



**HAL**  
open science

## Hash-ssessing the freshness of SPARQL pipelines

Damien Graux, Fabrizio Orlandi, Declan O'Sullivan

► **To cite this version:**

Damien Graux, Fabrizio Orlandi, Declan O'Sullivan. Hash-ssessing the freshness of SPARQL pipelines. ISWC 2021 - International Semantic Web Conference, Oct 2021, Virtual Event, United States. hal-03516436

**HAL Id: hal-03516436**

**<https://hal.inria.fr/hal-03516436>**

Submitted on 7 Jan 2022

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution| 4.0 International License

# Hash-ssessing the freshness of SPARQL pipelines

Damien Graux<sup>1,2</sup> , Fabrizio Orlandi<sup>2</sup> , and Declan O’Sullivan<sup>2</sup> 

<sup>1</sup> Inria, Université Côte d’Azur, CNRS, I3S, France

<sup>2</sup> ADAPT SFI Centre, Trinity College Dublin, Ireland  
{grauxd,orlandif,declan.osullivan}@tcd.ie

**Abstract.** The recent increase of RDF usage has witnessed a rising need of “verification” around data obtained from SPARQL endpoints. It is now possible to deploy Semantic Web pipelines and to adapt them to a wide range of needs and use-cases. Practically, these complex ETL pipelines relying on SPARQL endpoints to extract relevant information often have to be relaunched from scratch every once in a while in order to refresh their data. Such a habit adds load on the network and is heavy resource-wise, while sometimes unnecessary if data remains untouched. In this article, we present a useful method to help data consumers (and pipeline designers) identify when data has been updated in a way that impacts the pipeline’s result set. This method is based on standard SPARQL 1.1 features and relies on digitally signing parts of query result sets to inform data consumers about their eventual change.

## 1 Introduction

During the past decades, the number of linked open datasets has rapidly increased<sup>1</sup>. These datasets are structured following the W3C standard Resource Description Framework (RDF) [6] and share knowledge on various domains, from the generalist ones such as DBpedia [1] or WikiData [7] to the most specialised ones, *e.g.* SemanGit [5]. This abundance of open datasets and SPARQL [2] endpoints led not only researchers but also businesses to integrate RDF graphs into their complex data pipelines. In particular, businesses are increasingly leveraging Semantic Web technologies to structure their own data and create value, sometimes integrating external Linked Data to enrich their analyses [4].

Benefiting from the two decades of developments made by the community, it is now possible to deploy Semantic Web pipelines and to adapt them to a wide range of needs and use-cases. Recent developments have been, for example, focused on distributed systems or on connecting Semantic Web data management systems together with non-RDF centric systems, paving the road to querying heterogeneous data. As a consequence of this increasing complexity of the use-cases, the pipelines themselves are getting more complicated, and often rely on several distinct data sources in order to compute their final results.

---

<sup>1</sup> From 2010 to 2020, the LOD-cloud has grown from 203 to 1 255 datasets, approximately: <https://lod-cloud.net/>

Hence, as data available may change, these pipelines (or parts of them) are frequently re-run in order to get fresher results. However, lots of times they are re-run unnecessarily as datasets have not been updated in the meantime in ways that impact the result sets of the pipeline. All these operations are leading to a waste of computation power and loads on the network.

In this article, mainly dedicated to SPARQL practitioners and data pipeline designers, we review the possibilities provided by the SPARQL 1.1 standard [2] to sign query result sets. In particular, we will discuss how these methods can be used to optimise data pipelines avoiding expensive re-computation of results when data triples have not been updated.

## 2 SPARQL 1.1 hashing capabilities

The SPARQL standard provides a large set of built-in functions, from ones dedicated to strings to specific ones about dates. These can be used by query designers to refine their result set. In particular, the standard offers a set of five hash functions<sup>2</sup>: MD5, SHA1, SHA256, SHA384 & SHA512.

General signature of the hash functions:  <pre>simple literal hash_function (simple literal arg) simple literal hash_function (xsd:string arg)</pre>	Example using MD5:  <pre>H = md5("ab") = md5("ab"^^xsd:string) H = "187ef4436122d1cc2f40dc2b92f0eba0"</pre>
--	---

These functions accept either RDF literals or strings as argument and return the hash as a literal. In addition, a `xsd:string` or its corresponding literal should return the same result. In the ‘MD5’ example above, the hash value represents the result of a simple SPARQL query<sup>3</sup>.

Practically, these functions can be used to hash a complete RDF graph accessible through a SPARQL endpoint. Indeed, one can extract all the triples available with `select * where {?s ?p ?o}`, and then hash all of them, aggregated with a `group_concat` function. This could look like so:

```
SELECT (SHA1(GROUP_CONCAT(?tripleStr ; separator=' \n'))) AS ?nTriples
WHERE { ?s ?p ?o
  BIND(CONCAT(STR(?s), " ", STR(?p), " ", STR(?o)) AS ?tripleStr) }
```

In the previous query, the triples `?s ?p ?o` are cast by element to a string (`STR`), and then concatenated to form a “triple”. The recomposed list of triples is then grouped into one single string (`GROUP.CONCAT`) and finally hashed.

Although easy to understand, this “naïve” approach has some drawbacks. *First*, the result depends on the order of the triples returned by the triplestore: a workaround can be achieved adding *e.g.* `ORDER BY ?s ?p ?o datatype(?o) 1case(lang(?o))`. *Second*, this method has a scalability issue, as all the graph is loaded in-memory before the hash call. We therefore recommend this approach to sign small RDF graphs, *e.g.* ontologies or small result sets. *Finally*, this method

<sup>2</sup> <https://www.w3.org/TR/sparql11-query/#func-hash>

<sup>3</sup> `select * where{ values ?x {"ab" "ab"^^xsd:string} bind (md5(?x) as ?H)}`



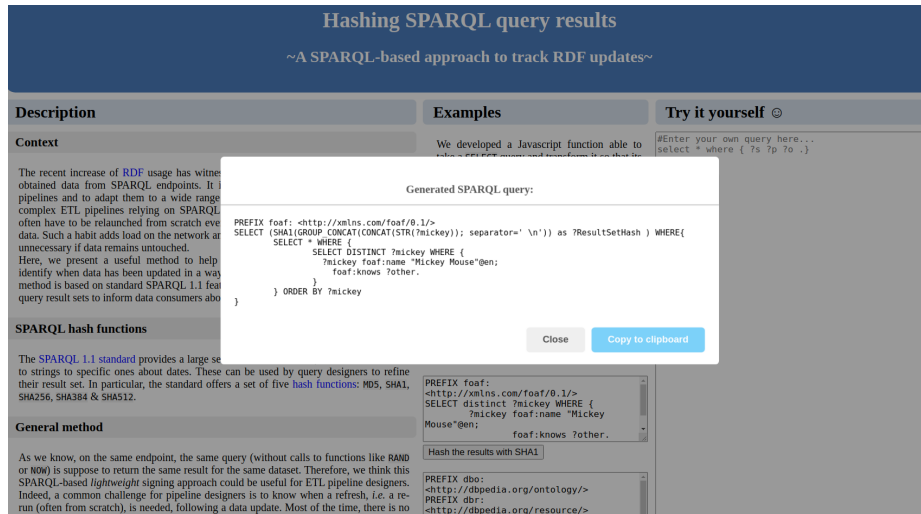


Fig. 1. Web Interface to obtain a  $Q_{hash}$  from a given  $Q$ .

The three steps to generate the query<sup>4</sup> that obtains the hash are easy to automate and allow users to know when to relaunch their pipelines. All this while making as much computations as possible on the endpoint side in charge of computing the hash.

*Performance:* For comparison, we reviewed three queries ( $Q_1$ ,  $Q_2$ ,  $Q_3$ ) on a YAGO4<sup>5</sup> KG loaded on Stardog, running on a 4 cores and 32GB memory VM. Their result sets respectively contain 2916, 50 000 and 100 000 results corresponding to 186KB, 5.3MB and 10.7MB and were on average computed in 331ms, 675ms and 1315ms. Their MD5 hashed versions all returned one hash string of 46 bytes and were computed in 364ms, 1353ms and 2492ms respectively. Thus, as expected, performing the hashes does not imply a large temporal overhead and greatly reduces the network traffic.

*Web-Interface:* To help SPARQL practitioners and pipeline designers with our method, we also developed a Web interface (see Figure 1 for a screenshot) and serve it [online](#) [↗](#). The page allows users to paste their SPARQL query ( $Q$ ) in order to obtain a new query ( $Q_{hash}$ ) which should be run to obtain the hash of the results of  $Q$ . The interface also provides a way to select the hash function among the ones from the standard. Technically the query generation is done through a JavaScript routine whose parser relies on SPARQL.js<sup>6</sup>.

<sup>4</sup> The queries can be tested directly on DBpedia: the [query  \$Q\$](#)  [↗](#) and its  [\$Q\_{hash}\$](#)  [↗](#). As of July 5<sup>th</sup> 2021,  $Q_{hash}$  returned `?H = "967d2c8c0a82038d8478d476fa41e14f"` and on September 6<sup>th</sup> it returned `"a41d8b97289ea1ef1af2a2ec54cff96c"`.

<sup>5</sup> YAGO4 has ~209 million triples: <https://yago-knowledge.org/downloads/yago-4>

<sup>6</sup> <https://github.com/RubenVerborgh/SPARQL.js>

## 4 Conclusions

This paper describes how to improve existing Semantic Web data pipelines with a SPARQL-based method that helps in identifying when query results have changed. It allows to re-run pipelines only when interesting parts of the original datasets have been updated. By using SPARQL to compute the signature of the query results, it avoids large result sets to be sent over the network while letting the triplestore optimise as much as possible all the computations. We hope this will inspire developers to use the hash functions provided by the standard, and serve our method at: <https://dgraux.github.io/SPARQL-hash/> where our query converter can be used directly by developers to generate queries computing the hash of their result sets.

**Acknowledgments.** This research was conducted with the financial support of the European Unions H2020 programme under the EDGE Marie Skłodowska-Curie grant agreement No. 713567 at the ADAPT SFI Research Centre at Trinity College Dublin, which is funded by Science Foundation Ireland and the European Regional Development Fund (ERDF) Grant #13/RC/2106\_P2.

## References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A nucleus for a web of open data. In: *The semantic web*. Springer (2007)
2. Harris, S., Seaborne, A., Prud'hommeaux, E.: Sparql 1.1 query language. W3C recommendation **21**(10), 778 (2013)
3. Hogan, A.: Canonical forms for isomorphic and equivalent RDF graphs: algorithms for leaning and labelling blank nodes. *ACM Transactions on the Web* (2017)
4. Junior, A.C., Orlandi, F., Graux, D., Hossari, M., O'Sullivan, D., Hartz, C., Dirschl, C.: Knowledge graph-based legal search over german court cases. In: *The Semantic Web: ESWC 2020 Satellite Events*. LNCS, vol. 12124, pp. 293–297. Springer (2020)
5. Kubitzka, D.O., Böckmann, M., Graux, D.: SemanGit: A linked dataset from git. In: *International Semantic Web Conference*. pp. 215–228. Springer (2019)
6. Manola, F., et al.: RDF primer. W3C recommendation (2004)
7. Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. *Communications of the ACM* **57**(10), 78–85 (2014)