

# WDQS Triple Store Evaluation: Benchmark Results Report

**Author:** [Wikidata Platform Team](#)

**Date:** 2026-01

## Executive Summary

This report presents the findings from our initial experimentation and benchmarking evaluation of alternative triple store databases to replace Blazegraph as the backend for the Wikidata Query Service (WDQS). The goal of this evaluation was not to identify the single best performing system, but to gather experience with open source triple store implementations and determine which ones meet minimum criteria as a replacement for Blazegraph. We evaluated two candidate systems, Virtuoso and QLever, against production-realistic criteria including performance, features, operations, and governance. We used publicly available benchmark suites (WDBench and WGPB) and reproduced existing community benchmarks in single, concurrent and read-after-write query scenarios. We gained experience in building, deploying and indexing Wikidata with both systems as well as better insights on what future hardware requirements will be for the next generation of WDQS.

**Key Findings:** Both Virtuoso and QLever meet our acceptance criteria and demonstrate superior performance compared to the current Blazegraph baseline. Either system represents a viable path forward for WDQS modernization.

## Evaluation Methodology

Our evaluation followed the test protocol outlined in our [evaluation methodology document](#), with testing conducted on AWS EC2 instances. Data ingestion in Virtuoso proved unstable with the initially proposed r8i.4xlarge instance (128GB ram), requiring us to upgrade to r8i.8xlarge instances (256 GB RAM, 32 cores).

## Test Environment

- **Hardware:** AWS cloud instances (r8i.8xlarge) with consistent network configuration. All instances were backed by SSD drives (gp3 EBS volumes) with 10,000 IOPS).

- **Datasets:** WGPB and WDBench
- **Test Conditions:** All databases evaluated under identical infrastructure and workload conditions.

## Performance Metrics Evaluated

We measured the complete set of metrics defined in our [evaluation methodology document](#) (Table 1):

- Query Response Time
- Error Rate
- Throughput (queries/second)
- Ingestion Rate (triples/second)
- Memory Utilization
- CPU Usage
- Storage Efficiency
- Read-After-Write performance (real-time index updates)

These metrics map to the success criteria we identified as necessary for the migration away from Blazegraph. Our goal is not to identify the best performance in absolute terms, but to establish a baseline to inform decision-making for database options that will power the WDQS next backend.

## Quantitative Results

All performance measurements were conducted during read-after-write scenarios, with databases actively receiving index updates to simulate production load conditions. CPU (single core) and memory utilization reached maximum levels during testing. Finer grained reporting of memory utilization will be provided in future experiments.

### Query Latency

Testing on [WGPB](#) and [WDBench](#) benchmark suites \*\* (single and multiple basic graph patterns) confirmed community benchmark results and exceeded our acceptance threshold of performance within 20% of Blazegraph baseline. In the section we aim to replicate a subset of the experiment presented in [Wikidata:Scaling Wikidata/Benchmarking/Final Report](#). Data has been collected and reported using the [benchmark-wikidata](#) suite of datasets and tools.

\*\* *WGPB and WDBench results. Single query latency. BGP = Basic Graph Pattern. Q1/Q2/Q3 = [Quartiles](#). Trimmed Mean excludes outliers.*

## Wikidata Graph Pattern Benchmark (WGPB)

850 queries • Complex graph patterns

Engine	Min	Q1 (25%)	Median (Q2)	Q3 (75%)	Max	Mean	Trimmed Mean	Timeout (total)	Errors (total)
Blazegraph	7	38	118	376	279,103	1,342	960	0	2
QLever	44	93	113	133	545	114	114	0	0
Virtuoso	1	5	14	47	180,309	441	299	0	2

*Table 1. WGPB benchmark results. Latency times are reported in milliseconds (ms)*

## WDBench (Single BGP)

280 queries, basic graph patterns

Engine	Min	Q1 (25%)	Median (Q2)	Q3 (75%)	Max	Mean	Trimmed Mean	Timeout (total)	Errors (total)
Blazegraph	1	14	19	995	600,488	12,622	4,284	3	0
QLever	7	13	17	182	16,123	765	765	0	0
Virtuoso	1	7	93	2,513	41,245	2,006	2,006	0	0

*Table 2. WDBench (Single BGP) benchmark results. Latency times are reported in milliseconds (ms).*

## WDBench (Multiple BGP)

678 queries, complex multi-pattern queries

Engine	Min	Q1 (25%)	Median (Q2)	Q3 (75%)	Max	Mean	Trimmed Mean	Timeout (total)	Errors (total)
Blazegraph	7	2,177	7,166	13,986	297,776	15,555	12,433	0	9
QLever	23	1,312	7,063	10,757	66,379	7,977	7,954	0	0
Virtuoso	1	73	3,743	6,614	180,891	5,240	4,356	0	142

*Table 3. WDBench (Multiple BGP) benchmark results. Latency times are reported in milliseconds (ms).*

All test runs have been executed three times. Reported numbers are the average of the test runs. No significant deviation was reported across runs.

Both candidates significantly outperformed baseline at extreme (min/max) and median levels, with latency within 20% of Blazegraph baseline. QLever shows consistency with the lowest maximum latencies and zero errors across both benchmarks. Virtuoso excels at minimum latencies, while Blazegraph shows high variability with extreme outliers.

### Throughput (Queries/Second)

Due to the limited amount of queries available, for these measurements we constructed a combined dataset consisting of queries from both benchmark suites and executed them in parallel using 50 concurrent workers. Queries are executed in random order, with sampling performed with replacement to maintain sustained query pressure. 50 workers have been selected as an estimate of an extreme traffic spike on a single node, based on average traffic observed on production WDQS. Under normal operating conditions we serve roughly 10 QPS per node. Measured post-warmup under sustained query load: 5-minute stress test comparison, 850 total queries, 50 concurrent workers. For this set of experiments we adjusted metrics reporting to account for performance indicators we measure in production settings (median and p95 latencies).

System	Total Queries	Throughput (q/s)	Success Rate	Median Latency (p50)	95th percentile latency (p95)
Blazegraph	3,113	10.38	95.12%	2,324	9,312
QLever	3,060	10.20	95.10%	5,140	5,493

System	Total Queries	Throughput (q/s)	Success Rate	Median Latency (p50)	95th percentile latency (p95)
Blazegraph	3,113	10.38	95.12%	2,324	9,312
QLever (-j 50)	39,632	132.10 **	99.62%	338	691
Virtuoso	7,203	24.01	97.75%	1,663	3,912

**\*\* Note:** QLever's exceptional throughput may reflect aggressive caching behavior and requires further investigation under more diverse query patterns.

*Table 4: Throughput (query/second) benchmark results with 50 concurrent clients. Latency times are reported in milliseconds (ms).*

By default QLever executes with a query concurrency level of 1. We raise the concurrency level to 50 (-j 50) simultaneous queries, and report results with both settings. These experiments confirmed latency profiles across databases, with QLever performing best in the median case.

## Data Load Performance

Blazegraph is not able to ingest the full Wikidata dataset, imposing an upper bound on dataset size. Even with federated graphs, the loading process is brittle and can take several days. This severely constrains architectural solutions (e.g., dataset reconciliation approaches) and poses a risk for disaster recovery scenarios. While we did not aggressively tune the database systems for these tasks, reasonably fast ingestion performance (<24h) is a key requirement.

Full graph reload times (complete Wikidata graph including lexemes, approximately 20 billion triples):

System	Load Time	RAM required
Blazegraph	Failed**	N/A
QLever	6 hours	128 GB

System	Load Time	RAM required
Blazegraph	Failed**	N/A
Virtuoso	20 hours	256 GB

\*\* *Blazegraph was unable to complete a full reload from dump and required manual bootstrapping from an existing journal file, representing a critical operational risk.*

*Table 5: data load performance*

QLever demonstrates 12x faster load performance than Virtuoso while requiring half the memory footprint. At ingestion time, QLever memory consumption maxed out at 70GB of RAM. Virtuoso peaked at >128GB. QLever was able to ingest the entity dumps (TTL) from a single file, whereas Virtuoso required a data preparation step to split into smaller chunks. Virtuoso data loader did not recognize the full dump file (compressed) as valid. As a baseline we split the log in chunks of approx 120MB each (compressed). The timing reported in the table above do not include data preparation. More precise measurements will be provided in future experimental runs.

## Storage Efficiency

Both candidates demonstrate significantly improved storage efficiency, reducing infrastructure costs and improving I/O performance.

System	Index Size	vs. Baseline
Blazegraph	1.3TB	-
QLever	450 GB	-65%
Virtuoso	920 GB	-29%

*Table 6: storage efficiency*

## Qualitative Analysis

### SPARQL Compliance and Feature Support

Both Virtuoso and QLever demonstrate strong SPARQL 1.1 compliance with support for federation, aggregation, and property paths as specified in our evaluation criteria.

**Critical Gap Identified:** we will need to reimplement, or rewrite queries, for the following custom Blazegraph services.

- `wikibase:mwapi` service (MediaWiki API integration)
- `wikibase:label` service (label resolution)

These services impact an estimated 50% of all WDQS queries as of December 2025.

**Migration Path:** We identified two approaches, that we will investigate in follow up work:

1. Query rewriting to SPARQL 1.1 standard constructs (feasible for graph traversal and geospatial queries)
2. Custom SPARQL extensions (required for label and API services lacking standard equivalents)

### Operations Assessment

**Deployment:** Both systems demonstrated successful deployment on our target infrastructure, with reproducible build environments.

**Monitoring:** While they do not provide an explicit Prometheus endpoint, [standard](#) monitoring integration with Observability Platform should be achievable for both. Virtuoso provides programmatic access to health and instrumentation metrics. QLever instrumentation is limited, with richer query analysis and statistics currently only available in UI and logs.

**Hardware Requirements:** Testing revealed that future WDQS nodes will require at least 256GB RAM to support reliable operation and timely data reloads across both candidates.

### Community Evaluation

Both QLever and Virtuoso maintain active open-source communities. Development tooling and documentation review revealed adequate support for enterprise deployment scenarios. To get a sense of how healthy these two projects are from a community perspective, we looked at their main GitHub repositories as a proxy (<https://github.com/openlink/virtuoso-opensource> and

<https://github.com/ad-freiburg/qlever>). This isn't a perfect methodology: both projects have related repos in their respective GitHub organizations that we're ignoring, and Virtuoso in particular has a long development history that predates its GitHub presence. But the metrics give us a reasonable indication of activity levels, community engagement, and project trajectory.

An independent review from the Wikimedia Security team for both vendors has been requested at <https://phabricator.wikimedia.org/T413229#11497256>.

## Overview

Virtuoso uses GPL-2.0 while QLever uses Apache-2.0, but neither project currently operates under a formal open source governance model (e.g. Apache Foundation). Virtuoso's development is directed by OpenLink Software, while QLever's roadmap is set by the core team at the University of Freiburg (who also run the commercial QLeverize entity). Both projects accept external contributions via GitHub, but strategic decisions remain with the respective core teams.

Criterion	Virtuoso	QLever
Virtuoso	935	754
Forks	220	106
Open Issues	648	188
Open PRs	19	100
Merged PRs	69	1176
Total Commits	1955	2274
Contributors	18	49
License	GPL-2.0	Apache 2.0

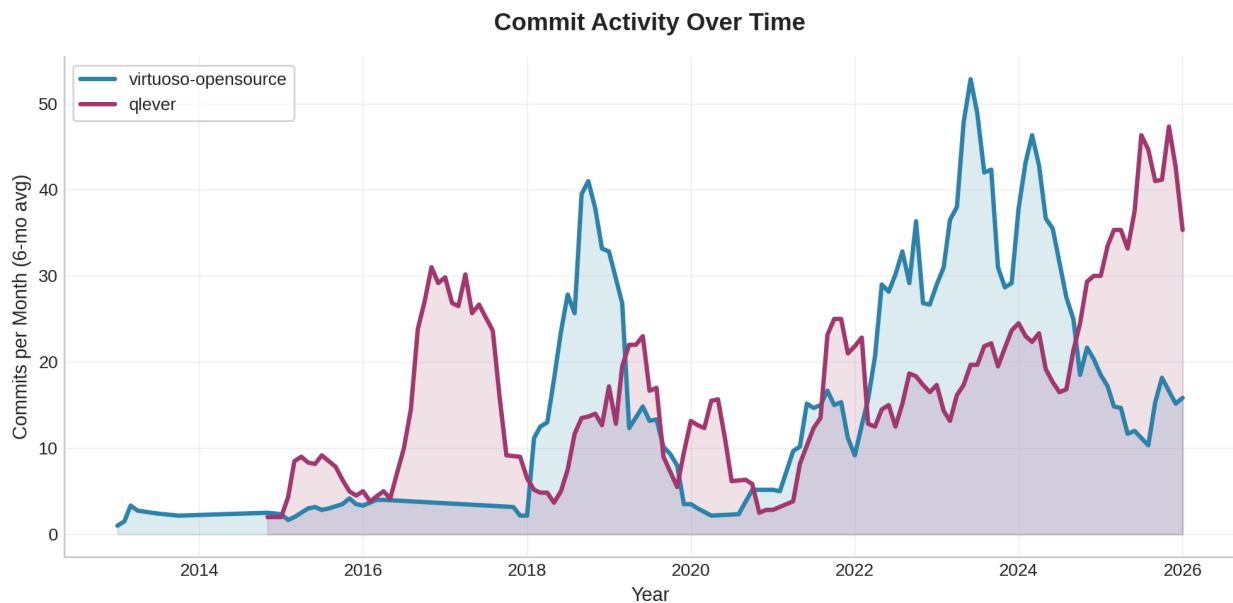
---

Primary Language	C	C++
------------------	---	-----

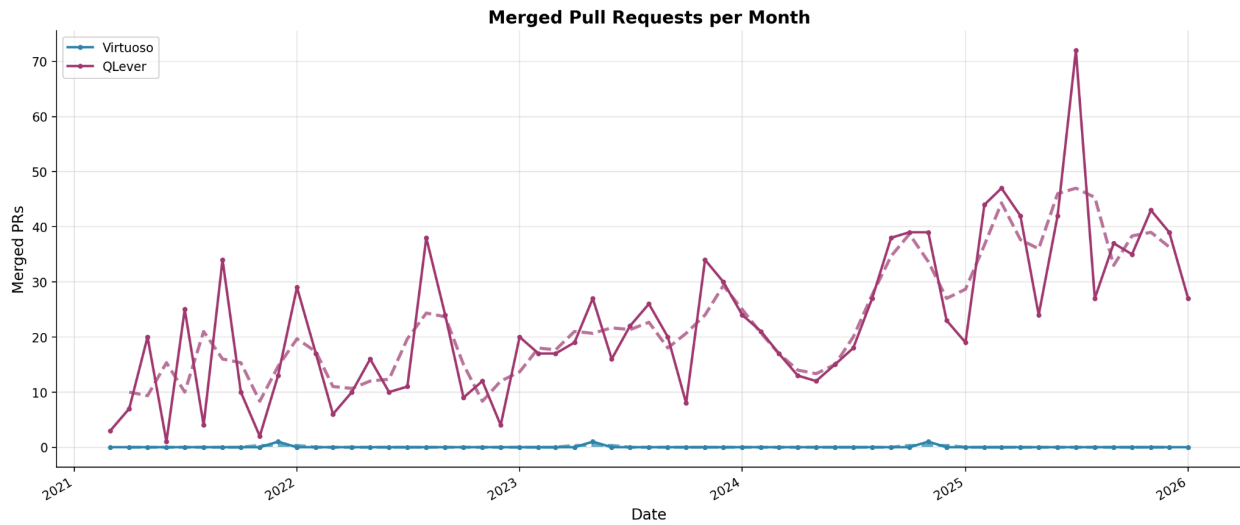
*Table 7: Virtuoso and QLever: Development activity on GitHub. Data was collected in January 2026.*

### Activity

Virtuoso codebase dates back to 1998, though the GitHub repo was set up around 2012. Development is led by OpenLink Software, with 18 contributors on the main repository. QLever is younger, starting as a research project around 2016-2017. It has 46 contributors, reflecting its academic origins where students and researchers regularly contribute alongside the core team.



*Figure 1: Virtuoso and QLever: commits per month*



*Figure 2: Virtuoso and QLever: merged PRs per month*

The projects show different patterns in issue and PR activity. Virtuoso has 648 open issues compared to QLever's 188, which likely reflects its larger and longer-established user base. QLever has more open pull requests (100 vs 19) and uses GitHub Discussions for community engagement. Both projects saw releases in 2025 and remain actively maintained.

### Contributors

The projects draw on different contributor pools. Virtuoso's development is anchored by OpenLink Software. QLever's academic origin means the core team (led by Prof. Hannah Bast) works alongside researchers and students, and the project has produced peer-reviewed publications (CIKM'17, CIKM'22, among others) documenting its architecture and performance.

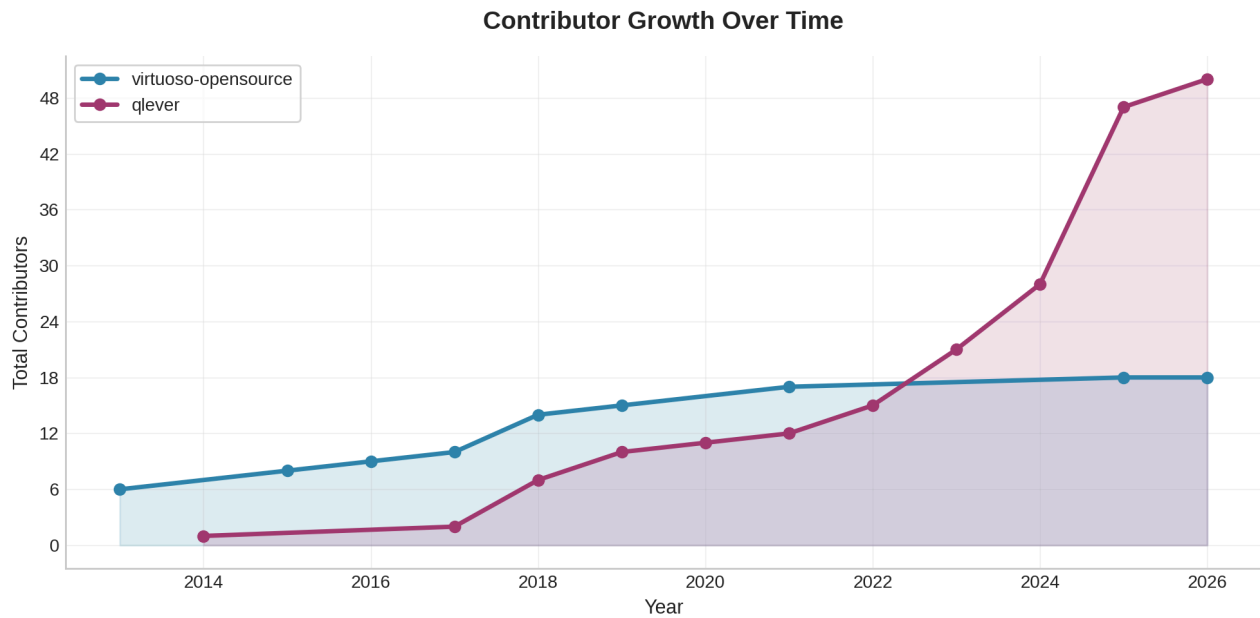


Figure 3: Virtuoso and QLever: contributors growth

## Weighted Decision Matrix

Following our evaluation framework, we calculated average comparative rankings across all criteria (lower rank is better):

Criterion	Virtuoso rank	QLever rank
Performance	2	1
Features	1	2
Operations	1	2
Community	2	1
<b>Average Ranking</b>	<b>1.5</b>	<b>1.5</b>

Table 8: Weighted decision matrix. Candidates ordered by rank (lower rank is better)

Both candidates achieved equivalent weighted rankings, indicating either represents a technically sound choice. The final selection will depend on strategic considerations including governance, feature development roadmap alignment and community partnership opportunities.

## Recommendations

Both Virtuoso and QLever meet our established success criteria:

- Capability to ingest and query 20+ billion triples
- Superior performance on primary use cases with respect to Blazegraph
- Acceptable SPARQL1.1 compliance
- Manageable migration complexity
- Support for real-time index updates

## Conclusion

This evaluation successfully validated two viable alternatives to Blazegraph, both demonstrating significant performance improvements and operational advantages. The quantitative results exceed our acceptance criteria across all measured metrics, while qualitative analysis has identified clear paths to address feature gaps. Future work will scale up our experiment and evaluate both systems with larger samples of queries and WDQS logs.

## References

- [Wikidata Triple Store Evaluation Methodology](#)
- [Community Benchmarking Repository](#)
- [Wikidata SPARQL Logs](#)
- [QLever SPARQL 1.1 Compliance Documentation](#)
- [T409769 - Hardware Requirements Specification](#)

## Appendices

### A. Testing Infrastructure

- Reproducible build and test environments: [repos / wikidata-platform / triplestores · GitLab](#)
- Event Platform client for real-time index updates: [go-wikidata-updater - repos](#)

- Load testing tooling: [repos / wikidata-platform / queryhammer · GitLab](#)

## **B. Additional Systems Evaluated**

During preliminary evaluation, we assessed additional open-source triple stores that did not meet minimum viability criteria:

- **MilleniumDB:** Excluded due to lack of real-time index update support
- **Apache Jena:** Unable to load full graph within acceptable timeframes
- **Oxigraph:** Missing critical SPARQL 1.1 features required for WDQS operation. Unable to load full graph within acceptable timeframes