HAL
open science

Exploring Labeled Graphs with Recursive Path Patterns

Amela Fejza, Pierre Genevès, Nabil Layaïda

▶ **To cite this version:**

Amela Fejza, Pierre Genevès, Nabil Layaïda. Exploring Labeled Graphs with Recursive Path Patterns. 2022. hal-03517826v2

HAL Id: hal-03517826

https://inria.hal.science/hal-03517826v2

Preprint submitted on 24 Oct 2022 (v2), last revised 9 Nov 2023 (v4)

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.

Exploring Labeled Graphs with Recursive Path Patterns

Amela Fejza*, Pierre Genevès†, Nabil Layaïda‡
Tyrex team, Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG
38000 Grenoble, France
Email: *amela.fejza@inria.fr, †pierre.geneves@inria.fr, ‡nabil.layaida@inria.fr,

***Abstract*—We demonstrate a system for recursive query answering over labeled graphs. The system is based on a complete implementation of the recent μ-recursive relational algebra, that has been extended with parsers and compilers adapted for queries over property graphs. As a result, one can formulate, optimize and answer queries that navigate recursively along paths in property graphs. We demonstrate the system on three real datasets, including the exploration of chains of drug interactions.**

I. Introduction

Labeled graph data structures become increasingly popular in various applications such as social networks, knowledge extraction, transportation networks, biological and clinical data with interactions such as proteins or drugs, etc. [11].

Two main labeled graph data models coexist: *knowledge graphs* in which edges are labeled with a single predicate (RDF graphs), and *property graphs* in which richer annotations are possible on both edges and nodes since they can also carry a list of key-value pairs. Technologies for knowledge graphs are mature and well standardized (with the SPARQL standard query language for instance). Query languages for property graphs are still under active development, with various proposals (e.g. Cypher [6], PGQL [12], G-Core [2]) from which a standard may emerge. All the most recent developments have shown a significant interest in equipping query languages with recursive query constructs such as regular path queries (RPQs) and their combinations. They enable queries to exploit the linked nature of graph data by allowing for navigations along deep paths in the graph. Current query language proposals for property graphs come with more and more expressive forms of recursive path patterns. However, the optimization of recursive queries is notoriously known as a difficult problem.

We demonstrate a system for recursive query answering over property graphs. The system includes a complete implementation of the recursive relational algebra and concepts introduced in [8], [9]. We extended the system with parsers and compilers so that one can formulate, optimize and answer queries that navigate recursively in property graphs. The novelty of the system resides in its ability to optimize and efficiently answer recursive path patterns in property graph queries.

The purpose of this demonstration is to show in an interactive manner that (i) the system is significantly more efficient than previous systems when answering queries with recursive path patterns on large real datasets; and (ii) that the system also supports expressive path patterns not supported by other systems, yet useful in formulating meaningul queries on real datasets.

II. Preliminary Background

The μ-relational algebra (μ-RA) [8] is an extension of Codd's relational algebra with a fixpoint operator so as to capture and to optimize recursive queries.

A term φ in μ-RA can be a *relation* X denoting a classical relational table, a *constant* $|c \rightarrow v|$ that assigns a value to a column name, a *filter* $\sigma_f(\varphi)$ that keeps only the tuples in φ that satisfy a condition f, an *antiprojection* $\widetilde{\pi}_a(\varphi)$ which removes the column named a from the term φ, a *renaming* operator $\rho_a^b(\varphi)$ that renames column a into b in the term φ, a binary operator such as *natural join* ($\varphi_1 \bowtie \varphi_2$), *antijoin*, and *union* ($\varphi_1 \cup \varphi_2$). Besides those rather classical operators, one of the main novelty of μ-RA is that a term can also be a *fixpoint operator* of the form $\mu X.\ \varphi$. This notation binds a relation X to the term φ in which X appears, hence it is an explicit way to write a recursive term in an algebraic manner.

For example, the term $\mu X.R \cup (D \bowtie X)$ denotes a relation obtained by repeatedly joining the initial relation R with a relation D until no new tuples are retrieved. The transitive closure relation R^+ is the particular case where $D = R$. The general fixpoint notation makes it possible to express not only transitive closures but also a variety of more expressive forms of recursion.

The other main novelty of μ-RA is that it comes with a set of algebraic rewrite rules specifically designed to transform fixpoint terms into semantically equivalent (yet more efficient) variants, generalizing the initial idea of Codd to recursive queries. In particular, transformations include algebraic rewritings where filters, antiprojections and joins are "pushed through" fixpoint terms (modulo some conditions are satisfied), yielding other fixpoint terms whose evaluation can be way more efficient than the initial ones. Some fixpoint terms can also be merged into a single fixpoint term, replacing two recursions by a single one. Such transformation rules, together with the necessary mechanisms to check the conditions under which they are valid, are detailed in [8]. This basically opens the way to much more efficient query evaluation plans for recursive queries (that are out of reach of earlier approaches, including those based on Datalog).

In this demonstration paper, we present a system which is an extension of a full implementation of μ-RA, including transformation rules, query evaluation plan space search, backend for evaluation, as well as new parsers and compilers to enable recursive query answering over property graphs.

III. System Architecture

The overall system architecture is illustrated in Figure 1. It is composed of several components:

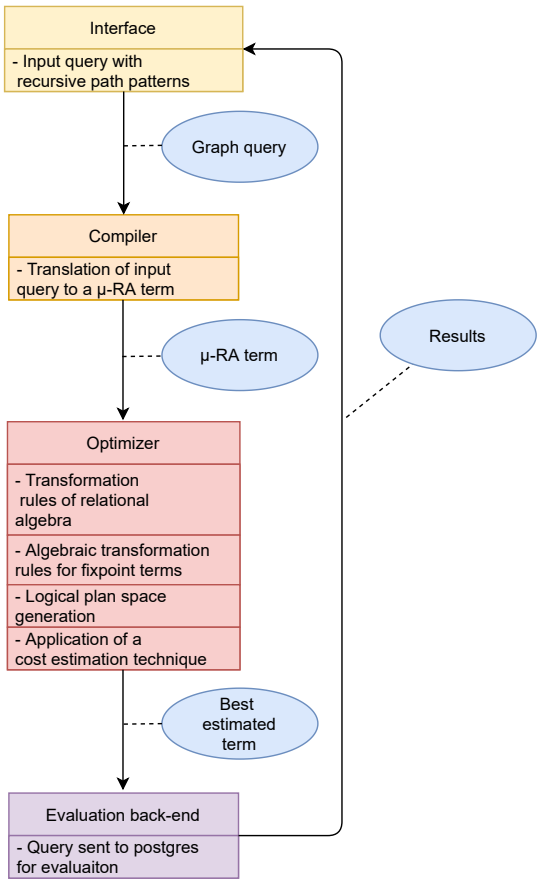

Fig. 1: Architecture of the system.

a) An interface: where the user provides a graph query with recursive path patterns. The interface also displays query answers and information and statistics on the evaluation process (query evaluation time, number of results, selected evaluation plan etc.)

b) A compiler: that parses the input query and translates it into a μ-RA term that operates on the RA representation of the property graph (see Section IV).

c) An optimizer: that applies all the algebraic rewrite rules of classical relational algebra together with the rewrite rules concerning fixpoint terms presented in [8], in a fully compositional way. By composition, all these rules enable new query evaluation plans (in particular merged fixpoints, and their further transformations) that were not possible with earlier approaches. Overall, this results in a richer query evaluation plan space, as illustrated in Figure 2. In order to pick a given plan from the plan space, a cost estimation technique inspired from [9] is used. It picks an estimated most efficient term, based on cost estimations for the operators and data statistics of the given dataset instance. This term is then sent to a backend for evaluation.

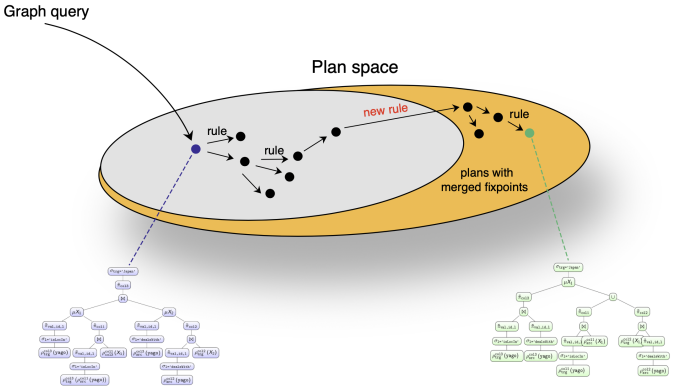

Fig. 2: Plan space exploration with fixpoint rewritings.

d) A backend evaluator: this is a relational database managment system in charge of evaluating the best estimated plan. For this demonstration, the backend used is PostgreSQL. After evaluation, query answers and information about the evaluation are presented in the interface.

IV. Application for Property Graphs

In the property graph data model, nodes and edges are labeled and they can also carry a list of properties with values. For example, Figure 3 presents a property graph of a social network, connecting people, forums and tags, with information about them.

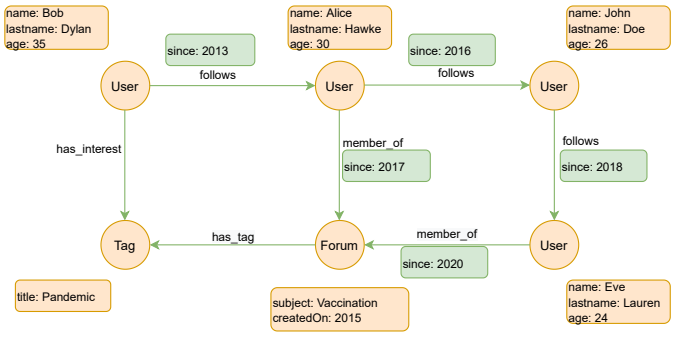

Fig. 3: Social network schema.

In the relational algebra data model, a property graph is represented as a set of relations, with one relation per edge type and one relation per node type. Each relation encodes the specific properties of each node (or edge respectively) using one column per property. Figure 4 illustrates this representation for the sample property graph of Figure 3.

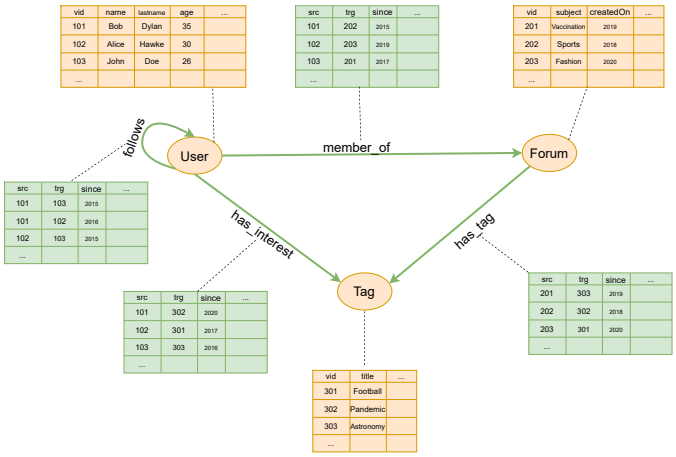

Fig. 4: Representation of social network property graph.

This representation comes with several assumptions to ensure consistency when applying μ-RA terms. In particular, nodes must be uniquely identified (the vid column) and those identifiers must be disjoint between node types. For each edge type, the corresponding relation contains at least the source and target nodes (which are foreign keys to node's vids). A knowledge graph is simply a particular case of a property graph with no particular property beyond source and target vids in edge relations, for example.

The system optimizes and evaluates recursive path queries in property graphs. The following query examples illustrate simple recursive path patterns for extracting information from the sample social network propery graph of Figure 4:

The query $\mathcal{Q}_1$ below finds all the users followed directly or indirectly by a young user, who have interest in a certain tag used in a Forum about vaccination:

$$\begin{aligned} ?y \leftarrow \quad & ?x : \text{User[age=20]} \;\; \text{follows+} \;\; ?y : \text{User}, \\ & ?y : \text{User} \;\; \text{has_interest/-has_tag} \;\; ?z : \text{Forum[subj=vax]} \end{aligned}$$

This query uses the pattern "follows+" which is the transitive closure of the relation "follows". Another example is query $\mathcal{Q}_2$ below that finds the totality of users followed by Bob (directly or indirectly) since 2013 and that are also members of a given Forum y since 2020:

$$\begin{aligned} ?x, ?y \leftarrow \quad & ?b : \text{User[name=Bob] follows[since=2013]+} \;\; ?x : \text{User}, \\ & ?x : \text{User} \;\; \text{member_of[since = 2020]} \;\; ?y : \text{Forum} \end{aligned}$$

This query navigates using the transitive closure of the filtered relation "follows[since=2013]". The left hand side of the query extracts all pairs $(?x, ?y)$ that can be retrieved in this manner.

This query $\mathcal{Q}_2$ can be translated into μ-RA. For instance the main body of $\mathcal{Q}_2$ could be translated as t:

$$\begin{aligned} & \rho^{\text{src}}_{\text{vid}}(\sigma_{\text{name=Bob}}(\text{User})) \quad \bowtie \\ & \sigma_{\text{since=2013}}(\mu X.\text{follows} \;\cup\; \widetilde{\pi}_m(\rho^{\text{m}}_{\text{trg}}(\text{follows}) \quad \bowtie \rho^{\text{m}}_{\text{src}}(\text{X}))) \\ & \bowtie \sigma_{\text{since=2020}}(\text{member_of}) \quad \bowtie \quad \rho^{\text{trg}}_{\text{vid}}(\text{Forum}) \end{aligned}$$

However, this translation happens to be particularly inefficient as it computes the whole transitive closure of the relation follows and filters the results afterwards. Automated algebraic transformations generate semantically equivalent and more efficient terms, such as the following term t':

$$\begin{aligned} & \mu X.\rho^{\text{src}}_{\text{vid}}(\sigma_{\text{name=Bob}}(\text{User})) \bowtie \sigma_{\text{since=2013}}(\text{follows}) \;\cup \\ & \quad \widetilde{\pi}_m(\rho^{\text{m}}_{\text{src}}(\sigma_{\text{since=2013}}(\text{follows})) \bowtie \rho^{\text{m}}_{\text{trg}}(\text{X}))) \\ & \bowtie \sigma_{\text{since=2020}}(\text{member_of}) \quad \bowtie \quad \rho^{\text{trg}}_{\text{vid}}(\text{Forum}) \end{aligned}$$

Because the fixpoint used in t navigates from left to right in sequences of edges labeled with "follows", it is possible to turn it into the fixpoint used in t' that starts only from the node Bob, and then navigates recursively with the filtered relation (without losing any result).

Another way to answer this recursive query is to navigate from right to left. In that case, the previous join and filters about Bob can no longer be pushed through the fixpoint (otherwise this may prevent the whole set of solutions to be reached). However, the subquery part concerning Forum can now be pushed inside the fixpoint. Hence another semantically equivalent term (also computed through the application of transformation rules) is t'':

$$\begin{aligned} & \rho^{\text{src}}_{\text{vid}}(\sigma_{\text{name=Bob}}(\text{User})) \quad \bowtie \\ & \mu X.\sigma_{\text{since=2013}}(\text{follows}) \bowtie \sigma_{\text{since=2020}}(\text{member_of}) \quad \bowtie \\ & \quad \rho^{\text{trg}}_{\text{vid}}(\text{Forum}) \quad \cup \; \widetilde{\pi}_m(\rho^{\text{m}}_{\text{trg}}(\sigma_{\text{since=2013}}(\text{follows})) \quad \bowtie \rho^{\text{m}}_{\text{src}}(\text{X}))) \end{aligned}$$

Depending on the nature of the dataset instance one term (for instance t' or t'') may be more efficient than the other. The cost estimation function depicted in Figure 1 helps in selecting a best estimated term.

V. Demonstration Plan

The aim of the demonstration is twofold: first, to show the performance gains in query evaluation time brought by the system in comparison to previous systems, and second, to point out the benefits of using expressive recursive path patterns in queries.

For these purposes, we provide three scenarios with three real datasets. For each scenario, the demonstration plan starts by running a few suggested queries (that we describe thereafter). Then, in a second stage the attendees will be able to interact with the system using their own custom queries.

a) Scenario – Yago: Yago [4] is a huge graph containing more than 62 million edges between more than 42 millions of nodes. We will run several third-party queries taken from the litterature, and compare the obtained query evaluation times with other systems (such as Neo4j and LB). This will be a live demonstration, where the same query answering problem instances will be triggered with the different systems. Suggested initial queries to start with are presented in Figure 5 which contains queries $\mathcal{Q}_{1-6}$ taken from [1] and that model military intelligence workloads, query $\mathcal{Q}_7$ taken from [7] and query $\mathcal{Q}_8$ taken from [8]. No histogram bar in Figure 5 means a timeout for the corresponding system. This will be the opportunity to see the efficiency of the recursive plans generated by the system in practice.

```
?x      ←  ?x isMarriedTo/livesIn/IsL+/dw+ Argentina     Q1
?x      ←  ?x hasChild/livesIn/IsL+/dw+ Japan            Q2
?x      ←  ?x influences/livesIn/IsL+/dw+ Sweden         Q3
?x      ←  ?x hasSuccessor/livesIn/IsL+/dw+ India        Q4
?x      ←  ?x hasPredecessor/livesIn/IsL+/dw+ Germany    Q5
?x      ←  ?x haa/livesIn/IsL+/dw+ Netherlands           Q6
?area   ←  wce -typ/(IsL+/dw|dw) ?area                   Q7
?a      ←  ?a IsL+/IsL Japan                             Q8
```

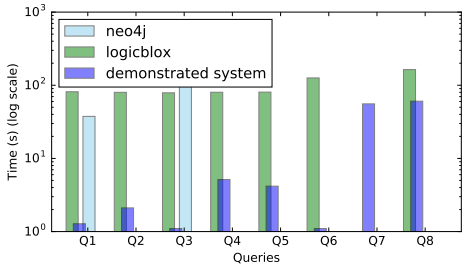

Fig. 5: Sample third-party queries over the Yago dataset.

b) Scenario – Bahamas Leaks: The Bahamas Leaks dataset [10] comes from the International Consortium of Investigative Journalists (ICIJ) and is based on the analysis of the island nation's corporate registry. It is a property graph that provides names of directors and owners of more than 175,000 Bahamian companies, trusts and foundations registered between 1990 and early 2016, together with their connections. The dataset includes a relation "same_name_as" between officers that has been infered using reverse engineering techniques by the ICIJ. In this scenario, we explore the network of officers and entities. We start by finding all the entities associated with officers sharing the same name:

$$?o, ?y \leftarrow ?x : \text{officer[name='x']} \text{ same_name_as+ } ?o : \text{officer},$$
$$?o : \text{ officer officer_of } ?y : \text{entity}$$

This gathers pairs of officers and entities they belong to. Sometimes, between an officer and an entity there is a intermediary. We can also retrieve this information:

$$?o, ?y, ?i \leftarrow ?x : \text{officer[name='x']} \text{ same_name_as+ } ?o : \text{officer},$$
$$?o : \text{ officer officer_of } ?y : \text{entity},$$
$$?i : \text{ intermediary intermediary_of } ?y : \text{entity}$$

The Bahamas Leaks dataset provides information about addresses registered. We can see from which countries these intermediaries come from:

$$?o, ?y, ?i, ?a \leftarrow ?x : \text{officer[name='x']} \text{ same_name_as+ } ?o : \text{officer},$$
$$?o : \text{ officer officer_of } ?y : \text{entity},$$
$$?i : \text{ intermediary intermediary_of } ?y : \text{entity}$$
$$?i : \text{intermediary reg_address ?a: address}$$

c) Drug interactions: This dataset is a network of drug-drug interactions that comes from a Brazilian study [3]. Nodes contain information about drugs and edges contain information about the the nature of their interaction (like the severity level etc.).

The following query navigates in the network of drug interactions. It retrieves the pairs of drugs that interact with each other (possibly indirectly) such that the severity level of the interaction is significant (either Moderate or Major) and concerns females:

$$?x, ?y \leftarrow ?x : \text{drug[hormone=True]}$$
$$(\text{ (interacts[severity = Moderate] } | \text{ interacts[severity = Major])}$$
$$/ \text{ interacts[gender = Female])+} \quad ?y : \text{drug}$$

Notice that this query contains a path pattern of the form $((a|b)/c)^+$ which is naturally captured and optimized by the system (whereas it is currently not expressible in the Cypher 9 language [5], for instance). During the demonstration the attendees will be able to explore other chains of drug interactions, using recursive path patterns, exploring for example chains of interactions between Central nervous system agents and Cardiovascular agents.

References

[1] Z. Abul-Basher, N. Yakovets, P. Godfrey, S. Ghajar-Khosravi, and M. Chignell. Tasweet : optimizing disjunctive regular path queries in graph databases. In B. Mitschang, V. Markl, S. Bress, P. Andritsos, K.-U. Sattler, and S. Orlando, editors, *20th International Conference on Extending Database Technology, 21-24 march 2017, Venice, Italy*, pages 470–473, Mar. 2017. EDBT/ICDT 2017 Joint Conference 20th International Conference on Extending Database Technology, EDBT 2017 ; Conference date: 21-03-2017 Through 24-03-2017.

[2] R. Angles, M. Arenas, P. Barcelo, P. Boncz, G. Fletcher, C. Gutierrez, T. Lindaaker, M. Paradies, S. Plantikow, J. Sequeda, O. van Rest, and H. Voigt. G-core: A core for future graph query languages. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 14211432, New York, NY, USA, 2018. Association for Computing Machinery.

[3] R. Brattig Correia, L. P. de Arajo Kohler, M. M. Mattos, and L. M. Rocha. City-wide electronic health records reveal gender and age biases in administration of known drugdrug interactions. *npj Digital Medicine*, 2(1).

[4] M. P. I. for Informatics and T. P. University. Yago: A high-quality knowledge base. https://www.mpi-inf.mpg.de/yago-naga/yago/, july 2019.

[5] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, M. Schuster, P. Selmer, and A. Taylor. Formal semantics of the language cypher. 02 2018.

[6] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 14331445, New York, NY, USA, 2018. Association for Computing Machinery.

[7] A. Gubichev, S. J. Bedathur, and S. Seufert. Sparqling kleene: Fast property paths in rdf-3x. In *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, New York, NY, USA, 2013. Association for Computing Machinery.

[8] L. Jachiet, P. Genevs, N. Gesbert, and N. Layada. On the optimization of recursive relational queries: Application to graph queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 681–697, 2020.

[9] M. Lawal, P. Genevès, and N. Layaïda. A Cost Estimation Technique for Recursive Relational Algebra. In *CIKM 2020 - 29th ACM International Conference on Information and Knowledge Management*, pages 1–4, Virtual Event, France, Oct. 2020.

[10] I. C. of Investigative Journalists. Bahamas leaks. https://www.kaggle.com/datasets/zusmani/paradisepanamapapers, 2017.

[11] 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, and E. Yoneki. The future is big graphs: a community view on graph processing systems. *Commun. ACM*, 64(9):62–71, 2021.

[12] O. van Rest, S. Hong, J. Kim, X. Meng, and H. Chafi. Pgql: a property graph query language. In *GRADES '16*, 2016.