

# Architecture for Mapping Relational Database to OWL Ontology: An Approach to Enrich Ontology Terminology Validated with Mutation Test

Cristiane A. G. Huve<sup>1,2</sup><sup>a</sup>, Alex M. Porn<sup>1</sup><sup>b</sup> and Leticia M. Peres<sup>1</sup><sup>c</sup>

<sup>1</sup>*Department of Informatics, Federal University of Paraná, Av. Cel. Francisco H. dos Santos, 100, Curitiba, Brazil*

<sup>2</sup>*Polytechnic School, Uninter, Rua Luiz Xavier, 103, Curitiba, Brazil*

**Keywords:** Mapping, Ontology, Relational Database, Mutation Test.

**Abstract:** Ontologies are structures used to represent a specific domain. One well-known method to simplify the ontology building is to extract domain concepts from a relational database. This article presents an architecture which enables an automatic mapping process from a relational database to OWL ontology. It proposes to enrich the terminology of ontology elements and it was validated with mutation tests. The architecture mapping process makes use of new and existent mapping rules and overcome lacks not previously addressed, such as the use of database logic model to eliminate duplicated elements of ontology and mapping inheritance relationships from tables and records. We stand out the structure of element mapping, which allows maintaining source-to-target traceability for verification. We validate our approach with two experiments: the first one focuses on architecture validation applying an experiment with three scenarios and the second one uses a testing engine applying a mutation test methodology to OWL ontology validation.

## 1 INTRODUCTION


In computing, an ontology is defined based on a set of concepts in which a domain of specific knowledge is modeled (Gruber, 1995). Ontologies offer advantages in their use, such as: providing an exact description and an exact vocabulary for representation and sharing of knowledge (Guarino, 1995). The process of its elaborating is a task which requires a great amount of effort (Staab and Studer, 2013; Telnarova, 2010).


Several approaches aim to convert relational databases into ontologies. Most parts of these solutions develop a mapping process from a set of rules that considering a relational database (RDB) and physical model. Michel et al. (2014), Spanos et al. (2012) and Sequeda et al. (2011) surveyed the motivations and the benefits of a mapping process from relational database to ontology, considering the challenges, and different application purposes. Although some authors use the R2RML language (Sequeda et al., 2012; Das et al., 2012; Arenas et al., 2012) and it is an important advance for the community, the R2RML language has restrictions and does


not support to record during the mapping the relationships between RDB and ontology concepts.

Concerning the creation of an ontology from scratch since an RDB, we analyzed related work as Astrova (2009); Būmans and Čerāns (2010); Cullot et al. (2007); Gherabi et al. (2012); Laclavik (2006); Li et al. (2005); Louhdi et al. (2013); Ramathilagam and Valarmathi (2013); Ren et al. (2012); Telnarova (2010); Vavliakis et al. (2010); Zhang and Li (2011) and Jain and Singh (2013). Tissot et al. (2019) describes in detail such related work mentioned above, being explained rules commonly used in the mapping process. Huve (2017) explores related work and developed experiments with Astrova (2009) work, which has made a considerable contribution to this research area. Astrova (2009) proposed QUALEG DB, a tool to deal with hierarchy, constraints, and restrictions of elements, however, it works with direct mapping. More recently, Jain and Singh (2013) compared different approaches developed to convert RDB to ontology, and they proposed adding new features to an existing framework. Nevertheless, this proposal provides a direct mapping between RDB and ontology elements, limiting the possibilities of ontology modeling.

In general, these works do not expose details about the terminology of naming ontological el-

<sup>a</sup>  <https://orcid.org/0000-0002-2038-9450>

<sup>b</sup>  <https://orcid.org/0000-0003-0832-5750>

<sup>c</sup>  <https://orcid.org/0000-0002-8922-6975>

ements, an important issue to ontology legibility. These mapping processes are quite abstract and there is no source-to-target relationship among the elements. An exception is presented by Ren et al. (2012), which append the database relation name to the column name when mapping columns. In related work, the name of database elements comes from database schema and the name of ontology elements retain the source name of database elements. When there is a type of encoding or abbreviation on the database element name are more difficult to understand the meaning of target ontology elements. This encoding is commonly declared to arrange the database elements in the source RDB schema. Another point to emphasize is the ontology validation of these mapping process, not being presented a solution that uses a testing engine, only being validated the mapping process.

Given the above context, the following motivations for the present study are: a) the existing mapping processes only perform a direct mapping of the elements and do not reuse pre-existent definitions for enriching the ontology terminology; b) the proposed solutions do not consider the information from the RDB physical and logical model at the same time; c) in face of different contributions, we identify the opportunity to consolidate the proposed functionalities of different works in a unique architecture; and d) we consider validate not only the mapping process but also the final result of generated ontology applying mutation test for owl ontology validation.

This paper is organized as follows: Section 2 presents the mapping process architecture and its set of components and mapping rules. Section 3 depicts the experiments performed in order to validate our approach and the produced OWL ontology. Finally, section 4 concludes with considerations and future work.

## 2 MAPPING ARCHITECTURE

In this section, we present our architecture to perform the mapping process between RDB-to-ontology. We developed a mapping architecture based on related work mentioned in section 1 and from observed needs. It allows store the mapping between RDB and ontology elements, offering a more complete solution than the one proposed by Cullot et al. (2007) and Būmans and Čerāns (2010). We rewrite rules mentioned in related work and we considered 3 rules proposed by Tissot et al. (2019). Our main contribution consists in the architecture modeling and in the adequacy of rules to overcome lacks not previously addressed. One of them consists in the use of database logic model rather than using uniquely

database schema or physical model. The terminology used in the logic model is more related to the real world definitions than the terminology used in the physical model, improving the semantics produced in the target ontology. Another concerning in the mapping of hierarchy relationships from tables and records is the elimination of produced duplicated ontology elements, however, maintaining the traceability between RDB and ontology elements. From these considerations, in Figure 1 we present our mapping architecture capable of overcoming all of these needs.

In the next subsection, we describe the main architecture components and second, we present a set of mapping rules handling the issues argued as our motivation.

### 2.1 Architecture Components

The architecture is divided into two parts: *External and Internal Components*. The *External components* is related to the source data to support the mapping process, being composed by the database schema, records and logic model. All data extract from the *External Component* process (being this a manual process), it is inserted into a single file, that we call as *Configuration Template*. Once the file is created, it is used as input of the *Mapping process*, being this process conducted automatically from the input of configuration template file, being this file analyzed by a set of rules, detailed in section 2.2, to perform the mappings. The input data and the mappings results are automatically stored in the *Mapping Model*, being it a metamodel prepared to produce a database schema from it and keep the database and ontology elements as well as the mapped relations. The *Generation of Ontology* process uses the data stored in *Mapping Model* containing the mapped ontology elements to produce the target OWL ontology.

### 2.2 Architecture Mapping Rules

In this section we present an set of mapping rules used to map each RDB element to the corresponding ontology element, being defined more than one rule to map each RDB element. We also considering criteria to name the ontology elements, which enables to establish conventions to be used by the community and which contribute to a greater legibility and understanding of the target ontology.

#### 2.2.1 Mapping Rules

The mapping process consider the Configuration Template to get information from tables, columns,

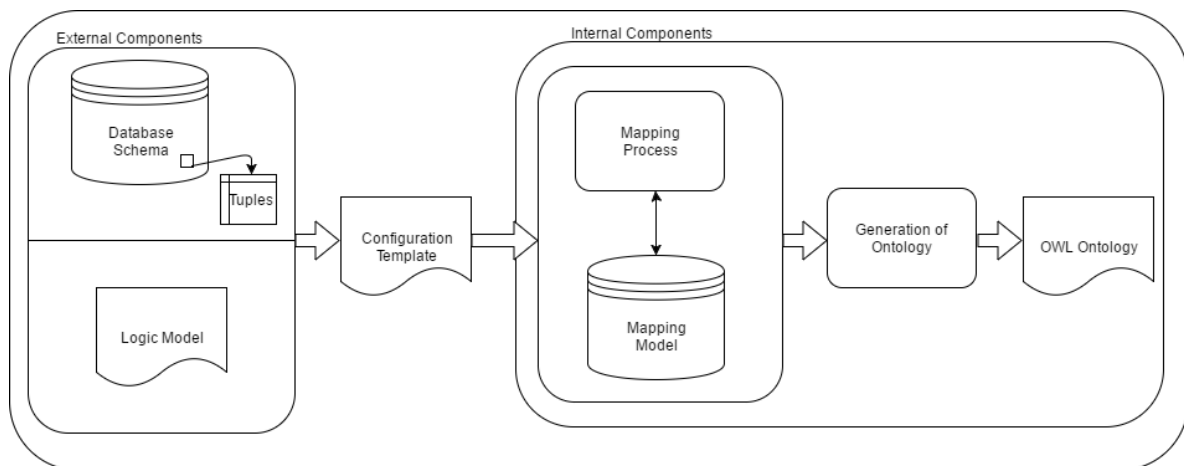


Figure 1: Architecture components.

records, check constraint and hierarchies. Thus, we separate the rules accord these database elements. Rules 1 and 2 are related to database table elements. Rules 3 and 4 are related to database columns elements. Rules 5 and 6 consider database columns elements marked as check constraint and at last, rules 9 and 10 deal with hierarchies. In Huve (2017) is formal described the rules using pseudocode and also in it is presented details of the sequence of mappings and *Configuration Template*. In the rules specification always is mentioned the logic model. However, when the logic name of database elements are not inserted, the mapping is performed considering the physical name of the database elements. The 10 mapping rules are presented below:

- Rule 1 - Mapping Non-associative Tables: each non-associative table is mapped to one ontological class, wherein the class name is composed of: a) it starts with a capital letter and b) the value which correspond to the logic table name.
- Rule 2 - Mapping Associative Tables: we map the primary key columns of associative tables to an object property (mutually inverse) for each primary key column, unless the table columns are only foreign key. The first object property name is composed of: a) it starts with fixed value *has* - with lowercase letter; and b) the corresponding value for the logic column name - it starts with a capital letter. The second object property name is composed of: a) it starts with fixed value *is* - with lowercase letter; b) the corresponding value for the logic column name - it starts with a capital letter; and c) the fixed value *of* - with a capital letter. For each object property is specified minimum cardinality, declared as inverse, and the *domain* and *range* are assigned. The *range* is specified using the created class to the foreign key of the referenced table; and *domain* is specified using the created class to the referenced table to the other foreign key which compose the associative table. We can identify in the *Mapping Model* database schema whether a non-associative table has additional columns, finding a column which is not checked as a primary key. Whether the result returns a value we apply the steps above, and together with it these steps: a) we create a class to the non-associative table, according to Rule 1; and b) we create a datatype property for each non primary key column, according to Rule 3.
- Rule 3 - Mapping Columns to Datatype Properties: Table columns that are not part of these column types (primary, unique, foreign key or check constraint) are mapped to a datatype property, at first moment, wherein the datatype property name is composed of: a) it starts with lowercase letter; and b) the value corresponding to the logic column name.
- Rule 4 - Mapping Columns to Object Properties: primary or unique key columns are mapped to two functional object properties (mutually inverse), wherein the first object property name is composed of: a) it starts with fixed value *has* - with lowercase letter; and b) the value corresponding to the logic column name - it starts with a capital letter. The second object property name is composed of: a) it starts with fixed value *is* - with lowercase letter; b) the value corresponding to the logic column name - it starts with a capital letter; and c) the fixed value *of* - with a capital letter. Each object property is declared as functional, inverse and its *range* is specified using the created class from the table column. Each object property

has minimum cardinality of restriction assigned.

- Rule 5: Mapping Records to Instances: Every tuple from tables classified as specific concepts of a domain (D) are mapped to instances. For this, the instance name is composed of: a) tuple values, whereby we recommend the value corresponding to the primary key column and the column content (or columns) - with a capital letter. In case of instance name, as we can have many values, we separate the values with underscore.
- Rule 6 - Mapping Records to Classes: A proportion of tables in the database have a small number of tuples which represent specific definitions of a particular domain. Considering classes are concrete representations of concepts, we classified tables with specific definitions of a domain as common concepts of multiple domains (C) and all tuples from these tables are mapped to classes, wherein the class name is the tuple content - it starts with a capital letter.
- Rule 7 - Mapping Check Constraint Concept: The *check constraint concept* represents a term which we use to represent a set of values defined in a check constraint. We propose to map the database check constraints and create a class to represent the set of values defined in the check constraint. For this, in the Configuration Template can be filled the check constraint concept and can be filled the set of values defined for this check constraint. For the check constraint concept we map this information to one ontological class, wherein the class name is composed of the value corresponding to the check constraint concept - with a capital letter. Every possible value of this check constraint is mapped to an instance, wherein the instance name is composed of: a) the value corresponding to the class constraint name - with lowercase letter; b) we separate the values with underscore; and c) the value corresponding to the constraint name - with a capital letter.
- Rule 8 - Mapping Check Constraint Column: In the Configuration Template, the field checked as check constraint is mapped to one object property, wherein the object property name is composed of: a) the value corresponding to the logic column name - it starts with lowercase letter. For this object property the *domain* and *range* are assigned, where the *domain* is specified using the class created from the table column; *range* is specified using the class created with the Rule 7, which represents the global concept of this set of check constraints values.
- Rule 9 - Mapping Inheritance Relationships from

Tables: For each foreign key which is equivalent to the primary key in non-associative tables, the class representing the referenced table is defined as a superclass of the class representing the non-associative table.

- Rule 10 - Mapping Table Record Hierarchy: In Rule 6 the mapping of all tuples to classes, from tables of common concepts of multiple domains (C), was proposed. These tables have relationships with another tables and to their relation, as we describe in Rule 9, for each related table we create a subclass hierarchy based on foreign keys definition. Creation of class hierarchy based on classes mapped from tuples of these tables is proposed in this rule. The table record hierarchy can be mapped when both related tables belong to tables classified as (C). We performed this mapping for non-associative tables and for tables in which the foreign key columns number is not greater than one in the referenced table.

### 3 VALIDATION

First, we present the obtained results executing the mapping process to validate the rules and the prototype. For this, we use a *Database Schema* and *Logic Model* from dental care domain. Second, we use a testing engine to apply mutation operators and executing mutation tests methodology, proposed by Porn and Peres (2017), to validate the owl ontology automatically generated by architecture.

#### 3.1 Experiment 1: Prototype Validation

We develop a tool to put into operation our architecture. Then, we evaluate our tool and check whether the ontology was correctly built, using a reasoner to infer logical consequences. For this, we consider three different scenarios: a) in the first one we used only the database schema; b) in the second one we used the database schema and logic model; and c) in the third one an analysis by a specialist was performed to remove from second scenario RDB elements that were not part of the application domain, such as information about system configuration and logs. We used a *Database Schema* (private source) from a dental care system, containing a scenario of 25 tables, 45 primary keys, 1 unique key, 19 foreign keys, 28 check constraints, 225 columns with no constraints in the first two scenarios and 74 at the third, and 696 tuples. The logic model of this database schema has 232 columns with the logical name in the first two

scenarios and 140 columns with the logical name at the third.

Each scenario was manually extracted and it was submitted to the architecture prototype. Then we performed an ontology verification in two steps: first, an analysis of mapped elements enable verifying whether the defined mapping rules were applied correctly by the *Mapping Process*; and second a verification applying in each OWL file using the reasoner Pellet<sup>1</sup>. The inferences carry out the ontology classification, the computation of inferred instances and the validation of ontology consistency. Both activities were performed using the Protégé tool (Musen, 2015).

Table 1 presents the number of generated ontology elements for each scenario (columns) and in four categories of ontology elements (lines). In the *Class* category we presented the total of *Class* and *Subclass* that were generated. The elements of the *Class* category were generated from Rules 1, 2, 6, 7, 9 and 10, as covered in the 2.2.1 section. We also presented the total number of *Object property* elements and their definitions of *Functional*, *Inverse*, *Domain*, *Range* and *Min cardinality* for Object property. The elements of the *Object property* category were generated from Rules 2, 4 and 8. In the *Datatype property* category we presented the total number of Datatype property elements and their *SubProperty*, *Domain* and *Range* definitions. The elements of the *Datatype property* category were generated from Rules 3. Finally, we presented the total number of *Instances* and *Class assertion* generated from Rules 5 and 7.

Scenarios 1 and 2 has the same number of database elements. The difference is on the name on the output elements for scenario 2 since we used the RDB logic model as additional input. However, we can observe in Table 1 a different number of ontology elements that were generated. On the one hand, the logical name can present specification which differentiates the elements, generating a greater amount, such as the elements of *Object property*, which go from 68 in the first scenario to 74 in the second scenario. On the other hand, the logical name of few elements can be the same, generating a smaller number of elements, such as the *Datatype property* elements, which go from 142 in the first scenario to 122 in the second scenario. We noticed that the naming of the elements using *Logic Model* increase the ontology legibility.

During the analysis by a specialist in the third scenario, out of the 151 columns that were excluded: 75 controlled the security of database tuples and 76 controlled operations of the system which populates data in the database. As a result of this, RDB columns with

no constraints in the third scenario is 74, resulting in a smaller amount of data properties. However, besides a domain representation with greater legibility and definitions more coherent than first and second scenarios, which produced a large amount of information not related to the domain.

The prototype validation was performed in each of the architecture components. We then executed the Pellet reasoner and there was no identification of inference errors in target ontologies. Other approaches Astrova (2009); Jain and Singh (2013) performed only 1:1 mapping, but our architecture can map 1:N. Due to this structure,  $n$  database elements can be mapped to 1 element in the ontology, and vice versa. A practical example is the representation of a property generated from a primary key column. This column may exist in different tables, being represented by a foreign key column. We understand that it does not make sense to represent  $n$  times the same element that represents the same concept, but inserted in different domains.

This experiment allowed to analyze the coherence of the defined rules and regarding the well functioning of the proposed architecture. Through this, we observed a greater clarity in the name of ontology elements in the second scenario compared to the first scenario, highlighting the obtained benefit from the use of the logic model, which increases the ontology legibility.

### 3.2 Experiment 2: OWL Ontology Validation using Mutation Test

Mutation testing is a technique for software testing to evaluate the quality of a set of test cases (DeMillo et al., 1978; Jia and Harman, 2011). This technique has been shown effective in detecting faults (DeMillo et al., 1978; Budd, 1980) through systems evaluation and their sets of tests in various domains (Jia and Harman, 2011). Recent studies (Porn and Peres, 2014, 2017; Bartolini, 2016) adopted Mutation testing to ontology evaluation.

The ontology evaluation is an important process of ontology engineering that allows identifying whether the objectives have been reached and whether there are no occurrences of failures. From this, we applied the mutation test for evaluating the ontologies obtained through the mapping process aiming to identify the existence of faults. Based on this, we performed the mutation test for ontologies in this experiment using the generated ontologies of the three scenarios of Experiment 1, as presented in Table 1.

We performed the mutation test for ontologies as proposed by Porn and Peres (2017): 1) Mutation op-

<sup>1</sup><https://www.w3.org/2001/sw/wiki/Pellet>

Table 1: Number of ontology elements.

Number of elements		First scenario	Second scenario	Third scenario
Ontology elements				
Class		109	109	108
	SubClass	121	121	121
Object property		68	74	73
	Functional	50	54	54
	Inverse	30	32	32
	Domain	105	107	107
	Range	84	86	85
	Min cardinality	76	76	76
Datatype property		142	122	64
	SubProperty	5	5	5
	Domain	141	128	67
	Range	142	128	66
Instances		668	668	668
	Class assertion	668	668	668

erator selection; 2) Mutants generation; 3) Test data generation; 4) Original ontology execution; 5) Mutants ontology execution; and 6) Result analysis. We used 12 mutation operators, as detailed below:

- ACSD: it changes to DisjointWith the definition of an axiom defined as SubClassOf;
- AEDN: it adds the operator “not” in the definition of an axiom;
- CEUO: it removes the operator “OR” in the definition of an axiom;
- CIS: it changes to SubClassOf the definition of an individual;
- CUC: it changes the superclass of a subclass;
- PDD: it changes the properties domain to a subclass of the domain class;
- PDUP: it changes the properties domain to a superclass of the domain class;
- ACMiEx: it replaces the “minCardinality” operator with the “Cardinality”;
- ACMiMA: it replaces the “minCardinality” operator with the “maxCardinality”;
- PDU: it removes the domain definition in properties;
- PRU: it removes the range definition in properties;
- PDC: it converts a primitive class to a defined class.

As presented in Porn and Peres (2017), the test data set used was generated from each existing Descriptive Logic (DL) constraint of the ontology under test, and from each mutated DL constraint.

Table 2 presents the results of mutation test for each scenario. In the first scenario (Scen. 1) 2846 mutants were obtained to the 12 mutation operators, 941 mutants were killed with the test data used and 1905 mutants remained alive. Based on these results it was possible to obtain a mutation score of 33%.

In the second scenario (Scen. 2) the obtained results were similar to the first scenario, because there were few changes between the ontologies from each scenario. In it, 2791 mutants were obtained, 941 killed and 1850 mutants that remained alive, thus obtaining a mutation score of 34%.

Table 1 presents a smaller number of ontology elements in the third scenario than the first two scenarios. This smaller amount of ontology elements generates less mutants during the mutation test execution. Thus, in the third scenario (Scen. 3), 2136 mutants were generated, with 898 killed and 1238 remaining alive, resulting in a mutation score of 42%.

This analysis made it possible to identify a lack of rules for the creation of logical axioms and associations between classes and individuals into ontologies. However, we do not consider this occurrence as a type of fault existing in the ontology, but as implementations to be made in the mapping process. Therefore, for the mutation score calculation, we consider the quotient between the number of killed mutants and the remainder between the number of generated mutants and the number of equivalent mutants (DeMillo et al., 1978). In this case, we obtained a mutation score of 100% in each scenario, thus validating the set of test data used and the absence of faults in the tested ontologies.

All of the mutants shown in Table 2 that remained alive have been changed in the definitions of domain

Table 2: Mutation test results.

Mutation Operator	Mutants			Kill			Alive		
	Scen. 1	Scen. 2	Scen. 3	Scen. 1	Scen. 2	Scen. 3	Scen. 1	Scen. 2	Scen. 3
ACSD	121	121	121	121	121	121	0	0	0
AEDN	109	102	21	0	0	0	109	102	21
CEUO	109	102	21	0	0	0	109	102	21
CIS	686	686	668	686	686	668	0	0	0
CUC	101	101	101	15	15	15	86	86	86
PDD	921	889	642	0	0	0	921	889	642
PDUP	168	159	124	0	0	0	168	159	124
ACMiEx	126	126	76	0	0	0	126	126	76
ACMiMa	126	126	76	0	0	0	126	126	76
PDU	107	107	107	0	0	0	107	107	107
PRU	153	153	85	0	0	0	153	153	85
PDC	119	119	94	119	119	94	0	0	0
TOTAL	2846	2791	2136	941	941	898	1905	1850	1238
Mutation Score							0,33	0,34	0,42

and range of object and data properties. As in none of the 3 scenarios the object and data properties were used in the definition and representation of classes, all the mutants that remained alive were defined as equivalent, due to the fact that these mutations did not have an effect on the ontologies. The criteria used to define the mutants as equivalent must it is because there are no rules in the mapping process for the creation of descriptive logical axioms that define the ontology conceptual representation.

## 4 CONCLUSIONS

We have presented an approach for mapping relational databases to ontologies using database schema and logical data model. The use of the Logic Model contributes to the generation of a more legible ontology. The Mapping Model contributes to the mapping traceability and the elimination of duplicate elements. Our solution takes stock of previous works and it provides new rules, handling cases uncovered by the literature and collaborating in a significant way for adequate ontology modeling.

We validated our approach with real-world scenarios. We evaluate the target ontology from our prototype applying mutation test technique. This methodology simulates possible errors which can occur in OWL ontologies. We did not find faults in the target ontologies, but our evaluation method identified the absence of rules in the mapping process for the creation of logical descriptions and associations between classes and individuals into ontologies.

As future work, we aim to investigate and propose new mapping rules for ontology build and consider-

ing new mutation operators as also other techniques for ontology evaluation. We also intend to work in the creation of descriptive logical axioms to define the conceptual ontology representation. The architecture is generic enough and could be applied in the backward scenario, using the mapping for bidirectional transformation and traceability on querying the relational database through the ontology concepts.

## ACKNOWLEDGEMENTS

We thank the partial support provided from CAPES and Department of Informatics of Federal University of Paraná. Leticia M Peres has a grant of PET/MEC. Cristiane A G Huve was supported by Uninter. This work was conducted using Protégé resource, which is supported by grant GM10331601 from the National Inst. of General Medical Sciences of the US National Inst. of Health.

## REFERENCES

- Arenas, M., Bertails, A., Prud'hommeaux, E., and Sequeda, J. (2012). A direct mapping of relational data to rdf. <http://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927>. W3C Recommendation working draft 27 September 2012.
- Astrova, I. (2009). Rules for mapping sql relational databases to owl ontologies. In *Metadata and Semantics*, pages 415–424. Springer.
- Bartolini, C. (2016). Mutation owls: semantic mutation testing for ontologies. In *Proceedings of the International Workshop on domain specific Model-based ap-*

- proaches to verification and validation, pages 43–53, Rome - Italy.
- Budd, T. A. (1980). *Mutation Analysis of Program Test Data*. PhD thesis, Yale University, Yale, NH - USA.
- Būmans, G. and Čerāns, K. (2010). Rdb2owl: A practical approach for transforming rdb data into rdf/owl. In *Proceedings of the 6<sup>th</sup> Int. Conf. on Semantic Systems*, pages 1–3, New York, NY, USA.
- Cullot, N., Ghawi, R., and Yétongnon, K. (2007). Db2owl: A tool for automatic database-to-ontology mapping. In *Proceedings of the 15<sup>th</sup> Italian Symposium on Advanced Database Systems - SEBD*, pages 491–494, Torre Canne - Italy.
- Das, S., Sundara, S., and Cyganiak, R. (2012). R2rml: Rdb to rdf mapping language. [www.w3.org/TR/r2rml](http://www.w3.org/TR/r2rml). W3C Recommendation 27 September 2012.
- DeMillo, R. A., Lipton, R. J., and Sayward, F. G. (1978). Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41.
- Gherabi, N., Addakiri, K., and Bahaj, M. (2012). Mapping relational database into owl structure with data semantic preservation. *Int. Journal of Computer Sci. and Inform. Security - IJCSIS*, 10(1):42–47.
- Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *Int. Journal of human-computer studies*, 43(5-6):907–928.
- Guarino, N. (1995). Formal ontology, conceptual analysis and knowledge representation. *Int. Journal of human-computer studies*, 43(5):625–640.
- Huve, C. A. G. (2017). An architecture for mapping relational database to ontology. Master's thesis.
- Jain, V. and Singh, M. (2013). A framework to convert relational database to ontology for knowledge database in semantic web. *Int. Journal of Scientific & Technology Research*, 2:9–12.
- Jia, Y. and Harman, M. (2011). An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678.
- Laclavik, M. (2006). Rdb2onto: Relational database data to ontology individuals mapping. *Tools for Acquisition, Organisation and Presenting of Inform. and Knowledge*, pages 86–89.
- Li, M., Du, X., and Wang, S. (2005). A semi-automatic ontology acquisition method for the semantic web. In *Advances in Web-Age Inform. Management*, pages 209–220. Springer.
- Louhdi, M. R. C., Behja, H., and Alaoui, S. O. E. (2013). Transformation rules for building owl ontologies from relational databases. *Computer Sci. & Inform. Technology (CS & IT)*, 3:271–283.
- Michel, F., Montagnat, J., and Faron-Zucker, C. (2014). A survey of rdb to rdf translation approaches and tools. Technical Report ISRN I3S/RR 2013-04-FR, Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis, Université Nice, Sophia Antipolis, France. Research Report.
- Musen, M. A. (2015). The protégé project: a look back and a look forward. *AI Matters. Association of Computing Machinery Specific Interest Group in Artificial Intelligence*, 1(4):4–12.
- Porn, A. M. and Peres, L. M. (2014). Mutation test to owl ontologies. In *13<sup>th</sup> Int. Conf. on WWW/Internet - ICWI*, pages 123–130, Porto - Portugal.
- Porn, A. M. and Peres, L. M. (2017). Semantic mutation test to owl ontologies. In *19<sup>th</sup> Int. Conf. on Enterprise Inform. Systems - ICEIS*, volume 2, pages 434–431, Porto - Portugal.
- Ramathilagam, C. and Valarmathi, M. (2013). A framework for owl dl based ontology construction from relational database using mapping and semantic rules. *Int. Journal of Computer Applications*, 76(17):31–37.
- Ren, Y., Jiang, L., Bu, F., and Cai, H. (2012). Rules and implementation for generating ontology from relational database. In *2<sup>th</sup> Int. Conf. on Cloud and Green Computing - CGC*, pages 237–244. IEEE.
- Sequeda, J. F., Arenas, M., and Miranker, D. P. (2012). On directly mapping relational databases to rdf and owl. In *Proceedings of the 21<sup>st</sup> Int. Conf. on World Wide Web*, pages 649–658. ACM.
- Sequeda, J. F., Tirmizi, S. H., Corcho, O., and Miranker, D. P. (2011). Survey of directly mapping sql databases to the semantic web. *The Knowledge Engineering Review*, 26(4):445–486.
- Spanos, D.-E., Stavrou, P., and Mitrou, N. (2012). Bringing relational databases into the semantic web: A survey. *Semantic Web*, 3(2):169–209.
- Staab, S. and Studer, R. (2013). *Handbook on ontologies*. Springer Sci. & Business Media.
- Telnarova, Z. (2010). Relational database as a source of ontology creation. In *International Multiconference on Computer Sci. and Inform. Technology - IMCSIT*, pages 135–139.
- Tissot, H., Huve, C. A. G., Mara, P. L., and Del Fabro, M. D. (2019). Exploring logical and hierarchical information to map relational databases into ontologies. *Int. Journal of Metadata, Semantics and Ontologies*, (ISSN 1744-2621 - to appear).
- Vavliakis, K. N., Grollios, T. K., and Mitkas, P. A. (2010). Rdote - transforming relational databases into semantic web data. In Polleres, A. and Chen, H., editors, *9<sup>th</sup> International Semantic Web Conference - ISWC Posters&Demos*, volume 658 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- Zhang, L. and Li, J. (2011). Automatic generation of ontology based on database. *Journal of Computational Inform. Systems*, 7:4:1148–1154.