WDQS Backend Alternatives

The Process, Details and Results

WDQS Search Team

Version 1.1, 29 March 2022

Overview

The Wikidata Query Service (WDQS) is part of the overall data access strategy for Wikidata. Currently, the service is hosted on two private and two internal load-balanced clusters, where each server in the cluster is running an (open-source) Blazegraph instance, and providing a SPARQL endpoint for query access. The Blazegraph infrastructure presents a problem going forward since:

- There is no short- or long-term maintenance/support strategy, given the acquisition of the Blazegraph personnel by Amazon (for their Neptune product) – as a result Blazegraph is “end of life” software that is not actively maintained
- Blazegraph is a single-server implementation (with replication for availability) that is suffering performance problems at the current database size of ~14B triples[1]
- Query performance is based on synchronous response and ‘older’ technology, and is experiencing timeouts and general instability

Goals of this work are to define several, viable architectures for the WDQS RDF store and SPARQL endpoint. Each of these architectures must (at a minimum) support[2]:

- Storage of the entire Wikidata data set (although the data may be split across multiple backend instances)
- Update of the data store in real-time (<10 min delay)
- Execution of a mix of simple and complex queries

The above requirements translate into various implications for the system architectures and their ability to support read and write in high density, to parse full SPARQL 1.1, and to execute performant queries.

All of these topics are discussed in greater detail in this document, as well as the process and evaluation details that led to the conclusions.

[1] https://grafana.wikimedia.org/d/000000489/wikidata-query-service?orgId=1&refresh=1m&viewPanel=7
[2] The complete set of requirements is presented later in this document.

Table of Contents

Summary and Results

After examining the capabilities of over 20 open-source databases, the list of candidate alternatives has been narrowed down to the following, listed in alphabetical order:

- Apache Jena with the Fuseki SPARQL Server component
  - Apache V2 license
- QLever
  - Apache V2 license
  - Wikidata SPARQL endpoint hosted on QLever at https://qlever.cs.uni-freiburg.de/wikidata
- RDF4J V4 (with the LMDB store, still in development)
  - Eclipse Distribution license
- Virtuoso Open-Source (stable branch for V7, the latest open-source release)
  - OpenLink Software's Virtuoso Open-Source (VOS) license
  - Wikidata SPARQL endpoint hosted on Virtuoso at https://wikidata.demo.openlinksw.com/sparql/

Each of these candidates are ranked by the criteria shown in Tables 1 and 2, below. The criteria are overviewed in the section, Evaluation Criteria. The evaluations are explained in detail in the section, Evaluation Results.

Technical and Community Criteria Assessments

The following tables are the results of the assessments of the candidate alternatives. The first table holds the overall technical assessments. The scoring is 0-5 (where 0 indicates no support and 5 indicates exceptional support). Note that the table includes a column evaluating the current Blazegraph solution. The second table describes the implications to the users related to query times, complexity and data freshness for each of the alternatives.

Criteria	Blazegraph	Jena	QLever	RDF4J	Virtuoso
Scalability to 10B+ triples	5	5	5	3*	5
Scalability to 25B+ triples	0	0	5	1	5
Full SPARQL 1.1 capabilities	5	5	3*	5	3
Federated query	5	5	0*	5	5
Ability to define custom SPARQL functions	3	5	2	5	4
Ability to tune/define indexes and perform range lookups	2	5	5	5	4
Support for read and write at high frequency	5	5	0*	3*	3
Active open-source community	0	5	4	5	3
Well-designed and documented code base	5	5	5	5	2
Instrumentation for data store and query management	2	5	2	4	4
Query plan explanation	5	3	5	3	2
Query plan tuning/hints within SPARQL statement	5	4	2	3	3
Query without authentication	5	5	5	5	5
Ability to prevent write access	0	5	0	5	5
Data store reload in 2-3 days (worst case)	0	2	5	3	1
Query timeout and resource recovery	2	5	4	4	3
Support for geospatial data (e.g., POINT)	5	5	5	5	5
Support for GeoSPARQL	2	5	2*	4	3
Support for named graphs (quads)	5	5	0	5	5
Query builder interface (ease of use)	3	4	5	3	5
Dataset evaluation (SHACL, ShEX)	0	5	0	5	0

Table 1. Server Assessment by Technical Criteria

(* indicates that score could be improved after testing/evaluation of work-in-progress)

Criteria	Jena	QLever	RDF4J	Virtuoso
Permit long(er) and configurable query timeouts (which translates to additional query load)	Longer timeouts will likely be required due to federation; Timeouts configurable at global and query levels	Timeouts configurable at query level	Longer timeouts will likely be required due to federation; Timeouts configurable at query level	Timeout implications need investigation / evaluation based on query load; Anytime query is not possibility (since it is not deterministic); Timeouts are global and not configurable at query level
Query full set of triples	Slower performance on some queries due to need for federation	Full capability	Slower performance on some queries due to need for federation	Full capability
Reflect most current data (Requires ability to handle frequent writes)	Needs investigation / evaluation; There are capabilities for streamed update	Proposed solution for update needs investigation / evaluation; In theory, supports real-time updates	Needs investigation / evaluation; The LMDB store should be performant	Updates may necessitate index rebuild and affect performance and correctness; Needs investigation / evaluation
Good query response time (Requires performant indexing and join operations)	Some slower performance due to federation; Possible to tune indexes and configurations	Performant query demonstrated on sample endpoint; Queries that timeout on Blazegraph likely to succeed; All index permutations are supported	Some slower performance due to federation; Possible to tune indexes and configurations	Needs investigation / evaluation since column-wise data store may not be compatible with frequent writes; Complex queries may timeout or take a long time to complete
Ease of use/easier to use (All solutions support SPARQL 1.1; federation introduces additional complexity)	Queries will be more complex since they will reference different endpoints due to federation; Can evaluate HyperGraphQL for simple queries	Excellent UI (as demonstrated on sample endpoint) with autocomplete and graphical display of query plans; Need to test full SPARQL 1.1 compliance	With FedX support, queries should not have to change (changes would be due to splitting Wikidata into sub-graphs to reduce database size)	Queries will reference new prefixes (bif: and sql:) and use non-standard terminology; Query plans explained in SQL which could be confusing; Need to test full SPARQL 1.1 compliance

Table 2. Server Assessment by User Criteria

Evaluation Process

The task of finding a replacement for the Blazegraph RDF data store for Wikidata involved examining many different engineering, technical and usage aspects. The considerations must address initial load of the data, and deal with ongoing updates (for data freshness), DevOps, server uptime, and performant query. This section overviews these different aspects and how the final set of evaluation criteria were defined.

Pre-Implementation Data Store Analysis (~2014-2015)

About 7-8 years ago, an analysis of open-source RDF data stores and SPARQL endpoints was performed. From this analysis, Blazegraph was selected as the WDQS data store. The study, however, was not based on understanding actual requirements but instead based on anticipated needs. In addition, the study was incomplete in that all of the potential candidates were not fully evaluated, and its criteria were sometimes ambiguous or overlapping.

Despite its drawbacks, criteria from this earlier study are interesting to examine. The following tables list what criteria was retained versus discarded in the current analysis.

Criteria	**Addressed as …**
Sandboxing and timing out queries	Criterion: Query timeout and resource recovery
Modularity (plug-in indexes, data types, …)	Criteria: Ability to define custom SPARQL functions + Ability to tune/define indexes
Well commented source	Criterion: Well-designed and documented code base
Query planner	Criteria: Query plan explanation + Query plan tuning/hints
Index and range lookup + Indexes for multiple traversals (hash joins) + Vertex-centric indexes	Criteria: Ability to tune/define indexes and perform greater-than and less-than (range) checking
Geospatial indexes and query support	Criteria: Support for geospatial data (e.g., POINT) + Support for GeoSPARQL
Traversal order rewriting	Criterion: Query plan tuning/hints
Experimentation console	Criteria: Instrumentation for data store and query management + Query builder interface
Upstream support for bug fixes, … + Project health + Community health	Criterion: Active open-source community

SPARQL support	Criterion: Full SPARQL 1.1 capabilities
Ability to handle lots of writes	Criterion: Support for read and write at high frequency

Table 3. 2014-2015 Backend Evaluation Criteria Maintained in Current Analysis

Criteria	Reason Discarded
Easy to type query language + Can expose native query language + Native language expressiveness	The requirement is for SPARQL support and not “native” languages
Ability to maintain uptime	All products must be stable enough to remain functional, but establishing ‘uptime’ requires testing with the R/W query load for Wikidata
Fully free software + License type (Apache, GPL, AGPL)	Mandatory requirement for open-source license (not necessary as a separate line item)
Implements some standard spec	Mandatory requirement for SPARQL 1.1, not for “some”/any standard
Horizontal scalability	Not a requirement since WDQS has a load-balanced infrastructure in place; If this criterion instead referred to sharding, then investigations of partitioned databases yielded no prospects that met the mandatory requirements of open-source, SPARQL 1.1, and an active community
Maturity of software + Stability of storage layer	Mandatory requirement to have history of stable releases and to be intended for production (not necessary as a separate line item)
Packaging (Debian) and puppetization	Mandatory requirement to execute on Linux (not necessary as a separate line item); Building a valid Debian and puppet configuration is already part of the WDQS process
Cross-data center and multi-cluster replication	Not a requirement since this is handled by the existing WDQS infrastructure
Intersecting index lookups (e.g., population > 101, country = Germany)	Not a requirement since this is really about having effective query evaluation/planning (which will be evaluated as part of testing); Also, it is not possible to determine all the possible combinations of joins
Full text indexes (stemming, ranking, …)	Not a requirement since MWAPI can be used to execute full text search over any wiki (including Wikidata)
Handles ten thousand indexes natively	Not a reasonable requirement since the issue is index size and not number; The issue is having sufficient and tunable indexes, which is addressed by the criteria to tune/define indexes
Top-n queries	Not a requirement for the evaluation, but handled in testing
Online schema changes	The Wikidata schema/ontology is defined; Changes would be anticipated and not done without planning

Multi-operation ACID	Updates align with edits on Wikidata which are done atomically (one property at a time); Writes to the backend are done from the RDF stream updater, using transactions to write batches of data (for performance reasons and not following any kind of semantic coherence)

Criteria	Reason Discarded
Storage layer designed for graphs	Investigated partitioned graph and NoSQL backends; None met the mandatory requirements of open-source, SPARQL 1.1, and an active community (therefore, this criterion was dropped); Actual criteria is related to performance which will be established in testing
“Rough edges” + Work remaining + Hacking to graph database layer	Not necessary since there is a mandatory requirement to have a history of stable releases and that the software is intended for production
Supports checking qualifiers and references (checking qualifiers either by indexing or efficient json deserialization and rechecking)	Repetitive (since indexing qualifiers is just indexing prefixes, predicates, etc.) or not a requirement (since checking/validation is performed at data update or by executing queries, and JSON is not used for input); Note that there is a line item for dataset evaluation (e.g., via SHACL) in the criteria
Supports query continuation better than offset paging	Not a requirement at this time
Supports dumping all results	Not a requirement since the Wikidata dumps are produced separately and publicly available
WMF experience	Not a requirement since the team now has experience with RDF/SPARQL and the candidate backends are written in mature languages (Java, C and C++)
Gremlin support	Not a requirement at this time
Data inference and materialization	Not a requirement at this time

Table 4. 2014-2015 Backend Evaluation Criteria Not Relevant

User Survey

In August 2021, a user survey was conducted regarding WDQS. The top five priorities (based on the combined number of #1 and #2 rankings selected by the 222 respondents) were:

1. Timeouts — “it is important to me to be able to run long queries without timeouts”
2. Graph completeness — “it is important to me that I can query the entire Wikidata graph”
3. Data freshness — “I want to be able to query the most recent updates to Wikidata”e
4. Response latency — “It is important for me that my queries return results quickly”
5. Ease of use — “I want WDQS and/or the WDQS API to be easier to use”

Survey details can be seen in Figure 1.

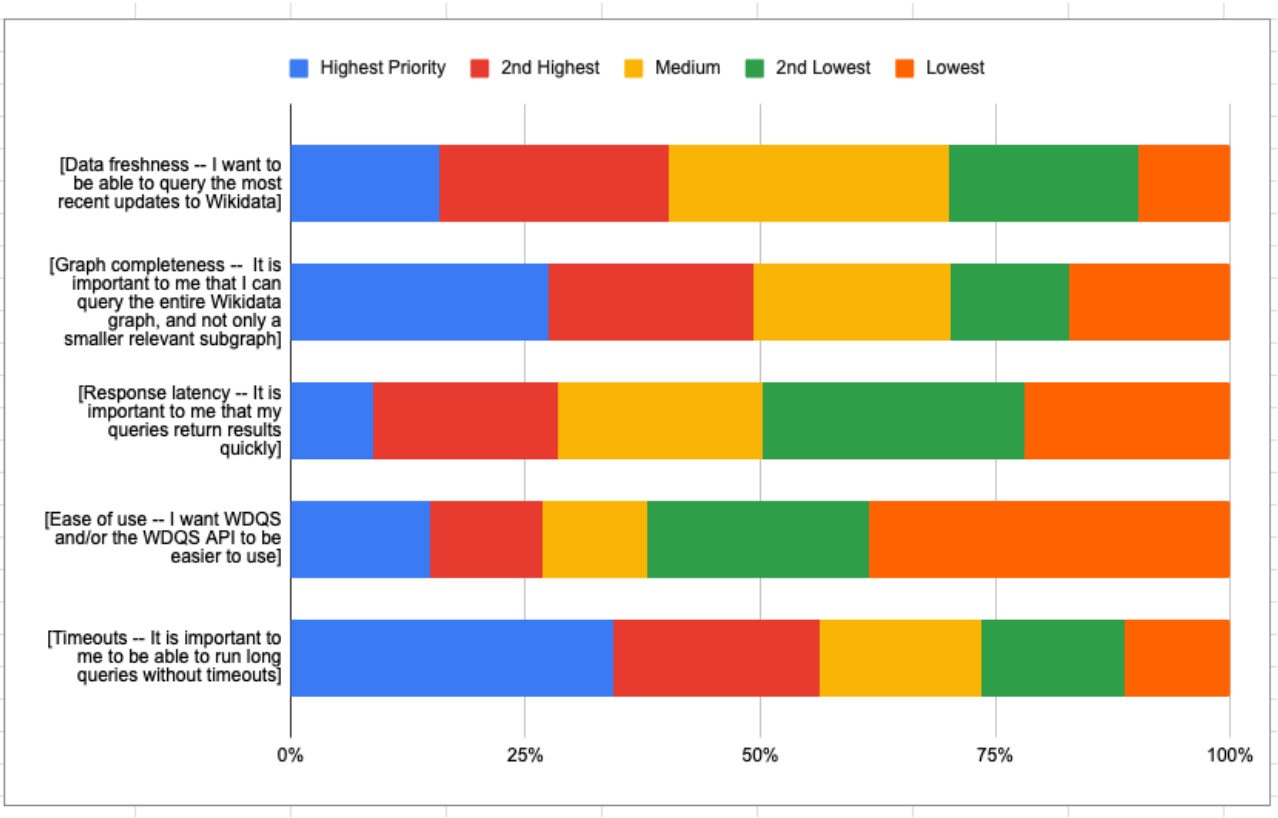


Figure 1. Bar chart of WDQS 2021 User Survey

Other feedback from the survey that influenced the current analysis[3] came from analyzing the free-text comments. Several topics are listed below, in no particular order:

- Support additional SPARQL functions such as XPath, math, etc.
- Support shape queries (e.g., is the P625 value of this item within / not within / on the boundary of that P3896 geoshape?)
- Allow query cancellation and user-defined timeouts

Phabricator Tickets

The Phabricator epic relating to investigating alternatives to Blazegraph is T206560. That epic includes tasks to define the evaluation criteria and review many, specific offerings. This document satisfies several tasks linked to the epic, specifically, T291207 (to define the criteria) and T275398 (to create a table/spreadsheet evaluating the alternatives by the criteria). Note that the only offerings that are fully evaluated in this paper are ones that are fully open-source, with an active development community, supporting SPARQL 1.1 and able to handle a minimum of 10B triples.

Literature Review

There are some excellent survey articles describing the underlying technologies of various RDF stores and SPARQL engines, and then comparing and contrasting many of the available offerings. The following papers were reviewed as part of this work:

- A Survey of RDF Stores & SPARQL Engines for Querying Knowledge Graphs
- Storage, Indexing, Query Processing and Benchmarking in Centralized and Distributed RDF Engines
- Persistence of RDF Data into NoSQL: A Survey and a Reference Architecture
- A GeoSPARQL Compliance Benchmark

[3] There are many comments in the survey, but they deal with items outside of this work - such as improvements to the user interface, documentation suggestions, etc. There are several comments regarding caching or saving query results, but again, that is outside of the scope of this document.

User Meetings

Two community meetings were held in mid-February 2022 to discuss requirements and use cases for SPARQL query and the underlying data store. Key takeaways from the first meeting on the use of SPARQL query were the importance of:

- Full SPARQL 1.1 functionality
- Property path traversals and any efficiencies/improvements that could be supported
- Ability to search labels and descriptions using wildcards, fuzzy search, etc.[4]
- Subqueries for performance
- Queries for information discovery AND for correctness and validation checking (for example, to find anomalies and duplicates)
- Improved query processing with fewer timeouts
- Geographical query (mostly for Earth-based data)
- CONSTRUCT statements for data extraction

Takeaways from the second meeting on SPARQL backends were:

- Alternatives for reducing duplication in the data (and therefore, reducing the overall size of the Wikidata dump) and for storing time series data
- Need for efficient federated query, better RDF benchmarks that reflect machine loading and query, and better documentation of what is supported and how to use it

For more details on both meetings, see the 17 February 2022 SPARQL query notes and the 21 February 2022 Backend discussion.

[4] The capabilities for full-text, wildcard and fuzzy search are important, and were discussed as being served by federation with the MediaWiki API. Overlapping capabilities in the query engine itself are not required. However, performant query support for language tags is important and will be addressed in testing.

Evaluation Criteria

The criteria for evaluation are listed below along with explanations and clarifications:

- Scalability to 25B+ triples
  - This number is an estimate of the number of triples in the Wikidata store based on a linear growth rate and 5 years of growth (using the worst case slope, from 23 November 21 to 07 December 21)
  - Ideally, the full data set is stored on a single server
- Scalability to 10B+ triples
  - Several of the possible alternatives cannot scale to the full Wikidata triples size (25B+). In this case, the triples would need to be split by some algorithm, stored on separate servers, and federated when queried.
  - This architecture is discussed in detail in the section, Wikidata Query Architectures
- Full support for SPARQL 1.1 with minimal errors
  - Any software will have errors. An error-free implementation is an ideal. But, some offerings are known to have incomplete and/or incorrect SPARQL 1.1 implementations. This criterion identifies where such problems exist.
- Federated query
  - The heart of linked open data is the ability to federate information from a variety of sources, both locally and on the web. Support for federation is mandatory for open knowledge.
  - Currently, the WDQS supports the following list of federated endpoints
  - Federated query also allows for the possibility of separating the Wikidata triples across multiple servers and databases (required if the server implementations cannot support 25B+ triples)
- Ability to define custom SPARQL functions
  - The ability to define custom SPARQL functions is necessary since SPARQL 1.1 has mandated only a small set of operations
  - In order to support other, desired math, XPath, string, aggregation and similar calculations and transformations, it must be possible to define custom functions
- Ability to tune/define indexes and perform greater-than and less-than checking
  - As for relational databases, indexes can improve query performance, but the performance comes at the cost of increased maintenance and storage requirements
  - It is possible to index triples using one or more of: subjects/objects (items), properties/paths and query/join patterns (either statically or dynamically defined)
  - There are six possible index permutations for triples (subject(s)-predicate(p)-object(o) pairs): spo, sop, pso, pos, ops, osp (these should be handled by several indexes)

    - Named graphs (or quads) add another element, resulting in 24 possible combinations
- Support for R/W at high frequency
    - As indicated in the user survey, many WDQS consumers require near-real-time update of the data
    - Analysis of the new WDQS stream update service indicates that spikes of ~1000 added/deleted triples/second can occur
    - The average update frequency is in the range, 50-200 triples/second
- Active open-source community
    - WDQS is supported by the WMF Search Platform Team, which must split resources between new functionality and bug fixes
    - Ideally, bug fixes should be focused on internal tools and services, and not on errors in the RDF store or SPARQL endpoint
    - For these reasons, an active community is needed to address storage and query problems
    - Given that "freedom and open source" are a guiding principle of the Wikimedia Foundation, there is a mandatory requirement that the backing infrastructure for WDQS be open-source
- Well-designed and documented code base
    - This criterion relates to the offering being maintainable by its community
- Instrumentation for data store and query management
    - When problems occur in the WDQS clusters, site reliability engineers must be able to acquire operational statistics on the underlying applications and services, and have the ability to affect change via CLI and browser-based tooling
    - Configuration of thread pools and memory management and management of executing queries are especially important
- Query plan explanation
    - Data stores execute queries by first determining their query plan (the sequence of steps that will result in the most efficient execution of the query), and then the actual execution of the plan
    - The most efficient execution strategy is to obtain results for the *most selective* clauses first (these are the basic graph patterns[5] that have the smallest number of results)
        - For example, *?s wdt:P569 ?date* is less selective than *wd:Q42 wdt:P569 ?date,* since the first triple returns the dates when any entity was born, but the second returns the date when Douglas Adams was born
    - Long-running queries always occur, but sometimes, they can be rewritten to execute more quickly
    - If a query's triple patterns return many matches (called intermediate result sets), those results increase memory usage and runtime, since they must be sorted, added to other results via a join, etc.

[5] https://www.w3.org/TR/rdf-sparql-query/#BasicGraphPatterns

    - In order to tune queries, the results of the query planning stage must be available
- Query plan tuning/hints
    - Ideally, a SPARQL engine allows user-defined instructions to define or change the query plan
- Query without authentication
    - Authentication in an open-access environment is problematic, since it could impact privacy
    - For this reason, query capabilities must be provided without authentication
    - However, advanced capabilities (such as complex queries with lengthy timeouts) may require authentication. This is still to be determined.
- Ability to prevent write access, except by the stream updater
    - Data is updated by editing Wikidata directly, and not through SPARQL INSERT/DELETE statements
    - However, additions, deletions and changes must be able to be performed by the WDQS stream update service, in a secure manner
- Data store reload in 2-3 days (worst case)
    - Current Blazegraph data loading (of an RDF dump) takes over 10 days
    - Reload might be needed to address data drifts (due to issues in the streaming update process) and to improve indexes
- Query timeout and resource recovery
    - Executing long-running queries consumes server resources and reduces the overall performance of the system
    - It is mandatory to be able to set hard timeouts (which cannot be violated by any query) as well as user-defined timeouts which would be set by the query writer (for example, when testing that a new query is valid before attempting to execute it)
- Support for geospatial (POINT) data
    - Geographical query was a topic highlighted at the first user meeting on Blazegraph alternatives, discussed above
    - The basis for this type of query is a geographical point (latitude and longitude, defined relative to a coordinate system) and the functions (such as “distance” or “within”) and constructs (such as polygons) that are built using points
- Support for GeoSPARQL
    - GeoSPARQL was published by the Open Geospatial Consortium on 10 September 2012
    - It defines an ontology of concepts and relationships, new inference rules and new SPARQL query functions
    - Note that there are *no* complete implementations of GeoSPARQL but three of the four candidate alternatives do provide partial or vendor-specific implementations
- Support for named graphs (quads)
    - Currently, the Blazegraph implementation does not use named graphs

    - These can be a useful concept to improve query performance since the evaluation of triples in a query may be limited to one or more graphs (and therefore improve selectivity and reduce the size of intermediate result sets)
- Query builder interface
    - The interface provided at https://query.wikidata.org/ includes a string based and templated builder
    - However, if a user interface is provided by the alternative offering, then this may be useful to replace or improve current capabilities
- Dataset evaluation by SHACL (SHape Constraint Language), ShEX (Shape EXpressions), or similar
    - Although integrity checking is done prior to loading in the WDQS store, querying the store provides a secondary means to verify the data and find more complex errors
    - If the backing store and query service can provide more advanced capabilities, then that is valuable
    - Integrity can always be validated using SPARQL functions. However, newer languages (such as SHACL) may make this task easier.

Evaluation Results

The sub-sections below discuss the detailed results evaluating each alternative against the criteria. Note that the alternatives are discussed in alphabetical order. At the end of this section, numerous data stores are listed that were investigated but dismissed.

Apache Jena

Jena's code layout is shown in Figure 2[6] and consists of multiple APIs and components to load, update, store and query RDF and OWL triples. The Fuseki component adds support for querying and updating the triples using SPARQL and HTTP. Beyond what is shown below, there are additional features such as command-line tooling, specialized indexes and more. These are discussed in more detail related to the evaluation criteria.

[6] The Jena architecture is explained in detail, starting at the web page, https://jena.apache.org/about_jena/architecture.html. Figure 2 is captured from that page.

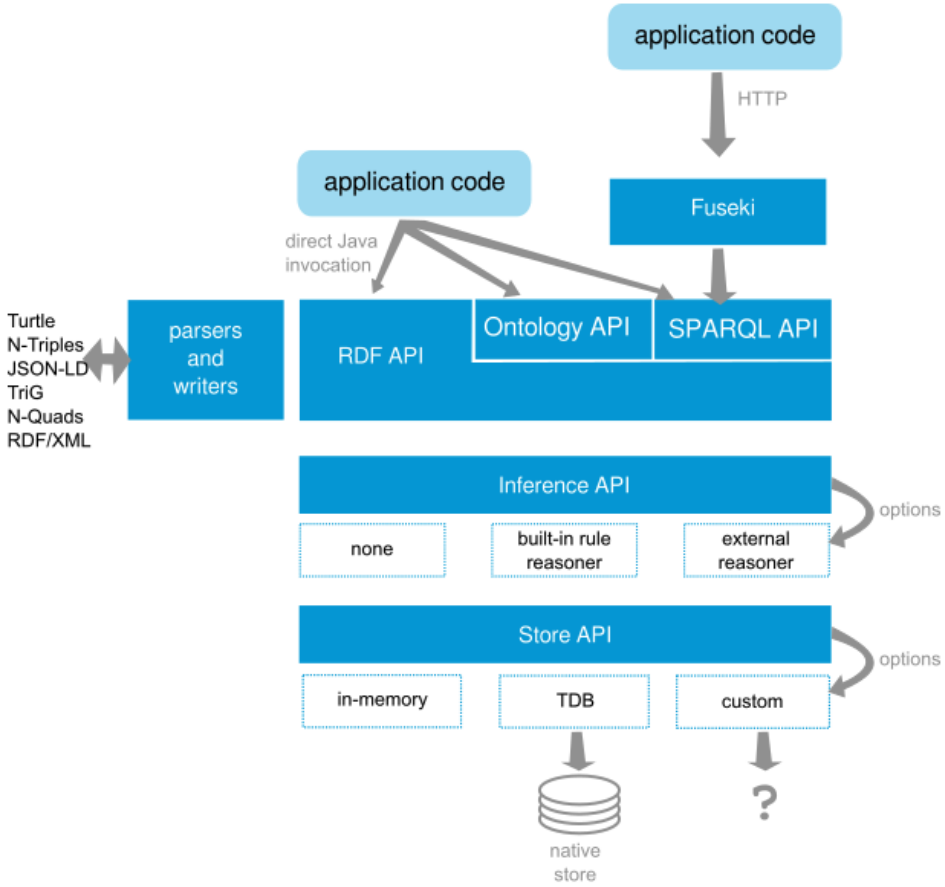


Figure 2. Jena Code Layout

Below, each of the evaluation criteria are listed with Jena's score (from Table 1) and a short explanation. Note that the scoring is 0-5 (where 0 indicates no support and 5 indicates exceptional support).

- Scalability to 25B+ triples - 0
  - Although scale to 16.7B triples has been achieved[7], this appears to be reaching the capabilities of the overall system
  - Therefore, the ability to scale to 25B+ triples is not expected
  - Splitting the Wikidata triples by subject area (such as scholarly articles, taxonomy entities, etc.) into separate stores and then using federated queries will be required
- Scalability to 10B+ triples - 5 (noted in previous bullet)
- Full support for SPARQL 1.1 with minimal errors - 5
  - ARQ is Jena's SPARQL processor with excellent support for V1.1 functionality as well as update capabilities[8]

[7] https://muncca.com/2019/02/14/wikidata-import-in-apache-jena/ and https://lists.apache.org/thread/rphn74r9vbovwjvylxjmrd6qnfvbt4t0 (see the update for 28 December 2021 and the log details at http://www.lotico.com/temp/LOG-45497)
[8] https://www.w3.org/2009/sparql/implementations/, last modified 20 March 2013

- Federated query - 5
    - Jena provides support for the SERVICE keyword as well as federation-specific configurations that can be set globally or per-query[9]
- Ability to define custom SPARQL functions - 5
    - ARQ can be extended by adding expression functions/operations in FILTERS, BIND and SELECT statements, and property functions to execute custom code associated with a predicate IRI[10]
    - In addition, the code[11] provides a SERVICE-level extension capability similar to what is used with Blazegraph today
- Ability to tune/define indexes and perform greater-than/less-than checking - 5
    - Jena's data store (TDB2) uses a 64bit ID for each RDF term including literals, IRIs and blank nodes (e.g., there are node <-> id dictionaries)
    - For triples, the default indexes are SPO, POS and OSP
    - For quads, the default indexes are GSPO , GPOS , GOSP , POSG , OSPG and SPOG
    - Each triple or quad is stored in an index using its appropriate IDs
    - By default, indexes are memory mapped files stored outside of the system heap[12]
    - Range lookups are implemented across a number of points in the code. For example, see the code at https://github.com/apache/jena/blob/main/jena-db/jena-dboe-index/src/main/java/org/apache/jena/dboe/index/RangeIndex.java
- Support for R/W at high frequency - 5
    - Jena supports processing RDF in a streaming mode[13]
- Active open-source community - 5
    - Jena's open-source code is stored at https://github.com/apache/jena, licensed under Apache V2 and written in Java
    - There are 78 contributors with recent commits in March 2022
- Well-designed and documented code base - 5
- Instrumentation for data store and query management - 5
    - Fuseki has a low-cost HTTP GET "ping" operation to test whether a server is responsive
    - Queries can be invoked and debugged via Jena command-line tools and Fuseki's HTTP GET/POST interface. Data store updates are handled similarly. In addition, data set statistics can be output, and the dataset compacted.[14]

[9] https://jena.apache.org/documentation/query/service.html
[10] https://jena.apache.org/documentation/query/extension.html and https://jena.apache.org/documentation/query/writing_functions.html
[11] https://github.com/apache/jena/tree/main/jena-arq/src/main/java/org/apache/jena/sparql/service
[12] https://stackoverflow.com/questions/55611943/what-indices-does-jena-tdb2-use and https://jena.apache.org/documentation/tdb/store-parameters.html
[13] https://jena.apache.org/documentation/io/streaming-io.html
[14] https://jena.apache.org/documentation/tools/

    - The TDB data store can optionally log query execution details to investigate “what is going on”[15]
    - There is a user interface for administration and query of Fuseki[16]. It works with any SPARQL endpoint.
- Query plan explanation - 3
    - Using the command line instructions, qparse or tdbquery -explain, a query’s algebra and/or interactions with the TDB data store can be described
    - The amount of information that is output is dependent on the logging level and output layouts
    - One concern is the complexity of the query algebra which would render the output unintelligible to most users. This can be mitigated by documentation, use of the SSE (SPARQL S-Expressions syntax[17]) and simplification of the output.
- Query plan tuning/hints - 4
    - 3 different query planning schemes are defined - no statement reordering, heuristics-based reordering and statistics-based reordering[18]
    - Although no default query hints are supported by text in the SPARQL query, these could be managed by a WDQS front-end that invokes different planning schemes based on observing a predefined string or comment text in the query.
    - Specific optimization tasks can be turned on/off using SystemARQ symbols[19] (such as "optPathFlatten")
- Query without authentication - 5
    - Apache Shiro is used as the authentication infrastructure for Jena
    - By default, SPARQL endpoints are open to the public but administrative functions are limited to localhost
- Ability to prevent write access, except by the stream updater - 5
    - Update (create, update and delete) actions can be restricted by editing the shiro.ini file
    - In addition, the query and update endpoints can use different URL paths. The update functionality can then be controlled and authenticated by a reverse proxy.
- Data store reload in 2-3 days (worst case) - 2
    - Loading, as documented in the links in footnote 7, took ~104h (4.3 days)
    - This was using the xloader (bulk load) command
    - This time will be reduced since the data will be split across multiple servers. Meeting the timing requirements needs to be validated (hence, the current score is 2).

[15] https://jena.apache.org/documentation/tdb/optimizer.html#investigating-what-is-going-on
[16] https://jena.apache.org/documentation/fuseki2/
[17] https://jena.apache.org/documentation/notes/sse.html
[18] https://jena.apache.org/documentation/tdb/optimizer.html#choosing-the-optimizer-strategy
[19] https://github.com/apache/jena/blob/main/jena-arq/src/main/java/org/apache/jena/query/ARQ.java

- Query timeout and resource recovery - 5
    - Timeouts can be defined globally and per query
- Support for geospatial (POINT) data - 5
    - See the bullet below for GeoSPARQL support
- Support for GeoSPARQL - 5
    - The Jena GeoSPARQL interface is described at https://jena.apache.org/documentation/geosparql/ and is integrated with Fuseki (the assembler for GeoSPARQL[20] is part of the jena-geosparql artifact and must be on the Fuseki server classpath, along with its dependencies)
    - WKT and GML serializations are supported, as well as all spatial relation families (Simple Feature, Egenhofer and RCC8)
    - Additional geospatial indexes are defined and configurable
    - According to the paper, *A GeoSPARQL Compliance Benchmark*[21], "GeoSPARQL Fuseki is the triplestore with the highest GeoSPARQL compliance score in our experiments"
- Support for named graphs (quads) - 5
    - As noted in the bullet regarding indexes
- Query builder interface - 4
    - There are a set of query build APIs[22], although there is no currently defined user interface
- Dataset evaluation by SHACL (SHape Constraint Language), ShEX (Shape EXpressions), or similar - 5
    - Jena supports both SHACL[23] and ShEX[24]

QLever

The QLever architecture is shown in Figure 3. Its design is easily identifiable by examining its codebase[25] on GitHub. The architecture uses a knowledge-base index supporting all the basic operations and enabling efficient query processing. The system is explained in more detail in the paper, *QLever: A Query Engine for Efficient SPARQL+Text Search*[26].

To-date, QLever does not support federation or system update, although a design for these features is proposed[27]. For this reason, several of the criteria scores are set to "0" until an implementation can be tested.

---

[20] https://jena.apache.org/documentation/geosparql/geosparql-assembler
[21] https://arxiv.org/pdf/2102.06139.pdf
[22] https://jena.apache.org/documentation/extras/querybuilder/index.html
[23] https://jena.apache.org/documentation/shacl/
[24] https://jena.apache.org/documentation/shex/
[25] https://github.com/ad-freiburg/qlever
[26] http://ad-publications.informatik.uni-freiburg.de/CIKM_qlever_BB_2017.pdf
[27] https://github.com/ad-freiburg/qlever/wiki/QLever-support-for-SPARQL-1.1-Update

A SPARQL endpoint for Wikidata is available for testing at https://qlever.cs.uni-freiburg.de/wikidata.

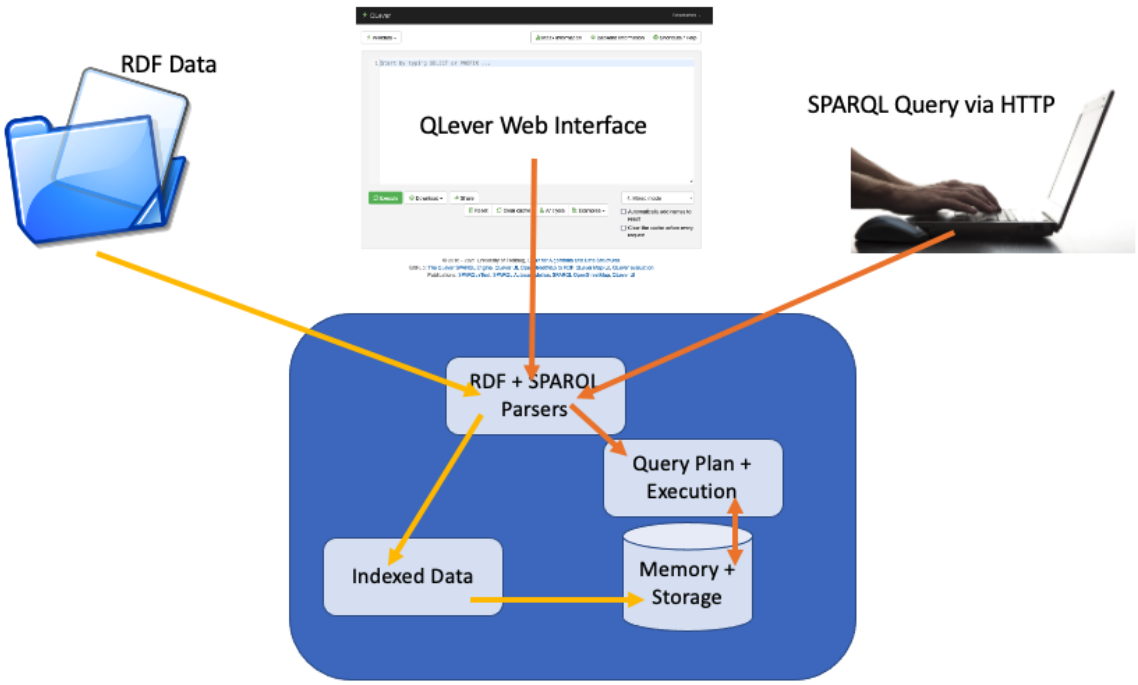


Figure 3. QLever Architecture

Below, each of the evaluation criteria are listed with QLever's score (from Table 1) and a short explanation. Note that the scoring is 0-5 (where 0 indicates no support and 5 indicates exceptional support).

- Scalability to 25B+ triples - 5
  - As shown at the testing endpoint noted above, the infrastructure currently supports loading 16.7B triples and is expected to scale well beyond this
- Scalability to 10B+ triples - 5
- Full support for SPARQL 1.1 with minimal errors - 3 (best case 5)
  - QLever is compliant with SPARQL V1.1 functionality but remains to be fully tested. This work is ongoing using the https://www.w3.org/2009/sparql/docs/tests/ suite[28].
  - Until this work is complete, the assigned score is 3
- Federated query - 0 (best case 5)
  - Design is described in footnote 27 which discusses SPARQL Update support. That solution also provides an ability to support results from a federated query.

[28] https://github.com/ad-freiburg/qlever/discussions/620

- Ability to define custom SPARQL functions - 2
    - Custom functions can be added as described on the GitHub repository Discussions page[29]
    - Since this approach involves modifying QLever directly (versus using an extension point), the assigned score is 2
- Ability to tune/define indexes and perform greater-than/less-than checking - 5
    - RDF triples are indexed in all possible ways - SPO, SOP, PSO, POS, OSP, OPS
    - For some query environments, two permutations (PSO and POS) may be sufficient and will reduce indexing time and storage. The impact is poor performance for variable (as opposed to bound) predicates.
    - Efficient range lookups are enabled since each literal has a unique ID, the literals are split into groups based on their data type, and each group has an interval of IDs which do not overlap and are in lexicographical, numeric or chronological order[30]
- Support for R/W at high frequency - 0 (best case 5)
    - Query support is exceptional, but there is no current support for SPARQL Update
    - A design for Update is referenced in footnote 27
- Active open-source community - 4
    - QLever's open-source code is stored at https://github.com/ad-freiburg/qlever, licensed under the Apache V2 License and written in C++
    - There are 15 contributors with recent commits in March 2022
    - At present, the contributors are all associated with the University of Freiburg
- Well-designed and documented code base - 5
- Instrumentation for data store and query management - 2
    - Limited statistics are available as can be seen in the SPARQL endpoint at https://qlever.cs.uni-freiburg.de/wikidata
    - In addition, arbitrary batches of queries can be evaluated as regards aggregated statistics and details for individual queries[31]
    - No command line tooling exists
- Query plan explanation - 5
    - Query planning is explained in detail in the paper referenced in footnote 25
    - The QLever user interface provides an excellent query plan visualization as can be seen in Figure 4 (the figure is provided since it is exceptional to the interfaces provided by WDQS or the other alternatives)

[29] https://github.com/ad-freiburg/qlever/discussions/592
[30] https://github.com/ad-freiburg/qlever/wiki/QLever's-IDs-for-IRIs-and-literals,-internal-and-external-vocabulary
[31] See https://qlever.cs.uni-freiburg.de/evaluation/www/?res=wikidata.sensitive

Query Analysis

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?name ?population WHERE {
  ?city wdt:P31/wdt:P279* wd:Q515 .
  ?city wdt:P17 wd:Q183 .
  ?city wdt:P1082 ?population .
  ?city rdfs:label ?name .
  FILTER (LANG(?name) = "de")
}
ORDER BY DESC(?population)
```

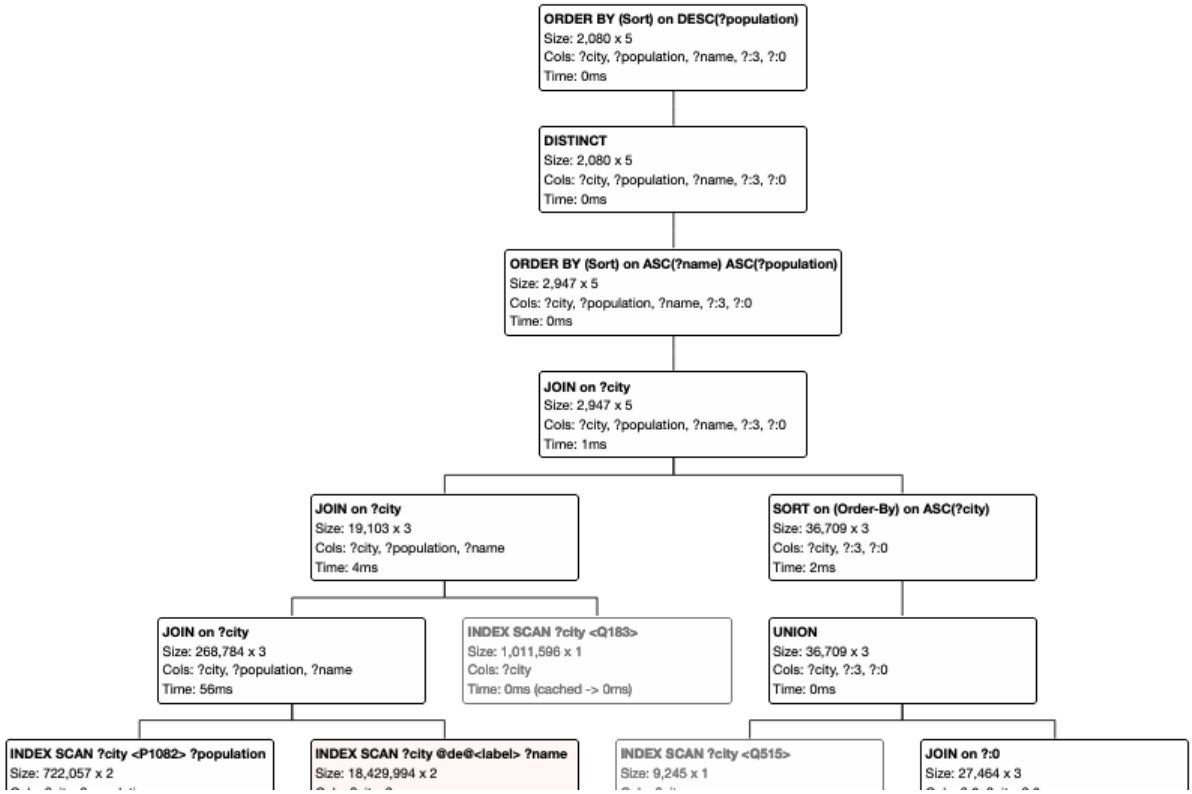


Figure 4. QLever Query Plan Explanation

- Query plan tuning/hints - 2
    - An evaluation strategy could be manually defined that executes a query without optimization, or that performs customized optimizations or different orderings of optimizations
    - No query hints are supported by text/keywords in the SPARQL query. These could be managed by a WDQS front-end that invokes different query plans based on observing a predefined string or comment text in the query.

- Query without authentication - 5
    - Access with no authentication can be seen on the Wikidata test page at https://qlever.cs.uni-freiburg.de/wikidata
- Ability to prevent write access, except by the stream updater - 0 (best case 5)
    - Restricted access would have to be added when SPARQL Update processing is added
- Data store reload in 2-3 days (worst case) - 5
    - The load time (to create the data indexes) is approximately 14 hours for the current Wikidata RDF dump file[32]
- Query timeout and resource recovery - 4
    - Per-query timeout with the URL parameter "timeout", see https://github.com/ad-freiburg/qlever/blob/master/src/engine/Server.cpp#L327-L335
    - There are explicitly coded "exit points" at regular time intervals in all time-intensive operations, which can easily be validated against a global timeout value
- Support for geospatial (POINT) data - 5
    - Geospatial geometries are supported as documented in the paper, *An Efficient RDF Converter and SPARQL Endpoint for the Complete OpenStreetMap Data*[33]
- Support for GeoSPARQL - 2
    - Support for GeoSPARQL operations like ogc:contains exists, but uses unique predicates, as described in the paper in the bullet above
- Support for named graphs (quads) - 0 (best case 5)
- Query builder interface - 5
    - An excellent UI is provided that has auto-complete capabilities exceeding the current WDQS implementation
- Dataset evaluation by SHACL (SHape Constraint Language), ShEX (Shape EXpressions), or similar - 0

RDF4J V4

The Eclipse RDF4J architecture is shown in Figure 5[34]. It is a modular Java framework for working with and querying RDF data. The architecture supports several native stores, and in V4 (release date still TBD) includes a new database store, LMDB[35]. LMDB is built on the Symas Lightning Memory-Mapped Database[36] and has the potential to scale to billions of triples. To-date, the exact performance of this new database has not been fully defined and will require further testing. For this reason,

[32] https://github.com/ad-freiburg/qlever/commit/4d21db7e43f69779a91daf436ceb0a3f6ee9cd29
[33] https://ad-publications.cs.uni-freiburg.de/SIGSPATIAL_osm2rdf_BBKL_2021.pdf
[34] The RDF4J framework is described on the page, https://rdf4j.org/about/. The figure is captured from that page.
[35] https://rdf4j.org/documentation/programming/lmdb-store/
[36] https://www.symas.com/lmdb

several of the criteria scores are set to “2” or “3” until more details can be provided and/or the evaluation testing is complete.

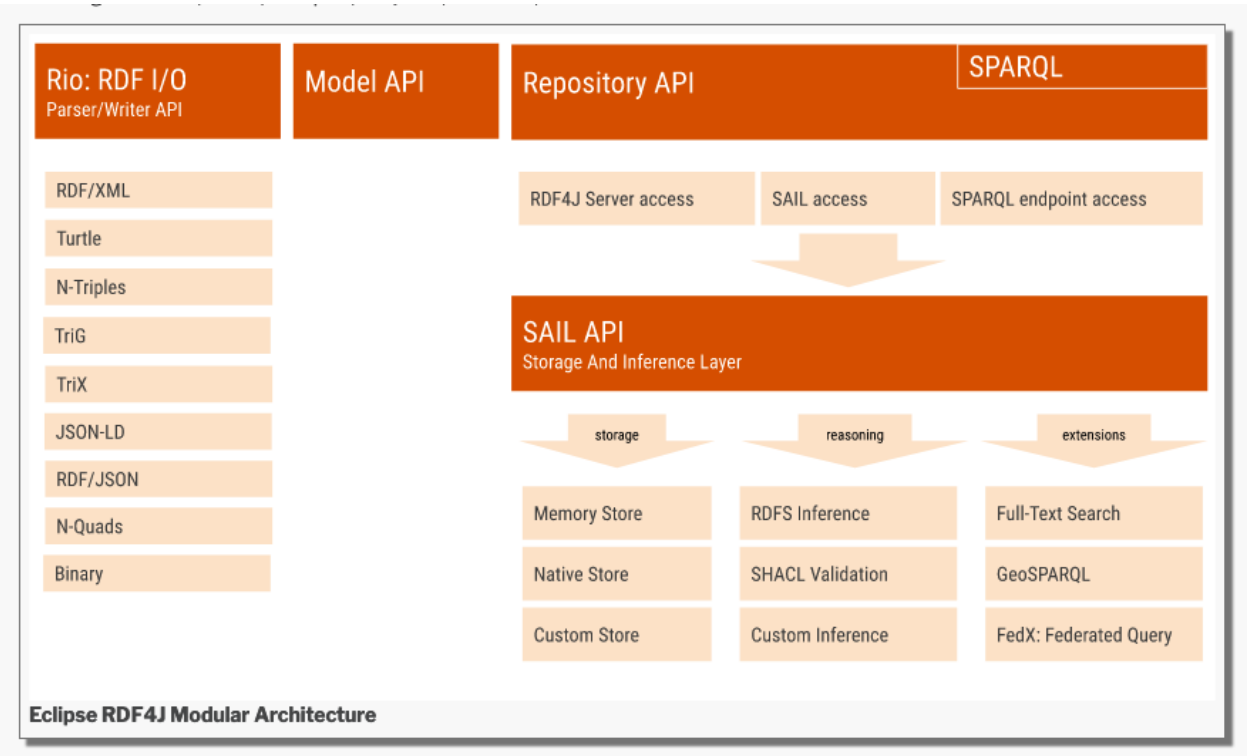


Figure 5. RDF4J Architecture

Below, each of the evaluation criteria are listed with RDF4J’s score (from Table 1) and a short explanation. Note that the scoring is 0-5 (where 0 indicates no support and 5 indicates exceptional support).

- Scalability to 25B+ triples - 1
  - There is an expected footprint of around 120-130 bytes per quad for the LMDB store, and each server should have sufficient RAM to hold the complete data set (for optimal performance)[37]
  - For these reasons, scalability to 25B triples may be possible but is not likely
- Scalability to 10B+ triples - 3 (best case 5)
  - The scoring is currently set to 3 until the RDF4J LMDB scalability can be validated
- Full support for SPARQL 1.1 with minimal errors - 5
  - RDF4J is compliant with SPARQL V1.1 functionality as well as supporting update functionality and transactions[38]

[37] https://github.com/eclipse/rdf4j/discussions/3706#discussioncomment-2285864
[38] https://rdf4j.org/documentation/reference/rest-api/

- Federated query - 5
    - SPARQL federated query is fully supported by RDF4J[39]
    - In addition, the open-source FedX solution[40] was added to RDF4J in 2019. It provides “transparent federation of multiple SPARQL endpoints under a single virtual endpoint. As an example, a knowledge graph such as Wikidata can be queried in a federation with endpoints that are linked to Wikidata as an integration hub. In a federated SPARQL query in FedX, one no longer needs to explicitly address specific endpoints using SERVICE clauses. Instead, FedX automatically selects relevant sources, sends statement patterns to these sources for evaluation, and joins the individual results.”[41] Join processing is optimized.
    - FedX capabilities could improve the usability of Wikidata’s white-listed federated endpoints, as well as making the splitting of the Wikidata into sub-graphs transparent
- Ability to define custom SPARQL functions - 5
    - RDF4J can be extended by adding custom functions[42]
    - Custom functions typically return a single value. Multiple values can be returned by implementing a TupleFunction (instead of Function), adding the function name to META-INF/services/org.eclipse.rdf4j.query.algebra.evaluation.function.TupleFunction and configuring the backend SAIL to use an ExtendedEvaluationStrategy (instead of the default)[43]
- Ability to tune/define indexes and perform greater-than/less-than checking - 5
    - RDF4J allows the specification of indexes referencing 's', 'p', 'o' and 'c' (where ‘c’ references the context or named graph)[44]
    - The default indexes are spoc and posc
    - Range lookups are supported by in the getTriplesUsingIndex function, from the file referenced in footnote 44
- Support for R/W at high frequency - 3 (best case 5)
    - Performance of LMDB is expected to be good, but testing is needed to understand the impacts of a large number of reads/writes
    - In addition, the current implementation is tracking two GitHub issues that affect performance[45]
    - For these reasons, the current score is 3
- Active open-source community - 5
    - RDF4J’s open-source code is stored at https://github.com/eclipse/rdf4j, licensed under the Eclipse Foundation (as a BSD 3-Clause license) and written in Java

[39] https://rdf4j.org/javadoc/3.4.3/org/eclipse/rdf4j/repository/sparql/federation/SPARQLFederatedService.html and https://github.com/eclipse/rdf4j-storage/tree/master/federation/src/main/java/org/eclipse/rdf4j/sail/federation
[40] https://github.com/VeritasOS/fedx/wiki/FedX-Short-Documentation
[41] https://rdf4j.org/news/2019/10/15/fedx-joins-rdf4j/
[42] https://rdf4j.org/documentation/tutorials/custom-sparql-functions/
[43] https://groups.google.com/g/rdf4j-users/c/6t2bV473DvA
[44] https://github.com/eclipse/rdf4j/blob/develop/core/sail/lmdb/src/main/java/org/eclipse/rdf4j/sail/lmdb/TripleStore.java
[45] https://github.com/eclipse/rdf4j/issues/3534 and https://github.com/eclipse/rdf4j/issues/3574

    - There are 76 contributors with recent commits in March 2022 on both the “main” and “develop” branches (the LMDB store is only available in the “develop” branch)
- Well-designed and documented code base - 5
- Instrumentation for data store and query management[46] - 4
    - RDF4J Console is a command-line application (.bat or .sh script) with support for creating and using data stores, verifying RDF files and executing queries
    - In addition, there is a web-based client UI, the RDF4J Workbench
- Query plan explanation - 3 (best case 5)
    - At this time, query explanations are an experimental feature but have excellent functionality[47]
    - There are four levels of explanation - from “Unoptimized” (which parses the query for errors) through “Timed” (which is a complete evaluation with result set sizes and performance timings)
    - In addition, there is a visualization method (for example, query.explain(Explanation.Level.Timed).toDot())
    - Support is only provided for local repositories. Explanations are not available if using REST/HTTP access (see https://github.com/eclipse/rdf4j/issues/2979).
    - It is unclear the extent of the support for LMDB, except for providing cardinalities/counts
- Query plan tuning/hints - 3
    - An EvaluationStrategy[48] could be manually defined that executes a query without optimization, or that performs customized optimizations or different orderings of optimizations
    - The default optimization strategy is defined at https://github.com/eclipse/rdf4j/blob/main/core/queryalgebra/evaluation/src/main/java/org/eclipse/rdf4j/query/algebra/evaluation/impl/StandardQueryOptimizerPipeline.java
    - No query hints are supported by text/keywords in the SPARQL query. These could be managed by a WDQS front-end that invokes different QueryOptimizerPipelines based on observing a predefined string or comment text in the query.
- Query without authentication - 5
    - RDF4J is deployed to a servlet container (usually Tomcat) which initially has no server roles defined[49]
    - Using a combination of URL patterns and HTTP methods (GET, POST, PUT and DELETE), user roles can be restricted

[46] https://rdf4j.org/documentation/tools/
[47] https://rdf4j.org/documentation/programming/repository/#explaining-queries
[48] https://github.com/eclipse/rdf4j/blob/main/core/queryalgebra/evaluation/src/main/java/org/eclipse/rdf4j/query/algebra/evaluation/EvaluationStrategy.java
[49] https://rdf4j.org/documentation/tools/server-workbench/#access-rights-and-security

    - For Tomcat, roles are defined in the file, web.xml, in the .../webapps/rdf4j-server/WEB-INF directory
    - In addition, there will be a spring-boot read-only SPARQL HTTP protocol server in V4
- Ability to prevent write access, except by the stream updater - 5
    - As noted above, with the definition of appropriate user roles
- Data store reload in 2-3 days (worst case) - 3 (best case 5)
    - The LMDB store is designed to be performant, but load times need to be tested
    - Hence, the score is currently set to 3
- Query timeout and resource recovery - 4
    - Timeouts are defined per query and there is no general/overriding setting (it would be possible to update the code in AbstractOperation[50] to provide an overriding, global timeout)
- Support for geospatial (POINT) data - 5
    - See the bullet below for GeoSPARQL support
- Support for GeoSPARQL - 4
    - RDF4J supports GeoSPARQL functions on top of data represented as Well-Known Text (WKT) strings
    - Most of the non-topological, common, simple, Egenhofer and RCC8 GeoSPARQL functions are supported[51]
    - Per the paper, *A GeoSPARQL Compliance Benchmark* (referenced in footnote 21), RDF4J's implementation was 58.33% compliant
- Support for named graphs (quads) - 5
    - As noted in the bullet regarding indexes
- Query builder interface - 3
    - There is an ability to define queries using the SPARQLBuilder[52], although there is no currently defined user interface
- Dataset evaluation by SHACL (SHape Constraint Language), ShEX (Shape EXpressions), or similar - 5
    - SHACL is supported[53]

[50] https://github.com/eclipse/rdf4j/blob/75b97f6106e6a73776d7f0f663f18a3edfb0a4ae/core/query/src/main/java/org/eclipse/rdf4j/query/impl/AbstractOperation.java#L20

[51] https://rdf4j.org/documentation/programming/geosparql/

[52] https://rdf4j.org/documentation/tutorials/sparqlbuilder/

[53] https://rdf4j.org/documentation/programming/shacl/

Virtuoso

Virtuoso's OpenLink architecture is shown in Figure 6[54]. It is a modular framework for working with any type of data from relational databases, XML sources, APIs and RDF triple definitions. The architecture uses an underlying relational store, converting triples and SPARQL queries to a SQL format. It currently stores (and allows querying) of data sizes of over 94B triples (e.g., the UniProt data[55] which is accessible from a single server[56]).

A SPARQL endpoint for Wikidata is available for testing at https://wikidata.demo.openlinksw.com/sparql/. (Note that the data currently loaded there is from a March 2020 RDF dump, of size 11B+ triples.)

[54] The Virtuoso architecture is described on the pages, http://wikidata.dbpedia.org/openlink-software-%E2%80%94-virtuoso-universal-server and http://docs.openlinksw.com/virtuoso/conceptarchitecture/. Figure 4 is captured from the first link.
[55] https://www.uniprot.org/
[56] https://sparql.uniprot.org/sparql/

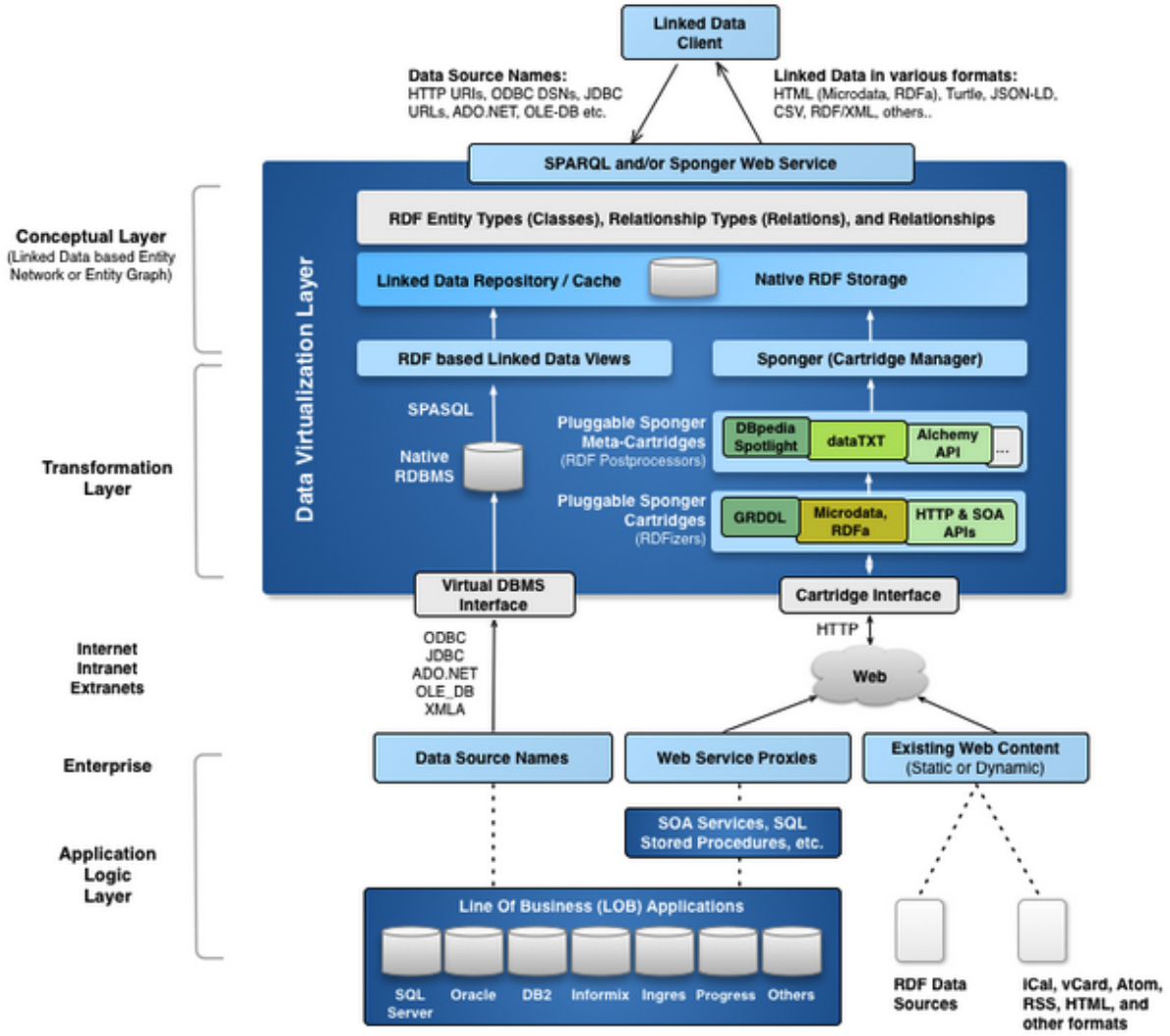


Figure 6. Virtuoso Architecture

Below, each of the evaluation criteria are listed with Virtuoso's score (from Table 1) and a short explanation. Note that the scoring is 0-5 (where 0 indicates no support and 5 indicates exceptional support).

- Scalability to 25B+ triples - 5
    - As noted, Virtuoso has been shown to scale to over 94B triples
- Scalability to 10B+ triples - 5
- Full support for SPARQL 1.1 with minimal errors - 3
    - Examining the current GitHub issues for Virtuoso[57], 356 of the 557 current issues deal with SPARQL. However, some are about queries running against Docker images or deal with data access issues. Examining the first page of results, 5 of the 25 issues (a fifth) could be excluded as not

[57] https://github.com/openlink/virtuoso-opensource/issues?q=is%3Aissue+is%3Aopen+SPARQL+

relevant to the Wikidata scenarios. Assuming a worst case of two-fifths not relevant, that still leaves 38% of the open issues. The oldest SPARQL issue that is still open appears to be from March 2012.

- DotNetRDF.org reports[58]: “Note that SPARQL Query and Update is subject to the peculiarities of the Virtuoso implementation which is well known for having various quirks and non-standard SPARQL extensions.”
- The complexity of converting from SPARQL to SQL can be seen in the Virtuoso’s documentation on troubleshooting SPARQL queries[59]:
  - “A short SPARQL query can be compiled into a long SQL statement, especially if data comes from many quad map patterns. A moderately sized application with 50 tables and 10 columns per table may create thousands of quad map patterns for subjects spanning hundreds of different types ... Thus it is to be expected that some queries will be rejected even if the same queries would work fine if the RDF data were held as physical quads in default storage, rather than synthesized through a Linked Data View.”
  - “SPARQL uses IRIs that are long and sometimes unreadable, but there is no "closed world" schema of the data so a typo in an IRI is not an error; it is simply some other IRI. So a typo in an IRI or in a namespace prefix causes missing bindings of some triple patterns of the query and an incomplete result, but usually no errors are reported. A typo in a graph or predicate IRI may cause the SPARQL compiler to generate code that accesses default (quad) storage instead of a relational source or generate empty code that accesses nothing.”
  - “The SQL compiler does not signal casting errors when it runs the statement generated from SPARQL, because the generated SQL code contains *option (QUIETCAST)* . This means that mismatches between expected and actual data types of values stay invisible and may cause rounding errors … and even empty joins ...”

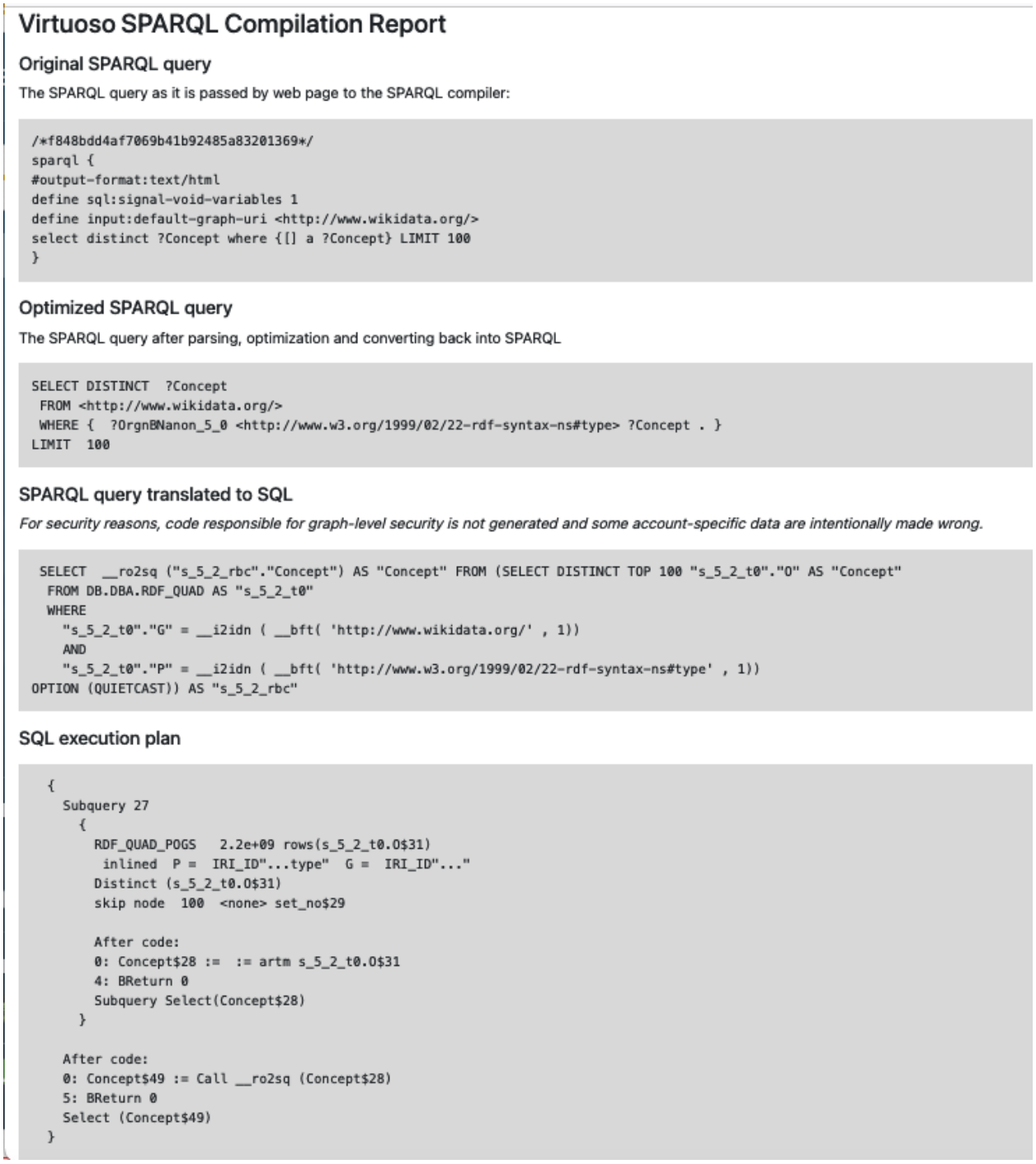
[58] https://dotnetrdf.org/docs/stable/user_guide/Storage-Virtuoso.html
[59] http://docs.openlinksw.com/virtuoso/sparqldebug/

- Federated query - 5
    - Per the Medium article, *What is a Virtuoso SPARQL Endpoint, and why is it important?*[60], Virtuoso supports federated query
    - In order to enable queries using the SERVICE keyword, appropriate permissions must be given[61]
- Ability to define custom SPARQL functions - 4
    - Custom functions are written as SQL stored procedures[62]
    - Because the functions are not written in code, but as SQL statements, the assigned score is 4
- Ability to tune/define indexes and perform greater-than/less-than checking - 4
    - Default RDF indexes are PSOG and POGS (which are full indexes over quad data) and SP, OP and GS (which are partial indexes). This approach favors queries where the predicate is specified (which is true for the majority of the Wikidata queries).
    - Indexes are column-wise by default in V7, but due to the high write frequencies required by the Wikidata stream updater, a column-store is likely not viable ("One should not use column-wise storage in cases where columns are frequently updated, especially if a single row is updated per statement. This will give performance substantially worse than row-wise storage."[63])
    - Alternate indexing schemes are possible[64] but are discussed in the context of the column store, and require testing. For this reason, the assigned score is 4.
    - Since range lookups are common in SQL, this function is supported as part of the SPARQL to SQL translation
- Support for R/W at high frequency - 3
    - As noted in the "indexes" bullet, Virtuoso is tuned for bulk-load with high frequency read, and not for read/write
    - Note that the SP, OP and GS (partial) indexes do not store duplicates (e.g., for the GS index, a subject will have one entry, even if it has many predicates) and entries in these indexes are not deleted. "[O]ver time, especially if there are frequent updates and values do not repeat between consecutive states, the SP, OP and GS indices will get polluted, which may affect performance. Dropping and recreating the indexes will remedy this situation."
    - The Wikidata environment of high frequency reads/writes needs to be tested. For this reason, the assigned score is 3.

[60] https://medium.com/virtuoso-blog/what-is-a-virtuoso-sparql-endpoint-and-why-is-it-important-5244df738a3e
[61] https://community.openlinksw.com/t/enabling-sparql-1-1-federated-query-processing-in-virtuoso/2477
[62] http://docs.openlinksw.com/virtuoso/rdfsqlfromsparql/
[63] http://docs.openlinksw.com/virtuoso/colstore/
[64] http://docs.openlinksw.com/virtuoso/colstore/

- Active open-source community - 3
    - Virtuoso's open-source code is stored at https://github.com/openlink/virtuoso-opensource, licensed under the GNU General Public License and written in C
    - There appear to be few active committers, although that is likely due to Virtuoso's development approach (Per the comment in the Virtuoso GitHub issues[65]: "'[D]ue to the patterns of our development work, PRs against the VOS repos can rarely be applied directly or completely -- but reproduction steps for any observable issues are always helpful, as are pointers to and suggested fixes for specific bugs in the code, which do help us resolve them in our internal codebase, and then apply those fixes to the VOS ...")
- Well-designed and documented code base - 2
    - Code is written in C and has a complex directory structure with many features beyond RDF and SPARQL. Although these are valuable in certain scenarios, they complicate the code base.
- Instrumentation for data store and query management - 4
    - There is a SQL command line interface[66] (ISQL) with the ability to invoke numerous commands, including "status"[67]
    - In addition, the system can be administered from the Virtuoso Conductor[68] browser interface, and tested/maintained through HTTP REST commands or using stored procedures
- Query plan explanation - 2
    - Query plan explanations can be generated from the SPARQL Query Editor interface (as can be seen at the test endpoint noted above) by selecting the checkbox, "Generate SPARQL compilation report" (at the bottom of the screen) and then clicking the "Explain Query" button. Sample output is shown in Figure 7.
    - Alternately, explanations are generated using the ISQL command line interface[69]
    - Due to the complexity of the SQL translation (a user will need to understand both SPARQL and SQL), the assigned score is 2

[65] https://github.com/openlink/virtuoso-opensource/issues/965
[66] http://docs.openlinksw.com/virtuoso/invokingisql/
[67] http://docs.openlinksw.com/virtuoso/fn_status/
[68] http://docs.openlinksw.com/virtuoso/conductorbar/
[69] http://docs.openlinksw.com/virtuoso/rdfperfcost/ and http://docs.openlinksw.com/virtuoso/fn_explain/



Virtuoso SPARQL Compilation Report

Original SPARQL query

The SPARQL query as it is passed by web page to the SPARQL compiler:

```
/*f848bdd4af7069b41b92485a83201369*/
sparql {
#output-format:text/html
define sql:signal-void-variables 1
define input:default-graph-uri <http://www.wikidata.org/>
select distinct ?Concept where {[] a ?Concept} LIMIT 100
}
```

Optimized SPARQL query

The SPARQL query after parsing, optimization and converting back into SPARQL

```
SELECT DISTINCT  ?Concept
 FROM <http://www.wikidata.org/>
 WHERE {  ?OrgnBNanon_5_0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?Concept . }
LIMIT  100
```

SPARQL query translated to SQL

For security reasons, code responsible for graph-level security is not generated and some account-specific data are intentionally made wrong.

```
 SELECT  __ro2sq ("s_5_2_rbc"."Concept") AS "Concept" FROM (SELECT DISTINCT TOP 100 "s_5_2_t0"."O" AS "Concept"
  FROM DB.DBA.RDF_QUAD AS "s_5_2_t0"
  WHERE
    "s_5_2_t0"."G" = __i2idn ( __bft( 'http://www.wikidata.org/' , 1))
    AND
    "s_5_2_t0"."P" = __i2idn ( __bft( 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type' , 1))
OPTION (QUIETCAST)) AS "s_5_2_rbc"
```

SQL execution plan

```
 {
   Subquery 27
     {
       RDF_QUAD_POGS   2.2e+09 rows(s_5_2_t0.O$31)
        inlined  P =  IRI_ID"...type"  G =  IRI_ID"..."
       Distinct (s_5_2_t0.O$31)
       skip node  100  <none> set_no$29

       After code:
       0: Concept$28 :=  := artm s_5_2_t0.O$31
       4: BReturn 0
       Subquery Select(Concept$28)
     }

   After code:
   0: Concept$49 := Call __ro2sq (Concept$28)
   5: BReturn 0
   Select (Concept$49)
 }
```

Figure 7. Virtuoso Query Explanation

- Query plan tuning/hints - 3
    - To override the query optimization order, an explicit instruction is added to the beginning of the SPARQL query (i.e., DEFINE sql:select-option "order")

    - Note that the effect of *sql:select-option* is “pervasive, extending inside unions, optionals, subqueries etc.”[70]
- Query without authentication - 5
    - Access with no authentication can be seen on the Wikidata test page at https://wikidata.demo.openlinksw.com/sparql/
- Ability to prevent write access, except by the stream updater - 5
    - SPARQL endpoint protection (for SPARQL_UPDATE) can be assigned using WebID, OAuth or SQL accounts[71]
- Data store reload in 2-3 days (worst case) - 1
    - As documented[72], the latest-all-nt dump file must be downloaded and split into hundreds (or thousands) of smaller files to be efficiently bulk loaded[73] (or smaller graph-specific files must be used, as will be the case for Apache Jena or RDF4J loading)
    - After this step, the geospatial data must be split out and removed (which takes additional time and appears to be what was done for the OpenLink implementation[74]) or Virtuoso must be patched[75] (which may cause other problems in the future)
    - As reported by OpenLink[76], after the above processing, an 11B triple Wikidata dump (from March 2020) was loaded in 10 hours using 8 parallel loaders
- Query timeout and resource recovery - 3
    - Query timeout can be globally defined, but cannot be set at the individual query level (this is the main reason for the assigned score of 3)
    - It should be noted that in many Virtuoso environments, queries will return results (not time out) due to a feature known as “anytime query”[77]
    - “Anytime query” returns partial, non-deterministic results if a query’s execution or connection timeout is exceeded
    - Due to the non-deterministic nature of “anytime query”, it is unlikely to be enabled for the Wikidata environment
- Support for geospatial (POINT) data - 5
    - See the bullet below for GeoSPARQL support
- Support for GeoSPARQL - 3
    - Virtuoso supports GeoSPARQL geometry functions on top of data that must be represented as Well-Known Text (WKT) strings[78]

---

[70] http://docs.openlinksw.com/virtuoso/rdfperfcost/
[71] http://docs.openlinksw.com/virtuoso/sparqlendpointprotection/ and the following pages
[72] https://community.openlinksw.com/t/loading-wikidata-into-virtuoso-open-source-or-enterprise-edition/2717 and https://stackoverflow.com/questions/56768463/wikidata-import-into-virtuoso
[73] http://vos.openlinksw.com/owiki/wiki/VOS/VirtBulkRDFLoader
[74] https://community.openlinksw.com/t/loading-full-wikidata-latest-ttl-dump-into-vos/1880
[75] Virtuoso non-terrestrial geo-literals bugfix, https://github.com/asanchez75/virtuoso-opensource/commit/5d7b1b9b29e53cb8a25bed69f512a150f9f05d50
[76] https://community.openlinksw.com/t/loading-wikidata-into-virtuoso-open-source-or-enterprise-edition/2717
[77] http://docs.openlinksw.com/virtuoso/anytimequeries/
[78] http://vos.openlinksw.com/owiki/wiki/VOS/VirtGeoSPARQLEnhancementDocs

    - Per the paper, *A GeoSPARQL Compliance Benchmark* (referenced in footnote 21), Virtuoso's implementation is 63.46% compliant
    - But, as noted in the data load bullet above, there is a bug in the implementation regarding non-terrestrial coordinates that affects Wikidata
- Support for named graphs (quads) - 5
    - As noted in the bullet regarding indexes

- Query builder interface - 5
    - A "SPARQL Query Editor" interface exists and is currently exposed for Wikidata testing at https://wikidata.demo.openlinksw.com/sparql/
    - In addition, an open-source Interactive SPARQL Query Builder interface[79] (iSPARQL) exists
    - Note that there are no capabilities for autocomplete in either interface
- Dataset evaluation by SHACL (SHape Constraint Language), ShEX (Shape EXpressions), or similar - 0
    - There is no support (or planned support) for SHACL or ShEX in the open-source version of Virtuoso
    - In addition, something similar to RDF4J cannot be used "in front of" Virtuoso (RDF4J does support SHACL) since Virtuoso's RDF4J Provider does not implement the interfaces required by RDF4J's ShaclSail[80]

Other Data Stores

In the course of this work, many other possible Blazegraph alternatives were found. The following list indicates the reason(s) why a possible solution was eliminated. Note that none of the proprietary RDF/SPARQL solutions were considered since they did not meet the mandatory criterion of being open-sourced.

- Open-source but early in development or for research purposes (not production):
    - gStore - Very incomplete SPARQL 1.1 implementation (no property paths, limited FILTER support, ORDER BY only for single variable, some characters such as tags, '<' and '>' not allowed); Some supporting documentation only available in Chinese
    - OxiGraph - Early in development (Version 0.3); Targeted as embedded store; Identified as "hobby project"
    - Wukong - Early in development (Version 0.2) with last code update Dec 2019
    - MillenniumDB - Very early in development; Missing ability to modify the database, OPTIONAL functionality, support for dates and lists, FILTER functionality, and more

[79] http://wikis.openlinksw.com/OATWikiWeb/InteractiveSparqlQueryBuilderOverview and https://github.com/openlink/iSPARQL
[80] https://github.com/openlink/virtuoso-opensource/issues/660

- Open-source but no recent updates:
  - Apache Rya - No recent activity for this project and JIRA issues appear to be languishing
- Open-source but no SPARQL support or problematic SPARQL infrastructure:
  - TerminusDB - No SPARQL support
  - LevelGraph - No SPARQL support
  - CM-Well - Non-native support for SPARQL requiring a separate load of data to a Jena instance; No support for ASK, DESCRIBE
  - quadstore - Designed as a client-side store with local query support; No SPARQL endpoint or support for federated query
  - SANSA-Stack - Incomplete SPARQL 1.1 support (no federated query, no property paths, EXISTS/NOT EXISTS, IN/NOT IN, … functions not supported); SPARQL support based on ontop to translate to SQL
  - Atomic Data Rust - No SPARQL support
  - DataCommons Mixer - Designed for GCP (Google Cloud Platform) and GKE (Kubernetes); SPARQL support is limited to a simple query structure: Prologue (prefixes), Select (variable name with DISTINCT), Where (an array of triples of the form, subject-predicate-array of objects), Orderby and Limit
- Open-source but unlikely to scale to billions of triples:
  - Corese - Expected to scale to 50M~100M triples per server
  - Parliament - Research implementation
  - LUPOSDATE - Research implementation
- Open-source but no backing store:
  - ontop - Also incomplete SPARQL support
- No development within last 2 years or more:
  - 4Store, AdaptRDF, Akutan, CliqueSquare, CumulusRDF, H2RDF, Halyard, HBase-RDF, Jena-HBase, Mulgara, RDFDB, Redland/RedStore, Wukong, …

Wikidata Query Architectures

Due to the lack of distributed, open-source solutions, the Wikidata Query architecture will remain based on a single engine/SPARQL server design, operating on commodity hardware, within a load-balanced cluster. This also permits individuals and enterprises to more easily host their own instances, in a cost-effective manner.

Note that it is possible to load the complete Wikidata graph into two of the alternative engines (QLever and Virtuoso). For the other options (Apache Jena and RDF4J), Wikidata will have to be split into at least 2 graphs, each of which are loaded into a different server instance. The latter will then require that SPARQL queries (that need access to the complete Wikidata graph) will have to be addressed to one of the servers, with a SERVICE clause federating the data from the other.

These implications and how the alternative infrastructures will be tested and exercised are briefly discussed in the next section.

Next Steps

There are several important tasks that must be completed before finally selecting a Blazegraph alternative. These are to:

- Determine how to support the current local SERVICE functions (labeling, geospatial calculations, etc.) in a more SPARQL-compliant manner
    - And then assessing the complexity of implementing these on the different engines
- Define a set of update and query workloads that exercise the engines and SPARQL endpoints
- Determine a set of optimal tuning parameters and indexes utilizing the details and results of the workload tests
- Define and test different algorithms for splitting the Wikidata graph, understand how the update and query workloads would change, and the implications for the RDF stream updater
- Investigate creation of a middleware layer (between the RDF store/SPARQL endpoint and users/applications) to remove dependencies on a specific implementation and reduce churn in potential, future migrations

This work has already begun and the community will be updated as new work products become available.