G

Graph Query Languages

Renzo Angles[1], Juan Reutter[2], and Hannes Voigt[3]
[1]Universidad de Talca, Talca, Chile
[2]Pontificia Universidad Catolica de Chile, Santiago, Chile
[3]Dresden Database Systems Group, Technische Universität Dresden, Dresden, Germany

Definition

A query language is a high-level computer language for the retrieval and modification of data held in databases or files. Query languages usually consist of a collection of operators which can be applied to any valid instances of the data structure types of a data model, in any combination desired.

In the context of graph data management, a *graph query language* (GQL) defines the way to retrieve or extract data which have been modeled as a graph and whose structure is defined by a graph data model. Therefore, a GQL is designed to support specific graph operations, such as graph pattern matching and shortest path finding.

Overview

Research on graph query languages has at least 30 years of history. Several GQLs were proposed during the 1980s, most of them oriented to study and define the theoretical foundations of the area. During the 1990s, the research on GQLs was overshadowed by the appearance of XML, which was arguably seen as the main alternative to relational databases for scenarios involving semi-structured data. With the beginning of the new millennium, however, the area of GQLs gained new momentum, driven by the emergence of applications where cyclical relationships naturally occur, including social networks, the Semantic Web, and so forth. Currently GQLs are a fundamental tool that supports the manipulation and analysis of complex big data graphs. Detailed survey of graph query language research can be found in Angles et al. (2017b).

Key Research Findings

Most of the languages that have been proposed are based on the idea of matching a *graph pattern*. In designing a GQL one typically starts with graph patterns, to which several other features are added. Examples of these features include set operations, relational algebra operations, the addition of path queries, closing graph patterns through recursion, etc. In the following we describe how graph pattern matching works, as well

This work was supported by Iniciativa Científica Milenio Grant 120004.

S. Sakr, A. Zomaya (eds.), *Encyclopedia of Big Data Technologies*,
https://doi.org/10.1007/978-3-319-63962-8_75-1

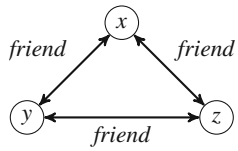
as the most typical features commonly added to graph patterns.

Since our focus is on the fundamentals of GQLs, we concentrate on the simplest data model for graphs, i.e., directed labeled graphs. Later on, we review how these fundamentals are translated into practical query languages in more complex graph data models.

Let Σ be a finite alphabet. A *graph database* G over Σ is a pair (V, E), where V is a finite set of nodes and $E \subseteq V \times \Sigma \times V$ is a set of edges. That is, we view each edge as a triple $(v, a, v') \in V \times \Sigma \times V$, whose interpretation is an a-labeled edge from v to v' in G.

Graph Pattern Matching

The idea of patterns is to define small subgraphs of interest to look for in an input graph. For example, in a social network one can match the following pattern to look for a clique of three individuals that are all friends with each other:



Let $\mathbf{V} = \{x, y, z, \ldots\}$ be an infinite set of variables. In its simplest form, a graph pattern is simply a graph $P = (V_P, E_P)$ where all the nodes in V_P are replaced by variables. For example, the pattern above can be understood as a graph over alphabet $\Sigma = \{friend\}$ whose set of nodes is $\{x, y, z\}$ and whose edges are $\{(x, friend, y), (y, friend, x), (x, friend, z), (z, friend, x), (y, friend, z), (z, friend, y)\}$.

The semantics of patterns is given using the notion of a match. A match of a pattern $P = (V_P, E_P)$ over a graph $G = (V_G, E_G)$ is a mapping μ from variables to constants, such that the graph resulting from replacing each variable for the corresponding constant in the pattern is actually a subgraph of G, i.e., for every pattern node $x \in V_P$, we have that $\mu(x) \in V_G$, and for each edge $(x, a, y) \in E_P$, we have that $(\mu(x), a, \mu(y))$ belongs to E_G.

The problem of pattern matching has been extensively studied in the literature, and several other notions of matching have been considered. The basic notion defined above is known as a *homomorphism* and constitutes the most common semantics. Additionally, one could also consider *simple path semantics*, which does not permit repeating nodes and used in, e.g., the languages G (Cruz et al. 1987b) and $G+$ (Cruz et al. 1989), *injective homomorphism*, that disallow variables to be mapped to the same element or semantics based on graph simulations (see, e.g., Miller et al. 2015 for a survey).

Although the set of matchings from P to G can be already seen as the set of answers for the evaluation of P over G, one is normally interested in tuples of nodes: if P uses variables $x_1, \ldots, x_n$, then we define the result of evaluating P over G, which we denote by $P(G)$, as the set of tuples $\{(\mu(x_1), \ldots, \mu(x_n)) \mid \mu \text{ is a match from } P \text{ to } G\}$.

Extensions. Graph patterns can be extended by adding numerous features. For starters, one could allow variables in the edge labels of patterns or allow a mixture between constant and variables in nodes or edges. In this case, mappings must be the identity on constant values, and if an edge (x, y, z) occurs in a pattern, then $\mu(y)$ must be a label in Σ, so that $(\mu(x), \mu(y), \mu(z))$ is an edge in the graph upon we are matching. Patterns can also be augmented with *projection*, and since the answer is a set of tuples, we can also define set operations (*union*, *intersection*, *difference*, etc.) over them. Likewise, one can define *filters* that force the matches to satisfy certain Boolean conditions such as $x \neq y$. Another operator is the left outer join, also known as the *optional* operator. This operator allows to query incomplete information (an important characteristic of graph databases).

Graph Patterns as Relational Database Queries. Graph patterns can also be defined as *conjunctive queries* over the relational representation of graph databases (see, e.g., Abiteboul et al. 1995 for a good introduction on relational database theory). In order to do this, given an alphabet Σ, we define $\sigma(\Sigma)$ as the relational schema that consists of one binary predicate

symbol E_a, for each symbol $a \in \Sigma$. For readability purposes, we identify E_a with a, for each symbol $a \in \Sigma$. Each graph database $G = (V, E)$ over Σ can be represented as a relational instance $\mathbf{D}(G)$ over the schema $\sigma(\Sigma)$: The database $\mathbf{D}(G)$ consists of all facts of the form $E_a(v, v')$ such that (v, a, v') is an edge in G (for this we assume that $\mathbf{D}$ includes all the nodes in V). It is then not difficult to see that graph patterns are actually *conjunctive queries* over the relational representation of graphs, that is, formulas of the form $Q(\bar{x}) = \exists \bar{y} \phi(\bar{x}, \bar{y})$, where $\bar{x}$ and $\bar{y}$ are tuples of variables and $\phi(\bar{x}, \bar{y})$ is a conjunction of relational atoms from σ that use variables from $\bar{x}$ to $\bar{y}$. For example, the pattern shown above can be written as the relational query $Q(x, y, z) = \mathit{friend}(x, y) \wedge \mathit{friend}(y, x) \wedge \mathit{friend}(x, z) \wedge \mathit{friend}(z, x) \wedge \mathit{friend}(y, z) \wedge \mathit{friend}(z, y)$.

The correspondence between patterns and conjunctive queries allows us to reuse the broad corpus on literature on conjunctive queries for our graph scenario. We also use this correspondence when defining more complex classes of graph patterns.

Path Queries

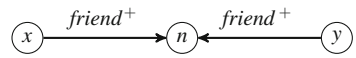
The idea of a path query is to select pairs of nodes in a graph whose relationship is given by a path (of arbitrary length) satisfying certain properties. For example, one could use a path query in a social network to extract all nodes that can be reached from an individual via friend-of-a-friend paths of arbitrary length.

The most simple navigational querying mechanism for graph databases is a *regular path query* (*RPQ*) (Abiteboul et al. 1999; Cruz et al. 1987a; Calvanese et al. 2002), which allows to specify navigation by means of regular expressions. Formally, an RPQ Q over Σ is a regular language $L \subseteq \Sigma^*$, and it is specified using a regular expression R. Their semantics is defined in terms of paths. A *path* ρ from v_0 to v_m in a graph $G = (V, E)$ is a sequence $(v_0, a_0, v_1), (v_1, a_1, v_2), \cdots, (v_{m-1}, a_{m-1}, v_m)$, for some $m \geq 0$, where each (v_i, a_i, v_{i+1}), for $i < m$, is an edge in E. In particular, all the v_i's are nodes in V and all the a_j's are letters in Σ. The *label* of ρ is the word $a_0 \cdots a_{m-1} \in \Sigma^*$.

The idea behind regular path queries is to select all pairs of nodes whose labels belong to the language of the defining regular expression. That is, given a graph database $G = (V, E)$ and an RPQ R, the evaluation of R over G, denoted $[\![R]\!]_G$, is the set of all pairs $(v, v') \in V$ such that there is path ρ between v and v' and the label of ρ is a word in the language of R.

More Complex Path Queries. RPQs have been extended in a myriad of ways. For example, we have *two-way regular path queries* (2RPQs) (Calvanese et al. 2000) that add to RPQs the ability to traverse edge backward via inverse symbols a^-, *nested* regular path queries (Barceló et al. 2012b) that further extend 2RPQs with node tests similar to what is done with propositional dynamic logic or XPath expressions, and several proposals adding variables to path queries (Wood 1990; Santini 2012; Fionda et al. 2015). There are also proposals of more expressive forms of path queries, using, for example, context-free languages (Hellings 2014), and several formalisms aimed at comparing pieces of data that may be associated with the nodes in a graph (see, e.g., Libkin and Vrgoč 2012). Another type of path queries, called *property paths*, extends RPQs with a mild form of negation (Kostylev et al. 2015).

Adding Path Queries to Patterns. Path queries and patterns can be naturally combined to form what is known as *conjunctive regular path queries*, or CRPQs (Florescu et al. 1998; Calvanese et al. 2000). Just as with graph patterns and conjunctive queries, we can define CRPQs in two ways. From a pattern perspective, CRPQs amount to replacing some of the edges in a graph pattern by a regular expression, such as the following CRPQ, which looks for two paths with an indirect friend n in common:



However, a CRPQ over an alphabet Σ can also be seen as a conjunctive query over a relational

representation of graphs that includes a predicate for each regular expression constructed from Σ. In our case, the corresponding conjunctive query for the pattern above is written as $Q(x, y) = friend^+(x, n) \wedge friend^+(y, n)$. Note that here we are using n as a constant; since CRPQs are again patterns, they can be augmented with any of the features we listed for patterns, including constants, projection, set operations, filters, and outer joins. We can also obtain more expressive classes of queries by including more expressive forms of path queries: using 2RPQs instead of RPQs gives rise to C2RPQs, or conjunctive 2RPQs, and using nested regular expressions gives rise to CNREs, which are also known as *conjunctive nested path queries*, or CNPQs (Bienvenu et al. 2014; Bourhis et al. 2014). Finally, Barceló et al. (2012a) consider restricting several path queries in a CRPQ at the same time by means of regular relations. The resulting language is capable of expressing, for example, that the two paths of friends in the pattern above must be of the same length.

Algorithmic Issues. The most important algorithmic problem studied for GQLs is *query answering*. This problem asks, given a query Q with k variables, a graph G and a tuple $\bar{a}$ of k values from G, whether $\bar{a}$ belongs to the evaluation of Q over G. All of the classes of path queries we have discussed can be evaluated quite efficiently. For example, all of RPQs, 2RPQs, and NREs can be evaluated in linear time in the size of Q and of G. Barceló (2013) provides a good survey on the subject.

When adding path queries to patterns, one again hopes that the resulting language will not be more difficult to evaluate than standard patterns or conjunctive queries. And indeed this happens to be the case: The evaluation problem for CRPQs, C2RPQs, and CNREs is NP-complete just as patterns. In fact, one can show this for any pattern augmented with a form of path query, as long as the evaluation for these path queries is in PTIME.

The second problem that has received considerable attention is *query containment*. This problem asks, given queries Q_1 and Q_2, whether the answers of Q_1 are always contained in the answers of Q_2. Since all queries we presented are based on graph patterns, the lower bound for this problem is NP, as containment is known to be NP-complete already for simple graph patterns. In contrast to evaluation, adding path queries to patterns can significantly alter the complexity of containment: for CRPQs the problem is already EXPSPACE-complete (Calvanese et al. 2000), and depending on the features considered, it may even become undecidable.

Beyond Patterns

For the navigational languages, we have seen, thus far, paths are the only form of recursion allowed, but to express certain types of queries, we may require more expressive forms of recursion. As an example, suppose now our social network admits labels *friends* and *works_with*, the latter for joining two nodes that are colleagues. Now imagine that we need to find all nodes x and y that are connected by a path where each subsequent pair of nodes in the path is connected by both *friend* and *works_with* relations. We cannot express this query as a regular expression between paths, so instead what we do is to adopt *Datalog* as a framework for expressing the repetitions of patterns themselves. Consider, for example, a rule of the form:

$$P(x, y) \leftarrow friend(x, y), works\_with(x, y).$$

This rule is just another way of expressing the pattern that computes all pairs of nodes connected both by *friend* and *works_with* relations. In this case, P represents a new relation, known as an *intensional* predicate, that is intended to compute all pairs of nodes that are satisfied by this pattern. We can then use P to specify the rule:

$$\text{Ans}(x, y) \leftarrow P^+(x, y).$$

Now both rules together form what is known as a *regular query* (Reutter et al. 2015) or nested positive 2RPQs (Bourhis et al. 2014, 2015). The idea is that $P^+(x, y)$ represents all pairs x, y connected by a path of nodes, all of them satisfying the pattern P (i.e., connected both by *friend*

and *works_with* relations). Regular queries can be understood as the language resulting of extending graph patterns with edges that can be labeled not only with a path query but with any arbitrary repetition of a binary pattern.

The idea of using Datalog as a graph query language for graph databases comes from Consens and Mendelzon (1990), where it was introduced under the name GraphLog. GraphLog opened up several possibilities for designing new query languages for graphs, but one of the problems of GraphLog was that it was too expressive, and therefore problems such as query containment were undecidable for arbitrary GraphLog queries. However, interest in Datalog for graphs has returned in the past few years, as researchers started to restrict Datalog programs in order to obtain query languages that can allow expressing the recursion or repetition of a given pattern, but at the same time maintaining good algorithmic properties for the query answering problem. For example, the complexity for evaluating regular queries is the same as for C2RPQs, and the containment problem is just one exponential higher. Besides regular queries, other noteworthy graph languages follow by restricting to monadic Datalog programs (Rudolph and Krötzsch 2013), programs whose rules are either guarded or frontier-guarded (Bourhis et al. 2015; Bienvenu et al. 2015), and first-order logic with transitive closure (Bourhis et al. 2014).

Composability

A particularly important feature of query languages is *composability*. Composable query languages allow to formulate queries with respect to results of other queries expressed in the same language. Composability is an important principle for the design of simple but powerful programming languages in general and query languages in particular (Date 1984). To be composable, the input and the output of a query must have the same (or a compatible) data model.

CQs (and C2RPQs) are not inherently composable, since their output is a relation and their input is a graph. In contrast, RPQs (and 2RPQs) allow query composition, since the set of pairs of nodes resulting from an RPQ can be interpreted as edges obviously. Regular queries are composable for the same reason. In general, Datalog-based graph query languages are composable over relations but not necessarily over graphs.

In the context of modern data analytics, data is collected to a large extent automatically by hard- and software sensors in fine granularity and low abstraction. Where users interact with data, they typically think, reason, and talk about entities of larger granularity and higher abstraction. For instance, social network data is collected in terms of *friend* relationship and individual messages send from one person to another, while the user is often interested in communities, discussion threads, topics, etc. User-level concepts are often multiple abstraction levels higher than the concepts in which data is captured and stored. The query language of database systems is the main means for users to derive information in terms of their user-level concepts for a database oblivious of user's concepts. To offer a capability of conceptual abstraction, a query language has to (1) be able to create entirely new entities within the data model to lift data into higher-level concepts and (2) be composable over its data model, so that the users can express a stack of multiple such conceptual abstraction steps with the same language.

RPQs (and 2RPQs) and regular queries allow conceptual abstraction for edges. For instance, *ancestor* edges can be derived from sequences of *parent* edges. Conceptual abstraction for nodes has been considered only recently (Rudolf et al. 2014; Voigt 2017) and allows to derive (or construct) an entirely new output graph from the input graph. In the simplest form of *graph construction*, a new node is derived from each tuple resulting from a pattern match (C2RPQs, etc.), e.g., deriving a *marriage* node from every pair of persons that are mutually connected by $married - to$ edges. The more general and more expressive variant of graph construction includes aggregation and allows to derive a new node from every group of tuples given a set of variables as grouping keys, e.g., deriving a *parents* node from every group of adults all connected by $parent - to$ to the same child.

Key Applications

There is a vast number of GQLs that have been proposed and implemented by graph databases and an even larger number of GQLs that are used in specific application domains (see, e.g., Dries et al. 2009; Brijder et al. 2013; Martín et al. 2011). In the following we focus on GQLs designed for the two most popular graph data models: property graphs and RDF.

Property Graph Databases

A property graph is a directed labeled multigraph with the special characteristic that each node or edge can maintain a set (possibly empty) of properties, usually represented as key-value pairs. The property graph data model is very popular in current graph database systems, including Neo4j, Oracle PGX, Titan, and Amazon Neptune.

There is no standard query language for property graphs although some proposals are available. Blueprints (http://tinkerpop.apache.org/) was one of the first libraries created for manipulating property graphs. Blueprints is analogous to the JDBC, but for graph databases. Gremlin (Rodriguez 2015) is a graph-based programming language for property graphs developed within the TinkerPop open-source project. Gremlin makes extensive use of XPath to support complex graph traversals. Cypher is a declarative query language defined for the Neo4j graph database engine, originally, and by now adopted by other implementers. Cypher supports basic graph patterns and basic types of regular path queries (i.e., paths with a fixed edge label or paths with any labels) with a no-repeating-edges matching semantics. The development of Cypher is a still ongoing community effort (http://www.opencypher.org/).

PGQL (van Rest et al. 2016) is a query language for property graphs recently developed as part of Oracle PGX, an in-memory graph analytic framework. PGQL defines a SQL-like syntax that allows to express graph pattern matching queries, regular path queries (with conditions on labels and properties), graph construction, and query composition.

G-CORE (Angles et al. 2017a) is a language recently proposed by the Linked Data Benchmark Council (LDBC). G-CORE advertises regular queries, nested weighted shortest path queries, graph construction with aggregation, and composability as the core feature of future graph query languages on property graphs.

RDF Databases

The Resource Description Framework (RDF) is a W3C recommendation that defines a graph-based data model for describing and publishing data on the web. This data model gains popularity in the context of managing web data, giving place to the development of RDF databases (also called triplestores). Along with these database systems, several RDF query languages were developed (we refer the reader to a brief survey by Haase et al. 2004). Currently, SPARQL (Prud'hommeaux and Seaborne 2008) is the standard query language for RDF. SPARQL was designed to support several types of complex graph patterns and, in its latest version, SPARQL 1.1 (Harris and Seaborne 2013), adds support for negation, regular path queries (called *property paths*), subqueries, and aggregate operators. The path queries support reachability tests.

Cross-References

- Graph Data Management Systems
- Graph Data Models
- Graph Pattern Matching
- Graph Query Processing

References

Abiteboul S, Hull R, Vianu V (1995) Foundations of databases. Addison-Wesley, Reading

Abiteboul S, Buneman P, Suciu D (1999) Data on the Web: from relations to semistructured data and XML. Morgan Kauffman, San Francisco

Angles R, Arenas M, Barceló P, Boncz PA, Fletcher GHL, Gutierrez C, Lindaaker T, Paradies M, Plantikow S, Sequeda J, van Rest O, Voigt H (2017a) G-CORE: a core for future graph query languages. The computing research repository abs/1712.01550

Angles R, Arenas M, Barceló P, Hogan A, Reutter JL, Vrgoc D (2017b) Foundations of modern query languages for graph databases. ACM Comput Surv 68(5):1–40

Barceló P (2013) Querying graph databases. In: Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, PODS 2013, pp 175–188

Barceló P, Libkin L, Lin AW, Wood PT (2012a) Expressive languages for path queries over graph-structured data. ACM Trans Database Syst (TODS) 37(4):31

Barceló P, Pérez J, Reutter JL (2012b) Relative expressiveness of nested regular expressions. In: Proceedings of the Alberto Mendelzon workshop on foundations of data management (AMW), pp 180–195

Bienvenu M, Calvanese D, Ortiz M, Simkus M (2014) Nested regular path queries in description logics. In: Proceeding of the international conference on principles of knowledge representation and reasoning (KR)

Bienvenu M, Ortiz M, Simkus M (2015) Navigational queries based on frontier-guarded datalog: preliminary results. In: Proceeding of the Alberto Mendelzon workshop on foundations of data management (AMW), p 162

Bourhis P, Krötzsch M, Rudolph S (2014) How to best nest regular path queries. In: Informal Proceedings of the 27th International Workshop on Description Logics

Bourhis P, Krötzsch M, Rudolph S (2015) Reasonable highly expressive query languages. In: Proceeding of the international joint conference on artificial intelligence (IJCAI), pp 2826–2832

Brijder R, Gillis JJM, Van den Bussche J (2013) The DNA query language DNAQL. In: Proceeding of the international conference on database theory (ICDT). ACM, pp 1–9

Calvanese D, De Giacomo G, Lenzerini M, Vardi MY (2000) Containment of conjunctive regular path queries with inverse. In: Proceeding of the international conference on principles of knowledge representation and reasoning (KR), pp 176–185

Calvanese D, De Giacomo G, Lenzerini M, Vardi MY (2002) Rewriting of regular expressions and regular path queries. J Comput Syst Sci (JCSS) 64(3):443–465

Consens M, Mendelzon A (1990) Graphlog: a visual formalism for real life recursion. In: Proceeding of the ACM symposium on principles of database systems (PODS), pp 404–416

Cruz I, Mendelzon A, Wood P (1987a) A graphical query language supporting recursion. In: ACM special interest group on management of data 1987 annual conference (SIGMOD), pp 323–330

Cruz IF, Mendelzon AO, Wood PT (1987b) A graphical query language supporting recursion. In: Proceeding of the ACM international conference on management of data (SIGMOD), pp 323–330

Cruz IF, Mendelzon AO, Wood PT (1989) G+: recursive queries without recursion. In: Proceeding of the international conference on expert database systems (EDS). Addison-Wesley, pp 645–666

Date CJ (1984) Some principles of good language design (with especial reference to the design of database languages). SIGMOD Rec 14(3):1–7

Dries A, Nijssen S, De Raedt L (2009) A query language for analyzing networks. In: Proceeding of the ACM international conference on information and knowledge management (CIKM). ACM, pp 485–494

Fionda V, Pirrò G, Consens MP (2015) Extended property paths: writing more SPARQL queries in a succinct way. In: Proceeding of the conference on artificial intelligence (AAAI)

Florescu D, Levy AY, Suciu D (1998) Query containment for conjunctive queries with regular expressions. In: Proceeding of the ACM symposium on principles of database systems (PODS), pp 139–148

Haase P, Broekstra J, Eberhart A, Volz R (2004) A comparison of RDF query languages. In: Proceeding of the international Semantic Web conference (ISWC), pp 502–517

Harris S, Seaborne A (2013) SPARQL 1.1 query language. W3C recommendation. http://www.w3.org/TR/sparql11-query/

Hellings J (2014) Conjunctive context-free path queries. In: Proceeding of the international conference on database theory (ICDT), pp 119–130

Kostylev EV, Reutter JL, Romero M, Vrgoč D (2015) SPARQL with property paths. In: Proceeding of the international Semantic Web conference (ISWC), pp 3–18

Libkin L, Vrgoč D (2012) Regular path queries on graphs with data. In: Proceeding of the international conference on database theory (ICDT), pp 74–85

Martín MS, Gutierrez C, Wood PT (2011) SNQL: a social networks query and transformation language. In: Proceeding of the Alberto Mendelzon workshop on foundations of data management (AMW)

Miller JA, Ramaswamy L, Kochut KJ, Fard A (2015) Research directions for big data graph analytics. In: Proceeding of the IEEE international congress on big data, pp 785–794

Prud'hommeaux E, Seaborne A (2008) SPARQL query language for RDF. W3C recommendation. http://www.w3.org/TR/rdf-sparql-query/

Reutter JL, Romero M, Vardi MY (2015) Regular queries on graph databases. In: Proceeding of the international conference on database theory (ICDT), pp 177–194

Rodriguez MA (2015) The Gremlin graph traversal machine and language. In: Proceeding of the international workshop on database programming languages. ACM

Rudolf M, Voigt H, Bornhövd C, Lehner W (2014) SynopSys: foundations for multidimensional graph analytics. In: Castellanos M, Dayal U, Pedersen TB, Tatbul N (eds) BIRTE'14, business intelligence for the real-time enterprise, 1 Sept 2014. Springer, Hangzhou, pp 159–166

Rudolph S, Krötzsch M (2013) Flag & check: data access with monadically defined queries. In: Proceeding of the symposium on principles of database systems (PODS). ACM, pp 151–162

Santini S (2012) Regular languages with variables on graphs. Inf Comput 211:1–28

van Rest O, Hong S, Kim J, Meng X, Chafi H (2016) PGQL: a property graph query language. In: Proceeding of the workshop on graph data-management experiences and systems (GRADES)

Voigt H (2017) Declarative multidimensional graph queries. In: Proceeding of the 6th European business intelligence summer schoole (BISS). LNBIP, vol 280. Springer, pp 1–37

Wood PT (1990) Factoring augmented regular chain programs. In: Proceeding of the international conference on very large data bases (VLDB), pp 255–263