Department of Informatics, University of Zürich

Vertiefungsarbeit

Interlinking the Swiss Feed Database with Linked Open Data Cloud

Manuel Schlegel

Matrikelnummer: 12-924-031

Email: manuel.schlegel@uzh.ch

February 02, 2016

supervised by Prof. Dr. Michael Böhlen and Georgios Garmpis

University of Zurich[UZH]

Department of Informatics

1 Introduction

For my in-depth work at the University of Zurich, I have worked together with my supervisor Georgios Garmpis at the Database Technology Group in order to interlink the Swiss Feed Database with the Linked Open Data Cloud. The goals of the project can be discretized in the following tasks:

1. Construct an ontology in OWL that describes the schema of the Swiss Feed Database.
2. Transform the dataset of the Swiss Feed Database to RDF using standard tools.
3. Demonstrate the new RDF dataset and the interlinking with an auxiliary dataset by posing various SPARQL queries using RDF stores.

The tasks have been successfully completed and will be summarized in this report, together with the knowledge about the various fields which has been acquired in order to accomplish them. With their completion, the project is now ready for the final goal, which is the interlinking with the Linked Open Data Cloud.

1.1 The Semantic Web

The WWW (World Wide Web) is a vast and endless resource of information. Local sport events, weather information, plane times, television guides, news articles and countless more kinds of data are presented by numerous sites on the web. The data is often displayed in HTML, which makes it easy for a human to understand and comprehend. However, since in HTML there is a lack of semantics, computers have difficulties understanding HTML documents. They lack the semantic knowledge and common sense to establish connections between pieces of information [1].
As a huge engineering solution, Tim Berners-Lee, inventor of the WWW, URIs, HTTP and HTML thought up the Semantic Web, which aims to support a *Web of data* in addition to the classic *Web of documents* described before. The goal of the Web of data is to enable computers to traverse the data published on the web and gather semantic knowledge. The result would be a gigantic, freely accessible knowledge base forming the foundation of a new generation of applications and services. Semantic Web technologies allow people to create data stores on the Web, build vocabularies and write rules for utilizing data [2] [5].
The Semantic Web is generally built on syntaxes which use URIs (Uniform Resource Identifier) to represent data, usually in triple based structures. Many triples of URI data can be stored in databases (usually known as RDF stores), or interchanged on the WWW using a set of special syntaxes developed specifically for the task. These syntaxes are called RDF (Resource Description Framework) syntaxes [1].

1.2 Linked Data

In order to make the Web of data a reality, there is the need to make relationships among data available. The collection of interrelated datasets on the Web is called Linked Data (as opposed

to a sheer collection of datasets). The idea is, that with Linked Data, when you have some of it, you can find other, related, data [3] [4].

Tim Berners-Lee outlined four rules of Linked Data [3]:

1. Use URIs as names for things.
2. Use dereferenceable HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Include links to other URIs, so that they can discover more things.

These four rules can be applied partially. They provide a basic recipe for publishing and connecting data using the infrastructure of the web, such that all published data becomes part of a single global data space, where datasets are connected with each other: the Linked Data Cloud. An example of a large Linked Dataset in the Linked Data Cloud is DBpedia [6] which, primarily, makes the content of Wikipedia infoboxes available in RDF. The important part is that DBpedia not only includes Wikipedia data, but that it also offers links to other datasets on the Web, *e.g. to GeoNames* [7]. Multiple ways of linking data exist, the most common one being that two resources are identical. With provision of these additional links, applications may exploit the extra, and possibly more precise, knowledge from other datasets. Using this knowledge, an application may be able to greatly enhance its usefulness and provide a much better user experience [3] [4].
Nonetheless, unlike the Web of documents, where relationships between documents are expressed using hyperlinks, in the Web of data a framework called RDF is used [3].

1.3 RDF

RDF, developed by the W3C (World Wide Web Consortium), is a framework for expressing information about resources. Resources are identified by URIs and can represent anything: documents, people, physical objects, and abstract concepts. It is a data model based upon the idea of making statements about resources. A statement is an expression of the form subject-predicate-object, also known as a *triple*. The subject and the object represent two resources being related, where the object describes the subject with either a resource or a literal and the predicate describes the nature of their relationship. For example, the statement [8]:

<the Mona Lisa> <was created by> <Leonardo da Vinci>.

can be represented by a RDF statement having:

- the subject *http://www.wikidata.org/entity/Q12418* (Mona Lisa)
- the predicate *http://purl.org/dc/terms/creator* (creator)

- the object *http://dbpedia.org/resource/Leonardo_da_Vinci* (Leonardo da Vinci)

Note how URIs are used to identify the subject, predicate and object. This allows us, for example, to use the predicate - since it is specified as a URI - as a subject or object in another RDF statement. Additionally, RDF lets us use literals for the object (but not the subject or predicate), which is shown in a second RDF statement below [8]:

<Leonardo da Vinci> <has the birth name> <"Leonardo di ser Piero da Vinci">.

which can be represented by a RDF statement having:

- the subject *http://dbpedia.org/resource/Leonardo_da_Vinci* (Leonardo da Vinci)
- the predicate *http://example.org/terms/birthName* (has birth name)
- the object *"Leonardo di ser Piero da Vinci"* ("Leonardo di ser Piero da Vinci", as a literal)

The focus of this section will be on the first RDF statement. RDF statements intrinsically represent a labeled, directed multi-graph. Figure 0.1 shows the example as an annotated graph.

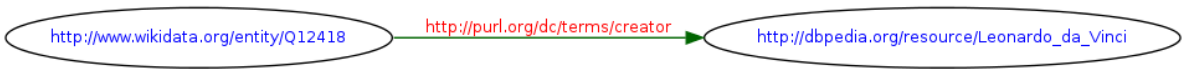

Figure 0.1: RDF example as an annotated graph [8]

However, there exist a number of different serialization formats for RDF graphs and so the particular way in which a set of triples is encoded varies from format to format. Still, different ways of serializing the same graph lead to exactly the same triples and statements, and are thus semantically equivalent. In the following two serializing formats of RDF, N-Triples and Turtle, will be introduced [8].

N-Triples

N-Triples provide a simple line-based, plain-text encoding for RDF graphs. Each line represents a triple and URIs are enclosed in angle brackets. At the end of the line, a period signals the end of the triple. The example from Figure 0.1 can be encoded in N-Triples and the result is shown in Listing 1 [8].

Listing 1: Example Graph in N-Triples format

```
<http://www.wikidata.org/entity/Q12418>
    <http://purl.org/dc/terms/creator>
    <http://dbpedia.org/resource/Leonardo_da_Vinci> .
```

Due to space limitations, the result is separated from one to three lines. The advantage of N-Triples is that it easily parsable. On the other hand, it is not easy for humans to read and understand. Furthermore N-Triples files take up more space than Turtle files. [8].

Turtle
Turtle can be considered an extension of N-Triples. In addition to the basic N-Triples syntax, it introduces a number of syntactic shortcuts, which include: support for namespace prefixes, lists and shorthands for datatyped literals. Turtle provides a good trade-off between ease of writing, ease of parsing and readability. It is designed to be a compact, human-friendly format. The same example from Figure 0.1 can be represented in Turtle as shown in Listing 2 [8].

Listing 2: Example Graph in Turtle format

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix wiki: <http://www.wikidata.org/entity/>.
@prefix dbpedia: <http://dbpedia.org/resource/> .

wiki:Q12418 dcterms:creator dbpedia:Leonardo_da_Vinci .
```

The example shows that through the use of namespace prefixes, one of the benefits of using Turtle, the example got easier for humans to understand.

2 Swiss Feed Ontology

The Swiss Feed Database [9] is a public service for companies, private farmers and research institutions and it provides feed data which characterizes the quality of grown, imported and synthetic feed types from feed samples collected in Switzerland. It contains information about the composition and nutritional value of the feeds and offers free and password-protected (which can be accessed based on a paid subscription) functionality to search the database [10] [11].
In order to transform the data from the Swiss Feed Database to RDF in a desirable way, there is a need for an additional element. RDF is, in contrary to relational databases, by default schemaless. This means that it is more flexible than a schema structure but also more ambiguous: While there are no rules for the data, so one can store any data they like (without prior knowledge of how the data looks like), the relationship information between the data elements is lost. Nonetheless, by using Ontologies, RDF can provide additional schema information.
Ontologies are essentially vocabularies on the Semantic Web. They are used to define the concepts and relationships used to describe and represent an area of concern. They help classify the terms that can be used in a particular application, characterize relationships and define possible constraints. A large palette of techniques exists to describe ontologies. We will mainly focus on RDFS (RDF Schema) and OWL (Web Ontology Language), as they are the ones most relevant for the project. They will be described and compared in the next section [12].

2.1 RDFS and OWL

RDFS, published by the W3C in 1998, is the basic RDF vocabulary that provides mechanisms for describing groups of related resources and the relationships between these resources. OWL, defined by W3C, is similar, but has a richer vocabulary, which lets you express more with it. In the following the main concepts of both techniques will be compared [13] [14].

The main concepts of RDFS include, but are not limited to: classes, instances, properties, domain and range of a property. In RDF, resources may be divided into groups called classes, whose members are known as instances of a class. In order to express that a resource R is an instance of a class C, we can write `R rdf:type C`. Note that this is still basic RDF, which does not use RDFS. However, using RDFS, we can state that a class C1 is a subclass of a class C2 using `C1 rdfs:subClassOf C2`, which is used to express that all instances of C1 are also instances of class C2. As in RDF, properties are relations between resources or between a resource and a literal. With RDFS we can specify that any resource that has a given property is instance of one or more classes by writing `P rdfs:domain C`, which expresses that the resources denoted as subjects of triples whose predicate is P are instances of the class C. Furthermore, a property can have a range, consisting of either a class or a literal datatype. If a range consists of a class, then the property is called *object property* and states that the object of a property is an instance of this class. Otherwise, if the range consists of a data type, it is known as a *data property*. For example, if C is a class, we can write `P rdfs:range C` to state that the resources denoted by the objects of triples whose predicate is P are instances of the class C. These are the main capabilities offered by RDFS [13].
Similarely, OWL has classes, properties and individuals. Nearly everything that can be described in RDFS can also be expressed in OWL, by reusing RDFS terms. This is the main benefit of OWL: it enriches RDFS and lets you express more aspects. The central features added by OWL are [14] [15]:

- Cardinality restrictions: OWL allows you to specify the exact, minimum and maximum cardinality, which permit the specification of the number of elements in a relation. *For example, we can specify that a person has exactly one biological father, using an exact cardinality.*

- Property characteristics: It is possible to specify property characteristics, which serve as a powerful mechanism for enhancing reasoning about a property. The most important one is the `transitive` characteristic. If a property P, is specified as transitive then for any x, y, and z: `P(x,y) and P(y,z) implies P(x,z).`

- Equivalence rules: OWL further allows you to specify that two different classes/properties (having different URIs) actually represent the same class/property. This is also possible for instances, where one can specify using `owl:sameAs` that two instances represent the same individual. This is most often used for the linking of data in Linked Data, mentioned in section 1.2.

2.2 Ontology created using Protégé

For the project, an ontology for the Swiss Feed Database has been created [9] using Protégé [16]. Protégé is a free, open-source ontology source editor which supports the OWL 2 Web Ontology language. It is being actively developed at the Stanford Center for Biomedical Informatics Research and is made available under the Mozilla Public License 1.1. Protégé is the most often used ontology editor and allows to easily build both simple and complex ontology-based applications [16].

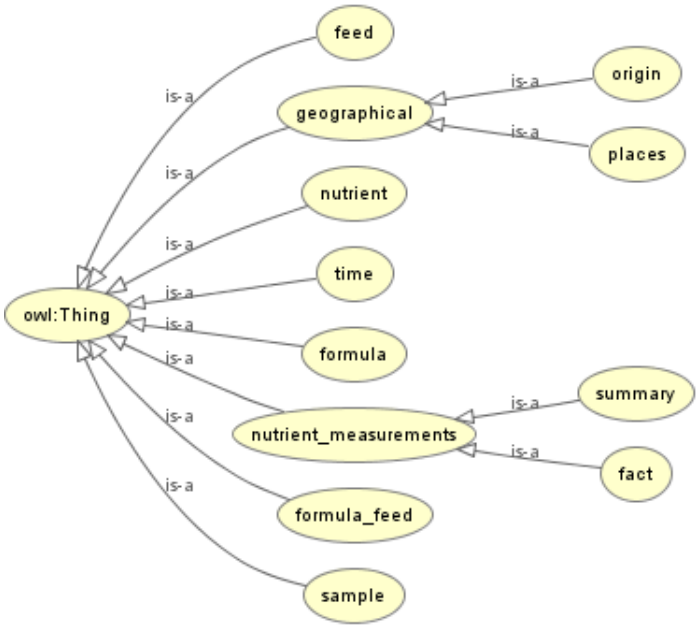

Figure 0.2: Swiss Feed Database Ontology Class Hierarchy

The final class hierarchy of the ontology, shown in Figure 0.2, consists of the depicted classes, where each original table of the Swiss Feed Database is mapped to a class. Where `origin`, `places` and `summary`, `fact` are subclasses of the superclass `geographical` and `nutrient_measurements`, respectively. The superclasses were created for the ontology in order to better structure the data.
Each class has data properties assigned to it which correspond to the columns of the matching tables in the original database. Furthermore, in order to express relationships between tables of the original database (expressed used foreign keys), object properties based on these foreign keys have been created.

3 Transforming relational data into RDF

In order to transform relational data into RDF, two steps are needed. First, a Mapping file containing rules on how to map relational data into RDF has to be created. Second, the Mapping file and its rules have to be applied to the database. This generates the data in RDF format. Both tasks were done using D2RQ [17], a system for accessing relational databases as virtual, read-only RDF graphs. It is Open Source software and published under the Apache license [17].
For the creation of the Mapping file, D2RQ provides a tool called `generate-mapping` which is able to connect to a database and automatically create mapping rules based on the structure of the database. However, since these automatically generated rules do not match with the ontology created for the project, the Mapping file was edited according to the ontology. This consisted mainly of renaming names, combining multiple data fields, such as lan-

guages, into one, and more small changes. The main rule types of D2RQ are `ClassMap` and `PropertyBridge`. `ClassMap` is used to create RDF resources and `PropertyBridge` allows us to add properties to resources.
Once the Mapping file was completed, the D2RQ tool `dump-rdf` was run, which uses a Mapping file as input in order to connect with the database, run its rules against the data and generate a data dump in a desired format. For the project the Turtle format was chosen, since it is, as mentioned before, easily readable by humans.

3.1 Example of Mapping rules and their output

In order to better explain the concepts introduced in this section, let us take a look at the following example. We are given table `d_feed`, which contains the columns `id` and `name`, and table `d_fact`, which contains the columns `id`, `name` and `feed_fkey`, where `feed_fkey` is the foreign key of table `feed`. We want to create mapping rules in order to transform the `d_fact` table into RDF.
We will create three rules and save them to a file and start D2RQ's `dump-rdf` using the mapping file as input after each new rule.

d_feed	
id	**name**
1	Herbage
2	Wheat

Table 0.1: sample of the `d_feed` table

d_fact		
id	**name**	**feed_fkey**
1	Fact A	1
2	Fact B	1

Table 0.2: sample of the `d_fact` table

In order to properly map the table `d_fact` to the class `fact`, we create a mapping rule as in Listing 3. This mapping rule uses D2RQ's `ClassMap` in order to map basic parts of the `d_fact` table to a class.

Listing 3: Mapping Rule 1

```
# Creates a new ClassMap rule named d_fact.
# This rule creates a class definition
# and creates an instance for each row
# of the table in the database.
map:d_fact a d2rq:ClassMap;
    # Reference to a d2rq:Database where the instance data is stored.
    d2rq:dataStorage map:database;
    # Specifies a URI pattern that will be used to
    # identify instances of this class.
    # The URI will be produced using
    # the constant "http://example.org/swd.ttl#d_fact/"
    # and @@d_fact.id@@, where the parts between @@'s
    # mark data columns in Table.Column notation.
    d2rq:uriPattern "http://example.org/swd.ttl#d_fact/@@d_fact.id@@";
    # Specifies the URI of the class definition.
    d2rq:class vocab:fact;
    # Specifies the label of this class using rdfs:label.
```

```
    # rdfs:label has no specific semantics and
    # is used to give a short description of resources.
    d2rq:classDefinitionLabel "d_fact" .
```

The mapping rule specified in Listing 3 creates the result in Listing 4.

Listing 4: Dump Result 1

```
vocab:fact
      rdf:type rdfs:Class ;
      rdfs:label "d_fact" .

<http://example.org/swd.ttl#d_fact/1>
      rdf:type vocab:fact ;
      rdfs:label "d_fact #1" .

<http://example.org/swd.ttl#d_fact/2>
      rdf:type vocab:fact ;
      rdfs:label "d_fact #2" .
```

Next, we map the column `name` of the `d_fact` table to the property `d_fact_name`. We do this by using D2RQ's `PropertyBridge`. The code for this is shown in Listing 5.

Listing 5: Mapping Rule 2

```
# Creates a new PropertyBridge rule named d_feed_name.
# This rule creates a property definition
# and creates a data property for each row
# of the table "d_fact" in the database
# where the column "name" is set.
map:d_fact_name a d2rq:PropertyBridge;
    # Specifies to which class the subject of this property belongs.
    d2rq:belongsToClassMap map:fact;
    # Specifies the URI of the property definition.
    d2rq:property vocab:d_fact_name;
    # Specifies the label of this property.
    d2rq:propertyDefinitionLabel "d_fact name";
    # Target table "d_fact",
    # column "name" in the original database.
    d2rq:column "d_fact.name" .
```

This second mapping rule, as specified in Listing 5, creates the additional result shown in Listing 6.

Listing 6: Dump Result 2

```
vocab:d_fact_name
      rdf:type rdf:Property ;
      rdfs:label "d_fact name" .

<http://example.org/swd.ttl#d_fact/1>
      vocab:d_fact_name "Fact A" .
```

```
<http://example.org/swd.ttl#d_fact/2>
        vocab:d_fact_name "Fact A" .
```

Last but not least, we want to map the foreign key `feed_fkey` of the `d_fact` table using D2RQ. We can do this again by using `PropertyBridge`. The mapping rule for this can be found in Listing 7. Note the difference between Listing 5 and Listing 7: In Listing 7, `d2rq:join` is used to specify that it creates an object property, while in Listing 5, `d2rq:column` is used to define that it creates a data property.

Listing 7: Mapping Rule 3

```
# Creates a new PropertyBridge rule named d_fact_feed_fkey__ref
# This rule creates a property definition
# and creates an object property for each row
# of the table "d_fact" in the database
# where the column "feed_fkey" is set.
map:d_fact_feed_fkey__ref a d2rq:PropertyBridge;
    # Specifies to which class the subject of this property belongs.
    d2rq:belongsToClassMap map:fact;
    # Specifies the URI of the property definition.
    d2rq:property vocab:feed_fkey;
    # Specifies to which class the property refers.
    d2rq:refersToClassMap map:feed;
    # Specifies how the two classes are linked
    # in the notation
    # Table1.Column => Table2.Column.
    d2rq:join "d_fact.feed_fkey => d_feed.id" .
```

For the final result, we save all the three mapping rules to a file and start D2RQ's `dump-rdf` using the mapping file as input. Assuming we have also properly mapped the table `d_feed`, we get the result shown in Listing 8. We can see that Listing 7 successfully generated the object properties.

Listing 8: Final Dump Result

```
vocab:fact
      rdf:type rdfs:Class ;
      rdfs:label "d_fact" .

vocab:d_fact_name
      rdf:type rdf:Property ;
      rdfs:label "d_fact name" .

vocab:feed_fkey
      rdf:type rdf:Property .

<http://example.org/swd.ttl#d_fact/1>
      rdf:type vocab:fact ;
      rdfs:label "d_fact #1" ;
      vocab:d_fact_name "Fact A" ;
```

```
        vocab:feed_fkey <http://example.org/swd.ttl#d_feed/1> .

<http://example.org/swd.ttl#d_fact/2>
        rdf:type vocab:fact ;
        rdfs:label "d_fact #2" ;
        vocab:d_fact_name "Fact B" ;
        vocab:feed_fkey <http://example.org/swd.ttl#d_feed/1> .
```

4 Querying RDF data

In order to be able to query RDF data, a language called SPARQL can be used. SPARQL, developed by W3C, is a recursive acronym for SPARQL Protocol and RDF Query Language. It is a semantic query language for RDF data [18].
The main components of a SPARQL query are the `SELECT` clause and the `WHERE` clause. The `WHERE` clause consists of a set of triple patterns, which are similar to RDF. Such a triple pattern may look like this: `?person vocab:name "Jasmin" .`. This triple pattern looks for all resources that have a `vocab:name` predicate and the object (literal) `"Jasmin"`. Note how any URIs inside the triple pattern can be replaced by a variable, *e.g.* `?person`. A set of triple patterns is called a graph pattern, which are used to find subgraphs that match the pattern. The `SELECT` clause states which variables the user wants to retrieve. In addition there exists the `PREFIX` clause, which allows the use of prefixes in queries [18].
For the last part in the project, the supervisor uploaded the generated RDF dataset in RDF format to a server running a SPARQL endpoint, allowing the use of SPARQL queries to access and manipulate the data. The SPARQL endpoint provided by the supervisor is called Strabon. Strabon is a RDF store that can be used to store RDF data and query it using SPARQL [19].

4.1 Example Queries

In order to better present the basic capabilities of SPARQL, several example queries which fulfill different goals have been implemented. Each of these example queries introduces one or more features of SPARQL. The queries have been tested against the generated RDF dataset on the SPARQL endpoint. Following are the queries that have been implemented, along with their goal, result and explanation of the features used in the query [19].

1. Query
Goal: Find all formulas that are connected with a feed with name (in German) "Heu 1. Schnitt"
Query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX fdb: <http://www.ifi.uzh.ch/dbtg/ontologies/swiss-feed-database.ttl#>

SELECT (?formula AS ?ConnectedFormulas)
WHERE {
    ?feed rdf:type fdb:feed .
    ?feed fdb:d_feed_name "Heu 1. Schnitt"@de .
    ?formulafeed fdb:formula_feed_feed ?feed .
```

```
    ?formulafeed fdb:formula_feed_formula ?formula .
}
```

Result:

```
ConnectedFormulas
http://www.ifi.uzh.ch/dbtg/ontologies/swiss-feed-database.ttl#t_formula/197
http://www.ifi.uzh.ch/dbtg/ontologies/swiss-feed-database.ttl#t_formula/403
http://www.ifi.uzh.ch/dbtg/ontologies/swiss-feed-database.ttl#t_formula/42
http://www.ifi.uzh.ch/dbtg/ontologies/swiss-feed-database.ttl#t_formula/218
http://www.ifi.uzh.ch/dbtg/ontologies/swiss-feed-database.ttl#t_formula/435
http://www.ifi.uzh.ch/dbtg/ontologies/swiss-feed-database.ttl#t_formula/287
```

Explanation: This is a basic SPARQL query containing only triple patterns. In the `SELECT` clause, we specify that we want to grab the results for the `?formula` variable and name the output `?ConnectedFormulas`. In the `WHERE` clause we specify that `?feed` is an instance of the class `fdb:feed` and that the `?feed` instance has a `fdb:d_feed_name` equal to "Heu 1. Schnitt" in the German language, as specified by the *language tag* `@de`. Furthermore we make a connection to the classes `fdb:formula_feed_feed` and `fdb:formula_feed_formula` to get the respective formulas.

2. Query

Goal: Find raw and clean values about the nutrient with English abbreviation "ADL" contained in feed with English name "Barley, bulk density unknown (lat. Hordeum vulgare)"
Query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX fdb: <http://www.ifi.uzh.ch/dbtg/ontologies/swiss-feed-database.ttl#>

SELECT (?cleanvalue AS ?CleanValue) (?rawvalue AS ?RawValue)
WHERE {
    ?feed rdf:type fdb:feed .
    ?feed fdb:d_feed_name ?feedName .
    ?summary fdb:summary_data_feed  ?feed .
    ?summary fdb:summary_data_nutrient ?nutrient .
    ?nutrient fdb:d_nutrient_abbreviation "ADL"@en .
    OPTIONAL { ?summary fdb:summary_data_clean_value ?cleanvalue } .
    OPTIONAL { ?summary fdb:summary_data_raw_value ?rawvalue } .
    FILTER ( ?feedName="Barley, bulk density unknown (lat. Hordeum vulgare)"@en ) .
}
```

Result:

```
CleanValue  RawValue
"6.49902"^^<http://www.w3.org/2001/XMLSchema#double>
    "6.3730316442"^^<http://www.w3.org/2001/XMLSchema#double>
    "4.42787"^^<http://www.w3.org/2001/XMLSchema#double>
```

Explanation: This query uses `OPTIONAL` and `FILTER`. `OPTIONAL` allows parts of a graph pattern to be optionally matched. In addition, we can express various complex conditions using `FILTER`. `FILTER` restricts the solutions of a graph according to a given expression. Specifically, `FILTER` removes any solutions that, when substituted into the expression, either result in a boolean value of false or produce an error. `FILTER` can be used with logical, comparison, math, regex and other conditions [18].

3. Query
Goal: Find the number of the distinct feeds exist in the database.
Query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX fdb: <http://www.ifi.uzh.ch/dbtg/ontologies/swiss-feed-database.ttl#>

SELECT (COUNT(?feeds) AS ?NumOfDistinctFeeds)
WHERE {
    ?feeds rdf:type fdb:feed .
}
```

Result:

```
NumOfDistinctFeeds
"92"^^<http://www.w3.org/2001/XMLSchema#integer>
```

Explanation: SPARQL offers aggregate functions. For example, `COUNT`, `SUM`, `AVG`, `MIN` and `MAX` can be used inside a query to evaluate expressions. In Query 3, `COUNT` has been used in order to count all results and return the result as `?NumOfDistinctFeeds` [18].

4. Query
Goal: Find the number of measurements that exist in `fact_table` per feed for each year.
Query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX fdb: <http://www.ifi.uzh.ch/dbtg/ontologies/swiss-feed-database.ttl#>

SELECT (?feed AS ?Feed) (?year AS ?Year) (COUNT(?fact) AS ?NumberOfMeasurements)
WHERE {
    ?feed rdf:type fdb:feed .
    ?fact fdb:fact_table_feed ?feed .
    ?fact fdb:fact_table_time ?time .
    ?time fdb:d_time_t_year ?year .
}
GROUP BY ?feed ?year
```

Result:

```
Feed    Year        NumberOfMeasurements
http://www.ifi.uzh.ch/dbtg/ontologies/swiss-feed-database.ttl#d_feed/131
    "2004"^^<http://www.w3.org/2001/XMLSchema#integer>
    "1"^^<http://www.w3.org/2001/XMLSchema#integer>
http://www.ifi.uzh.ch/dbtg/ontologies/swiss-feed-database.ttl#d_feed/1
    "2009"^^<http://www.w3.org/2001/XMLSchema#integer>
    "3"^^<http://www.w3.org/2001/XMLSchema#integer>
```

Explanation: Using `GROUP BY` one can calculate aggregate values for a solution. The solution is first broken down into one or more groups and the aggregate value is calculated for each group. Furthermore `HAVING` operates over grouped solution sets, just as `FILTER` operates over ungrouped ones [18].

5 Conclusion

While technological advancements continue to reduce the cost of generating and storing data, the Semantic Web (or Web 3.0, Linked Data Web, Web of Data...), thought up by Tim Berners-Lee, keeps on advancing. More and more content is created on the Semantic Web and standards for the Semantic Web are actively being worked on.
For the project, I got to know and applied several well-established standards of the Semantic Web: URIs, RDF, RDFS, OWL and SPARQL. In addition I have explored several tools that help working with these standards, including Protégé, a free, open-source ontology source editor, D2RQ, an open source system for accessing relational databases as virtual, read-only RDF graphs and Strabon, a semantic RDF store that allows the use of SPARQL queries to access and manipulate data stored in a RDF format.
The Swiss Feed Database is a great resource for companies, private farmers and research institutions that work with or use feed data. Interlinking it with the Open Data Cloud at subsequent point in time may improve an application's usefulness by allowing it to access not only the data from the Swiss Feed Database, but also the data from other datasets linked with the Swiss Feed Database.
The Semantic Web still needs a lot of work and time to become a reality, however, even bigger could the payoff be, once it is mature. Applications will be able to use the knowledge from multiple sources and accomplish more complex and challenging tasks. In addition, there is no need for a database schema, which means that no decisions about the structure and recording of the data are necessary. In order to push the Semantic Web forward, it is important to upgrade existing web content to be compatible with the Web of data and to create, publish and interlink more content with the Linked Data Cloud [20].

Bibliography

[1] The Semantic Web: An Introduction,
http://infomesh.net/2001/swintro/, January, 2016.

[2] Semantic Web - W3C,
https://www.w3.org/standards/semanticweb//, January, 2016.

[3] Linked Data - Design Issues,
https://www.w3.org/DesignIssues/LinkedData.html, January, 2016.

[4] Data - W3C,
https://www.w3.org/standards/semanticweb/data, January, 2016.

[5] The Web of Data: Creating Machine-Accessible Information,
http://readwrite.com/2009/04/18/web_of_data_machine_accessible_information, January, 2016.

[6] DBpedia,
http://dbpedia.org/, January, 2016.

[7] GeoNames,
http://www.geonames.org/, January, 2016.

[8] RDF 1.1 Primer,
https://www.w3.org/TR/rdf11-primer/, January, 2016.

[9] The Swiss Feed Database,
http://www.feedbase.ch/, January, 2016.

[10] UZH - Database Technology - The Swiss Feed Database,
http://www.ifi.uzh.ch/dbtg/research/sfdb.html, January, 2016.

[11] Agroscope - Swiss Feed Database,
http://www.agroscope.admin.ch/futtermitteldatenbank/index.html?lang=en, January, 2016.

[12] Ontologies - W3C,
https://www.w3.org/standards/semanticweb/ontology, January, 2016.

[13] RDF Schema 1.1,
https://www.w3.org/TR/rdf-schema/, January, 2016.

[14] OWL 2 Web Ontology Language Primer,
https://www.w3.org/TR/2009/REC-owl2-primer-20091027/, January, 2016.

[15] An Introduction to RDFSchema,
http://cgi.di.uoa.gr/~pms509/past_lectures/introduction-to-rdf-schema-revised.pdf, January, 2016.

[16] Protégé,
http://protege.stanford.edu/, January, 2016.

[17] The D2RQ Platform - Accessing Relational Databases as Virtual RDF Graphs,
http://d2rq.org/, January, 2016.

[18] SPARQL Query Language for RDF,
https://www.w3.org/TR/rdf-sparql-query/, January, 2016.

[19] Strabon - strabon.di.uoa.gr,
http://strabon.di.uoa.gr/about, January, 2016.

[20] Semantic Web Advantages,
http://www.brighthub.com/internet/web-development/articles/82954.aspx, January, 2016.