# PathFinder Demo: Returning Paths in Graph Queries

Vicente Calisto[2], Benjamín Farías[1,2], Wim Martens[3], Carlos Rojas[2] and Domagoj Vrgoč[1,2]

[1]Pontificia Universidad Católica de Chile, Santiago, Chile
[2]Instituto Milenio Fundamentos de los Datos (IMFD), Santiago, Chile
[3]University of Bayreuth, Bayreuth, Germany

**Abstract**

In this demonstration we showcase PathFinder, a unified approach to returning paths in graph database queries. Returning paths is a central feature of regular path queries in the new GQL graph query standard. In the demo we showcase how PathFinder works by establishing two public endpoints, which the attendees will be able to access using their browser to try different queries. The first endpoint will host a property graph version of Wikidata to test GQL-style path queries, while the second one contains Wikidata in RDF format and illustrates how SPARQL can be extended to return paths.

**Keywords**

GQL, regular path queries, SPARQL, property paths, returning paths

## 1. Introduction and Outline

Graph databases have gained significant popularity [1, 2] and are supported by numerous SPARQL engines [3, 4, 5], as well as industry vendors like Oracle, Amazon, and Neo4j. A core feature of all these systems is support for *path queries*. In SPARQL these are supported through property paths, which are a variant of *regular path queries (RPQs)* commonly used in the database literature [6, 7, 8, 9, 10, 11]. While property paths and RPQs simply test the existence of a path between two nodes in a graph, the recent ISO standardization of the *Graph Query Language (GQL)* [12] brought forth a significant new challenge in querying paths, namely the combination of (i) regular path queries; and (ii) returning different types of paths.

To illustrate these features, consider the property graph in Figure 1 representing a travel network. We might be interested to find all the places we can reach from the city of Bayreuth by taking one or more train rides. In a GQL-like syntax such a query could be written as:

```
MATCH (?x:City WHERE ?x.name='Bayreuth')-[:train+]->(?y)
```

Here the pattern following the `MATCH` keyword is matched to the graph. In our case, `?x` is matched to the city of Bayreuth (node `n1` in Figure 1), and `?y` to any city reachable by a non-empty sequence of edges labeled `train`, as specified by expression `-[:train+]->`. Unlike Cypher, GQL allows arbitrary regular languages to describe how nodes in the graph are connected, just as SPARQL. Note that here we merely test for the existence of a path.

*Posters, Demos, and Industry Tracks at ISWC 2024, November 13–15, 2024, Baltimore, USA*
vicente.calisto@imfd.cl (V. Calisto); bffarias@uc.cl (B. Farías); wim.martens@uni-bayreuth.de (W. Martens); cirojas6@uc.cl (C. Rojas); vrdomagoj@uc.cl (D. Vrgoč)

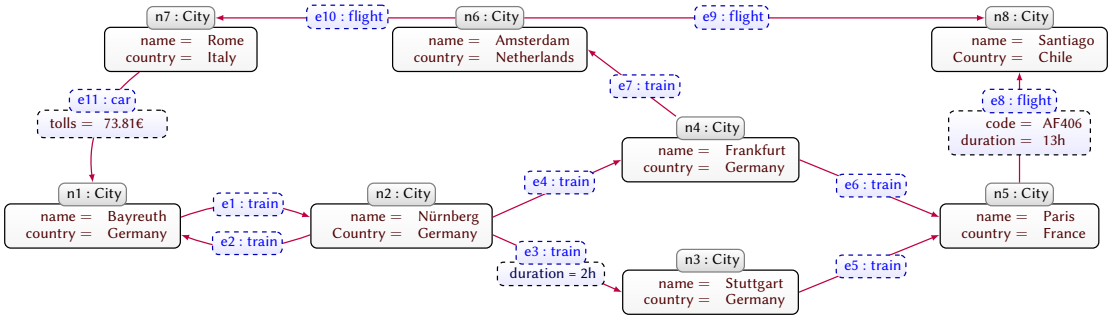


**Figure 1:** A sample graph database representing a (part of a) travel network.

The second key component of GQL is the ability to *return paths* in query answers. Examining the graph of Figure 1, we can immediately see that returning all the paths conforming to the query in our example is not feasible. Namely, the loop generated by the edges e1 and e2 allows us to generate an infinite number of paths. To prohibit infinite query answers, GQL uses *path modes* [12], which limit the type of paths we can return. Common modes are *simple* and *trail*, which forbid reusing nodes and edges on paths, respectively. Another option is *shortest*; for instance, if we wished to find a shortest way to reach Santiago from Bayreuth by using any number of train connections or flights, we would write:

```
MATCH (?x:City WHERE ?x.name='Bayreuth')
          -[?p ANY SHORTEST (:train | :flight)+]->
              (?y: City WHERE ?y.name='Santiago')
```

In this query the path itself is bound to the variable ?p and has non-deterministic semantics since there could be multiple shortest paths between the two nodes. For instance, in our example one answer is traced by the edges e1 $\rightarrow$ e3 $\rightarrow$ e5 $\rightarrow$ e8, another one by e1 $\rightarrow$ e4 $\rightarrow$ e7 $\rightarrow$ e9, etc. According to GQL, all these answers are valid, and there is no guarantee as to which one will be returned. Returning all of them can be done using the ALL SHORTEST mode.

Given the novelty of GQL [12] and the fact that queries that return paths are intrinsically difficult to evaluate [13], it is not surprising that their coverage is somewhat lacking in modern systems. For instance, most engines only support a few path modes out of the 15 that are possible in GQL [14]. To remedy these issues, we will showcase PathFinder, an add-on for graph database engines that allows processing path queries at scale and supports *all* GQL path modes. PathFinder is based on years of theoretical work on the subject, is implemented on top of the MillenniumDB graph database engine [15], and supports different storage mechanisms, showcasing its independence of the graph pipeline.

**Conference paper.** We remark that this demo accompanies our paper *PathFinder: Returning Paths in Graph Queries* [16] which will be presented in the ISWC research track. Compared to the conference version, here we put a strong focus on usability; namely, we develop a graphical interface for returning paths and host two endpoints where attendees can test path queries. We also propose an extension of SPARQL where paths can be returned in the query results. Finally, we remark that an additional author is added compared with the Research Track paper to work on providing user interfaces and SPARQL support for the demo.

## 2. The Demonstration

The demonstration will consist of the following use cases, all of which are supported through public query endpoints that will be available during the review process and later on.

**A public query endpoint.** The main highlight of our demonstration will be a public query endpoint where the attendees will be able to test GQL path queries using only their Web browser. In particular, our endpoint will be hosting a property graph version of Wikidata [17] which we used in our scalability tests. In particular, we use a curated version of the data set based on the truthy dump of Wikidata [18], which was used in WDBench [19]. The dataset is publicly available at [20]. During the demonstration we will also have a brief explanation of the underlying data and will prepare a series of instructive queries for the users to get acquainted with the language. The Wikidata endpoint is available for the review purposes at https://mdb.imfd.cl/path_finder/, with a series of illustrative queries included on the endpoint.

**PathFinder and SPARQL engines.** Since the PathFinder approach can also work as a part of a SPARQL engine, we first showcase its functionality in the context of evaluating property paths without returning the path themselves. This approach is already used in MillenniumDB's Wikidata SPARQL endpoint at https://wikidata.imfd.cl/, which leverages PathFinder to handle property path queries, and can be tried there. We remark that in this case the entire Wikidata dump is from 2023–07–16 and not Wikidata Truthy.

**Returning paths in SPARQL.** In addition to checking reachability via property paths, we would also like to illustrate how PathFinder can be used to return paths in SPARQL queries, through a syntactic extension of the language. One such proposal can be accessed directly through the PathFinder console as explained in our online repository [14]. For the demo presentation we also developed a Web interface for displaying paths that witness SPARQL property path query answers in a similar fashion as done for our property graph endpoint. To test this functionality, the attendees can access our (extended) SPARQL endpoint hosting the Wikidata Truthy dataset at https://mdb.imfd.cl/path_finder/. Figure 2 shows how this extension of SPARQL syntax and semantics works on our endpoint.

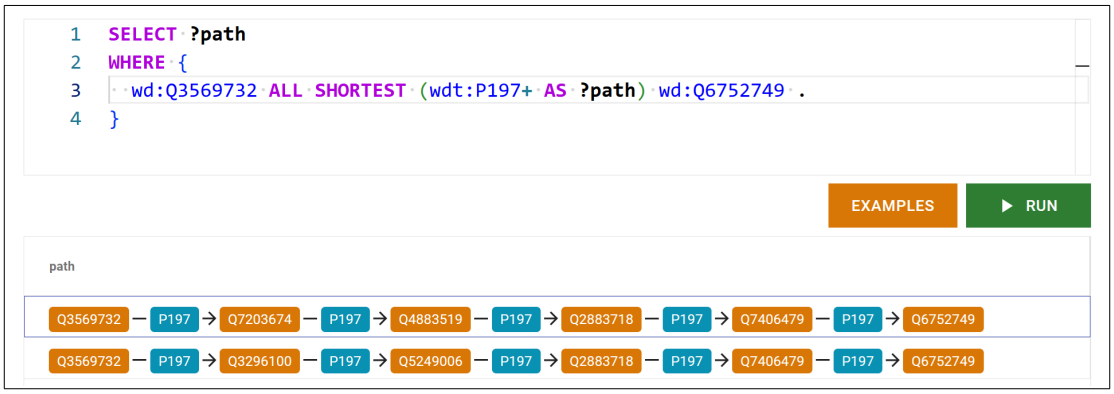


**Figure 2:** Returning paths in SPARQL query answers.

## Acknowledgments

Calisto, Farías, Rojas and Vrgoč were supported by ANID – Millennium Science Initiative Program – Code ICN17_002. Vrgoč was also supported by the ANID Fondecyt Regular project 1240346. Martens was supported by ANR project EQUUS ANR-19-CE48-0019; funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), project number 431183758.

## References

[1] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. L. Reutter, D. Vrgoč, Foundations of Modern Query Languages for Graph Databases, ACM Comput. Surv. 50 (2017). URL: https://doi.org/10.1145/3104031. doi:10.1145/3104031.

[2] S. Sakr, A. Bonifati, H. Voigt, A. Iosup, K. Ammar, R. Angles, W. G. Aref, M. Arenas, M. Besta, P. A. Boncz, K. Daudjee, E. D. Valle, S. Dumbrava, O. Hartig, B. Haslhofer, T. Hegeman, J. Hidders, K. Hose, A. Iamnitchi, V. Kalavri, H. Kapp, W. Martens, M. T. Özsu, E. Peukert, S. Plantikow, M. Ragab, M. Ripeanu, S. Salihoglu, C. Schulz, P. Selmer, J. F. Sequeda, J. Shinavier, G. Szárnyas, R. Tommasini, A. Tumeo, A. Uta, A. L. Varbanescu, H. Wu, N. Yakovets, D. Yan, E. Yoneki, The future is big graphs: a community view on graph processing systems, Commun. ACM 64 (2021) 62–71.

[3] J. Team, Jena TDB, 2021. URL: https://jena.apache.org/documentation/tdb/.

[4] B. Thompson, M. Personick, M. Cutcher, The Bigdata® RDF Graph Database, in: Linked Data Management, Chapman and Hall/CRC, 2014, pp. 193–237.

[5] O. Erling, Virtuoso, a Hybrid RDBMS/Graph Column Store, IEEE Data Eng. Bull. 35 (2012) 3–8. URL: http://sites.computer.org/debull/A12mar/vicol.pdf.

[6] I. F. Cruz, A. O. Mendelzon, P. T. Wood, A graphical query language supporting recursion, in: SIGMOD 1987, 1987, pp. 323–330.

[7] A. O. Mendelzon, P. T. Wood, Finding regular simple paths in graph databases, in: VLDB 1989, 1989, pp. 185–193.

[8] D. Calvanese, G. D. Giacomo, M. Lenzerini, M. Y. Vardi, Rewriting of regular expressions and regular path queries, J. Comput. Syst. Sci. 64 (2002) 443–465.

[9] M. Arenas, S. Conca, J. Pérez, Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard, in: Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012, ACM, 2012, pp. 629–638. URL: https://doi.org/10.1145/2187836.2187922. doi:10.1145/2187836.2187922.

[10] P. B. Baeza, Querying graph databases, in: PODS 2013, 2013, pp. 175–188. URL: https://doi.org/10.1145/2463664.2465216. doi:10.1145/2463664.2465216.

[11] K. Losemann, W. Martens, The complexity of regular expressions and property paths in SPARQL, ACM Trans. Database Syst. 38 (2013) 24. URL: https://doi.org/10.1145/2494529. doi:10.1145/2494529.

[12] A. D. et. al., Graph pattern matching in GQL and SQL/PGQ, in: SIGMOD '22, 2022. URL: https://doi.org/10.1145/3514221.3526057. doi:10.1145/3514221.3526057.

[13] W. Martens, M. Niewerth, T. Popp, C. Rojas, S. Vansummeren, D. Vrgoč, Representing paths in graph database pattern matching, Proc. VLDB Endow. 16 (2023) 1790–1803.
[14] B. Farías, W. Martens, C. Rojas, D. Vrgoč, PathFinder: A unified approach for handling paths in graph query languages, 2024. URL: https://github.com/AnonCSR/PathFinder.
[15] D. Vrgoč, C. Rojas, R. Angles, M. Arenas, D. Arroyuelo, C. Buil-Aranda, A. Hogan, G. Navarro, C. Riveros, J. Romero, Millenniumdb: An open-source graph database system, Data Intell. 5 (2023) 560–610. URL: https://doi.org/10.1162/dint_a_00229.
[16] B. Farias, W. Martens, C. Rojas, D. Vrgoč, Evaluating regular path queries in GQL and SQL/PGQ:, CoRR abs/2306.02194 (2023). URL: https://doi.org/10.48550/arXiv.2306.02194.
[17] D. Vrandecic, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Commun. ACM 57 (2014) 78–85.
[18] T. W. Foundation, Wikidata:database download, 2021. URL: https://www.wikidata.org/wiki/Wikidata:Database_download.
[19] R. Angles, C. B. Aranda, A. Hogan, C. Rojas, D. Vrgoč, Wdbench: A wikidata graph query benchmark, in: The Semantic Web - ISWC 2022, 2022. URL: https://doi.org/10.1007/978-3-031-19433-7_41. doi:10.1007/978-3-031-19433-7_41.
[20] R. Angles, C. B. Aranda, A. Hogan, C. Rojas, D. Vrgoč, WDBench Dataset Download, 2022. doi:10.6084/m9.figshare.19599589.