WikiQuery: A complete solution for knowledge base construction and multimodal querying

Hongtao Lin, Lisheng Wu
800 Dongchuan Rd. Shanghai Jiao Tong University
{Linkin_HearT, 573944287}@sjtu.edu.cn

Abstract

Knowledge base has become a trending focus in both field of database system and natural language processing (NLP). In this report, we focus on Wikidata, one of the largest open-source knowledge base available and analyzed the data structure in it. Based on several insights on data structure and demand from queries, we construct a minimal database for general queries. To further speed up the performance, we propose an efficient schema targeted for specific queries. Experiments show our design made a significant speed up in recursive compared to baseline. Also, we design a basic QA web system, WikiQuery, and demonstrate the promising potential for knowledge-based QA system.

Introduction

Since the prevalence of computer science, researchers has long sought for ways to store human knowledge into computer, making them as "knowledgeable" as our human beings. That's the very origin of knowledge graph. van de Riet and Meersman (van de Riet 1992), Stokman and de Vries (Stokman 1988), and Zhang (Zhang 2002), present a formal theory of knowledge graphs in terms of semantic networks, in which meaning is expressed as structure, statements are unambiguous, and a limited set of relation types are used. Following up, several definitions and variation of knowledge graph are proposed. (Corby 2010) focus on the structural properties (narrow down the graph as a directed labeled graph). (Pujara 2013) focus on using probabilistic soft logic (PSL) to manage uncertainty in knowledge graphs that have been extracted from uncertain sources.

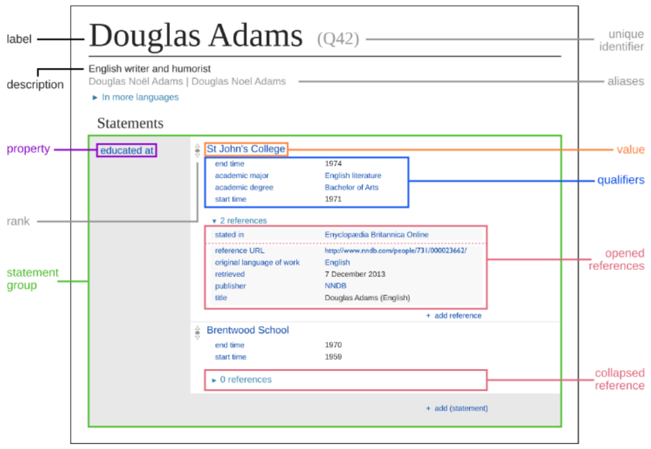
Besides storing and querying knowledge graph, constructing a knowledge graph is also a heated topic. Currently, most databases are edited and maintained by human, such as Freebase and Wikidata. There are also knowledges extracted from large-scale, semi-structured web knowledge bases such as Wikipedia, DBpedia and YAGO. Furthermore, information extraction methods for unstructured or semi-structured information are proposed, which lead to knowledge graphs like NELL, PROSPERA, or KnowledgeVault.

For our project, we select Wikidata as the main resource, since it's well-defined structural data and easy to access. Based on the data source, we perform database construction, optimization, query design on it.

The report is organized as follows: Section 2 gives an overview on the statistics and some example of data stored in Wikidata; Section 3 considers some optimization methods in database construction, based on the observations of data and demand from queries; Section 4 investigates several optimizations in terms on specific queries; Section 5 states about how we deal with natural language queries. Section 6 and 7 evaluate the overall system and suggest some future directions to go.

Wikidata Overview

Wikidata is a collaboratively edited knowledge graph, with edit permission to both human and machines. It's operated by the Wikimedia foundation, which also hosts the various language editions of Wikipedia. After the shutdown of Freebase, the data contained in Freebase is subsequently moved to Wikidata. A special highlight for Wikidata is that for each entity, provenance metadata like references and qualifiers can be included.


Douglas Adams (Q42)
English writer and humorist
Douglas Noël Adams | Douglas Noel Adams
▸ In more languages
Statements
educated at
St John's College

end time	1974
academic major	English literature
academic degree	Bachelor of Arts
start time	1971

▾ 2 references

stated in	Enyclopædia Britannica Online
reference URL	http://www.nndb.com/people/731/000023662/
original language of work	English
retrieved	7 December 2013
publisher	NNDB
title	Douglas Adams (English)

+ add reference
Brentwood School

end time	1970
start time	1959

▸ 0 references
+ add (statement)

Figure 1: A Sample Item from Wikidata

As illustrated in Figure 1, an entity in Wikidata has the commonly used label, description and aliases, associated with several statements. Besides, for each statements, it may also have qualifiers (education start time) and references (the statistics is stated in a trackable website). These information are the unique advantages for Wikidata as a complete and convincing source of knowledge graph.

According to the official website, Wikidata has just announced its fourth year celebration, with over 24 million entities and more than 2,000 properties in it.

Database Design

Wikidata is no doubt one of the largest knowledge base available on the web. But the question that comes after is: "Do we need to store all of them so as to make a knowledge graph?" The answer is definitely no, which we will illustrated below.

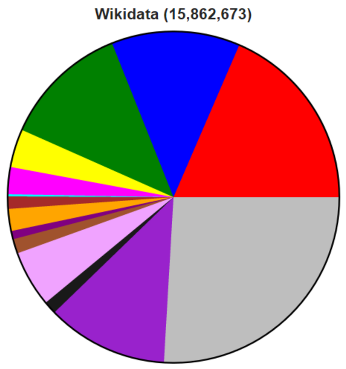


Figure 2: Wikidata Statatistics of Entities

According to the official report released in Oct. 2015 (See Figure 2), over 25% of the entities does not have a subclass/instance of as its property (Illustrated as gray), which means it may be isolated and less informative. Also, we found some of the entities does not even have a label with it, and the portion of these data takes up to 32%. Besides, according to our investigation on all properties, we found that more than 80% of them are used by a small portion of statements (small means less than 5000). Besides,

Based on the above analysis, we choose to filter out only entities without labels. For the properties, we only leave out informative ones with high frequency. Specifically, we rank the properties according to their "usage", i.e., the number of statement related to it, and filter out those with more than 4,000 usages. Also, since some of the properties with type "external-id" would be uninformative in querying, we decide to filter them out. The above process left only 12% of properties (around 300 in number), a significant decrease in number.

On the other hand, we consider our schema from perspective of required queries, from which we observed some insights, and made the following three adjustments in database accordingly:

Drop the "Reference" table. Since no query asks to illustrate the source of data for statement, we found the table "Reference" to be useless in such a condition. Thus we choose to discard it in database we actually stored. Same reason is also applicable for "Sitelink" and "Badge".

Ignore difference in datavalues. The original JSON file classifies datavalues into six value types (such as Time, Quantity, Globecoordinate), and assigned different structures to them. While it's a wise choice to make statements as clear as possible, our queries, however, do not ask for values in such details (e.g. the precision for a quantity measurement). Also, design datavalues with different structure would require different tables, which may yield inefficiencies and possible redundancies. Based on the above two reasons, we choose to concatenate the information of a value into a human-friendly format. Take the "Globecoordinate" for example, we may only store a string, representing latitude and longitude of a place, as a filed in table "Claim". That's enough for querying.

Store only part of information about entity. For tables like "Label", "Description" and "Alias", each is associated with a huge amount of languages. But we may only care about Chinese and English versions. So we choose to store the above data only if it's in Chinese or English.

As discussed above, we constructed a *simplified* version of Wikidata (See Figure 3). This schema is more like a standard triple representation of knowledge, (*subject*, *predicate*, *object*), formed in most knowledge graph system. In the following sections, we would continue to use this tripe denotation to refer to a claim without qualifiers and references. Besides this, our designation also supports qualifiers that further constraint on the triple.

The schema refinement leads to a huge deduction in time spent, from 15min per 10,000 entities to 40s.

Note that we also built scripts that can store the *complete* information in the dumped JSON file, which is included and configurable within script. The complete E-R model is included at the end of report (See Appendix A).

Query Optimization

The above constructed schema would serve as a general-purpose knowledge graph without much specifications. For the required queries, however, we can perform ever better by building individual optimizations. In this section, we will discuss the optimization on required tables.

Note that all queries below are based on our constructed schema. Due to time and space limitation, we do not import all data into SQL. Among all 20 million entities and associated statements, we proceeded to over half the process (10,300,000 entities).

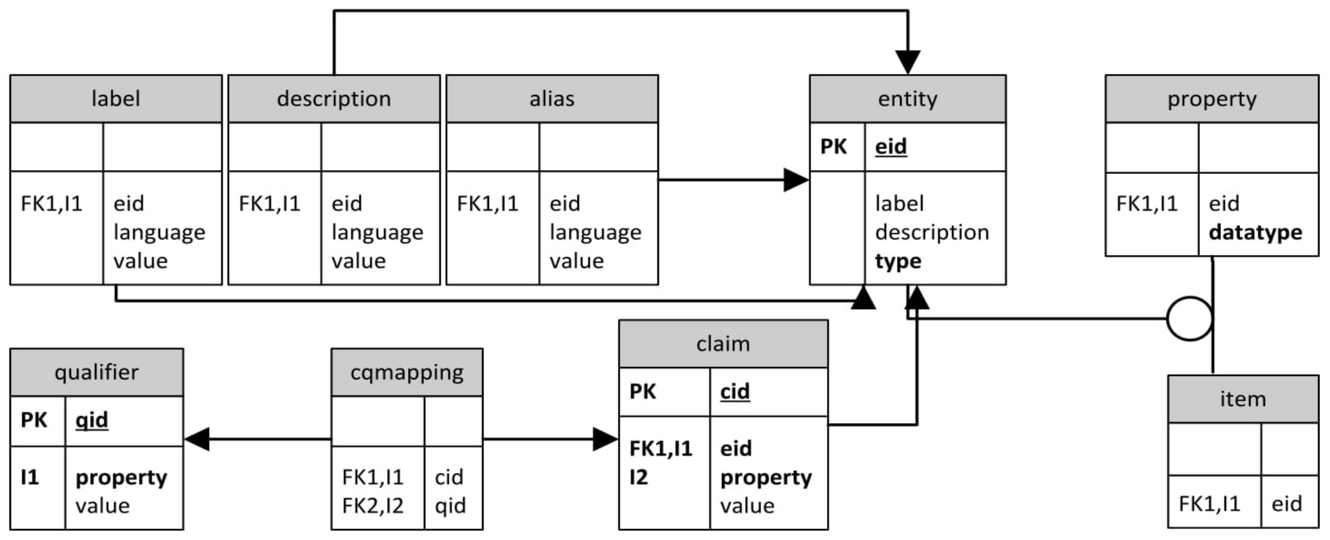


Figure 3: The Simplified ER Model

Given a name, return all the entities that match the name. A simple way is to query on table "Entity" directly:

SELECT eid, label, description FROM Entity WHERE label = $NAME

This single query would cost 120s on average, which is too long and unacceptable for users. But since we're dealing with a simple column (label), it's natural to think of building index on it:

CREATE INDEX Name2Id ON Entity (label);

Building indexes effectively decreases the query time, reducing to only 0.08s on average.

Given an entity, return all preceding categories (instance of and subclass of) it belongs to. We consider this problem under two conditions. If we only need to return one hierarchical relation, we can simply construct a new table "Preced", storing all the relationships to describe which category is preceding of one entity. Since table "Preced" has small space cost and is fast to process the query, so there is no need to accelerate it.

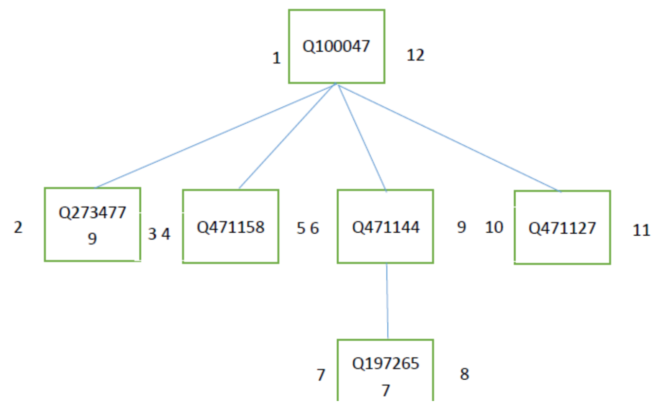


Figure 4: Preorder Tree Structure

On the other hand, we can move a step forward, returning all hierarchical relations. One naïve idea is to recursively query on this table. For example, entity Q18554966 has 11 hierarchical relations. After building index on Preced(eid), It costs 0.007s for jointly 11 queries. But we can definitely perform better. Here we proposed to use *preorder tree* built by modified preorder tree traversal algorithm.

In preorder tree structure, our hierarchy can still be maintained, as parent categories envelop their children. We represent this form of hierarchy in a table through the use of left and right values to represent the nesting of our nodes, i.e., each entry maintains the ID, lft, rgt. As illustrated in Figure 4, entity Q100047 is the four entities' preceding category, entity Q471144 is Q1972657's preceding category. The numbers listed besides those entities are lft and rgt respectively.

Thus, the problem is converted to: "Find those entities with lft less than lft in queried one AND rgt greater than rgt in queried one." If so, these entities must be the preceding categories of the queried entity no matter how many hierarchical relations between them.

We construct the tree by post-processing the table "Preced" and extract all the relations. By constructing this new structure and also build indexes in htree(eid) and htree(lft, rgt), we can greatly accelerate the query time to 0.0009s. Specifically, the query is listed as follows:

SELECT lft, rgt FROM htree WHERE eid = $ID;

SELECT eid FROM htree WHERE lft < $LFT AND rgt > $RGT;

By doing this, we can find the descending categories using the same table without writing another recursive function. It also reduces the I/O frequency when searching the preceding categories.

Given an entity, return all entities that are co-occurred with this entity in one statement. The result could be obtained by referring to the table Correlation(eid, weid, cid), where eid is the given entity id, weid is the co-occurred id, cid is the property id.

In order to cover all claims appeared, I query the entity in both columns and union them as the result.

```
SELECT DISTINCT(weid) FROM Correlation WHERE
eid = $ID
SELECT DISTINCT(eid) FROM Correlation WHERE
weid = $ID
```

Then we can union them and find the corresponding Entity by id.

Given an entity, return all the properties and statements it possesses. Based on the table Claim, Cqmapping, Qualifier. We can first search related Claim by the entity id. Then for each claim, we search related qualifiers from the table Cqmapping. Using the qualifier ID, we can find its corresponding value from the table Qualifier. Finally, we concatenate all of them to form a statement. Meantime, we still search the Entity to find corresponding names.

For detailed querying efficiency, you can refer to Appendix B.

Knowledge-based QA

There have been increasing interest on knowledge-based question answering (KB-QA) system. Starting from template-based QA system around 1990s, we have arrived at the era of deep learning, with substantial improvement on KB-QA (Yih 2015).

Due to limitation in time and resource, however, we turn to simple and effective template-based QA system. We expect our system to query a wide range of answers, which is categorized as two types.

The first type is the information in statements. Although it's possible to query on the *subject* and *predicate* of a tripe, here, we only consider the query of *object* value. We start from 300 selected properties described in Section 2, and articulated several templates target for this property. Supposed we want to match queries with properties of "date of birth", we can use regular expression like "when (is|was) the birthday of ($entity)". The above expression will extract a half-complete tripe: (*$entity*, "*date of birth*", *None*), thus we can query the "Claim" table for answer.

One thing to note is that we may encounter conditioned queries. Questions like "What's the population of China?" is ambiguous because we are not referring to the specific year. To deal with this kind of queries, we may need the information about qualifier, which is on our to-do list.

The second type is the description of a subject. Beside querying on triples, people are also interested in knowing the definitions of a subject. This type of question is often highly organized (e.g.: "Who is Obama?"). Thus we can easily extract the query from the template.

All templated stated here were integrated in the "Natural Language Querying" part of our web application. For more detailed documentation and usage, please turn to our demo website.

Future Work

We believe this piece of work is a solid starting point for future explorations. There may be a lost future work can be considered in the following two parts:

Firstly, on issues of efficiency in database design. Despite simplifying schema, we are not satisfied with the speed of data import. We also believe there leaves much room for improvement: we can try using more logical database decompression. Also, the storage size can be further compressed by calculating maximum length of each field.

Besides, we can deliberate more into KB-QA system. A more systematic way to do KB-QA, which has once populated during 2000s, is to do semantic parsing first, retrieving a more structured information about the sentence. Then we can convert the sentence into first-order logic (λ-calculus) and directly query from standard knowledge base. This is a simple but effective direction to go.

Conclusion

In this report, we investigated the whole process of processing, constructing and querying on Wikidata, an open-source knowledge base. Though optimization on querying about hierarchical relations by preorder tree, we obtained a significant speed up. Also we construct a simple and highly extensible web application, WikiQuery, to demonstrate the effectiveness of our databse.

References

RP van de Riet, RA Meersman. Knowledge Graphs. 97 (1992)

Frans N. Stokman, Pieter H. de Vries. Structuring Knowledge in a Graph. 186–206 (1988)

Lei Zhang. Knowledge graph theory and structural parsing. (2002)

Olivier Corby, Catherine Faron Zucker. The KGRAM Abstract Machine for Knowledge Graph Querying. (2010)

Jay Pujara, Hui Miao, Lise Getoor, William Cohen. Knowledge Graph Identification. 542–557 (2013)

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A Collaboratively Created Graph Database For Structuring Human Knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1247–1250, New York, 2008. ACM.

Denny Vrandecic and Markus Krötzsch. Wikidata: a Free ´ Collaborative Knowledge Base. *Communications of the ACM*, 57(10):78–85, 2014.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal*, 6(2), 2013

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A Core of Semantic Knowledge Unifying WordNet and Wikipedia. In *16th international conference on World Wide Web*, pages 697–706, New York, 2007. ACM

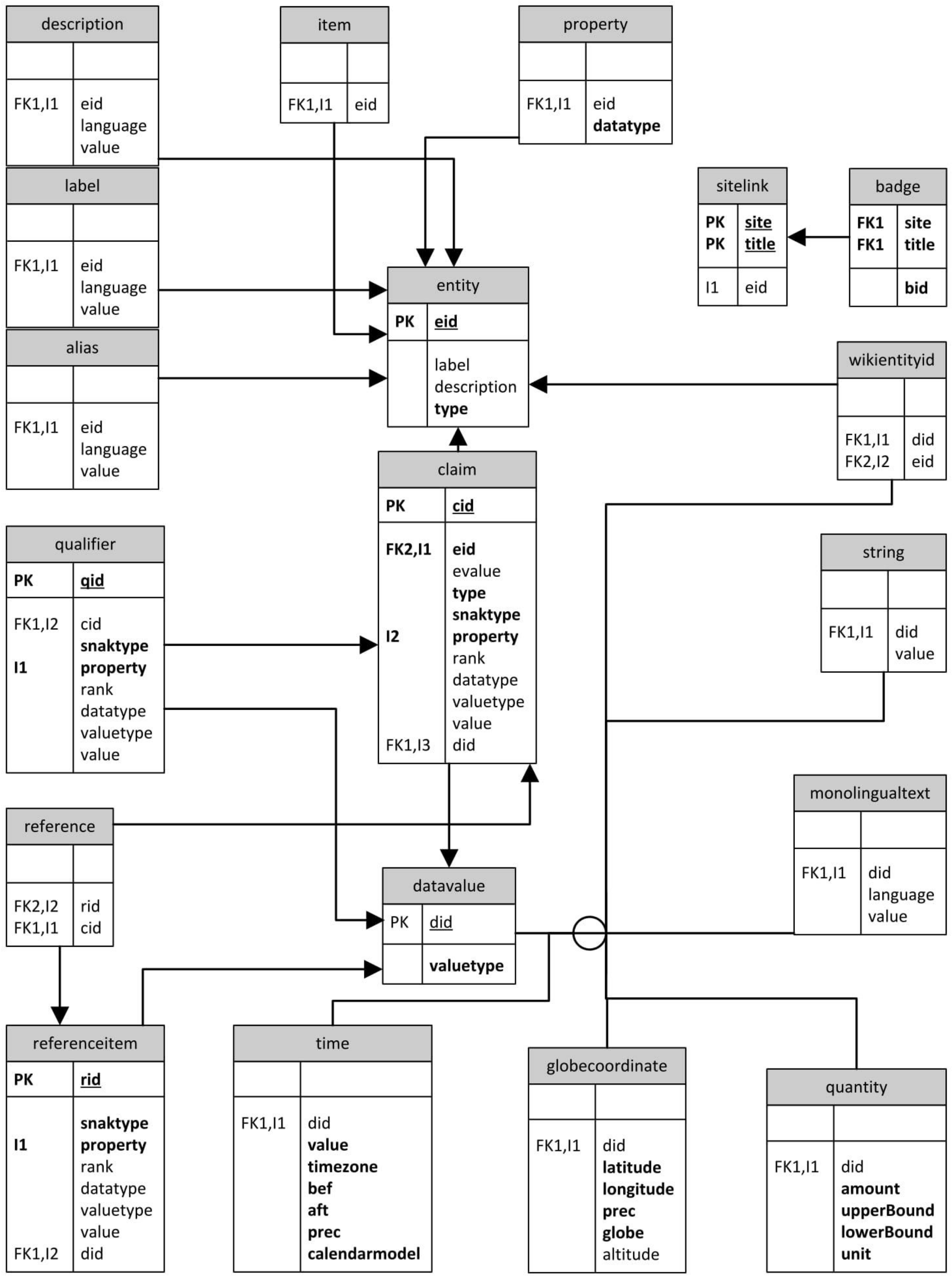
Andrew Carlson, Justin Betteridge, Richard C Wang, Estevam R Hruschka Jr, and Tom M Mitchell. Coupled semisupervised learning for information extraction. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 101–110, New York, 2010. ACM.

Xin Luna Dong, K Murphy, E Gabrilovich, G Heitz, W Horn, N Lao, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge Vault: A Web-scale approach to probabilistic knowledge fusion. In *20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601– 610, New York, 2014. ACM

Yih, Wen-tau, et al. "Semantic parsing via staged query graph generation: Question answering with knowledge base." *Association for Computational Linguistics (ACL).* 2015.

Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. Scalable knowledge harvesting with high precision and high recall. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 227– 236, New York, 2011

Appendix A: Complete ER Model



description
FK1,I1 eid language value
item
FK1,I1 eid
property
FK1,I1 eid datatype
label
FK1,I1 eid language value
alias
FK1,I1 eid language value
entity
PK eid
label description type
sitelink
PK site
PK title
I1 eid
badge
FK1 site
FK1 title
bid
wikientityid
FK1,I1 did
FK2,I2 eid
claim
PK cid
FK2,I1 eid evalue type snaktype
I2 property rank datatype valuetype value
FK1,I3 did
qualifier
PK qid
FK1,I2 cid snaktype
I1 property rank datatype valuetype value
string
FK1,I1 did value
monolingualtext
FK1,I1 did language value
reference
FK2,I2 rid
FK1,I1 cid
datavalue
PK did
valuetype
referenceitem
PK rid
snaktype
I1 property rank datatype valuetype value
FK1,I2 did
time
FK1,I1 did value timezone bef aft prec calendarmodel
globecoordinate
FK1,I1 did latitude longitude prec globe altitude
quantity
FK1,I1 did amount upperBound lowerBound unit

Note1: The circle associated with “datavalue” represents segregation relation.

Appendix B: Querying Efficiency

dHardware

Intel® Core™ i7-4790K CPU @ 4.00GHz × 8

Software environment

Ubuntu 16.04

	Optimization Method	Average Before	Average After
Select eid from Entity where label = $NAME	INDEX on Entity(label)	110.512	0.018
Select eid from Alias where alias = $ALIAS	INDEX on Alias(alias)	0.16s	0.0002s
Query the preceding categories	Build Tree	0.002	0.0003
Query the claim or statement	Decompose the Tables into three	150.3	0.03
	INDEX on Cqmapping(cid)	0.03	0.018

Result analysis, when querying one target is not from the primary key and the storage cost of the column behind the “from” is large. It’s very efficient to add the index on the column.