Environment. Technology. Resources, Rezekne, Latvia
Proceedings of the 10th International Scientific and Practical Conference. Volume III, 71-75

Storing an OWL 2 Ontology in a Relational Database Structure

Henrihs Gorskis, Arkady Borisov
Riga Technical University,
1 Kalku Str., Riga LV-1658, Latvia

Abstract. **This paper examines the possibility of storing OWL 2 based ontology information in a classical relational database and reviews some existing methods for ontology databases. In most cases a database is a fitting solution for storing and sharing information among systems, clients or agents. Similarly, in order to make domain ontology information more accessible to systems, in a comparable way, it can be stored and provided in a database form. As of today, there is no consensus on a specific ontology database structure. The main focus of this paper is specifically on OWL 2 as a basis for the description of ontology centric information in a database. The Web Ontology Language OWL 2 is a language for describing ontology information for the Semantic Web. As such it consists of a list of reserved words and grammatical rules for defining many parts of ontology knowledge. Based on this language specification this paper examines the possibility of storing information in a relational database for the description of domain ontology information. By creating a database structure based on OWL2 it is feasible to obtain an approach to storing information about the domain ontology in an utilizable way, by using its descriptive abilities. Nowadays multiple approaches to storing ontology information and OWL in databases exist; most of them are based on storing RDF data or provide persistence for specific OWL software libraries. The examination of the existing approaches provided in this paper, shows how they differ from the goal of obtaining a general, more easily usable and less software library specific database for domain ontology centric information. This paper describes a version of a simple relational database capable of holding and providing ontology knowledge on demand, which can be implemented on a database management system of choice.**

Keywords: **ontology, OWL2, relational database.**

I INTRODUCTION

Ontology knowledge is a powerful tool to share, describe and classify information about a given domain. The ontology describes concepts important to a domain. It does it by naming classes, individuals and relations and describing how these ontology elements interact with each other. By using reasoning on this information, new relationships between concepts and individuals can emerge, and individuals can be classified by reasoning about their attributes. The ontology knowledge can be provided in many different forms and using many different languages or notations to describe the information. In its most basic form, ontology knowledge is stored in a file on a computer. In order to use the ontology more easily, it would be desirable to access it with the same ease as a database. There are many approaches to storing ontology or similar information in a database. Some databases are structured around the information it contains and how the information is used; other approaches store the ontology in its most basic form in RDF triplets, and still other methods include storing API specific data structures in a database for persistence of these variable objects. This paper shows that having a natural and common structured relational database for storing ontology information can be useful for many applications. A general and common database is accessible to many different software applications or agents and requires only the understanding of the database structure by these agents. This makes it possible for the agents or software applications to be developed separately instead of requiring them to use the same API or be written in the same programming language. In order to store ontology knowledge, it is necessary to understand the kind of information stored in modern ontology modelling approaches.

II ONTOLOGY KNOWLEDGE

At its core any ontology describes concepts, individuals and properties and uses structures in order to convey information about elements of the domain. Ontology information can be described using many different notations and approaches. In order to be able to use the most common and full approach, the OWL 2 standard will be used as an ideal. OWL 2 is the web ontology language version 2 for the Semantic Web with formally defined meaning [1]. It provides many useful keywords to frame and describe knowledge. Using these keywords and structures makes it possible

ISSN 1691-5402

DOI: http://dx.doi.org/10.17770/etr2015vol3.168

to describe a doming ontology and the knowledge contained in it. First it provides a way to assert the association of certain pieces of information to classes, individuals or properties. This is done using the "Declaration" operator followed by "Named Individual", "Class", "Object Property", "Data Property" or "Data type" and the name of the entity being associated with one of these types. Next, the operator "Class Assertion" is used to classify an individual, by stating the individual's name and the class it belongs to. The operators "Sub Class Of" and "Sub Object Property Of" are used to create a hierarchy of classes and properties to form taxonomy. Ontology building requires the expert to follow the rules of how an ontology is structured. In order to create a correct taxonomy, any higher level classes must be more abstract than their lower level classes. Individuals belonging to a class must be the most distinct and unique element of the ontology. Further OWL 2 provides a way to define expressly equal or distinct classes and individuals using the operators "Equivalent Classes", "Disjoint Classes", "Different Individuals" and "Same Individual". On the lower level individuals can be described by using their relationship with other individuals. This is done using the "Object Property Assertion" operator and stating the name of the object property and the second individual in this relationship. The operator "Negative Object Property Assertion" can be used to express a distinct lack of a specific relationship between two individuals. For implying a class to members of a relationship, the operators "Object Property Domain" and "Object Property Range" are used. By defining these attributes of the object property it can be reasoned that any individual with this property is of the specified class. OWL 2 has many more operators for defining ontology knowledge. There are very similar operators for defining data properties between individuals and data types, for example "Data Property Assertion" and "Negative Data Property Assertion". Other operators are used for defining complex classes. The operators "Object Intersection Of", "Object Union Of" and "Object Complement Of" are used to define unnamed classes which arise from the interaction of other classes. A whole array of operators is used in order to define the specific attributes of a property; these include: "Symmetric Object Property"," Asymmetric Object Property" ," Reflexive Object Property", "Irreflexive Object Property", "Functional Object Property" , "Inverse Functional Object Property" and "Transitive Object Property".

All the above and other operators are used to meticulously define every single detail regarding the knowledge contained in the ontology about each of the important domain concepts. Using the logic associated with these operators even more information about the concepts can be derived from the specifically expressed definitions. However, not every application using ontology knowledge is required to know every detail about the domain, and not every agent will use reasoning on the ontology. Some software agents are completely content using ontology concepts simply as a dictionary. This makes it reasonable to store the ontology separately from the software applications which end up using it.

III EXISTING APPROACHES

There already exist several approaches to storing and recalling ontology information. Protégé is a very popular tool for creating ontology models. However, it does not provide a solution for accessing the ontology externally. It mainly provides the means of saving the ontology in a file or source so that another program can use the ontology file. Historically, in previous versions of the software many attempts have been made to create a solution for accessing and storing ontology models created in Protégé in a database [2] - [4].

Another popular tool for working with ontology is Apache Jena. It is a programming library for JAVA. Besides many other functions it offers two ways of storing ontology data in databases. Jena comes packaged with server software called "Fuseki". Fuseki is a SPARQL server. It stores ontology information in its own internal data structure and provides access to the data by sending SPARQL queries to it. The second solution Jena offers is called TDB (), which is a native high performance triple store [5]. This approach creates a triplet-based database table in a database of the user's chaise. Apache Jena does provide means to work with ontology structures, however, underneath it relies mostly on RDF data, with the ontology being higher level abstraction of it.

It is worth mentioning another software tool for ontology persistence called OWLDB [6]. OWLDB is a database backend for the OWL API. It provides persistence for OWL API data structures. This means that by using this tool objects created by the OWL API can be stored and recalled from a database. This, however, means that only application written by using both these tools can use this function. The reason for creating this backend, as stated by the authors, was to make the use of ontology information simpler, based directly on OWL and not reliant on previous RDF structures. The approach presented in this paper is in agreement with this sentiment. In order to work with the capabilities of the ontology itself it is not necessary to implement a backwards compatibility with pre-existing approaches. Unfortunately, this project is not being continued anymore.

There also exist many different database types, all of which have their own advantages [7]. The reason a classical relational database was chosen over more specialized ones, was to make the resulting database more accessible to a wider variety of possible uses and software applications.

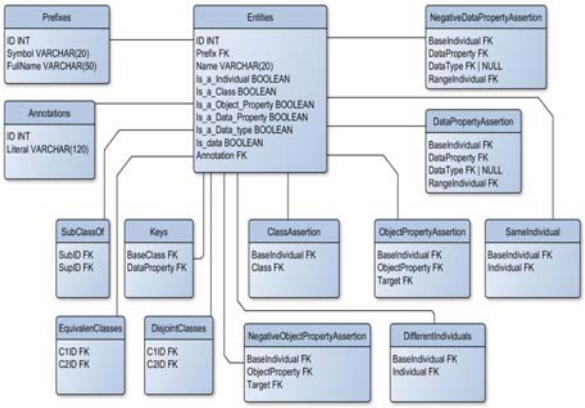

Fig. 2. Core tables

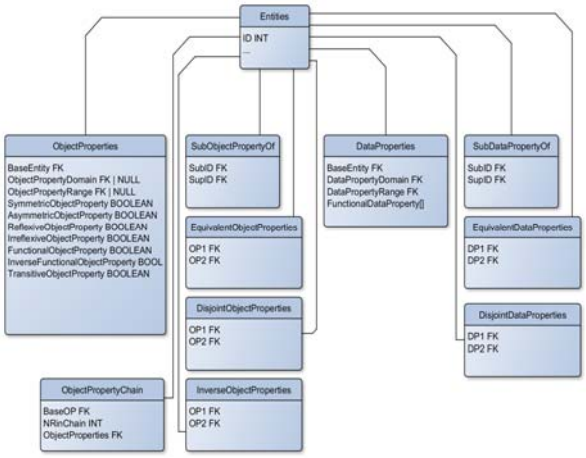

Fig. 1. Property tables

IV OWL 2 BASED DATABASE

This paper proposes a database structure based directly on OWL 2 and uses simple relations to describe ontology knowledge. At the centre of the proposed database architecture a main table is located which holds every element in the ontology (Fig. 1). In this context an element is any piece of information which describes some idea in the ontology. Without additional information such an element can potentially be a class, individual, property, literal, data type and so on. Additionally to the main table many other tables exist, each named after operators in the OWL2 specification. The main table shall hold a unique identifier, the elements name and a list of true or false Boolean variables to describe the element further. The unique identifier will be used in other tables to reference the unique element to provide additional information to it or to use it for the description of other elements of the ontology. This allows the database to reuse the named individual as many times as needed. The list of true or false operators provides hints to the type of element in order to make it easier to find additional information about the element. For example, if the element is hinted to be an individual or a class it is reasonable to search the "class assertion" table for more information about the element as to which classes the element belongs to, or which elements are individuals of this class element, depending on the type of the element. Having a list of hints requires it to be updated in addition when new entries are being made into other tables. However, at the same time, the list of hints simplifies searching for additional information immensely. Without this list of hints, it would be necessary to search the entire database and every table in it to obtain the full picture about every element. Additionally the entity table holds a reference to a prefix entry in the prefix table. By separating prefixes from the entity they can be reused. If every entity in the ontology has the same prefix, there will be only one prefix in the prefix table used by every entity. Finally, the entity holds a reference to an annotation in case it exists. The annotation provides additional information to a human user. Every other table in the database references one or more entities. For some tables the order of referencing entities is important, for other it is not. For example, the table "Sub Class Of" contains information about the hierarchy of classes. The column "Sub ID" holds references to the class which is the sub class in the hierarchy, while the column "Sup ID" holds the references to the class above it. So the order is important, and the naming of the columns reflects this fact. In contrast, the table "Equivalent Classes" has the columns "C1 ID" and "C2 ID". The order of the references is not important since the table describes equivalency. This must be taken into account during searches on the tables. If, for example, one wishes to find the equivalent classes to an entity, one must search for the entity's identifier in both columns, for it can have been placed in the first as well as in the second position during the creation of the ontology. Some tables reference many more entities. The table "Data Property Assertion" holds references to the entity of type individual, which has been given this attribute, the reference to the entity which describes the data property, the reference to a data type entity and the entity representing the data.

Some possible tables have been omitted from this database structure since the data they would have contained are more useful and tied directly to an entity. Besides the class, individual and other assertions being represented directly in the entity table, assertions about object property characteristics have also been added directly to the "Object property" table (Fig. 2). This table references a base entity representing this object property. In addition it can reference entities representing classes, to provide information about the domain and range of this object property. All of the possible characteristics of an

object property are provided as Boolean variables in the table.

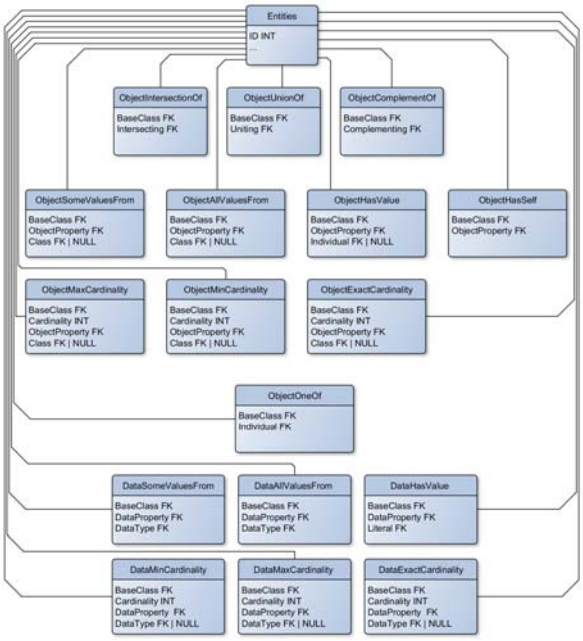

Fig. 3. Complex class tables

All the aforementioned tables provided pieces of information about an entity in a disconnected, but self-describing way. However, there are tables which hold more complex information and require all pieces of it to be obtained, before a conclusion can be reached. For example, the table "Object Intersection Of" (Fig. 3) provides parts of a description of a complex class. A complex class arises from the interaction of multiple other classes. In the case of intersection, a complex class is created, when the combination of other classes creates a new conceptual class. Since an unspecified number of classes can be involved in this interaction, a static table containing all required references cannot be created. Therefore the table "Object Intersection Of" holds only one reference to a participating class at a time. The reference to the base class is a reference to an entity describing the complex class itself, and the column "Intersecting" holds the reference to one of the intersecting classes. This means, all table entries concerning the complex class (having the same base class id) must be obtained, before it is known which classes are involved in the intersection. The same rule applies to all tables describing complex classes.

Finally, the database also holds a table capable of describing various datatypes for data properties (Fig. 4). Datatypes behave like classes with the difference that they do not have individuals, but instead govern literal data. Literals, datatypes and the data themselves are also entities in this database. Similar to complex object classes, complex data types also require multiple entries into tables. Just like the "Object Intersection Of" database table, the "Data Intersection Of" table provides information about all data types whose intersection form a new complex data type.

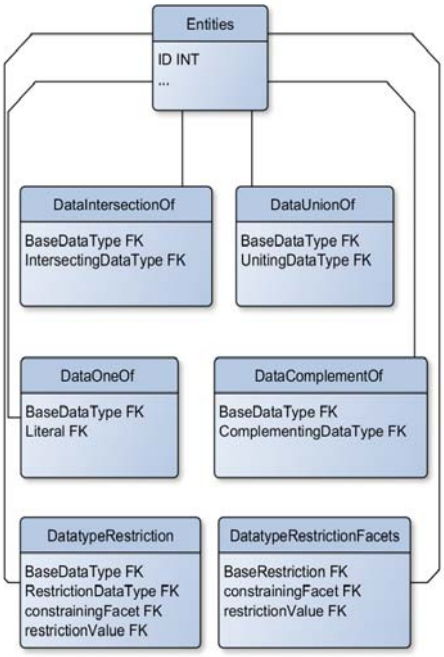

Fig. 4. Datatype tables

Access to the knowledge in the database is very simple. In the case, when a user or software agent is looking for a specific concept, the main entity table is searched for it by name. There can be two entities with the same name in the table. In such a case the related prefix can be consulted. If a prefix was specified within the search parameters, the unique entity can be found. Once the entity is found, it can be provided to the user. Some systems may be satisfied at this stage with the obtained information. Other systems may choose to obtain further information about the found entity. Based on the entity's characteristics further tables can be polled for additional information based on the identification of the entity. In most cases the result of searching other tables will be a list of identifications referencing other related entities. Again, depending on the nature of the obtained information some systems may choose what connected information must be researched further. To do this, the main table is searched again based on the identifications and other entities and their name and characteristics are obtained. This process is repeated until the user or system has obtained all the required information about the original and related entities.

This database structure can be implemented in any standard database system. For example, the creation script of the main entity table for MYSQL looks as follows:

```
DROP TABLE IF EXISTS
      'owl2db'.'entities' ;
CREATE TABLE IF NOT EXISTS
      'owl2db'.'entities' (
  'Id' INT(11) NOT NULL
      AUTO_INCREMENT ,
  'Prefix_fk' INT(11) NOT NULL,
  'Name' CHAR (50),
  'Is_a_Individual' Boolean NOT
      NULL,
  'Is_a_Class' Boolean NOT NULL,
  'Is_a_Object_Property' Boolean NOT
      NULL,
  'Is_a_Data_Property' Boolean NOT
      NULL,
  'Is_a_Data_type' Boolean NOT NULL,
  'Is_data' Boolean NOT NULL,
  'Annotation_fk' INT(11) NOT NULL,
  PRIMARY KEY ('Id'),
  CONSTRAINT FOREIGN KEY
      ('Prefix_fk') REFERENCES
      'prefixes' ('Id') ON DELETE
      CASCADE ON UPDATE CASCADE,
  CONSTRAINT FOREIGN KEY
      ('Annotation_fk') REFERENCES
      'annotations' ('Id') ON DELETE
      CASCADE ON UPDATE CASCADE
)
ENGINE = InnoDB
  AUTO_INCREMENT = 1;
```

V CONCLUSION

This paper described a novel approach to storing specifically ontologies based on OWL 2 in a simple and directly accessible database. Since the structure was based on the OWL 2 language, its capabilities for defining and describing ontology knowledge must be comparable. However, there are some potential downsides resulted from using such a database structure. The main entity table can become very large in size. This can slow down access to the ontology knowledge since every other quarry is using this table to determine the name and characteristics of an entity based on its identifier. This is amplified by storing not only named, but also unnamed entities in this table. Every complex class which does not necessarily have been given a name must still have an entity object in order to define every attribute of the complex class. Existing datatypes like "xsd:integer" also must have entity object in order to maintain the integrity and consistency of the database structure. All these factors contribute to a very large list of entities.

The proposed database does not guarantee or verify the reasonability of the knowledge described in it. There is no mechanism to prevent any entity from being a class, individual and property and other at the same time. It is the ontology expert's responsibility for the ontology knowledge to make sense. However, in some cases it can be of use to have elastic and multi-purpose element of the ontology. A concept being several things at the same time is not automatically a logical fallacy, as long as the resulting ontology is usable for its stated purpose. Another downside to storing ontology knowledge in a database is the lack of a reasoning mechanics. Any such functionality must be provided by an additional software solution connecting to the database, obtaining its contents and adding any new conclusions to the database. However, some basic reasoning functionality like an automatic classification of entities can be added directly to the database using triggers or scripting languages. This is dependent on the database management software capability.

At this point it is unclear how this database will or should handle imports from other ontologies. The prefix table provides some functionality to describe the origin of an entity. If an entity has a prefix from an outside source, further quarrying to other knowledge sources may be required.

Because of the large number of tables, obtaining all information can be difficult. At this point, knowledge obtaining must be performed using a dialog approach, which includes obtaining information piece by piece. This has positive as well as negative aspects to it. On the one hand, this allows the user of the database to obtain only those pieces of information which are important. By having a dialog, the user has a choice at every step. On the other hand, as a negative aspect, this means that in cases where all the related information has to be obtained, the process is slowed down immensely. As future work, an approach to obtaining the information more easily and preferably handled on the server side can be researched as well as an implementation of a quarrying language like SPARQL.

VI REFERENCES

[1] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider and S. Rudolph, *OWL 2 web ontology language primer. W3C recommendation*. 2009 [Online]. Available: http://www.w3.org/TR/owl2-primer/ [Accessed March 15, 2015]

[2] T. Redmond, "An Open Source Database Backend for the OWL API and Protege 4". In *OWLED*, Vol. 614, 2010.

[3] M. Horridge and S. Bechhofer, "The owl api: A java api for owl ontologies". *Semantic Web*, 2(1), 11-21, 2011.

[4] P. Cudré-Mauroux, I. Enchev, S. Fundatureanu, P. Groth, A. Haque, A. Harth and M. Wylot, "NoSQL databases for RDF: an empirical evaluation", in *The Semantic Web–ISWC* , Springer Berlin Heidelberg, 2013, pp. 310-325.

[5] M. Voigt, A. Mitschick and J. Schulz, "Yet Another Triple Store Benchmark? Practical Experiences with Real-World Data", in SDA 2012, September, pp. 85-94.

[6] J. Henss, J.Kleb, S. Grimm and J.Bock, *A Database Backend for OWL. 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2009)*, Chantilly, Virginia, USA,October 2009.

[7] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y.Chen and D. Wilkins, (2010, April). *A comparison of a graph database and a relational database: a data provenance perspective. In Proceedings of the 48th Annual Southeast Regional Conference* , p. 42. ACM.