PROCEEDINGS OF SCIENCE

A Storage Method of Ontology Based on Graph Database

Hui Zhang[1]
Beijing Information Science and Technology University, Beijing, 100101, China
E-mail: zhanghui241123@sina.com

Xia Hou[2]
Beijing Information Science and Technology University, Beijing, 100101, China
E-mail: houxia@bistu.edu.cn

Ning Li[3]
Beijing Information Science and Technology University, Beijing, 100101, China
E-mail: ningli.ok@163.com

Many problems may occur when storing ontology into the relational database, such as semantic information loss, information redundancy and so on. Graph database has a natural graph structure, which can relieve the above problems. So far many articles have only touched upon the RDF and graph database but not on the common ontology. A method is put forward in this paper used for storing ontology into the graph database. In the article, a general model of mapping ontology to graph is presented and its algorithm is given. A prototype system is implemented to map an ontology to a graph and store it into a graph-database automatically. The test results based on the system illustrate the mapping model and the algorithm are universal for the ontology. And many queries can be done in the graph-database which are similar to those in ontology.

ISCC2015
18-19, December, 2015
Guangzhou, China

[1]Corresponding Author
[2]This study is supported by Beijing Higher Introduction and Cultivation of High Level Talents Project.(CIT&TCD201504056).
[3]This study is supported by Beijing Municipal University Innovation Team Building and teacher occupation development program.(IDHT20130519).

PoS(ISCC2015)021

1.Introduction

With the rapid development of modern technology, knowledge from different areas is increasing quickly. In order to make the knowledge become shared and reusable, the study on the Semantic Web is very urgent. Ontology, as an important part in Semantic Web, often becomes the focus of related researches. As we know, an effective way of storing ontology affects not only the efficiency of the system, but also its overall performance[1].

At present, There are three main ontology storage methods. (1) File storage: A number of editors of the ontology, for example Onto and protégéall worke based on files. The advantages of this way are that it is easy to realize and understand, while their disadvantages are it's making contents redundant when storing the large scale ontology. (2) Memory storage: It means to store ontology into main memory and then do all operations above it. For example, OWLim and OWLJessKB, which are both systems based on the main memory[2-3]. The efficiency of this method is high. However, a big problem is the limitation of main memory. (3) Relational database storage: The basic idea of this method is storing ontology into one or more common tables. According to the differences of mapping principles, it can be roughly divided into hierarchical storage , vertical storage , analytical storage and mixed storage [4-5]. The shortage of those ways is their mapping structures' mismatch, which means some semantic information will be lost or redundant information may be produced[6].

In order to solve these problems, some researchers focus on the ontology and graph database,but many mainly discuss on RDF instead of common ontology[7-8]. In this paper, a storage method of ontology based on graph database is presented. The mapping rules from ontology to graph database are given, and its algorithm is proposed.

2.Mapping Rules from Ontology to Graph Database

2.1 Related Concepts

(1) Ontology

Ontology provides a standard for things to become shared and general. There are many definitions about ontology. This paper uses the idea put forward by Perez [9], who classifies the ontology into five elements as Definition1.

Definition 1: define an ontology *O=(C, R, F, A, I)*. In this quintuple, *C* represents the collections of classes, *R* is on behalf of collections of relations, *F* means the collections of functions, *A* stands for the set of axioms and *I* shows the collection of instances.

Ontology could be described by many languages, such as XML, RDF, RDFS, DAML+ OIL, and so on. OWL has many advantages compared with others and is used widely now. The experiments in this paper are all based on the OWL ontology.

(2) Graph database

Graph database is a kind of databases based on the graph structure. The operations in a graph-database, such as adding, deleting, update and query are all traversing a graph. Partner and Vukotic mentioned that graph databases have more advantages than relational databases in the treatment of the associated data[10]. The basic formal description of a graph database is a directed graph defined in Definition 2. A typical figure is shown in Figure 1.

Definition 2: define a graph database as a directed graph *G=(V, E),* where *V* represents the set of nodes and *E* is the set of directed edges which are the relation among nodes. And each node and relation can have many properties.

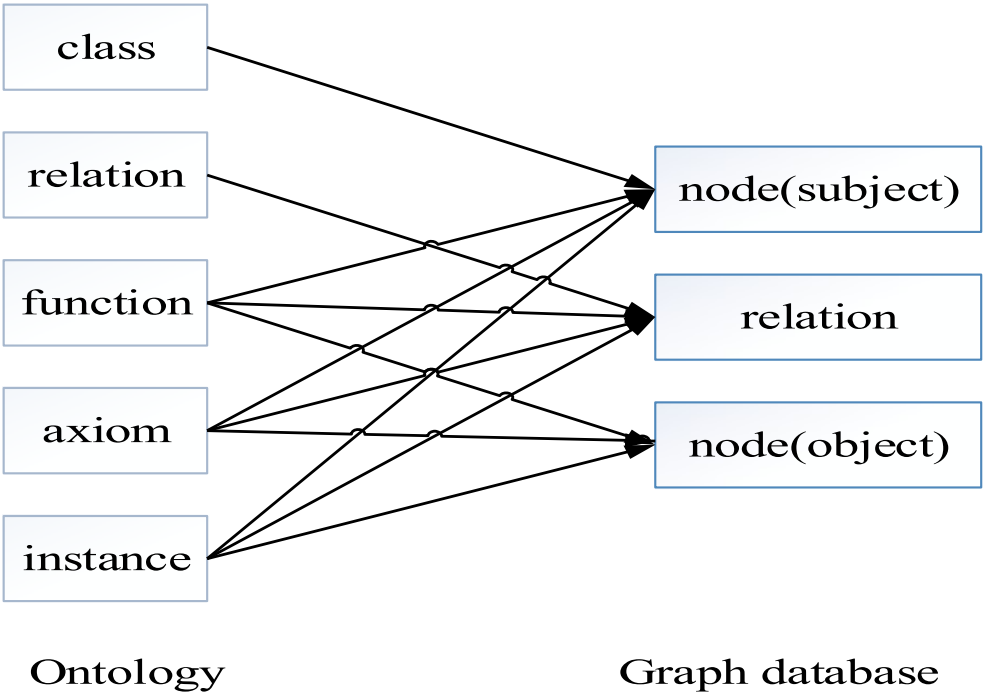

Figure 1: Graph Database Structure Unit

2.2 General Mapping Rules

In fact, the topology model of an ontology is a graph.,therefore storing an onlotogy in a graph database is feasible. Here are some general mapping rules to map $O=(C, R, F, A, I)$ into a graph-database $G=(V, E)$. Define an operation $\rightarrow$, which maps an element in O to an element in G.

(1) the mapping rule of classes

A class is on behalf of a concept or specific transaction in an ontology.,and it is a node in the graph model. Therefore, a class of ontology can be mapped to a node in a graph database. Namely, for arbitrary $c \in C$,

let $c \rightarrow v$, where $v \in V$.

(2) the mapping rule of relations

A relation represents the relationship between concepts and things. In the graph model, relations perform as edges. Therefore, a relation can be mapped to an edge connecting two nodes in the graph database. Suppose an arbitrary $r_{ij} \in R$ where $c_i, c_j \in C$, that means there is a relation r from c_i to c_j. The mapping rule is

$c_i \rightarrow v_m$, $c_j \rightarrow v_n$ *and* $r_{ij} \rightarrow e_{mn}$ where $e_{mn} \in E$ and $v_m, v_n \in V$.

(3) the mapping rule of functions

Function is a kind of special relation, which has the domain, range and corresponding rules. At present, function of one variable is only considered in this article. In graph model, the domain and the range often appear as the form of node, and the rule is the edge. So the function can give corresponding map to the nodes and relation in graph database. Assume an arbitrary $f_{ij} \in F$ where $c_i, c_j \in C$, it means that there is a function f from c_i *to* c_j. The mapping rule is

$c_i \rightarrow v_m$, $c_j \rightarrow v_n$ *and* $f_{ij} \rightarrow e_{mn}$ where $e_{mn} \in E$ and $v_m, v_n \in V$.

(4) the mapping rule of axioms

Axiom is the limitation of concepts in ontology field. All these limitations can be presented as the properties of concepts in ontology. Of course, the concepts are the nodes in graph database.

(5) the mapping rule of instances

The instance is the concrete representation of ontoloy model and the ontology model is the abstraction of the instance. In fact, layered thought will performa well in the instance and ontology model. In graph model, instance shows as a node. Namely, for arbitrary $i \in I$,

let $i \rightarrow v$, where $v \in V$.

2.3 Example

According to the mapping rules mentioned above, a special example can be given based on the code snippet of pizza.owl. The code snippet is shown in Fig. 2.

In Fig 3, there are a lot of classes, relations, properties and functions. Through the mapping rules, these elements could be mapped to the corresponding parts in a graph database. The mapping results are shown in Fig. 3.

```
(1) <owl:Class rdf:ID="Pizza">
(2) <owl:disjointWith>
(3) <owl:Class rdf:about="#PizzaBase"/>
(4) </owl:disjointWith>
(5) <owl:disjointWith>
(6) <owl:Class rdf:about="#PizzaTxopping"/>
(7) </owl:disjointWith>
(8) <owl:disjointWith>
(9) <owl:Class rdf:ID="IceCream"/>
(10) </owl:disjointWith>
(11) <rdfs:subClassOf>
(12) <owl:Class rdf:ID="DomainConcept"/>
(13) </rdfs:subClassOf>
(14) <rdfs:subClassOf>
(15) <owl:Restriction>
(16)<owl:onProperty>
(17) <owl:FunctionalProperty
rdf:ID="hasBase"/>
(18) </owl:onProperty>
(19)<owl:someValuesFrom>
(20)<owl:Class rdf:about="#PizzaBase"/>
(21)</owl:someValuesFrom>
(22) </owl:Restriction>
(23) </rdfs:subClassOf>
(24) </owl:Class>
```

Figure 2: Code Snippet of Pizza.owl

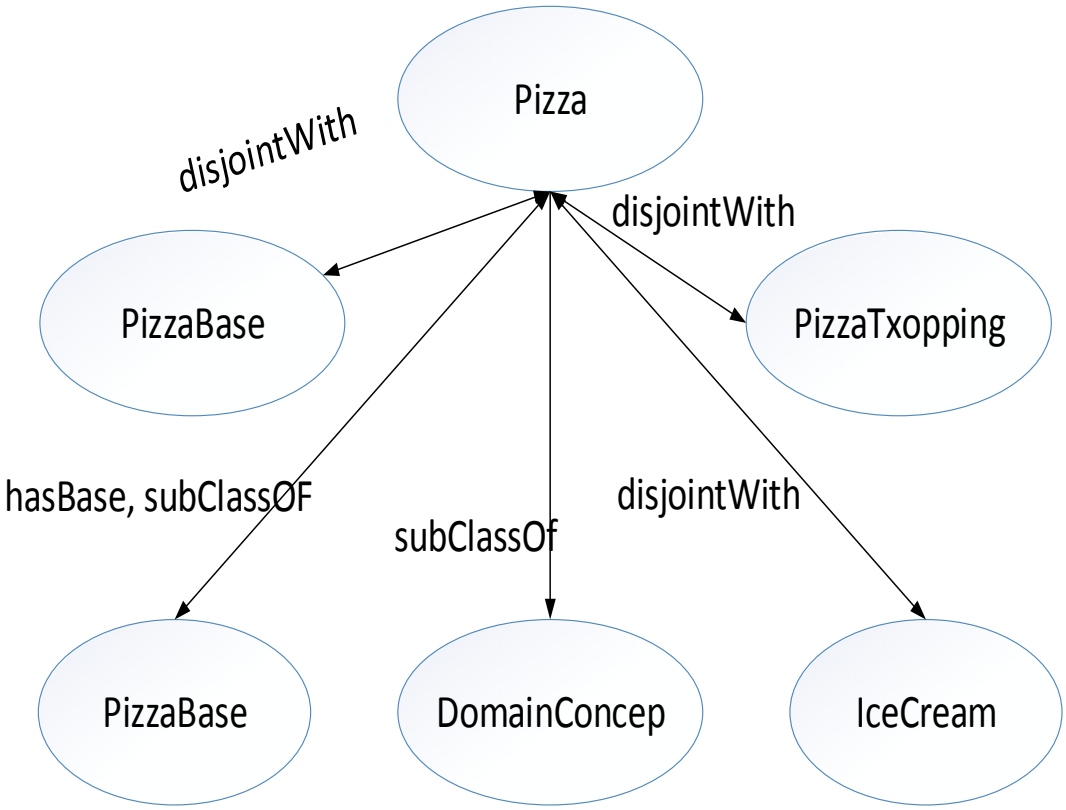

Figure 3: the Mapping Result

3. The Mapping Algorithm

Based on the mapping rules, a mapping algorithm is proposed in this section.

3.1 Adjacency Matrix

The adjacency matrix is a typical method of storing graph, which uses double dimensional array. The data in the matrix show the relation among nodes.

Suppose nodes v_i , $v_j \in V$ define an adjacency matrix K, where $k_{ij} \in K$ describes the relation from v_i to v_j. *Let*

k_{ij}=1, if there is a relation from v_i to v_j;

k_{ij}=0, if there is no relation from v_i to v_j;

k_{ij}=∞, if there is a relation from v_i to v_j when i=j.

In an ontology, there may be more than one relation from one node to another. So the adjacency matrix is extended in this paper. Let

k_{ij} = $name_1$,*name2, ..., nameN*, where $name_p$ is the relation's name and a comma is the name's separator.

k_{ij} = null, if there is no relation from v_i to v_j.

For example, according to the code snippet in Fig. 3, the part of the adjacency matrix is shown in Table 1.

	Pizza	**PizzaBase**	**PizzaTopping**	**IceCream**	**DomainConcept**
Pizza	null	hasBase,subClassOf	disjointWith	disjointWith	subClassOf
…...	…...	…...	…...	…...	…...

Table 1: Adjacency Matrix

3.2 The Algorithm

In the process of mapping the ontology to the graph database, the main two steps are included . The first step is the resolving of ontology and the second step is storing the contents into the graph database.

On one hand, when parsing the ontology, the class is the basic element. The most important setp is to traverse all the classes . If one class is not an anonymous class, it will be stored into an array and its corresponding relation and the object are are needed in the process. However, if the object does not exist, the relation must be neglected. The reason of this operation is that if we only get the relations expected of the subject and the object or only the subject or the relation without the object, the operation has no meaning. If the object exists, the corresponding relation must be stored into the two-dimensional array *K[i][j]*. In addition, as we all know that a single one node is also a graph, this phenomenon can't be neglected in the process of mapping ontology to graph database.

On the other hand, in the process of storing the contents into the graph database, steps are as follows. Firstly, create all the nodes in graph database according to the length of the array stored in the classes, and set the property for every node, such as the property of name, number and so on, and then traverse the array of *K[i][j]* until the traversing is over. In the process of traversing, if the intersection point of *i.th* and *j.th* column has value, it suggests that the relation involves node *i* and node *j*. In addition, the node's property is the name and number in graph database. As we can see from what has discussed above, a relation could be built. If the value of the interaction node is null, it means that the two nodes have no relation. This operation will go on until the traversing is over for *K[i][j]*. The algorithm is presented as follows:

(1)define *K* as String type 2d
(2)define *O* as an array of objects
(3)define *i , j* as Integer
(4)define *num* Integer
(5)define *name* String
(6)**ParseO2DB** (Ontolog *om*)
(7)*i*= 0
(8)foreach (every class in ontology named *a*)
(9) do if (*a* is not a anonymous class)
(10) then *a* store into*O[i]* , *i*++
(11) else do nothing
(12) end do
(13) foreach (every correspongding instance of *a* named *b*)
(14) do if (*b* does not exist and is not in *O[i]*)
(15) then store *b* into*O[i]* , *i*++
(16) else return
(17) end do
(18) end for
(19) end for
(20)for (*i*=0 to *O.size-1*)
(21) do create nodes in base and set property *name=O[i] num=i*
(22)end for
(23) foreach (every class in ontology named *a*)
(24) do if(*a* exists in *O*)
(25) then *i =a*
(26) foreach (every correspongding relation of *a* is named *r*)
(27) do if (the object of *r is* named *c*)
(28) then *j=c, K[i][j]=r*
(29) else return
(30) end do
(31) end for
(32)end for

Figure 4: the Mapping Algorithm

In this algorithm, some variables are defined. *O* is used to store all classes and *K* is used to store all relations among nodes. Firstly, traverse all the classes in ontolgoy. If one class is not an anonymous class, then store it into *O[i]*, where *i* is the sequence of the class. Secondly, for each class, traverse its corresponding instance. If one instance is not in *O[i],* it will be stored in *O[i]*. Otherwise the instance must be neglected. In addition, we must judge whether the two nodes have relations, if the relation exists, then store it into *K[i][j]*, where *i* and *j* represent the sequences of the correspongding two nodesuntil all the classes are traversed over. Finally, according to the traversing of the matrix *K[i][j]*, we can construct the graph database. The first row and column mean the nodes and the values in the matrix are their corresping relations. Until now, the ontology has been successfully mapped to graph database.

4. Experiments

A system is implemented by using Jena and Neo4j[11]. It uses the mapping algorithm to map a OWL ontology to a Neo4j graph database automatically. To verify that the mapping rules are general, some OWL ontologies in Protege3.6 are randomly chosen as the experiment data. They are SQWRLExample.owl, pizza.owl and collaborativePizza.owl, whose scales become gradually larger.

The following is the amount comparison of instances, relation and classes in ontologies and their corresping nodes, edges, nodes in the graph database. The results as shown are in Table 2. The results inlustrate all the elments in an ontology which have been mapped into a graph database.

Amount	Owl ontology			Amount	Graph db		
	sqwrlexamples.owl	**pizza.owl**	**collaborativepizza.owl**		**sqwrlexamples.owl**	**pizza.owl**	**collaborativepizza.owl**
classes	55	99	100	**nodes**	55	99	100
relations	124	1080	1082	**edges**	124	1080	1082
instances	41	5	5	**nodes**	41	5	5

Table 2: Result of Experiments

In addition to storing all the information of ontologies, graph databases can also support query operrations. The query language in Neo4j is Cypher [12], which is a kind of concise graph database query language. The query mode is similar to the way used for figures, which just need a precise way to describe the diagram and the wanted results. The differences of the Cypher and Sparql, the querying language of ontology, are shown in the Table 3.

Query target	Cypher	sparql
All relations and nodes	match a-[r]->b return a,r	select ?s ?p ?o where ?s ?p ?o
All the nodes have disjointWith relationship with Soho	Start a=node:index(name='Soho') match a-[:disjointWith]->(b) return b	Select ?o where'Soho' <http://www.w3.org/2002/07/ owl#DisjointWith> ?o.
All the nodes have indirect disjointWith relationship with Soho	start a=node:index(name='Soho') match a-[:disjointWith]->()-[:disjointWith]->(c) return c	select ?s where 'Soho' <http://www.w3.org/2002/07/ owl#DisjointWith> ?o."+"?o <http://www.w3.org/2002/07/ owl#DisjointWith>?s.

Table 3: Query contrast of cypher and sparql

According to the comparative analysis above, the query language of ontology can be replaced by Cypher in Neo4j. In addition, the form of Cypher's structure is more simple and easier to understand.

5.Conclusion

When using relational databases to store ontologies, there are problems of missing semantic information or information redundancy causedby the mismatch of their structures. In this paper, a new ontology storage method is presented based on graph databases, which aims to solve the problems. In addition, hundreds of millions of elements can be stored into graph databases, thus the scale limitation will be well solved. Of course, this paper only gives the basic discussion on the field of ontology and graph database. Topics such as how to optimize the query efficiency, and introduction of the ontology's reasoning mechanism to the field of graph databases, need to be further explored.

References

[1] W Bao G Y Li. *Ontology Storage Technology Research*[J]. Computer Technology and Development. 18(1): 146-150(2008)

[2] A Kiryakov , D Ognyanov , D Manov . *OWLIM – A Pragmatic Semantic Repository for OWL*[C]. In: M.Dean, et al., eds. Proc. of the WISE 2005 Workshops. Heidelberg: Springer – Verlag, 182-192(2005)

[3] T R Gruber . *A Translation Approach to Portable Ontology Specifications*[J]. Knowledge Acquisition, 5(2): 199-220(1993)

[4] R Agrawal ,A Somani , Y Xu . *Storage and Querying of E-Commerence Data*[C]. Proceedings of 27th International Conference on Very Large Data Bases. Morgan Kaufmann, Roma.149-158(2001)

[5] B McBride . *Jena:Implementing the RDF Model and Syntax Specification*[C]. Proceedings of the Second International Sematic Web Workshop. Technical Report, Hong Kong, 23(2001)

[6] T T Hu, MCao . *New storage model based on theory of ontology in relational database*[J]. Computer Engineering and Design. 35(9): 3075-3078(2014)

[7] X W He . *Probing Optimisation of RDF Semantic Data Storage in Big Data*[J]. Computer Applications and Software. 32(4): 38-42(2015)(In Chinese)

[8] L H Xiang. *Distributed Storage For Massive RDF Data based on Graph Database*[D]. Wuhan: Wuhan University of Science Technology(2013)(In Chinese)

[9] A GPerez , V R Benjamins . *Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem- Solving Methods*[C]. CEUR Workshop Proceedings. IJCAI and the Scandinavian AI Societies, New York, 1-15(1999) Partner J, Vukkotic A, Watt N. *Neo4j in Action*[M]. Manning Publications, American.304(2013)

[10] Carrol J,McBride B.*The Jena Semantic Web Toolkit*.Publica-pi,HP-Labs,Bristol. Avalableat:http://www.hpl.hp.com/semweb/jena-top.htm(2001)

[11] R GUrma , A Mycroft . *Source-code queries with graph databases—with application to programming language usage and evolution*[J]. Science of Computer Programming. 97(1):127-134(2015)