— INESC-ID Lisboa Tech. Rep. 06/2016 —

Valéria Pequeno[1] and Helena Galhardas[2]
[1]Av. Prof. Doutor Aníbal Cavaco Silva 2744-016 Porto Salvo, Portugal
INESC-ID
[2] Av. Prof. Doutor Aníbal Cavaco Silva 2744-016 Porto Salvo, Portugal
INESC-ID and Technical University of Lisbon

This revision: October 20, 2016

A Taxonomy of Data Quality Problems for the Semantic Web

Valéria Magalhães Pequeno[1] and Helena Galhardas[2]

[1] INESC-ID
IST Tagus Park, Av Prof Cavaco Silva 2780-990 Porto Salvo, Portugal
vmp@inesc-id-pt
[2] INESC-ID and Technical University of Lisbon
IST Tagus Park, Av Prof Cavaco Silva 2780-990 Porto Salvo, Portugal
helena.galhardas@ist.utl.pt

The Data Quality (DQ) is an important component for any kind of decision process. This subject has been widely studied for several years, but it has not received sufficient attention from the Semantic Web community. In this paper, a DQ taxonomy of RDF data is presented. The proposed taxonomy organizes the problems at different levels of abstraction. We also show SPARQL query templates, which can be instantiated into concrete data quality test queries, in order to detect DQ problems in the RDF datasets. The proposed taxonomy includes more types of problems than any of the existing taxonomies for the Semantic Web.

Keywords: Data Quality, RDF, Taxonomy, SPARQL

1 Introduction

Nowadays, besides the classic "Web of documents" there is the Semantic Web [Berners-Lee, 1998]. The fundamental difference between the Semantic Web and the World Wide Web (WWW) itself is that the first one shares the data in a way that can be used automatically by computers but can still be easily read by humans, while the WWW aims at sharing the data for human readers only. Succinctly, WWW links documents and applications while Semantic Web links data. Often the data in the Semantic Web is linked and published on the web following the principles of the *Linked Data* outlined by Berners-Lee [Berners-Lee, 2006]. The data in the Semantic Web is commonly organized through an ontology [Fensel, 2001] and is linked and published using the Resource Description Framework (RDF) model[3], thus creating a global space known as *Web of Data* [Bizer et al., 2009].

Currently, a large number of datasets is available on the Web of Data. These datasets come originally from traditional data sources (e.g., relational databases) or they are obtained from web pages. Some datasets are private while others are publicly available on the web (i.e., they are published as Linked Open Data (LOD) [Bizer et al., 2009]). LOD is linked data that is released under an open license, thus, it can be reused for free.

Various factors have contributed to the increase of the datasets on the Web of Data. Some of these factores are:

1. Use of a standard, expressive and simple data model RDF, which enables to integrate data available in different sources (independently if it is public or private);

[3] https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

2. Use of a standard query language (SPARQL) that allow us to use any SPARQL endpoint, no matter what the implementation is or what the schema looks like; and
3. Data are self-descriptive (if a vocabulary is unknown by a user or application, it can look up the URI and retrieve a description of the resource that is identified by the URI).

Despite the increasing amount of data available on the web, it is not ready to be used. Most part of this data has low quality (i.e., data cab be incorrect, incomplete or out-dated). Data Quality (DQ) problems occur, in part, because a DQ auditing process has not been performed before converting existing data sources to the RDF data model. Therefore, the DQ problems existing at the original representation are propagated to the target. DQ problems may also occur because there is no control of who update the data on the web (e.g., updates in Wikipedia). Thus, it is mandatory to examine the data in order to assure that it satisfies certain quality criteria before it is used.

Some methodologies and frameworks for assessing DQ have been proposed by the database community (see [Batini et al., 2009] for a survey). However, the evaluation of DQ on the Web of Data has new challenges. For example, it involves the analysis of independent data sources that are constantly being updated, possibly by people around the world in a decentralized way. Furthermore, it involves the analysis of the consistency of links to external datasets. Works focused on DQ on the Web of Data are relatively recent (see [Zaveri et al., 2015] for a survey). Proposals fit in one of the three categories: 1) cover a single DQ problem (e.g., measure the quality of links between external datasets [Guéret et al., 2012], and measure the age of data [Rula et al., 2012]); 2) can be used for a single specific domain (e.g., evaluate the DQ of the Dbpedia [Kontokostas et al., 2013]); or 3) include few types of DQ problems (e.g., [Fürber and Hepp, 2010a, Chen and Garcia, 2010]). There are only a few works that explicitly classified DQ problems on the Web of Data (e.g., [Fürber and Hepp, 2010b, Kontokostas et al., 2014]). However, they did not bother with a complete and accurate classification of the DQ problems.

The DQ problems handled by [Fürber and Hepp, 2010b] were adapted from the ones of relational databases and grouped into four basic types: 1) data "*inconsistency*" (problems that occurs when the current state is different from the required stated caused by, e.g., divergent syntax or data type); 2) "*lack of comprehensibility*" (i.e., data can be incorrectly interpreted by other applications or users, e.g., data are ambiguous); 3) "*heterogeneity*" (problems in which the representation of identical data varies, e.g., the same real-world object is represented by different schema elements); and 4) "*redundancy*" (problems in which the same real world object or relationship is represented more than once, e.g., existence of synonyms). In [Fürber and Hepp, 2010b], the same DQ problem may be included in more than one group. For example, the *existence of synonyms* is classified as *heterogeneity* and *redundancy*.

[Kontokostas et al., 2014] deals with some DQ problems also present in relational databases as well as some DQ problems that are specific to the Semantic Web world. In particular, [Kontokostas et al., 2014] examines the values of properties marked in the ontology as being disjoint. There are some DQ problems that were not handled by none of these previous works yet. For example, they do not examine values of two related properties looking for improbable values. For example, a person was born in "1899-03-22" and died in 2009, at the age of 200 years old,

is an improbable value. Furthermore, DQ problems involving multiple datasets are not present in any DQ taxonomy of RDF data.

The strategy most used for identifying DQ problems in a single RDF dataset is to perform SPARQL queries[4] (which is the standard query language for RDF) that return the data that has a DQ problem. [Fürber and Hepp, 2010a] developed generic SPARQL queries to identify: 1) missing properties or values; 2) illegal values and 3) functional dependency violation. In [Fürber and Hepp, 2010b], the authors developed a SPARQL query library to be used with SPIN[5] (stands for SParql Inferencing Notation) in order to identify DQ problems in Semantic Web data. [Kontokostas et al., 2014] developed SPARQL query templates, which can be instantiated into concrete data quality test queries. However, there are works in the literature that use different approaches (i.e., they do not use SPARQL to identify DQ problems). Sieve [Mendes et al., 2012], a quality evaluation and conflict resolution module for the LDIF tool [Schultz et al., 2011], uses metadata as quality indicators to produce data quality assessment scores. *Provenance, regency* and *reputation* are examples of metadata used by Sieve to assign a quality assessment score to a given graph. *Regency*, for example, uses a function, named "*TimeCloseness*", in order to measure the distance between two dates and to compute scores that indicate how recent a graph is. The authors assume that approximate duplicate problems are previously resolved and ask to the user to assign scores to the data through user-configured scoring functions. For example, the user can utilize the scoring function *TimeCloseness* to indicate the most recent data. Crocus [Cherix et al., 2014], a system to clean ontology data, uses algorithms to extract data from RDF datasets in a standardized way, and groups the data into clusters using different DQ measures. Outliers are identified and are given to a human for judgement of the data quality.

The main contributions of this paper are:

1. we exhaustively identify and classify the problems that affect DQ on the Web of Data, organizing them according to a taxonomy;
2. we propose a library of SPARQL query templates, which can be used to find the DQ problems;

The remainder of this article is structured as follows. Section 2 presents the basic notions of the RDF data model and SPARQL language. Section 3 shows a running example that will be used through the paper to illustrate our proposal. Section 4 describes our taxonomy of DQ problems on the Web of Data. Section 5 describes the SPARQL query templates proposed to detect DQ problems. Section 6 compares our proposal to related work. Finally, Section 7 presents the conclusions and future work directions.

2 Background

In this section, we present the main concepts that are essential to understand the rest of the paper.

[4] https://www.w3.org/TR/rdf-sparql-query/

[5] SPIN has become a standard to represent SPARQL rules and constraints on Semantic Web models. Available in: https://www.w3.org/Submission/2011/SUBM-spin-overview-20110222/

2.1 Ontologies

Ontologies are a formal specification of a shared, abstract and simplified view of the real world that we want to represent [Gruber, 2003]. They provide a shared and common understanding of a domain that can be communicated across people and application thanks to a logical semantics that enables reasoning.

Briefly speaking, an ontology is formed by vocabularies of terms, and constraints on how the terms in the ontology can be used. On the Semantic Web, vocabularies define the *concepts* (also called *classes*) and *relationships* that exist between the individuals representing a given domain. For example, consider the Music ontology[6]. It includes classes (e.g., `mo:MusicArtist`), which denote usual *concepts* concerning the music world. A class has a set of instances (the *individuals* in this class). For example, `ex:Elvis_Presley` is an instance of the class `mo:MusicArtist`. In addition, the Music ontology includes relationships between classes (e.g., `mo:compiler` is a relationship between the classes `mo:Record` and `mo:MusicArtist`). Relationships also have instances (e.g., `mo:compiler`(`ex:Love_Me_Tender`, `ex:Elvis_Presley`) means that the album *Love Me Tender* was compiled by *Elvis Presley*).

Note that all terms and individuals of the Music ontology have a *prefix* (respectively, "*mo*" and "*ex*"). All ontologies have a *namespace* that indicates the URL where the ontology specification can be found. The prefix *mo* before the term `mo:MusicalArtist` is an abbreviation of the Music ontology namespace, while the prefix *ex*, before the instance `ex:Elvis_Presley`, indicates where a given resource can be found.

2.2 RDF, RDFS and OWL

RDF stands for *Resource* (everything that can have a URI) *Description* (features and relations for the resources) *Framework* (model, languages and syntaxes for these descriptions). Everything described by RDF is called a *resource* and each piece of knowledge is broken down into a triple (***subject, predicate, object***). Hence, RDF is a triple model. For example, the sentence "Elvis Presley was born in January 8, 1935" can be represented in RDF as the triple:

$$\langle \texttt{ex:Elvis\_Presley} \quad \texttt{dbo:birthDate} \quad \text{“}1935-01-08\text{”} \rangle. \tag{1}$$

`ex:Elvis_Presley` (a short form for the URI *http://www.example.org/Elvis_Presley*) is the ***subject***, `dbo:birthDate` is the ***predicate*** (also named ***property***) about the subject and "1935-01-08" is a literal value indicating the date when Elvis Presley was born. RDF triples can be seen as arcs of a graph, where the subjects and objects are the vertices, and the properties are the edge. Hence, RDF is a graph model to link the descriptions of resources.

RDF has the advantage that we can load triples from diverse sources into a triplestore (using, for example, the *Apache Jena TDB*[7]) with no need of reconfiguration. However, it does not give

[6] mo:http://purl.org/ontology/mo/

[7] Apache Jena TDB is a persistent triplestore that stores directly to disk. Documentation available in http://jena.apache.org/documentation/tdb/

any special meaning to schema vocabulary (such as *subClassOf*). In order to resolve this issue, RDF data can be annotated with semantic metadata using two main languages: RDF Schema (RDFS)[8] and Web Ontology Language (OWL)[9].

RDFS is the most basic schema language generally used in the Semantic Web. It is object oriented, which means that, by using RDFS, we can define classes of things, relationships between classes, etc.. All resources described by RDF are instances of the class ***rdfs:Resources***, which is the class of everything. Related resources can be grouped into *classes* (***rdfs:class***). Classes are also resources. Classes are usually identified by URIs and may be described using RDF properties. The members of a class are known as *instances* of the class. Examples of RDF properties are described as follows and then exemplified in order to clarify their semantics.

- ***rdf:type***: used to identify the subject of a triple as *instance of* a class.
- ***rdfs:range***: assigns the values of a property to one or more classes.
- ***rdfs:domain***: states that the resource having a given property is an instance of one or more classes.
- ***rdfs:subClassOf***: states that all instances of a class are also instances of another one.

For better illustrate the differences between RDF and RDFS, consider, besides triple (1), the following ones: ⟨ `ex:Elvis_Presley rdf:type dbo:MusicalArtist`⟩ and ⟨`ex:Elvis_Presley dbo:nationality ex:USA` ⟩. A graphical representation of the RDF graph for this example is presented in Fig. 1, where words in bold indicate RDF terms, words in bold-italic indicate RDFS terms and dashed lines indicate triples that are RDFS-entailed (i.e., triples that are defined by inheritance).

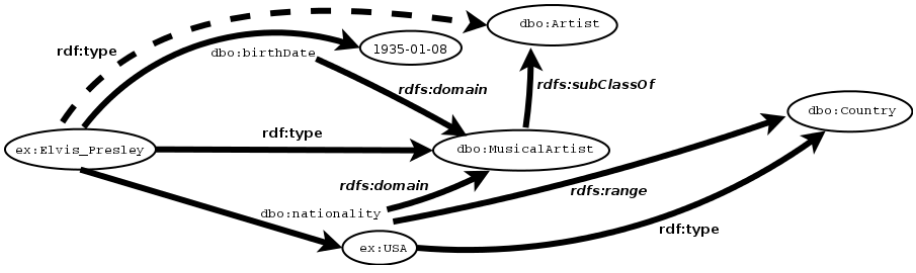


Fig. 1. RDF graph emphasizing both RDF and RDFS terms.

In Fig. 1, `dbo:MusicalArtist`, `dbo:Artist` and `dbo:Country` are classes defined in Dbpedia[10], where `dbo:MusicalArtist` is a subclass of `dbo:Artist`. `dbo:nationality` is a property with domain `dbo:MusicalArtist` and range `dbo:Country`. This means that `dbo:nationality` can be used

[8] https://www.w3.org/TR/rdf-schema/
[9] https://www.w3.org/TR/owl-features/
[10] http://dbpedia.org/ontology/

with `dbo:MusicalArtist` and accepts resources from `dbo:Country` as values. `dbo:birthDate` is a property with domain `dbo:MusicalArtist`.

The OWL language offers much more semantics to the data than RDFS. For example, it enables to define relationships between classes (e.g., *disjointWith*), equality between resources (e.g., *sameAs*) and richer properties (e.g., symmetrical). In addition, it imposes a much more rigid structure than RDFS.

OWL supports the development of ontologies, which can refer to or be referred from other OWL ontologies. In addition, ontologies described using OWL enables to create new triples based on existing ones, to deduce new facts based on stated facts. It is important to keep in mind that an OWL ontology does not define a single model. It contains a set of constraints that define a set of possible models. Examples of OWL constructs are described as follows and then exemplified to clarify their semantics:

- ***owl:class***- define a group of instances that belong together because they share same properties[11].
- ***owl:datatypeProperty***- a property whose range is a set of datatype values (e.g., string, integer, date, etc.),
- ***owl:objectProperty***- a property whose range is a set of resources identified by URIs,
- ***owl:disjointWith***- resources belonging to one class cannot belong to another one.

Figure 2 presents the same RDF and RDFS terms shown in Fig. 1, plus the OWL terms shown in gray.

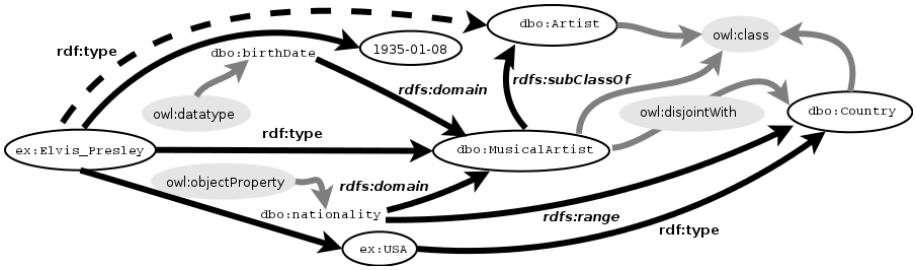


Fig. 2. RDF graph emphasizing terms used in an ontology.

Note that the RDF graph shown in Fig. 2 is semantically richer than the one shown in Fig. 1. We explicitly said that *dbo:nationality* is an object property and *dbo:birthDate* is a datatype property. It indicates that *dbo:nationality* keeps a link (URL) while *dbo:birthDate* keeps a value. Moreover, the graph also indicates that *dbo:MusicalArtist* and *dbo:Country* are disjoint. This means that instances of the former cannot be instances of the latter.

[11] *owl:class* is a subclass of *rdfs:Class* on OWL DL (and OWL Lite) and *owl:class* and *rdfs:Class* are equivalent in OWL Full. OWL DL, OWL Lite and OWL Full are sublanguages of the OWL - see http://www.w3.org/TR/owl-ref/ to more details.

2.3 SPARQL language

RDF triples can be queried through SPARQL, the W3C standard query language. SPARQL has an SQL-like syntax, which makes it easier to learn. A SPARQL query is built over *triple patterns* (i.e., RDF triples that can have variables[12] in the place of the subject, predicate and object position). For example, ⟨?s `foaf:name` ?n ⟩ is a triple pattern with two variables: ?s and ?n. This triple pattern matches with any instance containing a triple with the predicate *foaf:name*. In SPARQL, variable names are prefixed by "?" or "$". In a query, ?s and ?s identify the same variable.

A set of triple patterns is called a *graph pattern*. Thus, SPARQL is based on matching graph patterns against RDF graphs.

A SPARQL query consists of the following sections:

- *prefix declarations*, for abbreviating URIs. Example of SPARQL clauses used in *prefix declarations* are ***prefix*** and ***base***.
- *dataset definition*, states which RDF graphs are being queried (it is optional). Example of SPARQL clauses used in *dataset definition* are ***from*** and ***from named***.
- *result clause*, identifies what information is returned by the query. Example of SPARQL clauses used in *result clause* are ***select, describe, construct*** and ***ask***.
- *graph patterns*, specify the triple patterns to query in the underlying dataset (they are optional). In SPARQL the *graph patterns* are specified in the clause ***where*** (syntax ***where*** { *graph patterns* }).
- *query modifiers*, slice, order and rearrange query results (they are optional). Example of SPARQL clauses used in *query modifiers* are ***limit, order by, distinct, offset*** and ***reduced***.

Figure 3 shows an example of a SPARQL query, which returns the names and birth dates of people in the Portuguese Dbpedia that match the two graph patterns that are specified in the ***where*** clause given in the query. Figure 3(b) shows a possible result of the query shown in Fig. 3(a).

```
prefix foaf: ⟨http://xmlns.com/foaf.0.1/⟩ .
prefix dbo: ⟨http://dbpedia.org/ontology/⟩ .
select ?name ?birth
   from http://pt.dbpedia.org
   where { ?s foaf:name ?name .
           ?s dbo:birthDate ?birth . }
   limit 2
```

(a) SPARQL query.

name	birth
"Tony Carreira"	"1963-12-30"
"Adelaide Ferreira"	"1960-01-01"

(b) Answer of the query.

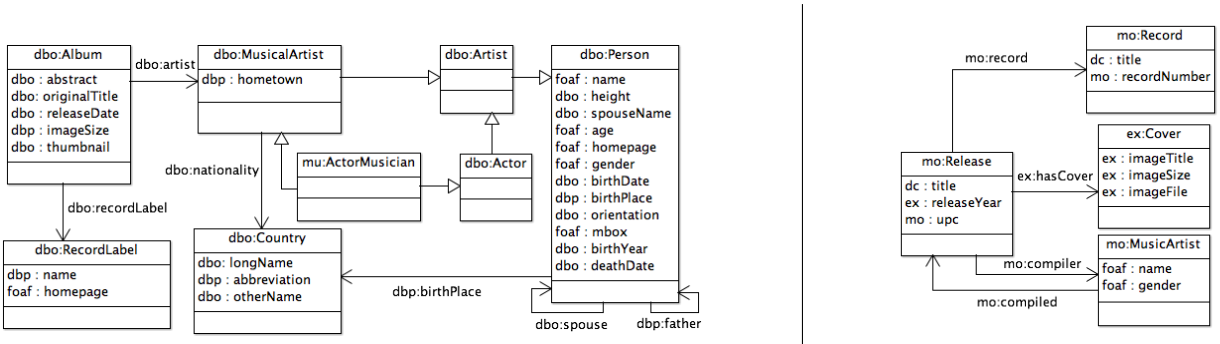
Fig. 3. Example of a SPARQL query with a possible result.

[12] RDF variables have the same semantics as in Prolog or Datalog.

In Fig. 3(a), the ***select*** statement specifies that the values of variables $?name$ and $?birth$ must be returned. The ***from*** clause states that the dataset used is the Portuguese Dbpedia. The ***where*** clause specifies the graph patterns to be matched: ?s *foaf:name* ?name and ?s *dbo:birthDate* ?birth. The triple patterns indicate that we want all resources (?s) that have values for `foaf:name` and `dbo:birthDate` and the result will be mapped into the variables ?*name* and ?*birth*. The ***limit*** clause limits the number of rows returned to 2.

3 Running Example

In order to best illustrate DQ problems, we present examples taken from the two ontologies shown in Fig. 4. Both ontologies are represented using a UML Class Diagram [Fowler, 2003] notation. The rectangles hold the class name on the top and the datatype properties in the middle. The arrow lines ($\rightarrow$) represent the object properties, where the tail indicates the property domain and the head indicates the property range. The arrow lines ($\rightarrow\triangleright$) indicate an inheritance relationship between two classes.



(a) `Music_pt` ontology. (b) `Music_br` ontology.

Fig. 4. Running example.

The `Music_pt` ontology (represented in Fig. 4(a)) models data about artists in a Portuguese dataset. It uses terms of two well-known ontologies: `Dbpedia`[13] (prefixes *dbo* or *dbp*) and `Foaf`[14] (prefix *foaf*) as vocabulary. For example, the class: `dbo:Person` represents people, regardless of being alive or dead and of being real or imaginary and the class `dbo:Artist` represents who practices art. In addition to these widely used terms, the `Music_pt` ontology uses proprietary vocabulary terms (identified by using the prefix "mu"). In particular, the class `mu:MusicActor` is a proprietary vocabulary term to represent the musicians that are also actors.

[13] dbo: http://dbpedia.org/ontology/ and dbp: http://dbpedia.org/property/
[14] http://xmlns.com/foaf/0.1/

The Music_br ontology (represented in Fig. 4(b)) keeps data about musicians in a Brazilian dataset. It uses terms of three well-known ontologies: MUSIC (prefix *mo*), Dublin core[15] (prefix *dc*) and Foaf (prefix *foaf*) as vocabulary. The prefix "ex" is used for proprietary terms defined in Music_br. For example, the class ex:Cover represents details about images used in the covers of the releases.

OWL terms were not included in the Fig. 4 in order to turn the picture simple. For example, there may be interlinks through ***owl:sameAs*** properties between Music_pt and Music_br ontologies, which indicates that there are individuals in both ontologies that actually refer to the same real world object.

4 Data Quality Problems

This section presents our taxonomy of DQ problems on the Semantic Web, which is mainly a systematization of DQ problems found in the literature. We distinguish DQ problems adapted from Relational Databases (RDBs) from DQ problems specific for RDF. The DQ problems that also occur in RDBs were mainly adapted from [Oliveira et al., 2005], but we also included DQ problems introduced in [Rahm and Do, 2000, Li et al., 2011, Gschwandtner et al., 2012] and [Fürber and Hepp, 2010b]. Some problems for RDF data have been proposed in [Kontokostas et al., 2014], others were defined based on DQ problems found in real-world LOD datasets or in [Paulheim, 2014].

As in [Oliveira et al., 2005], we follow a bottom-up approach, starting at the lowest level of data granularity (i.e., the ones that occur in a single property value of a single resource in a single class domain) and ending at the highest level of data granularity (i.e., those that involve multiple datasets). We divided DQ problems into two main groups: *single dataset* and *multiple datasets*, which are detailed in the following sections.

4.1 Single Dataset

This section presents the DQ problems identified within a single dataset. These problems are divided into two groups: those whose properties have the same domain and those whose properties have different domains. Both groups are sub-divided as shown in Fig. 5.

4.1.1 Single Class Domain (SCD)

The DQ problems encountered within properties with the same domain are organized into four groups and are described as follows.

4.1.1.1 SCD1: Value of a single property of a resource These problems are detected by analyzing just the property value of a single resource. For example, when we evaluate individually the value of the property dbo:spouseName, whose domain is dbo:Person, for each person (e.g., if dbo:spouseName has a value, if its syntax is right, etc.). For this group, we list the following DQ problems:

[15] http://purl.org/dc/terms/

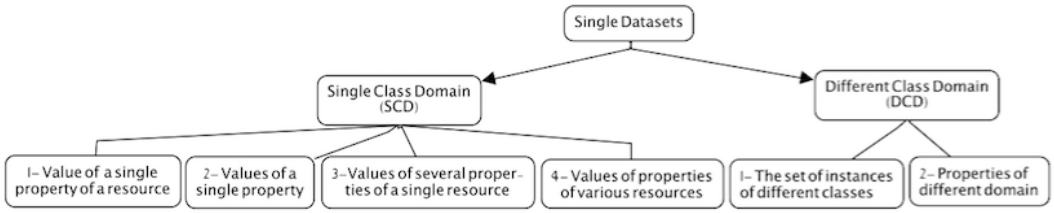


Fig. 5. Data quality taxonomy in a single dataset.

DQ problems that were directly adapted from the Relational Database Model (RDBM):

1. **Missing value [Oliveira et al., 2005] -** Lack of value in a required property. For example: absence of value in the property `foaf:name` in `dbo:Person`.
2. **Syntax violation [Oliveira et al., 2005] -** There is a discrepancy between the syntax of the value of the datatype property and the established syntax for the property. For example: `dbo:birthDate` has the value "1976"^^xsd:gYear rather than "1976-05-11"^^xsd:date. This means that the value of `dbo:birthDate` has only the year when the expected value is a date with format YYYY-MM-DD[16].
3. **Outdated value [Oliveira et al., 2005] -** The datatype property value is not updated and does not correspond to the real situation. For example: `foaf:homepage` stores a value which is not the updated artist homepage address.
4. **Interval violation [Oliveira et al., 2005] -** There is a violation of the interval of values that are considered valid for a datatype property whose type is numeric. For example: `dbo:birthYear` contains values between 0 and 100.
5. **Set violation [Oliveira et al., 2005] -** In an datatype property there is violation of the set of valid values. For example: `foaf:gender` contains "F", when the acceptable values are "female" or "male".
6. **Misspelled error [Oliveira et al., 2005] -** the datatype property contains a misspelling error, accidental or not. For example: the property `dbo:originalTitle`, whose domain is `dbo:Album`, has the value "Twyst and show" when the correct value is "Twist and show" .
7. **Misfielded value (value items beyond the attribute context [Oliveira et al., 2005]) -**The datatype property value does not fit in its context, but fits in the context of another property. For example: the value of `dbp:imageSize` is the name of an image (in fact, it should be the value of `dbo:thumbnail`).
8. **Embedded value (inadequate value to the attribute context [Oliveira et al., 2005]) -** Multiple items introduced all together as a value of a datatype property. Some of these items go beyond the property context. For example: the value "1935-01-08,Tupelo" goes beyond the context of the property `dbo:birthDate`, since it should only store the birth date.

[16] All values in RDF data model are strings. xsd:gYear and xsd:date are used to indicate, respectively, which "1976" is an integer and "1976-05-11" is a date, instead of a string.

9. **Meaningless value [Oliveira et al., 2005] -** Neither the value of a property, nor any subset of it, fit in its context, nor in the context of another existing property. For example: the value "XPTO" in the property `dbo:orientation` has no meaning.
10. **Imprecise meaning (value with imprecise or doubtful meaning [Oliveira et al., 2005]) -** It is a consequence of using abbreviations or acronyms in the value of a datatype property. For example: the value "I Records" (a record label) in the property `dbp:name` can stand for "Island Records" or "Interscope Records".
11. **Circularity violation (circularity among tuples in a self-relationship [Oliveira et al., 2005]) -** It corresponds to cycle situations when a resource has the same value of the subject URI and of the object URI. For example: a resource cannot be its own father (`dbp:father`).
12. **Cardinality violation (illegal value [Rahm and Do, 2000] -** A property assigned to a resource is not in accordance to a given cardinality. For example: A person can only have one birthdate (`dbo:birthDate`).
13. **Datatype violation (use of wrong data type [Li et al., 2011]) -** The value of a datatype property is not in accordance to the type (datatype) of the property. For example: The value of the property `dbo:orientation` has a link (URI), although its type is xsd:string.
14. **Reference violation (referential integrity violation [Oliveira et al., 2005]) -** In an object property, there is a triple whose object URI does not exist as a subject URI in the range class. For example: An artist is referenced in `dbo:artist` but he does not belong to `dbo:MusicalArtist`.
15. **Wrong reference (Outdated reference [Oliveira et al., 2005])-** The value (URI) exists as a subject URI in the range of the class, but it is wrong. For example, the value of the property `dbp:birthPlace` of the resource ex:Elvis_Presley is a link to ex:Canada, which is a wrong value since Elvis Presley was born in USA.
16. **Wrong/incomplete value (lack of data elements [Li et al., 2011]) -** The value that exists in a datatype property is within the range of the property, but it is wrong or incomplete. For example: the Universal Product Code (UPC) of a release (`mo:upc`) has value "0-12345" when its complete value is "0-12345-67890-5".

DQ problems specific to RDF data model:

17. **Language violation (ONELANG pattern [Kontokostas et al., 2014]) -** A datatype property whose type is literal must contain at most one literal for a given language. For example: in a given context, the `foaf:name` must have a single value assigned to the English language tag ("en") .
18. **Missing domain (RDFSDOMAIN pattern [Kontokostas et al., 2014]) -** The assignment of a property to a resource is only allowed when the resource is in the domain of the property. For example: in our example, the domain of `dbp:name` is `dbo:RecordLabel`. Thus, resources in `dbp:name` must also be instances of `dbo:RecordLabel`.
19. **Functional Property violation -** There cannot be two distinct values of a property for each resource (i.e., given a resource x and values y and z of a property p, there cannot be the triples $\langle x\ p\ y\rangle$ and $\langle x\ p\ z\rangle$. For example: a person can have only one age (`foaf:age`). Thus, we can have the triples ⟨ex:a1 `foaf:age` "25"^^xsd:integer⟩ and ⟨ex:a1 `foaf:age` "+25"^^xsd:int⟩, but we cannot have the triple ⟨ex:a1 `foaf:age` "15"^^xsd:integer⟩.

4.1.1.2 SCD2: Values of a single property These DQ problems are detected by analyzing the set formed by the values of a single property of a same domain. For example, two different persons cannot have the same e-mail box (`foaf:mbox`).
DQ problems that were directly adapted from the RDBM:

1. **Uniqueness value violation [Oliveira et al., 2005]**[17] **-** Two or more resources cannot be assigned to the same individual (or value) of the range of the property. For example: two different persons cannot be married with a same person (i.e., `dbo:spouse` and `dbo:spouseName` must have distinct values to different individuals).
2. **Synonym existence [Oliveira et al., 2005] -** Arbitrary use of syntactically different values with the same semantic meaning. For example: the property `dbo:orientation` contains simultaneously values "bisexual" and "ambisexual" for different individuals.

We have not found any DQ problems specific to the RDF data model when considering values of a single property (SCD2 group).

4.1.1.3 SCD3: Values of several properties of a single resource These DQ problems are detected by analyzing the set formed by the values of a single resource. For example, for the same individual, the date when he was born (`dbo:birthDate`) must be consistent with his age (`foaf:age`).
DQ problems that were directly adapted from the RDBM:

1. **Inconsistency between related properties (Inconsistency between related attributes [Oliveira et al., 2005]) -** There is a violation of an existing dependence between values of the datatype properties of a same domain and same resource. For example: the value in `dbo:birthYear` must be the same than the year in `dbo:birthDate`.
2. **Ordering violation (domain violation [Gschwandtner et al., 2012])**[18] **-** Two different values must have a specific ordering with respect to an operator ($=, <, >, \leq, \geq, \neq$), depending on the property semantics. For example: The date in `dbo:deathDate` must come after the date in `dbo:birthDate`.
3. **Conditional missing value (Functionally dependent missing datatype property [Fürber and Hepp, 2010a]) -** Some properties require a value only in certain contexts (i.e., when other properties obtain certain values). For example: `dbp:abbreviation` only requires a value when `dbo:longName` (from `dbp:Country`) has a value.
4. **Unlikely range [Gschwandtner et al., 2012]**[19] **-** Some properties have improbable values. For example: for a same resource, `dbo:birthDate` value is "1899-03-22", while `dbo:deadDate` value is "2009-03-22", so the individual is 200 years old!

DQ problems specific to RDF data model:

5. **Property disjoint violation (OWLDISJP pattern [Kontokostas et al., 2014]) -** The same value cannot be assigned to two different properties that are stated to be disjoint by an ***owl:disjointProperty***. For example: `dbp:father` is disjoint of `dbo:spouse`.

[17] In [Kontokostas et al., 2014] this data quality problem is named *INVFUNC* - property inverse functional.
[18] start, end or duration do not form a valid interval.
[19] Presence of discrepant numeric values.

6. **Sparse resource -** When there is only few properties assigned to a resource and it was supposed to be more. For example: if a person in `dbo:Person` only has values for `foaf:name` but has no value for `foaf:mbox` and both property values must be ground.

4.1.1.4 SCD4: Values of properties of various resources These DQ problems are detected by analyzing the set formed by the values of various resources. For example, if there is inconsistent or duplicated values between instances of the class `dbo:Person`.
DQ problems that were directly adapted from the RDBM:

1. **Redundancy about a resource (Redundancy about an entity [Oliveira et al., 2005]) -** The same resource is represented by an equal or equivalent representation in more than one triple. For example: in `dbo:Person`, we have the triples $<s_1$ `foaf:name` "Chico Buarque">, $<s_1$ `dbp:father` "Sergio Buarque de Holanda">, $<s_2$ `foaf:name` "Chico B. de Holanda">, $<s_2$ `dbp:father` "Sergio B. de Holanda">. s_1 and s_2 represent the same resource.
2. **Inconsistency about a resource (Inconsistency about an entity [Oliveira et al., 2005]) -** There are inconsistencies or contradictions among one or more values of the properties of the same resource. For example: the value of `dbo:spouseName` should be the same that the value of `foaf:name` of the resource referenced by `dbo:spouse`.
3. **Domain constraint violation [Oliveira et al., 2005] -** Violation of constraint involving a class as a whole. The constraint is intrinsic to the domain. For example: in a given context, the maximum number of user accounts (`sioc:account`) allowed to a forum (`sioc:Forum`) is 300[20].

DQ problems specific to RDF data model:

4. **Key violation -** When a set of properties are assigned as a key to a class (***owl:haskey***) and the values of these properties do not uniquely identify an instance of this class. For example: given `dbo:Person` **owl:hasKey** (`foaf:name dbo:birthDate`), there cannot be two different resources with same values to both `foaf:name` and `dbo:birthDate`.

4.1.2 Different Class Domains (DCD)

The DQ problems encountered within resources of different classes are organized into two groups and are described as follows.

4.1.2.1 DCD1: The set of instances of different classes These DQ problems are detected by analyzing the instances of distinct classes. For example, the instances of `dbo:Artist` must include the instances of `dbo:MusicalArtist`, since `dbo:MusicalArtist` is a sub-class of `dbo:Artist`. This category of DQ problems does not exists in RDBMs. The DQ problems specific to RDF data model are described as follows.

[20] The SIOC provides the main concepts and properties required to describe information from online communities on the Semantic Web. Namespace: http://rdfs.org/sioc/ns#.

1. **Class dependency violation (TYPEDEP pattern [Kontokostas et al., 2014]) -** When a resource of a subclass is not a resource of its (direct) superclass. For example: a musician that is a resource of `dbo:MusicalArtist` must also be a resource of `dbo:Artist` (its direct superclass).
2. **Class disjoint violation (OWLDISJC pattern [Kontokostas et al., 2014]) -** when a resource is instance of two disjoint (***owl:disjointWith***) classes. For example: given `dbo:Person` ***owl:disjointWith*** `dbo:RecordLabel`. Then, there cannot be an instance of `dbo:Person` that is also an instance of `dbo:RecordLabel`.
3. **Class intersection violation -** When a resource of a class, which is intersection of two other classes (say *A* and B), is not an instance of both classes *A* and B. For example: consider the class `mu:ActorMusician` be defined as `mu:ActorMusician` ***owl:IntersectionOf*** (`dbo:MusicalArtist dbo:Actor`). So, individuals in `mu:ActorMusician` must be instance of both `dbo:MusicalArtist` and `dbo:Actor`.
4. **Class union violation -** When a resource of a class, which is union of two other classes (say *A* and B), is not instance of at least one of classes *A* or B. For example: Consider the class `dbo:Artist` be defined as `dbo:Artist` ***owl:UnionOf*** (`dbo:MusicalArtist dbo:Actor`). Then resources in `dbo:Artist` must be instances of at least `dbo:MusicalArtist` or of `dbo:Actor`.

4.1.2.2 DCD2: Properties of different domains These DQ problems are detected by analyzing the values of properties of different domain of a same dataset. For example, the syntax of dates (`dbo:birthDate` and `dbo:releaseDate` in our example) must be the same in the whole ontology.
DQ problems that were directly adapted from the RDBM:

1. **Syntax inconsistency [Oliveira et al., 2005] -** There are different representation syntaxes for datatype properties of different domains but with the same data type (e.g.,String, Date, etc.). For example: in `dbo:Person`, the `dbo:birthDate` syntax is dd/mm/yyyy, while in `dbo:Album`, the syntax of `dbo:releaseDate` is yyyy/mm/dd.
2. **Inconsistency between related property values (Inconsistency among related attributes values [Oliveira et al., 2005]) -** There are inconsistencies between property values of different domains that are related to each other. For example: in `dbo:MusicalArtist`, if there is a resource with `dbp:birthPlace` value "Portugal" then its value for `dbo:nationality` must also be "Portugal".

DQ problems specific to RDF data model:

3. **Asymmetry violation (OWLASYMP pattern [Kontokostas et al., 2014]) -** When a property P is asymmetric and there are cases for which holds the pairs (x, y) and (y, x) are instances of P. For example: given that `dbo:birthPlace` is asymmetric, if there is the triple ⟨ ex:Elvis_Presley `dbo:birthPlace` ex:Tupedo ⟩, then we cannot have the triple ⟨ex:Tupedo `dbo:birthPlace` ex:Elvis_Presley ⟩.

4.2 Multiple Datasets (MD)

DQ problems among datasets may occur when these datasets must be analyzed together (e.g., when occurs an integration process), even though each individual dataset has no DQ problems. This section presents the DQ problems identified when analyzing a set of different datasets.
DQ problems that were directly adapted from the RDBM:

1. **Syntax inconsistency [Oliveira et al., 2005] -** depending on the dataset, there are different representation syntaxes for datatype properties which have the same data type. For example: in `mo:MusicArtist`, the`foaf:name` syntax is "last name, first name", while in `dbo:MusicalArtist`, the `foaf:name` is the form "first name last name".
2. **Different measure units [Oliveira et al., 2005] -** Depending on the dataset, different units or measures are used in datatype properties. For example, in `ex:Cover`, the `ex:imageSize` value is in *cm*, while in `dbo:Album`, the `dbp:imageSize` value is in *inch*.
3. **Representation inconsistency [Oliveira et al., 2005] -** different sets of values, from the same type or not, are used in datatype properties from distinct datasets to represent the same situation. For example, in `dbo:MusicalArtist`, the `foaf:gender` can have the values "female" or "male", while in `mo:MusicArtist`, the `foaf:gender` can be "F" or "M".
4. **Synonym existence [Oliveira et al., 2005] -** use of syntactically different property names with the same semantic meaning in properties from distinct datasets. For example: in `mo:Release`, the property `dc:title` has the same meaning than the property `dbo:originalTitle` in `dbo:Album`.
5. **Redundancy about a resource (Redundancy about an entity [Oliveira et al., 2005]) -** the same resource is represented by an equal or equivalent representation in more than one triple from different datasets. For example, in `mo:Release`, we can have the triple <s_1 `dc:title` "Cálice">, <s_1 `ex:releaseYear` "1978">, while in `dbo:Album` we can have the triple <o_1 `dbo:originalTitle` "Cálice">, <o_1 `dbo:releaseDate` "01-01-1978">.
6. **Inconsistency about a resource (Inconsistency about an entity [Oliveira et al., 2005]) -** There are inconsistencies or contradictions in more than one triple in different datasets. For example: in `mo:Release`, we can have the triple <s_1 `dc:title` "Cálice">, <s_1 `ex:releaseYear` "1976">, while in `dbo:Album` we can have the triple <o_1 `dbo:originalTitle` "Cálice">, <o_1 `dbo:releaseDate` "01-01-1978">.
7. **Domain constraint violation [Oliveira et al., 2005] -** instances of equivalent classes belonging to different datasets individually respect a constraint inherent to the domain, but these instances cannot respect the constraints when both classes are considered together or in a common scenario. For example: suppose that in `ex:Cover`, the maximum size (in cm) of an image (`ex:imageSize`) is "600 x 450", while in `dbo:Album`, the maximum size (in inch) of an image (`dbp:imageSize`) is "3.5 x 8.1" (i.e., 900 x 2050 cm). When considered together, there can be instances of the class `dbo:Album` whose size of image is out of range '600 x 450".

DQ problems specific to RDF data model:

8. **Dangling data/referential integrity violation [Paulheim, 2014] -** In a property that interlinks datasets, such as ***owl:sameAs***, there is a triple in a dataset whose object URI does not exists as a subject URI in the another dataset. For example, in `Music_pt` the ***owl:sameAs*** link ⟨s1 ***owl:sameAs*** <http://example.org/resources/369>⟩ can exist indicating that the resources $s1$ in `Music_pt` and 369 in `Music_br` represent the same individual. However, the resource 369 is not in `Music_br`.
9. **Wrong references (incorrect references [Paulheim, 2014]) -** In a property that interlinks datasets, such as ***owl:sameAs***, the resources interlinked are in the respective datasets. However, they do not represent the same individual. For example, in `Music_pt` the ***owl:sameAs*** link ⟨s1 ***owl:sameAs*** <http://example.org/resources/369>⟩ can exist indicating that the resources $s1$ in `Music_pt` and 369 in `Music_br` represent the same individual. However, $s1$ is an URI to "Paulo Ricardo Ribeiro", a writer, while 369 is an URI to "Paulo Ricardo", a singer. Hence, the ***owl:sameAs*** between the resources $s1$ and 369 is a wrong reference.

5 Using Sparql Queries to Detect DQ Problems

This section presents some SPARQL query templates that can be used to identify DQ problems on the Semantic Web data. We argue that our SPARQL query templates can be used for testing the existence of DQ problems in a single dataset (after they are instantiated into concrete queries), or they can be attached to classes of a vocabulary and be used to check constraint violations (e.g., cardinality violations). In the latter case, the SPARQL Inferencing Notation (SPIN) framework[21] can be used to attach our SPARQL query templates to the classes of entities that contains the data to be checked. After the creation of the query templates and its customization, SPIN can be used to automatically identify the DQ problems. This is part of our future work.

In the following text, we show some of our SPARQL query templates with the respective example. The list of our templates defined until now can be found in Appendix A. We adopted the following notation in the templates :

- The parameters to be passed to the queries are presented between ⟨ ⟩ ;
- ⟨$class$⟩ (also ⟨$class_i$⟩,1 ≤ i ≤ n) indicates that the required parameter is a class;
- ⟨prop⟩ (also ⟨$prop_i$⟩,1 ≤ i ≤ n) indicates that the required parameter is a (datatype/object) property.

Table 1. SPARQL query to detect DQ problems SCD1/5 (Set violation).

Sparql Query Template	Sparql Query Example
select distinct ?s ?v **where** { ?s a ⟨$class$⟩ . ?s ⟨$prop$⟩ ?v . **filter** (⟨nop⟩ (?v = ⟨v_1⟩|| ?v = ⟨v_2⟩|| ... || ?v = ⟨v_n⟩)). }	**select distinct** ?s ?v **where** { ?s a *dbo:Person* . ?s *foaf:gender* ?v . **filter** (! (?v = "female" || ?v = "male")) .}

[21] http://spinrdf.org/spin.html

Table 1 shows a SPARQL query template to identify DQ problems of type SCD1/5 (set violation). Also, the table presents an example of SPARQL query generated from this template. The query returns the resource $?s$ and the value $?v$ of the property ⟨prop⟩ when $?v$ does not belong to the set of values $\langle v_1\rangle$, $\langle v_2\rangle$, ..., $\langle v_n\rangle$ that are the valid values for ⟨prop⟩. In the example, the values of property *foaf:gender*, whose domain is *dbo:Person*, is examined and the query returns the resources $?s$ and values $?v$ for which $?v$ is different from "female" and "male".

Note that in Table 1 we have the parameter $\langle nop\rangle$. This parameter means the negation operator ("!" in SPARQL) and is optional in the query. We decided to put $\langle nop\rangle$ as a parameter because sometimes is easier to examine values that are valid and sometimes is easier to examine values that are invalid.

Table 2. SPARQL query to detect DQ problems SCD2/1 (Uniqueness value violation).

Sparql Query Template	Sparql Query Example
select distinct ?s1 **where** { ?s1 a ⟨*class*⟩ ; ⟨*prop*⟩ ?v1 . ?s2 a ⟨*class*⟩ ; ⟨*prop*⟩ ?v2 . **filter** ((?s1 != ?s2) && (str(?v1) = str(?v2))) . }	**select distinct** ?s1 **where** { ?s1 a *dbo:Person* ; *foaf:mbox* ?v1 . ?s2 a *dbo:Person* ; *foaf:mbox* ?v2 . **filter** ((?s1 != ?s2) && (str(?v1) = str(?v2))) . }

Table 2 shows a SPARQL query template to identify DQ problems of type SCD2/1 (uniqueness value violation), as well as it shows the example of a query generated from this template. The query returns the resources $?s1$ for which there is another resource $?s2$ (different from $?s1$), such that both resources have the same value to a given property $\langle prop\rangle$. In the example, the resources $?s1$ returned will be those of *dbo:Person* for which there is another resource $?s2$ with same value for the property *foaf:mbox* than the *foaf:mbox* of $?s1$.

Table 3. SPARQL query to detect DQ problems SCD3/6 (Sparse Resource).

Sparql Query Template	Sparql Query Example
select ?s **where** { ?s a ⟨*class*⟩ . **not exists** { ?s ⟨prop1⟩ ?v1 . } **not exists** { ?s ⟨prop2⟩ ?v2 . } ... **not exists** { ?s ⟨propn⟩ ?vn . } . }	**select** ?s **where** { ?s a *dbo:Person* . **not exists** { ?s *foaf:name* ?v1 . } **not exists** { ?s *foaf:mbox* ?v2 . } . }

Table 3 shows a SPARQL query template to identify DQ problems of type SCD3/6 (sparse resource), as well as it shows the example of a query generated from this template. The query returns the resources $?s$ that are instances of $\langle class\rangle$, such that some of the required properties ⟨prop1⟩, ⟨prop2⟩, ..., ⟨propn⟩ are missing. In the example, the resources $?s$ returned will be those of *dbo:Person* that have not values for *foaf:name* or *foaf:mbox*.

Table 4. SPARQL query to detect DQ problems SCD4/4 (Key violation).

Sparql Query Template	Sparql Query Example
select distinct ?s1 ?s2 **where** {	**select distinct** ?s1 ?s2 **where** {
optional {?s1 a ⟨*class*⟩ ;	**optional** {?s1 a *dbo:Person* ;
⟨K1⟩ ?v1 ; ... ; ⟨Kn⟩ ?vn . }	*foaf:name* ?v1 ; *dbo:birthDate* ?v2 . }
optional {?s2 a ⟨*class*⟩ ;	**optional** {?s2 a *dbo:Person* ;
⟨K1⟩ ?w1 ; ... ; ⟨Kn⟩ ?wn . }	*foaf:name* ?w1 ; *dbo:birthDate* ?w2 . }
filter ((?s1 != ?s2) &&	**filter** ((?s1 != ?s2) &&
((str(?v1) = str(?w1)) &&	((str(?v1) = str(?w1)) &&
... && (str(?vn) = str(?wn))) . }	&& (str(?v2) = str(?w2))) . }

Table 4 shows a SPARQL query template to identify DQ problems of type SCD4/4 (key violation), as well as it shows the example of a query generated from this template. The query returns the resources $?s1$ and $?s2$ for which the values of the properties ⟨K1⟩ ,. . . , ⟨Kn⟩ (defined in the ontology as ***owl:hasKey***) are, respectively, the same for each instance $?s1$ and $?s2$. In the example, the resources $?s1$ and $?s2$ returned will be those of *dbo:Person* that have the same value for the properties *foaf:name* and *dbo:birthDate*.

6 Comparison with Related Work

There are several taxonomies of DQ problems in the literature for relational data (see [Batini et al., 2009] for a survey). In this Section, we briefly discuss the most relevant ones.

[Rahm and Do, 2000] classify DQ problems for RDBMs into two groups: (i) single-source and multi-sources problems and (ii) schema- and instance-related problems. Schema level problems are those that can be addressed at a schema level, while instance-related problems refer to errors and inconsistencies in the actual data contents. The authors identified fourteen DQ problems involving single-source and six DQ problems involving multiple-sources.

[Kim et al., 2003] presented a classification of dirty data for RDBMs consisting of thirty-three DQ problems. The authors organized their taxonomy by categories, adopting a pattern of successive hierarchical decompositions. The taxonomy shown in [Kim et al., 2003] starts with a root node with only two child nodes: *missing data* and *non-missing data*. These nodes are further refined such that all obvious cases have been covered.

[Oliveira et al., 2005] presented a very complete taxonomy for RDBMs with thirty-five DQ problems identified. Their taxonomy is structured under six levels ranging from the lowest level DQ problems (in a single attribute value of a single tuple) to the highest DQ problems (multi-sources problems).

[Li et al., 2011] provided a rule-based taxonomy of DQ problems for RDBMs, which is used to make the selection of a specific group of types of DQ problems to satisfy different business needs. Their taxonomy is grouped in four different categories: Business entity rules, business attribute rules, data dependency rules and data validity rules. Thirty-eight different DQ problems were identified, covering DQ problems occurring in single and multiple data sources.

[Gschwandtner et al., 2012] provided a taxonomy of DQ problems to be addressed by time-oriented data (e.g., time points, intervals, etc.). They also classified DQ problems accorded to single-source and multi-sources. Each group is subdivided into non-rastered and rastered data[22].

Table 5 presents a comparison between the taxonomy for single-source scenario proposed in the article and the other DQ problems related in [Rahm and Do, 2000, Kim et al., 2003, Oliveira et al., 2005, Li et al., 2011, Gschwandtner et al., 2012, Fürber and Hepp, 2010a, Fürber and Hepp, 2010b, Fürber and Hepp, 2011, Kontokostas et al., 2014]. The first two lines of the table present the DQ problems in accordance to our taxonomy. Each proposed DQ problem item is compared with items described by other taxonomies. Items marked with "•" symbol indicate partial or full coverage of the problem, even if the nomenclature used is different. Cells that are unfilled indicate that the problem identified is not covered.

Some items of the proposed taxonomy are present in all other (i.e., SCD1/1- missing value, SCD1/14- reference violation , SCD2/1- uniqueness value violation). Other issues are more or less coverage by the works. However, there are DQ problems that has been disregarded by works in the context of this article (e.g., SCD1/12- cardinality violation and SCD3/4- unlikely range).

Table 5. Conventional DQ problems within a single source

Taxonomy of	**SCD1**																**SCD2**		**SCD3**				**SCD4**			**DCD2**	
DQ Problems	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1	2	1	2	3	4	1	2	3	1	2
[Rahm and Do, 2000]	●	●		●		●	●	●		●		●		●			●						●	●			●
[Kim et al., 2003]	●		●			●	●	●		●			●	●		●	●		●					●			
[Oliveira et al., 2005]	●	●	●	●	●	●	●	●	●	●	●			●	●		●	●	●				●	●	●	●	●
[Li et al., 2011]	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●		●				●	●			●
[Gschwandtner et al., 2012]	●	●	●	●			●	●		●			●	●	●	●	●			●		●	●	●		●	
Fürber and Hepp[1]	●	●	●	●		●	●		●	●				●	●	●	●	●			●		●	●	●		●
[Kontokostas et al., 2014]	●	●		●							●	●	●	●			●			●	●						

[1] DQ problems shown in [Fürber and Hepp, 2010a, Fürber and Hepp, 2010b, Fürber and Hepp, 2011].

Table 6 shows a comparison between the taxonomy for the multiple sources scenario proposed in this article and the other DQ problems related in [Rahm and Do, 2000, Kim et al., 2003, Oliveira et al., 2005, Li et al., 2011, Gschwandtner et al., 2012]. This Table uses the same notation than Table 5. How observed in Table 5, in Table 6 also there are DQ problems that was covered by all cited works (i.e., MD2- different measure units and MD3- representation inconsistency) and some DQ problems that were covered by ones and not by others. None of these works deal with the DQ problems MD8- dangling data and MD9- wrong references in the context of this work. However, there are in the literature works dedicated exclusively to deal with this type of DQ problem (e.g., [Volz et al., 2009a, Vidal et al., 2016, Isele et al., 2010, Volz et al., 2009b, Paulheim, 2014]).

[22] Raster data consists of a matrix of cells (or pixels) organized into rows and columns (or a grid).

Table 6. Conventional DQ problems within multiple sources

Taxonomy of DQ Problems	MD								
	1	2	3	4	5	6	7	8	9
[Rahm and Do, 2000]		●	●	●	●	●	●		
[Kim et al., 2003]		●	●			●			
[Oliveira et al., 2005]	●	●	●	●	●	●	●		
[Li et al., 2011]		●	●	●	●	●			
[Gschwandtner et al., 2012]	●	●	●						

Table 7. DQ problems specific for RDF triples handled in [Kontokostas et al., 2014].

Category	DQ Problem
SCD1/17	Language violation
SCD1/18	Missing domain
SCD3/5	Property disjoint violation
DCD1/1	Class Dependency violation
DCD1/2	Class disjoint violation
DCD2/3	Asymmetry violation

We can observe that [Oliveira et al., 2005] and [Li et al., 2011] cover most of the DQ problems presented in both Tables. Fürber and Hepp's articles and [Kontokostas et al., 2014] are both works that presented DQ problems for RDF data. Fürber and Hepp adapted their taxonomy from [Oliveira et al., 2005] and they do not deal with DQ problems specific to the Semantic Web world (e.g., those present in OWL schema axioms such as ***owl:hasKey*** and ***owl:disjointWith***). [Kontokostas et al., 2014] deals with DQ problems from relational databases as well as from specific to the Semantic Web world. In particular, [Kontokostas et al., 2014] examines the values of properties marked in the ontology as being disjoint and proposed a DQ problem (named *Triple pattern*), which consider a resource as "erroneous if there are corresponding hints contained in the dataset"). Table 7 shows DQ problems related in [Kontokostas et al., 2014] that are specific to the Semantic Web world. Their focus was not on presenting a DQ taxonomy. Instead, their work presented some SPARQL templates to detect DQ problems in RDF datasets.

In this paper, we exhaustively identified the problems that affect DQ in RDF data and grouped them in categories in accordance to the level of data granularity. The suggested taxonomy covers all problems shown in Tables 5, 6 and 7. We excluded some DQ problems related in the other works, which do not arise to RDF data (e.g., *different aggregation levels* presented in [Oliveira et al., 2005]) or that are very specific (e.g., the *Triple pattern* presented in [Kontokostas et al., 2014]). In addition, we identified the following new DQ problems:

- SCD1/19 - Functional property violation
- SCD3/6 - Sparse resource
- SCD4/4 - Key violation[23]
- DCD1/3 - Class intersection violation
- DCD1/4 - Class union violation

Besides the DQ taxonomy, we defined SPARQL templates to detect the DQ problems in datasets. Fürber and Hepp [Fürber and Hepp, 2010a, Fürber and Hepp, 2010b, Fürber and Hepp, 2011] and [Kontokostas et al., 2014] also presented generic SPARQL queries for the identification of DQ problems. The former only presented nine generic SPARQL queries to deal with

[23] Key constraints already exist in relational databases, but they are handled by DBMS. In the Web Semantic world, until we know, there is not an underlying system to deal with Key constraints.

problems such as missing value, uniqueness value violation and syntax violation, etc.. [Kontokostas et al., 2014] presented a pattern-based approach for the DQ tests of RDF datasets. The authors described a library consisting of seventeen DQ patterns. Each pattern has a name, a description of the DQ problem handled, a SPARQL query template and an example of use. We included some of the templates presented in these works in the current paper with some slight changes or improvements. In addition, we proposed fourteen SPARQL templates.

7 Conclusion

Motivated by the large use of SPARQL queries, by the easiness of using them and by the difficulty in found which DQ problems for RDF data were covered by existing works, we proposed a DQ taxonomy to RDF data. This taxonomy was defined based on:

- Analysis of the existing DQ problems in the relational world and its adaptation to the RDF world.
- Analysis of languages to define RDF data and the analysis of RDF data in order to identify DQ problems specific to the RDF data.
- Grouping of the DQ problems in accordance to the data granularity levels (similar to what was done for relational data in [Oliveira et al., 2005]).
- Classification of the existing DQ problems found in the literature in accordance to the suggested taxonomy.

Our taxonomy exhaustively identified the problems that affect DQ in RDF data and grouped them into two major groups: single-dataset and multiple-datasets. The former was sub-divided into 6 groups (in accordance to the data granularity level). We believe that this organization will assist the user in the process of identifying DQ problems, since that enables to make DQ tests in the datasets in accordance to the level of data granularity. In addition, the taxonomy can be used to measure the coverage of a DQ tool used in the process of detection/correction of DQ problems that affect data.

Our ongoing work focus on the development of a methodology to apply SPARQL query templates in an appropriate order. This enable us to define simple queries that can be performed in sequence in order to identify the DQ problems. We also are working in the definition of SPARQL templates to the DQ problems of the taxonomy for which there is not a template yet.

As future work, we plan to develop a tool to help the user to identify DQ problems in the Semantic Web, using our taxonomy and SPARQL templates.

ACKNOWLEDGEMENTS

This work has been supported by national funds through FCT - Fundação para a Ciência e a Tecnologia, under the project PEst-OE/EEI/LA0021/2013, DataStorm Research Line of Excellency funding (EXCL/EEI-ESS/0257/2012) and the grant SFRH/BPD/76024/2011.

References

Batini et al., 2009. Batini, C., Cappiello, C., Francalanci, C., and Maurino, A. (2009). Methodologies for data quality assessment and improvement. *ACM Comput. Surv.*, 41(3):16:1–16:52.

Berners-Lee, 1998. Berners-Lee, T. (1998). Semantic web road map. Online, available in http://www.w3.org/DesignIssues/Semantic.html.

Berners-Lee, 2006. Berners-Lee, T. (2006). Linked data. http://www.w3.org/DesignIssues/LinkedData.html.

Bizer et al., 2009. Bizer, C., Heath, T., and Berners-Lee, T. (2009). Linked data - the story so far. *International Journal on Semantic Web and Information Systems, Special Issue on Linked Data*, 5(3):1–22.

Chen and Garcia, 2010. Chen, P. and Garcia, W. (2010). Hypothesis generation and data quality assessment through association mining. In *Cognitive Informatics (ICCI), 2010 9th IEEE International Conference on*, pages 659–666, Beijing, China. IEEE.

Cherix et al., 2014. Cherix, D., Usbeck, R., Both, A., and Lehmann, J. (2014). Crocus: Cluster-based ontology data cleansing. In *Proceedings of the 2nd International Workshop on Semantic Web Enterprise Adoption and Best Practice*, pages 7–14, Anissaras, Crete, Greece. CEUR-WS.

Fensel, 2001. Fensel, D. (2001). *Ontologies*, pages 11–18. Springer Berlin Heidelberg, Berlin, Heidelberg.

Fowler, 2003. Fowler, M. (2003). *UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition)*. Addison-Wesley Professional.

Fürber and Hepp, 2010a. Fürber, C. and Hepp, M. (2010a). Using semantic web resources for data quality management. In *Proceedings of the 17th International Conference on Knowledge Engineering and Management by the Masses*, EKAW'10, pages 211–225, Berlin, Heidelberg. Springer-Verlag.

Fürber and Hepp, 2010b. Fürber, C. and Hepp, M. (2010b). Using sparql and spin for data quality management on the semantic web. In Abramowicz, W. and Tolksdorf, R., editors, *Business Information Systems*, volume 47 of *Lecture Notes in Business Information Processing*, pages 35–46. Springer-Verlag, Berlin, Heidelberg.

Fürber and Hepp, 2011. Fürber, C. and Hepp, M. (2011). Towards a vocabulary for data quality management in semantic web architectures. In *Proceedings of the 1st International Workshop on Linked Web Data Management*, LWDM '11, pages 1–8, New York, NY, USA. ACM.

Gruber, 2003. Gruber, T. (2003). It is what it does: The pragmatics of ontology for knowledge sharing. Invited presentation to the meeting of the CIDOC Conceptual Reference Model committee, Smithsonian Museum.

Gschwandtner et al., 2012. Gschwandtner, T., Gärtner, J., Aigner, W., and Miksch, S. (2012). A taxonomy of dirty time-oriented data. In Quirchmayr, G., Basl, J., You, I., Xu, L., and Weippl, E., editors, *Lecture Notes in Computer Science (LNCS 7465): Multidisciplinary Research and Practice for Information Systems (Proceedings of the CD-ARES 2012)*, pages 58 – 72, Prague, Czech Republic. Springer, Berlin / Heidelberg, Springer, Berlin / Heidelberg.

Guéret et al., 2012. Guéret, C., Groth, P., Stadler, C., and Lehmann, J. (2012). Assessing linked data mappings using network measures. In Simperl, E., Cimiano, P., Polleres, A., Corcho, O., and Presutti, V., editors, *Proceedings of the 9th International Conference on The Semantic Web: Research and Applications*, volume 7295 of *ESWC'12*, pages 87–102. Springer-Verlag, Berlin, Heidelberg.

Isele et al., 2010. Isele, R., Jentzsch, A., and Bizer, C. (2010). Silk server - adding missing links while consuming linked data. In *Proceedings of the First International Workshop on Consuming Linked Data*, China.

Kim et al., 2003. Kim, W., Choi, B.-J., Hong, E.-K., Kim, S.-K., and Lee, D. (2003). A taxonomy of dirty data. *Data Min. Knowl. Discov.*, 7(1):81–99.

Kontokostas et al., 2014. Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., and Zaveri, A. (2014). Test-driven evaluation of linked data quality. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 747–758, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

Kontokostas et al., 2013. Kontokostas, D., Zaveri, A., Auer, S., and Lehmann, J. (2013). Triplecheckmate: A tool for crowdsourcing the quality assessment of linked data. In Klinov, P. and Mouromtsev, D., editors, *Knowledge Engineering and the Semantic Web*, volume 394 of *Communications in Computer and Information Science*, pages 265–272. Springer Berlin Heidelberg, Berlin, Heidelberg.

Li et al., 2011. Li, L., Peng, T., and Kennedy, J. (2011). A rule based taxonomy of dirty data. *GSTF Journal on Computing (JoC)*, 1(2):140–148.

Mendes et al., 2012. Mendes, P. N., Mühleisen, H., and Bizer, C. (2012). Sieve: Linked data quality assessment and fusion. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, EDBT-ICDT '12, pages 116–123, New York, NY, USA. ACM.

Oliveira et al., 2005. Oliveira, P., Rodrigues, F., Henriques, P., and Galhardas, H. (2005). A taxonomy of data quality problems. In *2nd Int. Workshop on Data and Information Quality*, pages 219–233, Porto, Portugal. FEUP edições.

Paulheim, 2014. Paulheim, H. (2014). Identifying wrong links between datasets by multi-dimensional outlier detection. In *Third International Workshop on Debugging Ontologies and Ontology Mappings-WoDOOM14*, page 27. Citeseer.

Rahm and Do, 2000. Rahm, E. and Do, H. H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13.

Rula et al., 2012. Rula, A., Palmonari, M., and Maurino, A. (2012). Capturing the age of linked open data: Towards a dataset-independent framework. In *Semantic Computing (ICSC), 2012 IEEE Sixth International Conference on*, pages 218–225, Palermo, Italy. IEEE.

Schultz et al., 2011. Schultz, A., Matteini, A., Isele, R., Bizer, C., and Becker, C. (2011). Ldif - linked data integration framework. In *Proceedings of the Second International Conference on Consuming Linked Data*, volume 782 of *COLD'11*, pages 125–130, Aachen, Germany, Germany. CEUR-WS.org.

Vidal et al., 2016. Vidal, V., Casanova, M., Menendez, E., Arruda, N., Pequeno, V., and Leme, L. (2016). Using changesets for incremental maintenance of linkset views. In *17th International Conference on Web Information Systems Engineering – WISE 2016*, China.

Volz et al., 2009a. Volz, J., Bizer, C., Gaedke, M., and Kobilarov, G. (2009a). Discovering and maintaining links on the web of data. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 650–665, Berlin, Heidelberg. Springer-Verlag.

Volz et al., 2009b. Volz, J., Bizer, C., Gaedke, M., and Kobilarov, G. (2009b). Silk – a link discovery framework for the web of data. In *2nd Workshop about Linked Data on the Web (LDOW2009)*, Madrid, Spain.

Zaveri et al., 2015. Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., and Auer, S. (2015). Quality assessment for linked data: A survey. *the Semantic Web Journal*, 7(1):63–93.

Appendix A Generic SPARQL queries

This section presents generic SPARQL queries that can be used to detect DQ problems in the Web of Data or in a local dataset. We also present examples for each kind of query. Queries from related work are duly marked.

We adopted the following notation:

- The parameters to be passed to the queries are presented between ⟨ ⟩.
- The variables *class* and *prop* means, respectively, a class and a (datatype/object) property.
- $class_i$ ($1 \leq i \leq n$) mean classes.
- $prop_i$ ($1 \leq i \leq n$) mean (datatype/object) properties.
- v_i and w_i ($1 \leq i \leq n$) represent literal values or URI values.
- *nop* means the negation operator ("!" in SPARQL) and is optional in the query.
- *op* means the comparison operator ("<, >, <=, >=, =, !=" in SPARQL).

A.1 SCD1(1) Missing value

SPARQL Query (based on the template shown in [Fürber and Hepp, 2010a]):

```
select ?s where {
  {?s a ⟨class⟩ . ?s ⟨prop⟩ ?v .
   filter (?v = "" || ?v = " ") . }
  union {?s a ⟨class⟩ .
         not exists { ?s ⟨prop⟩ ?v . }}}
```

Example: Query to verify the absence of value in the property *foaf:gender*

```
select ?s where {
  {?s a dbo:Person . ?s foaf:gender ?v .
   filter (?v = "" || ?v = " ") . }
  union {?s a dbo:Person .
         not exists { ?s foaf:gender ?v . }}}
```

A.2 SCD1(2) Syntax violation

SPARQL Query (adapted from [Kontokostas et al., 2014])[24]:

```
select ?s ?v where {
  ?s a ⟨class⟩ . ?s ⟨prop⟩ ?v .
  filter (⟨nop⟩ regex(str(?v),"⟨regexExpr⟩")) . }
```

⟨*regexExpr*⟩ is the regular expression passed as parameter.

Example: Query to verify which values of property *foaf:name* do not contain only letters.

[24] This query differ to the original only because here we force the property belongs to a given domain.

```
select ?s ?v where {
  ?s  a dbo:Person . ?s  foaf:name ?v .
  filter (!regex(str(?v),"^[A–Za–z,.'_]*$")) . }
```

A.3 SCD1(4) Interval violation

SPARQL Query[24]:

Values between an interval (adapted from [Kontokostas et al., 2014]):

```
select distinct ?s ?v where {
  ?s a ⟨class⟩ . ?s ⟨prop⟩ ?v .
  filter (⟨nop⟩ (?v < ⟨v_min⟩ || ?v > ⟨v_max⟩ ) ) . }
```

$\langle v_{min} \rangle$ and $\langle v_{max} \rangle$ are, respectively, the minimum and maximum values passed as parameter.

Values exceed maximum allowed (adapted from [Fürber and Hepp, 2010b]):

```
select distinct ?s ?v where {
  ?s a ⟨class⟩ . ?s ⟨prop⟩ ?v .
  filter (?v > ⟨v_max⟩ ) . }
```

Values do not reach minimum allowed (adapted from [Fürber and Hepp, 2010b]):

```
select distinct ?s ?v where {
  ?s a ⟨class⟩ . ?s ⟨prop⟩ ?v .
  filter (?v < ⟨v_min⟩ ) . }
```

Example: Query to verify which values of property *dbo:birthYear* are before 1500.

```
select distinct ?s ?v where{
  ?s a dbo:Person . ?s dbo:birthdate ?v .
  filter (?v < "1499-01-01"^^xsd:date) . }
```

A.4 SCD1(5) Set violation

SPARQL Query:

```
select distinct ?s ?v where {
  ?s a ⟨class⟩ . ?s ⟨prop⟩ ?v .
  filter (⟨nop⟩ (?v = ⟨v1⟩ || ?v = ⟨v2⟩ || ... || ?v = ⟨vn⟩) ) }
```

Example: Query to verify which values of property *foaf:gender* are not ”female” or ”male”.

```
select distinct ?s ?v where{
  ?s a dbo:Person . ?s foaf:gender ?v .
  filter (!(?v = ”female” || ?v = ”male” ) ) . }
```

A.5 SCD1(6) Misspelled error

SPARQL Query:

Using a reliable source (adapted from a query shown in [Fürber and Hepp, 2010a]):

```
select ?s1 ?v where {
  ?s1 a ⟨class1⟩ . ?s1 ⟨prop1⟩ ?v .
  optional {service ⟨serv⟩ { ?s2 a ⟨class2⟩.?s2 ⟨prop2⟩ ?v.
            filter (lang(?v) = ”⟨lang⟩”) .}}
  filter (!bound(?s2)).}
```

The **optional** keyword enables a SPARQL query returns information even when part of the graph pattern does not match. The **service** keyword extends SPARQL to support queries that merge data distributed across the Web. $\langle serv \rangle$ is the URL to this service that must be passed as parameter. $\langle lang \rangle$ is the language tag passed as parameter to indicate the language of ?v. The clause ***filter*** *(lang(?v) = ”*$\langle lang \rangle$*”)* is optional since it is usual we have properties without lang tags defined.

Example: Query to verify which values of property *dbp:hometown* have misspelled errors. In this example linkedGeoData is used as the reliable source.

```
select ?s ?v where{
  ?s a dbo:MusicalArtist . ?s dbp:hometown ?v .
  optional{service <http://linkedgeodata.org/sparql> {
            ?s2 a lgdo:City . ?s2 rdfs:label ?v . }}
  filter(!bound(?s2)) . }
```

A.6 SCD1(9) Meaningless value

SPARQL Query:

```
select ?s1 ?v where {
  ?s1  a ⟨class1⟩ . ?s1 ⟨prop1⟩ ?v .
  filter(regex(str(?v), "[aeyiuo]{3,}", "i") ||
         regex(str(?v),"[b-df-hj-np-tv-z]{4,}","i" ) ||
         regex(str(?v), ".*([a-zA-Z])\\1{2,}.*") ||
         regex(str(?v), "xpto","i")||
         regex(str(?v), ".*[^a-zA-Z\\s].*"))   . }
```

This query does not cover all cases of meaningless value errors, just it looks for values of a property that have three vowels together, four consonants together, symbols (such as #, $, @), the word *xpto* and three consonants or vowels repeated.

Example: Query to verify which values of property *dbo:orientation* have a meaningless value.

```
select ?s ?v where{
  ?s a dbo:Person . ?s dbo:orientation ?v .
  filter(regex(str(?v), "[aeyiuo]{3,}", "i") ||
         regex(str(?v),"[b-df-hj-np-tv-z]{4,}","i" ) ||
         regex(str(?v), ".*([a-zA-Z])\\1{2,}.*") ||
         regex(str(?v), "xpto","i") ||
         regex(str(?v), ".*[^a-zA-Z\\s].*")) . }
```

A.7 SCD1(11) Circularity violation

SPARQL Query (adapted from [Kontokostas et al., 2014][24]):

```
select distinct ?s where {
  ?s  a ⟨class⟩ . ?s ⟨prop⟩ ?s . }
```

Example: Query to verify if there a resource in *dbo:Person* do not sign a father (*dbp:father*) to himself.

```
select distinct ?s where{
  ?s a dbo:Person . ?s dbp:father ?s . }
```

A.8 SCD1(12) Cardinality violation

SPARQL Query (adapted from [Kontokostas et al., 2014][24]):

```
select distinct ?s where {
  ?s  a ⟨class⟩ . ?s ⟨prop⟩ ?v . }
  group by ?s having(count(?v) ⟨op⟩ ⟨card⟩)
```

⟨*card*⟩ is a positive number indicating the desired cardinality.
Example: Query to verify if the cardinality of the property *dbo:birthdate* is 1 .

```
select distinct ?s where{
  ?s a dbo:Person . ?s dbo:birthDate ?v . }
  group by ?s having(count(?v) > 1)
```

A.9 SCD1(13) Datatype violation

SPARQL Query (adapted from [Kontokostas et al., 2014][24]):

```
select distinct ?s where {
  ?s  a ⟨class⟩ . ?s ⟨prop⟩ ?v .
  filter(datatype(?v) != ⟨dtype⟩) . }
```

⟨*dtype*⟩ is the desired datatype (e.g., xsd:string, xsd:integer, etc.).
Example: Query to verify if a value of *dbo:birthDate* is of the type xsd:date .

```
select distinct ?s where{
  ?s a dbo:Person . ?s dbo:birthDate ?v .
  filter(datatype(?v) != xsd:date . }
```

A.10 SCD1(14) Reference violation

SPARQL Query:

```
select distinct ?s where {
  ?s  a ⟨class⟩ . ?s ⟨prop⟩ ?v .
  ⟨prop⟩ rdfs:range ?range .
  filter not exists {?v a ?rangeV .
                     ?rangeV rdfs:subClassOf* ?range . } }
```

Example: Query to verify if an artist refereced in dbo:artist belongs to dbo:MusicalArtist.

```
select distinct ?s where{
  ?s a dbo:Album . ?s dbo:artist ?v .
  dbo:artist rdfs:range ?range .
  filter not exists {?v a ?rangeV .
                     ?rangeV rdfs:subClassOf* ?range . } }
```

A.11 SCD1(17) Language violation

SPARQL Query (adapted from [Kontokostas et al., 2014][24]):

```
select distinct ?s where {
  ?s a ⟨class⟩ . ?s ⟨prop⟩ ?v .
  bind( lang(?v) as ?l )
  filter (isLiteral(?v) && lang(?v) = ⟨langTag⟩) .
  } group by ?s having (count(?l) > 1)
```

Example: Query to verify if *foaf:name* has more than one value (the name of a person) in English.

```
select distinct ?s where {
  ?s a dbo:Person . ?s foaf:name ?v .
  bind( lang(?v) as ?l )
  filter (isLiteral(?v) && lang(?v) = "en") .
  } group by ?s having (count(?l) > 1)
```

A.12 SCD1(18) Missing domain

SPARQL Query (adapted from [Kontokostas et al., 2014][25]:

```
select distinct ?s where {
  ?s ⟨prop⟩ ?v .
  filter not exists {?s rdf:type ?Class .
                     ?Class rdfs:subClassOf* ⟨class⟩} }
```

Example: Query to verify if there is a resource that has a value to *foaf:name*, but is not an instance of *dbo:Person*.

```
select distinct ?s where {
  ?s foaf:name ?v .
  filter not exists {?s rdf:type ?Class .
                     ?Class rdfs:subClassOf* dbo:Person } }
```

A.13 SCD1(19) Functional Property violation

SPARQL Query:

```
select ?s (count(distinct ?newV) as ?countV) where {
  ?s  a ⟨class⟩ . ?s ⟨prop⟩ ?v .
  bind(str(?v) as ?newV) .
  } group by ?s having(count(?v) > 1 && ?countV > 1)
```

[25] This is a less verbose query than one presented in [Kontokostas et al., 2014]

Example: Query to verify if there is more than one *dbo:birthDate* for a same person (*dbo:Person*).

```
select ?s (count(distinct ?newV) as ?countV) where {
  ?s a dbo:Person . ?s dbo:birthDate ?v .
  bind(str(?v) as ?newV) .
  } group by ?s having(count(?v) > 1 && ?countV > 1)
```

A.14 SCD2(1) Uniquiness value violation

SPARQL Query (based on the template shown in [Kontokostas et al., 2014, Fürber and Hepp, 2010b]):

```
select distinct ?s1 where {
  ?s1 a ⟨class⟩ ; ⟨prop⟩ ?v1 .
  ?s2 a ⟨class⟩ ; ⟨prop⟩ ?v2 .
  filter ((?s1 != ?s2) && (str(?v1) = str(?v2)) ) }
```

Example: Query to verify if there is a same email (*foaf:mbox*) to different people (*dbo:Person*).

```
select distinct ?s1 where{
  ?s1 a dbo:Person ; foaf:mbox ?v1 .
  ?s2 a dbo:Person ; foaf:mbox ?v2 .
  filter ((?s1 != ?s2) && (str(?v1) = str(?v2)) ) }
```

A.15 SCD3(1) Inconsistency between related properties

SPARQL Query:

```
select distinct ?s where {
  s a ⟨class⟩ ; ⟨prop1⟩ ?v1 ; ⟨prop2⟩ ?v2 .
  filter (⟨funct1⟩(?v1) != ⟨funct2⟩(?v2)) } }
```

Example: query to verify if there is a violation between the properties *dbo:birthDate* and *dbo:birthYear* (both has as domain *dbo:Person*), since the value of *dbo:birthYear* must be the same that the year of *dbo:birthYear* when *dbo:birthYear* is bound.

```
select distinct ?s where{
 ?s a dbo:Person ;    dbo:birthDate ?v1 ; dbo:birthYear ?v2 .
 filter(year(xsd:date(?v1)) != xsd:integer(?v2)) } }
```

A.16 SCD3(2) Ordering violation

SPARQL Query (adapted from [Kontokostas et al., 2014][26]):

```
select  distinct ?s where {
  ?s a ⟨class⟩ ; ⟨prop1⟩ ?v1 ; ⟨prop2⟩ ?v2 .
  filter (⟨function⟩(?v1) ⟨op⟩ ⟨function⟩(?v2)) . }
```

Example: Query to verify if the value of *dbo:deathDate* is greater than the value of *dbo:birthDate*.

```
select distinct ?s where {
  ?s a dbo:Person ; dbo:deathDate ?v1 ; dbo:birthDate ?v2 .
  filter (xsd:date(?v1) < xsd:date(?v2)) . }
```

A.17 SCD3(3) Conditional missing value

SPARQL Query (adapted from [Fürber and Hepp, 2010a]):

```
select distinct ?s where {
  {?s a ⟨class⟩ ; ⟨prop1⟩ ?v1 .
   not exists{?s ⟨prop2⟩ ?v2 . } }
  union {
   ?s a ⟨class⟩ ; ⟨prop1⟩ ?v1 .
   ?s ⟨prop2⟩ "" . }
  union {
   ?s a ⟨class⟩ ; ?s ⟨prop1⟩ ?v1 .
   ?s ⟨prop2⟩ " " . } }
```

[26] This query differ to the original because here we force the property belongs to a given domain and we use functions to guarantee that the comparison is done in accordance to the type of the data (xsd:date, str, xsd:integer, etc.).

Example: Query to verify if *dbp:abbreviation* has a value when the country name (*dbo:longName*) is bound.

```
select distinct ?s where {
  {?s a dbo:Country ; dbo:longName ?v1 .
   not exists{?s dbp:abbreviation ?v2 . } }
  union {
    ?s a dbo:Country ; dbo:longName ?v1 .
    ?s dbp:abbreviation "" }
  union {
    ?s a dbo:Country ; dbo:longName ?v1 .
    ?s dbp:abbreviation " " . } }
```

A.18 SCD3(4) Unlikely range

This is a data quality problem specific to the domain, so we cannot propose a solution that covers all cases. Here we suggest a query to detect when two properties whose type is date, and are related to each other, have unlikely values. For instance, when the difference between the birth date and the death date is more than 100 years!

SPARQL Query:

```
select distinct ?s where {
  ?s a ⟨class⟩ ; ⟨prop1⟩ ?v1 ; ⟨prop2⟩ ?v2 .
  bind(xsd:date(?v1) as ?birth) .
  bind(xsd:date(?v2) as ?death) .
  filter(year(?death) - year(?birth) -
         if(month(?death) < month(?birth) ||
         (month(?death)=month(?birth) &&
         day(?death)<day(?birth)),1,0) ⟨op⟩ ⟨value⟩ ) . }
```

Example:

```
select distinct ?s where {
 ?s a dbo:Person ; dbo:birthDate ?v1 ; dbo:deathDate ?v2 .
 bind(xsd:date(?v1) as ?birth) .
 bind(xsd:date(?v2) as ?death) .
 filter(year(?death) - year(?birth) -
        if(month(?death)<month(?birth) ||
        (month(?death)=month(?birth) &&
        day(?death)<day(?birth)),1,0) > 100 ) . }
```

A.19 SCD3(5) Property disjoint violation

SPARQL Query (adapted from [Kontokostas et al., 2014]):

```
select distinct ?s where {
  ?s a ⟨class⟩ ; ⟨prop1⟩ ?v1 ; ⟨prop2⟩ ?v2 .
  filter(str(?v1) = str(?v2)) . }
```

Example: Query to verify if *dbp:abbreviation* has a value when the country name (*dbo:lonfName*) is bound.

```
select distinct ?s where {
  ?s a dbo:Person ; dbp:father ?v1 ; dbo:spouse ?v2 .
  filter(str(?v1) = str(?v2)) . }
```

A.20 SCD3(6) Sparse resource

SPARQL Query:

```
select ?s where {
  ?s a ⟨class⟩ .
  not exists { ?s ⟨prop1⟩ ?v1 . }
  not exists { ?s ⟨prop2⟩ ?v2 . }
  ...
  not exists { ?s ⟨propn⟩ ?vn . } }
```

Example: Query to verify if a resource of *dbo:Person* has not a values to some of the properties *foaf:name* and *dbo:mbox*.

```
select ?s where {
  ?s a dbo:Person .
  not exists { ?s foaf:name ?v1 .}
  not exists {?s dbo:mbox ?v2 . } }
```

A.21 SCD4(2) Inconsistency about a resource

SPARQL Query, where:
⟨prop1⟩ is functional dependent of ⟨prop2⟩/⟨prop3⟩

```
select distinct ?s1 where {
  ?s1 a ⟨class⟩ ; ⟨prop1⟩ ?s2 ; ⟨prop2⟩ ?v1 .
  ?s2 a ⟨class⟩ ; ⟨prop3⟩ ?v2 .
  filter(str(?v1) != str(?v2)) }
```

Example: query to verify if there is a violation between the properties *dbo:spouse* and *dbo:spouseName* (both has as domain *dbo:Person*), since the value of *dbo:spouseName* must be the same that the value of *foaf:name* of the resource referenced by *dbo:spouse*.

```
select distinct ?s1 where{
  ?s1 a dbo:Person ; dbo:spouse ?s2 ; dbo:spouseName ?v1 .
  ?s2 a dbo:Person ; foaf:name ?v2 .
  filter(str(?v1) != str(?v2)) } }
```

A.22 SCD4(4) Key violation

SPARQL Query:

```
select distinct ?s1 ?s2 where {
  optional {?s1 a ⟨class⟩ ; ⟨K1⟩ ?v1 ; ... ; ⟨Kn⟩ ?vn . }
  optional {?s2 a ⟨class⟩ ; ⟨K1⟩ ?w1 ; ... ; ⟨Kn⟩ ?wn . }
  filter ((?s1 != ?s2) && ((str(?v1) = str(?w1)) &&
          ... && (str(?vn) = str(?wn)))) }
```

Example: Query to verify if there is more than one resource assigned to the same values to the properties *foaf:name, dbo:birthDate, dbo:spouse* that uniquely identify instances of *dbo:Person*.

```
select distinct ?s1 ?s2 where {
  optional {?s1 a dbo:Person;foaf:name ?v1; dbo:birthDate ?v2 .}
  optional {?s2 a dbo:Person;foaf:name ?w1; dbo:birthDate ?w2 .}
  filter ((?s1 != ?s2) && ((str(?v1) = str(?w1)) &&
          (str(?v2) = str(?w2))) }
```

A.23 DCD1(1) Class dependency violation

SPARQL Query:

```
select distinct ?s ?superClass where {
  ?s a ⟨class⟩ .
  ⟨class⟩ rdfs:subClassOf ?superClass .
  filter not exists { ?s a ?superClass . } }
```

Example: Query to verify if a resource of *dbo:MusicalArtist* is not a resource of their direct superclass (in our example, it is just *dbo:Artist*).

```
select distinct ?s ?superClass where {
  ?s a dbo:MusicalArtist .
  dbo:MusicalArtist rdfs:subClassOf ?superClass .
  filter not exists { ?s a ?superClass . } }
```

A.24 DCD1(2) Class disjoint violation

SPARQL Query (from [Kontokostas et al., 2014]):

```
select distinct ?s where {
  ?s a ⟨class1⟩ .
  ?s a ⟨class2⟩ . }
```

Example: Query to verify if a resource of *dbo:Person* is also a resource of *dbo:RecordLabel*.

```
select distinct ?s where {
  ?s a dbo:Person .
  ?s a dbo:RecordLabel .}
```

A.25 DCD1(3) Class intersection violation

SPARQL Query:

```
select distinct ?s where {
  ?s a ⟨class⟩ .
  filter not exists {?s a ⟨class1⟩ . ?s a ⟨class2⟩ . }
                    ...
                    ?s a ⟨classn⟩ . } }
```

Example: Query to verify if a resource of *dbo:MusicActor* is also a resource of both *dbo:MusicalArtist* and *dbo:Actor*.

```
select distinct ?s where {
  ?s a mu:MusicActor .
  filter not exists{?s a dbo:MusicalArtist . ?s a dbo:Actor .} }
```

A.26 DCD1(4) Class union violation

SPARQL Query:

```
select distinct ?s where {
    ?s a ⟨class⟩ .
    filter not exists {?s a ⟨class1⟩ .
    filter not exists {?s a ⟨class2⟩ . }
    ...
    filter not exists {?s a ⟨classn⟩ . } }
```

Example: Query to verify if a resource of *dbo:MusicActor* is also a resource of both *dbo:MusicalArtist* and *dbo:Actor*.

```
select distinct ?s where {
  ?s a dbo:Artist .
  filter not exists{?s a dbo:MusicalArtist . }
  filter not exists{?s a dbo:Actor . } }
```

A.27 DCD2(1) Syntax inconsistency

SPARQL Query:

```
select distinct ?s ?v where {
  ?s a ⟨class⟩ ; ⟨prop⟩ ?v .
  filter (!regex(str(?v),"⟨regexExpr⟩")) .}
```

Example: Query to verify if the resources of *dbo:releaseDate* obey the same format that the resources in *dbo:birthDate* (i.e., YYYY-MM-DD).

```
select distinct ?s ?v where {
  ?s a dbo:Album ; dbo:releaseDate ?v .
  filter (!regex(str(?v),"^([0-9]{4})-([0-9]{2})-([0-9]{2})$")).}
```