





Mitigating Query Rot: Using snapquery for Sustainable SPARQL Query Set Management

Wolfgang Fahl^{1,2}, Christoph Lange^{1,3}, Tim Holzheim¹, and Stefan Decker^{1,3}

¹ RWTH Aachen University, Computer Science i5, Aachen, Germany

² BITPlan GmbH, Willich, Germany

³ Fraunhofer FIT, Sankt Augustin, Germany

Abstract. Will the “cats” Wikidata SPARQL query example still work in the future? While link rot has been a known issue, the *Query Rot* problem has not been investigated much in the past. We introduce an approach for quality assessment and refactoring of query sets and evaluate its implementation for SPARQL.

By applying information hiding and dependency inversion principles, we hide the details of the query and introduce the *snapquery* SPARQL query endpoint middleware to maintain expected query behavior independently of technical details, context, and even, as a future option, the query language. “snapquery cats” is designed to work consistently in the future no matter what changes. This approach enables the swapping of endpoints, conceals the complexity of (federated) queries, and manages SPARQL query sets by making the queries FAIR first-class citizens of the Knowledge Graph infrastructure. The increased abstraction aligns well with state-of-the-art Artificial Intelligence approaches that allow the use of natural language input to generate query sets.

This novel, systematic, and semi-automatic approach is generally useful in most knowledge graph management scenarios. It allows for the gathering of query metadata from real-world environments on the fly, supporting the creation of test suites, benchmarks, challenges, dashboards, and other analytical applications for query performance and health monitoring. Our approach contributes to advancing knowledge engineering by bridging gaps between knowledge graphs, software engineering, and large language models.

We reproduce use cases of the Scholia, QLever and general Wikidata projects to demonstrate functionality and measure non-functional quality improvements over those projects.

We conclude that advancing Knowledge Graph management requires 1. introducing queries as FAIR first-class citizens by developing platform-independent named parameterized queries (thus confirming a 2019 proposal), 2. structured, implementation-independent error messages, and 3. shifting to higher abstraction levels that support human and machine generated general text input. We suggest corresponding improvements to the SPARQL standard.

Keywords: SPARQL · Query Rot · Knowledge Graph Management · Quality Assessment · Refactoring · snapquery · Benchmarking · Sustainability

1 Introduction

Will the “cats” Wikidata SPARQL query example still work in the future? Listing 1.1 shows the SPARQL code and Table 1 the first six result rows⁴.

Listing 1.1: SPARQL Query for Cats

```
# "Cats" example SPARQL query
# https://www.wikidata.org
# /wiki/Wikidata:SPARQL_query_service/queries/examples#Cats
SELECT ?item ?itemLabel
WHERE
{
  ?item wdt:P31 wd:Q146. # Must be a cat
  SERVICE wikibase:label {
    bd:serviceParam
      wikibase:language "[AUTO_LANGUAGE],en".
  }
}
```

Table 1: Cats query result

item	itemLabel
http://www.wikidata.org/entity/Q378619	CC
http://www.wikidata.org/entity/Q498787	Muezza
http://www.wikidata.org/entity/Q677525	Orangey
http://www.wikidata.org/entity/Q851190	Mrs. Chippy
http://www.wikidata.org/entity/Q893453	Unsinkable Sam
http://www.wikidata.org/entity/Q1050083	Catmando
...	...

The *Cats* query most likely will not run in a few years due to *Query Rot*. The query might not run on endpoints other than the currently in-use “Blazegraph” SPARQL endpoint provided as the Wikidata Query Service by the Wikimedia Foundation.

Semantic web technologies that utilize RDF and SPARQL (see Section 3.2)) are commonly used as Knowledge Graph infrastructure. A prevalent issue analogous to link rot (see Section 3.3) in web content is Query Rot for the necessary data retrieval components. Query Rot refers to the gradual deterioration of query validity over time due to changes in the *Query Execution Context (QEC)* (see Section 4). The crucial reliability, robustness, and other non functional quality aspects of KG systems are challenged by Query Rot.

Query Rot is mitigated by repeatedly refactoring queries – an error-prone, manual and time-consuming task, which can only be done by knowledge engi-

⁴ The table was generated with the command line `snapquery -queryName cats -limit 6 -format latex`

neering experts. *snapquery* is a tool and method that facilitates the refactoring of queries using a systematic approach, which can be automated and accessed via (standard compliant) APIs as a middleware. Queries are treated as first-class FAIR [35] citizens having a PID⁵ in the form of a domain/namespace/name combination with a link to the original query source(s). The principle of information hiding is applied via the Dependency Inversion Principle (see Section 3.1) (part of the SOLID approach in software engineering). Queries are divided into a *black-box* abstract part that only provides a query signature, consisting of the query PID and its parameters and variables and the *white-box* part detailing the Query Execution Context.

The snapquery separation of concerns targets different audiences - the black-box part for general end users and system integrators and the white-box part for scholars, developers, people running the infrastructure and others with an interest in all graphic detail.

snapquery was introduced at the Wikimedia 2024 Hackathon [6], motivated by needs of the Scholia community arising from the upcoming Wikidata graph split (see Section 2.1).

snapquery allows for faster systematic and semi-automatic refactoring of queries by assigning metadata about functional and non-functional aspects to each named query and grouping sets of queries by domain and namespace. The knowledge about whether a query (or set of queries) works at all in a given Query Execution Context 2.2 and, e.g., how fast and reliable it runs is stored as metadata and used as a basis for standard refactoring and user feedback activities. The metadata is gathered proactively and automatically and annotated and analyzed with *Large-Language-Model (LLM)* support.

We aim at changing the mindset of KG project stakeholders working with SPARQL query sets. Our approach has potential value in mitigating software architecture and engineering challenges that plague current knowledge engineering projects.

This paper is structured around three primary use cases which are presented in section 2.1:

- **Wikidata** – SPARQL examples, tutorials and usage
- **Scholia** – Scholarly publishing
- **QLever** – SPARQL engine development

Section 2 introduces and defines the term *Query Rot* based on a black-box and white-box definition of *Query Execution Context* targeted to the audiences mentioned above. Section 3 contains prior work and work we build on. Section 4 presents snapquery as a method and tool for mitigating Query Rot. Section 5 details the implementation and Section 6 the evaluation of snapquery. Section 7 concludes and suggests future work.

Further background material and research documentation is available at Category:snapquery [8] in our research wiki.

⁵ FAIR and PIDs are well explained in our CEUR-WS semantification work [10]

2 Query Rot

2.1 Motivating use cases

Table 2: Motivating Usecases and Query Sets

Usecase	Namespace@Domain	Name Example	# Queries
Wikidata	examples @wikidata.org	Cats	302
	short-url @wikidata.org	Japanese_Libraries_Details	100
	federated-queries @bitplan.com	Editors of the WOP 2014	11
Subtotal			413
Scholia	named-queries @scholia.toolforge.org	author_list-of-publications	373
	challenge @ceur-ws.org	AllVolumes	28
	WikidataThesisToolkit @wikidata.org	LSE-doctoral-theses	27
Subtotal			428
QLever	issues-wikidata @qllever.cs.uni-freiburg.de	Issue858-query1	181
	performance-dblp @qllever.cs.uni-freiburg.de	All papers published in SIGIR	6
	examples @dblp.org	PublicationTypes	5
Subtotal			192
Total			1033

Table 2 shows the relevant use cases and query sets that have motivated the snapquery approach and tool and that are presented in this work.

Wikidata SPARQL examples and usage The Wikidata SPARQL Service examples wiki page (https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/) has accumulated over 300 queries since 2016, illustrating Wikidata’s usage. These queries are expected to work against the Wikidata service endpoint. However, this work reveals that this is not always the case and introduces methods to systematically analyze the reasons.

Wikidata’s query service (<https://query.wikidata.org/>) offers short URLs for entered queries. A random set of 100 such queries has been included for investigation in this study and automatically annotated with an LLM (see Section 6.4).

Additionally, a set of manually curated federated queries completes the Wikidata use case query sets, totaling 413 queries.

Scholia and Scholarly Publishing *Scholia*(<https://scholia.toolforge.org/>) is a project allowing users to search, browse, and analyze scholarly publishing data curated in the Wikidata knowledge graph [24]. It uses named parameterized queries, with Python and JavaScript as programming languages, SPARQL for querying, and Jinja Templates for query parameterization.

Scholia faces challenges due to Wikidata’s size limitations. The Blazegraph SPARQL engine backing Wikidata can hold up to 4 Terabytes, which is insufficient for the vast amount of scholarly publishing data available [11]. The Wikimedia Foundation’s decision to split the graph and migrate scholarly data to its own knowledge graph potentially invalidates all 373 current Scholia queries.

The query sets of 428 queries for the Scholia use case comes from multiple sources: 373 named queries extracted from Scholia’s GitHub repositorym 28 Semantic Publishing Challenge [18] queries presented in our research Semantic MediaWiki, and WikidataThesisToolkit [36] queries documented on a wiki page.

QLever SPARQL Engine Development QLever, developed at the University of Freiburg, is a high-performance SPARQL Engine written in C++ [3]. It is a potential replacement for Blazegraph as Wikidata’s main SPARQL engine. QLever is an open-source project [1] but is not yet feature-complete.

We proposed using Scholia queries as a test suite for QLever⁶. This work extends that idea by demonstrating how to construct a testsuite from SPARQL queries extractable from GitHub issues. A typical issue, such as the issue #896 CONCAT implementation⁷, follows a standard situation/action/expected result format, which can be used for systematic testing and development.

The query set of 192 queries for the QLever use case is derived from queries extracted from QLever’s GitHub issues, performance queries related to DBLP documented on a wiki page in the GitHub repository, and example queries from the QLever-UI portal for the DBLP example dataset.

2.2 Definitions

We present two complementary definitions of **Query Execution Context (QEC)**: Definition 2 – a concrete/white-box definition for scholars, developers, people running the infrastructure and others with an interest in all graphic detail. Definition 3 – an abstract/black-box definition for general end users, system integrators and the general public.

The white-box definition illustrates the high dimensionality of the solution space and can not be elaborated in all detail here, while the black-box definition describes the simplified problem space that we intend to focus on in the core *Query Rot* Definition 4.

Definition 1 (Named Parameterized Query).

A Named Parameterized Query Q is a tuple (N, P, V) , where:

⁶ <https://github.com/ad-freiburg/qlever/issues/859>

⁷ <https://github.com/ad-freiburg/qlever/issues/896>

- N is the unique name of the query, structured as `name--namespace@domain`
- $P = \{(p_1 : t_1), \dots, (p_k : t_k)\}$ is the set of typed input parameters
- $V = \{(v_1 : t_1), \dots, (v_m : t_m)\}$ is the set of typed output variables

The unique name convention is inspired by the Java Naming Convention [25].

Definition 2 (Query Execution Context (concrete/white-box)). A Query Execution Context QEC is a function $QEC : QS \times G \times B \times EE \times L \rightarrow R$, where:

- $QS = \{q_i \mid i \in I\}$ is a set of Named Parameterized Queries, where each q_i is of type Q as defined in Definition 1, and I is an index set,
- $G = (V, E)$ is the knowledge graph being queried,
- $B = \{b_1, b_2, \dots, b_k\}$ represents a set of boundary conditions,
- $EE = \{e_1, e_2, \dots, e_m\}$ represents a set of execution environment parameters,
- L represents the query language standard and feature set,
- $R = (S, T, M)$ is the result, where S is an output stream, T is the content type, and M is metadata about the query execution.

Definition 3 (Query Execution Context (abstract/black-box)). A Query Execution Context QEC is a function $QEC : QS \times G \rightarrow R$, where:

- $QS = \{q_1, q_2, \dots, q_n\}$ is a set of NamedQueries as defined in Definition 1,
- G is the knowledge graph being queried,
- $R = (S, T, M)$ is the result, where S is an output stream, T is the content type, and M is metadata about the query execution.

The black-box definition reduces the complexity of the problem space to $QS \times KG$, while the white-box definition exposes the full dimensionality of the solution space as $QS \times G \times B \times E \times L$. The abstraction enables more effective knowledge engineering by simplifying the user’s conceptual model while allowing for complex optimizations in the implementation and providing standardized APIs independent of the hidden details.

An example of a Query Execution Context in practice would consist of QS being the Scholia Queryset “`named_queries@scholia.toolforge.org`” (see Section 2.1) consisting of more than 373 queries in the context of Wikidata as the knowledge graph G . Boundary conditions B include legal rules such as data protection and copyright laws, and limits set by organizational rules. The execution environment EE describes the Wikimedia Foundation’s data center running a cluster of Blazegraph instances and the 1 minute time out for the public Wikidata Query Service. L represents the SPARQL 1.1 language with Blazegraph extensions.

For the specific query

`author_events--named_queries@scholia.toolforge.org`, we have:

$N = \text{author_events--named_queries@scholia.toolforge.org}$

$P = \{(author : Q80)\}$ – Q80 is the Wikidata ID for Tim Berners-Lee

$V = \{(Date: xsd:dateTime), (Event: IRI), (EventLabel: xsd:string), (EventUri: IRI), (Roles: xsd:string), (Locations: xsd:string)\}$

An example output row for $R = (S, T, M)$:

S: HTML table containing a row: Date: 2009-10-25,
 Event: <http://www.wikidata.org/entity/Q48026503>, EventLabel: The 8th International Semantic Web Conference, EventUri: /event/Q48026503, Roles: author,
 Locations: Washington, D.C.

T: HTML

M: {execution time: 0.4 seconds, result count: 25}

Definition 4 (Query Rot). *Query Rot is a phenomenon that occurs when a Query Execution Context QEC, which previously produced satisfactory results for all queries $q \in QS$, experiences degradation or failure of query executions due to changes in underlying system components, despite no relevant changes to the query set QS or knowledge graph G . Query Rot of a QEC manifests when for some q and some input parameter set of QS*

- q fails to execute,
- q returns unexpected, or inconsistent results, or
- the metadata of an execution q fails to meet specified functional or non-functional criteria or boundary conditions.

In the context of the white-box definition, Query Rot typically arises from unaccounted changes in:

- L : the query language standard or feature set,
- EE : the execution environment,
- B : the implicit or explicit boundary conditions, or
- G : the knowledge graph data or schema.

From the black-box perspective, Query Rot is observed as a change in the relationship between inputs (QS and KG) and output R , without apparent changes to these visible components.

3 Background and Related Work

3.1 Software Engineering Principles

Parnas introduced *Information Hiding* as a core principle of software engineering in his paper “On the Criteria to be Used in Decomposing Systems into Modules” [26]. He proposes modularization as a strategy to improve software quality. Parnas argues that instead of the conventional flowchart-based structure, a system decomposition approach should be applied: based on “information hiding”, it groups modules by hidden design decisions rather than processing steps.

Martin proposed the *Dependency Inversion Principle* [22] stating, “Depend upon Abstractions. Do not depend upon concretions”. This principle is a key strategy for achieving systematic information hiding. It advocates for the use of *black-box* intermediary interfaces to abstract away technical *white-box* implementation details, thereby reducing coupling between components and enhancing modularity, flexibility, and maintainability of software systems – thus reducing development time in the long run.

A comprehensive description of the state-of-the art of *Continuous Integration (CI)* and *Continuous Delivery (CD)* is given by Van Merode [23]

3.2 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) is the standard query language for RDF [5] data, recommended by the W3C. Since its introduction in 2008 [28], SPARQL has undergone several revisions, with SPARQL 1.1 being the current version as of 2013 [16].

SPARQL allows for complex queries on linked RDF datasets by mapping variables to solutions represented as multisets of triples. Note that the results might not necessarily be “truly tabular” [13].

SPARQL has been the focus of research for the past decades, prominently featured in conference series such as the International Semantic Web Conference (ISWC), the Extended Semantic Web Conference (ESWC), the World Wide Web Conference (WWW), and the International Conference on Knowledge Engineering and Knowledge Management (EKAW).

The traditional research approach applies a “white-box” view where the emphasis is on the technical and theoretical underpinnings and all details are explored such as in ontology design having researchers and developers as a target group.

The formal analysis of the semantics of the SPARQL language by Perez et al. [27] is an example. It establishes the algebraic foundation of SPARQL based on graph patterns and RDF triples, contrasting with relational algebra, which operates on tables and lacks native support for optional data and flexible pattern matching.

The “white-box” view with end-users and system-integrators as a target group is somewhat underrepresented. Paul Warren et al.’s work [32,31,33,34] is a notable exception. His work is rooted in the semantification of the BT Library, which is related to our scholarly publishing use case (Section 2.1). According to Warren [34], query expert knowledge is rare – 74% of his study participants claimed to have no knowledge at all. In a 2018 study [33], the minority of participants were end-users and large databases with billions of triples were in the majority to be targeted by query sets. Timeouts were named as the main difficulty with SPARQL.

Johannes Lorey proposes Latency, Throughput, Execution Time of Joins as metrics for the Quality of Query Execution Contexts [19]. He also worked on discovering query templates from query logs [21] which supports the proposal of introducing named parameterized queries. His PhD thesis “What’s in a Query: Analyzing, Predicting, and Managing Linked Data Access” [20] elaborates on the details.

The SPARQL standard does not call for enforcing structured error messages on failure: “The response body of a failed query request is implementation defined. Implementations may use HTTP content negotiation to provide human-readable or machine-processable (or both) information about the failed query request.” [15]

The introduction of parameterized queries to SPARQL was proposed as an addition to the W3C Standard by Vladimir Alexiev [2] in 2019. As of 2024 there are still only implementation specific solutions.

3.3 Link Rot

Link rot describes cases where hyperlinks are getting invalid over time [29]. This happens when the target resource has been relocated to a new address (“re-route”) or has become permanently unavailable “dead”. A study on “link decay” by Hennessy for the time span 1999–2010 found that the median lifespan of web pages was 9.3 years [17]. Link rot breaks KG functionality and leads to frustrations, particularly in scholarly publishing contexts where citations rely on links [4]. One cause of Link rot is an orphaned responsibility for maintaining a link. Zhou et al. propose archiving links and predicting potential link failure to mitigate and avoid “404 Not Found” errors [37].

3.4 Query Rot

Query Rot The issue of query performance and stability over time (see Definition 4), has been indirectly addressed in various studies on SPARQL querying such as Verborgh et al. “Querying Datasets on the Web with High Availability”[30], but has not been explicitly named or systematically studied before our work.

4 Mitigating Query Rot using snapquery

The first step in mitigating Query Rot is to proactively identify when a query is no longer working as expected. Snapquery makes query sets available in computer readable form and has a command line and web interface to execute a query set against fitting endpoints and knowledge graphs. A whole Query Set of a project / Query Execution Context may be analyzed automatically this way, e.g., in a Continuous Integration (CI) and Continuous Delivery (CD) pipeline.

Figure 1 shows an example analysis. A difference in the number of failures and successes for a set of endpoints indicates potential Query Rot.

In the example, the endpoints “Wikidata” and “Wikidata-scatter” should behave 100% identically but they do not. Wikidata-qlever performing differently on the examples and named-queries namespaces is to be expected since it does not support Blazegraph special syntax and is not feature complete yet.

Legend: ✔ Distinct Successful Queries ✘ Distinct Failed Queries 📊 Total Successful Runs

Domain	Namespace	Total ↓	Wikidata	Wikidata-triply	Wikidata-qlever	Wikidata-openlinksw	Wikidata-scatter
scholia.toolforge.org	named_queries	373	✔109 ✘69 📊218	✔14 ✘312 📊28	✔11 ✘339 📊22	✔15 ✘346 📊30	✔108 ✘65 📊216
wikidata.org	examples	302	✔84 ✘21 📊168	✔5 ✘128 📊10	✔41 ✘249 📊82	✔48 ✘245 📊96	✔5 ✘0 📊10
qlever.cs.uni-freiburg.de	issues.wikidata	181	✔5 ✘176 📊10	✔33 ✘91 📊66	✔106 ✘32 📊212	✔83 ✘21 📊166	✔0 ✘52 📊0
wikidata.org	short_url	100	✔86 ✘7 📊172	✔7 ✘75 📊14	✔11 ✘85 📊22	✔13 ✘85 📊26	✔52 ✘1 📊104
ceur-ws.org	challenge	23	✔17 ✘7 📊68	✔12 ✘1 📊58	✔19 ✘5 📊72	✔18 ✘5 📊106	✔17 ✘7 📊68

Fig. 1: Query Set Success Overview

4.1 Queries as FAIR first-class KG citizens

To make Queries FAIR, we propose to use persistent identifiers for Queries. Some SPARQL environments, such as the Wikidata Query Service and QLever, already offer the capability to create short URLs for queries, which may be used to uniquely identify a certain query text. Since the same query might have different textual representations such as versions or platform specific adaptations, this approach is not fit to create persistent identifiers. Instead we propose to create PIDs from unique fully identifying names as per Definition 1 "Cats--examples@wikidata.org" is such a PID that may be translated to a corresponding URL that is useable in the RDF context and for RESTful APIs.

4.2 Local Query Execution Context Knowledge Graph

We propose a query management Knowledge Graph based on the core entities Query, Endpoint, SPARQL-Engine, Graph and Execution which may be run locally.

Using such a local KG snapquery overcomes current limitations of SPARQL by introducing platform-independent named parameterized queries and structured error reports ⁸.

Figure 2 shows a UML class diagram for the core entities of the snapquery Query Execution Context KG.

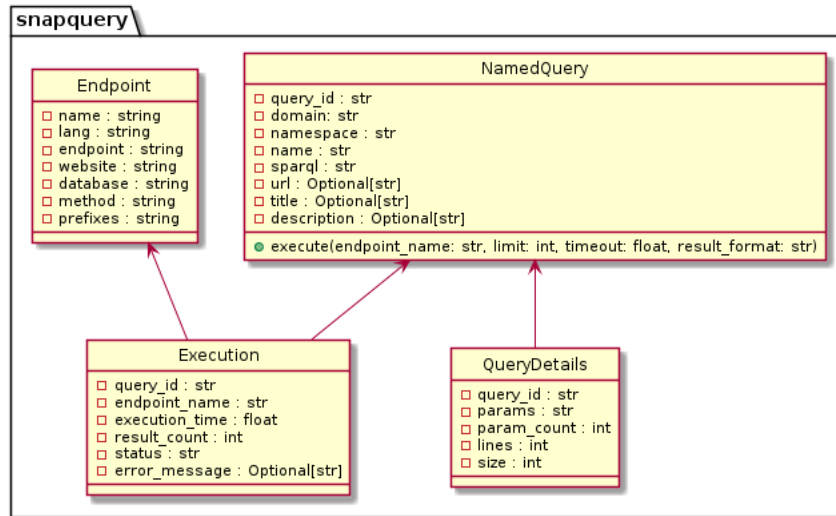


Fig. 2: UML class diagram for core QEC entities

⁸ (see Section 7.2) for corresponding SPARQL standard changes as future work

4.3 Semantification Workflow for Query Sets

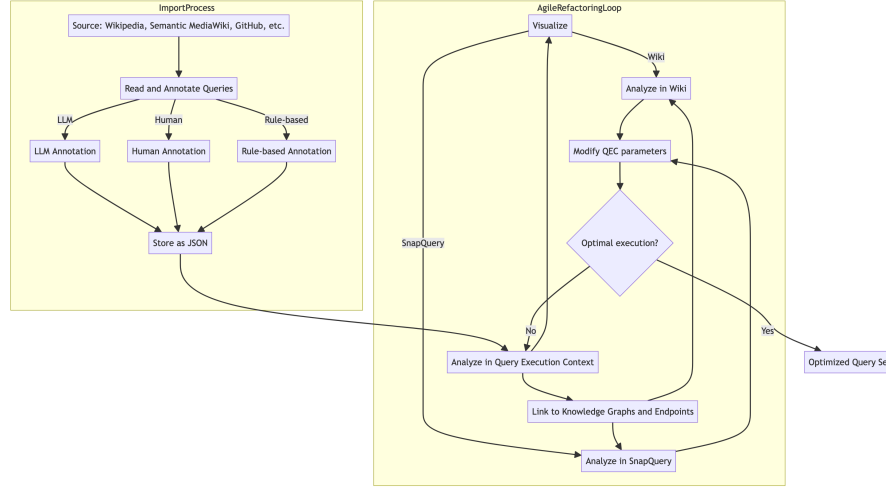


Fig. 3: Semantification Workflow for a Query Execution Context

The flowchart in Figure 3 shows the snapquery Workflow for Query Execution Contexts (QEC), including semantification and agile refactoring with the goal of QEC optimization (see also wiki page⁹).

The process starts with FAIR Query Set resources (e.g., Wikipedia pages, GitHub issues) containing computer-readable queries. These are annotated using LLMs, human input, or specialized software, and stored in JSON format for import and visualization.

The Agile Refactoring Loop analyzes queries within a QEC, links them to Knowledge Graphs and endpoints, and modifies QEC elements and queries to improve execution. This iterative process continues until satisfactory quality is achieved, balancing all QEC aspects. The goal is to be less error-prone, faster and need less sophisticated humans in the workflow and allow for proactive and automatic testing of query sets, e.g., as part of a CI/CD pipeline.

5 snapquery Implementation

The snapquery project is hosted on GitHub [14] and is available under the Apache-2.0 license. Links to further details and demos are found in the README.

The snapquery library is implemented in Python 3.9 or later. It utilizes a number of open source libraries for GUI, SPARQL, HTTP and Wiki handling,

⁹ https://cr.bitplan.com/index.php/QEC_Semantification_Workflow

authentication and other common tasks. Parts of the snapquery code have been developed with LLM support and a test-first attitude. snapquery is published via PyPI (version 0.0.12 as of 2024-07-12) and may be run locally.

snapquery is designed to be a SPARQL endpoint compatible middleware with RESTful APIs and a meta-query facility.

The snapquery design involves tradeoffs: it requires websockets and low-latency networks for optimal function, especially in real-time GUI interactions. Additionally, while aiming for compatibility with standard SPARQL endpoints, providing named parameterized queries is a non-standard feature necessitating client code modifications.

Running your own copies of Wikidata [12] is recommended but not necessary. We intend to make snapquery the backbone of an alternative Wikidata mirrors infrastructure as a mitigation measure.

6 Evaluation

6.1 Cats Wikidata Example

As stated in Section 1, most likely even the most prominent Wikidata SPARQL query example “Cats” will suffer from query rot. Applying the snapquery approach, we created an alternative query. Even though the query only consists of one query pattern, it uses the Wikidata label service¹⁰, thus making the query incompatible with other endpoints and query engines, as the label service is only available in the Blazegraph Wikidata infrastructure. This can be seen in Figure 4a, as the query only executes successfully on the Wikidata endpoint and on wikidata-scatter, a copy running the same infrastructure as Wikidata. On other endpoints running a copy of Wikidata with a different query engine, we see that the query fails every time with either a QueryBadFormed or EndpointInternalError error. By changing the query to directly query the *rdfs:label* instead of using the label service, the query becomes endpoint independent and executes on all endpoints successfully without increasing the execution time, as shown in Figure 4b.

6.2 Wikidata Thesis Toolkit

Query Rot can also appear when underlying data and schema changes, necessitating maintenance and updates for entire query sets. This challenge became particularly acute for the Wikidata Thesis Toolkit. Helen Williams and Ruth Elder, who are not SPARQL experts, have invested considerable time and effort¹¹ into creating a set of 27 queries for the toolkit. When they learned about the upcoming graph split of Wikidata¹², they feared their hard work might be invalidated. The prospect of evaluating and potentially adjusting each query for

¹⁰ a convention that xLabel is treated as a Label for x automatically

¹¹ alongside their primary job responsibilities.

¹² https://m.wikidata.org/wiki/Wikidata:SPARQL_query_service/WDQS_graph_split



Fig. 4: Execution statistics by endpoint of the cats query

correctness and reliability seemed daunting. We offered to demonstrate how the snapquery approach could address their concerns quickly and sustainably. The steps taken were: 1. importing the query set from the wiki page of the Wikidata Thesis Toolkit, 2. Monitoring the execution status for the complete set against multiple endpoints, 3. Quick determination of whether a failing query is due to endpoint issues or data changes requiring query adjustments.

To ensure compatibility, we evaluated all 27 queries, identifying and adjusting 6/27 queries affected by the split. Both original and adjusted query sets were executed against the graph split test endpoint for validation. This process took less than a day, significantly reducing Helen and Ruth’s anxiety about the impending changes.

6.3 General Failure reasons

snapquery provides an error message handling that filters and categorizes implementation dependent error message.

Table 3 shows error categories across different domains, namespaces, and endpoints as a result of trying the queries of Table 2 against different endpoints. Syntax errors are the most common issue, particularly for the

named_queries@scholia.toolforge.org namespace, while endpoint internal errors and timeouts also pose significant challenges, especially for the

examples@wikidata.org namespace. Systematic approaches may now be applied to solve the issues by further subcategorizing the errors.

6.4 LLM performance

Generating query annotations For the *short-url@wikidata.org* Query Set the annotation has been done automatically with gpt-3.5-turbo and gpt-4 via API. All 100 examples were successfully given a title, name and description. The

Table 3: Top Query failure error categories grouped by namespace@domain

Error Category	Total Namespace@Domain	Endpoint
Syntax Error	3790 named_queries@scholia (2044), examples@wikidata (788), issues.wikidata@qlever (612), ...	wikidata-triply (1100), wikidata-qlever (996), ...
EndPoint-Internal	1142 examples@wikidata (624), short_url@wikidata (242), named_queries@scholia (140), ...	wikidata-openlinksw (642), wikidata-qlever (474), ...
Timeout	399 examples@wikidata (180), issues.wikidata@qlever (148), ...	wikidata-triply (342), wikidata (57)
Service Unavailable	82 issues.wikidata@qlever (26), examples@wikidata (20), ...	wikidata-triply (82)
Other	56 named_queries@scholia (28), examples@wikidata (24), ...	wikidata (18), wikidata-openlinksw (14), ...
Too Many Requests	20 named_queries@scholia (12), examples@wikidata (4), ...	wikidata (18),...

result is in `wikidata-short-urls.json` in the samples of the GitHub project and available in the public demos. The prompt log is in our research wiki [7].

The average execution time was 3.2 secs. The total number of tokens for all 100 API calls was 0.1 million.

The precision, recall and F1 values are close enough to 1 (with gpt-4) on human inspection to not bother about the quality since the execution time and cost for the task can not be beaten by a human by a big margin. As of 2024-07 the cost would be less than a cent per annotation.

Since the code for this task is available in our repository, we could simply run it on thousands of Wikidata short URLs to get a large query set, which would be a good basis for further research. Providing the LLM annotation as service can easily be added to the query nomination feature of snapquery; this is just a matter of funding.

Query rewriting We intend to test query rewriting with our tool to find how well standard refactoring activities would be supported by different Conversational AI tools. For a pre-selection of capable LLM, we did an experiment that is documented in our research wiki [9]. The prompts were: **Rewrite to make blazegraph independent** followed by the Cats Wikidata example source code as shown in Section 1. If the LLM did not produce a working SPARQL query, a second attempt was made using the prompt: **the result does neither work on the wikidata query service nor on qlever**

Only one out of 4 LLMs could do the task on first attempt: ChatGPT-4o. Claude AI succeeded on the second attempt, Pi AI and Google Gemini failed. We intend to do further research with ChatGPT and Claude AI using the APIs.

7 Conclusion and Future Work

7.1 Conclusion

We introduce Query Rot and propose making Queries FAIR first-class citizens in a Knowledge Graph capturing Query Execution Context (QEC) metadata. The snapquery method and tool enable a semantification workflow for Query Sets, systematically improving QEC quality faster and with less expert support. This approach separates concerns using a black-box named parameterized view for end-users and system integrators, while hiding details of the white-box view (visible for developers and scholars). As a middleware, snapquery provides hot-swappable queries, allowing preparation and testing of alternatives for upcoming changes such as the Wikidata graph split. The Wikidata Thesis Toolkit case (27 queries) demonstrates snapquery’s ability to enable proactive refactoring on QEC changes, particularly benefiting non-SPARQL specialists. This approach can be extrapolated to larger cases such as Scholia (373 queries) and QLever (192 queries), potentially saving significant time and resources in maintaining and adapting larger query sets. By providing a framework for adapting to QEC changes, snapquery allows domain experts to focus on their primary tasks rather than worrying about knowledge graph query infrastructure.

7.2 Future Work

We intend to finalize the work on the three use cases presented – providing benchmarks, test suites and challenges based on the corresponding Query Sets. Prior SPARQL query log work may be revisited using the snapquery approach with the goal to make the results more stable against QEC changes along the same lines.

The LLM supported query refactoring activities for Scholia and QLever with the goal of integrating snapquery into the CI/CD pipelines will be in our focus.

Developing a comprehensive Query Execution Context Ontology is a worthwhile research goal.

We propose the following improvements to the W3C SPARQL standard:

1. Native support for named parameterized queries.
2. Standardized metadata for queries.
3. Structured error reporting.
4. Default prefix declarations per QEC.
5. Generally making queries FAIR first-class citizens.

7.3 Acknowledgements

We thank the Wikimedia hackathon 2024 contributors and the Scholia community and Hannah Bast and the QLever team for their open-source collaboration. Special thanks Helen Williams, Ruth Elder and Denis Priskorn for their valuable input. We acknowledge all researchers in Knowledge Graph management and SPARQL optimization. Finally, we appreciate our colleagues at RWTH Aachen University and Fraunhofer FIT for their feedback and support.

References

1. AD Freiburg: Qlever: The efficient query engine. <https://github.com/ad-freiburg/qlever> (2024), <https://github.com/ad-freiburg/qlever>, gitHub repository
2. Alexiev, V.: Query Parameterization · Issue #57 · w3c/sparql-dev — github.com. <https://github.com/w3c/sparql-dev/issues/57> (2019), [Accessed 12-07-2024]
3. Bast, H., Buchhold, B.: Qlever: A query engine for efficient sparql+text search. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. CIKM '17, ACM (Nov 2017). <https://doi.org/10.1145/3132847.3132921>, <http://dx.doi.org/10.1145/3132847.3132921>
4. Coble, Z., Karlin, J.: Reference rot in the digital humanities literature: An analysis of citations containing website links in DHQ. *Digit. Humanit. Q.* **17**(1) (2023), <http://www.digitalhumanities.org/dhq/vol/17/1/000662/000662.html>
5. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 Concepts and Abstract Syntax. <https://www.w3.org/TR/rdf11-concepts/> (2014), accessed: 2024-07-10]
6. Fahl, W.: T363894 Introduce Named Queries and Named Query Middleware to wikidata — phabricator.wikimedia.org. <https://phabricator.wikimedia.org/T363894> (2024), [Accessed 12-07-2024]
7. Fahl, W.: Workdocumentation 2024-05-12 - research wiki — cr.bitplan.com. https://cr.bitplan.com/index.php/Workdocumentation_2024-05-12#LLM_Query_name_annotation (2024), [Accessed 12-07-2024]
8. Fahl, W., Holzheim, T.: Category:Snapquery - research Wiki — cr.bitplan.com. <https://cr.bitplan.com/index.php/Category:Snapquery> (2024), [Accessed 12-07-2024]
9. Fahl, W., Holzheim, T.: Snapquery Cats LLM Rewrite Experiment - research wiki — cr.bitplan.com. https://cr.bitplan.com/index.php/Snapquery_Cats_LLM_Rewrite_Experiment (2024), [Accessed 12-07-2024]
10. Fahl, W., Holzheim, T., Lange, C., Decker, S.: Semantification of ceur-ws with wikidata as a target knowledge graph. In: Joint Proceedings of TEXT2KG 2023 and BiKE 2023. CEUR Workshop Proceedings (2023), https://ceur-ws.org/Vol-3447/Text2KG_Paper_13.pdf
11. Fahl, W., Holzheim, T., Lange, C., Decker, S.: Sempubflow: A novel scientific publishing workflow using knowledge graphs, wikidata, and llms – the ceur-ws use case (2024), manuscript submitted for publication
12. Fahl, W., Holzheim, T., Westerinen, A., Lange, C., Decker, S.: Getting and hosting your own copy of wikidata. In: 3rd Wikidata Workshop 2022. Proceedings. Fraunhofer-Gesellschaft (2022). <https://doi.org/10.24406/PUBLICA-976>, <https://publica.fraunhofer.de/handle/publica/437118>
13. Fahl, W., Holzheim, T., Westerinen, A., Lange, C., Decker, S.: Property cardinality analysis to extract truly tabular query results from wikidata. In: Kaffee, L.A., Razniewski, S., Amaral, G., Alghamdi, K.S. (eds.) Proceedings of the 3rd Wikidata Workshop 2022, co-located with the 21st International Semantic Web Conference (ISWC2022). CEUR Workshop Proceedings, vol. 3262. Wikidata Workshop 2022, CEUR Workshop Proceedings (Oct 2022), <https://ceur-ws.org/Vol-3262/paper7.pdf>
14. Fahl, W., Holzheim, T., Priskorn, D.: GitHub - WolfgangFahl/snapquery: Frontend to Introduce Named Queries and Named Query Middleware to wikidata — github.com. <https://github.com/WolfgangFahl/snapquery> (2024), [Accessed 12-07-2024]

15. Feigenbaum, L., Williams, G.T., et al.: Sparql 1.1 protocol. Tech. rep., W3C (2013), <https://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/>
16. Harris, S., Seaborne, A., et al.: Sparql 1.1 query language. Tech. rep., W3C (2013), <https://www.w3.org/TR/sparql11-query/>
17. Hennessey, J., Ge, S.X.: A cross disciplinary study of link decay and the effectiveness of mitigation techniques. *BMC Bioinformatics* **14** (2013). <https://doi.org/10.1186/1471-2105-14-s14-s5>
18. Lange, C., Di Iorio, A.: Semantic publishing challenge – assessing the quality of scientific output. In: *Communications in Computer and Information Science*, pp. 61–76. Springer International Publishing (2014). https://doi.org/10.1007/978-3-319-12024-9_8, https://doi.org/10.1007/978-3-319-12024-9_8
19. Lorey, J.: Sparql endpoint metrics for quality-aware linked data consumption. In: *Proceedings of International Conference on Information Integration and Web-based Applications & Services. IIWAS '13, ACM* (Dec 2013). <https://doi.org/10.1145/2539150.2539240>, <http://dx.doi.org/10.1145/2539150.2539240>
20. Lorey, J.: What’s in a Query: Analyzing, Predicting, and Managing Linked Data Access. Dissertation zur erlangung des akademischen grades doktor der ingenieurwissenschaften (dr.-ing.), University of Potsdam, Potsdam, Germany (Mar 2014)
21. Lorey, J., Naumann, F.: Detecting SPARQL Query Templates for Data Prefetching, p. 124–139. Springer Berlin Heidelberg (2013). https://doi.org/10.1007/978-3-642-38288-8_9, http://dx.doi.org/10.1007/978-3-642-38288-8_9
22. Martin, R.C.: Design principles and design patterns (2000), https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf, originally published on objectmentor.com. Archived by the Wayback Machine on September 6, 2015
23. van Merode, H.: *Continuous Integration (CI) and Continuous Delivery (CD): A Practical Guide to Designing and Developing Pipelines*. Apress (2023). <https://doi.org/10.1007/978-1-4842-9228-0>, <http://dx.doi.org/10.1007/978-1-4842-9228-0>
24. Nielsen, F.Å., Mitchen, D., Willighagen, E.: Scholia, scientometrics and wiki-data. In: *The Semantic Web: ESWC 2017 Satellite Events*. p. 237–259. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-70407-4_36, http://dx.doi.org/10.1007/978-3-319-70407-4_36
25. Oracle: Java naming conventions. <https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html> (1999), [Accessed 13-07-2024]
26. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. *Communications of the ACM* **15**(12), 1053–1058 (Dec 1972). <https://doi.org/10.1145/361598.361623>, <http://dx.doi.org/10.1145/361598.361623>
27. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)* **34**(3), 1–45 (2009)
28. Prud’hommeaux, E., Seaborne, A., et al.: SPARQL Query Language for RDF — w3.org. W3c recommendation, W3C (2008), <https://www.w3.org/TR/rdf-sparql-query/>, [Accessed 2024-06-26]
29. Tyler, D.C., McNeil, B.: Librarians and link rot: A comparative analysis with some methodological considerations. *portal: Libraries and the Academy* **3**, 615–632 (2003). <https://doi.org/10.1353/pla.2003.0098>

30. Verborgh, R., Hartig, O., Meester, B.D., Haesendonck, G., Vocht, L.D., Sande, M.V., Cyganiak, R., Colpaert, P., Mannens, E., de Walle, R.V.: Querying datasets on the web with high availability. In: *Lecture Notes in Computer Science, International Semantic Web Conference*. pp. 180–196. Springer, Springer International Publishing (2014). https://doi.org/10.1007/978-3-319-11964-9_12, http://dx.doi.org/10.1007/978-3-319-11964-9_12
31. Warren, P.: Knowledge management and the semantic web: From scenario to technology. *IEEE Intelligent Systems* **21**(1), 53–59 (Jan 2006). <https://doi.org/10.1109/mis.2006.12>, <http://dx.doi.org/10.1109/MIS.2006.12>
32. Warren, P., Alsmeyer, D.: The digital library: a case study in intelligent content management. *Journal of Knowledge Management* **9**(5), 28–39 (Oct 2005). <https://doi.org/10.1108/13673270510622438>, <http://dx.doi.org/10.1108/13673270510622438>
33. Warren, P., Mulholland, P.: Using SPARQL – The Practitioners’ Viewpoint, p. 485–500. Springer International Publishing (2018). https://doi.org/10.1007/978-3-030-03667-6_31, http://dx.doi.org/10.1007/978-3-030-03667-6_31
34. Warren, P., Mulholland, P.: A Comparison of the Cognitive Difficulties Posed by SPARQL Query Constructs, p. 3–19. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-61244-3_1, http://dx.doi.org/10.1007/978-3-030-61244-3_1
35. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.W., da Silva Santos, L.B., Bourne, P.E., Bouwman, J., Brookes, A.J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C.T., Finkers, R., Gonzalez-Beltran, A., Gray, A.J., Groth, P., Goble, C., Grethe, J.S., Heringa, J., ’t Hoen, P.A., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S.J., Martone, M.E., Mons, A., Packer, A.L., Persson, B., Rocca-Serra, P., Roos, M., van Schaik, R., Sansone, S.A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M.A., Thompson, M., van der Lei, J., van Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J., Mons, B.: The FAIR guiding principles for scientific data management and stewardship. *Scientific Data* **3**(1) (mar 2016). <https://doi.org/10.1038/sdata.2016.18>, <https://doi.org/10.1038/data.2016.18>
36. Williams, H.K.R., Elder, R.: Introducing the wikidata thesis toolkit. Paper presented at the CILIP Metadata and Discovery Group Conference: "ReDiscovery", IET Birmingham: Austin Court, Birmingham, United Kingdom (2023), <http://eprints.lse.ac.uk/120224/>, accessed: 2023-07-13
37. Zhou, K., Grover, C., Klein, M., Tobin, R.: No more 404s: Predicting referenced link rot in scholarly articles for pro-active archiving. In: *Proceedings of the 15th ACM/IEEE-CS Joint Conference on Digital Libraries*. ACM (2015). <https://doi.org/10.1145/2756406.2756940>