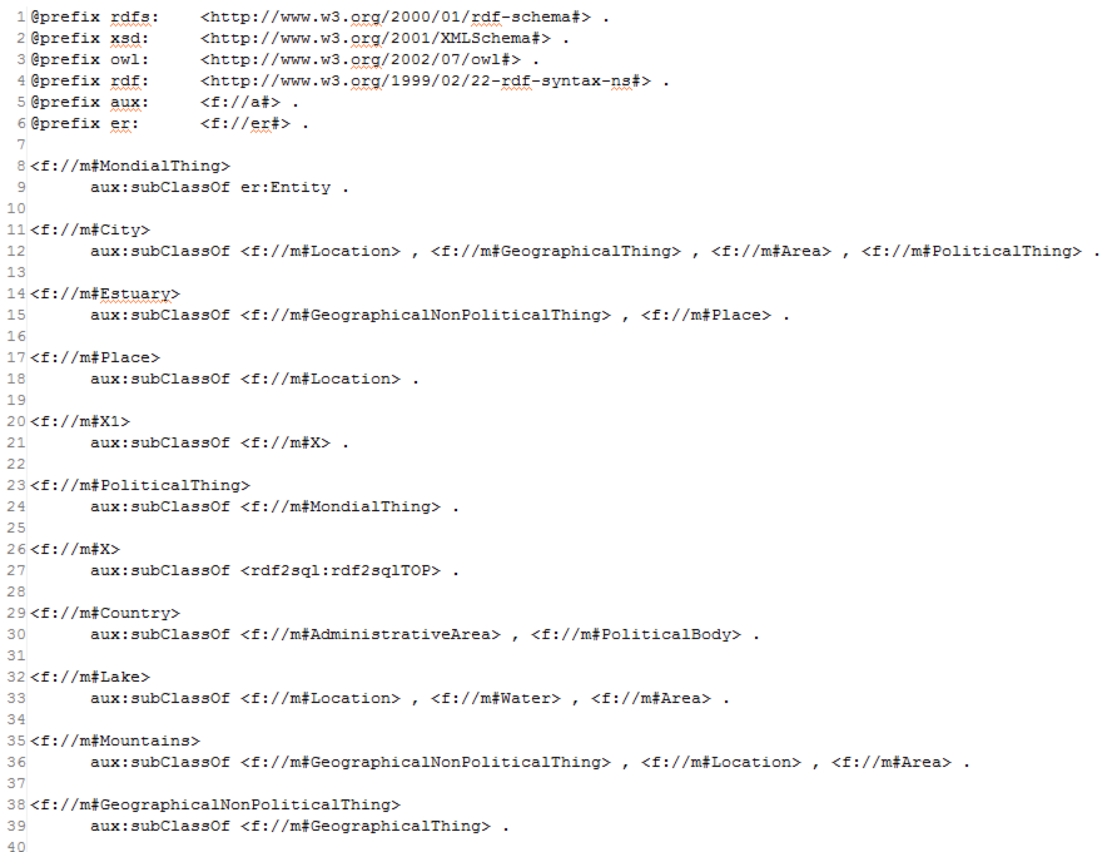
The model can be realized in a relational database management system, where the *relations* are the tables, every attribute of the *relation* is a column and each *tuple* is a row in those tables. With the help of such a relational database management system, like postgres [Grob] or Oracle for the database language SQL [SQL], it is possible to ask queries against those tables and perform algebraic operations on them.

2.2.2 Entity - relationship model - ER model

Entity–relationship modeling is an abstract data model that graphically represents the data structure of a relational database by using different shapes and lines to represent the *relations* and the connection between them (e.g. see Fig.1.3). Slightly different variants and ideas existed previously, but Peter Chen [Che76] is considered as the creator of the ER model, in 1967. His model which is still the basis of nowadays used ER models. The ER model gives the possibility to create classes, literals, different types of class connection with cardinality for those relations.

It was created to support database designer during the development and to give users an easier introduction into a new database. Therefore an ER model is mainly for humans to understand the data structure, but in contrast to ontologies, hardly readable information for machines.

2.2.3 Ontology



```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix aux:     <f://a#> .
@prefix er:      <f://er#> .

<f://m#MondialThing>
        aux:subClassOf er:Entity .

<f://m#City>
        aux:subClassOf <f://m#Location> , <f://m#GeographicalThing> , <f://m#Area> , <f://m#PoliticalThing> .

<f://m#Estuary>
        aux:subClassOf <f://m#GeographicalNonPoliticalThing> , <f://m#Place> .

<f://m#Place>
        aux:subClassOf <f://m#Location> .

<f://m#X1>
        aux:subClassOf <f://m#X> .

<f://m#PoliticalThing>
        aux:subClassOf <f://m#MondialThing> .

<f://m#X>
        aux:subClassOf <rdf2sql:rdf2sqlTOP> .

<f://m#Country>
        aux:subClassOf <f://m#AdministrativeArea> , <f://m#PoliticalBody> .

<f://m#Lake>
        aux:subClassOf <f://m#Location> , <f://m#Water> , <f://m#Area> .

<f://m#Mountains>
        aux:subClassOf <f://m#GeographicalNonPoliticalThing> , <f://m#Location> , <f://m#Area> .

<f://m#GeographicalNonPoliticalThing>
        aux:subClassOf <f://m#GeographicalThing> .

```

Figure 2.2: Example from the classes of mondial in RDF

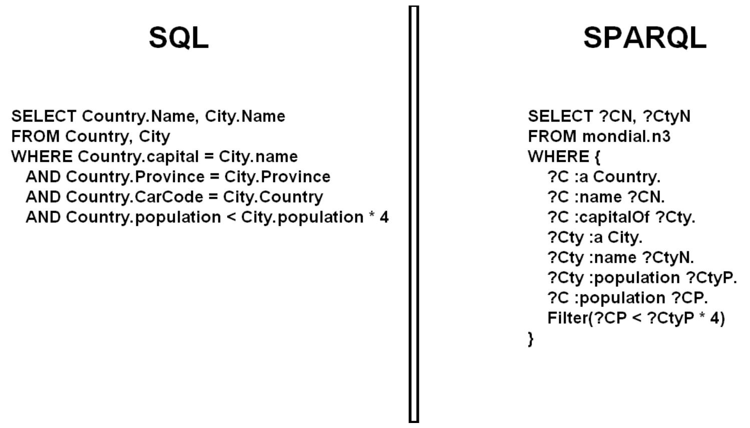
An Ontology is a formal way to describe classifications and taxonomies. Ontologies uses class hierarchies but in contrast to programming languages, the inheritance is made for constantly changing and other than most programming languages multi - inheritance is actually wanted and used to classify an entity into multiple groups. For example, a lake could be a subclass of Location, Water, and Area, while Mountains have the subclasses GeographicalNonPoliticalThing, Location and Area (see Figure 2.2). Through this the ontology states, that those two entities describe different thing, but have a lot in common, while other entities with different superclasses are not related to them. This superclasses usually have superclasses themselves up to a very high level of abstraction. These inheritance structure give those information a semantic meaning, while the entities of an ER model are just the class and the attributes of this class. In a ER model there is no hint, if a lake is more similar to a sea or an organization, since the ER model do not have a semantic context at all. Beside the inheritance an ontology can also define constraints and cardinalities for relations.

2.2.4 RDF & SPARQL

The Resource Description Framework (RDF) was developed by the W3C [W3Cc] as standard to describe meta data. Like the ER Model it is a model to describe the conceptional structure of a data storage. The main difference is that it is not a graphical but a textual representation. Mostly the languages XML [W3Cd] and n3 [W3Ca] are used for textual representation. It is slightly harder to understand for human, at least to get a first overview, but RDF is much easier to understand for machines. Therefore it functions as a human / machine interface.

Figure 2.3: RDF Icon

Nowadays RDF is one of the most essential parts of semantic web. The goal of the semantic web is to give information a semantic meaning. The machine and human readability of RDF provides the standard way to do so. A RDF Model itself consists of a directed graph. The nodes and edges of the graph are described by triples. Those triples consists of a subject, a predicate and an object in this order. The subject is always an entity, or in another words a node in the graph, while the object could be a literal, like *Germany; population; 81600000;* or an entity, like *Germany; borders; France* as well. The predicates between two entities creates the edges in the graph.



SQL

```
SELECT Country.Name, City.Name
FROM Country, City
WHERE Country.capital = City.name
  AND Country.Province = City.Province
  AND Country.CarCode = City.Country
  AND Country.population < City.population * 4
```

SPARQL

```
SELECT ?CN, ?CtyN
FROM mondial.n3
WHERE {
   ?C :a Country.
   ?C :name ?CN.
   ?C :capitalOf ?Cty.
   ?Cty :a City.
   ?Cty :name ?CtyN.
   ?Cty :population ?CtyP.
   ?C :population ?CP.
   Filter(?CP < ?CtyP * 4)
}
```

Figure 2.4: Equivalent query in SQL and SPARQL. The queries ask for the name of all countries and their capitals where at least 1/4 oft the population lives in the capital.

To access those graphs and ask queries against a RDF based databases SPARQL Protocol And RDF Query Language (SPARQL) [WC3] is the recommend language. SPARQL queries can be asked with tools like Apache Jena [Apaa]. The basic of SPARQL are very similar to those of SQL, both use the SFW - block (Select; From; Where). However in a graph based database, there are no tables which could be joined, therefore SPARQL selections and conditions are based on variables and properties. All conditions are formulated in triples (see Figure 2.4).

2.2.5 *Web Ontology Language* - OWL

The *Web Ontology Language* (OWL*)* is a specification from the W3C [W3Cb], based on the W3C XML standard [W3Cd], to create ontologies in formal description languages. The first version of OWL was published in 2004 and later on 27 October 2009 OWL2 was published. OWL expands the RDF ontologies with operations like intersection and union of classes but also makes constraints in order to archive decidability. Decidable ontologies can be used for reasoning, which is one of the most important features of the semantic web and the biggest difference to the traditional relational model. OWL2 is used in semantic editors like Protegé [KFNM04] and semantic reasoners like Pellet [SPG+07], FaCT++ [TH06] or RacerPro [HHMW12]. Reasoners like Pellet can be used from query languages like SPARQL to gain additional knowledge about entities in an ontology or ask first order logic based queries against an ontology.

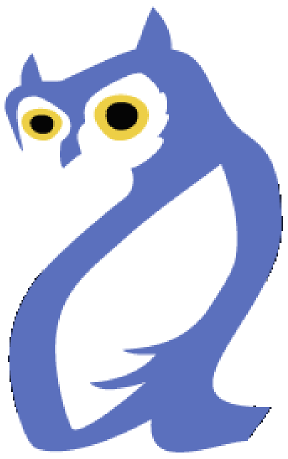


Figure 2.5: OWL Icon [OWL]

The W3C OWL specification define three variants of OWL: OWL Lite, OWL DL and OWL Full. Each variant has different level of expressiveness.

OWL Lite OWL Lite was created with the intention to give an easy to implement subset of this languages. It is mainly used for simpler taxonomies and constraints. OWL Lite is equally expressive as the description logic $\mathcal{SHIF}(D)$ [GHM+08]

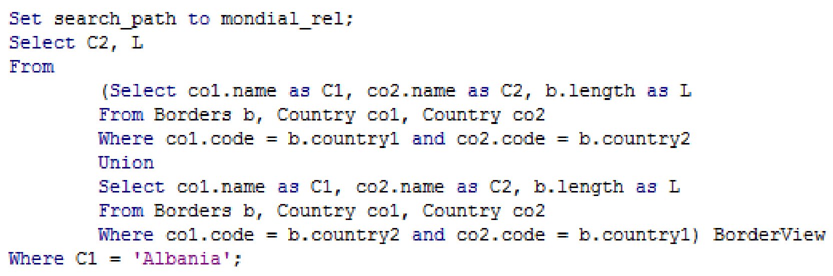
OWL DL OWL Description Logic has the expressiveness of the description logic $\mathcal{SHOIN}(D)$ while being computational completed and decidable. [GHM+08]

OWL FULL OWL Full was created to be fully compatible with RDF Schema, unlike the other OWL variances. Therefore OWL Full has different semantics, than the others. As a down side, OWL Full ontologies are not always decidable nor computational completely, i.g. reasoners can not perform complete reasoning for those ontologies.

2.2.6 Symmetric reification

A rather uncommon but expressive relational model strategy are so called symmetric reified tables. Those tables have two major characteristics:

- Reified: Those *relations* represent NM relationships, but those relationships also have own attributes, like the *Membership* in an *Organization* can have a additional attribute, the *type* of this *Membership.*
- Symmetric: Those *relations* are symmetric means the *tuple* (x,y) also implies that the *tuple* (y,x) exits but this *tuple* is not stored in the database, like the *mergesWith* attribute of *sea*.



```
Set search_path to mondial_rel;
Select C2, L
From
        (Select co1.name as C1, co2.name as C2, b.length as L
        From Borders b, Country co1, Country co2
        Where co1.code = b.country1 and co2.code = b.country2
        Union
        Select co1.name as C1, co2.name as C2, b.length as L
        From Borders b, Country co1, Country co2
        Where co1.code = b.country2 and co2.code = b.country1) BorderView
Where C1 = 'Albania';
```

Figure 2.6: Asking the symmetric reified relation border for all bordering countries of Albania and the length of those borders.

If a relationship fulfills both conditions, it is a symmetric reified table. An example in the Mondial_rel database for those relations is the *borders* table. The *border* class has the attributes *country1* and *country2* which are exchangeable, but also has the length of each border stored and is a NM *relation* between two *countries*. Those relations do need special treatment from the user respectively the QueryConverter (QC) (see Figure 2.6). A Union over both combination is necessary to get all *tuples*. Also a conjunction, so that either the one or the other value of both symmetric attributes is the fixes value, would be possible, but in this case the fixed value could be in both of those attributes.

Symmetric reification is not part of the W3C RDF standard [W3Cc], but it is possible to modulate that kind of relationship in RDF as well.

2.3 RDF2SQL

The RDF2SQL converter [RSa] has been developed in earlier works. It basically creates an equivalent SQL Database from an RDF database and an ontology mapping metadata description with the necessary information to convert a SPARQL query into an equivalent SQL query. The program itself consists of two subprograms, the DC [RS13] and the QC [RSa]. These two can be executed independently from each other but the QC needs always an MappingDictionary (MD) in the database. The original version of the **OC!** (**OC!**) also needed the exact database structure created by the DC.

RDF2SQL was originally developed to improve the performance of queries against these data sets since SPARQL queries tend to take much more time than equivalent SQL ones [RSa]. Aside from this advantage, the MD structure provides further possibilities. One of this possibilities is the transformation from one MD to another MD by just changing the classes and properties respectively to the tables and columns to transform one MD into the other.

2.3.1 DC - DatabaseConverter

The DatabaseConvter (DC) [RSa, TH, RS13] was created in [RS13] based on [TH].

The DC converts a RDF [W3Cc] OWL based ontology into an equivalent SQL [SQL] database. The DC uses a SPARQL [Apab] query chain [TH] to gather the necessary information about the database.

After creating the table structure and the MD the next step is to read in the triple. Therefore the information of the triple are stored until enough information is gathered to decide where it belongs, in so called TOP tables [TH]. With enough knowledge about the triples, the information is distributed across the tables. If the ontology is correct, these TOP Tables are empty in the end and can be removed.

2.3.2 QC - QueryConverter

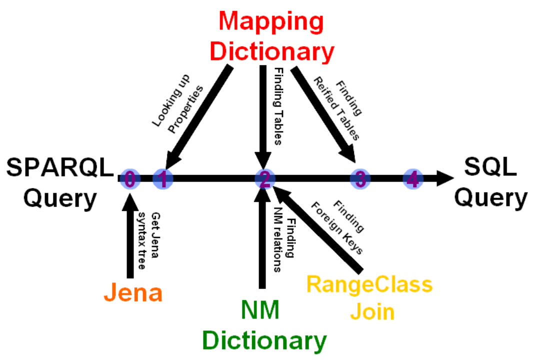


Figure 2.7: Conversion steps of the QC [RSa]

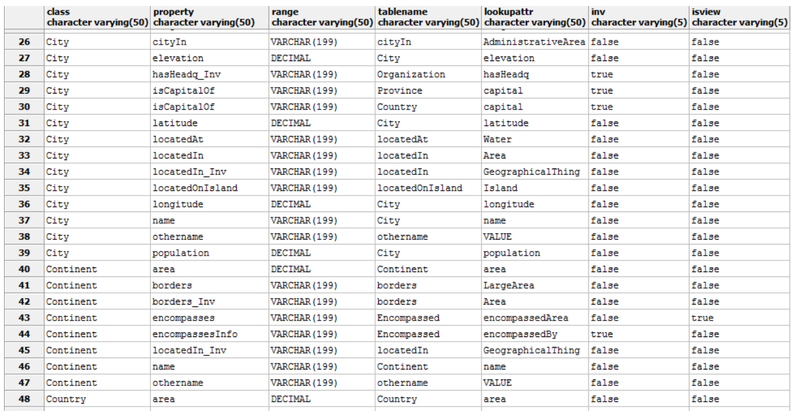
The QueryConverter (QC) [RSa, TH] was created in [RSa] based on [TH].

Its purpose is to convert a SPARQL [Apab] query into a SQL [SQL] query. To do so, the QC uses first Jena [Apaa] to create a syntax tree, from this tree it creates an own tree. The nodes of both trees represent algebraic operation like joins, filters, and selections in form of a spanning tree. The SPARQL variables are transformed into so called ClassVariable (CV), which stores all information about this certain variable, including its possible classes. With this tree the QC gather information about the CV of the tree. The information about the CV is only in the leafs of those trees. These leafs are the individual triples, while the higher nodes represent operations of the query. This means every node which is not a leaf, depends on the information of its children to actually get knowledge about the CV , therefore, every node is checking its information and propagates them to the higher node. This node combines the information and propagates them further up, until it reaches the root. At this point the root contains every available information of all CV, which means it knows all properties which can be assumed to be secure (further details of secure CV s. [RSa] Left Outer Join) and if the CV is defined as a specific class. With this information, it is possible to compute the class of a CV (in some cases this can be even stricter than the class defined in the query) or to notice that it is a literal if it does not have any properties. Then all gathered information of the root will be propagated down to every node, so every node has all information. After the CVs are created the tree computes the needed tables, selection and conditions. This is always done in a backward recursive way, which means it starts at the root and the root requests the CV it needs from its children. The nodes below add their own requests to this and ask their children. This is done for every node until it reaches a leaf where it will decide if the information is available at the node or not. If so it returns a result set with the requested information or otherwise a null value. During the backtracking, the nodes fuse the information of multiple children and add the requested information and the ones request from higher tiers to their SQLNodeInfo which contains the selection, the tables and the condition of a node. Because of this, the nodes can rely on the children to provide the needed information and just selected CV by their names as long as it is not a leaf, since the leaf have to take care about the renaming. For example in the case a CV *?CN*, which is a city name, only the leaf triple would need to know that city names can be found at the table city and the column name because it will select this CV as *Select city.name as '?CN'*. Every other node will just refer it as *?CN*. After the nodes have calculated which information are needed to translate the query, this information will be transformed into a SQL Query. Due to its nature, the node tree structure creates a very nested SQL query because every node selects its children in the *from part*. To reduce this nesting, an optimization will happen for the nodes with just one child, since they can easily be fused by keeping the *from part* of the lower node(s) and the *select part* of the higher node and conjunct the conditions. After this step, the equivalent SQL is ready to be send to the database.

2.3.3 MD - Mapping Dictionary

The *MappingDictionary* (MD) is a method to store ontological metadata and the major difference between an ordinary SQL database and a database created by the DC from an ontology, since it makes it possible to access the knowledge base of the underlying RDF knowledge base [RSa]. With the MD, it is possible to look up the properties, where they are stored and which references the properties have. These references are always toward the URI column of a class which is always the primary key. Since the data have their origin in RDF every class an URI and therefore every *home table* has a URI column which is the primary key. The details of the ontological metadata description are described next. It consists of seven tables and three additional tables (see Sections 3.1.1, 3.1.3, and 3.1.4) as part of this work.

2.3.3.1 md - The main MD Table



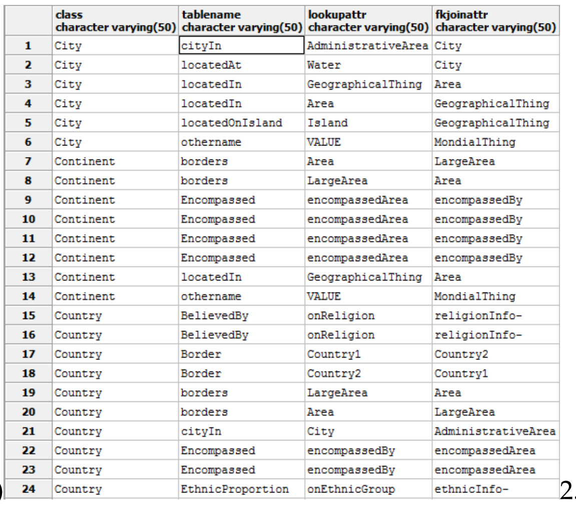
	class character varying(50)	property character varying(50)	range character varying(50)	tablename character varying(50)	lookupattr character varying(50)	inv character varying(5)	isview character varying(5)
26	City	cityIn	VARCHAR(199)	cityIn	AdministrativeArea	false	false
27	City	elevation	DECIMAL	City	elevation	false	false
28	City	hasHeadq_Inv	VARCHAR(199)	Organization	hasHeadq	true	false
29	City	isCapitalOf	VARCHAR(199)	Province	capital	true	false
30	City	isCapitalOf	VARCHAR(199)	Country	capital	true	false
31	City	latitude	DECIMAL	City	latitude	false	false
32	City	locatedAt	VARCHAR(199)	locatedAt	Water	false	false
33	City	locatedIn	VARCHAR(199)	locatedIn	Area	false	false
34	City	locatedIn_Inv	VARCHAR(199)	locatedIn	GeographicalThing	false	false
35	City	locatedOnIsland	VARCHAR(199)	locatedOnIsland	Island	false	false
36	City	longitude	DECIMAL	City	longitude	false	false
37	City	name	VARCHAR(199)	City	name	false	false
38	City	othername	VARCHAR(199)	othername	VALUE	false	false
39	City	population	DECIMAL	City	population	false	false
40	Continent	area	DECIMAL	Continent	area	false	false
41	Continent	borders	VARCHAR(199)	borders	LargeArea	false	false
42	Continent	borders_Inv	VARCHAR(199)	borders	Area	false	false
43	Continent	encompasses	VARCHAR(199)	Encompassed	encompassedArea	false	true
44	Continent	encompassesInfo	VARCHAR(199)	Encompassed	encompassedBy	true	false
45	Continent	locatedIn_Inv	VARCHAR(199)	locatedIn	GeographicalThing	false	false
46	Continent	name	VARCHAR(199)	Continent	name	false	false
47	Continent	othername	VARCHAR(199)	othername	VALUE	false	false
48	Country	area	DECIMAL	Country	area	false	false

Figure 2.8: Example of the md Table

The *md* (table - lower case) is the heart of the ontological metadata description. This table contains the class name, the property, the table name, the range, the domain and if it is a view and if its inverse. For a literal or a class itself, the information in the *md* is sufficient since it just needs the *table name* and the *lookUpAttr* for the selection of this property. If the property has references to other tables or is not in the *home table* (see Section 2.3.3.6) the *md* table is still the central information source. Based on it the other tables can be queried.

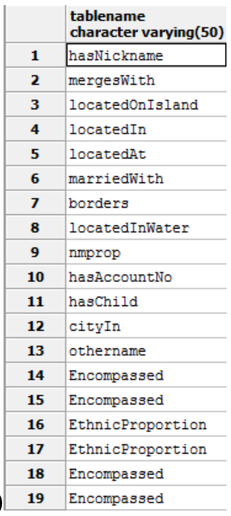
2.3.3.2 nmj & nmname - NM join table & NM names table

1.)



	class character varying(50)	tablename character varying(50)	lookupattr character varying(50)	fkjoinattr character varying(50)
1	City	cityIn	AdministrativeArea	City
2	City	locatedAt	Water	City
3	City	locatedIn	GeographicalThing	Area
4	City	locatedIn	Area	GeographicalThing
5	City	locatedOnIsland	Island	GeographicalThing
6	City	othername	VALUE	MondialThing
7	Continent	borders	Area	LargeArea
8	Continent	borders	LargeArea	Area
9	Continent	Encompassed	encompassedArea	encompassedBy
10	Continent	Encompassed	encompassedArea	encompassedBy
11	Continent	Encompassed	encompassedArea	encompassedBy
12	Continent	Encompassed	encompassedArea	encompassedBy
13	Continent	locatedIn	GeographicalThing	Area
14	Continent	othername	VALUE	MondialThing
15	Country	BelievedBy	onReligion	religionInfo-
16	Country	BelievedBy	onReligion	religionInfo-
17	Country	Border	Country1	Country2
18	Country	Border	Country2	Country1
19	Country	borders	LargeArea	Area
20	Country	borders	Area	LargeArea
21	Country	cityIn	City	AdministrativeArea
22	Country	Encompassed	encompassedBy	encompassedArea
23	Country	Encompassed	encompassedBy	encompassedArea
24	Country	EthnicProportion	onEthnicGroup	ethnicInfo-

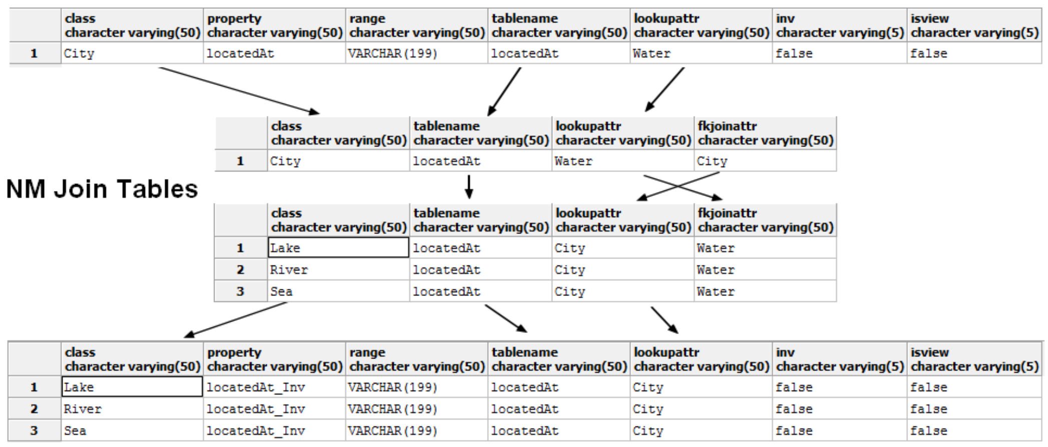
2.)



	tablename character varying(50)
1	hasNickname
2	mergesWith
3	locatedOnIsland
4	locatedIn
5	locatedAt
6	marriedWith
7	borders
8	locatedInWater
9	nmprop
10	hasAccountNo
11	hasChild
12	cityIn
13	othername
14	Encompassed
15	Encompassed
16	EthnicProportion
17	EthnicProportion
18	Encompassed
19	Encompassed

Figure 2.9: Examples of the NMJ (1) and the NMNames (2) tables

NM join table is used when an *nm* join between two classes is needed. If so the *table names* differs from the *home table* of the classes and the *table names* have an entries in the *nm names* table. Then the entry in *nmj* is searched where *class name*, *table name* and *lookUpAttr* are the same as in the *md* table. To create a join another entry is search where the *table name* is the same and the *fkjoinattr* value is the equal to the *lookUpAttr* of the first entry and the *lookUpAttr* equal to the *fkJoinAttr* (see Fig. 2.10).



	class character varying(50)	property character varying(50)	range character varying(50)	tablename character varying(50)	lookupattr character varying(50)	inv character varying(5)	isview character varying(5)
1	City	locatedAt	VARCHAR(199)	locatedAt	Water	false	false

	class character varying(50)	tablename character varying(50)	lookupattr character varying(50)	fkjoinattr character varying(50)
1	City	locatedAt	Water	City

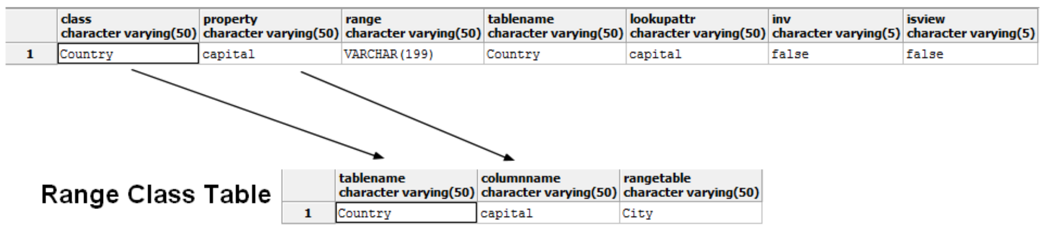
	class character varying(50)	tablename character varying(50)	lookupattr character varying(50)	fkjoinattr character varying(50)
1	Lake	locatedAt	City	Water
2	River	locatedAt	City	Water
3	Sea	locatedAt	City	Water

	class character varying(50)	property character varying(50)	range character varying(50)	tablename character varying(50)	lookupattr character varying(50)	inv character varying(5)	isview character varying(5)
1	Lake	locatedAt_Inv	VARCHAR(199)	locatedAt	City	false	false
2	River	locatedAt_Inv	VARCHAR(199)	locatedAt	City	false	false
3	Sea	locatedAt_Inv	VARCHAR(199)	locatedAt	City	false	false

Figure 2.10: Concept how to create an NM connection - →means pointed entry has to have the same value as the origin entry

As the last information, the *home table* for the other class is needed since up to this point only the *nm* entries are used which refer in *table name* to the nm table, not to their *home tables*. In the original version, the constraint that every *home table* is named after the class it is made of, makes this one easy, since the *class name* is also the *home table* name. In the current version, there is a table which stores the class to *home table* mapping (see Section 3.1.4). With these entries, a join between the *home table* of the CV, the nm table and the other *home table* can be made by using the conditions $hometable1.URI = nmtable.LookUpAttr$ *and* $hometable2.URI = nmtable.FkJoinAttr$.

2.3.3.3 RCTab - Range class table



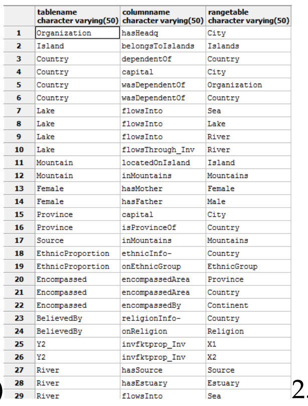
	class character varying(50)	property character varying(50)	range character varying(50)	tablename character varying(50)	lookupattr character varying(50)	inv character varying(5)	isview character varying(5)
1	Country	capital	VARCHAR(199)	Country	capital	false	false

	tablename character varying(50)	columnname character varying(50)	rangetable character varying(50)
1	Country	capital	City

Figure 2.11: Concept of non nm connections construction

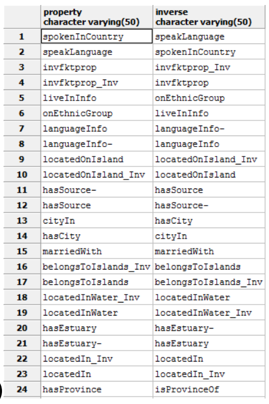
The *RCTab* stores the range classes of object valued properties. The table consists of the columns *tablename*, *columnname* and *rangetable*. The *columnname* is also the name of the property, the *tablename* is also the name of the class itself and the *rangetable* is also the name of the range class. A column rangeColumn is not needed, since every *home table* has a URI column which is the primary key, there is no need to define the referenced column and table explicitly. The *RCTab* is used for cases where the relations is a 1:1 or a 1:n. These relations do have, other than n:m relations, inverse directions and for the query construction always the non-inverse direction is used (see Section 2.3.3.4). To create the query, the join condition $rangeHomeTable.URI = tableName.columnName$ is needed.

1.)



	tablename character varying(50)	columnname character varying(50)	rangetable character varying(50)
1	Organization	hasHeadq	City
2	Island	belongsToIslands	Islands
3	Country	dependentOf	Country
4	Country	capital	City
5	Country	wasDependentOf	Organization
6	Country	wasDependentOf	Country
7	Lake	flowsInto	Sea
8	Lake	flowsInto	Lake
9	Lake	flowsInto	River
10	Lake	flowsThrough_Inv	River
11	Mountain	locatedOnIsland	Island
12	Mountain	inMountains	Mountains
13	Female	hasMother	Female
14	Female	hasFather	Male
15	Province	capital	City
16	Province	isProvinceOf	Country
17	Source	inMountains	Mountains
18	EthnicProportion	ethnicInfo-	Country
19	EthnicProportion	onEthnicGroup	EthnicGroup
20	Encompassed	encompassedArea	Province
21	Encompassed	encompassedArea	Country
22	Encompassed	encompassedBy	Continent
23	BelievedBy	religionInfo-	Country
24	BelievedBy	onReligion	Religion
25	Y2	invfktprop_Inv	X1
26	Y2	invfktprop_Inv	X2
27	River	hasSource	Source
28	River	hasEstuary	Estuary
29	River	flowsInto	Sea

2.)



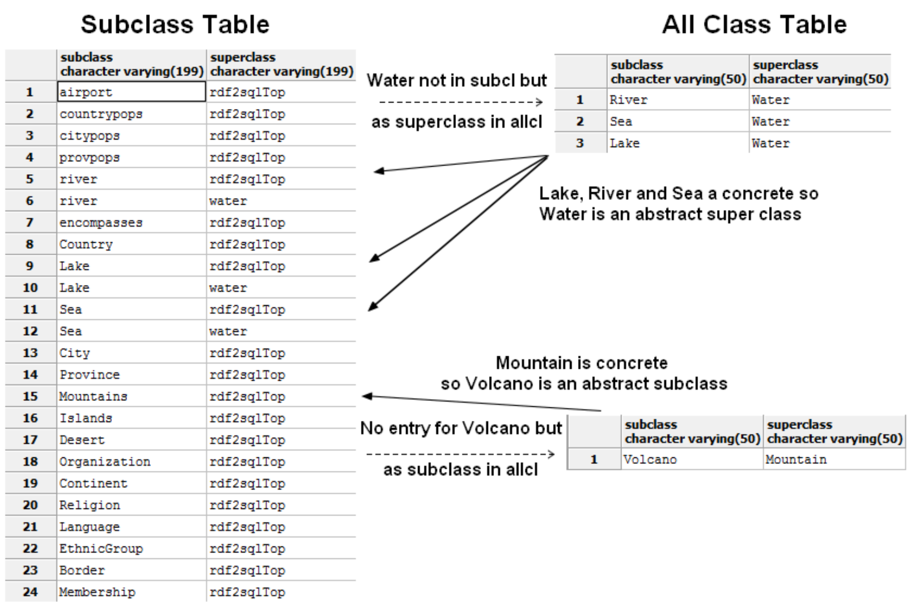
	property character varying(50)	inverse character varying(50)
1	spokenInCountry	speakLanguage
2	speakLanguage	spokenInCountry
3	invfktprop	invfktprop_Inv
4	invfktprop_Inv	invfktprop
5	liveInInfo	onEthnicGroup
6	onEthnicGroup	liveInInfo
7	languageInfo	languageInfo-
8	languageInfo-	languageInfo
9	locatedOnIsland	locatedOnIsland_Inv
10	locatedOnIsland_Inv	locatedOnIsland
11	hasSource-	hasSource
12	hasSource	hasSource-
13	cityIn	hasCity
14	hasCity	cityIn
15	marriedWith	marriedWith
16	belongsToIslands_Inv	belongsToIslands
17	belongsToIslands	belongsToIslands_Inv
18	locatedInWater_Inv	locatedInWater
19	locatedInWater	locatedInWater_Inv
20	hasEstuary	hasEstuary-
21	hasEstuary-	hasEstuary
22	locatedIn_Inv	locatedIn
23	locatedIn	locatedIn_Inv
24	hasProvince	isProvinceOf

Figure 2.12: 1.) Example RcTab Tabelle 2.) Example for the inv table

2.3.3.4 inv - Inverse table

Non-nm relations usually do have two directions. Often relations are defined with proper names in RDF but if not, [propertyName]_Inv is chosen. The inverse table is an 1:1 mapping of the property and the inverse and the other way around. It is used if a relation is defined in the query, for which only the other direction is implemented in the underlying database structure, but also to find the md property entry for the other side of nm relations (see Fig.2.10).

2.3.3.5 subcl - Subclass Table and allcl - All class Table



	subclass character varying(199)	superclass character varying(199)
1	airport	rdf2sqlTop
2	countrypops	rdf2sqlTop
3	citypops	rdf2sqlTop
4	provpops	rdf2sqlTop
5	river	rdf2sqlTop
6	river	water
7	encompasses	rdf2sqlTop
8	Country	rdf2sqlTop
9	Lake	rdf2sqlTop
10	Lake	water
11	Sea	rdf2sqlTop
12	Sea	water
13	City	rdf2sqlTop
14	Province	rdf2sqlTop
15	Mountains	rdf2sqlTop
16	Islands	rdf2sqlTop
17	Desert	rdf2sqlTop
18	Organization	rdf2sqlTop
19	Continent	rdf2sqlTop
20	Religion	rdf2sqlTop
21	Language	rdf2sqlTop
22	EthnicGroup	rdf2sqlTop
23	Border	rdf2sqlTop
24	Membership	rdf2sqlTop

	subclass character varying(50)	superclass character varying(50)
1	River	Water
2	Sea	Water
3	Lake	Water

	subclass character varying(50)	superclass character varying(50)
1	Volcano	Mountain

Figure 2.13: Concept of finding concrete classes

The concept of inheritance is common in RDF and a major part of a well-designed RDF knowledge base but it is not implemented in standard SQL . Therefore the ontolo has the tables *subcl* and *allcl*. The *subcl* table contains all concrete classes and the corresponding super classes and the *allcl* table contains all classes, concrete and abstract and all superclasses of every class. So the *subcl* table is used to find out if a class is abstract which automatically means, there is no *home table* to this class, since only concrete classes do have them. Therefore it must look up in the *allcl* table which sub and superclasses are there and in the *subcl* table if this subclass is concrete and if not if a superclass is concrete. If neither a sub- nor a superclass is concrete there is something wrong and the QC will throw an exception. Often an abstract superclass is used to select multiple classes for example water to get all lakes, seas and rivers (see Section 2.13).

2.3.3.6 Home tables

The term *home table* describes the tables which would contain the primary key(s) or the ID of an entity. Usually, these tables have the same name as the class they contain. The DC will always name the home table like the class it is made for and the other way around the RMC [Run] will name the classes like the table which holds the primary key. So the only case the tables and the class name are not the same is when one of them was renamed. This could be the case if a MD is transformed for another database. In this case, the ontology changed but not the database in the background, so a mismatch between the names could happen (see Section 3.1.4). Therefore the

ClassName table is a mapping to find the *home table* of a class. If no translation was performed on the MD this table holds just two times the class name respectively the table name.

2.3.3.7 Class - extension - tables [Run]

The term class extension table describes a table which has a 1:1 relation to its class, but is not part of the *home table*, even if this would be easily possible by just joining the columns onto the *home table*. In terms of the database normalization this table should not exist, but in practices there are cases where such tables exist and sometimes it makes things for human users actually easier, for example the economics table in Mondial. This table stores all economics related information of a country. It has only a 1:1 connection to the table Country, which means it could be simply joined onto this table without losing any information. However, since it is about a certain self-contained aspect of country the designer of the ER Model decides to outsource this information into an own table.

2.4 String similarity metrics

Similarity metrics calculate the distance/ the similarity of a word to another. In a valid metric precise rankings can be made and similarities can be put in relation to each other. Such metrics have a broad scope of application from search engines, over speech recognition to automatic spellchecking. In this Thesis the similarity metrics are used to calculate the similarity of class and attributes names of different ontologies. The easiest similarity metric is to check if the symbol chains match exactly, but this would not be enough to determine if there might be a connection or just a slight grammatical altering in names. Therefore more sophisticated methods are needed.

2.4.1 Jaccard similarity coefficient

The Jaccard similarity coefficient is originally coined "coefficient de communauté" by the botanist Paul Jaccard. It is used to measure the similarity / diversity of set samples. The closer a Jaccard similarity coefficient is to 1, the more similar two set are. It is computed by just dividing the number of shared elements of the two sets by the total amount of different elements (see Eq. 2.1). [TSK]

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.1}$$

2.4.2 Levenshtein similarity

Algorithm 2.1 Levenshtein Distance algorithm

$$\begin{aligned}
m &= |u| \\
n &= |v| \\
D_{0,0} &= 0 \\
D_{i,0} &= i, 1 \leq i \leq m \\
D_{0,j} &= j, 1 \leq j \leq n \\
D_{i,j} &= min \begin{cases} D_{i-1,j-1} & +0\, if\, u_i = v_j \\ D_{i-1,j-1} & +1\, (replace) \\ D_{i,j-1} & +1\, (Insertion) \\ D_{i-1,j} & +1\, (Deletion) \end{cases} \\
& 1 \leq i \leq m, 1 \leq j \leq n
\end{aligned}$$

The Levenshtein similarity is a measurement of similarity like the Jaccard similarity. However, the Levenshtein similarity is not based on set operations, but on symbol changing. It computes how many deletions, replacements, and insertions are needed to convert one symbol chain into another. It was developed by Wladimir Iossifowitsch Lewenstein in 1965. He himself never published an algorithm for computing this, but the computation can be done by an algorithm like Alg. 2.1. This algorithm returns a matrix which can be read to get the lowest cost value and then calculate a distance with it. Since there are some limitations to the Levenshtein distance one can calculate the minimal (see 3rd on fig.2.14) and maximal possible distance (see 2nd on fig.2.14) and convert this into a value between 0 and 1.

1. The distance is at least the difference of the sizes of the char sequence.
2. The distance is at maximum the length of the longer char sequence.
3. The distance is only 0 if the char sequences are exactly equal. (Metric condition)
4. The distance is not greater than the Hamming-Distance added up the length difference of the sequences.

Figure 2.14: Important bounds of the Levenshtein algorithm

Chapter 3

Analysis

3.1 Adaptation of the QueryConverter (QC)

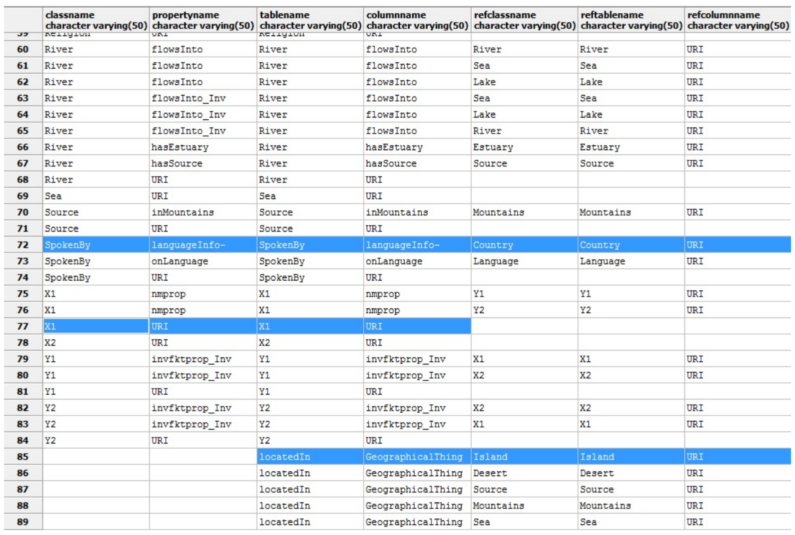
The QueryConverter (QC) was originally developed to work exclusively with databases created by the DatabaseConverter (DC). The DC itself only works with n3 - files and therefore some special cases of the database design could be assumed as given. For example that a single key column named Unique Resource Identifier (URI) exists, that an abstract subclass always is marked in the superclass table in the column *RDFTYPE* with its name or that the *home tables* (see Section 2.3.3.6) are always named as the class itself. In order to widen the range of the QueryConverter (QC) these constraints can no longer be assumed as given. Therefore some additional information must be stored in the MappingDictionary (MD) and the QC has to do some extra steps since some conversions got more complicated without these assumptions.

3.1.1 Multi column keys

The easiest way for maintaining primary keys are IDs, which only contain a unique value but no properties of the class. Aside from that, it is also possible to use one or several attributes of the class which can be assumed as unique. Many databases do not use this concept with good reasons, since one can often not be sure, if there is really no case were two different entities do have the same key values (for example first and last name of a person are generally not unique but the collision rate is low) but it is common enough to have to deal with this possibility. The relational Mondial version, for example, has some multi-column key classes like City. The primary key of City is compounded of the columns Name, Country,and Province. Beside being keys, those columns contain also information about the class itself, therefore properties exist, which refer to those columns. With such a key there arise two major problems:

1. Knowing which attributes are keys
2. Knowing which column has to be joined with which column in the other table

Before in DC generated databases every concrete class had a table with the same name which contains a single column key in the column URI. Because of this condition there was no need to store the information where a foreign key should refer to. With non-URI-primary keys it became necessary to store the name and the table of the key. Further, it became necessary to store which foreign key referred to which primary key. For example, City has the primary keys Country, Province, and Name. In the other tables, the column name is usually used for the own name attribute of the class described by that table and the column where the name of the city is stored is called City. In such cases, the matching of the key columns is no longer trivial.



	classname character varying(50)	propertyname character varying(50)	tablename character varying(50)	columnname character varying(50)	refclassname character varying(50)	reftablename character varying(50)	refcolumnname character varying(50)
59	Religion	URI	Religion	URI			
60	River	flowsInto	River	flowsInto	River	River	URI
61	River	flowsInto	River	flowsInto	Sea	Sea	URI
62	River	flowsInto	River	flowsInto	Lake	Lake	URI
63	River	flowsInto_Inv	River	flowsInto	Sea	Sea	URI
64	River	flowsInto_Inv	River	flowsInto	Lake	Lake	URI
65	River	flowsInto_Inv	River	flowsInto	River	River	URI
66	River	hasEstuary	River	hasEstuary	Estuary	Estuary	URI
67	River	hasSource	River	hasSource	Source	Source	URI
68	River	URI	River	URI			
69	Sea	URI	Sea	URI			
70	Source	inMountains	Source	inMountains	Mountains	Mountains	URI
71	Source	URI	Source	URI			
72	SpokenBy	languageInfo-	SpokenBy	languageInfo-	Country	Country	URI
73	SpokenBy	onLanguage	SpokenBy	onLanguage	Language	Language	URI
74	SpokenBy	URI	SpokenBy	URI			
75	X1	nmprop	X1	nmprop	Y1	Y1	URI
76	X1	nmprop	X1	nmprop	Y2	Y2	URI
77	X1	URI	X1	URI			
78	X2	URI	X2	URI			
79	Y1	invfktprop_Inv	Y1	invfktprop_Inv	X1	X1	URI
80	Y1	invfktprop_Inv	Y1	invfktprop_Inv	X2	X2	URI
81	Y1	URI	Y1	URI			
82	Y2	invfktprop_Inv	Y2	invfktprop_Inv	X2	X2	URI
83	Y2	invfktprop_Inv	Y2	invfktprop_Inv	X1	X1	URI
84	Y2	URI	Y2	URI			
85			locatedIn	GeographicalThing	Island	Island	URI
86			locatedIn	GeographicalThing	Desert	Desert	URI
87			locatedIn	GeographicalThing	Source	Source	URI
88			locatedIn	GeographicalThing	Mountains	Mountains	URI
89			locatedIn	GeographicalThing	Sea	Sea	URI

Figure 3.1: Example for the keys table. Row 72 is an example of a connection of properties via the other table. Row 77 is an example for an entry which represents a primary key and row 85 an example for an *nm table - home table* connection

Therefore the *keys table* is created. This table corresponds largely to the *RCTable* (see Section2.3.3.3) with some necessary extensions and some extensions to make the usage more comfortable and the tables more readable. The table consists of the columns *classname, propertyname, tablename, columnname, refclassname, reftablename, refcolunname* (see Section3.1). The *keys table* contains basically three types of information:

- First of all, there are the primary keys of each class. For primary keys only the columns *classname*, *propertyname*, *tablename* and *columnname* are used. *Classname* contains the name of the class, the propertyname is always URI, the *tablename* is the home table of this class, and in *columnname* the column which contains this key is stored. For multi-column primary keys a single class has multiple primary key entries in the keys tables.
- The 1:N relations use all columns of the table. They are important for multi-column keys since the columns *columnname* and *refcolumnname* are used to match the different key values to which they refer. The columns *classname* and *refclassname* describe the class and the class to which this property refers. The column *propertyname* contains the name of the property. This will only differ from *columnname* after translation and is primarily intended for humans to make checking and debugging easier. *Columnname* stores the name of the

column, which contains the foreign key to the table from *classname.* The column *tablename* contains the table in which the class from *classname* stores the property from *propertyname* in the column from *columnname.* A column *reftablename* is necessary in nm - relations. Other than the md table, the keys table has the actual column to column matching and not the abstract relation stored. This means that the Structured Query Language (SQL) join conditions can be directly be generated without matching other tables. The *refcolumnname* is needed to know with which part of the primary key this column should be matched, since the naming can differ between foreign key and primary key like in the previous example.

- The nm connection are stored in a way that the values *classname* and *propertyname* are empty, since an nm table does not belong to a single class and the values in the columns in *columnnames* and *refcolumnnames* are the foreign and primary keys of the class which refers to this nm table and no valid property of the ontology, therefore they do not even have a qualified name. If the key consists of information holding columns (for example the Country, Province and Name columns of City, which also contains the information of the same named properties) it still is not a part of the ontology to refer this information in the way that it is part of a key structure since the ontology does not care about the information storing structures. The class to which can be refer is stored in *refclassname.*

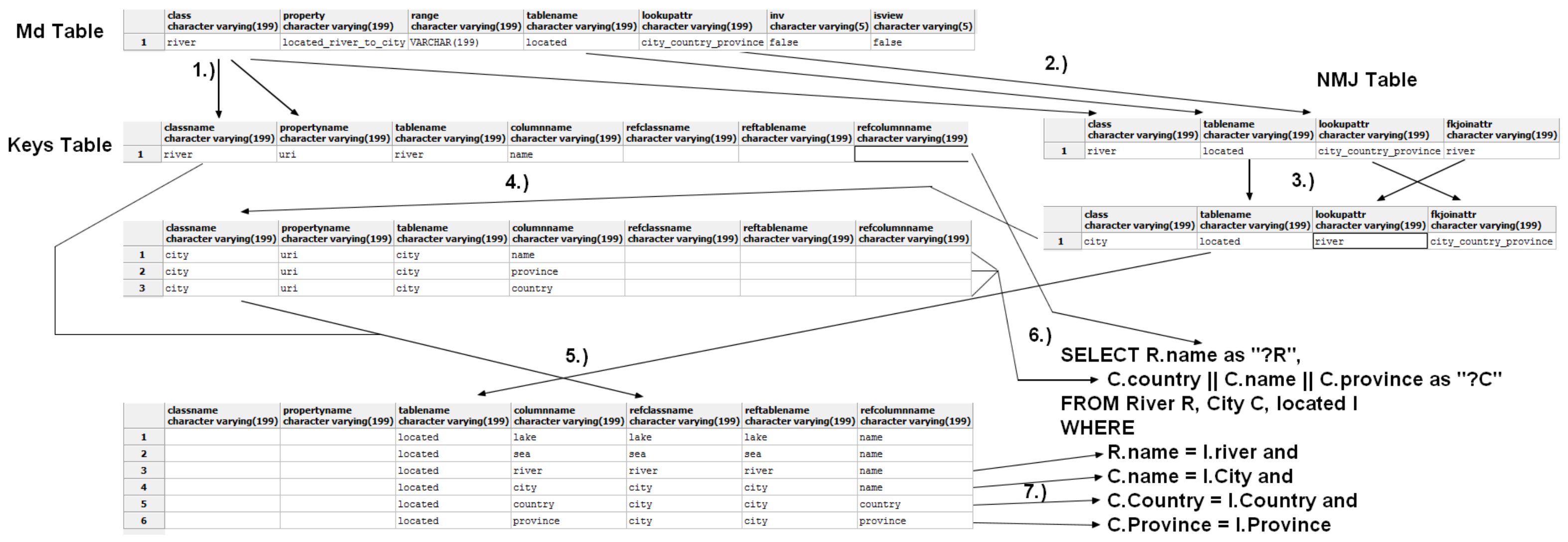
3.1.2 URI generation

In SPARQL Protocol And RDF Query Language (SPARQL) queries it is rather common to match entities with their URIs. Nevertheless URIs are not part of the ontology and therefore can not be part of a result set of a query over this ontology. They will still be used in join conditions. In relational databases without a single ID column like the URI column but with potential multi-column keys, this matching becomes more complex. To handle this problem two solutions become apparent.

On the one hand, this single variable can be split into several variables, each corresponding to one part of the multi-column key. The problem with this approach is primarily the implementation effort, since the QC was not implemented with this problem in mind and it violates the encapsulation of every node, since this would needed a reparsing of the whole tree to add these variables.

On the other hand, a single value could be generated out of the primary keys and make sure that this will be a unique and reproducible ID. The advantage of this approach is clearly that this is invisible to all but the root nodes of the query tree. The disadvantage of this approach is that databases are not good at string operations and it slows down the performance of the QC significantly. For an industrial version of this program the slowing might be the argument against the single value approach, but with the limited work time in mind, the approach with multiple columns fused in one variable was chosen.

To make sure that the combination of values does not result in an equal string by chance, the property names were always put in front of the property value and in alphabetical order of the home table columnnames. For example, the single value keys of key1 = a and key2 = bc would be the same as key1 = ab and key2 = c. With the property names, the multi-column key differs.



Md Table

	class character varying(199)	property character varying(199)	range character varying(199)	tablename character varying(199)	lookupattr character varying(199)	inv character varying(5)	isview character varying(5)
1	river	located_river_to_city	VARCHAR(199)	located	city_country_province	false	false

Keys Table

	classname character varying(199)	propertyname character varying(199)	tablename character varying(199)	columnname character varying(199)	refclassname character varying(199)	reftablename character varying(199)	refcolumnname character varying(199)
1	river	uri	river	name			

	classname character varying(199)	propertyname character varying(199)	tablename character varying(199)	columnname character varying(199)	refclassname character varying(199)	reftablename character varying(199)	refcolumnname character varying(199)
1	city	uri	city	name			
2	city	uri	city	province			
3	city	uri	city	country			

	classname character varying(199)	propertyname character varying(199)	tablename character varying(199)	columnname character varying(199)	refclassname character varying(199)	reftablename character varying(199)	refcolumnname character varying(199)
1			located	lake	lake	lake	name
2			located	sea	sea	sea	name
3			located	river	river	river	name
4			located	city	city	city	name
5			located	country	city	city	country
6			located	province	city	city	province

NMJ Table

	class character varying(199)	tablename character varying(199)	lookupattr character varying(199)	fkjoinattr character varying(199)
1	river	located	city_country_province	river

	class character varying(199)	tablename character varying(199)	lookupattr character varying(199)	fkjoinattr character varying(199)
1	city	located	river	city_country_province

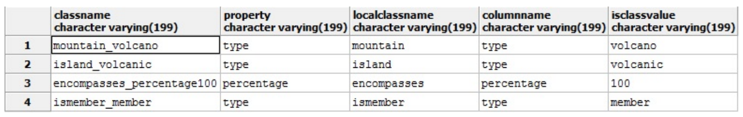
```
SELECT R.name as "?R",
    C.country || C.name || C.province as "?C"
FROM River R, City C, located l
WHERE
    R.name = l.river and
    C.name = l.City and
    C.Country = l.Country and
    C.Province = l.Province
```

1. Ask Mapping Dictionary for information about Class and Property
2. Find the primary key of River in the *Keys Table* by looking for the class and the property URI.
3. Find the *nm table* entry in *NMJ Table* (see Section 2.3.3.2 for details how to find out, that this is an NM connection).
4. Find the second class of the nm connection with $c1.lookUpAttr = c.2.fkJoinAttr\,and\,c2.lookUpAttr = c1.fkJoinAttr\,and\,c1.tablename = c2.tablename$.
5. Use the second class name to find primary keys for it.
6. Use class names and *NM table name* to find foreign keys
7. Use primary keys of both classes to generate URI (see Section3.1.2 for details, here simplified to keep it short)
8. Use foreign keys to create matching with *nm table* columns and primary keys.

Figure 3.2: Flow diagram Key Table - Creating an SQL query which selects all names of Cities which are located at a River and the names of those Rivers

3.1.3 Abstract Subclasses table

In case of abstract subclasses the most detailed entity type tend to hold a column where in some way it is stored that this entity is actually a abstract subclass. A common approach is having a column named something like *type*, which contains the subclass name, if the entity is of a subclass. This approach is also used by DC generated databases. In this cases, the most detailed entity type holds a column named *RDFType,* which contains the name of the subclass or is otherwise null, if it is not a subclass. Another way to deal with abstract subclasses is to have a column for each abstract subclass in the most detailed entity type and use the different columns as boolean values to distinguish, if this entity is of a subclass or not.



	classname character varying(199)	property character varying(199)	localclassname character varying(199)	columnname character varying(199)	isclassvalue character varying(199)
1	mountain_volcano	type	mountain	type	volcano
2	island_volcanic	type	island	type	volcanic
3	encompasses_percentage100	percentage	encompasses	percentage	100
4	ismember_member	type	ismember	type	member

Figure 3.3: Example AbstractColumnTable of the relational Mondial database.

The original QC automatically chooses the way of the DC and takes a look at the *RDFType* column of the *home table* to determine if it is a subclass. With arbitrary databases, a model is needed which is as flexible as possible to handle the different possibilities to store this information. The *abstractSubClCols table* is created to do this. It has the columns *classname, property, localclassname, columnname* and *isclassvalue*. The *classname* contains the name of the abstract subclass and the *localclassname* contains the name of the class to find out which *home table* holds the column were the subclass information is stored. The column *columnname* holds the information of the column which is the decisive factor, if this entity is a subclass and if so, which one. Therefore the *isclassvalue* column stores, which value the column, must have to mark that this entity is of a the mention in *classname* subclass. For example, these values can be “active volcano” or “inactive volcano” if the class names of several classes are stored in one column or it could be “true” if the subclasses marked in a boolean style and the “false” value actually means that this one does not describe a subclass.

Thus, with the *abstractSubclcol table*, the QC now looks up which property is deciding for the classes then make sure that the value of the column *columnname* in the home table of the class *localclassname* is equal to the value of the *isclassvalue* to determine, if the entries are of an abstract subclass or not.

3.1.4 Class name tables

The *class name table* is redundant until the MD is translated. In this case, the name of the classes can change while the database below will be left as it is, so the *table name* will still be the original class name. So the table just consists of the two column *classname* and *tablename* and the QC uses it as a simple name mapping, if the class name or the table name is necessary for the current context.

3.2 SM - SchemaMatcher

The task of the SchemaMatcher (SM) is to find for as many classes as possible from the one ontology corresponding to the classes of the other ontology or database using only the T-Box. Therefore it uses different methods, evaluates the impact of each and combines them to a similarity matrix and then tries to find a matching for each row and column. From an abstract point of view the SM handles a similarity problem and a weighted bipartite graph matching problem.

These problems are solved in six phases, consisting of one or multiple steps (see Table 3.1)

1. *Initialization*: creates the different classes and their attributes, puts them into matrices and establishes the references and inverses.
2. *Independent rating*: calculates the similarity of the names and the structure. Those do not need any previous results to be calculated. Each step produces its own matrix.
3. *Matrix Combination:* combines all the matrices from the previous step into a single matrix.
4. *Dependent rating:* modifies the combined matrix according to the class trees, sub classes and/or remove. Results in a modified matrix.
5. *Matching:* Calculates an alignment out of the similarity matrix.
6. *Translation and Transformation:* Use the class alignment to repeat phase 1 - 5 for the attributes of each aligned class. Then uses the alignment to transform the Focus Data Base (FDB) MD into an MD with the ontology of Ontology Data Base (ODB) and the database structure of the FDB.

Name	Phase	Function	Reference
ModelCreation	Model Creation	Creates a RelationalModel and an MD for an existing relational database	[Run]
DatabaseConverter	Model Creation	Creates a relational database and an MD for an existing for relational Model	[RS13]
Initialization	Initialization	Creates all Classes, Attributes and connections between those out of a MD	3.2.2
Synonym	Initialization	Adds synonyms from a given thesaurus to the class and attributes names	3.2.3.1
Name	Independent	Calculates the similarity score between names	3.2.3.1
Structure	Independent	Calculates the similarity score between data types	3.2.3.2
Foreign Key	Dependent	Compares the foreign keys of two objects	3.2.5.1
Remove	Dependent	Removes objects without any score above threshold	3.2.5.3
Inheritance	Dependent	Compares inheritance trees and subtrees	3.2.5.2
Nested	Dependent	Searches for classes in the attributes	3.2.5.4
Priority Permutation	Matching	Calculates best permutation out of the most promising candidates	3.2.6.2
Habre [Hab61]	Matching	Variance based greedy matching	3.15
Translation	Translation	Translates the FDB MD into a ODB compatible one.	3.2.7
QueryConverter	Translation	Translates a SPARQL Query into a SQL Query for an DC database	[RSa]
RODITest	Testing	Executes the QC the mondial queries of the RODI benchmark [PBJR$^+$15] and evaluates the results	[PBJR$^+$15]

Table 3.1: Overview of all matching steps

3.2.1 Notations

In this section the following notations were used, if not explicitly written otherwise :

Ontologie DataBase (ODB) is the database which provides the ontology.

Focus DataBase (FDB) is the database which has to be translated to the ontology of ODB.

R is a set of classes or attributes of the ODB, depending on whether the formula is used in the class or the attribute level. In matrices **R** is always the rows.

C is a set of classes or attributes of the FDB, depending on whether the formula is used in the class or the attribute level. In matrices **C** is always the columns.

x is an element of **R**. Depending on the context of **R, x** is either a class or an attribute.

y is an element of **C**. Depending on the context of **C, y** is either a class or an attribute.

Likelihood Object is the similarity of two classes or attributes, with references to those.

M is a SchemaMatchingMatrix consisting of **likelihood objects** with references to **x** and **y**. Every row is for one class or attribute out of **R** and every column is for one class or attribute of **C**.

$M_{x.y}$ is the reference to the **likelihood object** for the classes or attributes **x** and **y**.

$likelihood(M_{x,y})$ the similarity of the **likelihood object** of the classes or attributes **x** and **y**.

SM is the set of all M.

a_{ref} is a class, which has a relationship with the class a and is referenced by a foreign key of a (including inverse direction)

$R \rightarrow C$ is a matching from the space/ontology R to the space/ontology C.

$x \mapsto y$ means x matched with y, this relation is bidirectional.

3.2.2 Initialization

The initialization creates prefab matrices for all instances of the problem. This means one matrix for all classes of ODB against all classes of FDB and one matrix for each combination of classes from the ODB and FDB. All these matrices have the size $|R| \times |C|$. The size of R and C is respectively to the current context either the amount of classes from the one or the other ontology or to the amount of attributes the current class has. Additional to the matrix initialization also the classes and attributes themselves were initialized, means they got their names, ranges, references, inverses, and their inheritance tree. All those information are extracted from the MD of the ODB respectively the FDB. Those prefab matrices are copied in each step and filled with values, this results in far less effort than creating a new matrix, new attributes and new classes from the database every single time. Technically the prefab matrices would be enough to store the results and compute the values for each following step, if the step overrides the results of their predecessor. However, creating copies gives the user the opportunity to understand what has happened in each step and how every value is computed, which is under normal circumstances much more valuable.

3.2.3 Independent rating

The independent ratings are steps which can be executed without any further steps except initialization and do not influence each other. This also means that there is no real meaning to the execution order of steps from this class. These steps, however, can be considered the most important ones for the rating, since the other steps depends on the results of them and errors will be carried on to other steps.

3.2.3.1 Name Rating and Synonyms

Comparing the names of classes and attributes is the most naive but also the usually most successful way to match databases. Like most steps, the Name rating distinguishes between the matching of a class or an attribute.

In case of matching attributes the matching score will be calculated just on the name of the attribute. First it is checked if the name strings of the objects are exactly the same (all comparisons are made in lower case letters) this is the only way to make an 1.0 score. For every other evaluations at most a score of 0.95 is possible, to make sure that exact matches are still slightly better. Then it is checked if one name contains the other name. If so the score is at least 0.5 but if the more advanced matching techniques got a higher score, the higher score will be the final score. For comparison besides exactly the same and one is part of the other, the matching score is the maximum of the Jaccard token matching (see Section2.4.1) , the Levenshtein similarity (see Section 2.4.2) and the Inverse Levenshtein similarity in the matching score (see Equation 3.1). These three methods work best with different kinds of similarities and give a better overall similarity score if all used. Clearly, these methods are rather computation intensive and slow down the performance noticeable, especially if all three are executed. However, in this approach precision is a much more critical factor than time.

$$match(a,b) = filter(max(JaccardTokenMatching(a,b), max(Levenshtein(a,b), InverseLevenshtein(a,b))) \quad (3.1)$$

For classes, the name of the two classes are compared in the same way as the names of attributes, but as long as at least one class has attributes also the attributes of this two classes are taken in account (see Equation3.2).

$$score_{x,y} = filter(\underbrace{match(name(x), name(y)) * w}_{class\ name\ matching}) \tag{3.2}$$

$$+ \underbrace{\frac{\sum_{i=0}^{|Attribute(x)|} habre(match(name(Attributes(x)_i), name(Attributes(y))))}{max(|Attributes(x)|, |Attributes(y)|)} * (1-w)}_{attribute\ name\ average\ matching} \tag{3.3}$$

The class name matching part is the same as Equation 3.1 weighted by w. Then all attributes of the ontology R are matched with all attributes of C and a matching will be created with the variance matching by Habre (see Section 3.2.6.1). This matching technique is chosen even if it tends to mix up some matchings but the actual matchings do not matter at that point only the similarity score are important and since every class and attribute will be matched with each other it has to be done in a efficient way. Therefore this score is rather a good estimation. The sum of the matched similarity scores will now be divided by the greater amount of attributes of class x or class y. This creates a score penalty if the number of attributes is not equal, otherwise a class with just one attribute which fits perfectly to one of the other class would have a great score even if all other attributes of the second class do not have a matching partner at all. This attribute average matching score is now weighted by $1 - w$ to get a valid probability between 0 and 1. The value w is the weighting between the name of the class and the names of the attributes of a class. w is by default set to 0.6, i.g. the class name is slightly more important than the attribute name average. But this also means that two classes with the same name can still have a score of w if none of the attributes are matching. Therefore w should be coordinated with the threshold of the removing step (see Section 3.2.5.3). If $t_{remove} \geq t_{name}$ it can make sure that at least one attribute has to match or that there a non at all, since the score can not be higher than w without attributes.

The name matching itself creates some additional problems to handle: Words consiting of about 26 characters (depending on the language of the ontologies) and especially longer words will very likely have some shared sequences or characters. Therefore words without a real connection will still have relatively high score sometimes. Because of this matching between words which are not actually similar, scores which range from 0 up to 0.5 should not be taken into consideration for further calculations. To remove this flat score decreasing, a high pass filter is implemented for the string matching, which sets scores to 0 if they are lower than the threshold value. This value is set to 0.6 which proofed to be a solid value by testing, but actually it is rather arbitrary. For classes this filter is implemented as well, since there are often attributes which are nearly in every class, like name, and low scores because of this attributes do not help in evaluating the usefulness of the matching, so it also set to 0.0 if it is too low.

Another problem are synonyms. By only looking at the characters it is not possible to find out, if a different word could actually mean the same. Therefore a thesaurus [The] is needed to add to every class and attribute name also the synonyms and then the whole procedure is done for every synonym (the name itself will be added as a synonym as well) from the one class/attribute of R with every other class/attribute of C.

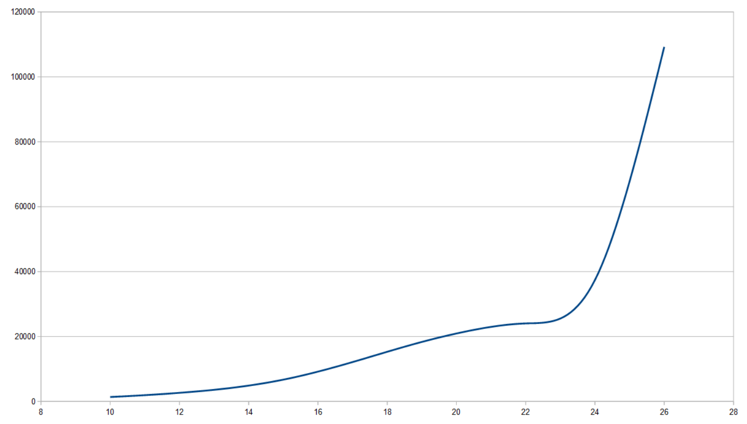


Figure 3.4: Name matching time for mondial_rdf2sql_standard to rel_mondial. Average over 50 runs on an average home computer. X-axes max word length/Y - axes time in milliseconds

Especially with synonyms, the computation time rises exponentially with the number of synonyms, but also with the length of the single words (see Section3.4). To keep the computation time reasonable it is possible to set a maximum word length value $maxL$. If a word is longer than $maxL$ it must be shortened. No matter which part of the word is cut out, information will always be lost. Since human reading of known words works primarily in reading the prefix and suffix of the word to make the grammatical adjustments and then remember the middle part of the word as a pattern the characteristic part of the word is rather the middle one, it shows better results when cut the front and end. Especially in English the really long names consist out of several words, so this technique should rather be used for trying and debugging than for actual runs.

3.2.3.2 Structure rating

The structure rating calculates the similarity based on the data type of the attributes of a class. Therefore it differentiates between seven structure classes

- Literals
  - Text all kinds of character sequences which do not fit into anything else
  - Numbers all kinds of numeric values
  - Dates all columns which use the date type of the database language
- References/Objects
  - 1:N & 1:1 all attributes which refer to home tables
  - NM Connection all attributes which refer to an NM table

  - Inverse all attributes which are inverse functional

- Other special data structures (for example the geographical coordinates "GeoCoord" from Mondial_Rel consisting of longitude and latitude pairs)

For each matching pair x, y the step calculates the score like Equation 3.6. Basically, all attributes will be assigned to one of the structure classes and then for every subclass the lower amount will be divided by the greater one. If the amount is equal the rating is 1, otherwise the rating is lower. The sum of all subscores is divided by the cardinality of the set of structure classes.

$$Is_{type}(X) = X \cap T \ : \ \forall t : Type(t) = type \rightarrow t \in T \tag{3.4}$$

$$subScore_{x,y}(at) = \begin{cases} \frac{min(|Is_{at}(Attributes(x)),|Is_{at}(Attributes(y)|}{max(|Is_{at}(Attributes(x)),|Is_{at}(Attributes(y)|} & If \ : \ |Is_{at}(Attributes(x))| \neq 0 \lor |Is_{at}(Attributes(y))| \neq 0 \\ 1 & If \ : \ |Is_{at}(Attributes(x))| = 0 \land |Is_{at}(Attributes(y))| = 0 \end{cases} \tag{3.5}$$

$$score_{x,y} \quad = \quad \frac{\sum subscores(x,y)}{7} \tag{3.6}$$

3.2.4 Matrix combination

The independent steps are all based on the initial matrix with only -1 as values. If now all independent steps were executed the dependent steps need matching matrices other than the initial matrix, but with actual similarity values. Therefore a method to combine all the SchemaMatchingMatrix (SMM) from the independent steps are needed. So this step has to be done at the transition from the independent steps to the dependent ones.

3.2.4.1 Multiplicative combination

It is a rather standard approach to combine probabilities by just multiply every value (see Equation 3.7). The first problem now is that the values will decrease or at best stay the same with every additional step. This can be solved by using normalization techniques but two even greater problems still exist. First, the influence of every matrix in this approach is always the same, but in general not every method works as good as every other method. Because of this there should be a weighting mechanism. But adding a weighting value would be exactly like adding an additional similarity value, because of the commutative law. The second problem is that as soon as only one similarity value of a single step is zero, there is no chance to recover from this at all. For example, if a class name is used, which is a not known synonym of the other class name, this will lead to a similarity value of zero which means, that these classes could not be matched, even if the scores of the other steps are extremely good, since it will always stay zero. So the multiplicative score of the classes $x \in R$ and $y \in C$ over a set of matching matrices SM calculates:

$$score_{x,y} = \frac{\prod_{i=0}^{|SM|} SM_{xy}^{i}}{|SM|} \tag{3.7}$$

3.2.4.2 Weighted Additive combination

To overcome the problems of the multiplicative combination the weighted additive combination is used. It has the clear advantage, that a single step failure does lower the similarity value, but does not put it to zero without any chance of recovery. Further it gives the opportunity to weight every single step, so it can have an influence depending on its effectiveness. So the weighted additive score of the classes $x \in R$ and $y \in C$ over a set of matching matrices SM calculates:

$$score_{x,y} = \frac{\sum_{i=0}^{|SM|} \left(SM_{x,y}^{i} * weight(SM^{i}) \right)}{\sum_{i=0}^{|SM|} weight(SM^{i})}$$

The weight could be calculated in different ways. In this thesis, there are only the *matrix sharpness evaluation* types (see Section3.2.4.3), but there might be some other methods to do so as well.

3.2.4.3 Matrix evaluation

To evaluate the quality of a similarity matrix in terms of their matching ability, the term *matrix sharpness* was defined. *Sharpness* describes the difference between the highest value of a row or column and the average value without the best value. This value is an important measurement, since a matrix with one 1 in each row and each column and the rest 0 would be the perfect matching base and also would have the highest possible sharpness, while a matrix with a *sharpness* of 0 would be one with all values equal, which also does not help the matching at all.

$$sharpnessRowWise(M,x) = \begin{cases} max(likelyhood(M_x) - \frac{\left(\sum_{i=0}^{|M_x|} likelyhood(M_{x,i})\right) - max(likelyhood(M_x))}{|M_x|-1} & If|\, M_x| > 1 \\ 1 & else \end{cases} \tag{3.8}$$

$$sharpnessColumnWise(M,y) = \begin{cases} max(likelyhood(M_{,y})) - \frac{\left(\sum_{i=0}^{|M_{,y}|} likelyhood(M_{i,y})\right) - max(likelyhood(M_x))}{|M_{,y}|-1} & If\,|M_{,y}| > 1 \\ 1 & else \end{cases} \tag{3.9}$$

$$sharpnessCellWise(M,x,y) = \frac{sharpnessRowWise(M,x) + sharpnessColumnWise(M,y)}{2} \tag{3.10}$$

$$sharpness(M) = \left(\frac{\sum_{i=0}^{|M_{,y}|} sharpnessRowWise(M,i)}{|M_{,y}|} + \frac{\sum_{i=0}^{|M_x|} sharpnessColumnWise(M,i)}{|M_x|} \right) /2 \quad (3.11)$$

With the functions (3.8)&(3.9) one has two opportunities to calculate the *sharpness*. Either using Equation (3.10) for each single cell or using 3.11. The cell wise method (3.10) has the big problem, that it can not differentiate between a medium value with bad other values or a good value with medium other values. But in the case of the matching, a higher score is much more valuable, since the medium score is still quite likely to be a wrong matching. The overall (3.11) method on the other hand has problems to treat every object within the matrix the same. This might solve the problem of 3.10 but in cases where some object can be identified very well by the method and some not at all it flattens the influence to a medium score with giving the right ones not the influence they would need and wrong ones the influence they do not deserve.

After testing both methods with some different database configuration it shows, that there was no case where the results of (3.11) is not better or equal then the results of (3.10).

3.2.5 Dependent rating

Dependent ratings modify the similarity matrices according to the relational structure, the nested class structure, or removes elements, if they do not reach a certain score. Other than the independent rating, these methods are context sensitive, therefore they have to be performed after the independent ones, so a context exists. These steps do not copy the prefab matrix but the existing matrix of the combination instead. Before the first dependent step is executed all matrices calculated by the independent steps were combined according to the chosen matrix combination method (see Equation 3.7) and used as basis for further calculations of the first dependent step. The other dependent steps use the matrix of the previous one. This means it is not necessary to combine the matrices after this type of steps again and also, that in this case the execution order actually makes a difference in results especially for the removing step.

3.2.5.1 Foreign key rating

The foreign key rating uses the foreign key references to influence the similarity scores. This score calculates differently, depending if the step is called with classes from the SchemaMatcher itself or with attributes from the translation step. Equation3.12 is used during the call of the step itself and the other is used during matching attributes with all classes already matched in the matching steps (see Section3.2.6). The main difference in these cases is, that if the classes $x \in X$ and $y \in Y$ are not already matched but only have similarity scores in the similarity matrix M. The foreign key matching does not know if a relationship from x to a class $x_{ref} \in X$ is actually the corresponding relationship from y to class $y_{re} \in Y$ therefore it can just use the existing similarity score of equation3.12 to improve or lower the score of $x \to y$.

$$subScore(x_i, y_j) = \begin{cases} max(M(reference(x_i), reference(y_j), M(x_i, y_j)) & If\ (reference(x_i) \neq \emptyset) \wedge \oplus(reference(y_i) \neq \emptyset) \\ 1 & If\ reference(x_i) \cup reference(y_i) = \emptyset \\ 0 & If\ (reference(x_i) \neq \emptyset) \oplus (reference(y_i) \neq \emptyset) \end{cases} \tag{3.12}$$

But if the matching $x \rightarrow y$ and $x_{ref} \rightarrow y_{ref}$ are known, the similarity score is no longer needed and it can make a boolean decision if this matches or not (see Equation3.13).

$$subScore(x_i, y_j) = \begin{cases} 1 & reference(matched(x_i)) = matched(reference(x_i)) \\ 0 & reference(matched(x_i)) \neq matched(reference(x_i)) \end{cases} \tag{3.13}$$

In both case the overall score for a class computes like equation 3.14 but depending on the point of calling equation3.13 is used if no matches are available and equation 3.13 is used if so.

$$score_{x,y} = M(x, y) * w + \frac{\sum_{i=0}^{|Attributes(x)|} subscore(x_i, y)}{|Is_{nmconnection}(x) \cup Is_{functionalConnection}(x)|} * (1 - w)\ : 0 \leq w \leq 1 \tag{3.14}$$

w is a variable to weight the influence on the overall score.

3.2.5.2 Inheritance rating

This step is only useful in cases where two ontologies are used with good Resource Description Framework (RDF) abstract classes usage or another shared detailed inheritance structure. It creates a set F for all classes with all super classes and sub classes, called*family* . Then it compares the *familiy* of each class of R with all of C. Here again, the problem is that one does not really know the mapping $R \rightarrow C : x_R \mapsto x_C$ so it orientates on the matching scores of previous steps and that is the reason why this method is dependent. Also it can only be used for classes, since the attributes do not have any super or sub-attributes. The score is calculated by computing the family sets of x_R and y_C then the mapping $R \rightarrow C : x_R \mapsto x_C$ and then multiplying the intersection of $x_C$and $y_C$by 2, since the sum of both families is at least two times higher even if $habre(F_{x_R}, F_{y_c}) = F_{y_C}$and then divided by the sum of $|F_x| + |F_y|$

$$\begin{aligned} F_X &= superclasses(x) \cup subclasses(x) \cup x \\ F_y &= superclasses(y) \cup subclasses(y) \cup y \\ score_{x,y} &= \frac{|habre(F_{x_R}, F_{y_C}) \cap F_{y_C}| * 2}{|F_x| + |F_y|} \end{aligned}$$

This step provides much more potential, for example reasoning like it is done in LogMap [CGJR11]. But since derived ontologies only have none to very basic class structures this step can only provide little information and only for very specific classes, so there was not spend to much effort into it.

3.2.5.3 Removing step

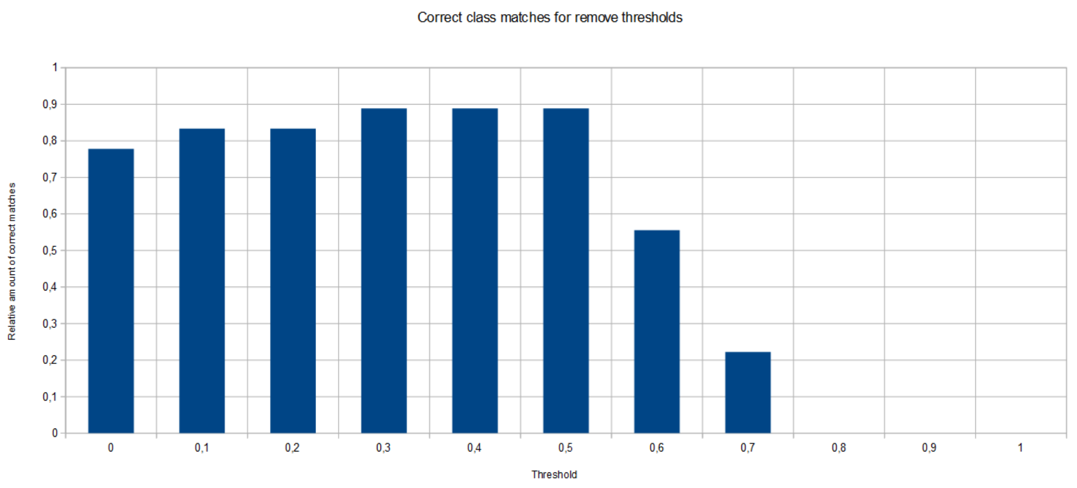


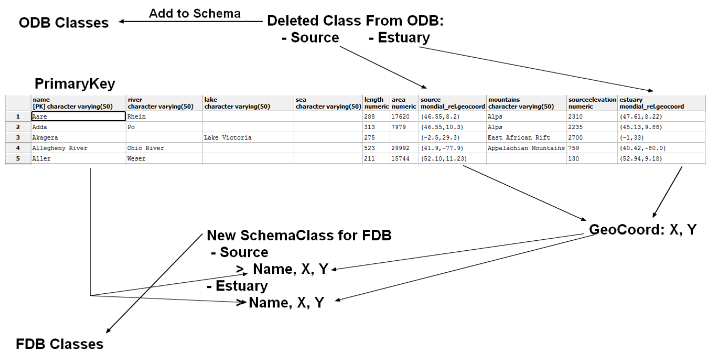
Figure 3.5: Influence of the threshold - on RDF2SQLMondial without abstract classes to rel_Mondial

If there is an object for which no mapping could be found, it is necessary to remove this obsolete classes and attributes. This usually happens often, since matching two identical databases does not make much sense and the mapping algorithm does not produce perfect results. Sometimes this can improve the results of the matching steps slightly (see Figure 3.5) and always improves the performance, since with fewer objects, there are less wrong matching possibilities and less computation to be done. The removing step removes every object in a matching matrix which does not have a similarity value with any partner over the threshold t with $0 \leq t \leq 1$. Without respecting any matching which means if $\exists y \in C : \exists x_1, x_2 \in R : x_1 \neq x_2 \wedge likelihood(M(x_1, y)) > t \wedge likelihood(M(x_2, y)) > t$ none of the x would be removed even if it obvious that only a single x could be matched to the y . But deciding which matching is the correct one is not part of this step. The t could stay the same, but the similarity values for attributes are usually weaker, since class similarity values are based on more components, therefore the score is more averaged and even bad scores tend to reach higher values. Therefore t is divided by two by default when the step is used for attributes.

$$remove(M,t): \quad M \to M_r \quad M_{x,y} \mapsto \begin{cases} M_{x,y} & If\, \exists y : likelihood(M_{x.y}) \geq t \vee \exists x : likelihood(M_{x.y}) \geq t \\ & Else\, likelihood(M_{x,y}) < t \end{cases}$$

For the final results, the influence of this step is rather low (see Fig. 3.5) as long as the threshold is not too high. This mainly depends on the matching algorithms. These are always greedy algorithms which prefer the high score values and this behavior results pretty much in the same then the RemoveStep does it itself, the main difference is that the matching steps will match those classes with any leftovers and this will pretty likely be completely wrong, while the RemoveStep will just delete those classes. In the end, this does not make a correct matching but it makes the other matched classes more reliable which is also important since you do not have to double check every working matching this much anymore.

3.2.5.4 Nested class Step



	name [PK] character varying(50)	river character varying(50)	lake character varying(50)	sea character varying(50)	length numeric	area numeric	source mondial_rel.geocoord	mountains character varying(50)	sourceelevation numeric	estuary mondial_rel.geocoord
1	Aare	Rhein			288	17620	(46.55,8.2)	Alps	2310	(47.61,8.22)
2	Adda	Po			313	7979	(46.55,10.3)	Alps	2235	(45.13,9.88)
3	Akagera		Lake Victoria		275		(-2.5,29.3)	East African Rift	2700	(-1,33)
4	Allegheny River	Ohio River			523	29992	(41.9,-77.9)	Appalachian Mountains	759	(40.42,-80.0)
5	Aller	Weser			211	15744	(52.10,11.23)		130	(52.94,9.18)

Figure 3.6: Schema of finding nested UDT classes in the Focus DataBase (FDB) out of deleted ones from the Ontology DataBase (ODB)

Sometimes a class on its own exists only in one of the ontologies and is a User Defined Type (UDT) nested in a column of another class (host class) in the other ontology. For example, the classes $estuary$ and $source$ only exists in Mondial_RDF2SQL but not in Mondial_rel. In Mondial_rel these classes are nested in the UDT *geocoord* in the columns $source$ and *estuary* of the $river$ table. During the similarity values calculations, those classes should have a quite weak score and were removed at the RemoveStep (see Section3.2.5.3). Removed SchemaObjects are not totally lost; they are rather stored in a list and no longer in the SMM. The nested class step uses this list to check all classes (removed and not removed) of the other ontology for an attribute with the same name as the removed class. If there are no or several such attributes the step will drop this class otherwise the single attribute will be investigated. If it is a number or a text, it is considered to be an abstract subclass and gets an entry in the MD as such. If it is a UDT, a new SchemaClass is created with the attributes of the UDT and the keys of the table which holds the UDT . This class will be added to the SMM and already matched with the one of the other ontology. The new Class gets a likehoodScore of 1 for the matched class and 0 for all other (see Figure 3.6).

ClassName	PropertyName	TableName	Range	Domain	LookUpAttr	Inverse
HC.ColumnName	UDT.AttributeName	HC.Table	UDT.Attribute.Range	UDT.Attribute.Domain	HC.ColumnName + "." + UDT.Attribute.Name	false

Table 3.2: MD Entry - HC = host class, NC = Nested Class

As the last step, the MD of the FDB has to be updated with the new classes created by this step. Therefore every attribute creates an MD entry with the table value of the host class, but the class name is the one of the nested class and the property is actually the column name of the host class where the nested class is stored connected with the attributes name of the UDT (see Table3.2).

This step has to be executed after the RemoveStep, otherwise the removed classes lists are empty.

3.2.6 Matching

The previous steps do not calculate a matching for the class matrix (some do for underlying matrices), but at this point there are only several similarity matrices. These matrices can now support the decision which $x \in R$ matches to which $y \in C$. So the matching step performs the mapping function and gives a matrix only consisting of 0 and at most a single 1 per row and column back. This matrix format is the only one the next step can use (see Section 3.2.7).

The matching is a bipartite weighted graph matching. There exists some algorithms to solve this problem but many common algorithms are not usable, since they often have the condition that there must be a perfect matching or at least the matrix must be a squared one. Both is in general not the case. Another condition different from the general problem of graph matching is that there is only one right matching or none. Therefore this problem can not be seen as a common knapsack problem (further readings [SM90]), since it is not directly about the maximization of the score but getting as much high value matching as possible. So for this case greedy algorithms are the method of choice, since that is exactly what they do.

3.2.6.1 Greedy variance based matching - Habre matching

This method is designed after [Hab61] and is only slightly altered. The idea behind it is to find anomalies in the variance of the matrix to then greedily chose matching pairs. To do so, the score of a single cell is added with an overall average of the matrix and then subtracted by the average of this row and this column. This is done for every cell of the matrix. Next, the cell with the highest score which row and column are not already used will be chosen. This step repeats until there are no unused row or columns left (see Section 3.1). (As a reminder: μdenotes the variance)

$$habreScore_{x,y} = likelihood(M_{x,y}) - \mu(likelihood(M_{x})) - \mu(likelihood(M_{,y})) + \mu(likelihood(M))$$
(3.15)

Algorithm 3.1 Habre Variance based Matching

```
V = M;
i = 0;
foreach(cell c in V){
    V[c.x,c.y] = M.habreScore(c.x,c.y);
}
while(i < V.height & i < V.width){
    max = getMaxUnblockedCell(V);
    V.blockColumn(max.y)
    V.blockRow(max.x);
    M.setRowToZero(max.x);
    M.setColumnToZero(max.y);
    M[max.x,max.y] = 1.0;
    i++
}
```

This algorithm is extremely fast compared to the *Priority permutation matching* (see Section 3.2.6.2) algorithm, but the matching quality is worse, since it tends to mix up some rather obvious matchings. Therefore the Habre algorithm is not used for final matchings but to speed up attribute matchings for the class matching, since this has to be done for each attribute of $x \in R$ with each attribute of each $y \in C$. The class matching itself and the matching of the attribute of already matched classes is then done by a more reliable algorithm.

3.2.6.2 Priority permutation matching

The priority permutation matching is based on the idea to check every possible permutation of matching combination. This does of course not really work in terms of computational complexity (For Mondial_rdf2sql with abstract classes are 80! $\approx 2,6 * 10^{118}$permutations possible), further not every permutation is a valid combination since an object can only be matched once. Therefore a greedy attempt is necessary to reduce the computational effort. To do so every $x \in R$ is checked for it highest similarity value to $y_1 \in C$ and maximum of n next highest other object $y_{2...n} \in C$ with $y_k \geq y_1 * w \, : \, 0 \leq w \leq 1$ or less if there are no close competitors. Next, every permutation is checked for their overall score. Therefore a $|C|$ digit long number in an n digit system is generated and then counted from zero to its maximal value. Every digit represents a $y \in C$ and the value of this digit is the position in the score ranking which is used for the try. I.g. every possibility is tried out. For example if there were $|C| = 4$ and $n = 2$ the first number would be 0000, i.g. that for all classes a matching is tried with only the highest similarity scores. The next try would be 0001, i.g. for y_1, y_2, y_3 it would still be the matching with the highest score but for $y_4$the second best one is chosen. From this set of chosen matching the one with the highest score is matched first and marked in which row and column this matching happens. If one of the following matches wants to use either the row or the column, this one would not be used in this attempt and the score matching will be calculated with 0 for this object.

Algorithm 3.2 Priority Permutation Matching - Pseudocode

```
List listOfBestLists = new List();
For(i = 0; i < |R|;i++){
    listOfBestLists.add(getBest(n));
}
counter[] = IteratorOnNDigitSystem();
List BestPathList = new List();
double BestPathScore = 0;
while(counter.next()){
    List PathList = new List();
    double PathScore = 0;
    usedX = new boolean [|R|];
    usedY = new boolean [|C|];
    for(c = 0; c < listOfBestLists;c++){

       PathList.add(ListOfBestLists.getObject(c).getScore(counter[c]);
    }
    PathList.sort();
    foreach(Cell c in PathList){
       if(!usedX[c.x] && !usedY[c.y]){
          usedX[c.x] = true;
          usedY[c.y] = true;
          PathScore = c.score;
       }
    }
    if(PathScore > BestPathScore){
       BestPathScore = PathScore;
       BestPathList = PathList;
    }
}
return BestPathList;
```

3.2.7 Translating

When this step is reached, all other steps were at least performed once and the class matching can be considered to be done, but not the attribute matching inside the class pairs. To save resources the attribute matching is only be done after the class matching, otherwise it had to be done for each possible combination of classes. As mention earlier matching class pairs or matching attribute pairs is not much of a difference, at least outside the individual step. So the translating step takes the list of previous steps and executes every step on the class pair matrix as well. The steps themselves decide, if they have a different way to handle attributes or not. In the end, the translation step has one matrix with all the class pairs and a matrix for each class pair attribute matching.

Beside the normal entry, every entry also tries to insert his inverse information as long as it has an inverse. This does make sense if the inverse itself does not have a matching, since if the one direction matches the other also have to match, otherwise the matching is inconsistent and definitely wrong. To decide if the normal matching or the matching of the inverse is the better choice the similarity value acts as a confidence. The attribute with the higher confidence will be used to generate the normal and the inverse entry of an attribute, therefore it can be guaranteed that the entries are consistent and the chance of getting the right one is higher.

The translation itself is actually a very easy part, as soon as one knows which classes and attributes are related to each other, it is not much more than renaming all classes and properties in the md, nmj, nm, home table, abstracSubCl and key tables and copy the inverse, allcl and subcl table from the ODB MD to the FDB MD. This is possible due to the nature of the MD, since it already was a translation from an ontology to a database. Therefore, it always had the abstract ontology part with class names and properties and the concrete implementation specific part of the database with table and column names. So the translation to another ontology has only been done in the abstract part, since the internal structure of the database stays the same. The tables which only contain information of the ontology like *allcl*, *subcl* and *inverse* have to be copied from the ODB, since it does not matter anymore how the FDB handles those.

Chapter 4

Design

The whole system is divided into four parts: the DatabaseConverter (DatabaseConverter (DC)) [RS13], the QueryConverter (QueryConverter (QC)) [RSa], the RelationalModelBuilder [Run] (RelationalModelBuilder (RMB)) and the SchemaMatcher (SchemaMatcher (SM)), the part which is done is this thesis. To combine all these parts, a graphical user interface (Graphical User Interface (GUI)) (see Section4.3) is implemented as an additional part of the thesis. The focus of the GUI is to improve the usability of the large size and amount of matrices of the SM , but it also gives the other three steps the opportunity to make the settings and argument transfer easier. To make it more flexible the matching step structure (see Section 4.2) was chosen. This structure takes care of the parallelization of computations, the execution order, and the transfer of arguments independent from the underlying processes. Further it integrates new steps automatically into the GUI.

For the SM the SchemaObjects are introduced as superclass of SchemaClass and SchemaAttribute, which are relevant for the similarity calculation. During the calculations, it must often be differentiated between classes and attributes but the steps are very similar for all classes of an ontology or all attributes of a class. Therefore this classes can use both use the SchemaMatchingMatrix (SMM).

In this work some adaptations to the QC and the whole new creation of the SM and the GUI were done. The QC adaptations and the SM creation follow different desig styles: The QC tries to be as transparent as possible and is a classical black box since if the query and the MappingDictionary (MD) are correct, there are no uncertainties. In contrast, the SM works with probabilities, threshold, and uncertainties, which means that there some adjustments can always be made to the actual case and the results could be different by just slight threshold shifting. Therefore the SM has to give the opportunity to make these adjustments.

4.1 SchemaObjects

The *SchemaObject* is the super class of the SchemaClass and SchemaAttribute. Further, there are the SchemaMatchingMatrices (SMM), the *SchemaSummary* and the *SchemaStepResult*. Together with several subclasses, these classes are the elements for the SM similarity calculations. They were designed in a way that makes it possible to have as little work as possible to introduce a new matching method or a new visual representation.

4.1.1 SchemaObject Classes

The *SchemaObject* is for every entity of an ontology like classes and attributes which can have a similarity score. Itself contains not much more than the name of the object and the synonyms and the matched object, because these are the shared properties of the subclasses and normally the calculation methods differentiate between the sub classes, since the calculation can not be the same for those, but the containing data structure and the step order can be.

SchemaClass is the representation of the classes of an ontology. It contains all the attributes and the inheritance of this class and methods to extract this information out of the MD [RSa].

SchemaAttribute is the representation of the attributes of a class of an ontology. It contains the data type, potential keys and connections, the domain, the table it is stored in, and methods to extract this information out from the MD.

SchemaMatchingMatrix holds probability values for *SchemaObjects*, the reference to *SchemaObjectsi* and further the matching and methods to get these values and get the matched $y \in C$ for a specific $x \in R$.

SchemaStepResult is a compact version of the SMM, which stores the intermediate results where no further operations have to be done except presenting it to the user.

4.2 Matching Step

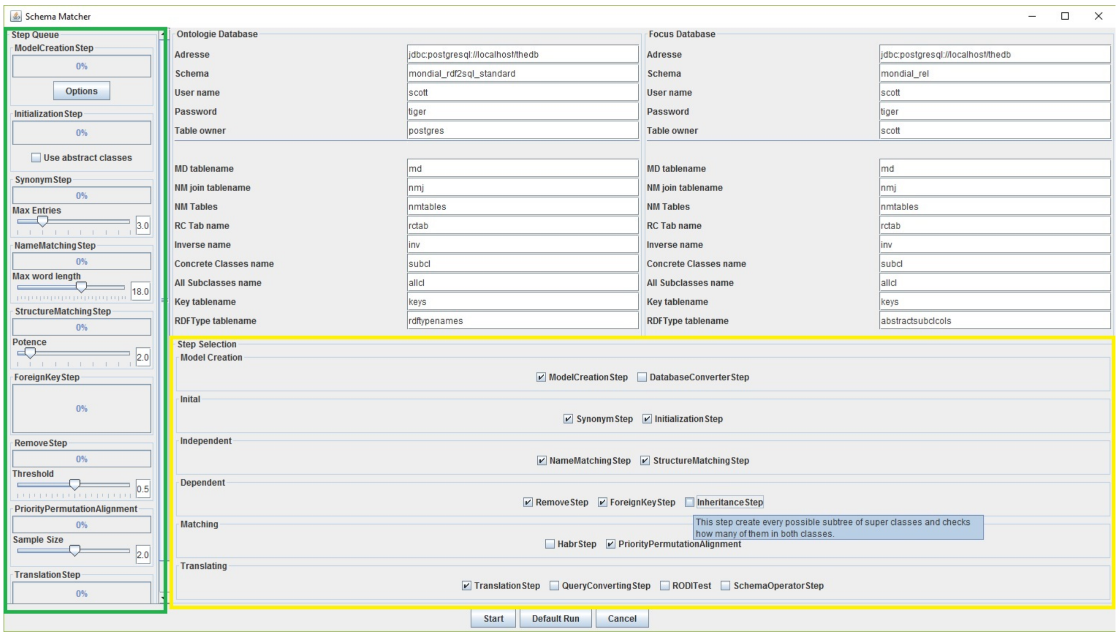
The matching step is designed to make use of parallelization on the one hand and to integrate the steps automatically into the GUI and the Step Queue (see Section 4.3.1) on the other. A matching step has an initialization, a matching, and an end phase. Actually, a step does not have to do every phase and can skip any. The difference between these phases is that the initialization and the end do not need any arguments and work as usual methods but the matching step needs one SchemaObject (see Section.4.1) from each ontology and will be called for each possible combination of $x \in R$ and $y \in C$. Further, it will divide the task between several threads, depending on the number of CPU cores. This increases the performance especially for steps like NameMatchingStep and PriorityPermutationMatchingStep. In addition to the matching and calculation methods functionality, the matching steps provide a progress bar. This is neccessary since some steps, especially with high settings, can be very time consuming. The bar gives an opportunity to see the progress and make sure the program did not crash. Further, every step can design its own options panel or use the slider prefab to transfer arguments and settings.

4.3 User Interface

The user interface has four major parts, the Step Queue, the Database Settings, the Step Selection, and the Step Results. The Database Settings and the Step Queue are only available before starting the matcher and the Step Result are iteratively generated for each step while the following steps were executed.

4.3.1 Step Queue and Step selection

The step selection provides the opportunity to select only the wanted steps or a default execution order. It is divided into several segments corresponding to the phases of the steps. The steps are always executed in the same order as the phases are, but if two were from the same phase, the selection order also decides upon the execution order.



Step Queue

ModelCreationStep: 0%

InitializationStep: 0%
☐ Use abstract classes

SynonymStep: 0%
Max Entries: 3.0

NameMatchingStep: 0%
Max word length: 18.0

StructureMatchingStep: 0%
Potence: 2.0

ForeignKeyStep: 0%

RemoveStep: 0%
Threshold: 0.5

PriorityPermutationAlignment: 0%
Sample Size: 2.0

TranslationStep: 0%

	Ontologie Database	Focus Database
Adresse	jdbc:postgresql://localhost/thedb	jdbc:postgresql://localhost/thedb
Schema	mondial_rdf2sql_standard	mondial_rel
User name	scott	scott
Password	tiger	tiger
Table owner	postgres	scott
MD tablename	md	md
NM join tablename	nmj	nmj
NM Tables	nmtables	nmtables
RC Tab name	rctab	rctab
Inverse name	inv	inv
Concrete Classes name	subcl	subcl
All Subclasses name	allcl	allcl
Key tablename	keys	keys
RDFType tablename	rdftypenames	abstractsubclcols

Step Selection

Model Creation: ✓ ModelCreationStep ☐ DatabaseConverterStep

Inital: ✓ SynonymStep ✓ InitializationStep

Independent: ✓ NameMatchingStep ✓ StructureMatchingStep

Dependent: ✓ RemoveStep ✓ ForeignKeyStep ☐ InheritanceStep

This step create every possible subtree of super classes and checks how many of them in both classes.

Matching: ☐ HabrStep ✓ PriorityPermutationAlignment

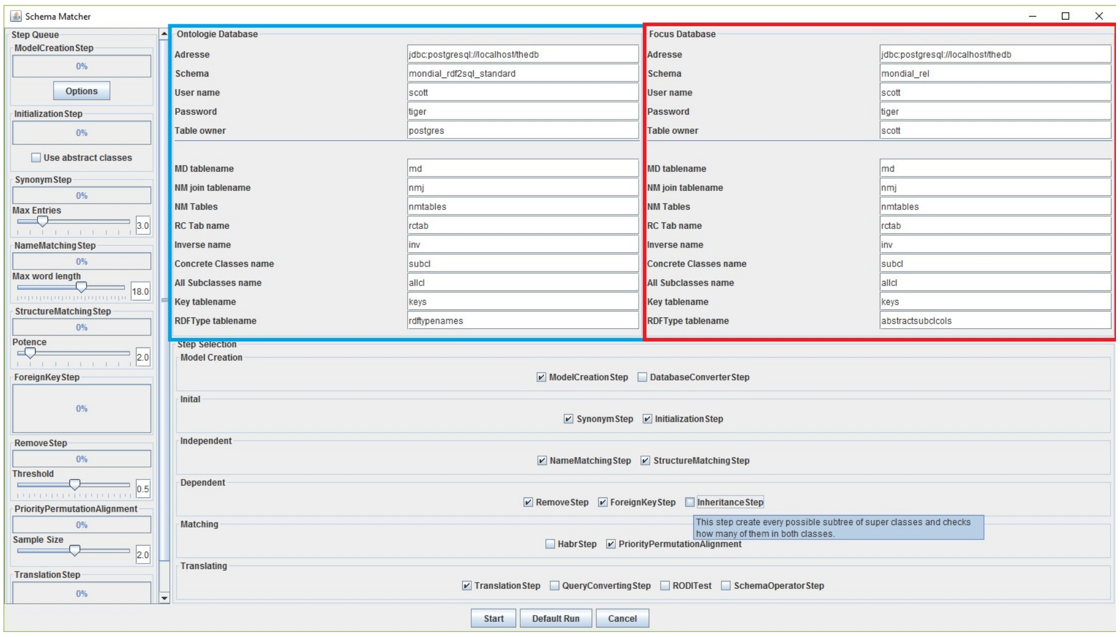
Translating: ✓ TranslationStep ☐ QueryConvertingStep ☐ RODITest ☐ SchemaOperatorStep

Figure 4.1: The SchemaMatcher UI. Step Queue marked in green, Step Selection marked in yellow

If a step is selected, its progress and options panel is displayed at the position it would be executed in the Step Queue. This panel allows to change some settings directly or in a separate window if there are several options.

4.3.2 Database Settings

The database settings provide text fields for all necessary information to establish a connection to a database and to use or create the MD on it. There are two databases settings panels for the Ontology Data Base (ODB) and the Focus Data Base (FDB). The Ontology database provides the ontology and the metadata, while the focus database provides the data sets. Besides the connection properties also the names of the tables of the ontological metadata are available and can be altered if needed, this does only make sense if the default names were already used or the ontological metadata were created in a different way than using the DC or RMB .



Ontologie Database		Focus Database	
Adresse	jdbc:postgresql://localhost/thedb	Adresse	jdbc:postgresql://localhost/thedb
Schema	mondial_rdf2sql_standard	Schema	mondial_rel
User name	scott	User name	scott
Password	tiger	Password	tiger
Table owner	postgres	Table owner	scott
MD tablename	md	MD tablename	md
NM join tablename	nmj	NM join tablename	nmj
NM Tables	nmtables	NM Tables	nmtables
RC Tab name	rctab	RC Tab name	rctab
Inverse name	inv	Inverse name	inv
Concrete Classes name	subcl	Concrete Classes name	subcl
All Subclasses name	allcl	All Subclasses name	allcl
Key tablename	keys	Key tablename	keys
RDFType tablename	rdftypenames	RDFType tablename	abstractsubclcols

Step Selection

Model Creation: ✓ ModelCreationStep ☐ DatabaseConverterStep

Inital: ✓ SynonymStep ✓ InitializationStep

Independent: ✓ NameMatchingStep ✓ StructureMatchingStep

Dependent: ✓ RemoveStep ✓ ForeignKeyStep ☐ InheritanceStep

This step create every possible subtree of super classes and checks how many of them in both classes.

Matching: ☐ HabrStep ✓ PriorityPermutationAlignment

Translating: ✓ TranslationStep ☐ QueryConvertingStep ☐ RODITest ☐ SchemaOperatorStep

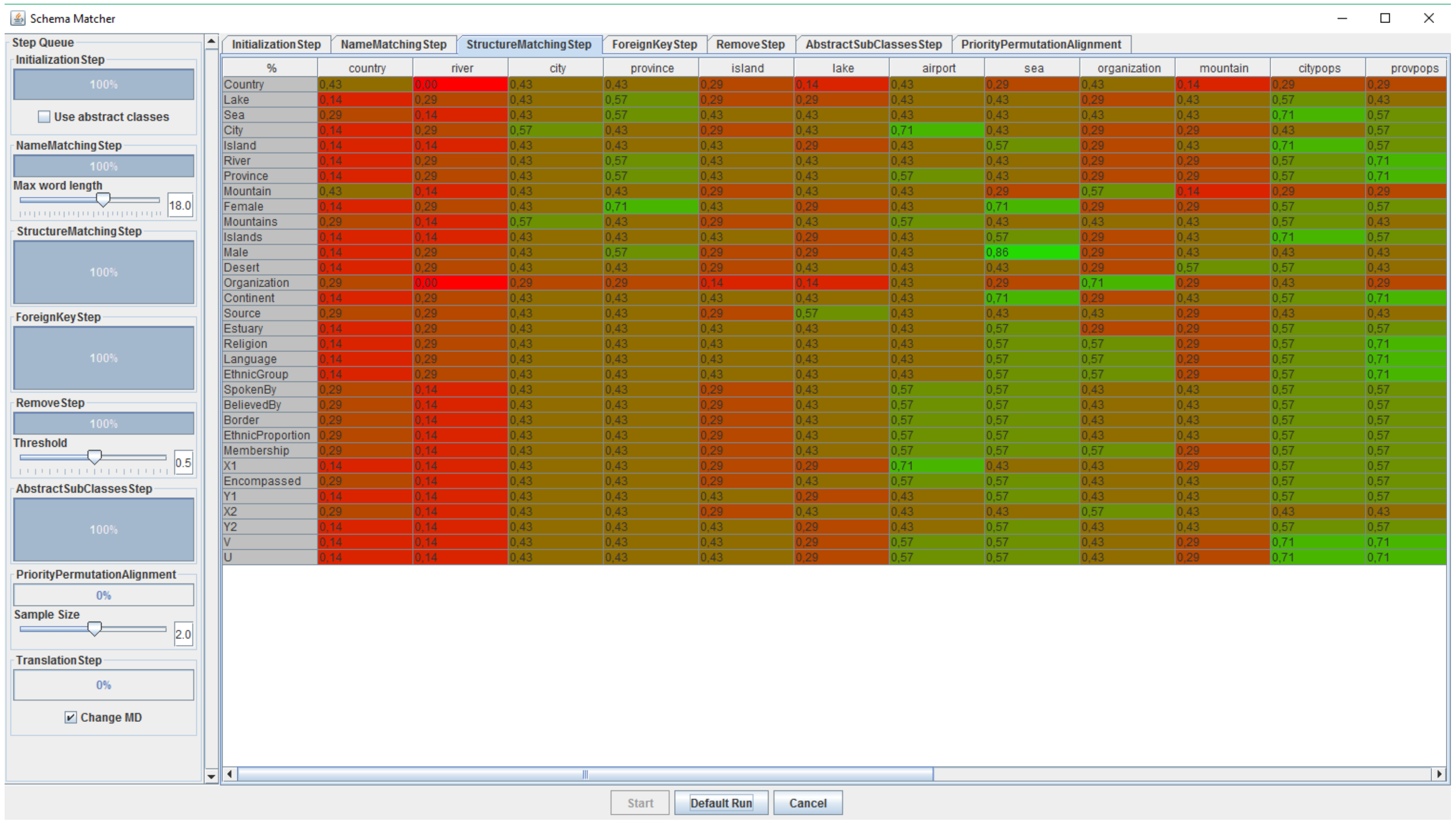
Figure 4.2: SchemaMatcher User Interface. Ontology Database settings panel marked in blue, focus database panel marked in red.

4.3.3 Step Results

When a step is done, even if other steps still need to be computed, the SM shows a step result. This is by default a matrix table with all entries colored from red to green (see Figure 4.3), depending on their values. The Step Results in the SchemaMatcher window are always an overview of all classes. If further details about the similarity score of $x \in R$ and $y \in C$ are necessary, a click on this value opens the detailed view. This is a StepResult as well but this time for the attribute of $x \in R$ and the attribute of $y \in C$. (see Figure 4.4).

However, for some steps a default StepResult would not be informative or shows the information in an unnecessary complicated way, if displayed as default StepResult. Those steps can use a subclass of SchemaResult and can create their own panel to provide the user with information he would rather be interested in. For example, for the initialization step the matrix table would only contains -1. This is not only uninteresting, it also conceals the relevant information, how the classes and attributes were initialized from the user. Therefore this step has a special StepResult to provide the relevant data in a pleasing way (see Figure 4.5). Also the translation step does not use the usual StepResult, but a special designed panel. Even if its possible to display the result in a matrix, a list with the matching provides better overview and easier access to the details of the result (see Figur 4.6).

Steps which do not belong to the SM, but representing the other parts of the RDF2SQL system, the QueryConverter, the DatabaseConverter and RelationalModelBuilder do also have own StepResults. The default StepResult would not make sense for those, since these programs do not compute on matrices. Therefore each of them has an individual StepResult (see Figures 4.7 & 4.8).


Schema Matcher

Step Queue

InitializationStep: 100%
☐ Use abstract classes

NameMatchingStep: 100%
Max word length: 18.0

StructureMatchingStep: 100%

ForeignKeyStep: 100%

RemoveStep: 100%
Threshold: 0.5

AbstractSubClassesStep: 100%

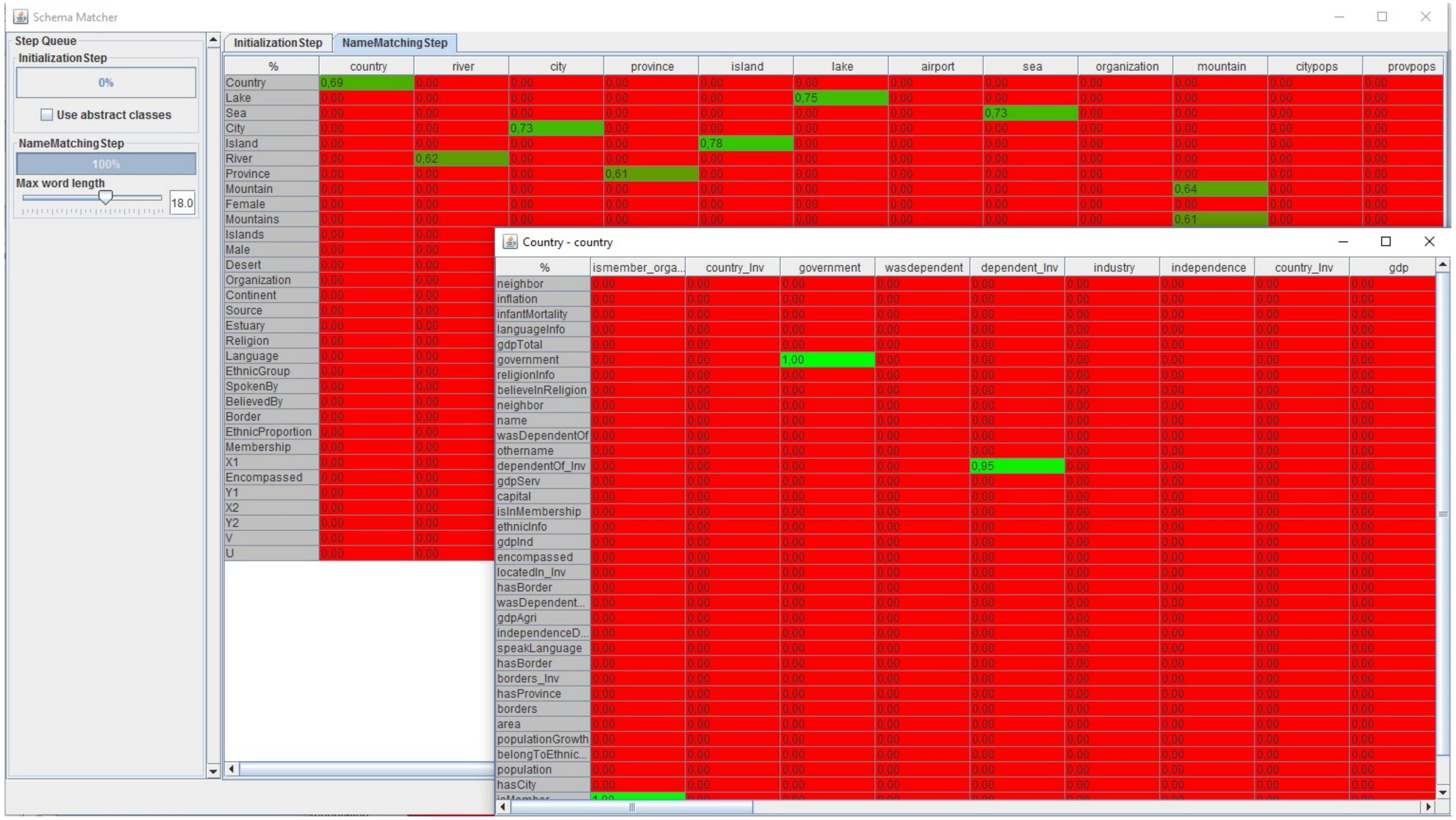
PriorityPermutationAlignment: 0%
Sample Size: 2.0

TranslationStep: 0%
☑ Change MD

InitializationStep | NameMatchingStep | StructureMatchingStep | ForeignKeyStep | RemoveStep | AbstractSubClassesStep | PriorityPermutationAlignment

%	country	river	city	province	island	lake	airport	sea	organization	mountain	citypops	provpops
Country	0,43	0,00	0,43	0,43	0,29	0,14	0,43	0,29	0,43	0,14	0,29	0,29
Lake	0,14	0,29	0,43	0,57	0,29	0,29	0,43	0,43	0,29	0,43	0,57	0,43
Sea	0,29	0,14	0,43	0,57	0,43	0,43	0,43	0,43	0,43	0,43	0,71	0,57
City	0,14	0,29	0,57	0,43	0,29	0,43	0,71	0,43	0,29	0,29	0,43	0,57
Island	0,14	0,14	0,43	0,43	0,43	0,29	0,43	0,57	0,29	0,43	0,71	0,57
River	0,14	0,29	0,43	0,57	0,43	0,43	0,43	0,43	0,29	0,29	0,57	0,71
Province	0,14	0,29	0,43	0,57	0,43	0,43	0,57	0,43	0,29	0,29	0,57	0,71
Mountain	0,43	0,14	0,43	0,43	0,29	0,43	0,43	0,29	0,57	0,14	0,29	0,29
Female	0,14	0,29	0,43	0,71	0,43	0,29	0,43	0,71	0,29	0,29	0,57	0,57
Mountains	0,29	0,14	0,57	0,43	0,29	0,43	0,57	0,43	0,43	0,43	0,57	0,43
Islands	0,14	0,14	0,43	0,43	0,43	0,29	0,43	0,57	0,29	0,43	0,71	0,57
Male	0,14	0,29	0,43	0,57	0,29	0,29	0,43	0,86	0,29	0,43	0,43	0,43
Desert	0,14	0,29	0,43	0,43	0,29	0,43	0,43	0,43	0,29	0,57	0,57	0,43
Organization	0,29	0,00	0,29	0,29	0,14	0,14	0,43	0,29	0,71	0,29	0,43	0,29
Continent	0,14	0,29	0,43	0,43	0,43	0,43	0,43	0,71	0,29	0,43	0,57	0,71
Source	0,29	0,29	0,43	0,43	0,29	0,57	0,43	0,43	0,43	0,29	0,43	0,43
Estuary	0,14	0,29	0,43	0,43	0,43	0,43	0,43	0,57	0,29	0,29	0,57	0,57
Religion	0,14	0,29	0,43	0,43	0,43	0,43	0,43	0,57	0,57	0,29	0,57	0,71
Language	0,14	0,29	0,43	0,43	0,43	0,43	0,43	0,57	0,57	0,29	0,57	0,71
EthnicGroup	0,14	0,29	0,43	0,43	0,43	0,43	0,43	0,57	0,57	0,29	0,57	0,71
SpokenBy	0,29	0,14	0,43	0,43	0,29	0,43	0,57	0,57	0,43	0,43	0,57	0,57
BelievedBy	0,29	0,14	0,43	0,43	0,29	0,43	0,57	0,57	0,43	0,43	0,57	0,57
Border	0,29	0,14	0,43	0,43	0,29	0,43	0,57	0,57	0,43	0,43	0,57	0,57
EthnicProportion	0,29	0,14	0,43	0,43	0,29	0,43	0,57	0,57	0,43	0,43	0,57	0,57
Membership	0,29	0,14	0,43	0,43	0,29	0,43	0,57	0,57	0,57	0,29	0,57	0,57
X1	0,14	0,14	0,43	0,43	0,29	0,29	0,71	0,43	0,43	0,29	0,57	0,57
Encompassed	0,29	0,14	0,43	0,43	0,29	0,43	0,57	0,57	0,43	0,43	0,57	0,57
Y1	0,14	0,14	0,43	0,43	0,43	0,29	0,43	0,57	0,43	0,43	0,57	0,57
X2	0,29	0,14	0,43	0,43	0,29	0,43	0,43	0,43	0,57	0,43	0,43	0,43
Y2	0,14	0,14	0,43	0,43	0,43	0,29	0,43	0,57	0,43	0,43	0,57	0,57
V	0,14	0,14	0,43	0,43	0,43	0,29	0,57	0,57	0,43	0,29	0,71	0,71
U	0,14	0,14	0,43	0,43	0,43	0,29	0,57	0,57	0,43	0,29	0,71	0,71

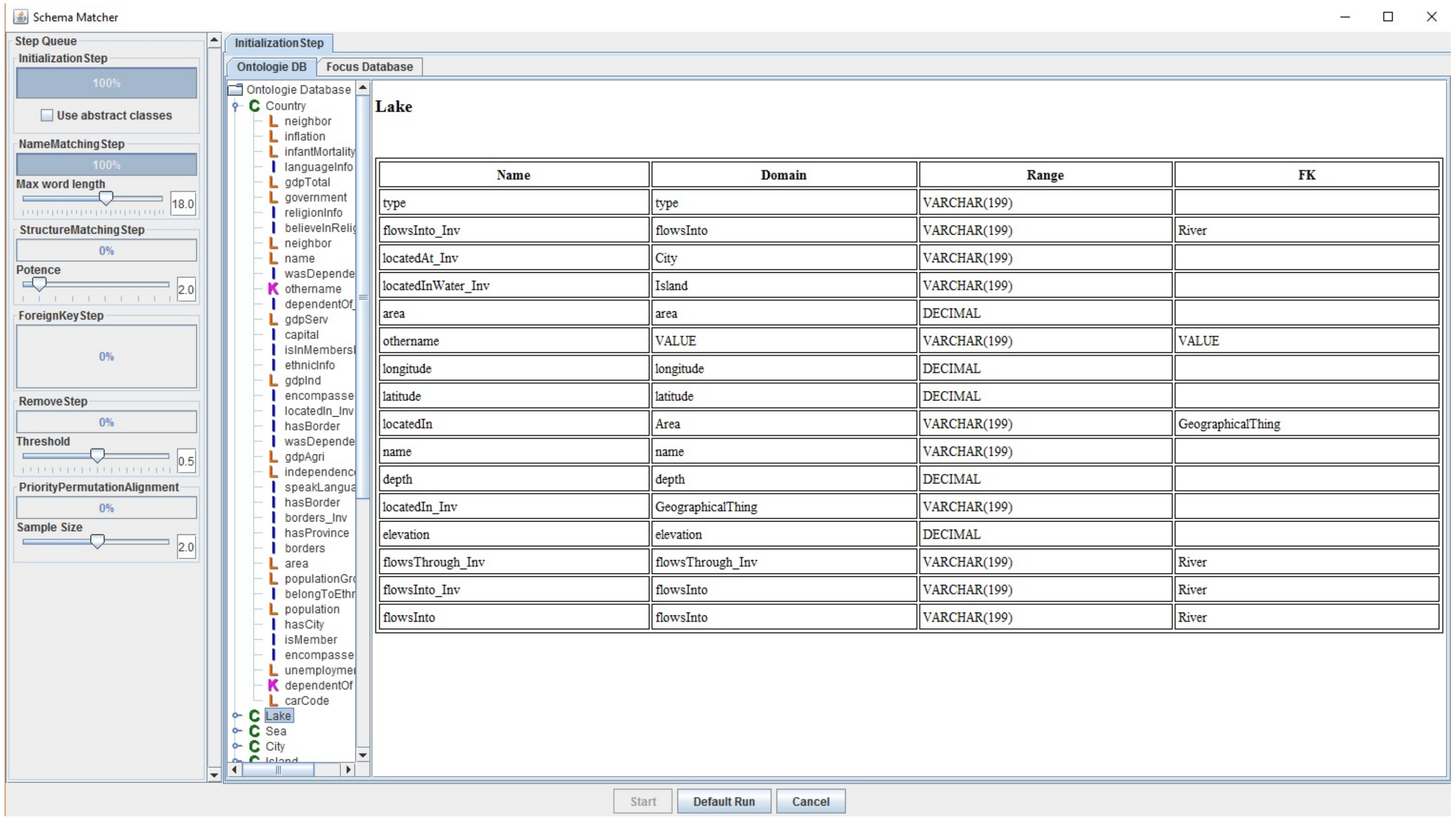
Figure 4.3: Colors fading from red to green with increasing similarity values. Every cell has the RGB - code (255 *(1 -score),255 * score,0)



%	country	river	city	province	island	lake	airport	sea	organization	mountain	citypops	provpops
Country	0,69	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Lake	0,00	0,00	0,00	0,00	0,00	0,75	0,00	0,00	0,00	0,00	0,00	0,00
Sea	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,73	0,00	0,00	0,00	0,00
City	0,00	0,00	0,73	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Island	0,00	0,00	0,00	0,00	0,78	0,00	0,00	0,00	0,00	0,00	0,00	0,00
River	0,00	0,62	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Province	0,00	0,00	0,00	0,61	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Mountain	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,64	0,00	0,00
Female	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Mountains	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,61	0,00	0,00
Islands	0,00	0,00										
Male	0,00	0,00										
Desert	0,00	0,00										
Organization	0,00	0,00										
Continent	0,00	0,00										
Source	0,00	0,00										
Estuary	0,00	0,00										
Religion	0,00	0,00										
Language	0,00	0,00										
EthnicGroup	0,00	0,00										
SpokenBy	0,00	0,00										
BelievedBy	0,00	0,00										
Border	0,00	0,00										
EthnicProportion	0,00	0,00										
Membership	0,00	0,00										
X1	0,00	0,00										
Encompassed	0,00	0,00										
Y1	0,00	0,00										
X2	0,00	0,00										
Y2	0,00	0,00										
V	0,00	0,00										
U	0,00	0,00										

%	ismember_orga...	country_Inv	government	wasdependent	dependent_Inv	industry	independence	country_Inv	gdp
neighbor	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
inflation	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
infantMortality	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
languageInfo	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
gdpTotal	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
government	0,00	0,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00
religionInfo	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
believeInReligion	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
neighbor	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
name	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
wasDependentOf	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
othername	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
dependentOf_Inv	0,00	0,00	0,00	0,00	0,95	0,00	0,00	0,00	0,00
gdpServ	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
capital	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
isInMembership	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
ethnicInfo	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
gdpInd	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
encompassed	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
locatedIn_Inv	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
hasBorder	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
wasDependent...	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
gdpAgri	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
independenceD...	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
speakLanguage	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
hasBorder	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
borders_Inv	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
hasProvince	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
borders	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
area	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
populationGrowth	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
belongToEthnic...	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
population	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
hasCity	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Figure 4.4: Default SchemaMatchingMatrix overview from NameMatchingStep and detail view



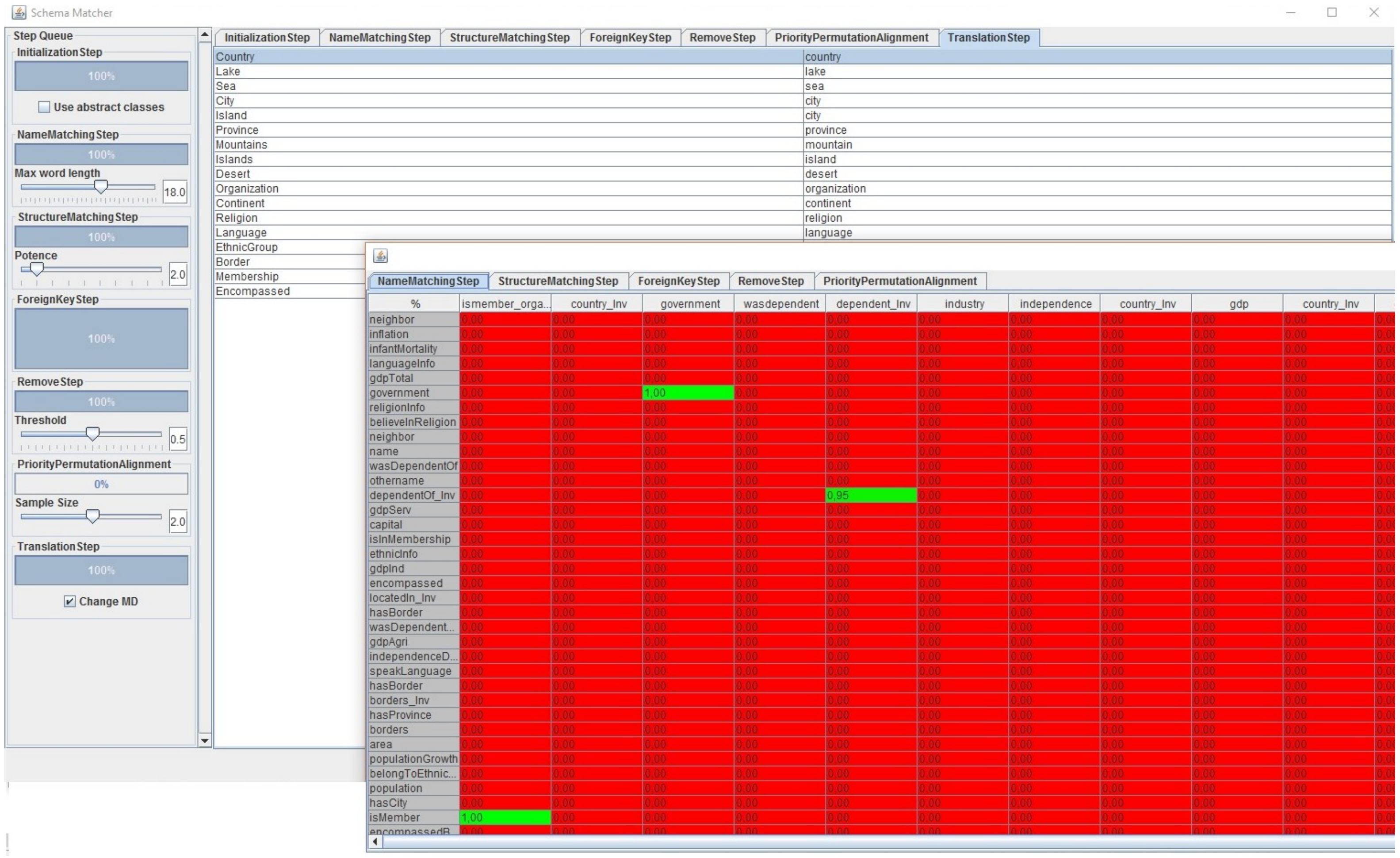
Max word length: 18.0
Potence: 2.0
Threshold: 0.5
Sample Size: 2.0

Ontologie Database
- Country
  - neighbor
  - inflation
  - infantMortality
  - languageInfo
  - gdpTotal
  - government
  - religionInfo
  - believeInRelig
  - neighbor
  - name
  - wasDepende
  - othername
  - dependentOf_
  - gdpServ
  - capital
  - isInMembersl
  - ethnicInfo
  - gdpInd
  - encompasse
  - locatedIn_Inv
  - hasBorder
  - wasDepende
  - gdpAgri
  - independenc
  - speakLangua
  - hasBorder
  - borders_Inv
  - hasProvince
  - borders
  - area
  - populationGr
  - belongToEthr
  - population
  - hasCity
  - isMember
  - encompasse
  - unemploymen
  - dependentOf
  - carCode
- Lake
- Sea
- City
- Island

Lake

Name	Domain	Range	FK
type	type	VARCHAR(199)	
flowsInto_Inv	flowsInto	VARCHAR(199)	River
locatedAt_Inv	City	VARCHAR(199)	
locatedInWater_Inv	Island	VARCHAR(199)	
area	area	DECIMAL	
othername	VALUE	VARCHAR(199)	VALUE
longitude	longitude	DECIMAL	
latitude	latitude	DECIMAL	
locatedIn	Area	VARCHAR(199)	GeographicalThing
name	name	VARCHAR(199)	
depth	depth	DECIMAL	
locatedIn_Inv	GeographicalThing	VARCHAR(199)	
elevation	elevation	DECIMAL	
flowsThrough_Inv	flowsThrough_Inv	VARCHAR(199)	River
flowsInto_Inv	flowsInto	VARCHAR(199)	River
flowsInto	flowsInto	VARCHAR(199)	River

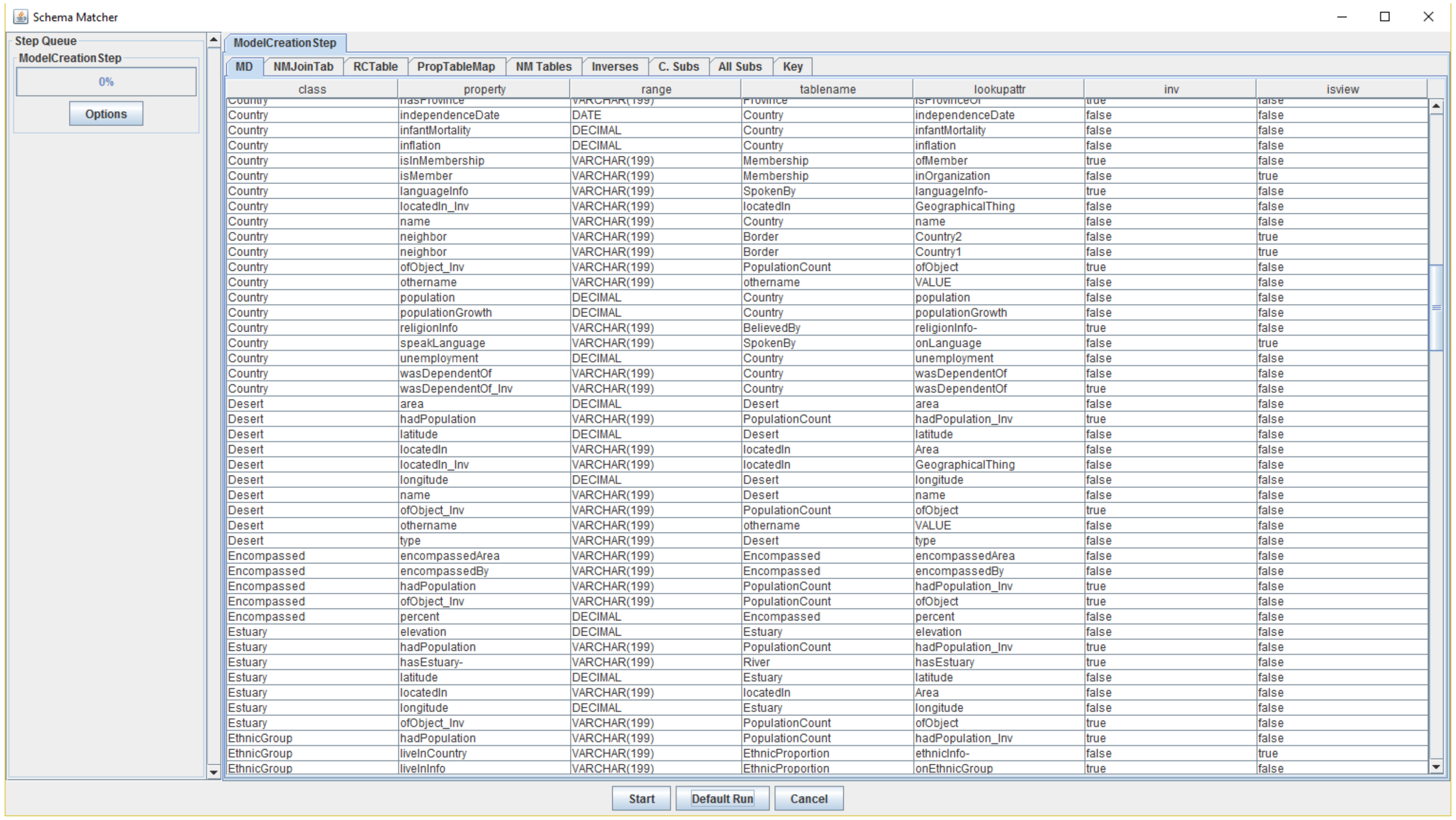
Figure 4.5: Special step result for Initialization Step. Showing all classes in a tree structure and MD entries if clicked.



Country	country
Lake	lake
Sea	sea
City	city
Island	city
Province	province
Mountains	mountain
Islands	island
Desert	desert
Organization	organization
Continent	continent
Religion	religion
Language	language
EthnicGroup	
Border	
Membership	
Encompassed	

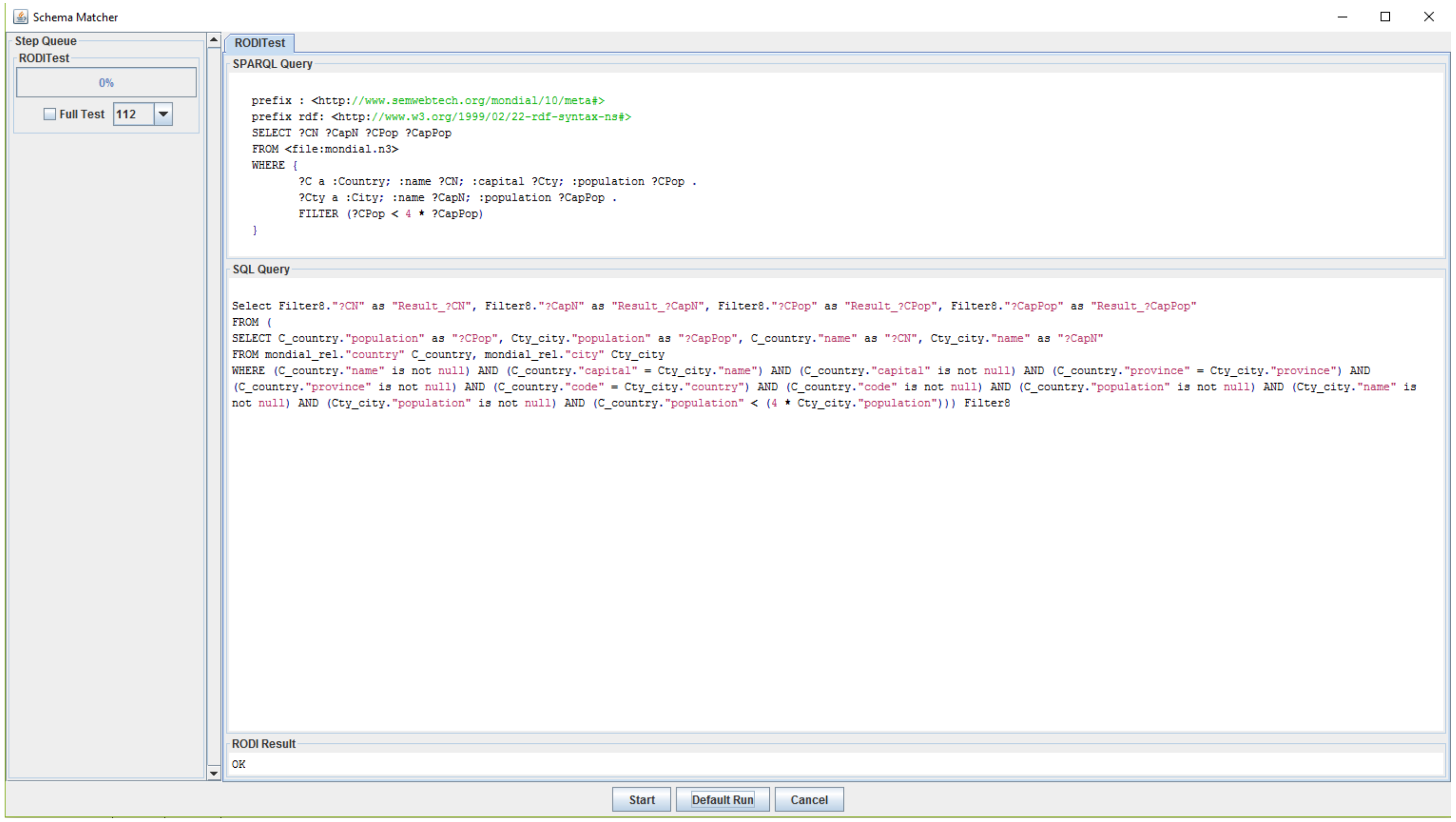
%	ismember_orga...	country_Inv	government	wasdependent	dependent_Inv	industry	independence	country_Inv	gdp	country_Inv
neighbor	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
inflation	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
infantMortality	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
languageInfo	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
gdpTotal	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
government	0,00	0,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
religionInfo	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
believeInReligion	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
neighbor	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
name	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
wasDependentOf	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
othername	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
dependentOf_Inv	0,00	0,00	0,00	0,00	0,95	0,00	0,00	0,00	0,00	0,00
gdpServ	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
capital	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
isInMembership	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
ethnicInfo	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
gdpInd	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
encompassed	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
locatedIn_Inv	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
hasBorder	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
wasDependent...	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
gdpAgri	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
independenceD...	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
speakLanguage	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
hasBorder	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
borders_Inv	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
hasProvince	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
borders	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
area	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
populationGrowth	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
belongToEthnic...	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
population	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
hasCity	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
isMember	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Figure 4.6: Special step result for Translation Step. Showing the class matches and the attribute matches and how they were calculated, if clicked on the class name.



class	property	range	tablename	lookupattr	inv	isview
Country	hasProvince	VARCHAR(199)	Province	isProvinceOf	true	false
Country	independenceDate	DATE	Country	independenceDate	false	false
Country	infantMortality	DECIMAL	Country	infantMortality	false	false
Country	inflation	DECIMAL	Country	inflation	false	false
Country	isInMembership	VARCHAR(199)	Membership	ofMember	true	false
Country	isMember	VARCHAR(199)	Membership	inOrganization	false	true
Country	languageInfo	VARCHAR(199)	SpokenBy	languageInfo-	true	false
Country	locatedIn_Inv	VARCHAR(199)	locatedIn	GeographicalThing	false	false
Country	name	VARCHAR(199)	Country	name	false	false
Country	neighbor	VARCHAR(199)	Border	Country2	false	true
Country	neighbor	VARCHAR(199)	Border	Country1	false	true
Country	ofObject_Inv	VARCHAR(199)	PopulationCount	ofObject	true	false
Country	othername	VARCHAR(199)	othername	VALUE	false	false
Country	population	DECIMAL	Country	population	false	false
Country	populationGrowth	DECIMAL	Country	populationGrowth	false	false
Country	religionInfo	VARCHAR(199)	BelievedBy	religionInfo-	true	false
Country	speakLanguage	VARCHAR(199)	SpokenBy	onLanguage	false	true
Country	unemployment	DECIMAL	Country	unemployment	false	false
Country	wasDependentOf	VARCHAR(199)	Country	wasDependentOf	false	false
Country	wasDependentOf_Inv	VARCHAR(199)	Country	wasDependentOf	true	false
Desert	area	DECIMAL	Desert	area	false	false
Desert	hadPopulation	VARCHAR(199)	PopulationCount	hadPopulation_Inv	true	false
Desert	latitude	DECIMAL	Desert	latitude	false	false
Desert	locatedIn	VARCHAR(199)	locatedIn	Area	false	false
Desert	locatedIn_Inv	VARCHAR(199)	locatedIn	GeographicalThing	false	false
Desert	longitude	DECIMAL	Desert	longitude	false	false
Desert	name	VARCHAR(199)	Desert	name	false	false
Desert	ofObject_Inv	VARCHAR(199)	PopulationCount	ofObject	true	false
Desert	othername	VARCHAR(199)	othername	VALUE	false	false
Desert	type	VARCHAR(199)	Desert	type	false	false
Encompassed	encompassedArea	VARCHAR(199)	Encompassed	encompassedArea	false	false
Encompassed	encompassedBy	VARCHAR(199)	Encompassed	encompassedBy	false	false
Encompassed	hadPopulation	VARCHAR(199)	PopulationCount	hadPopulation_Inv	true	false
Encompassed	ofObject_Inv	VARCHAR(199)	PopulationCount	ofObject	true	false
Encompassed	percent	DECIMAL	Encompassed	percent	false	false
Estuary	elevation	DECIMAL	Estuary	elevation	false	false
Estuary	hadPopulation	VARCHAR(199)	PopulationCount	hadPopulation_Inv	true	false
Estuary	hasEstuary-	VARCHAR(199)	River	hasEstuary	true	false
Estuary	latitude	DECIMAL	Estuary	latitude	false	false
Estuary	locatedIn	VARCHAR(199)	locatedIn	Area	false	false
Estuary	longitude	DECIMAL	Estuary	longitude	false	false
Estuary	ofObject_Inv	VARCHAR(199)	PopulationCount	ofObject	true	false
EthnicGroup	hadPopulation	VARCHAR(199)	PopulationCount	hadPopulation_Inv	true	false
EthnicGroup	liveInCountry	VARCHAR(199)	EthnicProportion	ethnicInfo-	false	true
EthnicGroup	liveInInfo	VARCHAR(199)	EthnicProportion	onEthnicGroup	true	false

Figure 4.7: The ModelCreationStep - StepResult shows all MD tables and entries.



```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?CN ?CapN ?CPop ?CapPop
FROM <file:mondial.n3>
WHERE {
        ?C a :Country; :name ?CN; :capital ?Cty; :population ?CPop .
        ?Cty a :City; :name ?CapN; :population ?CapPop .
        FILTER (?CPop < 4 * ?CapPop)
}
```

```
Select Filter8."?CN" as "Result_?CN", Filter8."?CapN" as "Result_?CapN", Filter8."?CPop" as "Result_?CPop", Filter8."?CapPop" as "Result_?CapPop"
FROM (
SELECT C_country."population" as "?CPop", Cty_city."population" as "?CapPop", C_country."name" as "?CN", Cty_city."name" as "?CapN"
FROM mondial_rel."country" C_country, mondial_rel."city" Cty_city
WHERE (C_country."name" is not null) AND (C_country."capital" = Cty_city."name") AND (C_country."capital" is not null) AND (C_country."province" = Cty_city."province") AND
(C_country."province" is not null) AND (C_country."code" = Cty_city."country") AND (C_country."code" is not null) AND (C_country."population" is not null) AND (Cty_city."name" is
not null) AND (Cty_city."population" is not null) AND (C_country."population" < (4 * Cty_city."population"))) Filter8
```

```
OK
```

Figure 4.8: The RODI Test Step shows the SPARQL query, the resulting SQL query and whether the result of the SQL query is equivalent to the control SQL query

Chapter 5

Implementation

The implementation was done in Java 7 with the usage of this external packages: OJDBC7 [Ora], Jena [Apaa], Pellet [SPG$^+$07], postgressql [Grob], syntaxOsterMiller [Ost], saxon [Lim].

5.1 QC - QueryConverter

The QueryConverter itself was already implemented in a previous work. For details, see [RSa]. In this thesis only the adapted and new parts are mentioned (for UML diagrams of the important parts see Appendix A).

5.1.1 Class name table

The Class names table is an additional part of the MappingDictionary (MD) (see Section 2.3.3.1). The names are stored in a hash map to keep the access time as short as possible. The hash map is filled during the normal update of the mapping dictionary, accesses the class name table in the database and takes the class name as key and the table name as value. The class names can be accessed by the static (if only one MD is used) and non-static (dealing with multiple MDs) methods getClassTable(classname). These methods replace every case in the QueryConverter (QC) code where the class name was used, when the table name is meant. Without the translations in mind this was absolutely legitimate, since the class name defines the table name of the home table, but after a matching onto another ontology the class names can of course differ.

5.1.2 multi-column keys

The multi-column keys are more complicated to implement than the class names. It must be differentiated between the two different information which are stored in one table.

- *getUriSelection(ClassVariable cv, String className, Node n)* returns a SelectVarInfo which contains the Unique Resource Identifier (URI) or the constructed URI in a single variable.

Even though a ClassVariable (CV) can have multiple classes, this function neither can handle this nor is it needed. Since multiple classes always cause a union and the selection is done in the leaves which are UnionObjectParts and these only deal with a single class each and become fused in the higher tiers to a multi-class. This is also the reason, why a className is needed to make sure it deals with the right class in each UnionObjectPart. The function itself calls a URI Generator instance, which takes care of the right selections (see for further details Section 5.1.4)

- *getNMUriSelection(ClassVariable cv, String className,Table nmTable, Node n, boolean sym = false)* this function is used to get the key connection from a home table to an nm table. Therefore the CV, the node and the class name are needed for the same reasons as for the normal URI selection, but it also needs the table it should be linked to, since the column would not be enough for multi-column keys. For the case of a symmetric reified table, there is an additional parameter which handles the problem of assigning multiple MD entries to a single property. If this is true, it takes care of the the pref entry value which will be assigned to 1 for the one and 0 for the other and select the first or the second entry (further details see [RSa]).

5.1.3 Abstract Subclasses columns

Instead of just taking a look at the column "*RDFType*" in the home table, now every time the method *AbstractSubclColEntry(String classname)* is called from the MD, when an additional condition for an abstract subclass is generated. This function just needs the name of the abstract subclass and gives back an *AbstractSubclColEntry* Object (see Figure A.3). This *AbstractSubclColEntry* object can be straightforwardly generated from the database *AbstractSubColTable* and is stored in a list when the MD is loaded.

5.1.4 URI generation

The URI generation is done by the *UriGenerator*. This class contains user made or automatically created *UriGeneratorRule*-Objects, a prefix, and a separator which decide how a URI will be generated. The *UriGenerator* sorts and executes every rule of a certain class to create the URI.

A single *UriGenerationRule* will handle one key property and has a classname, a columnname, a propertyname, and priority. The priority and if its equal the propertyName decide in which order this rules will be executed. The *UriGeneratorRule* creates a $"PropertyName" + seperator + HomeTable(Class).columnname$ and the *UriGenerator* creates $prefix + separator + UriGeneratorRule_1 + separator + UriGeneratorRule_2 + separator...UriGeneratorRule_n + separator$ string, which uses a special case of SelectedVarInfo [RSa]. This special type of SelectedVarInfo places just a string instead of $TableName.ColumnName$ in the selection, otherwise the URI would not be presented in a valid way.

5.2 SM - SchemaMatcher

The SchemaMatcher (SM) is conceived around the SchemaMatchingMatrices and the Matching-Steps. The matrices are created or altered in every MatchingStep, up to the point when a matrix contains the result.

5.2.1 SchemaMatcher - Main Class

The main class of this program the SchemaMatcher. It starts the graphical user interface Graphical User Interface (GUI), but its major part is after selection of the steps to be executed. This class takes care of the MD creation from the database. Therefore the DatabaseSettings object from the UI is used to set the connection to the database and then call the update Method of the MD. After the MD creation its execute, every selected step in the StepQueue on the main SchemaMatchingMatrix (SMM), which takes care of eventual further step calls on sub-matrix. Further between the steps, the SchemaMatcher is also responsible for the combination of the matrices.

5.2.2 MatchingSteps (MS)

There are 19 different MatchingStep (MS) subclasses (see FigureB.4) with different purpose and areas of application. But they all share the basic concept of all MS, that the calculation is divided into three phases and that the MS creates its own Option/Progress panel in the UI (see FigureB.3).

The three phases of the MS are the init(), match(SchemaObject x,SchemaObject y) and end() phase. Those phases are abstract and have to be implemented in every concrete subclass. Howerver, with the hasInit(), hasEnd() and hasMatching() methods every subclass can decide, if it actually wants to use them or leave them empty. While init() and end() are just called without any arguments and just execute the routines of the respective step, the match function is called for every row-column-combination of the MS and has a calculation for SchemaClass and SchemaAttributes. As long as the matrix does not use its own values, like it would do in a dynamic programming application, parallelization of this matrix wide calculations is rather unproblematic. After the init() - method theMS automatically creates a number of threads either depending on the cores of the CPU or a fixed set value, since it turns out that it could not be that easily be said which number of threads it the optimal one.

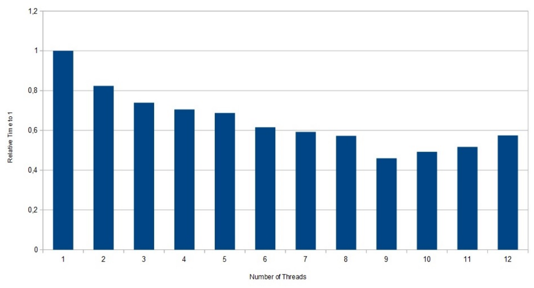


Figure 5.1: Relative computation time of the NameMatchingStep with different numbers of threads on a QuadCore CPU [Int] . Averaged over 50 runs

In a test run with different settings (see Figure5.1), it turns out that 9 threads seem to be the best choice on this machine, but it seems very hardware-dependent. Depending on the number of threads which are created, the MS divides the SMM cells into pieces with the same amount of cells and tells every threat from which point it should start and up to which point it should calculate. Therefore there are no race conditions or access complications, since every thread has its own memory. The only critical point is actually the progress bar. Since it can take a while, the progress bar is relatively important for the user to know that the program is still running, but if multiple threads want to increase the progress bar at the same time the classical race conditions problem occurs. Since that is a well-known problem, there is a java built-in semaphore which can solve this problem. To avoid that the waiting time for the writing access to the progress bar nullifies the gained speed by threading the task, the number of accesses to the progress bar - progress value are limited to an amount where actually a progress is visible (it only displays full percent values) Because of this the program only changes the value every x calculations and not every time.

The MS joins all threads and only after every thread is done execute the end() - method. Actually it would be more efficient, to split the calculations of the threads randomly or diagonal, so that there are no bad luck threads which have to make computation heavy calculations all the time, but since this approach is not focused on short runtimes but a proof of concept, it was decided to use the easier and faster to implemented way of splitting the payload address wise.

5.2.3 SchemaMatchingMatrix (SMM)

The primary function of a SchemaMatchingMatrix (SMM) is to store on the one side two sets of SchemaObjects and an array of likelihood objects and on the other side to provide access to all of these elements in a regulated way so that the right likelihood is accessed for every combination of the two SchemaObjects. Also, it gave the MatchThreads a structure to work on. The sets of SchemaObjects are stored in lists and HashMaps as well to provide fast iteration and single object access. The likelihood array has the size of each list and the list position represents the array x or y coordinate.

The SMM is also the object which executes the MS and evaluate themselves.

5.2.4 SchemaObjects Package

The SchemaObjects are the common representation of the classes and attributes of the ontologies. Besides the SchemaObjects and its subclasses, there are several other classes in this package (see FigureB.6), which are not related with the SchemaObject, but deal with it storage or visual representation.

5.2.4.1 SchemaObject and Subclasses

SchemaClass and SchemaAttribute do not have too much in common. In both cases a matrix and similarity values are needed, therefore this classes have a shared superclass (see Figure B.5) and the calculation always check if they have to do the calculation for a class or an attribute. This is the reason why SchemaObject does not have much more than the name and synonyms and some abstract methods like copy() or getMatchable(). Copy() is used to create an exact copy of this SchemaObject, so the old one stays in its state while the new one can be changed without losing the previous results in case of saving interim results.

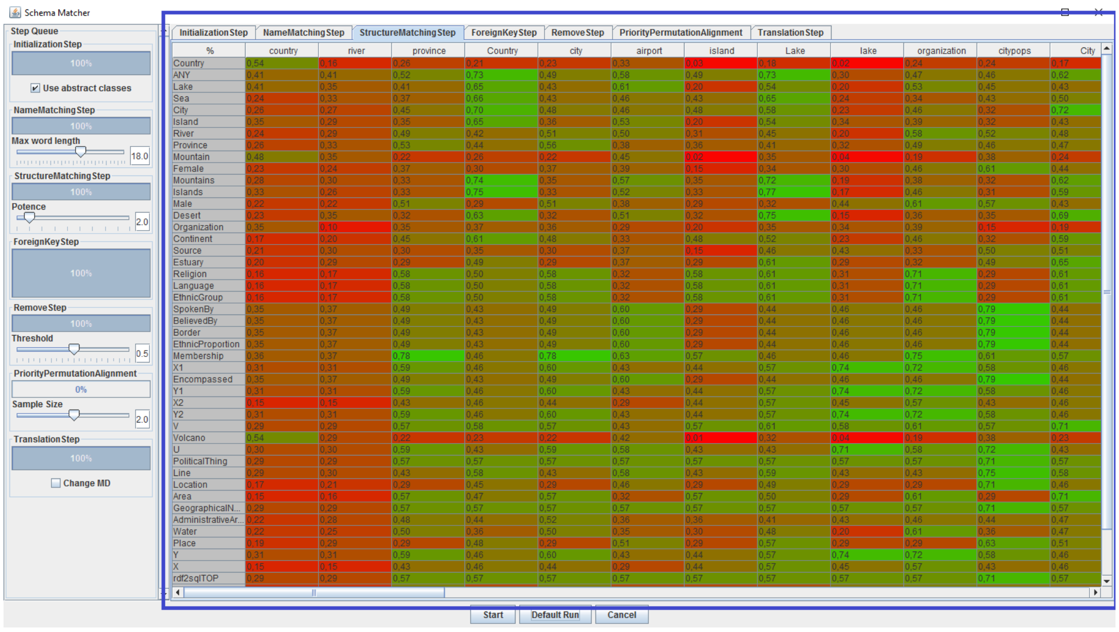
The SchemaClass has an inheritance structure (which does not make any sense for an attribute), a matched class, and the attributes list it contains. Further, it has a list with references to all matrix results where the SchemaClass was involved as a class.

The SchemaAttribute has more detailed information about itself since it can be inverse, a primary or foreign key, a nm or 1:n connection and has a table, a range, a domain, and a reference to a SchemaClass as its class. Usually the computation of the SchemaAttribute is also part of the SchemaClass, since the MS score of a class never only depends on the class it self without considering the attributes, even if there are none, it influences the class score.

5.2.4.2 SchemaTable

SchemaTable is the central tool of representing the results of a SMM either of SchemaClasses or SchemaAttributes (see Figure5.2). If the MS does not have a getDetails() method override, the results of the MS will be represented in the SchemaTable. The SchemaTable is an extension of the JTable, which means it represents an array of objects with a toString() method and an array with header values. Further, the extension converts the first column into another header and adds a MouseAdapter to the table, which opens another SchemaTable with the details of the selected cell on click. Additional it has a TableCellRenderer, which primary function is to change the background color of a cell depending on its value from zero as red gradually to one more green, other values in black except -1. Further, the TableCellRenderer also makes the cell uneditable.

5.2.4.3 SchemaHistory


Schema Matcher

Step Queue

InitializationStep 100%

☑ Use abstract classes

NameMatchingStep 100%

Max word length 18.0

StructureMatchingStep 100%

Potence 2.0

ForeignKeyStep 100%

RemoveStep 100%

Threshold 0.5

PriorityPermutationAlignment 0%

Sample Size 2.0

TranslationStep 100%

☐ Change MD

InitializationStep | NameMatchingStep | StructureMatchingStep | ForeignKeyStep | RemoveStep | PriorityPermutationAlignment | TranslationStep

%	country	river	province	Country	city	airport	island	Lake	lake	organization	citypops	City
Country	0,54	0,16	0,26	0,21	0,23	0,33	0,03	0,18	0,02	0,24	0,24	0,17
ANY	0,41	0,41	0,52	0,73	0,49	0,58	0,49	0,73	0,30	0,47	0,46	0,62
Lake	0,41	0,35	0,41	0,65	0,43	0,61	0,20	0,54	0,20	0,53	0,45	0,43
Sea	0,24	0,33	0,37	0,66	0,43	0,46	0,43	0,65	0,24	0,34	0,43	0,50
City	0,26	0,27	0,45	0,70	0,48	0,46	0,48	0,58	0,23	0,46	0,32	0,72
Island	0,35	0,29	0,35	0,65	0,36	0,53	0,20	0,54	0,34	0,39	0,32	0,43
River	0,24	0,29	0,49	0,42	0,51	0,50	0,31	0,45	0,20	0,58	0,52	0,48
Province	0,26	0,33	0,53	0,44	0,56	0,38	0,36	0,41	0,32	0,49	0,46	0,47
Mountain	0,48	0,35	0,22	0,26	0,22	0,45	0,02	0,35	0,04	0,19	0,38	0,24
Female	0,23	0,24	0,37	0,30	0,37	0,39	0,15	0,34	0,30	0,46	0,61	0,44
Mountains	0,28	0,30	0,33	0,74	0,35	0,57	0,35	0,72	0,19	0,38	0,32	0,62
Islands	0,33	0,26	0,33	0,75	0,33	0,52	0,33	0,77	0,17	0,46	0,31	0,59
Male	0,22	0,22	0,51	0,29	0,51	0,38	0,29	0,32	0,44	0,61	0,57	0,43
Desert	0,23	0,35	0,32	0,63	0,32	0,51	0,32	0,75	0,15	0,36	0,35	0,69
Organization	0,35	0,10	0,35	0,37	0,36	0,29	0,20	0,35	0,34	0,39	0,15	0,19
Continent	0,17	0,20	0,45	0,61	0,48	0,33	0,48	0,52	0,23	0,46	0,32	0,59
Source	0,21	0,30	0,30	0,35	0,30	0,37	0,15	0,46	0,43	0,33	0,50	0,51
Estuary	0,20	0,29	0,29	0,49	0,29	0,37	0,29	0,61	0,29	0,32	0,49	0,65
Religion	0,16	0,17	0,58	0,50	0,58	0,32	0,58	0,61	0,31	0,71	0,29	0,61
Language	0,16	0,17	0,58	0,50	0,58	0,32	0,58	0,61	0,31	0,71	0,29	0,61
EthnicGroup	0,16	0,17	0,58	0,50	0,58	0,32	0,58	0,61	0,31	0,71	0,29	0,61
SpokenBy	0,35	0,37	0,49	0,43	0,49	0,60	0,29	0,44	0,46	0,46	0,79	0,44
BelievedBy	0,35	0,37	0,49	0,43	0,49	0,60	0,29	0,44	0,46	0,46	0,79	0,44
Border	0,35	0,37	0,49	0,43	0,49	0,60	0,29	0,44	0,46	0,46	0,79	0,44
EthnicProportion	0,35	0,37	0,49	0,43	0,49	0,60	0,29	0,44	0,46	0,46	0,79	0,44
Membership	0,36	0,37	0,78	0,46	0,78	0,63	0,57	0,46	0,46	0,75	0,61	0,57
X1	0,31	0,31	0,59	0,46	0,60	0,43	0,44	0,57	0,74	0,72	0,58	0,46
Encompassed	0,35	0,37	0,49	0,43	0,49	0,60	0,29	0,44	0,46	0,46	0,79	0,44
Y1	0,31	0,31	0,59	0,46	0,60	0,43	0,44	0,57	0,74	0,72	0,58	0,46
X2	0,15	0,15	0,43	0,46	0,44	0,29	0,44	0,57	0,45	0,57	0,43	0,46
Y2	0,31	0,31	0,59	0,46	0,60	0,43	0,44	0,57	0,74	0,72	0,58	0,46
V	0,29	0,29	0,57	0,58	0,57	0,43	0,57	0,61	0,58	0,61	0,57	0,71
Volcano	0,54	0,29	0,22	0,23	0,22	0,42	0,01	0,32	0,04	0,19	0,38	0,23
U	0,30	0,30	0,59	0,43	0,59	0,58	0,43	0,43	0,71	0,58	0,72	0,43
PoliticalThing	0,29	0,29	0,57	0,57	0,57	0,57	0,57	0,57	0,57	0,57	0,71	0,57
Line	0,29	0,30	0,43	0,58	0,43	0,58	0,43	0,59	0,43	0,43	0,75	0,58
Location	0,17	0,21	0,29	0,45	0,29	0,46	0,29	0,49	0,29	0,29	0,71	0,46
Area	0,15	0,16	0,57	0,47	0,57	0,32	0,57	0,50	0,29	0,61	0,29	0,71
GeographicalN...	0,29	0,29	0,57	0,57	0,57	0,57	0,57	0,57	0,57	0,57	0,71	0,57
AdministrativeAr...	0,22	0,28	0,48	0,44	0,52	0,36	0,36	0,41	0,43	0,46	0,44	0,47
Water	0,22	0,25	0,50	0,36	0,50	0,35	0,30	0,48	0,20	0,61	0,36	0,47
Place	0,19	0,29	0,29	0,48	0,29	0,51	0,29	0,57	0,29	0,29	0,63	0,51
Y	0,31	0,31	0,59	0,46	0,60	0,43	0,44	0,57	0,74	0,72	0,58	0,46
X	0,15	0,15	0,43	0,46	0,44	0,29	0,44	0,57	0,45	0,57	0,43	0,46
rdf2sqlTOP	0,29	0,29	0,57	0,57	0,57	0,57	0,57	0,57	0,57	0,57	0,71	0,57

Figure 5.2: SchemaHistory (blue frame) with a SchemaTable open

The SchemaHistory is majorly a holder for a JTabbedPane and a list of SchemaStepRecords. Out of those it creates the result panel (see Figure5.2) during and after the SchemaMatcher is computing.

5.2.4.4 SchemaStepResult

The SchemaStepResult is normally a SchemaTable, but if the MS has a special getDetails() method, it also can be one of it subclasses. In the end, the SchemaHistory just expects to get a JPanel and the content does not matter at all and the SchemaStepResults can appear in very different ways. Those different appearances are subclasses of SchemaSummary. SchemaSummary itself is used by the TranslationStep and looks like in figure4.6.SchemaSummaryInitialization creates an overview over all classes, like the one in figure 4.5 and is used by the InitializationStep, the SchemaSummaryQueryResult is used by the QueryStep and SchemaSummaryRODIResult is used by the RODIStep.

5.2.5 Translator

The Translation Step needs a matching Step like the PriorityPermutations (see Section3.2.6.2) to be executed before. This means it gets a decision matrix with already matched classes.

In its initialization phase the translation step runs all other used steps, which comes after the initialization phase and before the translation phase, to create an attribute matching for each class pair. After this, the matching phase alters all classname according to the class matching and then update and inserts the matched attributes into the MD. In the end phase all entries with old class names will be removed as long as the name is not in the Ontology Data Base (ODB) otherwise it will be check attribute wise.

The Translation Step consists of the translationStep class and the translator class. While the step coordinates the execution order as well as performs the attribute and class matching the translator is responsible for the communication with the database and the altering of the MD

Chapter 6

Conclusion

The relational model of a database is the abstract view on a complex real world problem, which means that certain aspects have to be ignored, groups of objects have to be constructed and some abstract classifications have to be made to transform this problem into a database model or an ontology. This step has many degrees of freedom and can result a large number of possibilities which are more or less useful. Rules like normalization of databases could reduce this amount of possible structures to a much easier to handle number but one cannot rely on these rules, since too many databases do not follow these rules. Therefore this is a task which is too complex to be completely solved in this thesis but a solid step in the in the right direction is done. With the test databases a great amount of the matchable pairs were found and provide access for the QueryConverter (QC) to the gained information. However, the test databases were quite different in the end and the most things which could not be found are actually pretty hard ones [PBJR+15] but never the less, cases you have to deal with in reality. In comparison with the other matching tools, this one performs more than comparable.

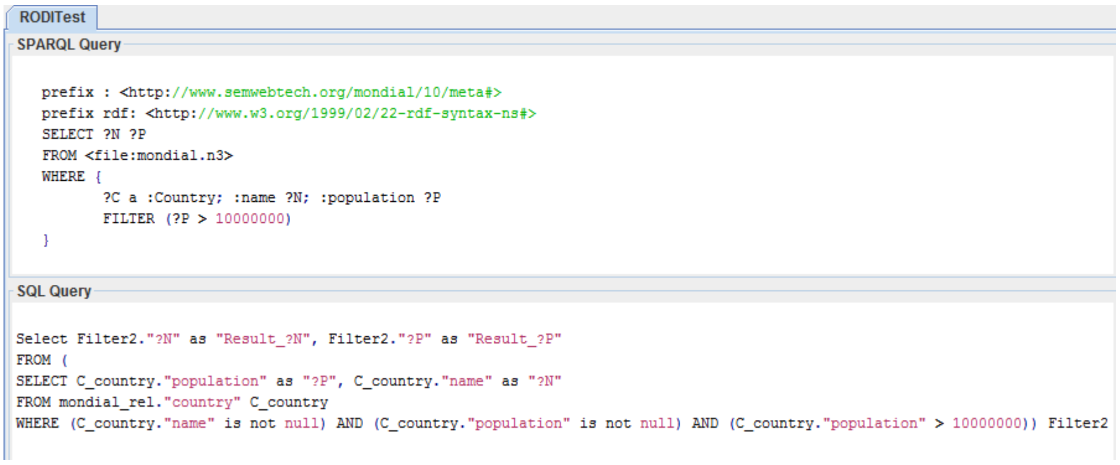
6.1 Results

All test were performed on the RODI - Mondial - Benchmark - query set, which was used as a scenario in the benchmark [PBJR+15], but the results were too disappointing so they did not make it into the published version of the publication. To use the whole benchmark could have given more significant results, but setting these tests up is a very complex task. It even took the creators of the benchmark several weeks to set up the test for their own tools. This was not possible to achieve for an outsider in the time frame of this thesis. Further the focus of this thesis is to create a tool which performs better on this specific scenario and points out the major issues of it. Therefore the decision was made to use the RODI query set to have comparable result on one hand and on the other hand a test environment with an reasonable implementation effort.

Beside the query set the ontological metadata can also be used as a negative measurement for the results. For every class-property combination existing in the ontological metadata of the Ontology Data Base (ODB) but not in the one of the Focus Data Base (FDB) one knows that this property or even the whole class was not matched. However this only works in the negative way, because of just being matched an existing matching/an entry do not have to be correct.

It shows, that the SchemaMatcher (SM) has answered 28 of 67 ($\approx 42\%$) correct and has found 169 of the 275 ($\approx 61\%$) of the MappingDictionary (MD) entries of the ODB.

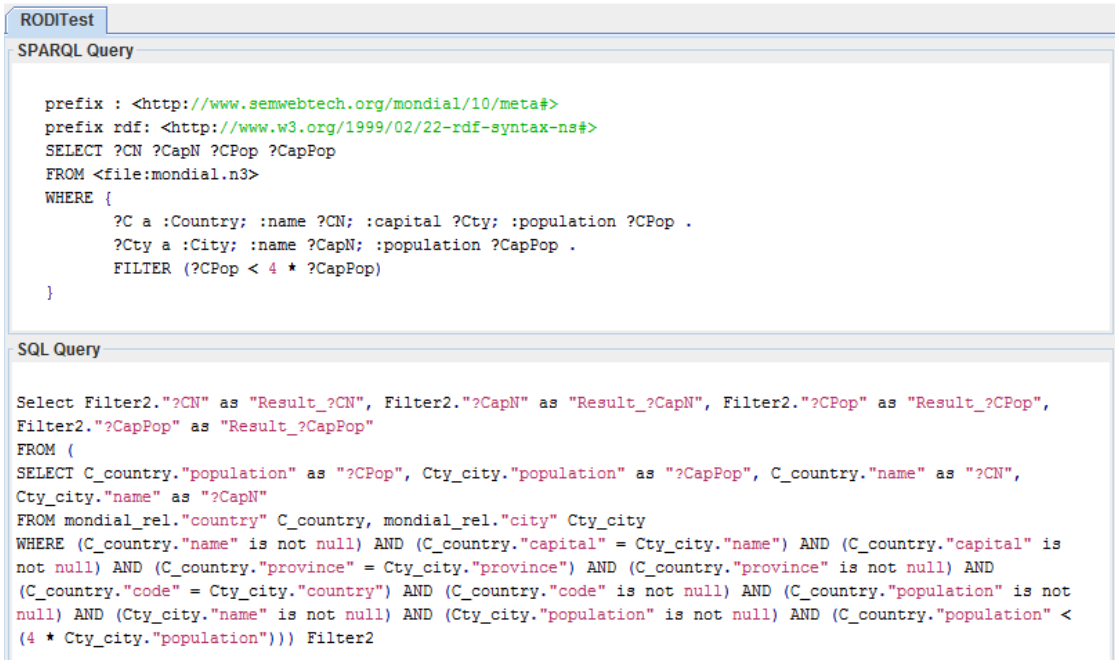
6.1.1 Working examples



```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?N ?P
FROM <file:mondial.n3>
WHERE {
        ?C a :Country; :name ?N; :population ?P
        FILTER (?P > 10000000)
}
```

```
Select Filter2."?N" as "Result_?N", Filter2."?P" as "Result_?P"
FROM (
SELECT C_country."population" as "?P", C_country."name" as "?N"
FROM mondial_rel."country" C_country
WHERE (C_country."name" is not null) AND (C_country."population" is not null) AND (C_country."population" > 10000000)) Filter2
```

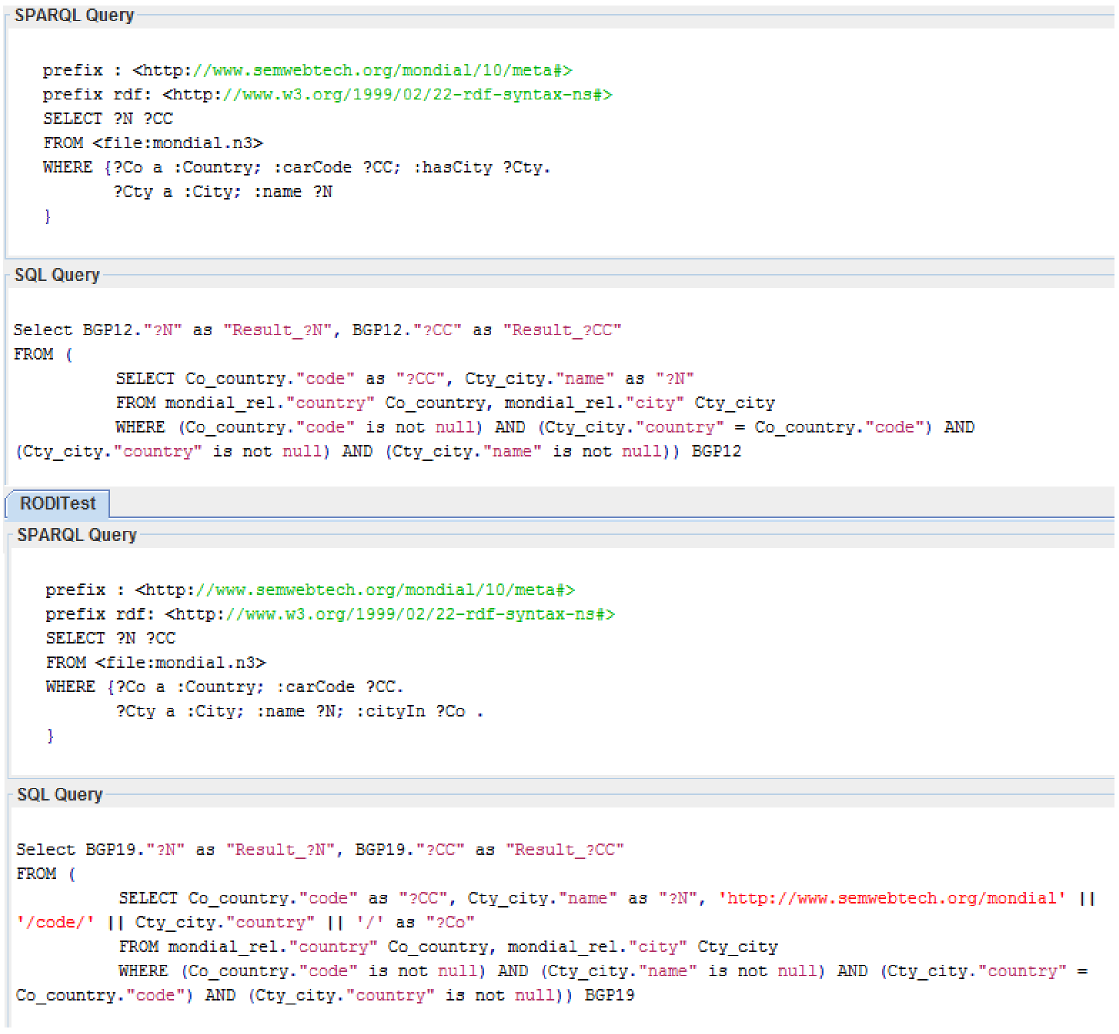
Figure 6.1: RODI 111 - Warm up query. *Country* is one of the harder classes since it differs a lot in the ontologies



```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?CN ?CapN ?CPop ?CapPop
FROM <file:mondial.n3>
WHERE {
       ?C a :Country; :name ?CN; :capital ?Cty; :population ?CPop .
       ?Cty a :City; :name ?CapN; :population ?CapPop .
       FILTER (?CPop < 4 * ?CapPop)
}
```

```
Select Filter2."?CN" as "Result_?CN", Filter2."?CapN" as "Result_?CapN", Filter2."?CPop" as "Result_?CPop",
Filter2."?CapPop" as "Result_?CapPop"
FROM (
SELECT C_country."population" as "?CPop", Cty_city."population" as "?CapPop", C_country."name" as "?CN",
Cty_city."name" as "?CapN"
FROM mondial_rel."country" C_country, mondial_rel."city" Cty_city
WHERE (C_country."name" is not null) AND (C_country."capital" = Cty_city."name") AND (C_country."capital" is
not null) AND (C_country."province" = Cty_city."province") AND (C_country."province" is not null) AND
(C_country."code" = Cty_city."country") AND (C_country."code" is not null) AND (C_country."population" is not
null) AND (Cty_city."name" is not null) AND (Cty_city."population" is not null) AND (C_country."population" <
(4 * Cty_city."population"))) Filter2
```

Figure 6.2: RODI 112 - Multi-column keys and functional connection with a easily to mixed up attribute. *hasCity* and *capital* are very similar especially since the names differ a lot in the FDB and the foreign key reference is the same for both attributes)


SPARQL Query

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?N ?CC
FROM <file:mondial.n3>
WHERE {?Co a :Country; :carCode ?CC; :hasCity ?Cty.
        ?Cty a :City; :name ?N
}
```

SQL Query

```
Select BGP12."?N" as "Result_?N", BGP12."?CC" as "Result_?CC"
FROM (
          SELECT Co_country."code" as "?CC", Cty_city."name" as "?N"
          FROM mondial_rel."country" Co_country, mondial_rel."city" Cty_city
          WHERE (Co_country."code" is not null) AND (Cty_city."country" = Co_country."code") AND
(Cty_city."country" is not null) AND (Cty_city."name" is not null)) BGP12
```

RODITest

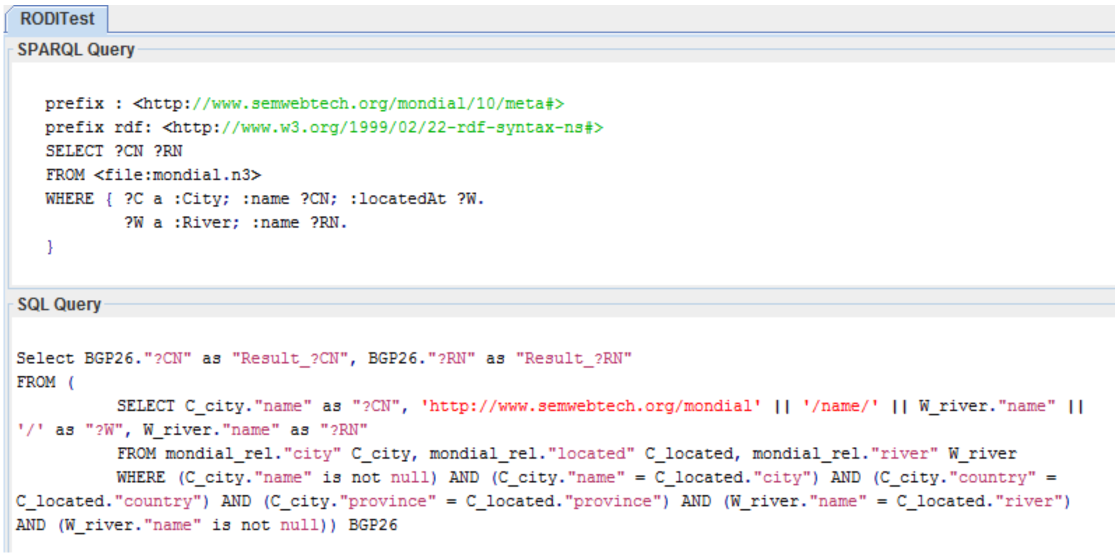
SPARQL Query

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?N ?CC
FROM <file:mondial.n3>
WHERE {?Co a :Country; :carCode ?CC.
        ?Cty a :City; :name ?N; :cityIn ?Co .
}
```

SQL Query

```
Select BGP19."?N" as "Result_?N", BGP19."?CC" as "Result_?CC"
FROM (
          SELECT Co_country."code" as "?CC", Cty_city."name" as "?N", 'http://www.semwebtech.org/mondial' ||
'/code/' || Cty_city."country" || '/' as "?Co"
          FROM mondial_rel."country" Co_country, mondial_rel."city" Cty_city
          WHERE (Co_country."code" is not null) AND (Cty_city."name" is not null) AND (Cty_city."country" =
Co_country."code") AND (Cty_city."country" is not null)) BGP19
```

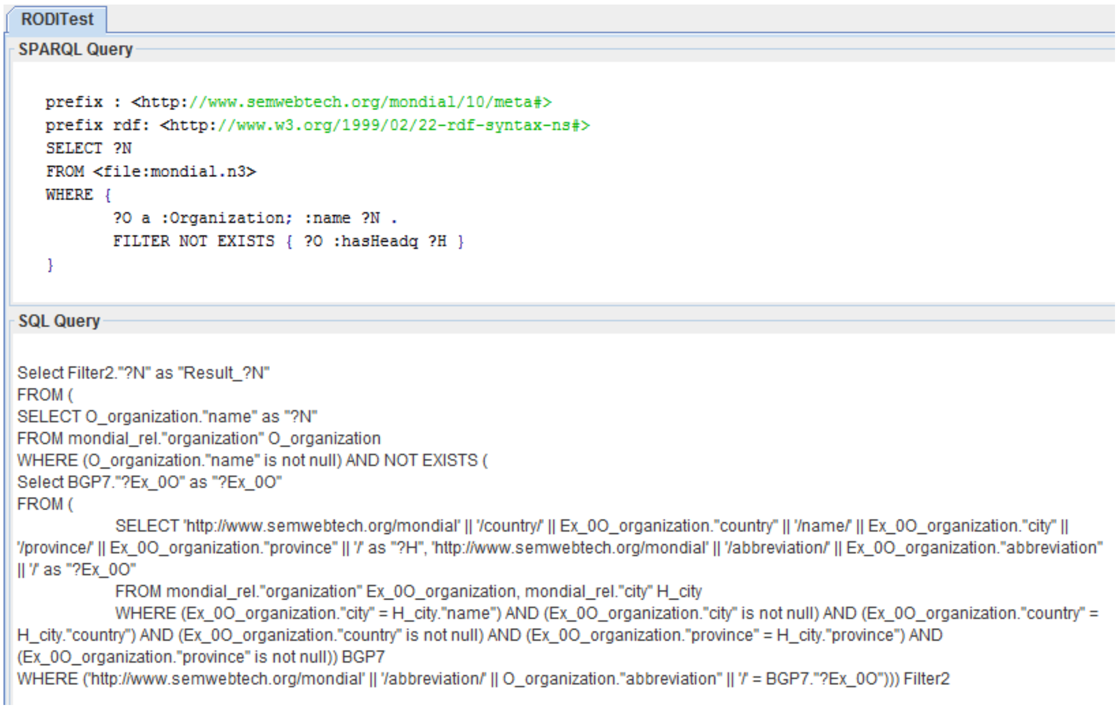
Figure 6.3: RODI 114a & 114b - Functional (bottom) and inverse functional (top) relationship



```
   prefix : <http://www.semwebtech.org/mondial/10/meta#>
   prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   SELECT ?CN ?RN
   FROM <file:mondial.n3>
   WHERE { ?C a :City; :name ?CN; :locatedAt ?W.
           ?W a :River; :name ?RN.
   }
```

```
Select BGP26."?CN" as "Result_?CN", BGP26."?RN" as "Result_?RN"
FROM (
          SELECT C_city."name" as "?CN", 'http://www.semwebtech.org/mondial' || '/name/' || W_river."name" ||
'/' as "?W", W_river."name" as "?RN"
          FROM mondial_rel."city" C_city, mondial_rel."located" C_located, mondial_rel."river" W_river
          WHERE (C_city."name" is not null) AND (C_city."name" = C_located."city") AND (C_city."country" =
C_located."country") AND (C_city."province" = C_located."province") AND (W_river."name" = C_located."river")
AND (W_river."name" is not null)) BGP26
```

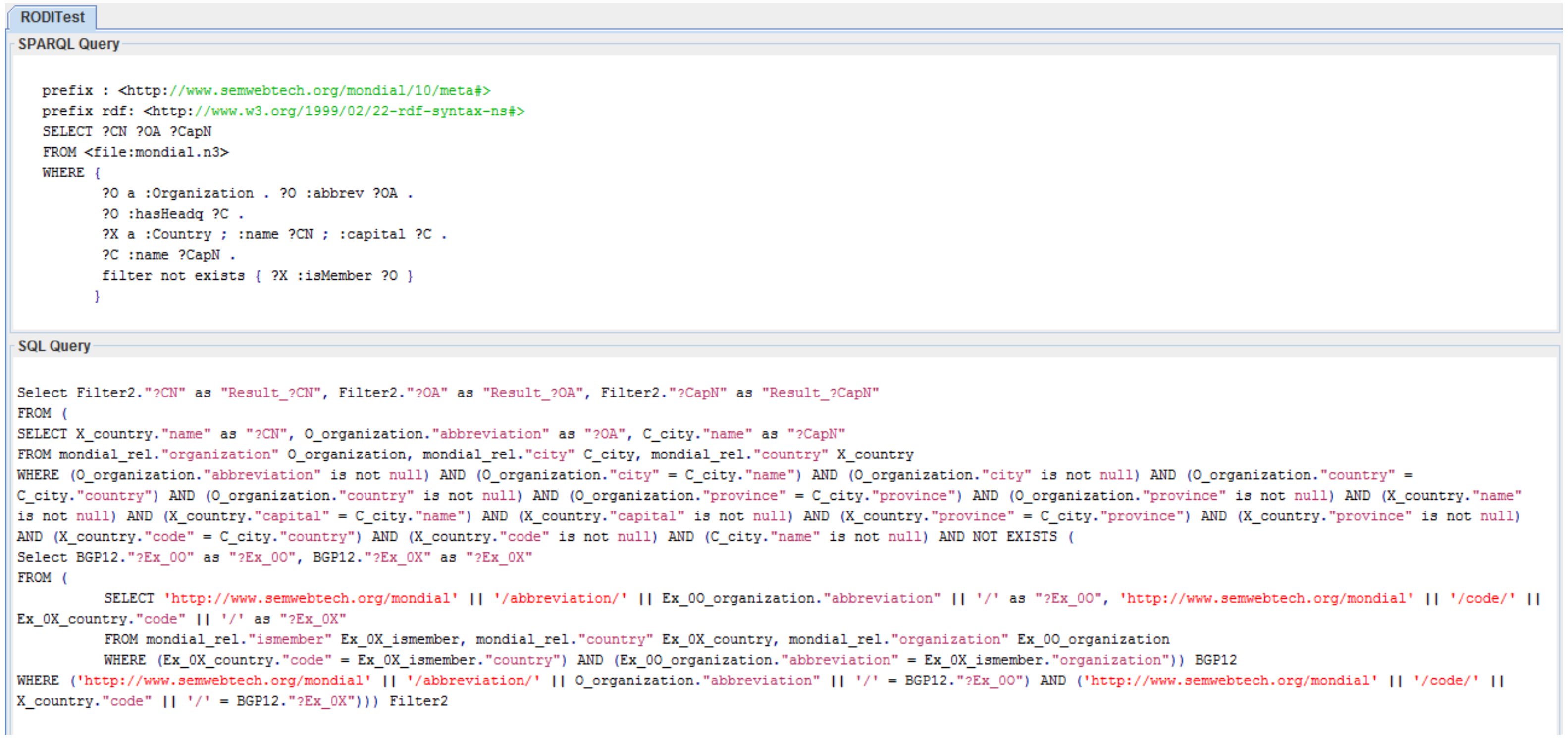
Figure 6.4: RODI 116 - NM relation with multi class property. *City.locatedAt* ranges over *Sea*, *Lake*, *River* and *ANY*.



```
   prefix : <http://www.semwebtech.org/mondial/10/meta#>
   prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   SELECT ?N
   FROM <file:mondial.n3>
   WHERE {
          ?O a :Organization; :name ?N .
          FILTER NOT EXISTS { ?O :hasHeadq ?H }
   }
```

```
Select Filter2."?N" as "Result_?N"
FROM (
SELECT O_organization."name" as "?N"
FROM mondial_rel."organization" O_organization
WHERE (O_organization."name" is not null) AND NOT EXISTS (
Select BGP7."?Ex_0O" as "?Ex_0O"
FROM (
          SELECT 'http://www.semwebtech.org/mondial' || '/country/' || Ex_0O_organization."country" || '/name/' || Ex_0O_organization."city" ||
'/province/' || Ex_0O_organization."province" || '/' as "?H", 'http://www.semwebtech.org/mondial' || '/abbreviation/' || Ex_0O_organization."abbreviation"
|| '/' as "?Ex_0O"
          FROM mondial_rel."organization" Ex_0O_organization, mondial_rel."city" H_city
          WHERE (Ex_0O_organization."city" = H_city."name") AND (Ex_0O_organization."city" is not null) AND (Ex_0O_organization."country" =
H_city."country") AND (Ex_0O_organization."country" is not null) AND (Ex_0O_organization."province" = H_city."province") AND
(Ex_0O_organization."province" is not null)) BGP7
WHERE ('http://www.semwebtech.org/mondial' || '/abbreviation/' || O_organization."abbreviation" || '/' = BGP7."?Ex_0O"))) Filter2
```

Figure 6.5: RODI 192 - Query with *NOT EXISTS*

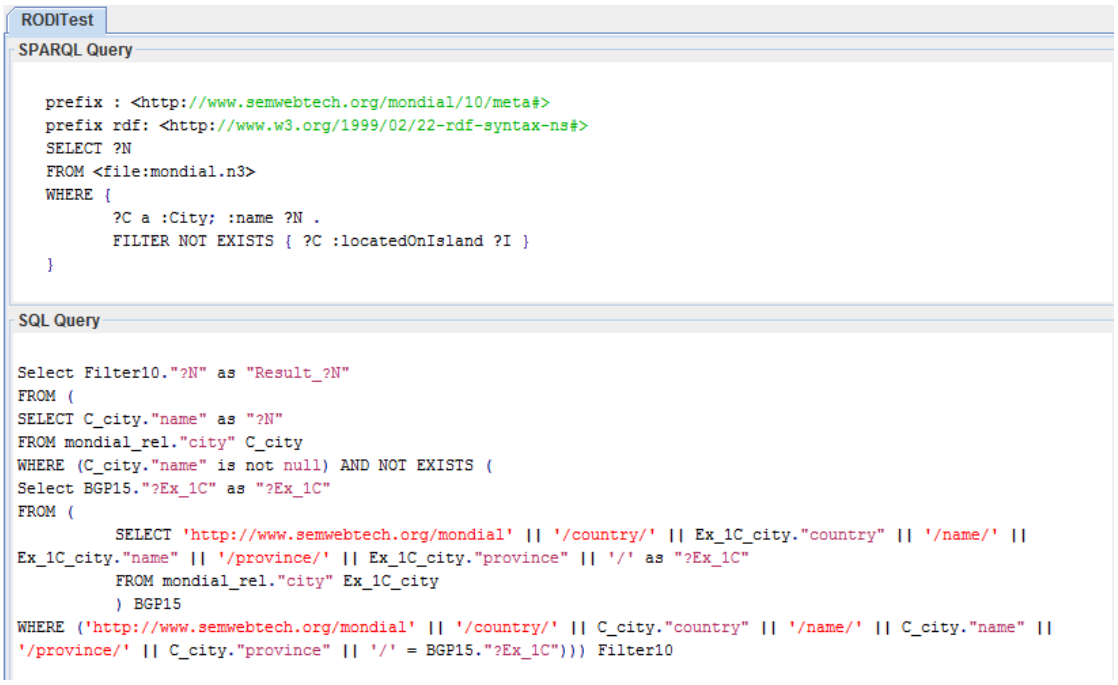


```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?CN ?OA ?CapN
FROM <file:mondial.n3>
WHERE {
        ?O a :Organization . ?O :abbrev ?OA .
        ?O :hasHeadq ?C .
        ?X a :Country ; :name ?CN ; :capital ?C .
        ?C :name ?CapN .
        filter not exists { ?X :isMember ?O }
       }
```

```
Select Filter2."?CN" as "Result_?CN", Filter2."?OA" as "Result_?OA", Filter2."?CapN" as "Result_?CapN"
FROM (
SELECT X_country."name" as "?CN", O_organization."abbreviation" as "?OA", C_city."name" as "?CapN"
FROM mondial_rel."organization" O_organization, mondial_rel."city" C_city, mondial_rel."country" X_country
WHERE (O_organization."abbreviation" is not null) AND (O_organization."city" = C_city."name") AND (O_organization."city" is not null) AND (O_organization."country" =
C_city."country") AND (O_organization."country" is not null) AND (O_organization."province" = C_city."province") AND (O_organization."province" is not null) AND (X_country."name"
is not null) AND (X_country."capital" = C_city."name") AND (X_country."capital" is not null) AND (X_country."province" = C_city."province") AND (X_country."province" is not null)
AND (X_country."code" = C_city."country") AND (X_country."code" is not null) AND (C_city."name" is not null) AND NOT EXISTS (
Select BGP12."?Ex_00" as "?Ex_00", BGP12."?Ex_0X" as "?Ex_0X"
FROM (
          SELECT 'http://www.semwebtech.org/mondial' || '/abbreviation/' || Ex_00_organization."abbreviation" || '/' as "?Ex_00", 'http://www.semwebtech.org/mondial' || '/code/' ||
Ex_0X_country."code" || '/' as "?Ex_0X"
          FROM mondial_rel."ismember" Ex_0X_ismember, mondial_rel."country" Ex_0X_country, mondial_rel."organization" Ex_00_organization
          WHERE (Ex_0X_country."code" = Ex_0X_ismember."country") AND (Ex_00_organization."abbreviation" = Ex_0X_ismember."organization")) BGP12
WHERE ('http://www.semwebtech.org/mondial' || '/abbreviation/' || O_organization."abbreviation" || '/' = BGP12."?Ex_00") AND ('http://www.semwebtech.org/mondial' || '/code/' ||
X_country."code" || '/' = BGP12."?Ex_0X"))) Filter2
```

Figure 6.6: RODI 183 - Very complex Query over the major political entities

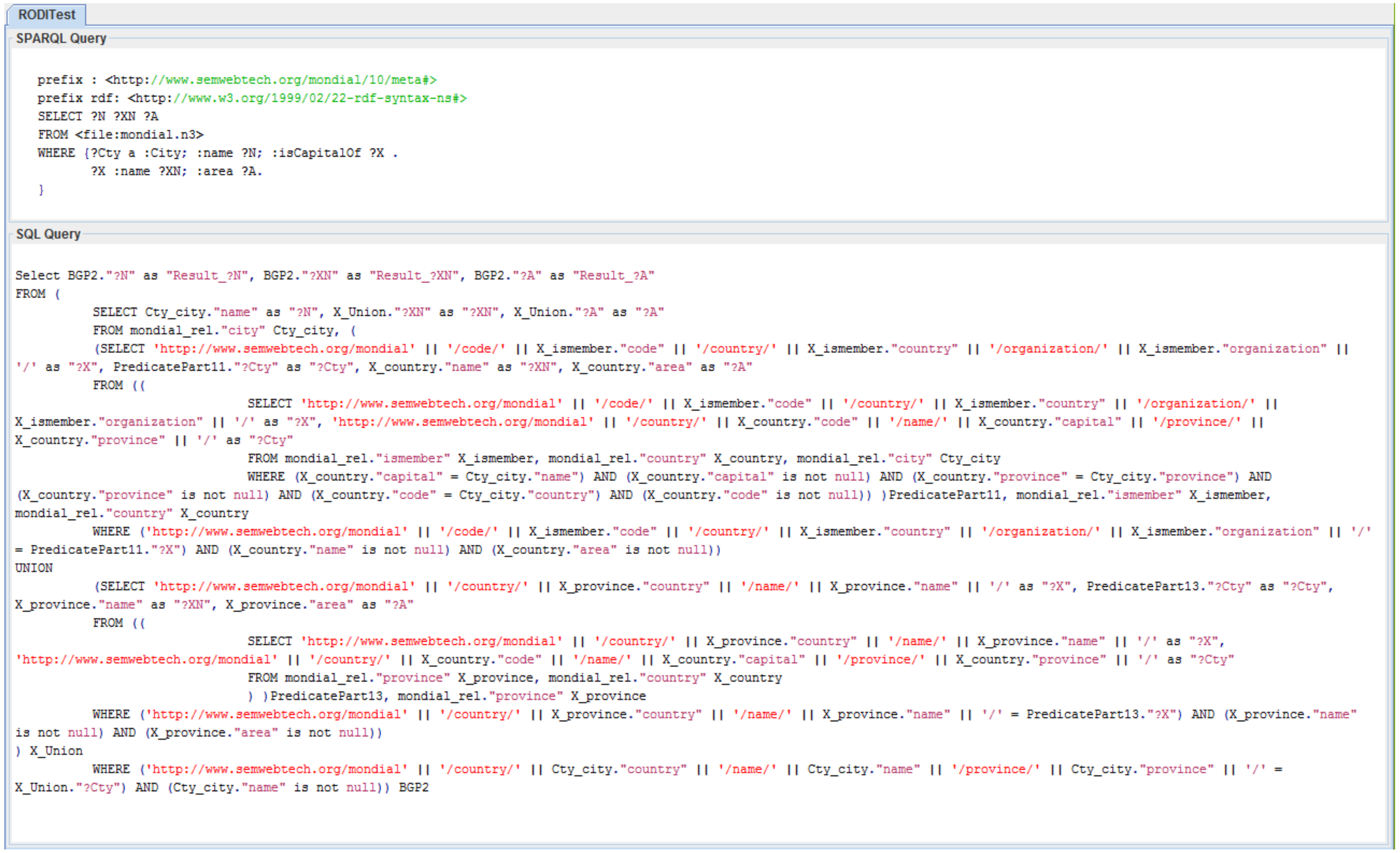
6.1.2 Not working examples



```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?N
FROM <file:mondial.n3>
WHERE {
        ?C a :City; :name ?N .
        FILTER NOT EXISTS { ?C :locatedOnIsland ?I }
}
```

```
Select Filter10."?N" as "Result_?N"
FROM (
SELECT C_city."name" as "?N"
FROM mondial_rel."city" C_city
WHERE (C_city."name" is not null) AND NOT EXISTS (
Select BGP15."?Ex_1C" as "?Ex_1C"
FROM (
          SELECT 'http://www.semwebtech.org/mondial' || '/country/' || Ex_1C_city."country" || '/name/' ||
Ex_1C_city."name" || '/province/' || Ex_1C_city."province" || '/' as "?Ex_1C"
          FROM mondial_rel."city" Ex_1C_city
          ) BGP15
WHERE ('http://www.semwebtech.org/mondial' || '/country/' || C_city."country" || '/name/' || C_city."name" ||
'/province/' || C_city."province" || '/' = BGP15."?Ex_1C"))) Filter10
```

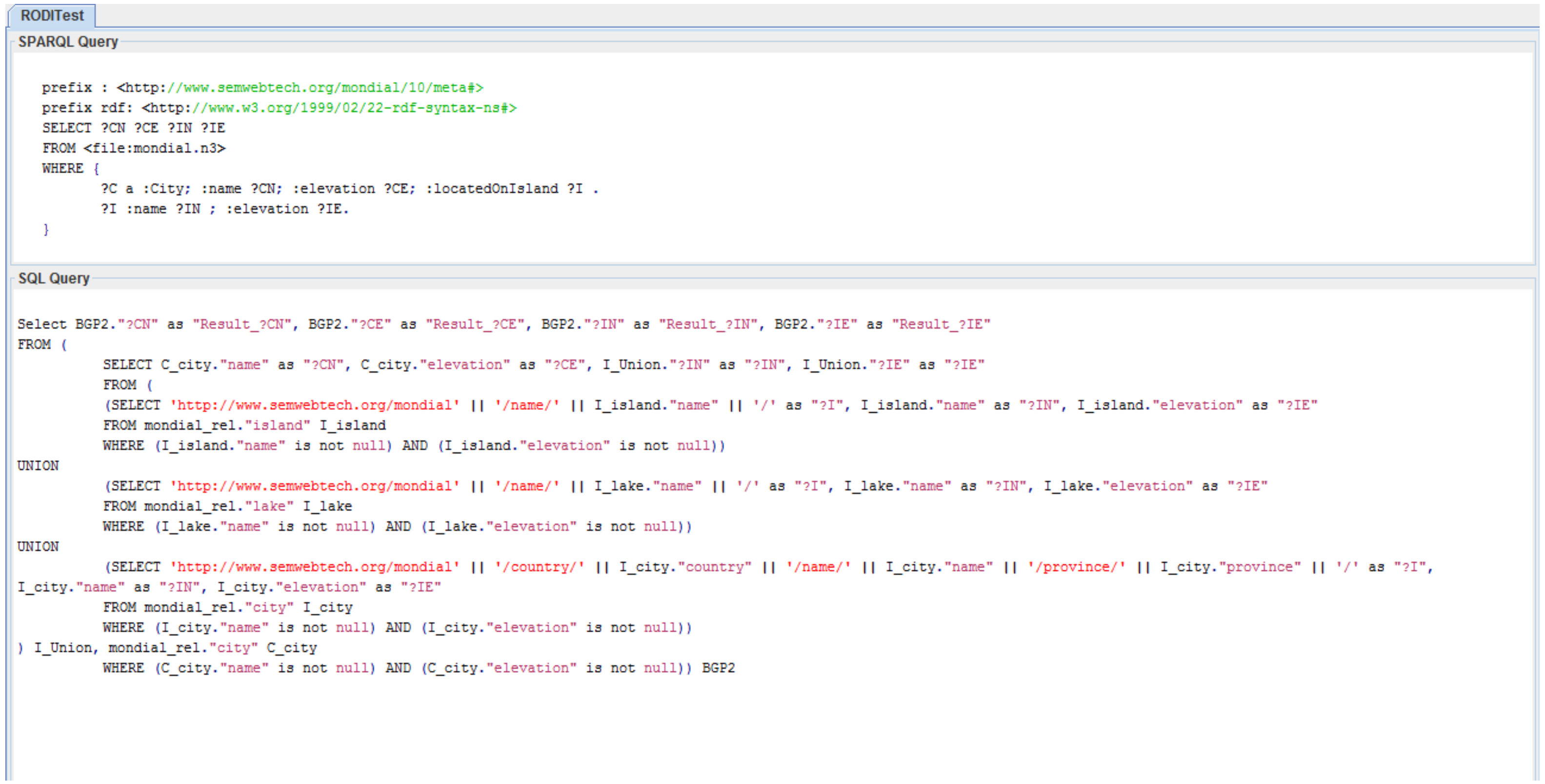
Figure 6.7: RODI 193 - Missing the attribute *locatedOnIsland*.



```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?N ?XN ?A
FROM <file:mondial.n3>
WHERE {?Cty a :City; :name ?N; :isCapitalOf ?X .
       ?X :name ?XN; :area ?A.
}
```

```
Select BGP2."?N" as "Result_?N", BGP2."?XN" as "Result_?XN", BGP2."?A" as "Result_?A"
FROM (
          SELECT Cty_city."name" as "?N", X_Union."?XN" as "?XN", X_Union."?A" as "?A"
          FROM mondial_rel."city" Cty_city, (
          (SELECT 'http://www.semwebtech.org/mondial' || '/code/' || X_ismember."code" || '/country/' || X_ismember."country" || '/organization/' || X_ismember."organization" ||
'/' as "?X", PredicatePart11."?Cty" as "?Cty", X_country."name" as "?XN", X_country."area" as "?A"
          FROM ((
                              SELECT 'http://www.semwebtech.org/mondial' || '/code/' || X_ismember."code" || '/country/' || X_ismember."country" || '/organization/' ||
X_ismember."organization" || '/' as "?X", 'http://www.semwebtech.org/mondial' || '/country/' || X_country."code" || '/name/' || X_country."capital" || '/province/' ||
X_country."province" || '/' as "?Cty"
                              FROM mondial_rel."ismember" X_ismember, mondial_rel."country" X_country, mondial_rel."city" Cty_city
                              WHERE (X_country."capital" = Cty_city."name") AND (X_country."capital" is not null) AND (X_country."province" = Cty_city."province") AND
(X_country."province" is not null) AND (X_country."code" = Cty_city."country") AND (X_country."code" is not null)) )PredicatePart11, mondial_rel."ismember" X_ismember,
mondial_rel."country" X_country
          WHERE ('http://www.semwebtech.org/mondial' || '/code/' || X_ismember."code" || '/country/' || X_ismember."country" || '/organization/' || X_ismember."organization" || '/'
= PredicatePart11."?X") AND (X_country."name" is not null) AND (X_country."area" is not null))
UNION
          (SELECT 'http://www.semwebtech.org/mondial' || '/country/' || X_province."country" || '/name/' || X_province."name" || '/' as "?X", PredicatePart13."?Cty" as "?Cty",
X_province."name" as "?XN", X_province."area" as "?A"
          FROM ((
                              SELECT 'http://www.semwebtech.org/mondial' || '/country/' || X_province."country" || '/name/' || X_province."name" || '/' as "?X",
'http://www.semwebtech.org/mondial' || '/country/' || X_country."code" || '/name/' || X_country."capital" || '/province/' || X_country."province" || '/' as "?Cty"
                              FROM mondial_rel."province" X_province, mondial_rel."country" X_country
                              ) )PredicatePart13, mondial_rel."province" X_province
          WHERE ('http://www.semwebtech.org/mondial' || '/country/' || X_province."country" || '/name/' || X_province."name" || '/' = PredicatePart13."?X") AND (X_province."name"
is not null) AND (X_province."area" is not null))
) X_Union
          WHERE ('http://www.semwebtech.org/mondial' || '/country/' || Cty_city."country" || '/name/' || Cty_city."name" || '/province/' || Cty_city."province" || '/' =
X_Union."?Cty") AND (Cty_city."name" is not null)) BGP2
```

Figure 6.8: RODI 117 - Bug which could be avoided with more time and are not directly connected to the database. The *key talbe* has a wrong entry for the key *ismember* is a wrong table.. Addtional the there is a problem with the weak entity of province.country.



```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?CN ?CE ?IN ?IE
FROM <file:mondial.n3>
WHERE {
        ?C a :City; :name ?CN; :elevation ?CE; :locatedOnIsland ?I .
        ?I :name ?IN ; :elevation ?IE.
}
```

```
Select BGP2."?CN" as "Result_?CN", BGP2."?CE" as "Result_?CE", BGP2."?IN" as "Result_?IN", BGP2."?IE" as "Result_?IE"
FROM (
          SELECT C_city."name" as "?CN", C_city."elevation" as "?CE", I_Union."?IN" as "?IN", I_Union."?IE" as "?IE"
          FROM (
          (SELECT 'http://www.semwebtech.org/mondial' || '/name/' || I_island."name" || '/' as "?I", I_island."name" as "?IN", I_island."elevation" as "?IE"
          FROM mondial_rel."island" I_island
          WHERE (I_island."name" is not null) AND (I_island."elevation" is not null))
UNION
          (SELECT 'http://www.semwebtech.org/mondial' || '/name/' || I_lake."name" || '/' as "?I", I_lake."name" as "?IN", I_lake."elevation" as "?IE"
          FROM mondial_rel."lake" I_lake
          WHERE (I_lake."name" is not null) AND (I_lake."elevation" is not null))
UNION
          (SELECT 'http://www.semwebtech.org/mondial' || '/country/' || I_city."country" || '/name/' || I_city."name" || '/province/' || I_city."province" || '/' as "?I",
I_city."name" as "?IN", I_city."elevation" as "?IE"
          FROM mondial_rel."city" I_city
          WHERE (I_city."name" is not null) AND (I_city."elevation" is not null))
) I_Union, mondial_rel."city" C_city
          WHERE (C_city."name" is not null) AND (C_city."elevation" is not null)) BGP2
```

Figure 6.9: RODI Query 122 - The property *locatedOnIsland* is not know. Therefore it is ignored and because the class of ?I is not defined it becames all classes with name and elevation. Without locatedOnIsland the join condition is also lost.

6.1.3 All Results

Query Number	Status	Notice
111	OK	
112	OK	
112b	OK	
113	OK	
114a	OK	
114b	OK	
115a	OK	
115b	OK	
116	OK	
117	Wrong	Capital for Province
118	Wrong	Capital for Province
121-126	Wrong	The Attribute LocatedOnIsland is not found
128	OK	
129	OK	
132a	Wrong	LocatedIn
137a	OK	
141	OK	
142	OK	
151-153	Wrong	The Attribute LocatedOnIsland is not found
161	OK	
162	Wrong	believeInReligion
171-173	Wrong	Symmetric reified does not work at the moment
177	Wrong	Symmetric reified does not work at the moment
181	OK	
182	OK	
183	OK	
184	OK	
191	OK	
192	OK	
193	Wrong	locatedOnIsland
211	OK	
212	OK	
213	Wrong	Can not find Mountains as class
214	Wrong	Source.Elevation not found
215	OK	
216	OK	
217	OK	
218	OK	

Table 6.1: Results of the RODI Benchmark

6.2 Problems

6.2.1 Parameters

The parameters decide which cases of matching can be done and if it works with the current kind of structures. Different threshold values or execution orders bring very different results. The default settings can deal with most situation and an automated setting choosing might help but for the critical cases it is not so easily and when one case is found, another one is not found anymore. Repeating everything with different settings and choosing the best results is also not an option, as long as the program can not evaluate the results on its own.

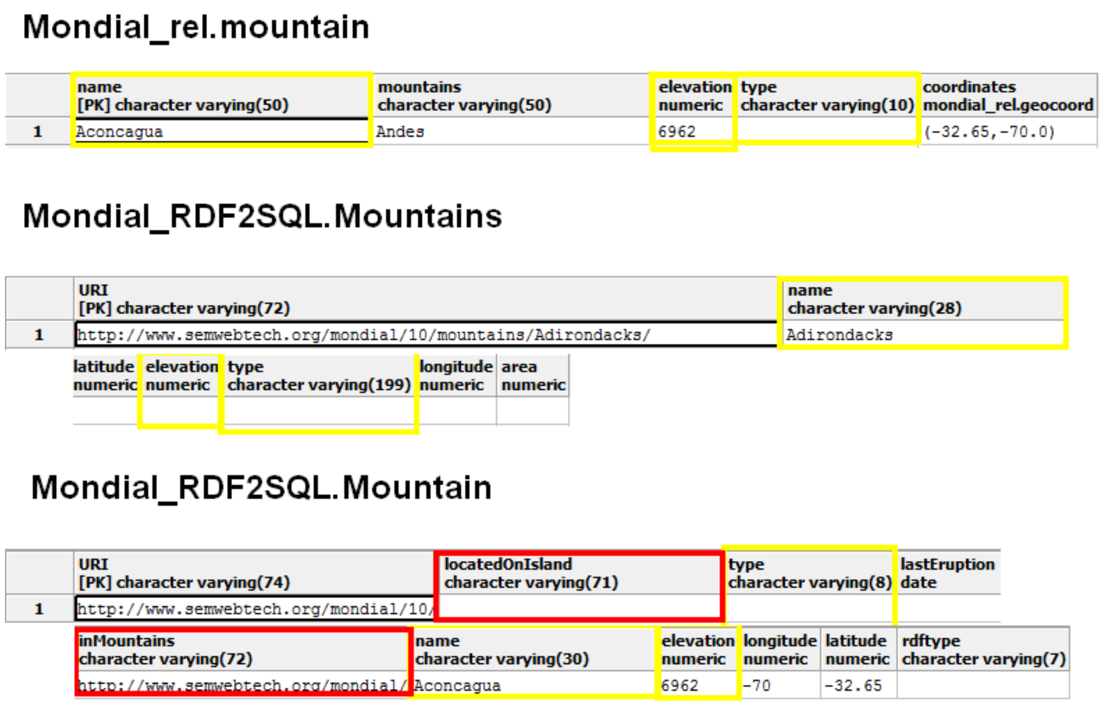
6.2.2 RODI query structure

The RODI [PBJR$^+$15] queries have the great weakness to rely very often on a specific attribute for multiple queries. Around this attribute successively more complex queries and different cases were built to test different scenarios. However, if this attribute is not found, all these queries can not be answered, no matter if the actual case which should be tested with the query could be handled if the basic attribute were found or not.

One of the major problems is the property LocatedOnIsland. Most oft the not working queries use this attribute but it is actually one of the hardest in the whole ontology, since there is no property with this name but several properties with similar names in both ontologies, Island itself is a hard class which can easy be mixed up with Islands and in the mondial_rel LocatedOnIsland splits into two different tables and in mondial_RDF2SQL_standard the city connection is nm and the mountain connection inverse functional.

6.2.3 Competing Classes weighting

A big problem are classes which should be matched but are stored extremely differently, due to other ways to design the ER-model. In Mondial the class Mountain from Mondial_RDF2SQL and the class mountain from Mondial_rel are a good example for this case. Actually Mountain should be matched to the class mountain, but the structure of them is quite different and there is a second class which has nearly the same scores and no other class to match to (see Figure 6.10). Therefore the classes Mountain and Mountains from Mondial_RDF2SQL compete for the same class in Mondial_rel. In those cases only the weighting of the punishment for not matchable foreign keys or names decide. On the one hand Mountain has the exact matching, while Mountains has only a 0.95 score due to the similarity. On the other hand the Mountain Mondial_RDF2SQL class has two foreign keys, which the mountain of Mondial_rel does not have as a foreign key but as a string value, since the class mountains itself does not exists in Mondial_rel. The decision which of both is more important, heavily depends on the relational model and can not be decided in general.



Mondial_rel.mountain

	name [PK] character varying(50)	mountains character varying(50)	elevation numeric	type character varying(10)	coordinates mondial_rel.geocoord
1	Aconcagua	Andes	6962		(-32.65,-70.0)

Mondial_RDF2SQL.Mountains

	URI [PK] character varying(72)	name character varying(28)
1	http://www.semwebtech.org/mondial/10/mountains/Adirondacks/	Adirondacks

latitude numeric	elevation numeric	type character varying(199)	longitude numeric	area numeric

Mondial_RDF2SQL.Mountain

	URI [PK] character varying(74)	locatedOnIsland character varying(71)	type character varying(8)	lastEruption date
1	http://www.semwebtech.org/mondial/10/			

inMountains character varying(72)	name character varying(30)	elevation numeric	longitude numeric	latitude numeric	rdftype character varying(7)
http://www.semwebtech.org/mondial/	Aconcagua	6962	-70	-32.65	

Figure 6.10: The Mountian(s) Tables of Mondial_rdf2sql and Mondial_rel
Shared attributes of all three are marked in yellow; Foreign keys which do not match are marked in red.

6.2.4 Attributes in other class

A problem which the SM can not solve at all is the case where attributes are in other classes. For example the *population* of a city is in Mondial_RDF2SQL in an attribute of *City* and in Mondial_rel this attribute is part of the class *cityprops.* Since the class *cityprops* is not just a class extension table but a functional relation of *city*, because a *city* can have multiple ones for different dates, this has to be considered a different class.

6.2.5 Indistinguishable Attributes

In some cases it is just a game of chances, if the attributes match right or wrong, since two or more attributes of two classes are indistinguishable.

6.2.6 Complex joining

This problem occur when the same property has to different complex structures in the ontology like Mondial_RDF2SQL.locatedOnIsland has. In Mondial_RDF2SQL this property has two ranges, *City* and *Mountain,* in Mondial_rel this property is split into locatedOnIsland for *city* and mountainLocatedonIsland for *mountain*.

- While the relationship to mountain is inverse functional, the relationship to city is NM.
- LoactedOnIsland is also a relation to Water and subclasses,

6.3 Further works

As already mentioned this problem is far from being solved and might never be solved entirely, but this does not mean that one should not try. The major weak point is the lack of well designed test databases with automated result checking for evaluating. The RODI [PBJR$^+$15] benchmark is a good attempt to do so, however it is too complicated to setup for not involved persons. This means the full testing can only be done after publishing the converter, which is usually too late, or with salvageable fragments of the benchmark, like it was done in this thesis. However, with such a limited test sample, there is always the risk of overfitting or that some cases are not dealt with.

Another major point is the automation of the parametrization and the extension of the evaluation. By improving parametrization to an automatic progress the results could have boosted directly and the versatility of the program could be improved. Also an automated internal overall evaluations would give the opportunity for multiple runs or learning algorithms.

Further it seems to me that there is a lack of information exchange between the different OBDA developers, which is rather ironic, since the idea behind this OBDA is combining knowledge. For some tools it has proven very hard if not impossible to get any detailed information or run this tools on ones own computer. Most publications are very superficial and are more about the individual performances than about sharing details. During the development several problems arose, where it was necessary to decide for one or the other way (e.g. Name matching algorithm s.3.2.3.1, matrix combination s.3.7, and step evaluation 3.2.4.3). These must be problems in which the other ODBA tool developer also had run into and an open discussion about those problems would promised a much more well-founded and better performing solution than making the same mistakes or missing the alternatives.

Bibliography

[Age] C. I. Agency. The World Factbook 2013-14. 1.2

[Apaa] Apache. Jena. 2.2.4, 2.3.2, 5

[Apab] Apache. SPARQL Tutorial. Website. `http://jena.apache.org/tutorials/sparql.html`; visited on 2016.11.20. (document), 1, 2.3.1, 2.3.2

[CGJR11] B. Cuenca Grau and E. Jimenez-Ruiz. LogMap: Logic- Based and Scalable Ontology Matching. 2011. (document), 1.3, 1.3.1, 1.4, 1.3.1, 1.3.1, 1.4, 3.2.5.2

[Che76] P. P.-S. Chen. The Entity-relationship Model&Mdash;Toward a Unified View of Data. *ACM Trans. Database Syst.*, 1(1):9–36, March 1976. 2.2.2

[Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387, June 1970. 2.2.1

[DHEM] E. J.-R. E. K. Dmitriy, Z. I. Horrocks, C. P. M. G. S. Evgenij, and T. J. Mora. BOOTOX: Bootstrapping OWL 2 ontologies and R2RML mappings from Relational Databases. 1.3.3

[dMPC15] L. F. de Medeiros, F. Priyatna, and O. Corcho. *MIRROR: Automatic R2RML Mapping Generation from Relational Databases*, pp. 326–343. Springer International Publishing, Cham, 2015. 1.3, 1.3.2

[ea05] D. A. et al. Schema and Ontology Matching with COMA++. *SIGMOD*, 2005. 1.3, 1.3.2, 1.4

[ea12] C. A. K. et al. Semi-Automatically Mapping Structured Sources into the Semantic Web. *ESWC*, 2012. 1.3, 1.3.3, 1.4

[ea14] T. B. et al. The ontop Framework for Ontology Based Data Access. *CSWS*, 2014. 1.3, 1.3.2, 1.3.3, 1.4

[FLO] FLORID. 1.2

[Fou] E. Foundation. Eclipse Kepler Service Release 2. Website. `https://eclipse.org/`; visited on 2016.11.20. 2.1.1

[GHM+08] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler. OWL 2: The next step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322, 2008. 2.2.5

[Groa] T. P. G. D. Group. PostgreSQL Download. Website. `https://www.postgresql.org/download/`; visited on October 11st 2016. 2.1.1

[Grob] T. P. G. D. Group. PostgreSQL Homepage. Website. `https://www.postgresql.org/`;. 2.1.1, 2.2.1, 5

[Hab61] J. Habr. Die Frequenzmethode zur Loesung der Transportprobleme und verwandter linearer Programmierungsprobleme. *Wissenschaftliche Zeitung der Universitaet Dresden*, (10), 1961. 3.2, 3.2.6.1

[HHMW12] V. Haarslev, K. Hidde, R. Möller, and M. Wessel. The RacerPro knowledge representation and reasoning system. *Semantic Web Journal*, 3(3):267–277, 2012. 2.2.5

[Int] Intel. Intel-Core-i5-4670-Processor. Website. `http://ark.intel.com/de/products/75047/Intel-Core-i5-4670-Processor-6M-Cache-up-to-3_80-GHz`;. 5.1

[JRKZ+15] E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, C. Pinkel, M. G. Skjæveland, E. Thorstensen, and J. Mora. *BootOX: Practical Mapping of RDBs to OWL 2*, pp. 113–132. Springer International Publishing, Cham, 2015. 1.3, 1.3.2, 1.3.3

[KFNM04] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The Protégé OWL plugin: An open development environment for semantic web applications. In *International Semantic Web Conference*, pp. 229–243. Springer, 2004. 2.2.5

[Lim] S. Limited. SAXON XSLT. Website. `http://saxon.sourceforge.net`; visited on 2016.11.20. 5

[May] W. May. The Mondial Database. Website. `https://www.dbis.informatik.uni-goettingen.de/Mondial/`;. 1.3, 2.1

[May99] W. May. Information Extraction and Integration with FLORID: The MONDIAL Case Study. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999. Available from `http://dbis.informatik.uni-goettingen.de/Mondial`. 1, 2.1.2

[Ora] Oracle. Oracle Database 12.1.0.1 JDBC Driver. Website. `http://www.oracle.com/technetwork/database/features/jdbc/jdbc-drivers-12c-download-1958347.htm`; visited on 2016.11.20. 5

[Ost] S. Ostermiller. Syntax Highlighting. Website. `http://ostermiller.org/syntax/`; visited on 2016.11.20. 5

[OWL] OWL. Icon. Website. `http://martin.dzborovci.com/images/owl.png`; visited on 2016.11.20. 2.5

[PBJR+15] C. Pinkel, C. Binnig, E. Jiménez-Ruiz, W. May, D. Ritze, M. G. Skjæveland, A. Solimando, and E. Kharlamov. *RODI: A Benchmark for Automatic Mapping Generation in*

Relational-to-Ontology Data Integration, pp. 21–37. Springer International Publishing, Cham, 2015. 1, 1.3.2, 1.3.3, 1.4, 2.1.2, 3.2, 6, 6.1, 6.2.2, 6.3

[PBKH13] l. C. Pinke, C. Binnig, E. Kharlamov, and P. Haase. IncMap: Pay As You Go Matching of Relational Schemata to OWL Ontologies. In *Proceedings of the 8th International Conference on Ontology Matching - Volume 1111*, OM'13, pp. 37–48, Aachen, Germany, Germany, 2013. CEUR-WS.org. 1.3, 1.3.2, 1.4

[pDT] pgAdmin Development Team. pg Admin III. Website. `https://www.pgadmin.org/`; visited on 2016.11.20. 2.1.1

[RDF] RDF. (document)

[RSa] L. Runge and S. Schrage. Auswertung von SPARQL - Anfragen mit relationaler Speicherung. (document), 1, 2.1.2, 2.3, 2.3.1, 2.7, 2.3.2, 2.3.3, 3.2, 4, 4.1.1, 5.1, 5.1.2, 5.1.4

[RSb] L. Runge and S. Schrage. RDF2SQL Schema Matcher. (document)

[RS13] L. Runge and S. Schrage. Forschungspraktikumsbericht RDF-to-SQL-Konverter. Technical report, Universität Göttingen, Institut für Informatik, 2013. 2.3, 2.3.1, 3.2, 4

[Run] L. Runge. Extraction of Ontological Metadata and Generation of an OBDA Mapping from a Relational Schema. (document), 1, 2.3.3.6, 2.3.3.7, 3.2, 4

[SM90] P. T. Silvano Martello. *Knapsack problems*. John Wiley & Sons Ltd., 1990. 3.2.6

[SPG$^+$07] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007. 2.2.5, 5

[SQL] SQL. (document), 1, 2.2.1, 2.3.1, 2.3.2

[SSK05] G. Stoilos, G. Stamou, and S. Kollias. A string metric for ontology alignment. In *International Semantic Web Conference*, pp. 624–637. Springer, 2005. 4b

[TH] W. M. Thomas Hornung. Efficient, Schema-Aware Storage of RDF. (document), 1, 2.3.1, 2.3.2

[TH06] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *International Joint Conference on Automated Reasoning*, pp. 292–297. Springer, 2006. 2.2.5

[The] Open Office Thesaurus. 3.2.3.1

[TSK] P.-N. Tan, M. Steinbach, and V. Kumar. Introduction to Data Mining. 2.4.1

[W3Ca] W3C. W3C Notation3. Website. `https://www.w3.org/TeamSubmission/n3/`;. 2.2.4

[W3Cb] W3C. W3C OWL 2 Web Ontology Language Recommandation. Website. `https://www.w3.org/TR/2009/REC-owl2-syntax-20091027/`;. 2.2.5

[W3Cc] W3C. W3C RDF 1.1 Recommendation. Website. `https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#section-Introduction`;. 1, 2.2.4, 2.2.6, 2.3.1

[W3Cd] W3C. W3C XML. Website. `https://www.w3.org/XML/`;. 2.2.4, 2.2.5

[WC3] WC3. WC3 Recommendation. 2.2.4

List of Algorithms

Appendix A

QueryConverter

A.1 Main Classes

A.2 Added Classes

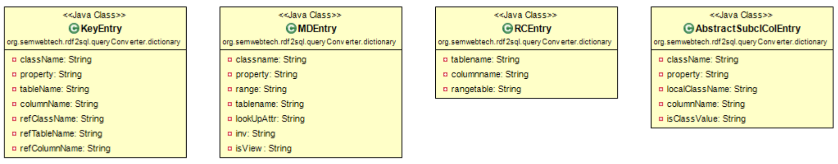


Figure A.3: UML Diagram of the different Entry classes of the Mapping Dictionary

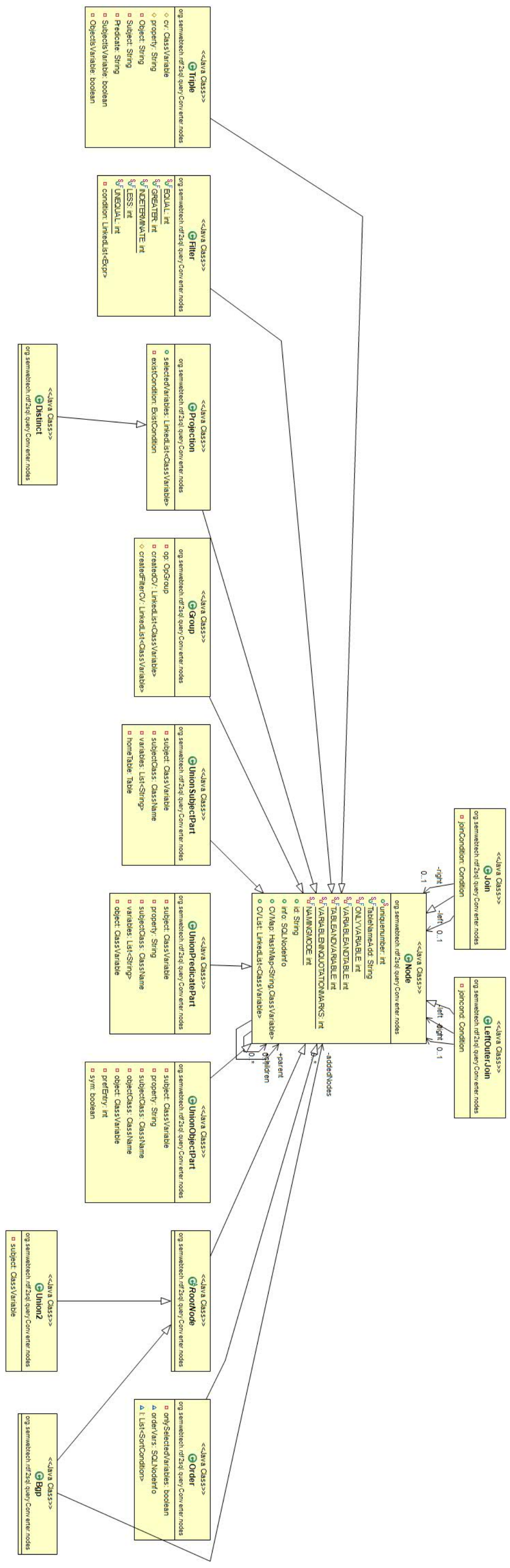
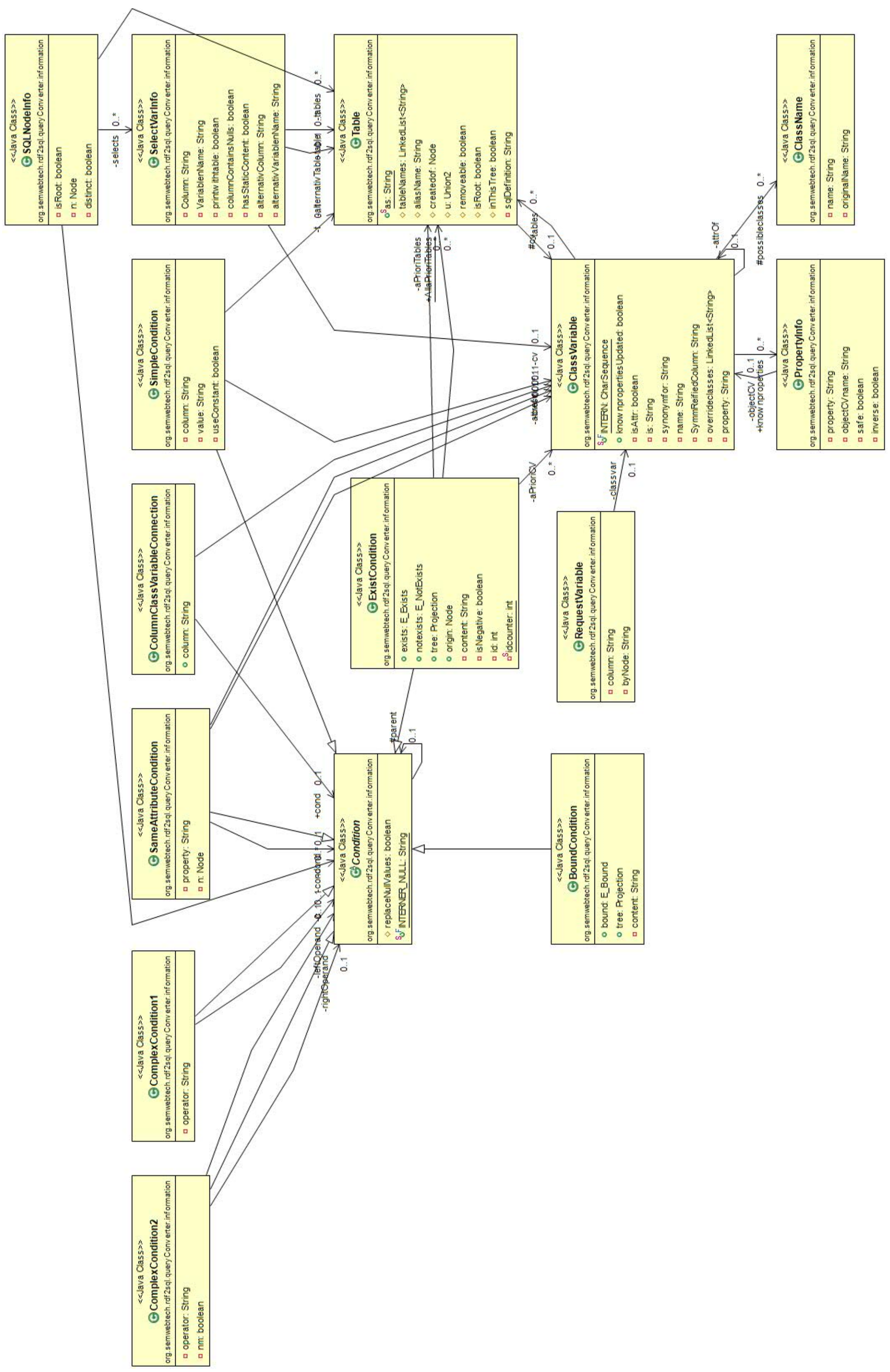


Figure A.1: UML Diagram - Node and subclasses



<<Java Class>>
SQLNodeInfo
org.semwebtech.rdf2sql.queryConverter.information
isRoot: boolean
n: Node
distinct: boolean
-selects 0..*
<<Java Class>>
SelectVarInfo
org.semwebtech.rdf2sql.queryConverter.information
Column: String
VariablenName: String
printwithtable: boolean
columnContainsNulls: boolean
hasStaticContent: boolean
alternativColumn: String
alternativVariablenName: String
-tables 0..*
-table
-alternativTable
<<Java Class>>
Table
org.semwebtech.rdf2sql.queryConverter.information
as: String
tableNames: LinkedList<String>
aliasName: String
createdof: Node
u: Union2
removeable: boolean
isRoot: boolean
inThisTree: boolean
sqlDefinition: String
#ctables 0..*
0..1
-aPrioriTables
+AliaPrioriTables
0..*
<<Java Class>>
ClassName
org.semwebtech.rdf2sql.queryConverter.information
name: String
originalName: String
#possibleclasses 0..*
-attrOf
0..1
<<Java Class>>
SimpleCondition
org.semwebtech.rdf2sql.queryConverter.information
column: String
value: String
useConstant: boolean
<<Java Class>>
ClassVariable
org.semwebtech.rdf2sql.queryConverter.information
INTERN: CharSequence
know npropertiesUpdated: boolean
isAttr: boolean
is: String
synonymfor: String
name: String
SymmReifiedColumn: String
overrideclasses: LinkedList<String>
property: String
-cv
0..1
-aPrioriCV
0..*
-objectCV 0..1
+know nproperties 0..*
<<Java Class>>
PropertyInfo
org.semwebtech.rdf2sql.queryConverter.information
property: String
objectCVname: String
safe: boolean
inverse: boolean
<<Java Class>>
ColumnClassVariableConnection
org.semwebtech.rdf2sql.queryConverter.information
column: String
<<Java Class>>
ExistCondition
org.semwebtech.rdf2sql.queryConverter.information
exists: E_Exists
notexists: E_NotExists
tree: Projection
origin: Node
content: String
isNegative: boolean
id: int
idcounter: int
-classvar
0..1
<<Java Class>>
RequestVariable
org.semwebtech.rdf2sql.queryConverter.information
column: String
byNode: String
<<Java Class>>
SameAttributeCondition
org.semwebtech.rdf2sql.queryConverter.information
property: String
n: Node
+cond 0..1
#parent
0..1
<<Java Class>>
Condition
org.semwebtech.rdf2sql.queryConverter.information
replaceNullValues: boolean
INTERNER_NULL: String
-leftOperand
-rightOperand
0..1
<<Java Class>>
BoundCondition
org.semwebtech.rdf2sql.queryConverter.information
bound: E_Bound
tree: Projection
content: String
<<Java Class>>
ComplexCondition1
org.semwebtech.rdf2sql.queryConverter.information
operator: String
<<Java Class>>
ComplexCondition2
org.semwebtech.rdf2sql.queryConverter.information
operator: String
nm: boolean

Figure A.2:

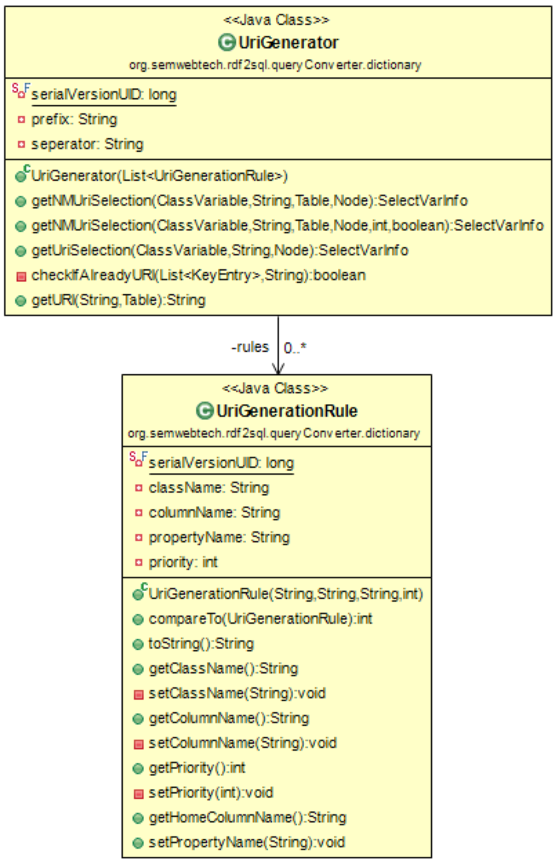

<<Java Class>>
UriGenerator
org.semwebtech.rdf2sql.queryConverter.dictionary
serialVersionUID: long
prefix: String
seperator: String
UriGenerator(List<UriGenerationRule>)
getNMUriSelection(ClassVariable,String,Table,Node):SelectVarInfo
getNMUriSelection(ClassVariable,String,Table,Node,int,boolean):SelectVarInfo
getUriSelection(ClassVariable,String,Node):SelectVarInfo
checkIfAlreadyURI(List<KeyEntry>,String):boolean
getURI(String,Table):String
-rules
0..*
<<Java Class>>
UriGenerationRule
org.semwebtech.rdf2sql.queryConverter.dictionary
serialVersionUID: long
className: String
columnName: String
propertyName: String
priority: int
UriGenerationRule(String,String,String,int)
compareTo(UriGenerationRule):int
toString():String
getClassName():String
setClassName(String):void
getColumnName():String
setColumnName(String):void
getPriority():int
setPriority(int):void
getHomeColumnName():String
setPropertyName(String):void

Figure A.4: UML Diagram of the URIGenerator and the URIGeneratorRule

Appendix B

SchemaMatcher

Uml Diagramms of the different SchemaMatcher packages

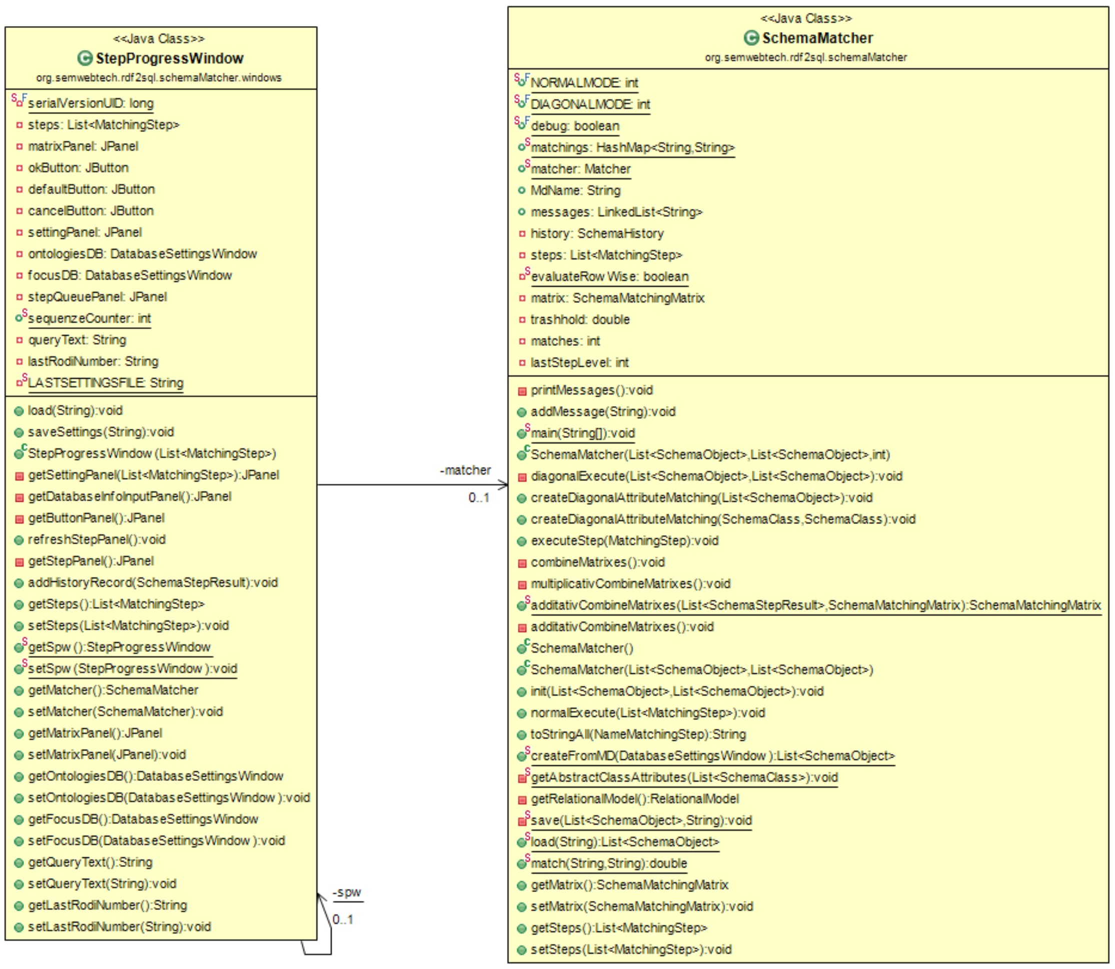

<<Java Class>>
StepProgressWindow
org.semwebtech.rdf2sql.schemaMatcher.windows
serialVersionUID: long
steps: List<MatchingStep>
matrixPanel: JPanel
okButton: JButton
defaultButton: JButton
cancelButton: JButton
settingPanel: JPanel
ontologiesDB: DatabaseSettingsWindow
focusDB: DatabaseSettingsWindow
stepQueuePanel: JPanel
sequenzeCounter: int
queryText: String
lastRodiNumber: String
LASTSETTINGSFILE: String
load(String):void
saveSettings(String):void
StepProgressWindow(List<MatchingStep>)
getSettingPanel(List<MatchingStep>):JPanel
getDatabaseInfoInputPanel():JPanel
getButtonPanel():JPanel
refreshStepPanel():void
getStepPanel():JPanel
addHistoryRecord(SchemaStepResult):void
getSteps():List<MatchingStep>
setSteps(List<MatchingStep>):void
getSpw():StepProgressWindow
setSpw(StepProgressWindow):void
getMatcher():SchemaMatcher
setMatcher(SchemaMatcher):void
getMatrixPanel():JPanel
setMatrixPanel(JPanel):void
getOntologiesDB():DatabaseSettingsWindow
setOntologiesDB(DatabaseSettingsWindow):void
getFocusDB():DatabaseSettingsWindow
setFocusDB(DatabaseSettingsWindow):void
getQueryText():String
setQueryText(String):void
getLastRodiNumber():String
setLastRodiNumber(String):void
-matcher
0..1
-spw
0..1
<<Java Class>>
SchemaMatcher
org.semwebtech.rdf2sql.schemaMatcher
NORMALMODE: int
DIAGONALMODE: int
debug: boolean
matchings: HashMap<String,String>
matcher: Matcher
MdName: String
messages: LinkedList<String>
history: SchemaHistory
steps: List<MatchingStep>
evaluateRowWise: boolean
matrix: SchemaMatchingMatrix
trashhold: double
matches: int
lastStepLevel: int
printMessages():void
addMessage(String):void
main(String[]):void
SchemaMatcher(List<SchemaObject>,List<SchemaObject>,int)
diagonalExecute(List<SchemaObject>,List<SchemaObject>):void
createDiagonalAttributeMatching(List<SchemaObject>):void
createDiagonalAttributeMatching(SchemaClass,SchemaClass):void
executeStep(MatchingStep):void
combineMatrixes():void
multiplicativCombineMatrixes():void
additativCombineMatrixes(List<SchemaStepResult>,SchemaMatchingMatrix):SchemaMatchingMatrix
additativCombineMatrixes():void
SchemaMatcher()
SchemaMatcher(List<SchemaObject>,List<SchemaObject>)
init(List<SchemaObject>,List<SchemaObject>):void
normalExecute(List<MatchingStep>):void
toStringAll(NameMatchingStep):String
createFromMD(DatabaseSettingsWindow):List<SchemaObject>
getAbstractClassAttributes(List<SchemaClass>):void
getRelationalModel():RelationalModel
save(List<SchemaObject>,String):void
load(String):List<SchemaObject>
match(String,String):double
getMatrix():SchemaMatchingMatrix
setMatrix(SchemaMatchingMatrix):void
getSteps():List<MatchingStep>
setSteps(List<MatchingStep>):void

Figure B.1: UML Diagram of the main class SchemaMatcher and the UI main class StepProgress-Window

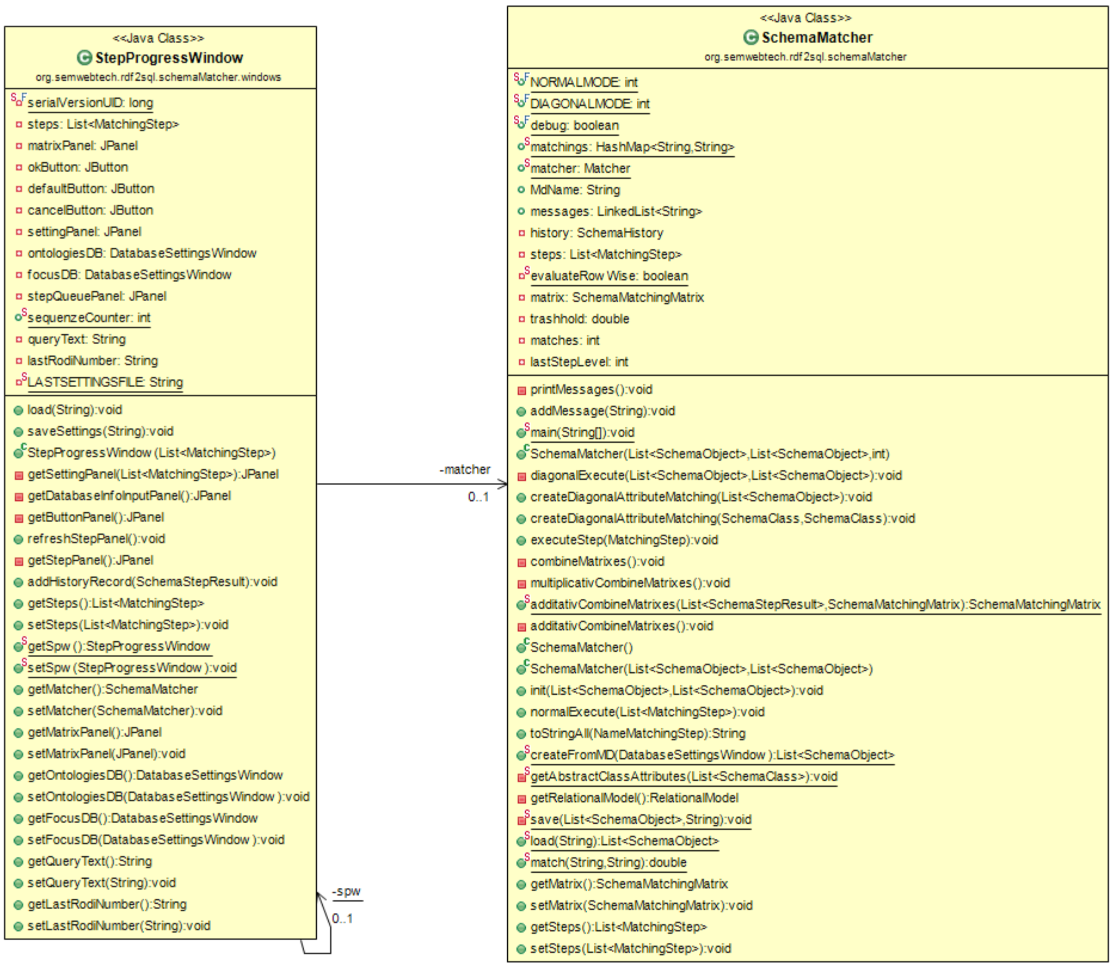

<<Java Class>>
StepProgressWindow
org.semwebtech.rdf2sql.schemaMatcher.windows
serialVersionUID: long
steps: List<MatchingStep>
matrixPanel: JPanel
okButton: JButton
defaultButton: JButton
cancelButton: JButton
settingPanel: JPanel
ontologiesDB: DatabaseSettingsWindow
focusDB: DatabaseSettingsWindow
stepQueuePanel: JPanel
sequenzeCounter: int
queryText: String
lastRodiNumber: String
LASTSETTINGSFILE: String
load(String):void
saveSettings(String):void
StepProgressWindow(List<MatchingStep>)
getSettingPanel(List<MatchingStep>):JPanel
getDatabaseInfoInputPanel():JPanel
getButtonPanel():JPanel
refreshStepPanel():void
getStepPanel():JPanel
addHistoryRecord(SchemaStepResult):void
getSteps():List<MatchingStep>
setSteps(List<MatchingStep>):void
getSpw():StepProgressWindow
setSpw(StepProgressWindow):void
getMatcher():SchemaMatcher
setMatcher(SchemaMatcher):void
getMatrixPanel():JPanel
setMatrixPanel(JPanel):void
getOntologiesDB():DatabaseSettingsWindow
setOntologiesDB(DatabaseSettingsWindow):void
getFocusDB():DatabaseSettingsWindow
setFocusDB(DatabaseSettingsWindow):void
getQueryText():String
setQueryText(String):void
getLastRodiNumber():String
setLastRodiNumber(String):void
-spw
0..1
-matcher
0..1
<<Java Class>>
SchemaMatcher
org.semwebtech.rdf2sql.schemaMatcher
NORMALMODE: int
DIAGONALMODE: int
debug: boolean
matchings: HashMap<String,String>
matcher: Matcher
MdName: String
messages: LinkedList<String>
history: SchemaHistory
steps: List<MatchingStep>
evaluateRowWise: boolean
matrix: SchemaMatchingMatrix
trashhold: double
matches: int
lastStepLevel: int
printMessages():void
addMessage(String):void
main(String[]):void
SchemaMatcher(List<SchemaObject>,List<SchemaObject>,int)
diagonalExecute(List<SchemaObject>,List<SchemaObject>):void
createDiagonalAttributeMatching(List<SchemaObject>):void
createDiagonalAttributeMatching(SchemaClass,SchemaClass):void
executeStep(MatchingStep):void
combineMatrixes():void
multiplicativCombineMatrixes():void
additativCombineMatrixes(List<SchemaStepResult>,SchemaMatchingMatrix):SchemaMatchingMatrix
additativCombineMatrixes():void
SchemaMatcher()
SchemaMatcher(List<SchemaObject>,List<SchemaObject>)
init(List<SchemaObject>,List<SchemaObject>):void
normalExecute(List<MatchingStep>):void
toStringAll(NameMatchingStep):String
createFromMD(DatabaseSettingsWindow):List<SchemaObject>
getAbstractClassAttributes(List<SchemaClass>):void
getRelationalModel():RelationalModel
save(List<SchemaObject>,String):void
load(String):List<SchemaObject>
match(String,String):double
getMatrix():SchemaMatchingMatrix
setMatrix(SchemaMatchingMatrix):void
getSteps():List<MatchingStep>
setSteps(List<MatchingStep>):void

Figure B.2: UML Diagram - SchemaMatcher

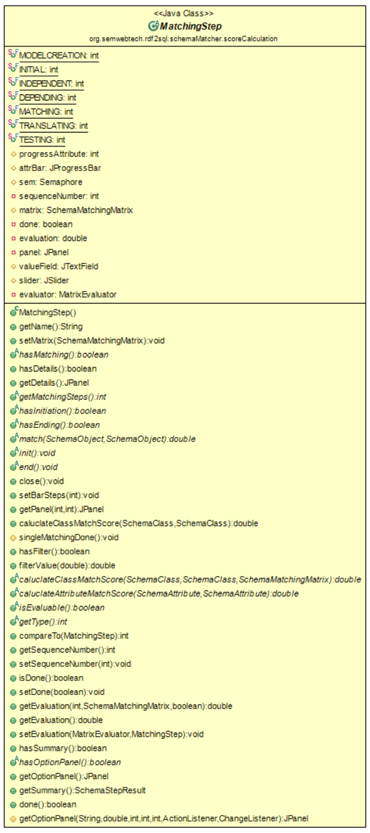
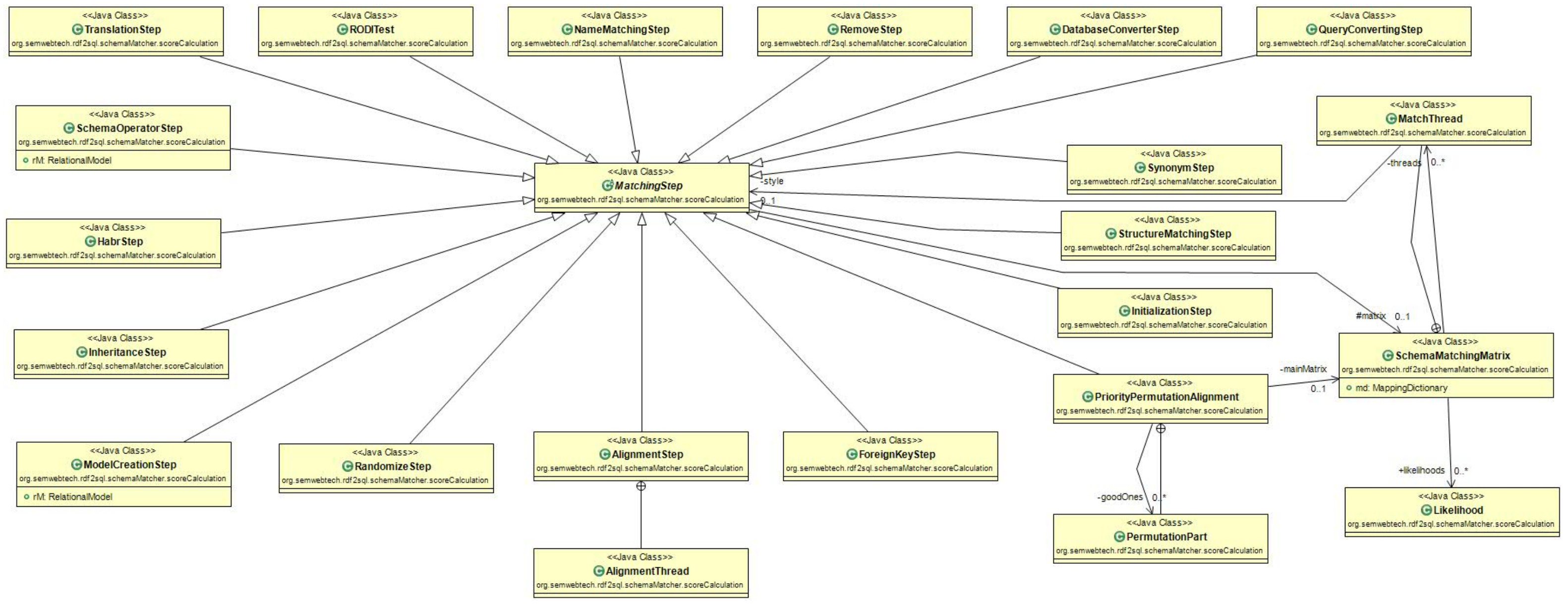
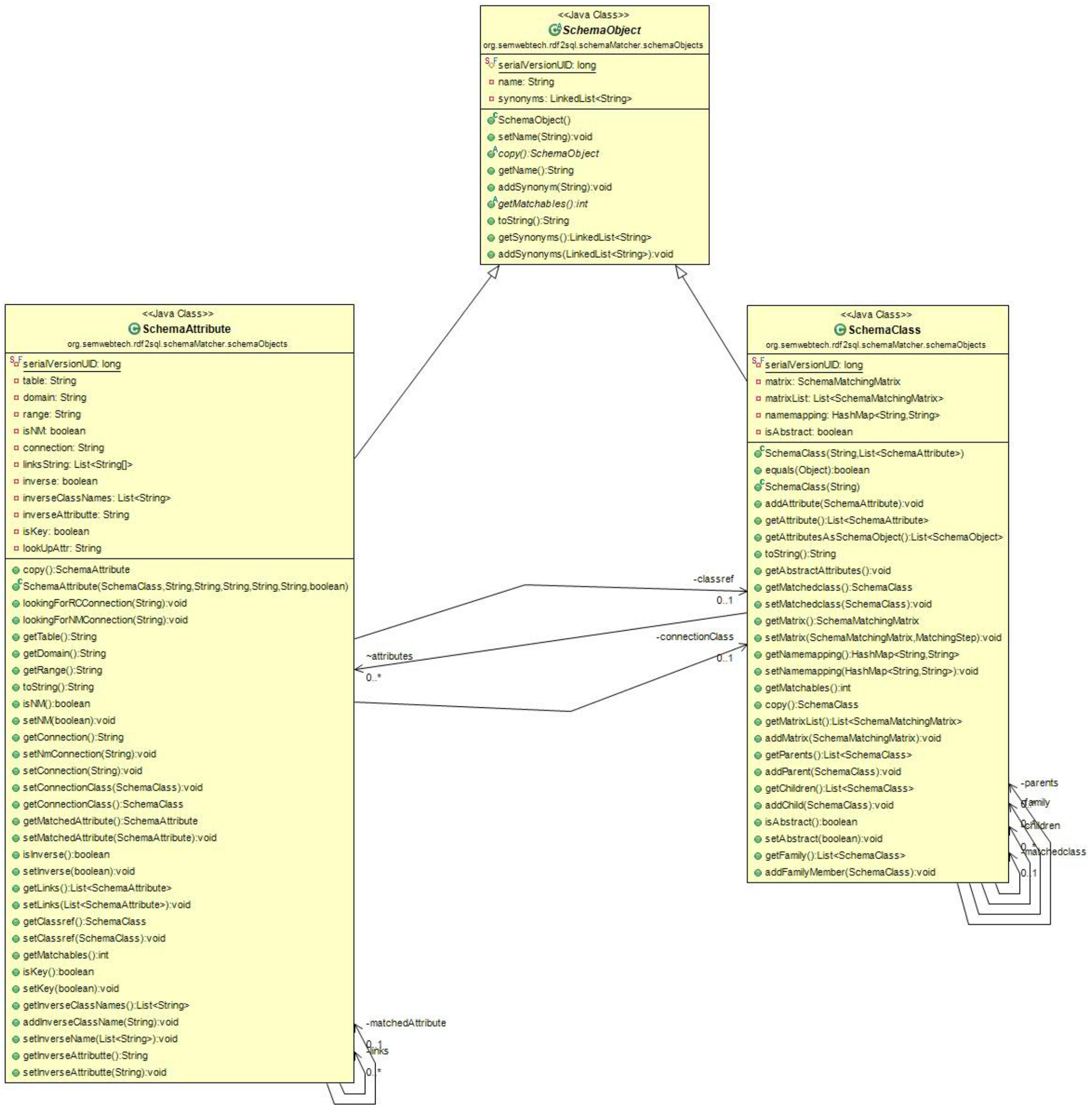


Figure B.3: MatchingStep Class



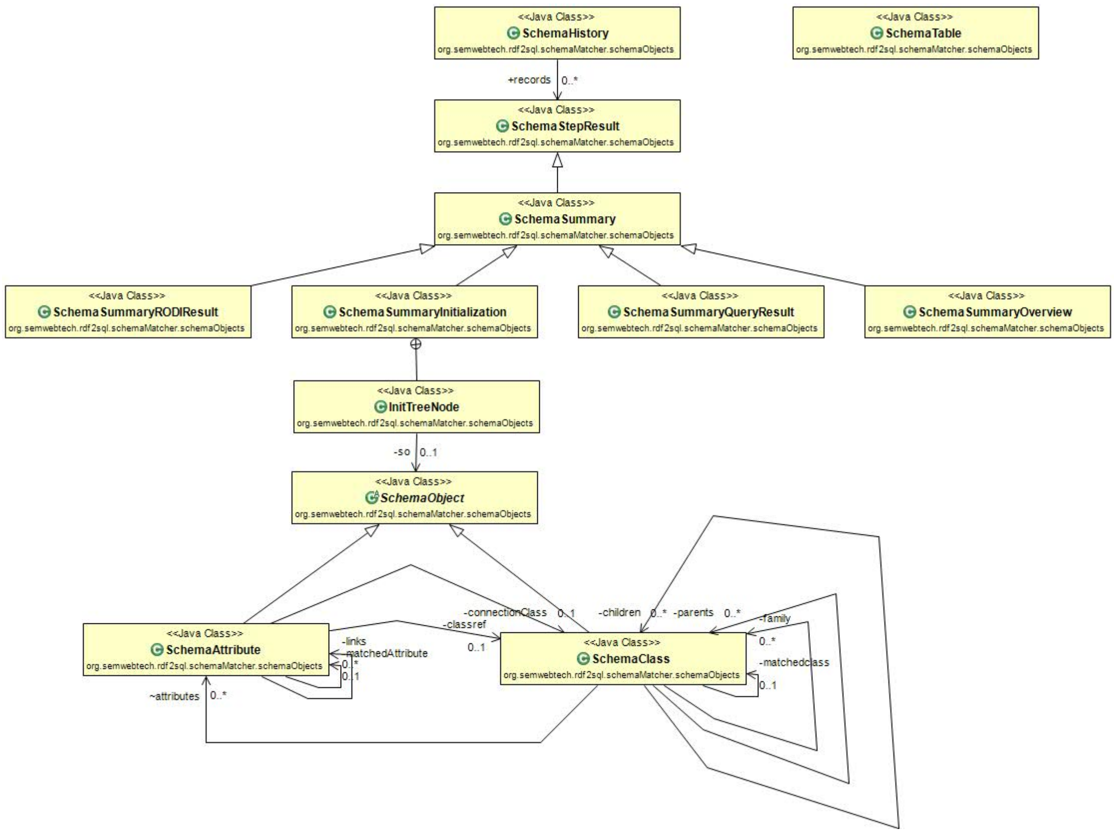
<<Java Class>>
TranslationStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
RODITest
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
NameMatchingStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
RemoveStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
DatabaseConverterStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
QueryConvertingStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
SchemaOperatorStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
rM: RelationalModel
<<Java Class>>
MatchingStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
-style
0..1
<<Java Class>>
MatchThread
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
-threads
0..*
<<Java Class>>
SynonymStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
HabrStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
StructureMatchingStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
InitializationStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
#matrix
0..1
<<Java Class>>
InheritanceStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
SchemaMatchingMatrix
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
md: MappingDictionary
-mainMatrix
0..1
<<Java Class>>
PriorityPermutationAlignment
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
ModelCreationStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
rM: RelationalModel
<<Java Class>>
RandomizeStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
AlignmentStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
ForeignKeyStep
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
+likelihoods
0..*
-goodOnes
0..*
<<Java Class>>
Likelihood
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
PermutationPart
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
<<Java Class>>
AlignmentThread
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation

Figure B.4: UML Diagram of SchemaMatchingMatrix (SMM), its subclasses and other classes of the scoreCalculation package



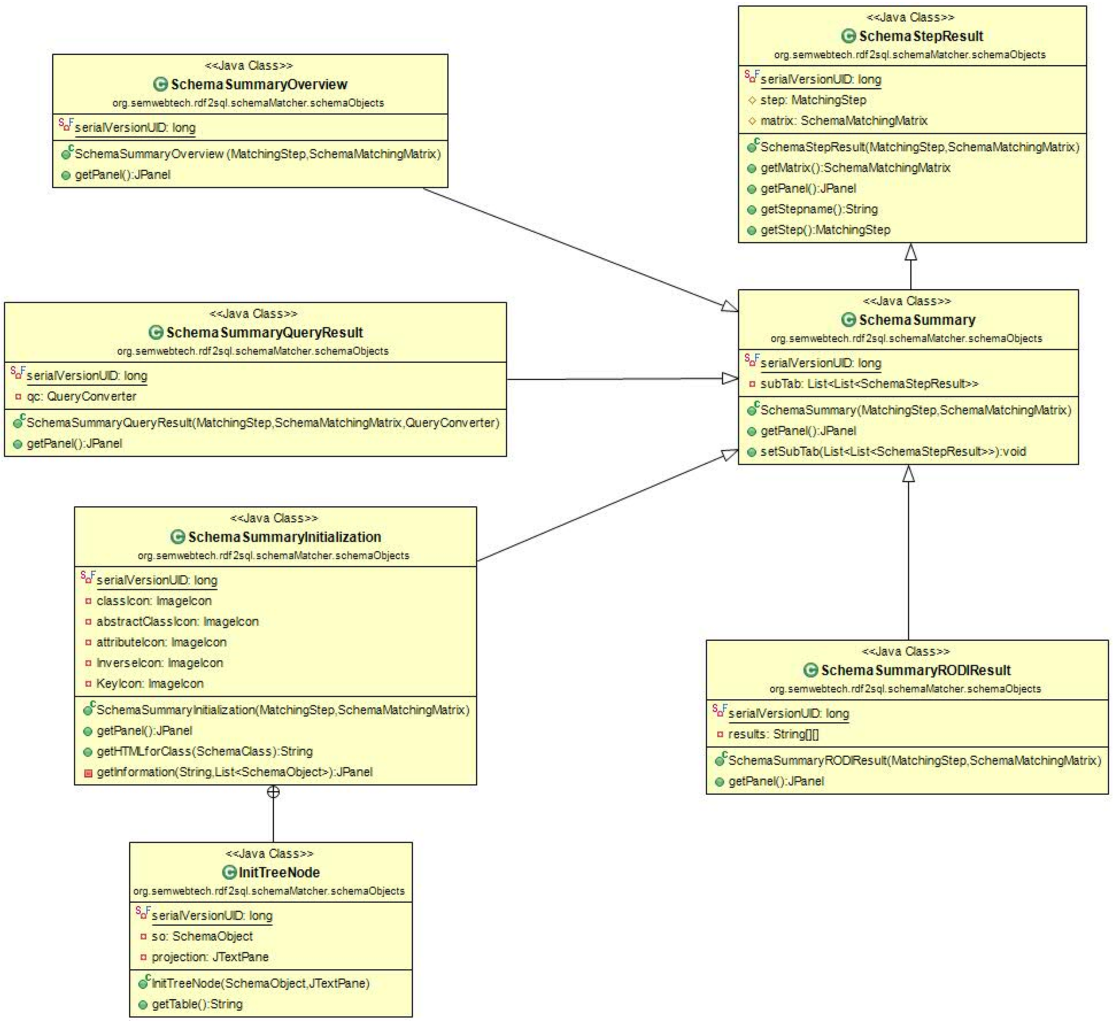
<<Java Class>>
SchemaObject
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
serialVersionUID: long
name: String
synonyms: LinkedList<String>
SchemaObject()
setName(String):void
copy():SchemaObject
getName():String
addSynonym(String):void
getMatchables():int
toString():String
getSynonyms():LinkedList<String>
addSynonyms(LinkedList<String>):void
<<Java Class>>
SchemaAttribute
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
serialVersionUID: long
table: String
domain: String
range: String
isNM: boolean
connection: String
linksString: List<String[]>
inverse: boolean
inverseClassNames: List<String>
inverseAttributte: String
isKey: boolean
lookUpAttr: String
copy():SchemaAttribute
SchemaAttribute(SchemaClass,String,String,String,String,String,boolean)
lookingForRCConnection(String):void
lookingForNMConnection(String):void
getTable():String
getDomain():String
getRange():String
toString():String
isNM():boolean
setNM(boolean):void
getConnection():String
setNmConnection(String):void
setConnection(String):void
setConnectionClass(SchemaClass):void
getConnectionClass():SchemaClass
getMatchedAttribute():SchemaAttribute
setMatchedAttribute(SchemaAttribute):void
isInverse():boolean
setInverse(boolean):void
getLinks():List<SchemaAttribute>
setLinks(List<SchemaAttribute>):void
getClassref():SchemaClass
setClassref(SchemaClass):void
getMatchables():int
isKey():boolean
setKey(boolean):void
getInverseClassNames():List<String>
addInverseClassName(String):void
setInverseName(List<String>):void
getInverseAttribute():String
setInverseAttributte(String):void
<<Java Class>>
SchemaClass
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
serialVersionUID: long
matrix: SchemaMatchingMatrix
matrixList: List<SchemaMatchingMatrix>
namemapping: HashMap<String,String>
isAbstract: boolean
SchemaClass(String,List<SchemaAttribute>)
equals(Object):boolean
SchemaClass(String)
addAttribute(SchemaAttribute):void
getAttribute():List<SchemaAttribute>
getAttributesAsSchemaObject():List<SchemaObject>
toString():String
getAbstractAttributes():void
getMatchedclass():SchemaClass
setMatchedclass(SchemaClass):void
getMatrix():SchemaMatchingMatrix
setMatrix(SchemaMatchingMatrix,MatchingStep):void
getNamemapping():HashMap<String,String>
setNamemapping(HashMap<String,String>):void
getMatchables():int
copy():SchemaClass
getMatrixList():List<SchemaMatchingMatrix>
addMatrix(SchemaMatchingMatrix):void
getParents():List<SchemaClass>
addParent(SchemaClass):void
getChildren():List<SchemaClass>
addChild(SchemaClass):void
isAbstract():boolean
setAbstract(boolean):void
getFamily():List<SchemaClass>
addFamilyMember(SchemaClass):void
-classref
0..1
-connectionClass
0..1
~attributes
0..*
-parents
-family
-children
-matchedclass
0..1
-matchedAttribute
0..1
-links
0..*

Figure B.5: UML Diagram of SchemaObject and Subclasses



<<Java Class>>
SchemaHistory
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
<<Java Class>>
SchemaTable
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
+records 0..*
<<Java Class>>
SchemaStepResult
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
<<Java Class>>
SchemaSummary
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
<<Java Class>>
SchemaSummaryRODIResult
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
<<Java Class>>
SchemaSummaryInitialization
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
<<Java Class>>
SchemaSummaryQueryResult
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
<<Java Class>>
SchemaSummaryOverview
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
<<Java Class>>
InitTreeNode
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
-so 0..1
<<Java Class>>
SchemaObject
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
-connectionClass 0..1
-classref 0..1
-children 0..*
-parents 0..*
-family 0..*
-matchedclass 0..1
<<Java Class>>
SchemaAttribute
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
-links
-matchedAttribute
0..*
0..1
~attributes 0..*
<<Java Class>>
SchemaClass
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects

Figure B.6: UML Diagram of the schemaObject package



<<Java Class>>
SchemaStepResult
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
serialVersionUID: long
step: MatchingStep
matrix: SchemaMatchingMatrix
SchemaStepResult(MatchingStep,SchemaMatchingMatrix)
getMatrix():SchemaMatchingMatrix
getPanel():JPanel
getStepname():String
getStep():MatchingStep
<<Java Class>>
SchemaSummaryOverview
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
serialVersionUID: long
SchemaSummaryOverview(MatchingStep,SchemaMatchingMatrix)
getPanel():JPanel
<<Java Class>>
SchemaSummaryQueryResult
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
serialVersionUID: long
qc: QueryConverter
SchemaSummaryQueryResult(MatchingStep,SchemaMatchingMatrix,QueryConverter)
getPanel():JPanel
<<Java Class>>
SchemaSummary
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
serialVersionUID: long
subTab: List<List<SchemaStepResult>>
SchemaSummary(MatchingStep,SchemaMatchingMatrix)
getPanel():JPanel
setSubTab(List<List<SchemaStepResult>>):void
<<Java Class>>
SchemaSummaryInitialization
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
serialVersionUID: long
classIcon: ImageIcon
abstractClassIcon: ImageIcon
attributeIcon: ImageIcon
InverseIcon: ImageIcon
KeyIcon: ImageIcon
SchemaSummaryInitialization(MatchingStep,SchemaMatchingMatrix)
getPanel():JPanel
getHTMLforClass(SchemaClass):String
getInformation(String,List<SchemaObject>):JPanel
<<Java Class>>
SchemaSummaryRODIResult
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
serialVersionUID: long
results: String[][]
SchemaSummaryRODIResult(MatchingStep,SchemaMatchingMatrix)
getPanel():JPanel
<<Java Class>>
InitTreeNode
org.semwebtech.rdf2sql.schemaMatcher.schemaObjects
serialVersionUID: long
so: SchemaObject
projection: JTextPane
InitTreeNode(SchemaObject,JTextPane)
getTable():String

Figure B.7: UML Diagram of the SchemaStepResults

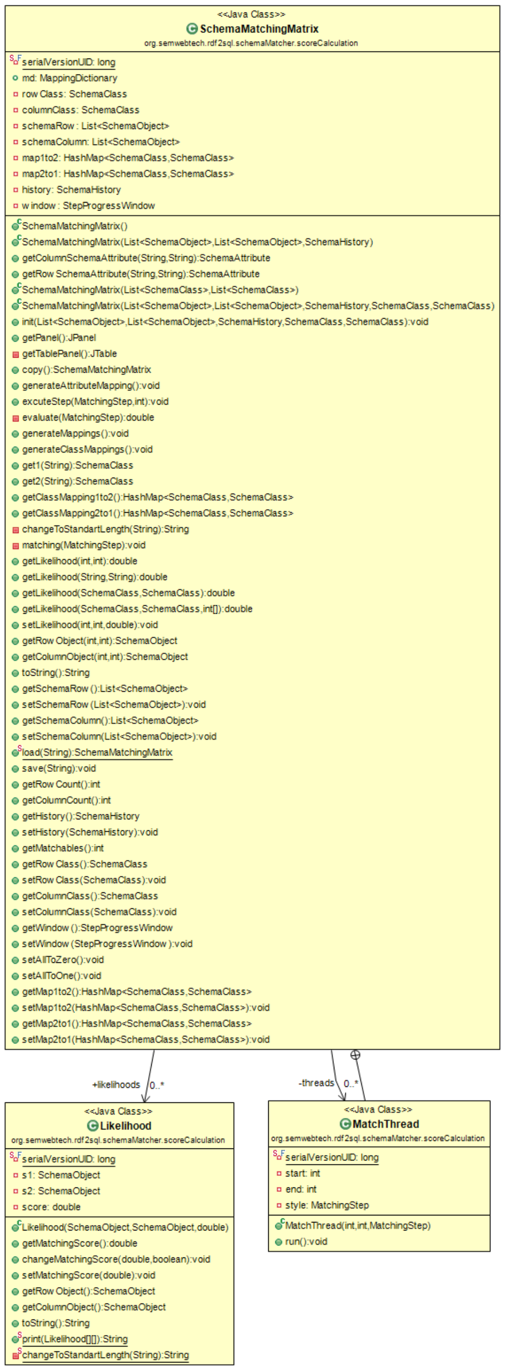

<<Java Class>>
SchemaMatchingMatrix
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
serialVersionUID: long
md: MappingDictionary
row Class: SchemaClass
columnClass: SchemaClass
schemaRow : List<SchemaObject>
schemaColumn: List<SchemaObject>
map1to2: HashMap<SchemaClass,SchemaClass>
map2to1: HashMap<SchemaClass,SchemaClass>
history: SchemaHistory
w indow : StepProgressWindow
SchemaMatchingMatrix()
SchemaMatchingMatrix(List<SchemaObject>,List<SchemaObject>,SchemaHistory)
getColumnSchemaAttribute(String,String):SchemaAttribute
getRow SchemaAttribute(String,String):SchemaAttribute
SchemaMatchingMatrix(List<SchemaClass>,List<SchemaClass>)
SchemaMatchingMatrix(List<SchemaObject>,List<SchemaObject>,SchemaHistory,SchemaClass,SchemaClass)
init(List<SchemaObject>,List<SchemaObject>,SchemaHistory,SchemaClass,SchemaClass):void
getPanel():JPanel
getTablePanel():JTable
copy():SchemaMatchingMatrix
generateAttributeMapping():void
excuteStep(MatchingStep,int):void
evaluate(MatchingStep):double
generateMappings():void
generateClassMappings():void
get1(String):SchemaClass
get2(String):SchemaClass
getClassMapping1to2():HashMap<SchemaClass,SchemaClass>
getClassMapping2to1():HashMap<SchemaClass,SchemaClass>
changeToStandartLength(String):String
matching(MatchingStep):void
getLikelihood(int,int):double
getLikelihood(String,String):double
getLikelihood(SchemaClass,SchemaClass):double
getLikelihood(SchemaClass,SchemaClass,int[]):double
setLikelihood(int,int,double):void
getRow Object(int,int):SchemaObject
getColumnObject(int,int):SchemaObject
toString():String
getSchemaRow ():List<SchemaObject>
setSchemaRow (List<SchemaObject>):void
getSchemaColumn():List<SchemaObject>
setSchemaColumn(List<SchemaObject>):void
load(String):SchemaMatchingMatrix
save(String):void
getRow Count():int
getColumnCount():int
getHistory():SchemaHistory
setHistory(SchemaHistory):void
getMatchables():int
getRow Class():SchemaClass
setRow Class(SchemaClass):void
getColumnClass():SchemaClass
setColumnClass(SchemaClass):void
getWindow ():StepProgressWindow
setWindow (StepProgressWindow):void
setAllToZero():void
setAllToOne():void
getMap1to2():HashMap<SchemaClass,SchemaClass>
setMap1to2(HashMap<SchemaClass,SchemaClass>):void
getMap2to1():HashMap<SchemaClass,SchemaClass>
setMap2to1(HashMap<SchemaClass,SchemaClass>):void
+likelihoods
0..*
-threads
0..*
<<Java Class>>
Likelihood
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
serialVersionUID: long
s1: SchemaObject
s2: SchemaObject
score: double
Likelihood(SchemaObject,SchemaObject,double)
getMatchingScore():double
changeMatchingScore(double,boolean):void
setMatchingScore(double):void
getRow Object():SchemaObject
getColumnObject():SchemaObject
toString():String
print(Likelihood[][]):String
changeToStandartLength(String):String
<<Java Class>>
MatchThread
org.semwebtech.rdf2sql.schemaMatcher.scoreCalculation
serialVersionUID: long
start: int
end: int
style: MatchingStep
MatchThread(int,int,MatchingStep)
run():void

Figure B.8: UML Diagram - SchemaMatchingMatrix