

Benchmarking SPARQL Engines on Wikidata Queries

Peter F. Patel-Schneider

Abstract

Four open-source SPARQL engines are evaluated on three existing and one new benchmarks for queries against Wikidata, a large community-built knowledge graph with wide usage. Of the engines benchmarked—Blazegraph, MillenniumDB, QLever, and Virtuoso—QLever is the fastest. Blazegraph, which is the SPARQL engine used in the official Wikidata Query Service, is significantly slower than some other engines. All of the engines have deviations from the SPARQL standard.

1. Introduction

Wikidata [1] is a general-purpose, widely used “free and open knowledge base” that “anyone can edit” with over 117 million items as of May 2025 [2]. Wikidata is similar to other large knowledge graphs such as the Google Knowledge Graph. Wikidata is built on Wikibase [3]—open-source software that supports collaborative editing of linked knowledge graphs.

There are several ways to access Wikidata. One of these, and the most powerful, is to run SPARQL queries [4] against the RDF [5] dump of Wikidata [6]. This dump encodes all the features of Wikibase knowledge graphs into RDF, permitting advanced querying of Wikidata. Querying Wikidata in this way requires users to often know the details of the encoding and thus it may be difficult to create correct queries.

There is an official query service for Wikidata at <https://query.wikidata.org/> that runs SPARQL queries against an RDF encoding of Wikidata. The service keeps a current version of Wikidata in RDF by transforming edits made to Wikidata into RDF updates and maintaining an RDF graph corresponding to the information currently in Wikidata. The query service is at times overloaded and cannot keep up with the updates. As well, queries run in the service often time out. These are both due to the SPARQL engine that the query service uses—Blazegraph [7], which is an older, unmaintained SPARQL query engine.

This has led some users to use other services that run SPARQL queries against Wikidata, particularly a service provided by a team in the University of Freiburg, which is available at <https://qllever.cs.uni-freiburg.de/wikidata>. This service uses QLever [8]—a modern SPARQL engine. The version of Wikidata that the server uses comes from dumps of Wikidata into RDF that are provided weekly and thus may lag behind the current information in Wikidata.

The problems with the official Wikidata query service have also resulted in a significant change to it. The service no longer evaluates queries against the entire RDF dump of Wikidata. Instead there are two services, one for the scholarly publication data in Wikidata and one for the rest of Wikidata. This split was done to lessen the update load on the service, because the update stream will be split into two, and may also lessen the query load, because queries can be directed to the service that provides the information required. Queries that need information from both service will need to be rewritten to use SPARQL federation. This will put an added burden on users and may end up significantly reducing the gains because federated queries can be expensive to evaluate.

Wikidata is growing rapidly [9] and even with the split, use of Blazegraph will continue to be a problem [10]. A better SPARQL engine is needed to allow for future growth in Wikidata.

Anecdotal evidence shows that QLever is much faster than Blazegraph on many Wikidata queries. There are queries where QLever is several orders of magnitude faster than Blazegraph. Similar evidence shows that other modern SPARQL query engines are also much faster than Blazegraph on some Wikidata queries. There are some first-party benchmarks showing that modern SPARQL engines, including MillenniumDB [11] and QLever [8, 12] are faster than Blazegraph on the Wikidata RDF dump, but no large third-party comparison of the engines.

To better test the performance of modern SPARQL engines over Blazegraph an effort to benchmark the query performance of several open-source SPARQL engines on the entire Wikidata RDF dump was undertaken. This is only a part of what is needed in a replacement for Wikidata but is an important part. The analysis of the benchmark results here was designed to be more useful in determining overall performance of a service and not so much designed to determine expected performance as seen by users of the service.

Three open-source systems that were known to be able to reasonably load Wikidata RDF dumps and run SPARQL queries on them were selected. These systems are MillenniumDB [11], QLever, and Virtuoso Open Source [13]. Three existing Wikidata benchmarks were selected and a new benchmark based on Scholia [14] was created. An October 2024 RDF dump of Wikidata was loaded into each of the modern engines and Blazegraph. The benchmarks were run on all four engines and their performance is reported and analyzed here. More information about the benchmarking, including the benchmarks and all code used, is available at <https://github.com/wikius/benchmark-wikidata>.

The closest third-party study of SPARQL engines on Wikidata was performed by Lam *et al* [15]. They tested the query performance of several SPARQL engines, including an earlier version of QLever, on Wikidata, using 328 sample queries. This early version of QLever performed poorly in their testing. They did not test any of the other performant open-source SPARQL engines and QLever has undergone major improvements since their study.

2. Wikidata in RDF

There is an encoding of Wikidata into RDF, and RDF dumps of Wikidata are made weekly. There are two different kinds of dumps. One kind includes only truthful statements (triples), that is, statements without their qualifiers and other information, no deprecated statements, and normal rank statements only if there is no preferred rank statement for the same subject and predicate. The other kind of dump is a full dump that has both truthful statements and a complex encoding of all statements that includes the rank, qualifiers, and other information about each statement. As of October 2024, the full dumps of Wikidata had about 20 billion triples. The full dumps in Turtle [16] were over 100GB compressed and over 850GB uncompressed.

There are public services that evaluate SPARQL queries against full dumps of Wikidata for all four of the SPARQL engines selected. The official service, that uses Blazegraph, has the most up-to-date information, generally lagging by only a few seconds as updates to Wikidata are processed and then incorporated into its RDF graph. The QLever service uses similar information and also lags only slightly. The QLever service lags by around a week, as it can process the weekly dumps in well under a day. The MillenniumDB service uses the weekly dumps and thus lags by somewhat over a week. The data used by the Virtuoso service is only updated irregularly and can lag by months.

3. The Benchmarks

Three existing benchmarks were selected. These were chosen to provide a varied set of queries with different selection criteria and difficulty.

WGPD [17] consists of 50 instantiations of 17 simple¹ query patterns. A pattern is, in essence, a small graph whose nodes are shared variables in a set of SPARQL BGPs. Each pattern is instantiated by picking Wikidata properties for each edge and constructing the BGPs, which are then expanded into a full SPARQL query. Finally a `LIMIT 1000` is added, resulting in 850 SPARQL queries.

WDBench [18] consists of query fragments from the anonymized Wikidata SPARQL Logs² evaluated by the Wikidata Query Service in 2017 and 2018 [19]. The queries used were chosen from those that had timed out. The BGPs, property paths, and some other portions of the queries were extracted and categorized into those with a single BGP (280 queries), those with multiple BGPs (681 queries), those with `OPTIONAL` clauses (498 queries), those with property paths (660 queries), and those that did not fit into any of the above categories (539 queries). Each of these five sets of query fragments are treated as a single benchmark here.

These query fragments have to be expanded into full queries by adding the `SELECT` portion. The query fragments do not retain `FILTER` or other limits on the size of the answer set and can return very large answer sets. The original benchmarking thus added a `LIMIT 100000` to limit the number of answers. To stress modern SPARQL engines this benchmarking arbitrarily uses instead `LIMIT 10000000`.

WDQS [12] consists of a set of 298 queries extracted from Wikidata Query Service logs. This benchmark was used to evaluate the comparative performance of several SPARQL engines. Several of the queries return hundreds of millions of answers. For these a `LIMIT 10000000` is added to the query here.

A new benchmark was created from the queries used by the Scholia [14] interface to Wikidata. This interface is designed to show information related to scholarly articles. A request for Scholia information is in the form of one, usually, but sometimes more, Wikidata identifiers. The class(es) of these identifier(s) in Wikidata are determined and a template HTML document is selected based on the class(es). There is a default template if there is no template specifically for the type(s). The template document has sections that are replaced by information constructed from the results of SPARQL queries constructed by inserting the identifier(s) in a query template.

For example, a Scholia request for Wikidata identifier `Q39246`, the item for Richard Feynman, would query Wikidata to find that the item with this identifier is a human and use the author document template to determine what queries to construct and how to create the HTML document partly shown in Figure 1.

Some of the Scholia queries are difficult for the Wikidata Query Service to evaluate and queries time out, resulting in documents with errors in them. Further, running these difficult queries puts a significant load on the Wikidata Query Service. The group maintaining Scholia is thus interested in determining whether a different SPARQL engine would do better.

The advantage of using Scholia to construct a benchmark is that many queries can be constructed from the templates. However, there are only about 375 query templates, and some of the templates are similar to each other, so there is not a wide variety of different queries. Another problem with using Scholia query templates for benchmarking is that they use extensions to SPARQL that are specific to Blazegraph.

The Scholia benchmark was constructed by determining the query templates for 33 different classes. The query templates were then turned into standard SPARQL by expanding named queries replacing the Wikidata Label Service with query fragments to determine English-language labels, and making a few other, minor modifications. For each of these classes, five items belonging to the class were determined. In a few cases these items were selected by hand but in most cases the items are the first five answers to a query that returned instances of the class that had values

¹A simple query here is one with only one SPARQL construct or a small number of similar SPARQL constructs. A complex query has several different SPARQL constructs.

²The logs of the Wikidata Query Service are considered to be private as they might contain personally-identifying information so constructing public benchmarks from them is not easy.

author / Q39246

Improve data

Richard Feynman (Q39246)

Richard Phillips Feynman was an American theoretical physicist. He is best known for his work in the path integral formulation of quantum mechanics, the theory of quantum electrodynamics, the physics of the superfluidity of supercooled liquid helium, and in particle physics, for which he proposed the parton model. For his contributions to the development of quantum electrodynamics, Feynman received the Nobel Prize in Physics in 1965 jointly with Julian Schwinger and Shin'ichirō Tomonaga. (Read more on English Wikipedia)

Related: Max Delbrück · Alfred Sturtevant · Victor Weisskopf · Steven Pinker · Elliott H. Lieb · Clinton Hart Merriam · Simon Baron-Cohen · Josiah Willard Gibbs · Edward Feigenbaum · Jonathan A. Eisen

http://richardfeynman.com/

Table of Contents

- List of publications
 - Number of publications per year
 - Number of pages per year
- Topics
 - Topic scores
 - Topics of authored works
 - Topics-works matrix
- Use
- Venue statistics
- Review statistics
- Co-author graph
- Co-author map
- Other locations
- Timeline
- Academic tree
- Citation statistics
 - Most cited works
 - Citations by year
 - Citing authors
- Associated images
- Events



List of publications RSS

Reload

Show 10 entries

Search:

Date	Work	Type	Pages	Venue	Authors
2017-01-01	Física Em 12 Lições (Em Portugues do Brasil)	version, edition or translation			Richard Feynman
2013-01-01	Feynmanovy přednášky z fyziky: revidované vydání s řešenými příklady	version, edition or translation	435		Richard Feynman, Matthew Sands, Robert B. Leighton
2010-03-03	Richard Feynman: Physics is fun to imagine	TED talk			Richard Feynman
2005-04-05	Perfectly Reasonable Deviations from the Beaten Track	version, edition or translation			Richard Feynman
2005-04-05	Perfectly Reasonable Deviations from the Beaten Track	letter collection			Richard Feynman
2004-01-01	Sei pezzi meno facili	version, edition or translation	223		Richard Feynman
2003-01-01	Radost z poznání	version, edition or translation	332		Richard Feynman
2001-01-01	Neobyčejná teorie světa a látky: kvantová elektrodynamika	version, edition or translation	158		Richard Feynman
2001-01-01	To nemyslíte vážně, pane Feynmane!	version, edition or translation	302		Richard Feynman
2000-01-01	Sei pezzi facili	version, edition or translation	212		Richard Feynman

Wikidata Query Service

author: list-of-publications.sparql

Showing 1 to 10 of 74 entries

Previous 1 2 3 4 5 ... 8 Next

Number of publications per year

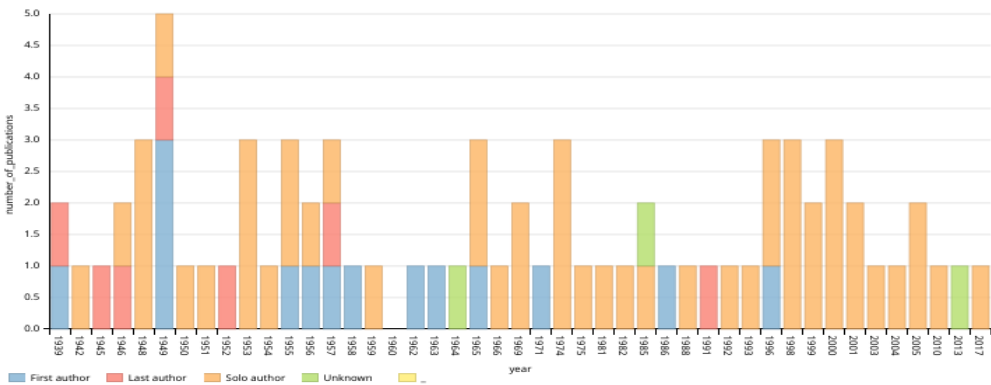


Figure 1: Part of Scholia Page for Richard Feynman (Q39246)

for properties uses in one or more of the query templates.

Some of the templates are complex. For example, here is a query template for the author document after conversion to standard SPARQL, edited to present better. The `target:` prefix is instantiated with the URL for the Wikidata identifier being used.

```
SELECT ?year (count(?work) AS ?numb_of_publs) ?role WHERE {
  { SELECT (str(?year_) AS ?year) (0 AS ?pp) ("_" AS ?role) WHERE {
    ?year_item wdt:P31 wd:Q577 .
    ?year_item wdt:P585 ?date .
    BIND(year(?date) AS ?year_)
    { SELECT (min(?year_) AS ?e_year) (max(?year_) AS ?l_year) WHERE {
      ?work wdt:P50 target: .
      ?work wdt:P577 ?publ_date .
      BIND(year(?publ_date) AS ?year_) } }
    BIND(year(now())+1 AS ?next_year)
    FILTER (?year_ >= ?e_year && ?year_ <= ?l_year) }
  } UNION {
    { SELECT ?work (min(?years) AS ?year)
      (count(?coauthors) AS ?numb_of_authors) ?author_numb WHERE {
        ?work (p:P50|p:P2093) ?author_statement .
        ?author_statement ps:P50 target: .
        optional { ?author_statement pq:P1545 ?author_numb . }
        ?work (wdt:P50|wdt:P2093) ?coauthors .
        ?work wdt:P577 ?dates .
        BIND(str(year(?dates)) AS ?years) . }
      GROUP BY ?work ?author_numb }
    BIND(COALESCE(if(?numb_of_authors = 1, 'Solo author',
      if(xsd:integer(?author_numb) = 1, 'First author',
      if(xsd:integer(?author_numb) = ?numb_of_authors,
      'Last author', 'Middle author'))), 'Unknown') AS ?role)
  } } GROUP BY ?year ?role ORDER BY ?year
```

A few queries returned large answer sets, which is not useful when constructing the final document, so `LIMIT` clauses were added. A few queries had errors, which caused them to return incorrect answer sets, and were fixed. All these changes were sent to the Scholia repository and have been incorporated into it.

A query run then consists of instantiating each query template with each item and evaluating the resultant query.

4. Running the Benchmarks

The benchmarks were all run on a machine with a Ryzen 9950X CPU, 192GB of main memory, and fast NVMe SSD drives running the Fedora Linux distribution. MillenniumDB, QLever, and Virtuoso were downloaded from their open-source repositories, using the version current as of 05 March 2025 for MillenniumDB, 22 March 2025 for QLever, and 19 March 2025 for Virtuoso.

They were compiled using scripts from the repositories. Blazegraph is run from a docker image for the current version of Blazegraph because of issues with Java. This may slow down Blazegraph by up to 10%, but probably only slows Blazegraph down a few percent. This possible penalty does not affect the main conclusions of the evaluation.

Wikidata RDF dumps from late October 2024 were loaded into all four engines using settings determined in consultation from developers where possible. Loading was relatively easy for MillenniumDB, QLever, and Virtuoso and took less than a day for each, with QLever being fastest at about 4.5 hours. Loading the dumps into Blazegraph took over 10 days and the first try failed, probably due to a bug related to concurrent access to some data. As loading into Blazegraph was difficult no attempt was made to use newer dumps of Wikidata.

Settings for the engines during benchmarking were determined in consultation from developers where possible and set up so that about 3/4 of main memory was used by the engine. This is more memory than is commonly allowed in the public Wikidata services but was chosen to better reflect expected memory growth in the near future. The engines are allowed to use multiple threads, but all except Blazegraph are only lightly threaded when querying.

Each query is run with a 10-minute timeout. This is larger than most public Wikidata services, which generally use a 1-minute timeout, and was chosen to see behavior of the engines on a longer timeframe and to provide some indication about behavior in future with faster computers.

Each benchmark run is performed from a cold start, with system caches emptied, and timed after any startup done by the engine. This means that any engine that defers startup until the first query is evaluated will be slightly penalized. No engine spends more than a few seconds on startup and almost all runs took multiple minutes or even hours so the penalty is insignificant. This also means that any adaptation by the engine to the data in Wikidata or normal queries is considered to be part of the benchmark timing.

Then the multiple queries in each benchmark run are evaluated in succession, with no attempt to clear any cached information between queries. The input and output formats were the same for each engine. The benchmark runs, with the exception of the Scholia benchmark, had hundreds of queries. This much better simulates the situation with a query service than attempting to remove caches.

The controlling program is run on the same computer as the engine. It generally took minimal resources, except when the queries return very large answer sets and receiving the answer set takes some resources on the computer. The processing power required for this does not impact the benchmarking as there are always many threads unused. The memory taken to store the result does have some impact, competing for main memory with the system disk cache. Running the controlling program on the same computer as the engines, however, eliminates the overhead in both time and memory to send the results to a different computer. This overhead can be considerable, even when both computers are in the same local network, so running the controlling program on the same computer was deemed better.

The controlling program records the elapsed time between sending the query to the engine and receiving the answers from the engine. This includes any time to transmit the information between the controlling program and the engine, but not all engines provide internal timing information. If this time is longer than the maximum time the query evaluation is determined to have timed out. The output from the engine is checked for any reported errors. For each successful query one piece of information about the answer set is recorded. For queries with multiple or no answers the number of answers is recorded. For queries with one answer the value of the first variable in the query is recorded.

The benchmarking process lasted from late October 2024 to late March 2025. Benchmarks were run multiple times to remove problems in the early runs and as new versions of some of the engines were made available. Initial results of the benchmarks were publicized and made public at https://www.wikidata.org/wiki/Wikidata:Scaling_Wikidata/Benchmarking and newly-discovered bugs and anomalies were communicated to the teams responsible for the engine involved, resulting in new versions of both QLever and MillenniumDB being available. The results here are for the latest runs for each engine.

Each set of queries for the existing benchmarks was run three times—once as described above, once with the query modified to only return the count of the number of answers, and once with the query modified to return only distinct answers. The second run was performed to eliminate the overhead of transmitting large answer sets. The third run was performed to help see how many times Virtuoso returned incorrect answer sets for transitive path queries. The Scholia benchmark queries were only run unmodified, after the changes described above, as most of them only returned a few answers with no duplicates. In a few cases the engine terminated when evaluating a query. These cases are marked and the engine restarted with the next query.

WDQS Benchmark Statistics, Unadjusted Timings										
Engine	Count	min	q1	q2	q3	max	Mean	Error	Timeout	Diverge
Blazegraph	298	11	88	511	6155	600018	12560	31	1	14
MillenniumDB	298	1	31	588	24176	602338	103271	0	43	5
QLever	298	2	26	103	559	301655	4583	3	0	12
Virtuoso	298	1	68	461	3264	600754	14645	13	2	30
WDQS Benchmark Statistics, Adjusted Timings										
Engine	Count	min	q1	q2	q3	max	Mean	Error	Timeout	Diverge
Blazegraph	298	17	89	520	6403	60000	10236	21	16	13
MillenniumDB	298	1	31	588	24176	60000	15482	0	64	3
QLever	298	2	26	103	559	60000	2290	1	6	12
Virtuoso	298	2	80	577	4503	60000	8506	12	13	28
WDQS Benchmark Statistics, Counted Answers, Unadjusted Timings										
Engine	Count	min	q1	q2	q3	max	Mean	Error	Timeout	Diverge
Blazegraph	298	18	97	465	6803	600037	28682	23	6	14
MillenniumDB	298	1	31	571	23744	600987	102989	0	43	4
QLever	298	2	22	78	445	298940	4158	3	0	12
Virtuoso	298	1	52	381	2632	600735	14130	12	2	26
WDQS Benchmark Statistics, Counted Answers, Adjusted Timings										
Engine	Count	min	q1	q2	q3	max	Mean	Error	Timeout	Diverge
Blazegraph	298	24	98	466	7464	60000	10683	13	24	14
MillenniumDB	298	1	31	571	23744	60000	15341	0	64	3
QLever	298	2	22	78	445	60000	1908	1	5	12
Virtuoso	298	2	57	449	3318	60000	7908	11	13	24
WDQS Benchmark Statistics, Distinct Answers, Unadjusted Timings										
Engine	Count	min	q1	q2	q3	max	Mean	Error	Timeout	Diverge
Blazegraph	298	12	83	469	5333	600023	12220	32	1	14
MillenniumDB	298	2	30	592	24094	602783	103326	0	43	3
QLever	298	2	26	101	558	297839	4730	3	0	13
Virtuoso	298	1	69	501	3321	600741	14266	12	2	23
Engine	Count	min	q1	q2	q3	max	Mean	Error	Timeout	Diverge
Blazegraph	298	18	84	479	5777	60000	10177	19	18	14
MillenniumDB	298	2	30	592	24094	60000	15510	0	64	1
QLever	298	2	26	101	558	60000	2461	1	7	13
Virtuoso	298	3	82	589	4489	60000	8251	11	14	22

Table 1
Statistics for WDQS Benchmark, timings in milliseconds

5. Results

For each engine the results of each set of queries were analyzed to compute the minimum and each quartile elapsed times, the mean elapsed time, the number of timeouts, the number of errors encountered, and the number of times the retained answer information diverges from a single mode for the four engines. The arithmetic mean is used to show how the queries would consume time on servers as opposed to show expectations by users, where geometric means are normally used.

As well, adjusted statistics were computed, where elapsed time is capped at 60 seconds, with times at least this long counting as a timeout, and any error counted as 60 seconds. This adjusted time is computed mostly to penalize engines that had many errors, but also to more closely mirror times in current public services.

The statistics for all three variations of the WDQS benchmark with both unmodified and adjusted timings are shown in Table 1. On this benchmark QLever is significantly the fastest for all three variations, no matter whether the timings are adjusted or not. The relative difference in

speed between QLever and MillenniumDB, the slowest engine, is about 25 times for unadjusted timings and about 7 times for adjusted timings. QLever never takes the full 600 seconds for any query and only takes more than 60 seconds for a few, whereas MillenniumDB times out on about 1 in 7 queries.

Blazegraph has quite a few errors on this benchmark, mostly due to running out of memory. Virtuoso has a few errors mostly due to refusal to evaluate the query due to high estimated times, incorrect syntax processing, or issues with transitive paths. The Qlever errors are due to running out of memory. Each engine diverges from a common mode in a few cases. Virtuoso diverging the most, mostly due to invalid duplicates from transitive paths. Most of the divergences for MillenniumDB appear to be from a bug in embedded query processing. The divergences for Blazegraph appear to be mostly from the Blazegraph loading process removing some triples related to Wikidata labels. Some divergences, and most of the QLever divergences, are due to extra processing of numeric and GeoSPARQL values.

6. Summarization and Analysis

These statistics were further processed to produce summaries, removing the some of the statistical information to show combined performance of each engine on the benchmarks, with the five components of WDBench shown separately. This allows the timings and issues for each engine to be shown in a smaller format. For the existing benchmarks six blocks of information are generated—for adjusted and unadjusted on each way the benchmarks have been run. This information is shown in Tables 2 and 3. For the Scholia benchmark again both unadjusted and adjusted information is shown in Table 4, but only for some of the query classes.

Existing Benchmarks The summaries for the existing benchmarks show that all the engines have divergences from a single mode, likely indicating deviations from the SPARQL standard. Penalizing for divergences was not done, because it was not always certain that divergences are incorrect answers. The detailed results were examined and some queries run outside the benchmarking process to determine some reasons for these divergences.

The large number of divergences for Virtuoso are mostly due to two known issues. Virtuoso returns duplicates from transitive path matching where the standard requires no duplicates. Virtuoso also silently only produces at most 1048576 answers for any query. In the WDBench benchmarks this produces over one thousand divergences, with the second cause producing over 60% of the divergences, as shown by the statistics for when only counts are returned. This large number of divergences should be taken into account when considering Virtuoso.

The divergences for MillenniumDB appear to be mostly due to not returning duplicates for alternatives in property paths. Other divergences for MillenniumDB appear to be from a bug in embedded query processing.

Many divergences for Blazegraph are from the Blazegraph loading process removing some triples related to Wikidata labels, and are thus not a problem with Blazegraph itself. Other divergences for Blazegraph come from an incorrect ordering of `DISTINCT` and `LIMIT` processing.

QLever and possibly other engines transform numeric RDF literals into internal data, which does not conform to the RDF and SPARQL standards. For example, `"1"^^xsd:integer` and `"01"^^xsd:integer` incorrectly become the same RDF node. This causes the majority of the divergences for QLever. GeoSPARQL datatypes were also a source of divergences.

The summaries for the existing benchmarks also show a considerable number of errors, so penalizing the engines for errors is appropriate.

Most of the errors for QLever result from running out of memory. QLever query processing appears to trade off space for time, and QLever can request large amounts of memory for queries, thus running out of space. Optional clauses and requiring results to be distinct appear to affect this tradeoff, so much that QLever runs out of memory very often when either of these constructs is present and resultant penalty in the adjusted timings is significant.

Benchmark	Blazegraph	MillenniumDB										QLever										Virtuoso									
Unmodified queries, Unadjusted timings																															
	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	
WGFB	644	632	2	0	0	21	9	0	0	0	12	0	0	0	0	978	966	0	1	0	978	966	0	1	0	978	966	0	1	0	
single BGPs	3103	2602	0	0	15	501	0	0	0	0	769	268	0	0	5	1726	1225	0	0	51	1726	1225	0	0	51	1726	1225	0	0	51	
multiple BGPs	9653	5023	10	0	0	5442	812	0	0	0	4630	0	6	0	3	7608	2978	73	4	383	7608	2978	73	4	383	7608	2978	73	4	383	
optionals	20851	3833	23	3	3	25364	8346	0	10	0	17018	0	127	0	8	25566	8548	50	6	290	25566	8548	50	6	290	25566	8548	50	6	290	
paths	25277	21001	21	12	4	7293	3017	1	4	18	4276	0	5	2	0	5691	1415	49	2	422	5691	1415	49	2	422	5691	1415	49	2	422	
others	96114	92944	15	54	23	82337	79167	0	55	3	3170	0	2	0	8	18239	15069	20	9	320	18239	15069	20	9	320	18239	15069	20	9	320	
WDQS	12560	7977	31	1	14	103271	98688	0	43	5	4583	0	3	0	12	14645	10062	13	2	30	14645	10062	13	2	30	14645	10062	13	2	30	
TOTALS	168202	134012	102	70	59	224229	190039	1	112	26	34458	268	143	2	36	74453	40263	205	24	1496	74453	40263	205	24	1496	74453	40263	205	24	1496	
Unmodified queries, Adjusted timings																															
	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	
WGFB	501	489	1	1	0	21	9	0	0	0	12	0	0	0	0	342	330	0	1	0	342	330	0	1	0	342	330	0	1	0	
single BGPs	2073	1572	0	3	14	501	0	0	0	0	769	268	0	0	5	1726	1225	0	0	51	1726	1225	0	0	51	1726	1225	0	0	51	
multiple BGPs	9302	4573	6	11	0	4784	55	0	2	0	4729	0	6	0	3	10415	5686	73	7	381	10415	5686	73	7	381	10415	5686	73	7	381	
optionals	16885	4385	19	22	2	12500	0	0	26	0	21201	8701	115	29	8	15137	2637	50	36	260	15137	2637	50	36	260	15137	2637	50	36	260	
paths	8665	6336	0	64	3	2708	379	1	11	18	2329	0	3	8	0	6686	4357	43	15	415	6686	4357	43	15	415	6686	4357	43	15	415	
others	20716	18327	9	133	9	15295	12906	0	100	2	2389	0	1	4	8	7767	5378	19	28	303	7767	5378	19	28	303	7767	5378	19	28	303	
WDQS	10236	7946	21	16	13	15482	13192	0	64	3	2290	0	1	6	12	8506	6216	12	13	28	8506	6216	12	13	28	8506	6216	12	13	28	
TOTALS	68378	43628	56	250	41	51291	26541	1	203	23	33719	8969	126	47	36	50579	258297	197	100	1438	50579	258297	197	100	1438	50579	258297	197	100	1438	
Counted results, Unadjusted timings																															
	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	
WGFB	1078	1066	0	1	0	21	9	0	0	0	12	0	0	0	0	762	750	0	1	0	762	750	0	1	0	762	750	0	1	0	
single BGPs	1619	1611	0	0	15	17	9	0	0	0	8	0	0	0	4	116	108	0	0	1	116	108	0	0	1	116	108	0	0	1	
multiple BGPs	18111	17571	7	4	5	1300	760	0	0	0	540	0	6	0	8	4122	3582	0	4	0	4122	3582	0	4	0	4122	3582	0	4	0	
optionals	46248	31697	192	4	7	19975	5424	0	10	0	14551	0	127	0	9	31140	16589	1	14	0	31140	16589	1	14	0	31140	16589	1	14	0	
paths	23206	19337	15	15	4	6766	2897	1	4	20	3869	0	5	2	3	5065	1196	49	3	365	5065	1196	49	3	365	5065	1196	49	3	365	
others	97506	95490	7	56	26	81011	78995	0	54	3	2016	0	2	0	8	17441	15425	16	9	240	17441	15425	16	9	240	17441	15425	16	9	240	
WDQS	28682	24524	23	6	14	102989	98831	0	43	4	4158	0	3	0	12	14130	9972	12	2	26	14130	9972	12	2	26	14130	9972	12	2	26	
TOTALS	216450	191296	244	86	71	212079	186925	1	111	27	25154	0	143	2	44	72776	47622	78	33	632	72776	47622	78	33	632	72776	47622	78	33	632	

Table 2

Summary Information for Existing Benchmarks (Part 1), times in milliseconds

Slow=Difference from fastest time, Err=Error, TO=Timeout, Div=Diverge

Benchmark	Blazegraph	MillenniumDB										QLever					Virtuoso				
		Counted results, Adjusted timings																			
		Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div
WGPB		443	431	0	1	0	21	9	0	0	0	12	0	0	0	0	126	114	0	1	0
single BGPs		1619	1611	0	0	15	17	9	0	0	0	8	0	0	0	4	116	108	0	0	1
multiple BGPs		14457	13814	7	18	5	681	38	0	2	0	643	0	6	0	8	735	92	0	6	0
optionals		33758	26254	108	105	7	7504	0	0	24	0	19066	11562	115	28	9	7769	265	1	46	0
paths		8627	6695	10	42	3	2227	295	1	10	20	1932	0	3	8	3	5830	3898	43	11	363
others		22455	21212	3	140	12	14154	12911	0	100	2	1243	0	1	4	8	6098	4855	15	28	229
WDQS		10683	8775	13	24	14	15341	13433	0	64	3	1908	0	1	5	12	7908	6000	11	13	24
TOTALS		92042	78792	141	330	56	39945	26695	1	200	25	24812	11562	126	45	44	28582	15332	70	105	617
Distinct results, Unadjusted timings																					
		Counted results, Adjusted timings																			
		Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div
WGPB		626	611	2	0	0	22	7	0	0	0	15	0	0	0	0	839	824	0	1	0
single BGPs		3873	3114	20	0	15	759	0	0	0	0	769	10	0	0	5	1769	1010	0	0	51
multiple BGPs		16109	9066	363	0	0	7659	616	0	0	0	26657	19614	241	2	2	7043	0	45	3	410
optionals		21980	0	275	3	4	27913	5933	0	10	0	29291	7311	134	0	8	29383	7403	27	8	308
paths		26716	20690	65	12	2	8318	2292	1	4	1	22472	16446	9	19	1	6026	0	50	2	111
others		96895	82301	99	53	4	83032	68438	0	56	0	14594	0	42	3	11	19519	4925	19	9	111
WDQS		12220	7490	32	1	14	103326	98596	0	43	3	4730	0	3	0	13	14266	9536	12	2	23
TOTALS		178419	123272	856	69	39	231029	175882	1	113	4	98528	43381	429	24	40	78845	23698	153	25	1014
Distinct results, Adjusted timings																					
		Counted results, Adjusted timings																			
		Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div
WGPB		502	487	1	1	0	22	7	0	0	0	15	0	0	0	0	203	188	0	1	0
single BGPs		5908	5149	19	3	14	759	0	0	0	0	769	10	0	0	5	1769	1010	0	0	51
multiple BGPs		36675	29687	359	8	0	6988	0	0	2	0	29916	22928	231	64	2	8328	1340	45	7	407
optionals		37221	23812	266	19	4	14907	1498	0	26	0	25021	11612	108	66	8	13409	0	27	38	275
paths		11777	8613	31	65	2	3164	0	1	11	0	4435	1271	3	32	1	6801	3637	44	15	105
others		27408	19733	85	132	4	15976	8301	0	100	0	8827	1152	37	26	11	7675	0	18	28	105
WDQS		10177	7716	19	18	14	15510	13049	0	64	1	2461	0	1	7	13	8251	5790	11	14	22
TOTALS		129668	95197	780	246	38	57326	22855	1	203	1	71444	36973	380	195	40	46436	11965	145	103	965

Table 3

Summary Information for Existing Benchmarks (Part 2), times in milliseconds

Slow=Difference from fastest time, Err=Error, TO=Timeout, Div=Diverge

Benchmark	Blazegraph				MillenniumDB				QLever				Virtuoso							
	Unadjusted timings																			
	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div
author	8530	8058	15	0	0	130689	130217	10	19	6	3006	2534	10	0	5	472	0	27	0	0
chem-class	102438	101993	5	5	0	219533	219088	0	10	0	445	0	5	0	0	590	145	10	0	0
chem-elem	150137	149549	0	5	0	150029	149441	0	5	0	588	0	0	0	0	2257	1669	0	0	0
complex	164211	163961	5	5	0	158453	158203	0	5	0	300709	300459	0	10	0	250	0	5	0	0
event-series	102877	101999	20	5	0	312928	312050	0	24	0	44055	43177	7	3	0	878	0	25	0	0
project	75112	74861	0	5	0	142475	142224	0	7	0	370	119	0	0	0	251	0	15	0	1
property	243533	241116	0	4	0	16145	13728	0	0	0	2417	0	0	0	0	3979	1562	0	0	0
protein	61545	61417	5	2	0	360166	360038	0	15	0	9876	9748	9	0	0	128	0	20	0	0
topic	242863	239361	13	33	0	424204	420702	0	59	0	4552	1050	2	0	0	3502	0	10	0	10
venue	94222	93727	3	13	0	219271	218776	0	39	0	495	0	0	0	1	3495	3000	7	0	0
wikiproject	148828	148593	20	15	0	285685	285450	10	30	0	235	0	10	0	0	2317	2082	15	0	1
work	9426	9173	10	0	0	94370	94117	15	6	1	1830	1577	11	0	0	253	0	10	0	4
TOTALS	1632775	1611180	145	107	5	4929571	4907976	45	351	10	548662	527067	93	13	17	51456	29861	221	1	38
	Adjusted timings																			
	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div	Mean	Slow	Err	TO	Div
author	8268	2313	15	0	0	23387	17432	10	19	6	5955	0	10	0	5	15172	9217	27	0	0
chem-class	20097	9669	5	5	0	22012	11584	0	10	0	10428	0	5	0	0	20407	9979	10	0	0
chem-elem	15130	14542	0	5	0	15008	14420	0	5	0	588	0	0	0	0	2257	1669	0	0	0
complex	30046	14958	5	5	0	23426	8338	0	5	0	30499	15411	0	10	0	15088	0	5	0	0
event-series	30033	14061	20	5	0	32411	16439	0	24	0	15972	0	7	3	0	30807	14835	25	0	0
project	7609	7239	0	5	0	15015	14645	0	7	0	370	0	0	0	0	22602	22232	15	0	1
property	19950	17533	0	4	0	12894	10477	0	0	0	2417	0	0	0	0	3979	1562	0	0	0
protein	16903	0	5	2	0	36119	19216	0	15	0	24161	7258	9	0	0	48001	31098	20	0	0
topic	39134	35204	13	33	0	48812	44882	0	59	0	3930	0	2	0	0	9944	6014	10	0	10
venue	17304	16809	3	13	0	27053	26558	0	39	0	495	0	0	0	1	6999	6504	7	0	0
wikiproject	32586	23153	20	15	0	41684	32251	10	30	0	9433	0	10	0	0	15553	6120	15	0	1
work	8677	0	10	0	0	18034	9357	15	6	1	11189	2512	11	0	0	8824	147	10	0	4
TOTALS	353023	203830	145	107	5	637292	488099	45	351	10	236235	87042	93	13	17	374081	224888	221	1	38

Table 4

Summary for Scholia Benchmark (Selected Templates), times in ms

Slow=Difference from fastest time, Err=Error, TO=Timeout, Div=Diverge

Blazegraph also often runs out of memory, with a significant resultant penalty. Blazegraph also regularly reports errors in access to its internal data structures.

Virtuoso first estimates the time it would take to evaluate a query and refuses to run the query if this estimate is out of bounds. Unfortunately, the query estimator regularly produces unbelievable estimates resulting Virtuoso frequently refusing to run a query. The penalty for these errors is significant.

MillenniumDB and QLever are not complete implementations of SPARQL and a few queries contain constructs that they do not handle. Virtuoso also reports a few queries that it cannot handle, mostly relating to transitive property paths. MillenniumDB has very few errors overall. This does need to be balanced against the large number of timeouts for MillenniumDB.

The timings show QLever as the fastest engine for the existing benchmarks, except for the versions with distinct results where Virtuoso is fastest. Otherwise Virtuoso is the second-fastest, but this needs to be balanced with the large number of divergences for Virtuoso.

In unadjusted timings, MillenniumDB is the slowest overall. MillenniumDB is fast when there are simple queries or limited answers but is very slow when there are complex queries (WDBench others and WDQS). It thus appears that MillenniumDB is speedy on atomic operations but does not do a good job of producing good query plans for complex queries. When timings are adjusted to account for errors, Blazegraph is the slowest by a large margin over QLever and Virtuoso.

Scholia benchmark The Scholia benchmark also shows the need to adjust timings to account for errors. In the unadjusted timings Virtuoso is the fastest, but it has the most errors. When timings are adjusted, Virtuoso sinks to third and QLever is fastest by a ratio of about two-thirds over Blazegraph. MillenniumDB is the slowest on this benchmark, timing out on many of the queries, and is about 2.7 times slower than QLever. The slowness of MillenniumDB is likely due to the complex queries in the benchmark.

Almost all of the 145 errors for Blazegraph in the Scholia benchmark are due to running out of memory. MillenniumDB produces no output for its 45 errors so the cause cannot be determined, but it is likely that the cause for most of them is unrecognized answers from service calls. Close to half of the 93 errors for QLever are due to running out of memory, with most of the rest due to unrecognized answers from service calls and a few due to unimplemented syntax. Of the 221 errors for Virtuoso, over half are due to unimplemented syntax and most of the rest due to high estimated execution times with most of these estimated times being excessive or abnormal.

There are some divergences in the answers from the engines. As before, Virtuoso has the most divergences, with Blazegraph having the fewest. The reason for most of these divergences is unknown due to the complex nature of the queries. Some divergences appear to be due to the reasons identified above.

7. Summary and Recommendation

QLever is the fastest engine overall, but is slower for distinct answers. Virtuoso is fast but diverges the most by far mostly due to several known causes. MillenniumDB and Blazegraph are the slowest. MillenniumDB is fast on simple queries, but slow on complex queries.

None of the engines are free of errors or divergences, even Blazegraph. That Blazegraph has divergences is a bit surprising because Blazegraph was in use for the official Wikidata Query Service while it was still being maintained. Both QLever and MillenniumDB are under active development, which should improve their performance and reduce their errors and divergences.

From the results in these benchmarks, a Wikidata Query Service based on QLever would be significantly faster and produce more answers and fewer errors than one based on Blazegraph. QLever now appears to be a viable replacement for Blazegraph in the official Wikidata Query Service as it has recently been extended to allow its RDF graph to be updated while it is running.

Acknowledgments: This work was partly supported by a grant from Wikimedia Switzerland.

References

- [1] D. Vrandečić, M. Krötzsch, Wikidata: A free collaborative knowledgebase, *C. of the ACM* 57 (2014) 78–85.
- [2] Wikidata, Wikidata main page, https://www.wikidata.org/wiki/Wikidata:Main_Page, 2025. Accessed 30 April 2025.
- [3] Wikimedia Deutschland, Wikibase, <https://wikiba.se/>, 2025. Accessed 30 April 2025.
- [4] SPARQL, SPARQL 1.1 query language, W3C Recommendation, <https://www.w3.org/TR/sparql11-query/>, 2013.
- [5] Richard Cyganiak and David Wood and Markus Lanthaler, RDF 1.1 concepts and abstract syntax, W3C Recommendation, <https://www.w3.org/TR/rdf11-concepts/>, 2014.
- [6] Wikidata:RDF, Wikidata:RDF, <https://www.wikidata.org/wiki/Wikidata:RDF>, 2025. Accessed 30 April 2025.
- [7] Blazegraph, Welcome to Blazegraph, blazegraph.com, 2020. Accessed 23 July 2024.
- [8] H. Bast, B. Buchhold, QLever: A query engine for efficient SPARQL+text search, in: *CIKM '17: ACM Conference on Information and Knowledge Management*, 2017.
- [9] G. Lederrey, L. Pintscher, D. Causse, Wikidata query service: Where are we? Where is it going?, *Data Reuse Days 2025*, <https://docs.google.com/presentation/d/1DHxnmkZKwly9AKONOJtvfTk6ls6DBw1Ab6gHdODM5XA>, 2025.
- [10] Wikidata SPARQL Query Service Backend Update, Wikidata SPARQL query service backend update, https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/WDQS_backend_update, 2025. Accessed 30 April 2025.
- [11] D. Vrgoč, C. Rojas, R. Angles, M. Arenas, D. Arroyuelo, C. Buil-Aranda, A. Hogan, G. Navarro, C. Riveros, J. Romero, MillenniumDB: An open-source graph database system, *Data Intelligence* 5 (2023).
- [12] H. Bast, QLever performance evaluation and comparison to other SPARQL engines, <https://github.com/ad-freiburg/qlever/wiki/QLever-performance-evaluation-and-comparison-to-other-SPARQL-engines>, 2025. Accessed 8 May 2025.
- [13] Virtuoso, Virtuoso open-source edition, <https://vos.openlinksw.com/owiki/wiki/VOS>, 2024. Accessed 30 April 2025.
- [14] F. Å. Nielsen, D. Mietchen, E. Willighagen, Scholia and scientometrics with Wikidata, in: *Scientometrics 2017*, 2017, pp. 237–259. URL: <https://arxiv.org/pdf/1703.04222>.
- [15] A. N. Lam, B. Elvesæter, F. Martin-Recuerda, in: *The Semantic Web: 20th International Conference, ESWC 2023*, 2023, pp. 679–696. doi:http://dx.doi.org/10.1007/978-3-031-33455-9_40.
- [16] RDF 1.1 Turtle, RDF 1.1 Turtle, W3C Recommendation, <https://www.w3.org/TR/turtle/>, 2014.
- [17] A. Hogan, C. Riveros, C. Rojas, A. Soto, A worst-case optimal join algorithm for SPARQL, in: *Proceedings of the 18th International Semantic Web Conference (ISWC)*, 2019.
- [18] R. Angles, C. B. Aranda, A. Hogan, C. Rojas, D. Vrgoč, Wdbench: A wikidata graph query benchmark, in: U. Sattler, A. Hogan, M. Keet, V. Presutti, J. P. A. Almeida, H. Takeda, P. Monnin, G. Pirrò, C. d’Amato (Eds.), *The Semantic Web – ISWC 2022*, Springer, 2022, pp. 714–731.
- [19] S. Malyshev, M. Krötzsch, L. González, J. Gonsior, A. Bielefeldt, Getting the most out of Wikidata: Semantic technology usage in Wikipedia’s knowledge graph, in: D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, E. Simperl (Eds.), *Proceedings of the 17th International Semantic Web Conference (ISWC’18)*, Springer, 2018, pp. 376–394.