**Graph Database Management Systems – the past, the present and the future**

Kornelije Rabuzin

*Faculty of Organization and Informatics, University of Zagreb, Croatia*

Martina Šestak

*Faculty of Organization and Informatics, University of Zagreb, Croatia*

**INTRODUCTION**

Nowadays, the increased amount and complexity of connected data stimulated by the appearance of social networks has shed a new light on the importance of managing such data, especially handling information about the connections. The most natural way of representing connected data is to represent them as nodes connected with relationships forming a graph. The concepts of graph theory have been used in many occasions to handle connected data in databases over the years.

The idea of storing data as a set of nodes and edges comprising a graph was implemented in various forms in data models used in the past. The network data model, developed in late 1960s, can be considered as the first data model, which most accurately incorporated this idea. However, it was not long before the relational data model appeared, and took over the entire database market for years, which it dominates even nowadays.

Even though several solutions have been introduced, which can be used to store graph-like data in available relational database management systems (RDBMSs), such as Oracle Graph or MariaDB OQGraph, NoSQL graph databases are considered to be the most efficient solution for storing such data, since the idea of storing highly connected data is their primary goal and purpose.

The possibilities of using graph databases in various application domains continue to grow. For instance, the most popular graph databases are Facebook Social Graph developed by Facebook to view connections between friends, or Amazon’s graph-based recommendation system (Amazon Neptune service). Sieger discusses the possibilities of introducing concepts of graph databases to modern businesses, and how graph databases could be used for Supply Chain Management or finding deeper connections between patients with similar diseases (Sieger, 2016). Moreover, according to D. Woods, due to their ability to efficiently gain insight into connections between pieces of information, graph databases play (and will play) an important role in transforming modern businesses to data-driven organizations, which make use of their data through the concept of Master Data Management (Woods, 2015).

Since their beginnings, there have been many graph database management systems (GDBMSs) available on the database market (e.g., Neo4j, TitanDB, AllegroGraph, FlockDB, InfiniteGraph, etc.). Recently, new emerging trends can be observed on the GDBMS market; aside from Neo4j, which is still the most popular and constantly developed GDBMS, other „native“ GDBMSs are starting to be replaced and „outgrown“ by multi-model databases, such as OrientDB, ArangoDB, etc.

The aforementioned trends and solutions indicate that it is worth exploring how graph-like data has been stored and manipulated over the years in various data storage solutions, and which current data storage options are available to store and manipulate such data.

Therefore, the objective of this article is to give a timeline overview of developed graph data storage solutions in order to gain insight into past, present and future trends of GDBMSs. Additionally, throughout this article, the most influential factors and reasons for changes in trends in GDBMSs' usage will be explored and analyzed.

**BACKGROUND**

In general, there are various definitions of a graph database; De Virgilio et al. defined graph database as a “multigraph g=(N,E), where every node is associated with a set of pairs <key, value>, and every edge is associated with a label” (De Virgilio, Maccioni, & Torlone, 2013), whereas He and Singh defined graph database as a set of graphs D={G1, G2, …, Gm}, where graph G is denoted by (V, E), V being a set of all vertices, and E being a set of all edges (He & Singh, 2006).

Graph database model can be defined as a data model, in which “data structures for the schema and instances are modeled as graphs or their generalizations, and data manipulation is expressed by graph-oriented operations and type constructors” (Angles & Gutierrez, 2008). Graph database model consists of three components (Angles & Gutierrez, 2008):

* structural component (graph data structures),
* operational component (graph-oriented operators), and
* integrity component (integrity constraints).

Nowadays, the most commonly used graph database model is the property graph data model (Figure 1), which can be defined as a “multigraph data structure, in which graph elements (vertices and edges) can have properties/attributes”(Ciglan, Averbuch, & Hluchy, 2012).

Data stored in a graph database can be queried by using a graph query language, which can provide support for different graph-related operations, such as graph union/intersection/difference, graph filtering, adjacency, path traversal, or pattern matching queries (Kaplan, Abdulla, Brugger, & Kohn, 2007).

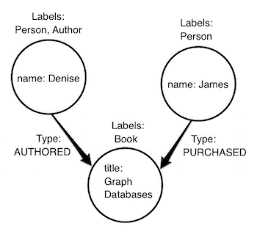


Figure 1. Sample property graph data model (Neo4j Inc., 2018)

Hence, a graph database management system (GDBMS) is “a database management system with CRUD (Create, Read, Update, Delete) operations based on property graph data model” (Saha, 2017).

**GRAPH DATABASE MANAGEMENT SYSTEMS DEVELOPMENT OVERVIEW**

To achieve the objective of this article, i.e., give an overview of past, current and future research trends in the field of graph databases and GDBMSs, a literature review has been conducted to identify relevant published research papers. The analyzed papers were then divided based on their research topic into three categories:

* papers discussing GDBMSs in the past (before 2009),
* papers and topics discussing GDBMs in the present (2009 – current), and
* papers and topics discussing potential future research trends related to GDBMSs.

**Graph Database Management Systems in the Past**

The story of storing both entities and their relationships in the database began with hierarchical and network database models, which appeared in 1960s. Between these two, the network database model can be considered as the first “complete” solution for storing connected data in the database at that time due to its ability to store complex and many-to-many relationships (as opposed to only one-to-many relationships available in the hierarchical database model), i.e., in the network database model, a child record can be connected to multiple parent records and vice versa (Maleković, Rabuzin, & Šestak, 2016). The network database model is represented as an inverted tree consisting of records connected with owner-member relationships, which can be implemented as pointers or indexes on the physical level (Harrington, 2002). The most popular network database model used at that time was the one developed in 1971 by CODASYL DBTG[[1]](#endnote-1) (sample model shown in Figure 2), which was then implemented in various DBMSs, and is still used nowadays (Hainaut, 2009).



Figure 2. Sample network data model (Khalil, 2018)

Even though network databases were soon “replaced” by relational databases, nowadays it is still possible to explore their characteristics through Raima Database Manager (RDM) solution developed by Raima Inc., which provides a query language, tools and other utilities to implement a network database model (Maleković et al., 2016).

**Graph Database Management Systems in the Present**

Research papers published from 2009 until now indicate that graph databases and GDBMSs face challenges common to other database technologies as well.

One of the most important challenges nowadays is storing the growing amount and size of data generated by different applications and businesses. Even though graph databases show good performance in handling such data, it is still necessary to explore how new and trending concepts, mechanisms and technologies can be used to improve their performance and stability in storing and retrieving necessary data in the shortest amount of time (e.g., data compression). The era of cloud solutions and distributed environment represents an additional challenge in scaling and partitioning graph databases without major performance losses.

On a conceptual level, the issue of graph database modeling and introducing some form of graph database schema (less strict than in relational databases) has proved to be of great importance to be standardized and constantly explored, since a correct and accurate graph database model can significantly influence the performance of future graph queries, changes in database structure, etc.

Another great issue quite popular in the research community is the lack of a standardized graph query language. Nowadays, it happens quite often that each modern GDBMS (e.g., Neo4j, TinkerGraph, InfiniteGraph) uses different approaches and query languages to query the underlying graph database, which can be seen in comparison of selected GDBMSs (AllegroGraph, JanusGraph, Neo4j and TinkerGraph) in Table 1. When it comes to graph query languages, there are two important aspects, which need to be considered, which make a query language effective and preferable among its users:

1. the language syntax must remain as simple and understandable as possible for different categories of users ranging from beginners to professionals and domain experts, and
2. the query formulation process must be simplified, but also ensure that the query performance is efficient, effective and stable.

Table 1. Comparison of selected graph DBMSs (Adapted from https://db-engines.com/en/)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **AllegroGraph** | **JanusGraph (former Titan)** | **Neo4j** | **TinkerGraph** |
| **Primary database model** | Graph DBMS  RDF store | Graph DBMS | Graph DBMS | Graph DBMS |
| **Data scheme** | yes (RDF schema) | yes | schema-free and schema-optional | schema-free |
| **Access methods** | RESTful HTTP API  SPARQL query language | Java API  Tinkerpop computing framework | Cypher query language  Java API  Neo4j Object Graph Mapping library  RESTful HTTP API | Tinkerpop 3 graph computing framework |
| **Transaction concepts** | ACID[[2]](#endnote-2) | ACID | ACID | no |
| **Foreign keys** | no | yes (implemented as relationships in graphs) | yes (implemented as relationships in graphs) | no |
| **License** | commercial | open-source | open-source | open-source |

The increasing popularity of graph databases (and NoSQL technologies in general) also presents a challenge to convert existing relational database solutions to graph databases to ensure an easy and issue-free transition between the two paradigms. Additionally, in their beginnings, the primary application domain of graph databases and GDBMSs were social networks, fraud detection and recommendation systems. Over the years, the achieved research progress and new GDBMS features have expanded this domain in order to keep up with current technology trends.

Finally, there is a growing tendency to use graph databases for analytical purposes (not only operational) as well, which has led to developing various systems, modules and tools for fast data analysis, as well as different algorithms and methods for data mining in graph databases. Furthermore, the support for implementing RDF (Resource Description Framework) data model in GDBMSs offers an opportunity to handling semantics of data stored in this graph database.

The aforementioned issues and challenges are continually being solved through novel GDBMS architectures, algorithms, frameworks, methods, etc., introduced in both theory and practice, which will be discussed in detail in the following section.

**SOLUTIONS AND RECOMMENDATIONS**

The topics of graph database modeling and graph query languages have been studied in several research papers. In (Angles et al., 2017), the authors discuss edge-labelled and property graphs as two popular graph data models, different graph querying approaches (graph patterns and navigational expressions), and outline the importance of formalization of graph query languages, such as Cypher, Gremlin or SPARQL. Graph database models and query languages are also discussed in (Soussi, Aufaure, & Baazaoui, 2011).

The process of converting existing relational database solutions to the context of graph databases has been addressed in (Roy-Hubara, Rokach, Shapira, & Shoval, 2017), in which the authors investigate how to create a graph database schema based on an entity-relationship diagram of the application domain. Also, in (Park, Shankar, Park, & Ghosh, 2014), the authors propose a “3NF equivalent graph” transformation to be used for migrating healthcare data from a relational to a graph database.

Modern GDBMSs, such as Neo4j, offer a wide range of features for storing and querying data from the underlying database. However, topics covered in the research community (e.g., graph data partitioning, novel querying approaches, algorithms for query optimization) are often covered only in theory, i.e., rarely implemented in GDBMSs. The most popular graph query language is Cypher, mostly due to its easy-to-learn syntax and pattern matching approach to query processing. According to many users, the biggest drawback for using graph databases is the lack of a standardized query language, which is currently being solved through a so-called “openCypher” project, i.e., an initiative for transforming Cypher language into a standardized graph query language.

Storing billions of nodes and edges is not possible without using some kind of compression techniques and methods. In (Sutrisna, Saleh, & Gozali, 2015), the authors use Graph Algorithm Clustering to compress the graph database, and to produce a lossless and compressed database**.**

Furthermore, the process of retrieving data from graph database can improve in various segments by using different approaches. S. Das et al. propose a query optimizer for graph databases in (Das, Goyal, & Chakravarthy, 2016). In (Castelltort & Laurent, 2016), the authors discuss how to use in-memory architectures to extract graph database summaries to speed up the graph database querying process. In (Spyropoulos, Vasilakopoulou, & Kotidis, 2016), the authors present “Digree, an experimental middleware system that can execute graph pattern matching queries over databases hosting voluminous graph datasets”. In (Zhang, Gao, Wu, Li, & Gao, 2011), the authors propose a new approach for processing supergraph queries, and a new algorithm for testing subgraph isomorphisms. Another interesting challenge is to find answers to queries over inconsistent graph databases, which is addressed by Barceló and Fontaine in (Barceló & Fontaine, 2017).

In (Yung & Chang, 2017), the authors explore reachability queries, and propose a “How-to-Reach” query to be used when a user does not know, or is unsure how to reach from source to the destination. Similar ideas are presented in (Vasilyeva, Thiele, Bornhövd, & Lehner, 2016), in which the authors propose “Why Empty?” and “Why So Many?” queries, which should explain why a result is empty, or contains too many rows. Queries returning empty answers are also discussed in (Vasilyeva, Thiele, Bornhoevd, & Lehner, 2014); the authors propose the use of top-k differential queries, which should deliver the missing parts of the query graph, so that users have insight into what was missing in the query, and why the query returned no result.

On low-level graph query processing, various algorithms and methods have been introduced to improve the subgraph isomorphism and similarity search processes, which can result in improved query performance (Chodpathumwan, Aleyasen, Termehchy, & Sun, 2016; Kiran & Sivadasan, 2015; Luo, Guan, & Zhou, 2011; Y. Yuan, Wang, Chen, & Wang, 2015).

In addition to query execution performance improvements, several approaches and frameworks have been introduced to simplify the query construction process for end users, such as query auto-completion (Yi, Choi, Bhowmick, & Xu, 2017) and graph exploration (Didimo, Giacche, & Montecchiani, 2015).

Additionally, indexing is always an interesting choice to consider when database performance needs to be improved. In (Azaouzi & Ben Romdhane, 2017), the authors propose the so-called Structural Graph Indexing, called GIRAS, “an indexing feature based on Rare subGraphs (RGs)”, which can be used to decrease the size of the candidate answer set when querying the database. In (D. Yuan & Mitra, 2013), the authors propose a graph index called Lindex, which indexes subgraphs contained within the graph database. In (Jarrar & Deik, 2015), the authors propose a Graph Signature Index suitable for indexing and querying large data graphs. In (Sakr & Al-Naymat, 2010), the authors present different techniques for indexing and querying graph databases.

Another way of improving graph database querying is to use data partitioning. However, graph data partitioning still represents a challenging issue, and only several research papers have been published, which address the issue in question (Barguñó, Muntés-Mulero, Dominguez-Sal, & Valduriez, 2011; Ben Ammar, 2016). Graph data partitioning also represents a basis for building distributed graph databases.

Integrity constraints in graph databases and graph database schema are explored by J. Pokorný et al. in (Pokorný, Valenta, & Kovačič, 2017), in which the authors discuss the level of integrity constraints and schema support in Neo4j GDBMS. In recent years, various graph query languages have been proposed in addition to Cypher, Gremlin and SPARQL graph query languages, which introduce improvements in graph database querying in different segments; a detailed overview was published by P. Wood in (Wood, 2012). For instance, in (Chavarria-Miranda, Castellana, Morari, Haglin, & Feo, 2016), the authors propose a new language called GraQL for attributed graph databases, whereas in (He & Singh, 2008), the authors propose GraphQL query language based on the concept of graph patterns and graphs as the basic unit of information. Additionally, transaction support in graph databases is discussed in (Koloniari & Pitoura, 2016).

An interesting change in GDBMS usage can be noticed in GDBMS market share. Figure 1 shows the ranking of currently top 10 mostly used GDBMSs and their underlying database model according to DB-Engines knowledge base. Based on the figure, it can be observed that among listed DBMSs only two of them are based “purely” on a graph database model (Neo4j and Giraph), whereas the rest of DBMSs support other database models as well (e.g., documents, key-value pairs, etc.). Therefore, “pure” GDBMSs are slowly being “overtaken” by multi-model DBMSs, which can store data in other database models in addition to graph database model as well.



Figure 3. Top 10 GDBMS ranking (DB-Engines, 2018)

Currently, many published research papers focus on expanding the applicability of graph database technology to various application domains, such as genetics and medicine, networking, tourism, power systems, etc., which proves that the technology will continue to spread, and that the popularity of graph databases will continually grow in years to come.

**FUTURE RESEARCH DIRECTIONS**

Even though the popularity of graph databases is constantly increasing, there are still topics and issues that need to be solved. Nowadays, the increasing amount of papers and researched topics indicate several research and development trends in the field of and GDBMSs and graph databases, such as semantic graph databases, graph mining, graph data warehouses and graph cubes (Liu & Vitolo, 2013), graph query language standardization initiatives, etc.

For instance, in (Kostylev, Reutter, & Vrgoč, 2016), the authors discuss how to use XPath query language over graphs, whereas in (Lian, Chen, & Wang, 2016), the authors discuss different methods for “quality-aware subgraph matching over inconsistent probabilistic RDF graphs (QA-gMatch)”, which goal is to retrieve subgraphs isomorphic to a given query graph from inconsistent probabilistic RDF graphs.

Additionally, there has been a lot of research focusing on using graph concepts and graph databases for data mining purposes, e.g., frequent subgraph mining (Wang et al., 2016), but also for object, image (Lampoltshammer & Wiegand, 2015) and handwriting recognition (Stauffer, Fischer, & Riesen, 2016).

According to (Green, 2018), a new initiative has been undertaken recently to develop an industry-standardized property graph query language called GQL (Graph Query Language). GQL will combine properties of PGQL (Property Graph Query Language), G-CORE (graph query language used among researchers) and the openCypher language mentioned in the previous section, and will be used across different GDBMSs (as its counterpart SQL[[3]](#endnote-3) in relational DBMSs).

As highlighted in the previous section, current industry trends show, besides Neo4j as the most popular “pure” GDBMS, which is constantly being improved and extended with new features, graph features can also be used in multi-model DBMSs, such as OrientDB or Cosmos DB. Furthermore, the existing relational DBMSs, such as Oracle and Microsoft SQL Server, are starting to extend their support for graph databases as well. This initiative will certainly present an important challenge for Neo4j and other GDBMSs in the future.

**Conclusion**

In this article, an overview of past, current and potential future research and industry trends in the field of GDBMSs and graph databases has been presented. The lack of a standardized graph query language has been identified as currently their major drawback, but research papers published in recent years, as well as current industry initiatives for developing a standardized graph query language (openCypher, GQL) indicate that this problem will soon be resolved. For years, the most widely used GDBMS has been Neo4j. However, recent trends show that other existing DBMSs, which do not support graph data model natively, are continually introducing new features for managing graph-like data (e.g., Oracle). Therefore, it can be presumed that, in the future, Neo4j will be continually developed as a “pure” GDBMS, while other DBMSs will work on partially supporting graph data model.

**REFERENCES**

Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., & Vrgoč, D. (2017). Foundations of Modern Query Languages for Graph Databases. *ACM Computing Surveys (CSUR)*, *50*(5), 68.

Angles, R., & Gutierrez, C. (2008). Survey of graph database models. *ACM Computing Surveys*, *40*(1), 1–39. http://doi.org/10.1145/1322432.1322433

Azaouzi, M., & Ben Romdhane, L. (2017). A Minimal Rare Substructures-Based Model for Graph Database Indexing. In P. Madureira, AM and Abraham, A and Gamboa, D and Novais (Ed.), *Intelligent Systems Design and Applications (ISDA 2016)* (Vol. 557, pp. 250–259). http://doi.org/10.1007/978-3-319-53480-0\_25

Barceló, P., & Fontaine, G. (2017). On the data complexity of consistent query answering over graph databases. *Journal of Computer and System Sciences*, *88*, 164–194.

Barguñó, L., Muntés-Mulero, V., Dominguez-Sal, D., & Valduriez, P. (2011). ParallelGDB: a parallel graph database based on cache specialization. In *Proceedings of the 15th Symposium on International Database Engineering & Applications* (pp. 162–169).

Ben Ammar, A. (2016). Graph Database Partitioning: A Study. In *2016 7th International Conference on Information, Intelligence, Systems & Applications (IISA)*.

Castelltort, A., & Laurent, A. (2016). Extracting fuzzy summaries from NoSQL graph databases. In *Advances in Intelligent Systems and Computing* (Vol. 400, pp. 189–200). http://doi.org/10.1007/978-3-319-26154-6\_15

Chavarria-Miranda, D., Castellana, V. G., Morari, A., Haglin, D., & Feo, J. (2016). GraQL: A Query Language for High-Performance Attributed Graph Databases. In *2016 IEEE 30th International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (pp. 1453–1462). http://doi.org/10.1109/IPDPSW.2016.216

Chodpathumwan, Y., Aleyasen, A., Termehchy, A., & Sun, Y. (2016). Towards Representation Independent Similarity Search Over Graph Databases. In *CIKM’16: Proceedings of the 2016 ACM Conference on Information and Knowledge Management* (pp. 2233–2238). http://doi.org/10.1145/2983323.2983673

Ciglan, M., Averbuch, A., & Hluchy, L. (2012). Benchmarking traversal operations over graph databases. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on* (pp. 186–189).

Das, S., Goyal, A., & Chakravarthy, S. (2016). Plan Before You Execute: A Cost-Based Query Optimizer for Attributed Graph Databases. In M. S. and H. T (Ed.), *Big Data Analytics and Knowledge Discovery, DAWAK 2016* (Vol. 9829, pp. 314–328). http://doi.org/10.1007/978-3-319-43946-4\_21

DB-Engines. (2018). DB-Engines Ranking of Graph DBMS. Retrieved June 27, 2018, from https://db-engines.com/en/ranking/graph+dbms

De Virgilio, R., Maccioni, A., & Torlone, R. (2013). Converting relational to graph databases. In *First International Workshop on Graph Data Management Experiences and Systems* (p. 1).

Didimo, W., Giacche, F., & Montecchiani, F. (2015). KOJAPH: Visual Definition and Exploration of Patterns in Graph Databases. In D. E. and L. A (Ed.), *Graph Drawing and Network Visualization, GD 2015* (Vol. 9411, pp. 272–278). http://doi.org/10.1007/978-3-319-27261-0\_23

Green, A. (2018). GQL - initiating an industry standard property graph query language. Retrieved June 30, 2018, from https://db-engines.com/en/blog\_post/78

Hainaut, J.-L. (2009). Network Data Model. In L. LIU & M. T. ÖZSU (Eds.), *Encyclopedia of Database Systems* (pp. 1900–1906). Boston, MA: Springer US. http://doi.org/10.1007/978-0-387-39940-9\_246

Harrington, J. L. (2002). *Relational Database Design Clearly Explained*. Elsevier Science. Retrieved from https://books.google.hr/books?id=HHmBfmJRSL4C

He, H., & Singh, A. K. (2006). Closure-tree: An index structure for graph queries. In *Data Engineering, 2006. ICDE’06. Proceedings of the 22nd International Conference on* (p. 38).

He, H., & Singh, A. K. (2008). Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 405–418).

Jarrar, M., & Deik, A. (2015). The Graph Signature: A Scalable Query Optimization Index for RDF Graph Databases Using Bisimulation and Trace Equivalence Summarization. *International Journal of Semantic Web and Information Systems*, *11*(2), 36–65. http://doi.org/10.4018/IJSWIS.2015040102

Kaplan, I. L., Abdulla, G. M., Brugger, S. T., & Kohn, S. R. (2007). *Implementing graph pattern queries on a relational database*.

Khalil, A. (2018). Database Concepts. Retrieved October 29, 2018, from http://www.cs.aucegypt.edu/~csci253/DBConcepts v2.htm

Kiran, P., & Sivadasan, N. (2015). Scalable Graph Similarity Search in Large Graph Databases. In *Proceedings of the 2015 IEEE Recent Advances in Intelligent Computational Systems (RAICS)* (pp. 207–211).

Koloniari, G., & Pitoura, E. (2016). Transaction Management for Cloud-Based Graph Databases (pp. 99–113). http://doi.org/10.1007/978-3-319-29919-8\_8

Kostylev, E. V, Reutter, J. L., & Vrgoč, D. (2016). Static Analysis of Navigational XPath over Graph Databases. *Information Processing Letters*, *116*(7), 467–474.

Lampoltshammer, T. J., & Wiegand, S. (2015). Improving the Computational Performance of Ontology-Based Classification Using Graph Databases. *Remote Sensing*, *7*(7), 9473–9491. http://doi.org/10.3390/rs70709473

Lian, X., Chen, L., & Wang, G. (2016). Quality-Aware Subgraph Matching over Inconsistent Probabilistic Graph Databases. *IEEE Transactions on Knowledge and Data Engineering*, *28*(6), 1560–1574. http://doi.org/10.1109/TKDE.2016.2518683

Liu, Y., & Vitolo, T. M. (2013). Graph Data Warehouse: Steps to Integrating Graph Databases into the Traditional Conceptual Structure of a Data Warehouse. In *2013 IEEE International Congress on Big Data* (pp. 433–434). http://doi.org/10.1109/BigData.Congress.2013.72

Luo, Y., Guan, J., & Zhou, S. (2011). Towards Efficient Subgraph Search in Cloud Computing Environments. In R. Xu, J and Yu, G and Zhou, S and Unland (Ed.), *Database Systems for Advanced Applications, DASFAA 2011* (Vol. 6637, pp. 2–13).

Maleković, M., Rabuzin, K., & Šestak, M. (2016). Graph Databases-are they really so new. *International Journal of Advances in Science Engineering and Technology*, *4*(4), 8.

Nabti, C., & Seba, H. (2016). Subgraph Isomorphism Search in Massive Graph Databases. In R. M. and W. G. and W. R. and M. V. and C. V (Ed.), *IOTBD: Proceedings of the International Conference on Internet of Things and Big Data* (pp. 204–213). http://doi.org/10.5220/0005875002040213

Neo4j Inc. (2018). Getting Started with Neo4j and Ruby. Retrieved October 29, 2018, from https://neo4j.com/developer/ruby-course/

Park, Y., Shankar, M., Park, B.-H., & Ghosh, J. (2014). Graph Databases for Large-Scale Healthcare Systems: A Framework for Efficient Data Management and Data Services. In *2014 IEEE 30th International Conference on Data Engineering Workshops (ICDEW)* (pp. 12–19).

Pokorný, J., Valenta, M., & Kovačič, J. (2017). Integrity constraints in graph databases. *Procedia Computer Science*, *109*(Supplement C), 975–981. http://doi.org/10.1016/j.procs.2017.05.456

Roy-Hubara, N., Rokach, L., Shapira, B., & Shoval, P. (2017). Modeling Graph Database Schema. *IT Professional*, (6), 34–43.

Saha, A. L. (2017). *A Comparative Analysis of Graph Vs Relational Database For Instructional Module Development System*. Arizona State University.

Sakr, S., & Al-Naymat, G. (2010). Efficient Relational Techniques for Processing Graph Queries. *Journal of Computer Science and Technology*, *25*(6), 1237–1255. http://doi.org/10.1007/s11390-010-1098-z

Sieger, R. (2016, April 12). Graph Databases - Moving Business to the Future. Retrieved October 29, 2018, from https://www.linkedin.com/pulse/graph-databases-moving-business-future-robert-sieger/

Soussi, R., Aufaure, M.-A., & Baazaoui, H. (2011). Graph Database for Collaborative Communities. In P. E (Ed.), *Community-Built Databases: Research and Development* (pp. 205–234). http://doi.org/10.1007/978-3-642-19047-6\_9

Spyropoulos, V., Vasilakopoulou, C., & Kotidis, Y. (2016). Digree: A Middleware for a Graph Databases Polystore. In J. J. and K. G. and L. L. and H. X. and A. R. and X. Y. and X. W. and S. A. and R. S. and U. L. and Y. P. and G. R. and S. T (Ed.), *2016 IEEE Internation Conference on Big Data (BIG DATA)* (pp. 2580–2589).

Stauffer, M., Fischer, A., & Riesen, K. (2016). A Novel Graph Database for Handwritten Word Images. In R. RoblesKelly, A and Loog, M and Biggio, B and Escolano, F and Wilson (Ed.), *Structural, Syntactic, and Statistical Pattern Recognition, S+SSPR 2016* (Vol. 10029, pp. 553–563). http://doi.org/10.1007/978-3-319-49055-7\_49

Sutrisna, G. B. A., Saleh, K. R. W., & Gozali, A. A. (2015). Implementation of GRAC Algorithm (Graph Algorithm Clustering) in Graph Database Compression. In *2015 3rd International Conference on Information and Communication Technology (ICoICT)* (pp. 391–395).

Vasilyeva, E., Thiele, M., Bornhoevd, C., & Lehner, W. (2014). Top-k Differential Queries in Graph Databases. In M. Manolopoulos, Y and Trajcevski, G and KonPopovska (Ed.), *Advances in Databases and Information Systems (ADBIS 2014)* (Vol. 8716, pp. 112–125).

Vasilyeva, E., Thiele, M., Bornhövd, C., & Lehner, W. (2016). Answering “Why Empty?” and “Why So Many?” queries in graph databases. *Journal of Computer and System Sciences*, *82*(1), 3–22. http://doi.org/10.1016/j.jcss.2015.06.007

Wang, K., Xie, X., Jin, H., Yuan, P., Lu, F., & Ke, X. (2016). Frequent Subgraph Mining in Graph Databases Based on MapReduce. In G. Wang, G and Han, Y and Perez (Ed.), *Advances in Services Computing* (Vol. 10065, pp. 464–476). http://doi.org/10.1007/978-3-319-49178-3\_35

Wood, P. T. (2012). Query languages for graph databases. *ACM SIGMOD Record*, *41*(1), 50–60.

Woods, D. (2015). How Graph Databases Supercharge Master Data Management. *Forbes*. Retrieved from https://www.forbes.com/sites/danwoods/2015/08/26/how-graph-databases-supercharge-master-data-management/#18bf1df025b8

Yi, P., Choi, B., Bhowmick, S. S., & Xu, J. (2017). AutoG: a visual query autocompletion framework for graph databases. *VLDB Journal*, *26*(3), 347–372. http://doi.org/10.1007/s00778-017-0454-9

Yuan, D., & Mitra, P. (2013). Lindex: a lattice-based index for graph databases. *VLDB Journal*, *22*(2), 229–252. http://doi.org/10.1007/s00778-012-0284-8

Yuan, Y., Wang, G., Chen, L., & Wang, H. (2015). Graph similarity search on large uncertain graph databases. *VLDB Journal*, *24*(2, SI), 271–296. http://doi.org/10.1007/s00778-014-0373-y

Yung, D., & Chang, S.-K. (2017). Answering How-to-Reach Query in Big Attributed Graph Databases. In *2017 3rd IEEE International Conference on Big Data Computing Service and Applications (IEEE BIGDATASERVICE 2017)* (pp. 141–148). http://doi.org/10.1109/BigDataService.2017.20

Zhang, S., Gao, X., Wu, W., Li, J., & Gao, H. (2011). Efficient algorithms for supergraph query processing on graph databases. *Journal of Combinatorial Optimization*, *21*(2), 159–191. http://doi.org/10.1007/s10878-009-9221-1

Zheng, W., Zou, L., Lian, X., Wang, D., & Zhao, D. (2015). Efficient Graph Similarity Search Over Large Graph Databases. *IEEE Transactions on Knowledge and Data Engineering*, *27*(4), 964–978. http://doi.org/10.1109/TKDE.2014.2349924

**ADDITIONAL READINGS**

Angles, R. (2012, April). A comparison of current graph database models. In Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on (pp. 171-177). IEEE.

Angles, R., Arenas, M., Barcelo, P., Boncz, P., Fletcher, G., Gutierrez, C., ... & van Rest, O. (2018, May). G-CORE: A core for future graph query languages. In Proceedings of the 2018 International Conference on Management of Data (pp. 1421-1432). ACM.

Beis, S., Papadopoulos, S., & Kompatsiaris, Y. (2015). Benchmarking Graph Databases on the Problem of Community Detection. In Bassiliades, N and Ivanovic, M and KonPopovska, M and Manolopoulos, Y and Palpanas, T and Trajcevski, G and Vakali, A (Ed.), New Trends in Database and Information Systems II (Vol. 312, pp. 3–14). https://doi.org/10.1007/978-3-319-10518-5\_1

Holzschuher, F., & Peinl, R. (2013, March). Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j. In Proceedings of the Joint EDBT/ICDT 2013 Workshops (pp. 195-204). ACM.

Macko, P., Margo, D., & Seltzer, M. (2013, June). Performance introspection of graph databases. In Proceedings of the 6th International Systems and Storage Conference (p. 18). ACM.

Pokorny, J. (2015). Graph Databases: Their Power and Limitations. In Saeed, K and Homenda, W (Ed.), Computer Information Systems and Industrial Management (Vol. 9339, pp. 58–69). https://doi.org/10.1007/978-3-319-24369-6\_5

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., & Wilkins, D. (2010). A Comparison of a Graph Database and a Relational Database. In Proceedings of the 48th Annual Southeast Regional Conference (ACM SE 10) (pp. 223–228).

Wang, S.-T., Jin, J., Rivett, P., & Kitazawa, A. (2015). Technical Survey Graph Databases and Applications. INTERNATIONAL Journal of Semantic Computing, 9(4), 523–545. https://doi.org/10.1142/S1793351X15500129

**KEY TERMS AND DEFINITIONS**

**“pure” Graph Database Management System:** A system, which supports CRUD (Create, Read, Update, Delete) operations on graph data represented in the graph database model.

**Graph database model:** A conceptual representation of graph-like data in the form of graph nodes and edges.

**Graph data partitioning:** The process of logically partitioning data stored in a graph database into segments, which can then be stored in different locations.

**Graph query processing:** The process of transforming a query written using a given graph query language syntax into a low-level query execution plan.

**Graph data warehouse**: A system used for data analysis, in which a data source is a graph database.

**Graph pattern matching:** The process of finding all subgraphs matching a given graph query pattern in the original graph.

**Graph mining:** The process of discovering patterns in data stored in a graph database through complex data analysis.

**ENDNOTES**

1. CODASYL DBTG - Conference on Data Systems Languages – the Data Base Task Group was responsible for developing DBMS specifications for implementing the network data model (Hainaut, 2009). [↑](#endnote-ref-1)
2. ACID (Atomicity, Consistency, Isolation, Durability) – transaction concept often present in relational databases, which aims to ensure database integrity. [↑](#endnote-ref-2)
3. SQL (Standard Query Language) – standardized query language for relational databases. [↑](#endnote-ref-3)