Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph

Stanislav Malyshev[1(✉)], Markus Krötzsch[2(✉)], Larry González[2(✉)], Julius Gonsior[2(✉)], and Adrian Bielefeldt[2(✉)]

[1] Wikimedia Foundation, San Francisco, U.S.A.
smalyshev@wikimedia.org
[2] cfaed, TU Dresden, Dresden, Germany
{markus.krotzsch,larry.gonzalez,julius.gonsior,
adrian.bielefeldt}@tu-dresden.de

Abstract. Wikidata is the collaboratively curated knowledge graph of the Wikimedia Foundation (WMF), and the core project of Wikimedia's data management strategy. A major challenge for bringing Wikidata to its full potential was to provide reliable and powerful services for data sharing and query, and the WMF has chosen to rely on semantic technologies for this purpose. A live SPARQL endpoint, regular RDF dumps, and linked data APIs are now forming the backbone of many uses of Wikidata. We describe this influential use case and its underlying infrastructure, analyse current usage, and share our lessons learned and future plans.

1 Introduction

Since its inception in late 2012, Wikidata [19] has become one of the largest and most prominent collections of open data on the web. Its success was facilitated by the proximity to its big sister Wikipedia, which has supported Wikidata both socially and technically, e.g., with reliable server infrastructure and global user management. This has helped Wikidata to engage editors, and paved its way of becoming one of the most active projects of the Wikimedia Foundation in terms of contributors, surpassing most Wikipedia editions.[1] Many organisations now donate data and labour to help Wikidata's volunteers in selecting and integrating relevant pieces of information. A widely noted case was Google's migration of Freebase to Wikidata [17], but most organisational contributions naturally are coming from non-profits and research projects, such as Europeana.

Wikidata thereby has grown into one of the largest public collections of general knowledge, consisting of more than 400 million statements about more than

[1] Wikidata has attracted contributions from over >200 K registered editors (>37 K in Jan 2018).

© Springer Nature Switzerland AG 2018
D. Vrandečić et al. (Eds.): ISWC 2018, LNCS 11137, pp. 376–394, 2018.
https://doi.org/10.1007/978-3-030-00668-6_23

45 million entities. These figures still exclude over 60 million links from Wikidata entities to Wikipedia articles (familiar from Wikipedia's *Languages* toolbar), over 200 million labels and aliases, and over 1.2 billion short descriptions in several hundred languages. Wikidata thus has become the central integration point for data from all Wikipedia editions and many external sources, an authoritative reference for numerous data curation activities, and a widely used information provider. Applications range from user-facing tools such as Apple's Siri or EuroWings' in-flight app to research activities, e.g., in the life sciences [4] and in social science [20].

However, this success crucially depends on the ability of the underlying system to serve the needs of its growing community. Many thriving community projects have turned into deserted digital wastelands due to a failure to adopt to changing requirements. Unfortunately, the core of Wikidata is not well-adapted to the needs of data analysts and ontology engineers. It is built upon the well-tested ideas and methods of Wikipedia, which were, however, developed for encyclopedic texts. Internally, Wikidata's content likewise consists of strings, stored and versioned as character blobs in a MySQL database.

The underlying MediaWiki software that ensured the initial adoption and stability of Wikidata is therefore hardly suitable for advanced data analysis or query mechanisms. This was an acceptable trade-off for the first phases of the project, but it was clear that additional functionality would soon become a requirement for moving on. Most critically, the lack of reliable query features is a burden for the volunteer editors who are trying to detect errors, find omission or biases, and compare Wikidata to external sources.

The Wikimedia Foundation has considered many possible solutions for addressing this challenge, including own custom-built software and the use of existing NoSQL databases of several data models. The final conclusion was to build the core functionality on semantic technologies, in particular on RDF and SPARQL, as a mature technology which can address Wikidata's need to share, query, and analyse data in a uniform way. The main reason for this choice was the availability of well-supported free and open source tools for the main tasks, especially for SPARQL query answering.

Three years into the project, semantic technologies have become a central component in Wikidata's operations. The heart of the new functionality is a live SPARQL endpoint, which answers over a hundred million queries each month, and which is also used as a back-end for several features integrated with Wikidata and Wikipedia. Linked data web services and frequent RDF dumps provide further channels for sharing Wikidata content.

In this paper, we first discuss this influential application in detail. We present Wikidata and its RDF encoding and export services (Sects. 2 and 3), introduce the Wikidata SPARQL service *WDQS* (Sect. 4), and discuss its technical characteristics and current usage (Sects. 5 and 6). We conclude with some lessons we learned and a brief outlook.

2 Encoding Wikidata in RDF

The Wikidata knowledge graph is internally stored in JSON format, and edited by users through custom interfaces. We therefore start by explaining how this content is represented in RDF. Conceptually, the graph-like structure of Wikidata is already rather close to RDF: it consists of items and data values connected by properties, where both entities and properties are first-class objects that can be used globally. For example, Wikidata states that the item *Germany* for property *speed limit* has a value of *100* km/h. All items and properties are created by users, governed not by technical restrictions but by community processes and jointly developed guidelines. Properties are assigned a fixed datatype that determines which values they may take. Possible types include numbers, strings, other items or properties, coordinates, URLs, and several others. Unique identifiers are used to address entities in a language-agnostic way, e.g., *Germany* is Q183 and *speed limit* is P3086. It is natural to use these identifiers as local names for suitable IRIs in RDF.

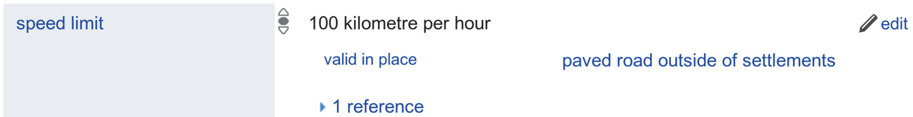

Fig. 1. Wikidata statement on the page about Germany (https://www.wikidata.org/wiki/Q183)

What distinguishes Wikidata from RDF, however, is that many components of the knowledge graph carry more information than plain RDF would allow a single property or value to carry. There are essentially two cases where this occurs:

- Data values are often compound objects that do not correspond to literals of any standard RDF type. For example, the value *100* km/h has a numeric value of 100 and a unit of measurement *km/h* (Q180154).
- Statements (i.e., subject-property-object connections) may themselves be the subject of auxiliary property-value pairs, called *qualifiers* in Wikidata. For example, the Wikidata statement for a speed limit of Germany is actually as shown in Fig. 1, where the additional qualifier *valid in place* (P3005) clarifies the context of this limit (*paved road outside of settlements*, Q23011975). Nesting of annotations is not possible, i.e., qualifiers cannot be qualified further.

Qualifiers used in statements can be arbitrary Wikidata properties and values, but Wikidata also has some built-in annotations of special significance. Most importantly, each statement can have one or more *references*, which in turn are complex values characterised by many property-value pairs. This can be a list of bibliographical attributes (title, author, publisher, ...), or something as simple as a reference URL.

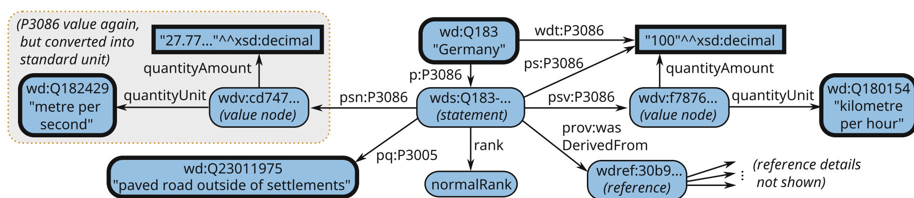

Fig. 2. RDF graph for the statement from Fig. 1, with added annotations and highlights for readability; properties with P3086 are facets of *speed limit*; pq:P3005 is the *valid in place* qualifier

Another type of built-in annotation on statements is the *statement rank*, which can be *normal* (default), *preferred*, or *deprecated*. Ranks are a simple filtering mechanism when there are many statements for one property. For example, cities may have many population numbers for different times, the most recent one being *preferred*. Such ranks simplify query answering, since it could require complex SPARQL queries to find the most recent population without such a hint. The rank *deprecated* is used for recording wrong or inappropriate statements (e.g., the statement that Pluto is a planet is deprecated).

We have developed a vocabulary and mapping for representing Wikidata in RDF. Details are found online,[2] so we focus on the main aspects here. The basic approach follows the encoding by Erxleben et al. [5], who had created early RDF dumps before the official adoption of RDF by Wikimedia. Complex values and annotated statements both are represented by auxiliary nodes, which are the subject of further RDF triples to characterise the value or statement annotations, respectively.

The statement in Fig. 1 therefore turns into an RDF graph as shown in Fig. 2, which we explain next. For readability, we have added labels for Wikidata items, and hints on the type of auxiliary nodes (whose lengthy IRIs are shortened). The two Wikidata properties P3086 (*speed limit*) and P3005 (*valid in place*) are associated with several RDF properties to disambiguate the use of properties in different contexts.

The simplest case is wdt:P3086, which relates a simplified version of the value (the number 100) directly to the subject. This triple is only generated for statements with the highest rank among all statements for the same Wikidata property and subject. As shown by our example, the simple encoding often captures only part of the available information, but this suffices for many applications, especially since values that are strings, URLs, or Wikidata entities can be represented in full detail as a single RDF entity.

Quantifiers and references can be accessed by following p:P3086 to the statement node. From the statement node, one can access again the simplified RDF value (via ps:P3086), or the complete value (via psv:P3086), which specifies the

[2] https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format, which also defines the namespace prefixes we use herein (all URLs in this paper were retrieved on 15 June 2018).

numeric amount and unit of measurement. The respective properties quantity-Value and quantityUnit are part of the OWL ontology of the underlying Wikibase software (http://wikiba.se/ontology#; we omit the namespace prefix). Property psn:P3086 leads to a normalised version of the value, which is generated by unit conversion for supported units. If available, each value node is linked to a normalised version of that value, which we do not show in the figure.

Qualifier values are also accessed from the statement node. The only qualifier here is P3005, represented by qualifier property pq:P3005 here. The statement further has a rank (*normal* in this case), and a reference, connected through RDF property prov:wasDerivedFrom from the W3C provenance vocabulary [8]. References also have their own RDF property IRIs for any Wikidata property used (not shown here).

Overall, this encoding of statements leads to graphs with many, sometimes redundant triples. This design is meant to simplify query answering, since users can easily ignore unwanted parts of this encoding, allowing queries to be as simple as possible and as complicated as needed. The remainder of the data of Wikidata, including labels, aliases, descriptions, and links to Wikipedia articles, is exported as described by Erxleben et al. [5], mostly by direct triples that use standard properties (e.g., rdfs:label).

3 Wikidata RDF Exports

Based on the above RDF mapping, the Wikimedia Foundation generates real-time linked data and weekly RDF dumps. Dumps are generated in two formats: a complete dump of all triples in Turtle, and a smaller dump of only the simplified triples for wdt: properties in N-Triples.[3] Dumps and linked data exports both are generated by Wikibase, the software used to run Wikidata, with RDF generated by the PHP library Purtle[4]. Both tools are developed by Wikimedia Germany, and published as free and open source software.

As of April 2018, the RDF encoding of Wikidata comprises over 4.9 billion triples (32 GB in gzipped Turtle). The dataset declares over 47,900 OWL properties, of which about 71% are of type owl:ObjectProperty (linking to IRIs) while the rest are of type owl:DatatypeProperty (linking to data literals). Most of these properties are used to encode different uses of over 4400 Wikidata properties, which simplifies data processing and filtering even where not required for avoiding ambiguity.

Besides the RDF encodings of over 415 million statements, there are several common types of triples that make up major parts of the data. Labels, descriptions, and aliases become triples for rdfs:label (206 M), schema:description (1.3 B), and skos:altLabel (22 M), respectively. Over 63 M Wikipedia articles and pages of other Wikimedia projects are linked to Wikidata, each using four triples for (schema:about, schema:inLanguage, schema:name, and schema:isPartOf).

[3] Current dumps are found at https://dumps.wikimedia.org/wikidatawiki/entities/.
[4] See http://wikiba.se/ and https://www.mediawiki.org/wiki/Purtle.

Finally, rdf:type is not used to encode the conceptually related Wikidata property *instance of* (P31) but to classify the type of different resources in RDF. There is a total of over 480 M triples of this kind.

Linked data exports for each Wikidata entity can be accessed in two ways: clients that support content negotiation will simply receive the expected data from the IRIs of Wikdiata entities. Users who wish to view a particular data export in a browser may do so using URLs of the form http://www.wikidata.org/wiki/Special:EntityData/Q183.nt. Other recognised file endings include ttl (Turtle) and rdf (RDF/XML).

During March 2018, the complete Turtle dump has been downloaded less than 100 times,[5] which is small as compared to the over 7,000 downloads of the similarly sized JSON dump. In contrast, more than 270 M requests for linked data in Turtle format have been made during that time, making this format more popular than JSON (16 M requests), RDF/XML (1.3 M requests), and NT (76 K requests) for per-page requests.[6] Part of this traffic can be attributed to our SPARQL endpoint, which fetches linked data for live updates, but there seem to be other crawlers that prefer linked data over dumps. The limited popularity of RDF dumps might also be due to the availability of a SPARQL query service that can directly answer many questions about the data.

4 The Wikidata Query Service

Since mid 2015, Wikimedia provides an official public Wikidata SPARQL query service (WDQS) at http://query.wikidata.org/,[7] built on top of the BlazeGraph RDF store and graph database.[8] This backend was chosen after a long discussion, mainly due to its very good personal support, well-documented code base, high-availability features, and the standard query language.[9] Data served by this endpoint is "live" with a typical update latency of less than 60 s. The service further provides several custom extensions and features beyond basic SPARQL support, which we describe in this section. Extensive user documentation with further details is available through the Wikidata Query Service help portal.[10] All software that is used to run WDQS is available online.[11]

4.1 User Interface

The most obvious extension of WDQS is the web user interface, shown in Fig. 3. It provides a form-based query editor (left), a SPARQL input with syntax highlighting and code completion (right), as well as several useful links (around the

[5] https://grafana.wikimedia.org/dashboard/db/wikidata-dump-downloads.
[6] https://grafana.wikimedia.org/dashboard/db/wikidata-special-entitydata.
[7] This is the user interface; the raw SPARQL endpoint is at https://query.wikidata.org/sparql.
[8] https://www.blazegraph.com/.
[9] https://lists.wikimedia.org/pipermail/wikidata-tech/2015-March/000740.html.
[10] https://www.wikidata.org/wiki/Help:SPARQL.
[11] https://github.com/wikimedia/wikidata-query-rdf.

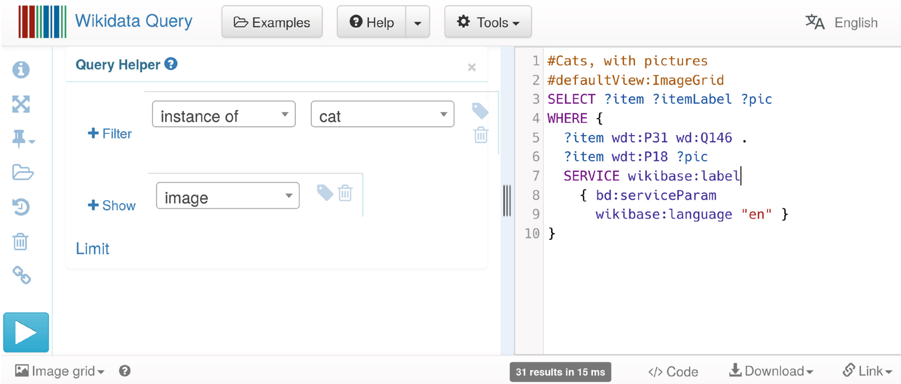

Fig. 3. Wikidata SPARQL query service UI with an example query

edges). As opposed to general-purpose SPARQL editors such as YASGUI [15], the interface has been customised for Wikidata to improve its functionality. For example, hovering the mouse pointer over any Wikidata entity in the SPARQL source displays a tooltip with a label in the user's browser language (or, if missing, a fallback) together with a short textual description from Wikidata (e.g., hovering on Q146 shows "cat (Q146): domesticated species of feline"). Likewise, auto-completion of Wikidata entities uses main labels and aliases, and ranks proposals by popularity in Wikipedia. These features are important to work with the opaque technical identifiers of Wikidata in an efficient way. Indeed, the interface is mostly targeted at developers and power users who need to design queries for their applications. A community-curated gallery of several hundred example queries (linked at the top) shows the basic and advanced features in practical queries.

Besides the usual tabular view, the front-end supports a variety of result visualisations, which can be linked to directly or embedded into other web pages. The most popular views that were embedded from January through March 2018 were timeline, map, bar chart, graph, image grid, and bubble chart.[12] The front-end further generates code snippets for obtaining the result of a given query in a variety of programming languages and tools, so as to help developers in integrating SPARQL with their applications.

4.2 Custom SPARQL Extensions

Besides the front-end, the SPARQL service as such also includes some custom extensions that are specific to Wikidata. In order to conform to the W3C SPARQL 1.1 standard, extensions were added using the SERVICE keyword, which SPARQL provides for federated queries to remote endpoints. Custom extensions then appear as sub-queries to special service URIs, which may, however, produce variable bindings in arbitrary ways. The query in Fig. 3 shows a

[12] https://grafana.wikimedia.org/dashboard/db/wikidata-query-service-ui.

call to the service wikibase:label, which takes as input variables from the outer query (e.g., ?item) and returns labels for the Wikidata entities they are mapped to as bindings to derived variable names (e.g., ?itemLabel). The labelling service is very popular, and it is the reason why the familiar query pattern

```
OPTIONAL { ?item rdfs:label ?itemLabel } FILTER (lang(?itemLabel) = 'en')
```

is rarely encountered in SPARQL queries. Besides efficiency, the main reason for introducing this service was to support user-specified language fallback chains that can be used to obtain labels even if there is no label in the preferred language.

Further custom services support searches for coordinates around a given point (wikibase:around) and within a given bounding box (wikibase:box), respectively. They are complemented by a custom function (used in SPARQL filter and bind clauses) to compute the distance between points on a globe, which one can use, e.g., to query for entities that are closest to a given location.

The special service wikibase:mwapi provides access to selected results from the MediaWiki Web API. For example, one can perform a full text search on English Wikipedia or access the (relevance-ordered) list of text-based suggestions produced by the search box on Wikidata when typing article names.[13]

Further services include several native services of BlazeGraph, such as graph traversal services that can retrieve nodes at up to a certain distance from a starting point, which is otherwise difficult to achieve in SPARQL.[14]

The service-based extension mechanism, which is already introduced by BlazeGraph, adds some procedural aspects to SPARQL query answering, but retains full syntactic conformance. A disadvantage of the current design is that the patterns that are used inside service calls may not contain enough information to compute results without access to the rest of the query, which one would not normally expect from a federated query.

Besides the custom services, the SPARQL endpoint also supports real federated queries to external endpoints. A white-list of supported endpoints is maintained to avoid untrusted sources that could be used, e.g., to inject problematic content into result views or applications that trust Wikidata.[15]

The only other customisations to BlazeGraph are some added functions, e.g., for URL-decoding, a greatly extended value space for xsd:dateTime (using 64bit Unix timestamps, for being able to accurately handle time points across the history of the universe), and support for a number of default namespace declarations.

[13] https://www.mediawiki.org/wiki/Wikidata_query_service/User_Manual/MWAPI.

[14] https://wiki.blazegraph.com/wiki/index.php/RDF_GAS_API.

[15] The current list of supported endpoints is at https://www.mediawiki.org/wiki/Wikidata_query_service/User_Manual/SPARQL_Federation_endpoints.

5 Realisation and Performance

In this section, we give technical details on the practical realisation of the Wikidata SPARQL query service, and performance figures obtained under current usage conditions.

The current release of Wikidata's query tools (v0.3.0) uses BlazeGraph v2.1.5. The graph database, including the live updater, is deployed on several servers of the Wikimedia Foundation, which share incoming traffic. As of April 2018, six machines in two U.S.-based data centres ("eqiad" in Ashburn and"codfw" in Carrollton) are utilised (CPU: dual Intel(R) Xeon(R) CPU E5-2620 v4 8 core; mem: 128 G; disk: Dual RAID 800 G SSD). The three servers in Carrollton are mostly for geographic redundancy and are currently not running under full load. The servers are not coordinated and each performs updates independently, so that the exact data may differ slightly between them.

Load balancing is managed by the Linux Virtual Server (LVS) software that is used throughout Wikimedia.[16] LVS is part of the operating system and does not require integration with BlazeGraph in any way. A web accelerator operates on top of the Web service endpoints by caching outputs. We use the Varnish system that is used for all Wikimedia page and API content. All responses (including error responses) are cached for 5 min, during which the exact same request will not be forwarded again to the endpoints.

The detailed configuration of BlazeGraph is part of the public source code of WDQS.[17] For the current content of almost 5 billion triples, the database needs more than 400 GB of storage space. The system is configured to use a query answering timeout of 60 s – queries running longer than this will be aborted. In addition, each client is allowed only up to five concurrent queries, and a limited amount of processing time and query errors. The latter is implemented with a *token bucket* protocol that allows clients an initial 120 s of processing time, and 60 query errors that they may use up. These buckets are refilled by 60 s/30 errors each minute, allowing clients to work at a steady rate. Clients that use up their allowed resources receive HTTP error code 429 (too many requests) with details on when they may resume querying. Importantly, WDQS does not impose limits on result size, and indeed one can effectively obtain partial data dumps of over a million results (subject to current traffic conditions).

The updater that synchronises the graph database with Wikidata is a separate process that polls Wikidata for recent changes, aggregates the modified entities, and retrieves RDF for all modified entities from the linked data service. Updates are performed using SPARQL update queries that delete all statements that are directly linked from modified entities and not retrieved with the new dataset. This may leave some auxiliary nodes (used for representing complex data values) orphaned, and another update query is used to remove such nodes

[16] See https://wikitech.wikimedia.org/wiki/LVS for technical details.

[17] https://github.com/wikimedia/wikidata-query-deploy/blob/master/RWStore.properties.

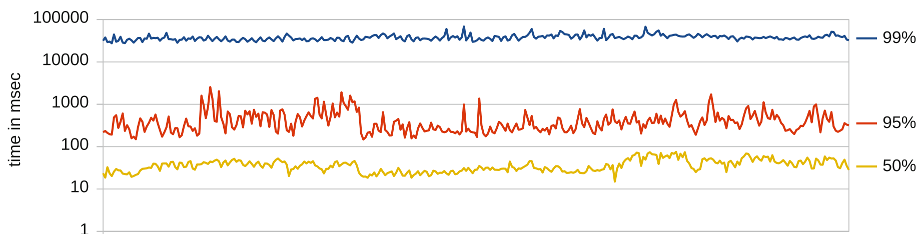

Fig. 4. Average query answering times for top 50%, 95%, and 99% of queries on a logarithmic scale; Ashburn cluster ("eqiad"); January through March 2018

(restricted to nodes used in the updated items' data before). Updates happen every 500 changes, or every few seconds if no changes occur.

Since its launch in 2015, WDQS has experienced a continued increase in query traffic. In 2018, there have been over 321 million requests within the twelve weeks from 1st January to 25th March, for a rate of 3.8 million requests per day or 44 per second. We have created several online dashboards with live usage metrics. Core load and performance figures are found in the Wikidata Query Service dashboard.[18] Among other metrics, the dashboard shows, for each server, the average number of requests and errors per second, the total number of triples stored, CPU load, and available memory. "Varnish latency" measures the time that the cache had to wait for content to be produced by the service, i.e., it is an upper bound for BlazeGraph's internal query answering times. Figure 4 shows the resulting average times that it took to answer various percentiles of queries on the (busier) "eqiad" cluster. The top 50% of the queries are answered in less than 40ms on average, while the top 95% finish in an average of 440 ms. The top 99% can reach peak latencies that are close to the 60 s timeout, but on average also stay below 40 s. At the same time, the total number of timeouts generated by BlazeGraph on all servers during this interval was below 100,000, which is less than 0.05% of the requests received.

Hernández et al. have studied the performance of query answering over Wikidata using several graph databases, including BlazeGraph and Virtuoso, and several ways of encoding statements [7]. They concluded that best performance might be achieved using Virtuoso and named graphs, which might seem at odds with our positive experience with BlazeGraph and statement reification. However, it is hard to apply their findings to our case, since they used BlazeGraph on spinning disks rather than SSD, which we discovered to have a critical impact on performance. Moreover, they used a plain version of BlazeGraph without our customisations, and focused on hypothetical query loads that heavily rely on accessing statements in full detail. It is therefore hard to tell if Virtuoso could retain a performance advantage under realistic conditions, making it an interesting topic for future investigations.

[18] https://grafana.wikimedia.org/dashboard/db/wikidata-query-service.

Table 1. Query dataset sizes with contribution of robotic and organic queries

	Start – End	Total	Valid	Robotic	Organic
I1	2017-06-12 – 2017-07-09	79,082,916	59,555,701	59,364,020	191,681
I2	2017-07-10 – 2017-08-06	82,110,141	70,397,955	70,199,977	197,978
I3	2017-08-07 – 2017-09-03	90,733,013	78,393,731	78,142,971	250,760
I4	2018-01-01 – 2018-01-28	106,074,877	92,100,077	91,504,428	595,649
I5	2018-01-29 – 2018-02-25	109,617,007	96,407,008	95,526,402	880,606
I6	2018-02-26 – 2018-03-25	100,133,104	84,861,808	83,998,328	863,480

Metrics on the usage of the Web query user interface are collected in a second dashboard.[19] A detailed usage analysis of this interface is not in scope for this paper, but we can note that the overall volume of queries that are posed through this interface is much lower, with several hundred to several thousand page loads per day. At the same time, the queries posed trigger many errors (over 50% on average since January), which are mostly due to malformed queries and timeouts. This suggests that the interface is indeed used to design and experiment with new queries, as intended.

6 SPARQL Service Usage Analysis

In this section, we focus on the current practical usage of WDQS, considering actual use cases and usage patterns. To this end, we analyse server-side request logs (Apache Access Log Files) of WDQS. As logs contain sensitive information (especially IP addresses), this data is not publicly available, and detailed records are deleted after a period of three months. For this research, we therefore created less sensitive (but still internal) snapshots that contain only SPARQL queries, request times, and user agents, but no IPs. We plan to release anonymised logs; see https://kbs.inf.tu-dresden.de/WikidataSPARQL.

6.1 Evaluation Data

We extend our recent preliminary analysis, which considered series of WDQS logs from 2017 [1] by including another set of logs recorded with some months distance, giving us a better impression of usage change over time. We consider the complete query traffic in six intervals, each spanning exactly 28 days, as shown in Table 1, which also shows the total number of requests for each interval. Further parts of this table are explained below.

We process all queries with the Java module of Wikidata's BlazeGraph instance, which is based on OpenRDF Sesame with minimal modifications in the parsing process re-implemented to match those in BlazeGraph. In particular,

[19] https://grafana.wikimedia.org/dashboard/db/wikidata-query-service-ui.

BIND clauses are moved to the start of their respective sub-query after the first parsing stage. This resulted in a total of 481,716,280 valid queries (208,347,387 from 2017 and 273,368,893 from 2018).

A major challenge in analysing SPARQL query logs is that a large part of the requests is generated systematically by software tools. Due to the huge impact that a single developer's script can have, such effects do not average out, at least not at the query volumes that we are observing in Wikidata. Traditional statistical approaches to the analysis of (user-generated) traffic therefore are heavily biased by the activities of individual users. There is no causal relation between the number of queries generated by a tool and the relevance or utility of these queries to the Wikidata community: in extreme cases, we have witnessed one and the same malformed SPARQL query being issued several million times, evidently generated by some malfunctioning script.

To address this issue, we have introduced an approach of classifying queries into *robotic* and *organic* requests [1]. Robotic traffic generally comprises all high-volume, single-source components of the total query load, whereas organic traffic comprises low-volume, multi-source queries. Our hypothesis is that organic traffic is dominated by queries that indicate an immediate information need of many users, and therefore is more interesting for usage analysis. Robotic traffic is still interesting from a technical perspective, since strongly it dominates the overall query load and therefore governs server performance. Notably, we do not want to single out hand-written queries of power users, as done, e.g., by Rietfeld and Hoekstra [14], but we let organic traffic include queries that users issue by interacting with software or websites, possibly without being aware of the underlying technology. The boundary between the two categories cannot be sharp, e.g., since some advanced browser applications may generate significant numbers of queries, and since some interactive applications may pre-load query results to prepare for a potential user request that may never happen. We believe that our classification is robust to the handling of such corner cases, since they do not dominate query load. As we will see below, the different traffic types exhibit distinctive characteristics.

We have classified queries using the following steps:

(1) All queries issued by non-browser user agents were considered robotic. These were mostly self-identified bots and generic programming language names (e.g., "Java").
(2) We scanned the remaining queries with browser-like user agent strings for comments of the form "`#TOOL:` *toolname*," which are conventionally used in the community to identify software clients in the query string. Of the tools that occurred, we considered *WikiShootMe* (nearby sites to photograph), *SQID* (data browser), and *Histropedia* (interactive timelines) as sources of organic traffic, and all others as robotic.
(3) We manually inspected the remaining browser-based queries to identify high-volume, single-source query loads. For this, we abstracted queries into query patterns by disregarding concrete values used for subjects, objects, LIMIT and OFFSET. We consider each agent that issues more than 2,000 queries

of the same type during one interval, and classify it as robotic if its temporal query distribution is highly irregular (typically in the form of peaks with thousands of queries for one or more hours, followed by inactivity during other days). It is often hard to determine the source of an identified traffic peak, but based on the number of distinct query types encountered, we estimate that at most 300 sources were filtered in this way (probably less, since some those query types are very similar and likely stem from one tool).

The outcome of our query type classification is shown in Table 1. The total of 2,980,154 organic queries (640,419 in 2017 and 2,339,735 in 2018) accounts for 0.6% of the traffic, and would therefore be overlooked completely in any non-discriminative analysis. We observe a general increase in query traffic over time, with a faster relative increase of organic traffic. This seems to be caused by the appearance of more and more user-space applications that use SPARQL to access Wikidata. The slight decline in overall traffic in I6 can be attributed to a single extremely active query source (the data-integration bot "auxiliary matcher"), which issued around 20.4 M queries in I5 but only 5.4 M in I6. This case illustrates how fragile query traffic statistics can be if such sources are not identified.

6.2 SPARQL Feature Analysis

We now report on several structural features of SPARQL queries throughout the datasets, which allow us to observe important differences between organic and robotic queries, but also between Wikidata and other SPARQL services. Our previous work includes an extended discussion of the different characteristics of organic and robotic traffic, including temporal distribution and language usage, which we do not repeat here [1].

Table 2 shows the prevalence of most common solution set modifiers, graph matching patterns, and aggregation functions. *Join* refers to the (often implicit) SPARQL join operator; *Filter* includes FILTER expressions using NOT EXIST; and SERVICE calls are split between the Wikidata labelling service (*lang*) and others. Queries with REDUCED, EXISTS, GRAPH, + in property paths, and other aggregation functions generally remained below 1%. Any other feature not shown in the table has not been counted.

The differences between organic and robotic traffic are clearly visible. Features that are much more common in organic queries include LIMIT, DISTINCT, OPTIONAL, ORDER BY, subqueries, services, and all aggregates. Robotic queries more often include BIND, UNION, and VALUES. FILTER and path queries occur in both traffic types to varying (high) degrees. In general, organic queries are more varied and use more analytical features, which is consistent with our expectations.

We can also see changes over time. Measures for robotic traffic are generally less stable, even from one month to the next, but especially when comparing I1–I3 with I4–I6, we can also see major changes for organic traffic. This includes strong increases in property paths, DISTINCT, LIMIT, and ORDER BY, and

Table 2. Prevalence of SPARQL features among valid queries in percent of total queries per dataset

Feature	Organic						Robotic					
	I1	I2	I3	I4	I5	I6	I1	I2	I3	I4	I5	I6
Limit	31.21	39.42	46.44	51.46	50.44	36.12	21.50	16.90	17.28	20.44	11.51	15.27
Distinct	26.71	31.37	18.67	59.19	60.12	64.22	14.96	5.53	4.21	4.29	7.54	10.84
Order By	17.18	14.27	12.80	46.51	46.16	34.15	12.08	8.17	6.77	8.81	7.74	17.82
Offset	0.38	2.75	0.35	0.08	0.07	0.04	7.94	6.04	6.29	0.10	0.07	0.10
Join	87.41	87.74	89.76	82.45	91.67	86.99	88.32	79.09	67.47	73.21	61.18	69.62
Optional	42.82	46.69	56.50	50.32	40.37	40.86	24.25	11.69	11.31	12.74	15.41	29.69
Filter	25.21	28.58	21.66	12.30	11.40	11.38	20.73	17.98	13.64	14.44	16.56	29.44
Path with *	14.91	15.21	12.55	40.20	31.24	29.69	16.89	19.58	14.81	20.65	17.37	23.71
Subquery	13.29	0.00	0.00	6.35	4.99	5.32	0.34	0.00	0.00	0.09	0.13	0.11
Bind	9.87	9.26	8.66	4.81	3.98	4.21	16.70	12.28	9.57	11.85	13.61	23.01
Union	5.14	5.75	12.78	2.46	1.90	2.68	11.56	8.79	7.61	14.03	13.16	18.93
Values	4.33	3.05	11.02	3.18	3.04	3.23	36.70	31.35	28.97	29.97	24.00	11.91
Not Exists	3.28	3.27	2.44	1.08	0.71	0.66	0.19	0.22	0.19	0.26	0.29	0.35
Minus	2.00	2.81	1.55	0.81	0.56	0.70	0.52	0.92	1.06	1.45	1.24	1.79
Service lang	44.74	41.82	54.98	50.19	40.51	42.05	9.41	6.25	4.27	7.13	7.85	8.95
Service other	11.55	10.66	10.38	7.50	13.51	2.30	4.62	0.18	1.15	0.16	0.15	0.39
Group By	17.18	19.86	12.88	6.89	5.19	5.06	0.41	0.37	0.46	0.20	0.21	0.36
Sample	8.97	11.15	4.68	1.61	1.64	1.64	0.03	0.03	0.05	0.04	0.03	0.09
Count	7.50	7.26	7.98	5.27	3.79	3.71	1.17	4.38	0.30	1.53	0.65	0.90
GroupConcat	1.80	2.79	1.14	0.87	0.85	0.74	0.06	0.09	0.02	0.03	0.02	0.28
Having	1.16	1.12	0.70	0.65	0.26	0.33	0.01	0.01	0.00	0.00	0.00	0.04

decreases in FILTER, UNION, Subquery, and aggregates. We should note that numbers are relative to each dataset: a decreasing value might be caused by an actual decline in queries using a feature, but also by an increase in queries with other features.

Our results show significant differences from previously reported findings. The most systematic recent overview across several datasets is given by Bonifati et al., who have analysed SPARQL logs from BioPortal, the British Museum, DBpedia, LinkedGeoData, OpenBioMed, and Semantic Web Dog Food [3]. They found SERVICE, VALUES, BIND, and property paths to occur in less than 1% of queries, while they have great relevance in all parts of our data. For SERVICE, this can be explained by the custom SPARQL extensions of WDQS discussed in Sect. 4, and the results seem to confirm their utility. VALUES and BIND are useful for building easy to modify query templates, with VALUES being particularly useful for (typically robotic) batch requests. Property paths express reachability queries, which are particularly useful for navigating (class) hierarchies and can in fact be used to emulate some amount of ontological reasoning in SPARQL [2]. The reduced occurrence of these features in other logs may be

Table 3. Co-occurrence of SPARQL features in percent of total queries per dataset (Join, Filter, Optional, Union, Path, Values, Subquery)

							organic		robotic	
J	**F**	**O**	**U**	**P**	**V**	**S**	I1–I3	I4–I6	I1–I3	I4–I6
		(none)					8.18	9.35	19.67	27.96
J							15.23	32.31	10.81	10.10
	F						1.09	0.94	1.95	1.27
J	F						8.87	2.37	2.61	1.50
J				P			2.93	1.63	13.72	14.09
J	F			P			2.49	0.58	0.39	0.06
J					V		0.41	2.04	30.91	17.65
		O					1.28	1.61	0.12	0.64
J		O					25.97	7.16	1.88	1.95
J		O		P			2.10	28.41	0.36	0.05
J	F	O					3.30	1.30	1.78	1.19
J		O	U				3.57	0.25	0.02	0.00
J		O			V		3.45	0.40	0.11	0.43
J		O		P	V		1.01	0.06	0.16	0.04
J						S	0.86	1.43	0.06	0.01
J		O				S	1.64	0.63	0.00	0.01
J	F					S	0.64	2.17	0.02	0.01
J	F	O		P			0.87	0.31	0.65	1.60
J			U	P	V		0.01	0.01	0.05	1.94
All cases shown							**83.90**	**92.96**	**85.27**	**80.50**

due to the fact that they are new in SPARQL 1.1, and therefore might have a lower acceptance in communities that have worked with SPARQL 1.0 before.

Low prevalence was also reported for subqueries, SAMPLE, and GroupConcat. Our robotic traffic is similar, but our organic traffic paints a different picture, and especially contains many subqueries, which are often needed for complex analytical queries.

We also analysed how features are combined in queries. Table 2 suggests that join, OPTIONAL, UNION, FILTER, subqueries, property paths, and VALUES should be taken into account here. SERVICE is also common, but mostly for labelling, where it has little impact on query expressivity. We therefore ignore the labelling service entirely. We do not count queries that use any other type of service, or any other feature not mentioned explicitly (the most common such feature is BIND). Solution set modifiers (LIMIT etc.) and aggregates (COUNT etc.) are ignored: they can add to the complexity of query answering only when used in subqueries, so this feature can be considered an overestimation of the amount of such complex queries.

Results are shown in Table 3 for all operator combinations found in more than 1% of queries in some dataset. Therein, path expressions include all queries with * or +. We separate the 2017 and 2018 intervals to show change. The shown combinations account for most queries (see last line of tables). Most remaining queries use some uncounted feature, especially BIND. As before, we can see significant variations across columns. Regarding organic traffic, the largest changes from 2017 to 2018 are the increase in join-only queries and the decrease of JO queries in favour of JOP queries. For robotic traffic, we see an increase of queries without any feature and a decrease of JV queries.

Other studies of SPARQL usage noted a strong dominance of conjunctive-filter-pattern (CFP) queries, typically above 65% [3,13]. The corresponding combinations *none*, J, F, and JF only account for 33%–45% in our datasets. We can cover many more robotic queries (58%–66%) by also allowing VALUES. However, filters are actually not very relevant in our queries. A much more prominent frag-

ment are *conjunctive 2-way regular path queries* (C2PRQs), which correspond to combinations of J, P, and (by a slight extension) V [6]. They cover 70%–75% or robotic traffic. To capture organic traffic, we need optional: queries with J, O, P, and V cover 60%–83% there. This is expected, since users are often interested in as much (optional) information as possible, while automated processing tends to consider specific patterns with predictable results.

We see very little use of UNION, whereas previous studies found UNION alone to account for as much as 7.5% of queries [3]. Since especially robotic traffic contains many queries with UNION (Table 2, we conclude that this feature typically co-occurs with features not counted in Table 3, most likely BIND.

7 Discussion and Lessons Learned

The decision to rely upon semantic technologies for essential functionality of Wikidata has taken Wikimedia into unknown territory, with many open questions regarding technical maintainability as well as user acceptance. Three years later, the work has surpassed expectations, in terms of reliability and maintainability, as well as community adoption.

The early "beta" query service has developed into a stable part of Wikimedia's infrastructure, which is used as a back-end for core functions in Wikimedia itself. For example, Wikidata displays warnings to editors if a statement violates community-defined constraints, and this (often non-local) check is performed using SPARQL. WDQS is also used heavily by Wikidata editors to analyse content and display progress, and to match relevant external datasets to Wikidata.[20] Results of SPARQL queries can even be embedded in Wikipedia (in English and other languages) using, e.g, the graph extension.

Wikimedia could not follow any established path for reaching this goal. Public SPARQL endpoints were known for some time, but are rarely faced with hard requirements on availability and robustness [18]. The use of continuously updated live endpoints was pioneered in the context of DBpedia [12], albeit at smaller scales: as of April 2018, the DBpedia Live endpoint reports 618,279,889 triples across all graphs (less than 13% of the size of Wikidata in RDF).[21] Moreover, the update mechanisms for DBpedia and Wikidata are naturally very different, due to the different ways in which data is obtained.

Many Wikidata-specific extensions and optimisations have been developed and released as free software to further improve the service. This includes custom query language extensions to integrate with Wikimedia (e.g., many-language support and Web API federation), a Wikidata-specific query interface, and low-level optimisations to improve BlazeGraph performance (e.g., a custom dictionary implementation reduces storage space for our prefixes). Some of our most important lessons learnt are as follows:

BlazeGraph Works for Wikimedia. The choice of graph database followed a long discussion of many systems, and some of the top-ranked choices did not use

[20] See, e.g., https://tools.wmflabs.org/mix-n-match/.
[21] http://live.dbpedia.org/sparql.

RDF at all. While there could have been other successful approaches, BlazeGraph has proven to be an excellent choice, regarding stability but also (open source) extensibility.

Customisation Matters. We are convinced that our own extensions to the query service played an important role in its success. The labelling, support for Wikidata's special value types (e.g., coordinates on other planets), and Wikidata-aware assistance in the Web UI improve utility and acceptance of the service.

SPARQL is Usable. Most of the Wikidata community, including developers, had no prior contact with SPARQL. An impressive amount of SPARQL-literacy has developed very quickly. There is extensive documentation and support now, including a community project *Request a Query*[22] where experts design queries for novices.

SPARQL is Affordable. Even for such a large site with a custom data model and a large global user base, the total cost of development and administrative effort, and of hardware needed for a highly reliable system is affordable.

Merging Views in RDF. Our RDF encoding merges several representations of the same data, ranging from very simple to very detailed. Combining these views in one graph proved extremely useful, since it allows queries to use just the right amount of complexity. Almost 10% of the queries we analysed for 2018 are using an RDF property that is part of the more complex, reified statement encoding.

Tangible Benefits. The effort of providing these services to the public has clearly paid off. Wikidata is benefiting hugely from the improved capabilities of data analysis and integration, used, e.g., for quality improvements. We are also seeing much use of the services in a wider public, e.g., in mobile apps and in journalism.

Slow Take-Up Among Semantic Web Researchers. We received surprisingly little input from this community so far. This is problematic since Wikidata seems to differ greatly from use cases studied before, as evident, e.g., from our analysis of SPARQL usage. Wikidata also faces hard ontology engineering challenges [16] and is a major integration point of open data.[23] It is one of the very few large knowledge graphs of a major organisation that is completely public, including its full history and all related software. We can only guess what might be holding up semantic web researchers from applying their findings in this context, and would like to invite comments from the community.

We will continue working on WDQS for the foreseeable future. At the time of this writing, ongoing development work includes, e.g., a better distribution of data updates by sharing infrastructure among the servers. The RDF encoding is also continuously extended, especially to integrate new features of Wikidata, such as the upcoming major extensions of Wikidata to support the management

---

[22] https://www.wikidata.org/wiki/Wikidata:Request_a_query.

[23] Wikidata links to over 2,500 external data collections though properties of type "External ID".

of media meta-data and lexicographic information.[24] We also plan to publish anonymised query logs based on the data we analysed, pending approval by the Wikimedia Foundation.[25] On the research side, we are exploring how to provide further ontological modelling support for the annotated graph structure of Wikidata [9,10], but we can see many other areas where Wikidata would benefit from increased research activities, and we hope that the semantic web community can still make many more contributions to Wikidata in the future.

Acknowledgements. This work was partly supported by the German Research Foundation (DFG) in CRC 912 (HAEC) and in Emmy Noether grant KR 4381/1-1 (DIAMOND).

References

1. Bielefeldt, A., Gonsior, J., Krötzsch, M.: Practical linked data access via SPARQL: the case of wikidata. In: Proceedings of WWW2018 Workshop on Linked Data on the Web (LDOW-18). CEUR Workshop Proceedings, CEUR-WS.org (2018)
2. Bischof, S., Krötzsch, M., Polleres, A., Rudolph, S.: Schema-Agnostic Query Rewriting in SPARQL 1.1. In: Mika, P., et al. (eds.) ISWC 2014, Part I. LNCS, vol. 8796, pp. 584–600. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11964-9_37
3. Bonifati, A., Martens, W., Timm, T.: An analytical study of large SPARQL query logs. Proc. VLDB Endow. **11**, 149–161 (2017)
4. Burgstaller-Muehlbacher, S., Waagmeester, A., Mitraka, E., Turner, J., Putman, T., Leong, J., Naik, C., Pavlidis, P., Schriml, L., Good, B.M., sSu, A.I.: Wikidata as a semantic framework for the Gene Wiki initiative. Database 2016, baw015 (2016)
5. Erxleben, F., Günther, M., Krötzsch, M., Mendez, J., Vrandečić, D.: Introducing wikidata to the linked data web. In: Mika, P. et al. [11], pp, 50–65
6. Florescu, D., Levy, A., Suciu, D.: Query containment for conjunctive queries with regular expressions. In: Mendelzon, A.O., Paredaens, J. (eds.) Proceedings of 17th Symposium on Principles of Database Systems (PODS 1998), pp. 139–148. ACM (1998)
7. Hernández, D., Hogan, A., Riveros, C., Rojas, C., Zerega, E.: Querying wikidata: comparing SPARQL, relational and graph databases. In: Groth, P., et al. (eds.) ISWC 2016, Part II. LNCS, vol. 9982, pp. 88–103. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46547-0_10
8. Lebo, T., Sahoo, S., McGuinness, D. (eds.): PROV-O: The PROV Ontology. W3C Recommendation, 30 April 2013. http://www.w3.org/TR/prov-o
9. Marx, M., Krötzsch, M.: SQID: Towards ontological reasoning for Wikidata. In: Nikitina, N., Song, D. (eds.) Proceedings of the ISWC 2017 Posters & Demonstrations Track. CEUR Workshop Proceedings, CEUR-WS.org, October 2017
10. Marx, M., Krötzsch, M., Thost, V.: Logic on MARS: Ontologies for generalised property graphs. In: Proceedings of 26th International Joint Conference on Artificial Intelligence (IJCAI 2017), pp. 1188–1194 (2017)

[24] See https://www.wikidata.org/wiki/Wikidata:WikiProject_Commons and https://www.wikidata.org/wiki/Wikidata:Lexicographical_data.

[25] See https://kbs.inf.tu-dresden.de/WikidataSPARQL for information on data availability.

11. Mika, P., et al.: ISWC 2014, Part I. LNCS, vol. 8796. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11964-9
12. Morsey, M., Lehmann, J., Auer, S., Stadler, C., Hellmann, S.: DBpedia and the live extraction of structured data from Wikipedia. Program: Electron. Libr. Inf. Syst. **46**(2), 157–181 (2012)
13. Picalausa, F., Vansummeren, S.: What are real SPARQL queries like? In: Virgilio, R.D., Giunchiglia, F., Tanca, L. (eds.) Proceedings of International Workshop on Semantic Web Information Management (SWIM 2011), p. 6. ACM (2011)
14. Rietveld, L., Hoekstra, R.: Man vs. machine: Differences in SPARQL queries. In: Proceedings of 4th USEWOD Workshop on Usage Analysis and the Web of Data. usewod.org (2014)
15. Rietveld, L., Hoekstra, R.: The YASGUI family of SPARQL clients. Seman. Web **8**(3), 373–383 (2017)
16. Spitz, A., Dixit, V., Richter, L., Gertz, M., Geiß, J.: State of the union: A data consumer's perspective on Wikidata and its properties for the classification and resolution of entities. In: Proceedings of ICWSM 2016 Wiki Workshop. AAAI Workshops, vol. WS-16-17. AAAI Press (2016)
17. Tanon, T.P., Vrandecic, D., Schaffert, S., Steiner, T., Pintscher, L.: From Freebase to Wikidata: The great migration. In: Bourdeau, J., Hendler, J., Nkambou, R., Horrocks, I., Zhao, B.Y. (eds.) Proceedings of 25th International Conference on World Wide Web (WWW 2016), pp. 1419–1428. ACM (2016)
18. Vandenbussche, P., Umbrich, J., Matteis, L., Hogan, A., Buil Aranda, C.: SPARQLES: monitoring public SPARQL endpoints. Seman. Web **8**(6), 1049–1065 (2017)
19. Vrandečić, D., Krötzsch, M.: Wikidata: A free collaborative knowledgebase. Commun. ACM **57**(10), 78–85 (2014)
20. Wagner, C., Graells-Garrido, E., Garcia, D., Menczer, F.: Women through the glass ceiling: gender asymmetries in wikipedia. EPJ Data Sci. **5**(1), 5 (2016)