QuaQue

Design and SQL Implementation of Condensed Algebra for Concurrent Versioning of Knowledge Graphs

Jey Puget Gil, Emmanuel Coquery, John Samuel, Gilles Gesquière

DBKDA 2026: València, Spain. 2026-03-08 — 2026-03-12

Overview

1. Introduction

2. State of the Art

3. Methodology

4. Experimental Results

5. Conclusion

Introduction

Introduction

Context and Motivation

Knowledge Graphs (KGs) are adopted in industry and academia for representing complex information [9].

Challenge: As KGs become dynamic assets, there is a need for **concurrent versioning systems**:

- Support for branching and merging
- Analysis of concurrent states
- Minimal data redundancy

Application Domain: Urban planning with parallel modifications by multiple stakeholders

Introduction

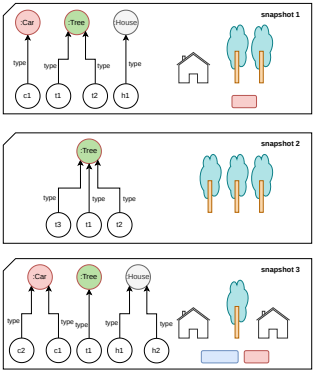
Problem Statement

Current Approaches:

- Named Graphs [9] to isolate versions
- Efficient for single-version access
- **Significant data duplication** across versions

Research Gap

Efficiently querying across **multiple concurrent versions** remains an open problem in database research.



Research Question: How to efficiently query across **multiple concurrent versions** of a Knowledge Graph while **minimizing data redundancy**?

Introduction

Our Contributions:

QuaQue is part of the ConVer-G project [7]:

1. Novel **condensed relational model** using bitstrings for version validity
2. Implementation of a **Condensed Algebra**
3. **SPARQL-to-SQL translator** leveraging PostgreSQL bitwise operations
4. Comparative benchmark demonstrating equivalent performance

Key Insight

Push version-filtering logic to the RDBMS using efficient **bitwise operations**.

State of the Art

State of the Art

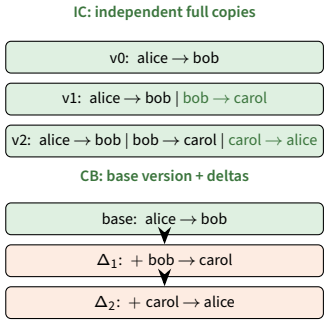
Versioning Strategies for RDF Data I

Independent Copies (IC) OSTRICH [11], SemVersion [13]

- Full copy per version: High storage redundancy
- Efficient single-version queries

Change-Based (CB) OSTRICH [11], R43ples [8], R&WBase [12]

- Base version + deltas: Space-efficient storage
- Costly reconstruction for queries



State of the Art

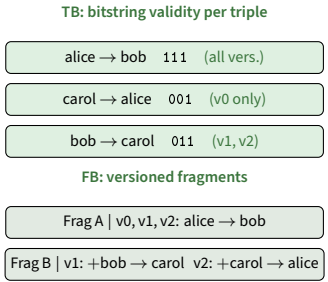
Versioning Strategies for RDF Data II

Timestamp-Based (TB) ConVer-G [7], Drydra [2], OSTRICH [11], v-RDFCSA [4]

- Validity intervals on triples
- Suited for linear evolution
- Complex for branching

Fragment-Based (FB) Quit Store [3]

- Independently versioned fragments
- Complex dependency management



State of the Art

Semantic Versioned Querying

Taelman et al. [10] formalize versioned query types:

Snapshot Retrieve data at a specific version

Longitudinal Track evolution across versions

Difference Identify changes between versions

Key Insight

Query languages should **natively support version-aware operations** rather than treating versioning as an afterthought.

State of the Art

The Case for Condensed Representation

Current Limitations:

- Storage vs. query efficiency trade-off
- Git-based solutions require expensive explicit checkout
- Cross-version analysis remains challenging

Our Approach:

1. **Condensed TB representation**: unique quads with version validity bitstrings
2. **Condensed Algebra**: operators for bitstring-annotated relations
3. **QuaQue Translator**: bridges algebra to efficient SQL execution

Methodology

Methodology

Core Concept: Bitstring-based version validity storage

Each **quad** (subject, predicate, object, graph) is stored **once** with a validity bitstring:

Table 1: Example of a versioned RDF dataset

Subject	Predicate	Object	Graph	Bitstring
:alice	ex:knows	:bob	:g1	111
:bob	ex:likes	“pizza”	:g1	011
:alice	ex:likes	“sushi”	:g1	101
:carol	ex:knows	:alice	:g2	001
:bob	ex:knows	:carol	:g2	011

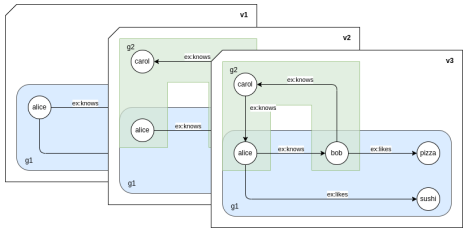


Figure 1: Graphical representation of the versioned RDF dataset

Methodology

Translation Process Overview

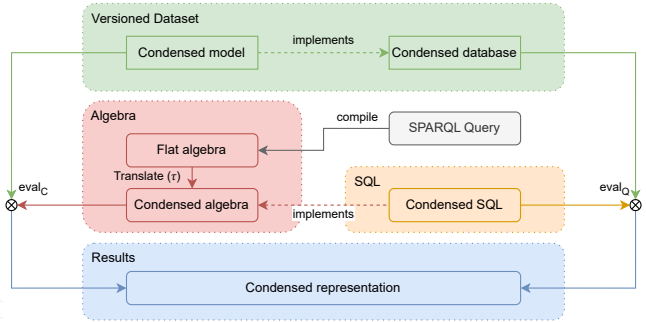


Figure 2: SPARQL to SQL translation in QuaQue

Methodology

Quad Pattern Translation

Query: "Retrieve all subjects and objects connected by `ex:knows` *in any graph."*

SPARQL quad pattern:

```
?s <ex:knows> ?o ?g .
```

Generated SQL:

```
SELECT t0.validity as bs$g, t0.id_subject
    as v$s, t0.id_object as v$o, t0.
    id_named_graph as ng$g
FROM versioned_quad t0
WHERE bit_count(t0.validity) <> 0
  AND t0.id_predicate = (id of <ex:knows>)
```

Table 2: Simplified evaluation of the quad pattern

v$s	v$o	ng$g	bs$g
:alice	:bob	:g1	111
:carol	:alice	:g2	001
:bob	:carol	:g2	011

Methodology

Join Operation with Bitwise AND

Query: “Find subjects that know someone who likes something, in any graph.”

SPARQL join:

```
?s <ex:knows> ?o ?g .
?o <ex:likes> ?liked ?g .
```

Key SQL construct:

```
SELECT (t0.validity & t1.validity) as bs$g, ...
FROM versioned_quad t0, versioned_quad t1
WHERE bit_count(t0.validity & t1.validity) <> 0
    AND t0.id_object = t1.id_subject
  AND t0.id_named_graph = t1.id_named_graph
```

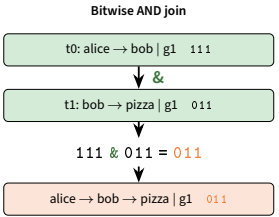


Table 3: Simplified evaluation of the join operation

v$s	v$o	v$liked	ng$g	bs$g
:alice	:bob	“pizza”	:g1	011

Methodology

Aggregate Operation

Query: "Count, across all versions, how many subjects know each object."

SPARQL aggregate:

```
SELECT ?o (COUNT(?s) AS ?count)
WHERE { ?s <ex:knows> ?o ?g . }
GROUP BY ?o
```

Generated SQL:

```
SELECT *, agg0 AS v$count
FROM (SELECT v$o, SUM(bit_count(bs$g)) AS agg0
    FROM (Quad Pattern) gp
GROUP BY (v$o)) ext
```

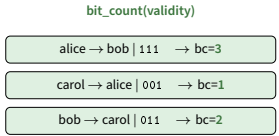


Table 4: Simplified evaluation of the aggregate operation

v$o	agg0 = v$count
:bob	3
:alice	1
:carol	2

Experimental Results

Experimental Results

Benchmark Setup

Environment: PAGODA cloud platform (LIRIS) with AMD EPYC 7443 24-Core Processor, 64GB RAM

Dataset: BEAR-B [5] with **daily (89 versions)** granularity (Time-Based policy)

- Available granularities: instant (21,046 versions), hourly (1,299 versions)
- Daily snapshot of DBpedia live data; $\sim$3.4M total versioned triples

Comparison:

- **QuaQue**: Condensed representation
- **QuaQue-flat**: Baseline (quad-version pairs stored separately)
- **Jena TDB2**: Native RDF triple store

Experimental Results

Storage Efficiency and Query Performance Results

Tool	Policy	Space (MB)	Join (ms)	Pred.-Obj. (ms)	Predicate (ms)
			idx: PSO	*idx: POS*	*idx: PSO/POS*
Jena TDB2	TB	**694.39**	8096	7798	7644
QuaQue-flat	TB	6489.91	7006	**7007**	6730
QuaQue	TB	4707.63	**6936**	7309	**6533**

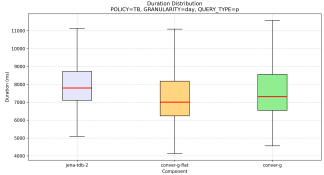
Storage and average query execution time (ms)

Improvements over Jena TDB2: Query: ~14% for predicate queries, ~6% for predicate-object queries, ~14% for join queries

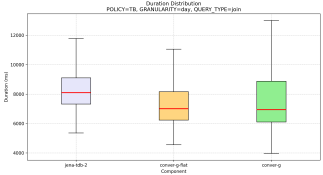
Deteriorations over Jena TDB2: Storage: 4.7 GB vs. 694 MB (~6.8x larger).

Experimental Results

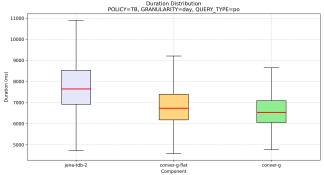
Benchmark: Query Time Distributions



Predicate queries



Join queries



Predicate-object queries

QuaQue exhibits **lower median times** on Join and Predicate-Object queries and **reduced variability** on Predicate-Object queries.

Conclusion

Conclusion

Summary of Contributions

1. **Condensed Relational Model**: Bitstring-based version validity storage
2. **QuaQue Translator**: SPARQL algebra to SQL with bitwise operations
3. **Benchmark**: Demonstrated equivalent performance over Jena TDB2

Key Result

QuaQue improves native RDF stores for multi-version query scenarios while leveraging standard SQL infrastructure.

Conclusion

Future Work

Planned Extensions:

- **Extended benchmarks**: Aggregate queries and different versioning policies (CB, IC)
- **Recursive queries**: Support for path traversal using Alpha operator [1]
- **Optimization**: Storage vs. performance trade-off analysis

New implementation:

- **In-memory version**: For faster query execution and reduced storage overhead
- **HDT-bitstring**: Compact RDF storage format with versioning support

Thank You

Questions?

Contact: jey.puget-gil@liris.cnrs.fr

Reproducibility:

- Open-source software (ConVer-G): `https://github.com/VCityTeam/ConVer-G`
- Benchmark results on Zenodo [6]: `https://zenodo.org/records/17780464`
- Docker containerized environment

References I

R. Agrawal.
Alpha: An extension of relational algebra to express a class of recursive queries.
IEEE Transactions on Software Engineering, 14(7):879–885, 2002.

J. Anderson and A. Bendiken.
Transaction-time queries in dydra.
MEPDaW/LDQ@ ESWC, 1585:11–19, 2016.

N. Arndt.
Distributed Collaboration on Versioned Decentralized RDF Knowledge Bases.
PhD thesis, Universität Leipzig, 2020.

References II

A. Cerdeira-Pena, A. Fariña, J. D. Fernández, and M. A. Martínez-Prieto.
Self-indexing rdf archives.
In *2016 Data Compression Conference (DCC)*, pages 526–535, 2016.

J. D. Fernández, J. Umbrich, A. Polleres, and M. Knuth.
BEAR – benchmark for rdf archive versioning systems.
`https://aic.ai.wu.ac.at/qadlod/bear.html`.
Accessed: 2025.12.01.

References III

J. P. Gil, E. Coquery, J. Samuel, and G. Gesquiere.
Quaque benchmark results.
`https://zenodo.org/records/17780464`.
Accessed: 2025.12.01.

J. P. Gil, E. Coquery, J. Samuel, and G. Gesquière.
Conver-g: Concurrent versioning of knowledge graphs.
arXiv preprint arXiv:2409.04499, 2024.

M. Graube, S. Hensel, and L. Urbas.
R43ples: Revisions for triples.
In *Proceedings of the 1st Workshop on Linked Data Quality*, 2014.

References IV

A. e. a. Hogan.
Knowledge graphs.
ACM Computing Surveys, 54(4):1–37, 2021.

R. Taelman, H. Takeda, M. Vander Sande, and R. Verborgh.
The fundamentals of semantic versioned querying.
In *SSWS2018, the 12th International Workshop on Scalable Semantic Web Knowledge Base Systems*, pages 1–14, 2018.

R. Taelman, M. Vander Sande, and R. Verborgh.
Ostrich: versioned random-access triple store.
In *Companion Proceedings of the The Web Conference 2018*, pages 127–130, 2018.

References V

M. Vander Sande, P. Colpaert, R. Verborgh, S. Coppens, E. Mannens, and R. Van de Walle.
R&wbase: git for triples.
LDOW, 996, 2013.

M. Völkel, W. Winkler, Y. Sure, S. R. Kruk, and M. Synak.
Semversion: A versioning system for rdf and ontologies.
In *Proceedings of the 2nd European Semantic Web Conference (ESWC)*, pages 193–202, 2005.

Condensed Relational Model

Bitstring-based version validity storage

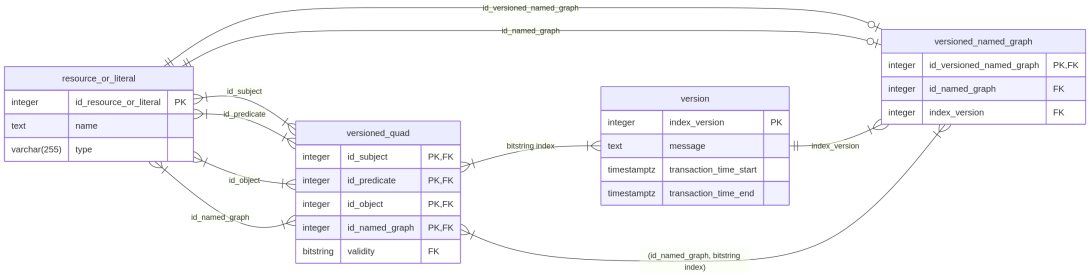


Figure 3: Relational model for condensed versioned RDF data