Querying the Edit History of Wikidata

Thomas Pellissier Tanon, Fabian M. Suchanek

Querying the Edit History of Wikidata

Thomas Pellissier Tanon and Fabian Suchanek

Télécom ParisTech

Abstract. In its 7 years of existence, Wikidata has accumulated an edit history of millions of contributions. In this paper, we propose a system that makes this data accessible through a SPARQL endpoint. We index not just the diffs done by a revision, but also the global state of Wikidata graph after any given revision. This allows users to answer complex SPARQL 1.1 queries on the Wikidata history, tracing the contributions of human vs. automated contributors, the areas of vandalism, the big schema changes, or the adoption of different values for the "gender" property across time.

1 Introduction

Recent years have seen the rise of Wikidata [13], a generalist collaborative knowledge base. The project started in 2012. As of July 2018, it has collected 5B statements about 50M entities. More than 700M revisions have been done by 56k contributors.

Wikidata provides a SPARQL endpoint[1] to query this data. However, this endpoint emits only the latest version of the data, and is blind to the edit history of the knowledge base. Therefore, users cannot even ask simple queries such as "What was the number of people in Wikidata two years ago?". This poses a problem for Wikidata contributors who wish to trace the progress of Wikidata over time, measure the amount of contributions by bots, identify areas of vandalism, or learn corrections from the data [9]. All these analyses require easy access to historical data.

The only way to access the full historical revisions of Wikidata is to download a 250GB set of compressed XML dumps, which contain a JSON blob for each revision. This dump is hard to manipulate, and even harder to index. It would be prohibitively expensive to create one RDF graph for each of the 700 million revisions, each with billions of triples. If one indexes only the differences between the revisions, one loses the global state at each revision.

In this paper, we propose a system that smartly indexes the revisions, so that the full history of Wikidata edits becomes usable in a SPARQL endpoint. Our endpoint is able to:

1. Retrieve the diff, i.e. the set of added and/or removed triples, for any Wikidata revision.
2. Evaluate any triple pattern against the global state of Wikidata after a given revision.

[1] `https://query.wikidata.org`

3. Retrieve the revisions that have added/removed triples matching a given triple pattern.

With this, we can answer questions such as "How many cities existed in Wikidata across time?", "How many entities were modified exclusively by bots?", or "How were different values for the gender property used across time in Wikidata?". All these queries can be asked in standard SPARQL, without any additions to the language.

2 Related Work

SPARQL endpoints for versioned RDF datasets have been proposed by several authors. [5] first discussed the problem of querying linked data archives. [6] provides an extensive discussion of known solutions. Our work is concerned with an actual implementation of such a versioned RDF store for Wikidata. This causes practical problems of size that have not been considered in previous works.

There are several systems that allow storing versioned RDF: Quad stores (e.g., [1]) and archives (e.g., [4]) could be used to store each triple annotated with their revision. However, this would mean that each triple would have to be stored once for every revision in which it appears. In the case of Wikidata, this would amount to quadrillions of triples to be stored. `tgrin` [10] allows annotating RDF triples with a timestamp. However, the system does not provide time range support or, indeed, an available implementation. R&Wbase [12] is a wrapper on top of a SPARQL 1.1 endpoint that provides a git-like system for RDF graphs. However, it stores only the diff between different revisions. Therefore, it does not allow efficient querying of the full graph state at a given point of time. x-RDF-3X [8] is a SPARQL database that annotates each triple with the timestamp at which it was added or remove. It annotates triples with validity ranges, thus permitting queries for any state of the database. It also provides advanced consistency features. However, by design, it does not allow loading data with already known timestamps or version IDs. Thus, it is not usable in our case. `v-RDFCSA` [2] allows efficient storage and retrieval of versioned triples. However, it does not allow subsequent additions of revisions, and does not support revision ranges. Therefore, the system is hard to use with the huge number of revisions that Wikidata brings. OSTRICH [11] allows storing version annotated triples, as well as querying them based on the version ID. It uses a HDT file for storing the base version and then stores a changeset with respect to this initial version. This system is not tailored to the case of Wikidata, where the base version is empty. Thus, the system would have to store the full global state for each revision.

3 System overview

Our goal is to provide a SPARQL endpoint that allows querying not just for the differences between Wikidata revisions, but also for the global state after each revision. At the same time, we want to use only existing SPARQL features. For this purpose, we designed the following data model:

Global State Graph: We will have one named graph per revision, which contains the global state of Wikidata after the revision has been saved.

Addition Graph: For each revision, we will have one graph that contains all the triples that have been added by this revision.

Deletion Graph: In the same spirit, we will have one graph for each revision that stores the triples that were removed.

Default Graph: The default graph contains triples that encode metadata about each revision: the revision author, the revision timestamp, the id of the modified entity, the previous revision of the same entity, the previous revision in Wikidata, and the IRIs of the additions, deletions, and global state graphs.

This data model allows us to query both the global state of Wikidata after each revision and the modifications brought by reach revision – without any change of the SPARQL 1.1 semantics or syntax [7]. We use the same schema as the official Wikidata dumps [3].

To store our data model, we cannot just load it as is into a triple store. The revision graphs alone would occupy exabytes of storage. Therefore, we use RocksDB[2]. RocksDB is a scalable key value store, optimized for a mixed workload between query and edits. This choice anticipates the possibility of live updates from Wikidata in the future. We create the following indexes:

Dictionary Indexes: Following a well-known practice in RDF storage systems, we represent every string by an integer id. We use one dictionary index to map the strings to their ids, and another one to map the ids to the strings.

Content Indexes: Each triple (s, p, o) appears in three content indexes. The indexes have as keys the permutations *spo*, *pos* and *osp*, respectively, and as value a set of revision ranges. Each range is of the form $[start, end[$, where *start* is the id of the revision that introduced the triple, and *end* is the id of the revision that removed it (or $+\infty$ if the triple has not been removed).

Revision indexes: We use two revision indexes, which give the set of triples that have been added or removed by a given revision. Since the Wikidata edits affect only a single entity and usually only a single statement, the number of triples added and removed per edit is small. Therefore, we can easily store all of them in the value part of the key-value store.

Meta indexes: We use several metadata indexes to store, for each revision, its author, the previous revision, etc.

Thanks to the content indexes, it is easy to retrieve for a given triple pattern when the triples have been added and, if relevant, when they have been removed. The indexes can also be used to evaluate a triple pattern against the global state graph, by filtering the results against the revision ranges. This is efficient because out of the 4931M triples that have existed in the Wikidata history as of July 1st 2018, only 475M have been removed. Therefore, the largest revision contains 90% of the total number of Wikidata triples that ever existed.

These indexes allow us to evaluate all possible quad patterns. If the quad pattern targets revision metadata, we use the metadata indexes. If it targets the content triples, we proceed as follows:

[2] `https://rocksdb.org/`

1. If the graph name is set to an addition graph or a deletion graph, we use the revision indexes to retrieve all the triples that were added or deleted, and we evaluate the triple pattern on them.
2. Otherwise, we do a prefix search in one of the content indexes, building the prefix from the bound parts of the triple pattern and choosing the index that allows us to have the longest prefix, and so the highest selectivity.
   (a) If the graph name is set and is a global state graph, we filter the triples that are returned from the prefix search by using the revision range as a filter.
   (b) Otherwise, we iterate through the matching triples, and build quads by using, for each triple, the revision range to find the ids of the addition graphs and the deletion graphs in which the triple appears. We do not return the ids of the global state graphs, because a query for the graph ids of a single triple pattern could return hundreds of millions of ids.

This storage system allows us to answer queries for a single quad pattern. To answer SPARQL queries, we plug our quad pattern evaluator into Eclipse RDF4J[3]. This system evaluates an arbitrary SPARQL 1.1 query by repeated calls to our quad pattern evaluator. It supports a wide range of query optimizations, including join reordering with static cardinality estimations that are based on the structure of the RDF dataset. For example, every revision has exactly a single author, and the number of triples per Wikidata entity is usually small.

Due to storage space constraints on the server provided by the Wikimedia Foundation, only the direct claim relations are loaded in our demo endpoint. We cover the range from the creation of Wikidata to July 1st, 2018. Our demonstration instance stores more than 700M revisions and 390M content triples about 49M items. The RocksDB indexes are using 64Go on disk with the RocksDB gzip compression system, after having compacted the database.

4 Demonstration setting

Our system allows users to query the entire Wikidata edit history. For example, Listing 1 asks for the 10 most frequent replacements for the value of the "gender" property. The query retrieves first the set of triples with the gender property (`wdt:P21`) (Line 2), annotated with the name of addition graphs. Then, it retrieves the names of the deletion graphs for each revision that have seen such additions (Line 3). Finally, it retrieves any deleted triple with the same subject and predicate (Line 4). After that it uses the usual SPARQL 1.1 features to compute the final result (Lines 1 and 5).

Such an analysis yields a number of interesting insights: First, we can observe that users correct erroneous gender values (such as "man") by their intended variants ("male"). Second, a slightly modified variant of the query allows us to quantify the gender gap in Wikidata over time: while the absolute difference between the number of men and women in Wikidata keeps increasing over time, the relative difference actually decreases. Finally, a similar query allows

[3] `http://rdf4j.org/`

```
SELECT ?addedGender ?deletedGender (COUNT(?revision) AS ?count) WHERE {
  GRAPH ?addGraph { ?s wdt:P21 ?addedGender }
  ?revision hist:additions ?addGraph ; hist:deletions ?delGraph .
  GRAPH ?delGraph { ?s wdt:P21 ?deletedGender }
} GROUP BY ?addedGender ?deletedGender ORDER BY DESC(?count) LIMIT 10
```

Listing 1: Retrieving the most common replacements of a "gender" value.

us to trace the increasing presence of non-traditional sex/gender values, such as "trans-male" or "non-binary" in the dataset.

5 Conclusion

In this paper, we propose a system that efficiently indexes the entire Wikidata edit history, and that allows users to answer arbitrary SPARQL 1.1 queries on it. Our system allows queries on both the revision diffs and the global state of Wikidata after each revision. Our system is available online at `https://wdhqs.wmflabs.org`. For future work, we plan to update the content of our system live based on the Wikidata edit stream.

Acknowledgement. Partially supported by the grant ANR-16-CE23-0007-01 ("DICOS").

References

1. Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., Velkov, R.: OWLIM: A family of scalable semantic repositories. Semantic Web **2**(1), 33–42 (2011)
2. Cerdeira-Pena, A., Fariña, A., Fernández, J.D., Martínez-Prieto, M.A.: Self-indexing RDF archives. In: DCC. pp. 526–535 (2016)
3. Erxleben, F., Günther, M., Krötzsch, M., Mendez, J., Vrandečić, D.: Introducing Wikidata to the linked data web. In: ISWC. pp. 50–65 (2014)
4. Fernández, J.D., Martínez-Prieto, M.A., Polleres, A., Reindorf, J.: HDTQ: managing RDF datasets in compressed space. In: ESWC. pp. 191–208 (2018)
5. Fernández, J.D., Polleres, A., Umbrich, J.: Towards efficient archiving of dynamic linked open data. In: DIACRON @ ESWC. pp. 34–49 (2015)
6. Fernández, J.D., Umbrich, J., Polleres, A., Knuth, M.: Evaluating query and storage strategies for RDF archives. In: SEMANTICS. pp. 41–48 (2016)
7. Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 query language (2013)
8. Neumann, T., Weikum, G.: x-RDF-3X: Fast querying, high update rates, and consistency for RDF databases. PVLDB **3**(1), 256–263 (2010)
9. Pellissier Tanon, T., Bourgaux, C., Suchanek, F.: Learning how to correct a knowledge base from the edit history. In: WWW (2019)
10. Pugliese, A., Udrea, O., Subrahmanian, V.S.: Scaling RDF with time. In: WWW. pp. 605–614 (2008)
11. Taelman, R., Sande, M.V., Verborgh, R.: OSTRICH: versioned random-access triple store. In: WWW. pp. 127–130 (2018)
12. Vander Sande, M., Colpaert, P., Verborgh, R., Coppens, S., Mannens, E., Van de Walle, R.: R&wbase: git for triples. LDOW @ WWW **996** (2013)
13. Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM **57**(10), 78–85 (2014)