# Annotation of existing databases using Semantic Web technologies: making data more FAIR

Johan van Soest[1,2], Ananya Choudhury[1], Nikhil Gaikwad[1], Matthijs Sloep[1], Michel Dumontier[2], Andre Dekker[1]

[1] Department of Radiation Oncology (MAASTRO), GROW School for Oncology and Developmental Biology, Maastricht University Medical Centre+, Maastricht, the Netherlands
[2] Institute of Data Science, Maastricht university, Maastricht, The Netherlands
*johan.vansoest@maastro.nl

**Abstract.** Making data FAIR is an elaborate task. Hospitals and/or departments have to invest into technologies usually unknown and often do not have the resources to make data FAIR. Our work aims to provide a framework and tooling where users can easily make their data (more) FAIR. This framework uses RDF and OWL-based inferencing to annotate existing databases or comma-separated files. For every database, a custom ontology is build based on the database schema, which can be annotated to describe matching standardized terminologies. In this work, we describe the tooling developed, and the current implementation in an institutional datawarehouse pertaining over 3000 rectal cancer patients. We report on the performance (time) of the extraction and annotation process by the developed tooling. Furthermore, we do show that annotation of existing databases using OWL2-based reasoning is possible. Furthermore, we show that the ontology extracted from existing databases can provide a description framework to describe and annotate existing data sources. This would target mostly the "Interoperable" aspect of FAIR.

**Keywords:** FAIR, annotations, terminologies, linked data.

## 1 Introduction

Semantic interoperability has been a topic in medical informatics since the introduction of the digital patient chart [1]. However, in recent years, the interoperability aspect is only one of the issues, covered by the FAIR (findable, accessible, interoperable, reusable) principles [2]. These principles extend the interoperable aspect towards methods to find and access information, and hence promoting reuse of clinical data for primary or secondary purposes.

Although these principles are perceived as the way forward, implementation is an elaborate task. Hospitals and/or their departments have to invest into concepts usually unknown to them, with limited direct insights into the benefit of making data FAIR. Furthermore, it needs many competences from different specializations ranging from the IT department (where is what information stored?), and the medical specialty

itself (what does a specific value actually mean, and what's the provenance of this information?).

Next to these organizational hurdles, the current tools to make data FAIR are not guiding novice users. Most of the tools are heavily tied into Semantic Web technologies and require knowledge about these technologies. This approach works for institutes and companies investing into these technologies, however, doesn't scale towards making data (more) FAIR for hospitals with limited dedicated IT staff. As an effect, the adoption of these new technologies becomes even more difficult as the return on investment is never shown due to a lack of investigation.

Hence, our aim was to provide a framework and tooling where users can easily make their data (more) FAIR, while reducing the amount of information needed to get started. Specifically, to annotate existing datasets using RDFS and OWL statements.

## 2 Methods

This section is split into two parts. First, we will introduce the methods used to make data more FAIR. The second part describes our experimental setup.

### 2.1 FAIR data descriptions extraction

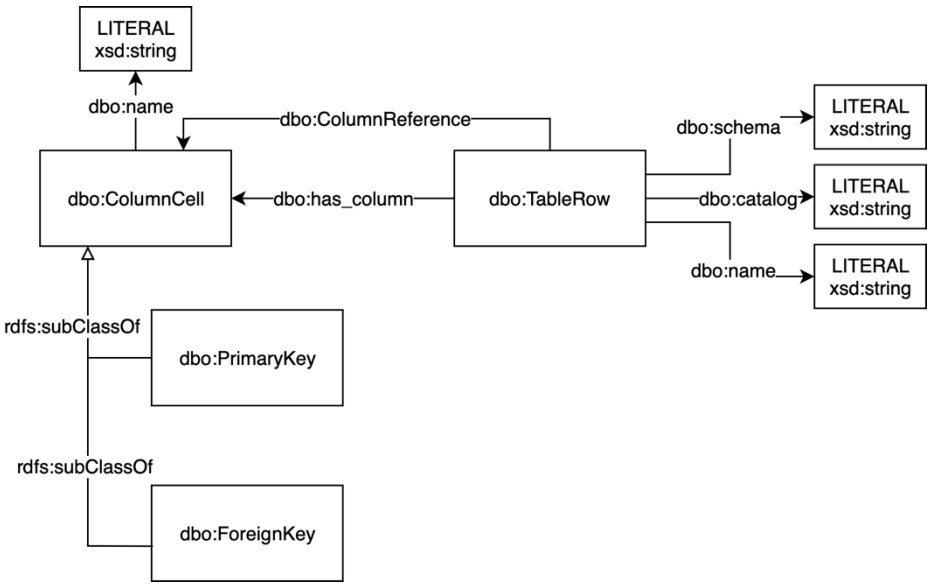
Our implementation expects a database, or folder with comma-separated values files as input of our framework. In this process, the data description language (DDL) is being read from the given database structure. This DDL contains the table definitions (table name, column names) as well as key information (primary keys, foreign keys). This information is used to build an ontology description, where all tables and columns are unique classes. More specifically, all table class definitions are subclasses of the overarching class *dbo:TableRow*, and all columns are subclasses of the overarching class *dbo:ColumnCell*. All *dbo:ColumnCell* classes are annotated to which *dbo:TableRow* they are belonging. Primary key columns are specified as a subclass of *dbo:PrimaryKey*, where this class itself is a subclass of *dbo:ColumnCell*. The same holds for ForeignKey, with additional class annotations regarding the table and column it refers to. A visual representation of this schema is given in Figure 1.

After describing the database structure in the given ontology, instance triples can be materialized for every row and cell in the given database. Hence, every database row will be an instance of the subclass of *dbo:TableRow*, and every cell for this row will be an instance a subclass of *dbo:ColumnCell*. Instances of *dbo:TableRow* and *dbo:ColumnCell* are associated using the predicate *dbo:has_column*. The URIs of *dbo:TableRow* instances will be based on the table name and primary key column(s). This URI is also used as the base for the *dbo:ColumnCell* URIs. The actual cell values are literals connected to the instance of *dbo:ColumnCell*. For foreign key relation-

ships, no instances for *dbo:ForeignKey* will be created, only a direct relation from *dbo:TableRow* to the referred instance of *dbo:ColumnCell* will be created.

This process of materialization can be executed regularly (e.g. daily) to extract updated information from clinical systems, or regularly updated databases.

**Fig. 1.** Visual representation of base ontology structure




Although the ontology specifies the database schema and relationships, it does not add any binding to standardized terminologies for the values in the database. In this process, the subclasses of *dbo:ColumnCell* are annotated with rdfs:equivalentClass axioms. For example, the classes *myOntology:myTable.age* and *myOntology:myTable.gender* (both a subclass of *dbo:ColumnCell*), can have equivalent class definitions *ncit:C25150* and *ncit:C28421*, respectively. Using reasoning, it would make the instances of *myOntology:myTable.age* also findable as the more standardized term *ncit:C17357*.

Depending on the type of variable, this would solve the terminology binding issue for numerical values (depending on the strictness of the equivalent class term definition). However, for categorical variables, the literal value of the instance does not correspond to a category defined in a target terminology. We can overcome this issue by defining restriction classes. For example, when we have gender coded as “m” and “f”, we could implement the following definitions described in Manchester syntax:

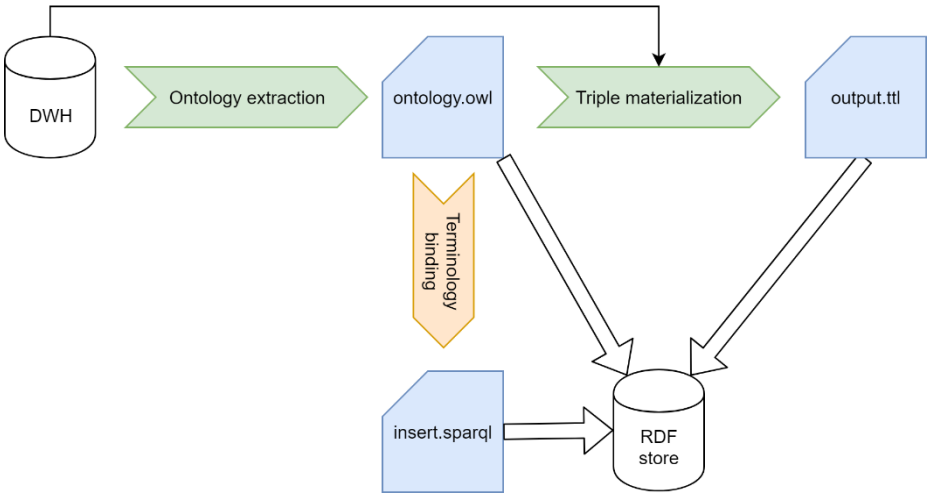
```
Equivalent class for Male (ncit:C20197):
ncit:C28421 and (dbo:has_value “m”^^xds:string)

Equivalent class for Female (ncit:C16576):
ncit:C28421 and (dbo:has_value “f”^^xds:string)
```

By adding these RDF statements to the ontology, the reasoner will assign the NCIT term for male or female to the instance of *myOntology:myTable.gender*. This is possible as it was previously defined that this was an equivalent class with *ncit:C28421*.

This database ontology, combined with annotations to bind standardized terminology terms, can be published as a description of the dataset at hand. Being RDF data, we can search in this ontology using SPARQL to find which data elements are available, and its database schema. Furthermore, when materialization of database contents is performed, we can directly query the instances as well. The workflow of all steps in this process are described in Figure 2.

**Fig. 2.** Workflow of the complete process. Green elements are automated processes, orange elements are manual tasks, and blue elements are the products of these processes.



### 2.2 Testing environment

We tested the above explained methods by developing a tool which can extract the DDL from the given database to build the given ontology. Furthermore, based on this ontology, the tool can materialize the triples for the database rows and cells.

This tool was executed on a data-mart of our institutional datawarehouse containing diagnostic, treatment and questionnaire information of over 4000 rectal cancer patients treated with radiation therapy [3]. This data mart is nightly refreshed, and hence our materialization process was synchronized with this nightly refresh.

Afterwards we defined our terminology binding using the provided methods in the previous section. We stored these triples in a separate graph (together with the ontology) to make these triples persistent over the nightly refresh process.

All of this was executed on a virtual machine running Ubuntu 18.04 with 2 vCPUs and 8GB of RAM. We used Ontotext GraphDB version 8.4.1 as our Graph database management system, with "OWL2-RL (optimized)" reasoning enabled in the endpoint. The Java Virtual Machine (GraphDB execution environment) was configured with a maximum memory consumption of 4GB RAM. We measure the times for the following individual steps in the process: (1) extracting the ontology from the given database, (2) materializing the triples, (3) uploading the ontology (4) uploading the triples into the RDF store (5) adding the annotation reasoning rules. This process was measured for 20 consecutive executions.

## 3 Results

The tool to extract the ontology, and to materialize the triples is publicly available at https://gitlab.com/UM-CDS/fair/tools/triplifier. This tool is available as a stand-alone java application or can be executed as a service in a docker container. Further instructions are given in the repository itself.

The created ontology, and the annotation triples are stored in https://gitlab.com/UM-CDS/fair/data/maastro-rectum. Here we can see that 9 equivalent classes are defined, and 13 rules for mapping terminological values for 4 categorical variables. Next to these separate files, there is also an integrated ontology file, where both extracted ontology and annotation rules are combined. We also included an example query to retrieve whether a given variable is available, and in which table this information is available. Furthermore, an extended query is available which also retrieves the instances for a given patient. Due to patient privacy reasons, this query can only be executed on the institutional RDF endpoint.

The time measurements for the conversion process are given in Table 1. This shows reasonable performance for a daily refresh where we see that the actual upload and parsing process within GraphDB takes most of the time.

**Table 1.** Time measurements for specific steps in the process

| Step description | Mean time in seconds (SD) | Remarks |
|---|---|---|
| Extract the ontology | 0.6 (0.03) | 2 tables with |
| Materializing triples | 66.4 (1.14) | 3.38 million triples |
| Upload ontology in RDF store | 0.1 (0.02) | |
| Upload materialized triples | 482,7 (3.46) | |
| Adding annotation reasoning rules | 103.7 (1.03) | Dependent on number of annotations |

## 4 Conclusion & Discussion

We have shown that it is possible to make data more FAIR, by automatically extracting existing data schemas, and annotating these schemas with inferencing rules. Furthermore, we have shown this process can be performed with materialization of data, however it can be performed without the materialization step as well. In the latter case, the ontology and reasoning rules can be used to deduce where to find specific information in an existing database. Afterwards, specific database query mechanisms can be used to retrieve the actual information. Hence, the materialization step in our current workflow is an optional step, which can be used if necessary.

This also applies to the reasoning rules used to annotate the given database. We used an inferencing-enabled database in the current example; however this is not mandatory. Although this is a machine-readable approach of describing the data, we can describe the given information in a human-interpretable manner (e.g. male gender, defined by *ncit:C20197*, is in table "patient" column "gender", coded as the cell value "m").

Instead of following the R2RML direct mapping recommendation [4], we built our own tool which does not make use of this recommendation. To deviate from this recommendation, we had several arguments. First, R2RML Direct Mapping means we have another intermediate description format (based on RDF), where we omitted this by developing an ontology directly linked to the database. Hence, in our solution, the end-users do not need to understand an additional description language (R2RML) to understand where specific information is available. Second, different R2RML implementations build specific SQL queries, which sometimes become more complex (using join conditions) based on the database schema given. In our approach, we only perform one SQL query to materialize the data ("SELECT * FROM <tablename>"), which is almost sure to work on every RDBMS system (even a folder of CSV files using the CSV JDBC connector[1]). Third, the W3C direct mapping specification defines every cell value as a literal, belonging to an instance of the row. This is different from our current implementation, where every cell is an instance of the column class.

[1] http://csvjdbc.sourceforge.net/

The latter approach makes it possible to reason over equivalent classes, which is not possible when table cells are literals. This could be solved by making custom R2RML files, however these would not follow the direct mapping specification.

Our work is currently limited in only handling RDBMS and CSV/TSV filesystems, however the concept of annotating RDF classes to standardized terminologies is more broadly applicable. For example, these annotations can be used in combination with the Data2Services approach [5]. With respect to Data2Services, our approach differs in the process of terminology binding. Where Data2Services requires one to build SPARQL queries to convert terminologies and the data schema in one (or multiple) queries, our approach only performs the terminology binding. This can be seen as both a limitation and strength. The limitation is that the schema is going to vary, making the task for the data consumer more intensive. On the other hand, by not changing the data schema, it is clearer to the data consumer what the intrinsic issues with the source data are [6]. Furthermore, the knowledge to convert data (using SPARQL queries) is not readily available in many hospitals or data providers. Hence, we feel that a choice between our approach and Data2Services depends on the use case, and who should handle data conversion.

This brings us to the future work, where we want to build a web-interface on top of this framework, to guide hospitals to annotate their data source(s). This would hide the complexity of the reasoning rules to annotate specific database fields. Furthermore, we want to use this platform (including the web-interface proposed) in the Personal Health Train infrastructure [7, 8], to more rapidly include more FAIR data stations to expand the network [9].

## References

1. Shortliffe, E.H., Cimino, J.J. eds: Biomedical Informatics: Computer Applications in Health Care and Biomedicine. Springer-Verlag, London (2014).
2. Wilkinson, M.D., Dumontier, M., Aalbersberg, Ij.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L.B., Bourne, P.E., Bouwman, J., Brookes, A.J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C.T., Finkers, R., Gonzalez-Beltran, A., Gray, A.J.G., Groth, P., Goble, C., Grethe, J.S., Heringa, J., 't Hoen, P.A.C., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S.J., Martone, M.E., Mons, A., Packer, A.L., Persson, B., Rocca-Serra, P., Roos, M., van Schaik, R., Sansone, S.-A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M.A., Thompson, M., van der Lei, J., van Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J., Mons, B.: The FAIR Guiding Principles for scientific data management and stewardship. Sci. Data. 3, 160018 (2016). https://doi.org/10.1038/sdata.2016.18.
3. Meldolesi, E., van Soest, J., Alitto, A.R., Autorino, R., Dinapoli, N., Dekker, A., Gambacorta, M.A., Gatta, R., Tagliaferri, L., Damiani, A., Valentini, V.: VATE: VAlidation of high TEchnology based on large database analysis by learning machine. Colorectal Cancer. 3, 435–450 (2014). https://doi.org/10.2217/crc.14.34.

4. A Direct Mapping of Relational Data to RDF, https://www.w3.org/TR/rdb-direct-mapping/.
5. Emonet, V., Malic, A., Zaveri, A., Grigoriu, A., Dumontier, M.: Data2Services: enabling automated conversion of data to services. 10.
6. Lovis, C.: Digital health: A science at crossroads. Int. J. Med. Inf. 110, 108–110 (2018). https://doi.org/10.1016/j.ijmedinf.2017.12.006.
7. van Soest, JohanP.A., Dekker, AndreL.A.J., Roelofs, E., Nalbantov, G.: Application of Machine Learning for Multicenter Learning. In: El Naqa, I., Li, R., and Murphy, M.J. (eds.) Machine Learning in Radiation Oncology. pp. 71–97. Springer International Publishing (2015).
8. Deist, T.M., Jochems, A., van Soest, J., Nalbantov, G., Oberije, C., Walsh, S., Eble, M., Bulens, P., Coucke, P., Dries, W., Dekker, A., Lambin, P.: Infrastructure and distributed learning methodology for privacy-preserving multi-centric rapid learning health care: euroCAT. Clin. Transl. Radiat. Oncol. 4, 24–31 (2017). https://doi.org/10.1016/j.ctro.2016.12.004.
9. Lambin, P., Roelofs, E., Reymen, B., Velazquez, E.R., Buijsen, J., Zegers, C.M.L., Carvalho, S., Leijenaar, R.T.H., Nalbantov, G., Oberije, C., Scott Marshall, M., Hoebers, F., Troost, E.G.C., van Stiphout, R.G.P.M., van Elmpt, W., van der Weijden, T., Boersma, L., Valentini, V., Dekker, A.: 'Rapid Learning health care in oncology' – An approach towards decision support systems enabling customised radiotherapy'. Radiother. Oncol. 109, 159–164 (2013). https://doi.org/10.1016/j.radonc.2013.07.007.