Towards a Standard for Triggers in Property Graphs

Luigi Bellomarini[1,†], Anna Bernasconi[2], Stefano Ceri[2], Alessia Gagliardi[2], Davide Magnanimi[1,2,†] and Davide Martinenghi[2,*]

[1]*Banca d'Italia, Rome, Italy*
[2]*Politecnico di Milano, Milan, Italy*

Abstract

Graph databases are emerging as the leading data management technology for storing large knowledge graphs; significant efforts are ongoing to produce new standards (such as the Graph Query Language, GQL) and enrich them with properties, types, schemas, and keys. In this article, we present PG-Triggers, a proposal for adding triggers to Property Graphs, along the direction marked by the SQL3 Standard.

We define the syntax and semantics of PG-Triggers and briefly discuss how they can be implemented on top of the most popular graph databases.

The main objective of this article is to discuss an active database standard for graph databases as a first-class citizen at a time when reactive graph management is in its infancy, so as to minimize the conversion efforts towards a full-fledged standard proposal.

Keywords

property graphs, standards for graph databases, trigger standardization

1. Introduction

Graph databases are becoming increasingly important as frameworks for representing and understanding the intricate connections that exist in the real world [1]. Thanks to their expressive query languages, rich customer support, and strong performance, they are steadily more used to store large knowledge graphs, in a variety of domains that include, e.g., mobility, social, and biological networks.

As customary in data management evolution, standards for graph databases are emerging, most importantly the Graph Query Language (GQL) [2], whose roadmap is followed with interest by the major companies in the field. In addition, the research community has proposed various formalizations so as to enrich the semantics of graph databases, first by shaping them in the form of *Property Graphs* [3], and then by defining the notions of *PG-Keys* [4] and *PG-Schema* [5].

In this paper, we follow up along this trend and propose *PG-Triggers*. Triggers exist since the birth of relational databases [6], have been studied in [7], and formalized in the ISO-ANSI SQL3 Standard [8]. So far, they have not been formalized by the graph database research community,

SEBD 2024: 32nd Symposium on Advanced Database Systems, June 23-26, 2024, Villasimius, Sardinia, Italy
*Corresponding author.
†The views and opinions expressed in this paper are those of the authors and do not necessarily reflect the official policy or position of Banca d'Italia.
luigi.bellomarini@bancaditalia.it (L. Bellomarini); anna.bernasconi@polimi.it (A. Bernasconi); stefano.ceri@polimi.it (S. Ceri); alessia1.gagliardi@mail.polimi.it (A. Gagliardi); davide.magnanimi@bancaditalia.it (D. Magnanimi); davide.martinenghi@polimi.it (D. Martinenghi)
0000-0001-6863-0162 (L. Bellomarini); 0000-0001-8016-5750 (A. Bernasconi); 0000-0003-0671-2415 (S. Ceri); 0000-0002-6560-8047 (D. Magnanimi); 0000-0002-2726-7683 (D. Martinenghi)

although they can be informally supported by most graph database systems, even if in diversified ways (see Section 2). Hence, our proposal for PG-Triggers has the potential to influence future standard development as well as to suggest new directions to the evolution of graph databases.

In this paper we summarize some of the findings discussed in [9]. While here we focus on the syntactic and semantics aspects of the proposed standard, in [9] we also demonstrate our idea in practice on Neo4j. We show that Neo4j already supports all the required components for implementing our PG-Trigger concepts; however, it does so within a community-supported library, called *Awesome Procedures on Cypher* (APOC) [10], and not as part of Cypher [11] – the declarative graph query language adopted by Neo4j. Our approach easily extends to Memgraph. Our translation schemes show that current Neo4j or Memgraph solutions could rather easily evolve into unified and solid abstractions, along the PG-Trigger proposal, so as to favor a wider use of empowered triggers.

Contributions. Our main contribution is a proposal for adding reactive behavior in the form of triggers to graph databases, which follows from our comparative review of graph database technologies, presented in Section 3. This proposal aims to be natural and useful:

- Natural, as it suitably adapts the recommendations of the SQL3 standard to a graph setting, thereby adhering to the principle of least surprise, for the great benefit of people already acquainted with the well-known corresponding relational notions.
- Useful, as knowledge management through reactive behavior proves to be very effective in numerous *knowledge-intensive* (instead of just data-intensive) scenarios.

Adapting the notion of triggers to Property Graphs is, however, far from trivial, as, on one hand, it requires dealing with non-relational concepts such as nodes, relationships, labels, and properties, and, on the other hand, it lacks a full correspondence with the notion of table, which, in the relational case, is the source of events that can be monitored by triggers. To this end, our proposal identifies the notion of label as the most suitable and natural choice for defining a set of target elements in the graph, much in the same way in which tables do in SQL triggers.

It is worth mentioning that, although PG-Triggers can be implemented on top of Neo4j through the APOC trigger library (which provides evidence of the feasibility of our proposal), this step also requires extending APOC triggers with a few important, missing ingredients, among which the support for a correct cascading of triggers (occurring when a trigger's action causes the activation of other triggers), for event-specific triggering action times, and for instance-level vs set-level trigger granularities. We support these features in our PG-Triggers proposal, whose syntax and semantics are streamlined as much as possible, so as to combine ease of use with expressivity, in the hopes that our proposal can drive forthcoming standardization choices in Property Graphs.

2. Related Work

The Property Graph data model applies to a directed graph where nodes and edges are labeled, and each can have associated ⟨property, value⟩ pairs. The Property Graph data model has gained significant popularity and adoption in various graph database systems, including Neo4j [12], Memgraph [13], JanusGraph [14], Amazon Neptune [15], Nebula Graph [16], TigerGraph [17],

and more. This large participation and attention on Property Graphs led to the idea of creating a standalone Property Graph query language to complement SQL, which was raised by ISO SC32/WG3 members in early 2017 and is echoed in the GQL manifesto of May 2018 [18]. Closely related efforts include: the Linked Data Benchmark Council (LDBC) Groups [19], the Graph Query Language Core (G-CORE) [20], PG-Keys [4], and PG-Schema [5].

Only a few reactive extensions are discussed for research prototypes using graph databases. Among them, GraphFlow [21] and TurboFlux [22].

3. Comparative Review of Graph Database Technology

We analyze some of the commercial graph database systems, highlighting how they support reactive computations. In particular, we first analyze products focused on supporting graph (or RDF) data and then we consider systems mixing graph data with other kinds of data, including relational and document/key-value data.

3.1. Graph Databases

3.1.1. Graph Databases with trigger support

- **Neo4j.** Neo4j is an open-source NoSQL native graph database; it is the most widely adopted graph database [23]. It has introduced Cypher, a declarative language for querying and manipulating graph data, the *de facto* standard in graph databases. Neo4j does not support triggers natively, however, triggers are included in APOC (Awesome Procedures on Cypher), a popular extension library for Neo4j.
- **Memgraph.** Memgraph is another open-source graph database compatible with Neo4j, built for real-time streaming. It offers an implementation of triggers that supports the execution of any openCypher [24] query.

3.1.2. Graph Databases with event listeners

- **JanusGraph.** JanusGraph is an open-source, distributed graph database system built on the graph computing framework Apache TinkerPop; it is efficient in handling very large graphs with billions of vertices and edges. In JanusGraph, triggers can be produced through the "JanusGraph Bus", a collection of configurable logs to which JanusGraph writes changes to the graph.
- **Dgraph.** Dgraph is an open-source, horizontally scalable, distributed graph database. It provides a flexible query language for querying and manipulating data, called DQL. Dgraph provides the Dgraph Lambda framework [25], which can be used to react to events through Typescript or Javascript functions.
- **Amazon Neptune.** Amazon Neptune is a fully managed graph database service provided by Amazon Web Services. Amazon Neptune supports both the Apache TinkerPop Gremlin graph traversal language and the openCypher query language for the Property Graph data model. For the Resource Description Framework (RDF) data model, Neptune supports the standard open language SPARQL query language [26]. Neptune uses Amazon Simple Notification Service (Amazon SNS) to provide notifications when a Neptune event occurs.

	Tr-G	Tr-R	Ev-L
Neo4j [12], **Memgraph** [13]	✓		-
JanusGraph [14], **Dgraph**[27], **Amazon Neptune** [15], **Stardog** [28]	-		✓(JSBus, Lambda, SNS, Java)
Nebula Graph [16], **TigerGraph** [17], **GraphDB** [29]	-		-
Oracle Graph Database [30], **Virtuoso** [31], **AgensGraph** [32]	-	✓	-
Microsoft Azure Cosmos DB [33], **OrientDB** [34], **ArangoDB** [35]	-		✓(JS, Hooks, AQL)

Table 1
Comparison of graph databases, focused on their management of reactive aspects. We consider Trigger Support in Graph Data (Tr-G) and in Relational Data (Tr-R), and availability of Event Listener (Ev-L), which can be exploited for building reactive behaviors as managed outside of the graph database.

- **Stardog.** Stardog is a commercial graph DBMS with a connected Enterprise Knowledge Graph platform and virtualization capability. It supports GraphQL for the graph store and SPARQL for the RDF store. It uses Java event handlers to capture changes in the data.

3.1.3. Other Graph Databases

- **Nebula Graph.** NebulaGraph is an open-source distributed, linear scalable database supporting efficient graph patterns. It supports Cypher and implements its own nGQL language; it does not support reactive aspects.
- **TigerGraph.** TigerGraph is a commercial parallel graph computing platform; it provides a graph query language called GSQL, which allows user-defined functions and procedures. It does not include reactive aspects.
- **GraphDB.** GraphDB is a commercial graph database and RDF store, with efficient reasoning support. Queries are accepted in GraphQL and SPARQL; triggers are not supported.

3.2. Mixed Graph-Relational Systems

Mixed graph-relational systems are built by integrating two engines: a graph database and a relational database. The graph system supports a graph (Cypher-like) query language, whereas the relational system supports SQL. Queries can be built by assembling Cypher and SQL statements, where the former operates on graph data and the latter operates on tables. Relational engines support triggers, compliant with the SQL3 standard, but these do not operate on graph data.

- **Oracle Graph Database and Graph Analytics.** The Oracle Graph Database supports the Property Graph data model, enabling graph analytics capabilities. Oracle Graph Database integrates with Oracle's broader ecosystem, including its SQL-based data management and triggers.
- **Virtuoso.** Virtuoso is a multi-model DBMS developed by OpenLink (available both in open-source and commercial editions), providing SQL, XML, and RDF data management in a single multithreaded process. The trigger support is provided in the relational DBMS.
- **AgensGraph.** Agens Graph is an open-source tool whose graph system supports the Property Graph as well as the relational data model; the latter is built upon PostgreSQL. Taking advantage of relational technology, AgensGraph supports triggers.

3.3. Mixed Graph-Document Databases

Mixed graph-document systems are built by integrating graph data with document bases.

- **Microsoft Azure Cosmos DB.** Cosmos DB is a globally distributed, horizontally scalable, multi-model database service, queried through graph database APIs (Gremlin) and document database APIs (Mongo DB or Cassandra). A JavaScript-integrated query API can be used to write triggers, which are classified into pre-triggers (executed before modifying a database item) and post-triggers (after modifying a database item). These must be specified for each database operation where their execution is expected.
- **OrientDB.** OrientDB is an open-source, multi-model database management system that combines characteristics of document databases and of graph databases. It employs a SQL-like query language called OrientSQL and the native graph query language Gremlin. In this context, triggers (renamed as Hooks) enable the triggering of actions in response to document creation, modification, or deletion.
- **ArangoDB.** ArangoDB is a multi-model, open-source database system that combines the features of document, key-value, and graph databases into a single platform. It provides a native graph querying language called AQL (ArangoDB Query Language) for efficient graph traversals and graph-based analytics. ArangoDB supports different events that can be monitored through a listener (AbstractArangoEventListener).

3.4. Comparison and Discussion

Table 1 offers a synoptic view of how graph databases, with their main extensions, support reactive computations, either directly through triggers or indirectly through event listeners. Note, however, that trigger support in mixed relational systems operates just upon tables, i.e., the relational component. This comparison shows that, although forms of reactive processing exist in many commercial graph databases, they are not yet well developed.

In summary, as of today, native triggers on graph data are available just in Neo4j and Memgraph; moreover, Neo4j triggers are still supported within community-defined APOC libraries and are not part of the standard language. Many other graph databases and hybrid systems support ingredients for building reactive systems (e.g., Hooks of OrientDB) but they are far away from supporting full-fledge trigger systems. This is the ideal time for setting the requirements for a trigger standard, so as to avoid misaligned deployment of similar but not identical features; we will show that signs of such misalignment exist already by focusing on Neo4j and Memgraph, the two graph databases with stronger trigger support, as well as market leaders within graph database according to [23].

Note that a trigger standard is quite parsimonious, as triggers are compositions of event-condition-action rules that base all their syntax and semantics on a limited number of ingredients, which define: when they should be activated after a data creation, modification, or deletion; when their condition should be considered; and when their action should be executed if the condition is true. The essence of a trigger system should be as much as possible agnostic to the detailed aspects of the underlying query language - be it SQL, Cypher, or GQL. Fixing these aspects ahead of language standardization may lead to convergence – at this time being

inexpensive for graph database vendors, whereas -at the time of the SQL3 trigger standard- the development to obtain convergence among relational databases was quite hard [36].

4. PG-Triggers Definition

We next propose PG-Triggers, by discussing their syntax and semantics.

```
CREATE TRIGGER <name> <time> <event> ON <label>[.<property>] [REFERENCING <alias>...]
FOR <granularity> <item> [WHEN <condition>]
BEGIN <statement> END

<time> ::= { BEFORE | AFTER | ONCOMMIT | DETACHED }
<event> ::= { CREATE | DELETE | SET | REMOVE }
<granularity> ::= { EACH | ALL }
<item> ::= { NODE | RELATIONSHIP }
<alias> ::=  [OLDNODES | OLDRELS] AS <alias for old items> |
             [NEWNODES | NEWRELS] AS <alias for new items> |
             OLD AS <alias for old single item> | NEW AS <alias for new single item>
```

Figure 1: PG-Trigger Syntax

4.1. Syntax

The PG-Triggers syntax, shown in Figure 1, is borrowed – as much as possible – from the SQL3 standard, as discussed, e.g., in Chapter 11 of [8]. In our notation, upper case letters are reserved for terminal symbols; nonterminals are in a low case and enclosed within <> (angle brackets); optionality is denoted by [] (square brackets); items of which only one is required are enclosed within braces {}; alternatives are separated by the | symbol; and ellipsis points show that the preceding element can be repeated.

Note that, due to the richness of the graph data model w.r.t. the relational model, the syntax has many more options. In particular, note that graph items can either be nodes or relationships; we use a given label to select, out of all items, the specific set of items that is the trigger's *target*. Note also that trigger events in graph databases are richer than those in relational databases, as they include the creation and deletion of nodes/relationships as well as the setting and removal of their labels and properties. In analogy with the UPDATE event of the SQL3 standard, the SET and REMOVE events can refer to properties; thus, the ON clause may refer to labels (for nodes, relationships, and label themselves) but also to properties, identified by a `<label>.<property>`) pair.

4.2. Semantics

As usual, triggers include a `<condition>` predicate, which is considered at given action `<time>`(s), and a connected action `<statement>`, which is executed only if the corresponding condition predicate holds. We next describe the trigger semantics along the classical dimensions, by making explicit reference to their syntactic elements.

- **Granularity.** We assume that each trigger execution is linked to a given query in a graph query language (GQL or Cypher); from our point of view, a query operates on an

initial state and produces a final state, by creating or removing <item>s (i.e., nodes and relationships), or by setting or removing their labels and properties. These changes are considered at two possible levels of <granularity>: either individually (FOR EACH clause) or collectively as a set (FOR ALL clause).

- **Action Time.** As in relational databases, we consider triggers occurring BEFORE and AFTER the statement; in addition, we offer the ONCOMMIT and DETACHED option. As with relational triggers, BEFORE statements should not produce arbitrary changes, but just condition NEW states. ONCOMMIT execution occurs before the commit execution, within the same transaction (possibly causing a rollback of the transaction as a whole), while DETACHED execution occurs after a successful commit and operates within an autonomous transaction. Triggers sharing the same action time must be ordered (see next).
- **Targeting.** Triggers in relational databases are targeted to tables. We opted for using labels as providers of an analogous context; therefore, in the ON clause, we select as target the items with a particular <label>.
- **Event Types.** Events refer to either nodes or relationships and include their creation or deletion, and the setting or removal of labels and properties.
- **Transition Variables.** With individual-level granularity, OLD and NEW refer to the old and new state, respectively. With set-level granularity, transition variables are postfixed by the NODES keyword or by the RELS keyword; clearly, in this case, the item must be the same as in the FOR clause. Along with [8], our proposed syntax offers an option for renaming transition variables through the AS clause, for referring to them mnemonically with respect to the application domain.

Discussion. This definition of semantics is quite close to the relational one, but some differences require further discussion.

- **Choice of** LABELS**.** Adopting labels for identifying the trigger's target appears to be the most natural choice in the case of Property Graphs – every node and relationship, sharply, either belongs or does not belong to the set of items with a given label. Still, the situation is more complex than in the relational case, as a node or relationship may have no label, or instead it may have more than one associated label, whereas a tuple belongs to exactly one table in the relational model.[1] For clarity of the execution semantics, we also make the assumption that *no trigger can monitor the setting or removal of its target label* and that *the target label cannot be set or removed within the <statement>*.
- The ONCOMMIT option, which is not supported in relational databases, is supported by Neo4j and Memgraph, although with several options and different semantics for each option. Our interpretation of ONCOMMIT is to consider and execute the trigger when the transaction reaches the commit point, and then include also triggers' side effects before actually committing. The ONCOMMIT option is in part justified in graph databases by the interleaving of MATCH clauses with node and relationship creations, updates and deletions, which make the context-of-operation less well-defined than in a relational database.[2]

[1] Labels could be substituted by PG-Types (with STRICT option, which makes them compulsory for all nodes or relationships) if they will become widely used and accepted as standard.

[2] The ONCOMMIT option is sometimes advocated for relational databases, e.g., in practitioners' blogs.

- Similarly, for what concerns the order of execution of different triggers that are activated by the same Cypher or GQL query, these queries are generally much more powerful than relational update queries, targeted to a single table, as they can update multiple nodes and relationships, with different labels. Hence, the most sensible option for prioritizing them is to resort to the trigger creation time, thus providing a total order.[3] In all cases, with a given execution order, the execution semantics of cascading triggers should mimic the relational one, with the stack of trigger execution contexts as described, e.g., in [8].

4.3. Example

Consider a graph database whose schema, among other things, contains nodes of type `Mutation` (with strings `name` and `protein`) and `CriticalEffect` (with the `description` string), connected by arcs of type `Risk`. The simple PG-Trigger shown below reacts to the fact that a new mutation is associated with a critical effect by creating an alert with the name of the mutation.

```
CREATE TRIGGER NewCriticalMutation AFTER CREATE ON 'Mutation' FOR EACH NODE
WHEN EXISTS (NEW)-[:Risk]-(:CriticalEffect)
BEGIN CREATE (:Alerttime:DATETIME(),desc:'New critical mutation',mutation:NEW.name) END
```

Besides cases of reactions to node, relationship or property creation, more complex scenarios include conditions using thresholds or state comparisons, or even side effects in the action of the trigger. For these kinds of examples and their rendering as APOC triggers, the interested reader can refer to [9].

5. Discussion and Conclusion

In this article, we have shown that adding reactive components to graph databases is at the same time very natural, along the SQL3 standard, and also very useful, as reactive programming can leverage graph databases in supporting important applications. We have shown that the concepts of PG-Triggers descend naturally from relational concepts, although they require adaptation to the richer graph model, which includes nodes, relationships, labels, and properties.

We have argued that PG-Triggers can be supported on top of Neo4j by making use of the APOC trigger procedures. A major drawback is that APOC triggers miss some important ingredients for being fully compliant with standardization needs, including the mastering of activation before or after operations that cause triggering and a complete and correct management of cascading changes. One objective of this article is also to motivate graph database companies, such as Neo4j, to support triggers as part of their standard offer.

Acknowledgments

This work was carried out within the MICS (Made in Italy - Circular and Sustainable) Extended Partnership and received funding from Next-Generation EU (Italian PNRR - M4 C2, Invest 1.3 - D.D. 1551.11-10-2022, PE00000004). CUP MICS D43C22003120001. This work is part of Davide Magnanimi's Executive PhD program at Politecnico di Milano.

[3] Another option would be to use a total order based on the names of the triggers, as some relational databases do (e.g., PostgreSQL; see https://www.postgresql.org/docs/current/sql-createtrigger.html).

References

[1] S. Sakr, A. Bonifati, H. Voigt, A. Iosup, K. Ammar, R. Angles, W. Aref, M. Arenas, M. Besta, P. A. Boncz, et al., The future is big graphs: a community view on graph processing systems, Communications of the ACM 64 (2021) 62–71.

[2] A. Green, P. Furniss, T. Lindaaker, P. Selmer, H. Voigt, S. Plantikow, Iso, tech. rep: GQL scope and features, https://s3.amazonaws.com/artifacts.opencypher.org/website/materials/sql-pg-2018-0046r3-GQL-Scope-and-Features.pdf, 2019. Last accessed online: Nov 20th, 2023.

[3] A. Bonifati, G. Fletcher, H. Voigt, N. Yakovets, Querying Graphs, Springer International Publishing, Cham, 2018.

[4] R. Angles, A. Bonifati, S. Dumbrava, G. Fletcher, K. W. Hare, J. Hidders, V. E. Lee, B. Li, L. Libkin, W. Martens, F. Murlak, J. Perryman, O. Savković, M. Schmidt, J. Sequeda, S. Staworko, D. Tomaszuk, PG-Keys: Keys for Property Graphs, in: Proceedings of the 2021 International Conference on Management of Data, Association for Computing Machinery, New York, NY, USA, 2021, pp. 2423–2436.

[5] R. Angles, A. Bonifati, S. Dumbrava, G. Fletcher, A. Green, J. Hidders, B. Li, L. Libkin, V. Marsault, W. Martens, F. Murlak, S. Plantikow, O. Savković, et al., PG-Schema: Schemas for Property Graphs, in: Proceedings of the 2023 International Conference on Management of Data, Association for Computing Machinery, New York, NY, USA, 2023.

[6] K. P. Eswaran, Aspects of a trigger subsystem in an integrated database system, in: Proceedings of the 2nd International Conference on Software Engineering, ICSE '76, IEEE Computer Society Press, Washington, DC, USA, 1976, p. 243–250.

[7] J. Widom, S. Ceri, Active database systems: Triggers and rules for advanced database processing, Morgan Kaufmann, 1995.

[8] J. Melton, A. R. Simon, SQL: 1999: understanding relational language components, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[9] S. Ceri, A. Bernasconi, A. Gagliardi, D. Martinenghi, L. Bellomarini, D. Magnanimi, PG-Triggers: Triggers for Property Graphs, in: Proceedings of the 2024 ACM SIGMOD/PODS International Conference on Management of Data – SIGMOD 2024, June 9-15, 2024, Santiago, Chile, 2024. To appear.

[10] APOC Library, Awesome Procedures for Neo4j 5.9.0.x (Extended), https://github.com/neo4j-contrib/neo4j-apoc-procedures, 2023. Last accessed online: Nov 20th, 2023.

[11] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, A. Taylor, Cypher: An evolving query language for property graphs, in: Proceedings of the 2018 International Conference on Management of Data, Association for Computing Machinery, New York, NY, USA, 2018, pp. 1433–1445.

[12] Neo4j, Neo4j, https://neo4j.com/, 2023. Last accessed online: Nov 20th, 2023.

[13] Memgraph, Memgraph, https://memgraph.com/, 2023. Last accessed online: Nov 20th, 2023.

[14] JanusGraph, JanusGraph, https://janusgraph.org/, 2023. Last accessed online: Nov 20th, 2023.

[15] Amazon, Amazon Neptune, https://aws.amazon.com/it/neptune/, 2023. Last accessed online: Nov 20th, 2023.

[16] Nebula Graph, Nebula Graph, https://vaticle.com/, 2023. Last accessed online: Nov 20th, 2023.
[17] TigerGraph, TigerGraph, https://www.tigergraph.com/, 2023. Last accessed online: Nov 20th, 2023.
[18] GQL Manifesto, The GQL manifesto - one property graph query language, https://gql.today/, 2018.
[19] LDBC, LDBC Graph Query Working Group, https://ldbcouncil.org/gql-community/overview/, 2023. Last accessed online: Nov 20th, 2023.
[20] R. Angles, M. Arenas, P. Barceló, P. Boncz, G. Fletcher, C. Gutierrez, T. Lindaaker, M. Paradies, S. Plantikow, J. Sequeda, et al., G-CORE: A core for future graph query languages, in: Proceedings of the 2018 International Conference on Management of Data, Association for Computing Machinery, New York, NY, USA, 2018, pp. 1421–1432.
[21] C. Kankanamge, S. Sahu, A. Mhedbhi, J. Chen, S. Salihoglu, Graphflow: An active graph database, in: Proceedings of the 2017 ACM International Conference on Management of Data, Association for Computing Machinery, New York, NY, USA, 2017, pp. 1695–1698.
[22] K. Kim, I. Seo, W.-S. Han, J.-H. Lee, S. Hong, H. Chafi, H. Shin, G. Jeong, Turboflux: A fast continuous subgraph matching system for streaming graph data, in: Proceedings of the 2018 International Conference on Management of Data, Association for Computing Machinery, New York, NY, USA, 2018, pp. 411–426.
[23] solid IT consulting, DB-Engines ranking of graph DBMS, https://db-engines.com/en/ranking/graph+dbms, 2023.
[24] opencypher, http://www.opencypher.org/, 2023.
[25] Dgraph, Dgraph Lambda, https://dgraph.io/blog/post/dgraph-lambda/, 2023. Last accessed online: Nov 20th, 2023.
[26] W3C, SPARQL, https://www.w3.org/TR/rdf-sparql-query/, 2023. Last accessed online: Nov 20th, 2023.
[27] Dgraph, Dgraph, https://dgraph.io/, 2023. Last accessed online: Nov 20th, 2023.
[28] Stardog, Stardog, https://www.stardog.com/, 2023. Last accessed online: Nov 20th, 2023.
[29] Ontotext, GraphDB, https://www.ontotext.com/, 2023. Last accessed online: Nov 20th, 2023.
[30] Oracle, Graph Database and Graph Analytics, https://www.oracle.com/database/graph/, 2023. Last accessed online: Nov 20th, 2023.
[31] OpenLink, Virtuoso, https://virtuoso.openlinksw.com/, 2023. Last accessed online: Nov 20th, 2023.
[32] Bitnine, AgensGraph, https://bitnine.net/agensgraph, 2023. Last accessed online: Nov 20th, 2023.
[33] Microsoft, Azure Cosmos DB, https://azure.microsoft.com/services/cosmos-db, 2023. Last accessed online: Nov 20th, 2023.
[34] OrientDB, OrientDB, https://orientdb.org/, 2023. Last accessed online: Nov 20th, 2023.
[35] ArangoDB, ArangoDB, https://www.arangodb.com/, 2023. Last accessed online: Nov 20th, 2023.
[36] J. Widom, S. Ceri, Standards and commercial systems, in: J. Widom, S. Ceri (Eds.), Active database systems: Triggers and rules for advanced database processing, Morgan Kaufmann, 1996, pp. 233–258.