GRAPH DATABASE MANAGEMENT SYSTEMS AND GRAPH THEORY

Kornelije Rabuzin[1]
Sonja Ristić[2]
Robert Kudelić[3]

DOI: https://doi.org/10.31410/ITEMA.2020.39

Abstract: *In recent years, graph databases have become far more important. They have been proven to be an excellent choice for storing and managing large amounts of interconnected data. Since graph databases (GDB) rely on a graph data model based on graph theory, this study examines whether currently available graph database management systems support the principles of graph theory, and, if so, to what extent. We also show how these systems differ in terms of implementation and languages, and we also discuss which graph database management systems are used today and why.*

Keywords*: Neo4j, MS SQL server, Oracle, Cypher.*

INTRODUCTION

The importance of relational databases has been evident for many years. Transactional systems, CRM systems, ERP systems, and so on rely on relational databases. Last year, we witnessed the 50th year of relational databases. With a solid mathematical foundation and a background in set theory and logic, relational databases are not estimated to vanish, although some authors like to think so. However, in recent years, we have seen large amounts of data arriving from different sources. There are many V's describing the nature of data, and the term "Big Data" has been introduced and used extensively. The first V that one should consider is data volume. Although relational databases can store large amounts of data, PetaBytes (PB) or ExaBytes (EB) can cause problems for relational databases. It takes time to write and read the data from the database. The second V stands for variety since data are heterogeneous in nature. The third V denotes velocity, which means that data are being produced quickly, and they have to be processed rapidly. Sometimes there is not enough time to store the data and then later process the data; the data have to be processed immediately.

In order to deal with Big Data and its challenges, two solutions have been identified: NoSQL databases (including document-oriented, column-oriented, key value and graph databases) or distributed file systems, like the Hadoop framework. Graph databases belong to NoSQL databases, and they rely on a graph data model. In graph databases, data can be stored within the nodes and can have relationships that do connect the nodes. Many examples can be found and modelled using graph databases (computer networks, social network analysis, traffic, etc.). In this study, we present the different types of graphs that are known in graph theory, and we show how these graphs can be implemented in Neo4j. Then we show how MS SQL Server and

[1] University of Zagreb, Faculty of organization and informatics, Pavlinska 2, Varaždin, Croatia
[2] University of Novi Sad, Faculty of technical sciences, Trg D. Obradovića 6, Novi Sad, Serbia
[3] University of Zagreb, Faculty of organization and informatics, Pavlinska 2, Varaždin, Croatia

Oracle deal with the graph data model. Furthermore, we also discuss some major trends in GDBMS's global market.

The rest of this study is organized as follows: first we say a few words about graph databases, graph database management systems, and graphs in general. Then we show different types of implementations, including Neo4j, MS SQL Server, and Oracle. Then we discuss some open issues and the potential for future research directions. We also demonstrate a few improvements that we implemented in Neo4j. In the end, the conclusion is presented.

THE GRAPH DATABASE MANAGEMENT SYSTEMS (GDBMS) PERSPECTIVE

As far as we know in graph databases, we use the terms nodes and relationships (edges or loops in graph theory) that connect the nodes. The number of nodes and relationships can be vast. Since nodes are directly connected through their relationships, graph databases are extremely suitable for problem domains that contain large amounts of interconnected data. In order to create and use graph databases, we have to use a specific graph database management system. There are many purely graph database management systems available, including OrientDB, Neo4j, ArangoDB, etc. In this study, we use Neo4j since it is the most popular, and therefore, has had the widest impact, with support for many functional characteristics. However, the importance of having purely graph database management systems (GDBMS) has decreased over time. If we looked at the page https://db-engines.com/en/ranking, a few years ago, Neo4j was on a few occasions placed in the top 10 of systems. However, now it is ranked below that at 22nd place, and other graph database management systems are ranked even lower. Arango and OrientDB are at 59th and 71st place, respectively.

Table 1. Top 10 database systems (https://db-engines.com/en/ranking)

System	Primarily	Other supported models
Oracle	Relational	Document Store, Graph DBMS, RDF store
MySQL	Relational	Document Store
MS SQL Server	Relational	Document Store, Graph DBMS
PostgreSQL	Relational	Document Store
MongoDB	Document	Search engine
IBM DB2	Relational	Document Store, RDF store
Elasticsearch	Search engine	Document Store
Redis	Key-value	Document Store, Graph DBMS, Search engine, Time series DBMS
SQLite	Relational	
Cassandra	Wide column	

One major change is that all major DBMSs have turned multi-model. In Table 1, we see that many systems support the document model, and that three systems that are in the top 10 support the graph data model: Oracle, MS SQL Server, and Redis. Indeed, the only purely relational database management system in the top 10 is SQLite. It is lightweight and popular due to its size and the possibilities it affords that are extremely useful for smart phones, for example. Some other companies like Mozilla use SQLite for storing cache files, etc.

Since major DB vendors support the graph data model, it is important to keep in mind that in SQL Server, one can use SQL statements and constraints that most users are familiar with, and one can also use the graph database, as we demonstrate later on. Namely, in order to use Neo4j, one needs to learn Cypher or Gremlin, and there are some challenges regarding how to move

data to and from the graph database. So, this could be the reason why GDBMS are not that popular anymore. There were some attempts to build a single language or interface that would make it possible to use graph databases using already known query languages. For more information, look at (He & Singh, 2008) and (Holzschuher & Peinl, 2013). Many studies have been written to show how graph databases outperform relational databases, or to measure which graph database management system is better, such as (Chen et al., 2020) etc. If you are interested in graph databases in general, a good reference is (Robinson et al., 2013). A nice review of relational and graph databases is (Gupta et al., 2020) and (Maleković et al., 2016).

NEO4J

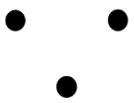
A good book on graphs and graph theory is (Wilson, 1996) And the definitions that are presented in the chapter below come from this book. We have also done some significant work in the field of graph theory, for example (Kudelić, 2016).

Generally speaking, in other studies, people usually talk about nodes and edges. However, there are other concepts presented in graph theory, and in this chapter, we show how they could be implemented in GDBMS Neo4j. "A simple graph G is a structure that consists of a non-empty finite set V(G) of elements called vertices, and a finite set E(G) of distinct unordered pairs of distinct elements of V(G) called edges" (Wilson, 1996). In this study we assume that simple graphs do not contain loops (i.e., an edge that joins a vertex to itself) and that there is at most one edge that connects a given pair of vertices (Wilson, 1996). If we allowed loops and multiple edges, then we would talk about graphs or multigraphs depending on the situation.

NULL GRAPH

A null graph's edge set is an empty set. Because of the fact that edges are not present, we can only see a vertex or more vertices that have no connections between them (Figure 1).

Figure 1. Null graph



We could have a graph database that contains nodes that have no relationships, but the importance of graph databases lies in relationships that connect the nodes and that do at the same time contain additional information. The following statement creates three "Person" nodes in an empty database and we check the database content (Figure 2):

```
CREATE (john:Person { firstname: "John", lastname: "Smith" }),
(mary:Person { firstname: "Mary", lastname: "Smith"}),
(jack:Person { firstname: "Jack", lastname: "Smith"})
MATCH (n:Person) RETURN n LIMIT 25
```

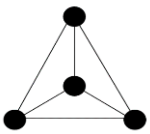
Figure 2. Creating nodes in Neo4j – null graph

We can see that null graphs can be implemented in Neo4j since we have three nodes that are not connected (Figure 2), but such graphs provide little information. As we said earlier, the connections between the nodes provide valuable information and this is where the strength of graph databases lies.

COMPLETE GRAPH

In order to explain complete graphs, several definitions are important (Wilson, 1996): "We say that two vertices v and w of a graph G are adjacent if there is an edge vw joining them, and the vertices v and w are then incident with such an edge. Similarly, two distinct edges e and f are adjacent if they have a vertex in common… A graph is connected if it cannot be expressed as the union of two graphs, and disconnected otherwise." In order to see whether complete graphs can be implemented, we should first recall the notion of adjacency. We should also take note that a complete graph has n(n-1)/2 edges. Basically, when an edge connects two vertices, they are adjacent. In a simple complete, graph this has to be true for "each pair of distinct vertices" (Wilson, 1996). Such graphs are denoted by Kn; K4 is presented in Figure 3.

Figure 3. K4

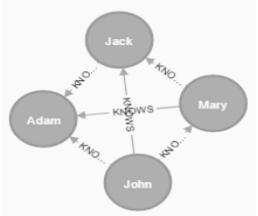


In the next example, we try to implement the complete graph in Neo4j. We would like to add one new person and to make a few relationships between the nodes.

```
MATCH (john:Person {firstname:"John"}), (mary:Person{firstname:"Mary"}),
(jack:Person {firstname:"Jack"})
CREATE (adam:Person { firstname: "Adam", lastname: "Smith"}),
(john)-[:KNOWS {since:2011}]->(mary), (john)-[:KNOWS {since:2012}]->(jack),
(john)-[:KNOWS {since:2014}]->(adam), (mary)-[:KNOWS {since:2012}]->(jack),
(mary)-[:KNOWS {since:2013}]->(adam), (jack)-[:KNOWS {since:2011}]->(adam)
```

When we look at the database state, we see that the nodes are connected and the relationships are directed, i.e. they have arrows (Figure 4):

Figure 4. Complete graph



This leads us to the concept of directed graphs. Informally, directed graphs (digraphs) contain vertices and arcs. However, it is not irrelevant which vertex comes first within an arc. In fact, arrows are used to indicate the ordering of vertices in the arc. Therefore, we are dealing with an ordered pair of vertices. This is supported in graph databases since the direction of a relationship can be specified when the relationship is created.

"The degree of a vertex v of G is the number of edges incident with v… A graph in which each vertex has the same degree is a regular graph" (Wilson, 1996). We can see that "complete graph Kn is regular of degree n-1" (Wilson, 1996). If we neglected the fact that relationships are directed, the graph above would be regular as well.

CYCLE GRAPHS

"A connected graph that is regular of degree 2 is a cycle graph. We denote the cycle graph on n vertices by Cn" (Wilson, 1996) (Figure 5).

Figure 5. Cycle graph

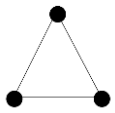


Figure 6. Path graph

If we remove one edge, we get a path graph (Figure 6). Social network analysis shows that if a and b are friends and b and c are friends, there exists a finite chance that a and c will be friends in the future. In social network analysis, some path graphs that include several persons are most likely to turn into connected graphs. Due to space limitations the example is omitted.

COMPLEMENT OF A SINGLE GRAPH

"If G is a simple graph with vertex set V(G), its complement $\bar{G}$ is the simple graph with vertex set V(G) in which two vertices are adjacent if and only if they are not adjacent in G" (Wilson, 1996). What are the implications for graph databases? If we had one instance of a database, it would be interesting to see which nodes are not connected. That way, we would be able to know who does not have friends, we would be able to know who does not ship products to certain countries, etc. The query below would simply find people, in the database created above, that are NOT connected (observe `NOT ((a)-[:KNOWS]-(b))`):

```
MATCH (a:Person), (b:Person)
WHERE a<>b AND NOT ((a)-[:KNOWS]-(b))
RETURN a, b;
```

GRAPH DATABASES IN MS SQL SERVER 2019

In the previous section, we used Neo4j to implement the concepts of graph theory. Here, we use MS SQL Server 2019. In MS SQL Server 2019, users can create one graph per database. A graph consists of an edge and node tables. According to https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-architecture?view=sql-server-ver15: "A node table is a collection of similar type of nodes. For example, a Person node table holds all the Person nodes belonging to a graph. Similarly, an edge table is a collection of similar type of edges. For example, a Friends edge table holds all the edges that connect a Person to another Person. Since nodes and edges are stored in tables, most of the operations supported on regular tables are supported on node or edge tables."

Let us look at one example and let us create one small graph using MS SQL Server 2019. We have several employees and we "manage" links that denote which employee is responsible for a certain another employee. First, we create the employee node table using SQL (observe the

"AS NODE" at the end of the CREATE statement) and then we create the "manages" table (observe "AS EDGE" at the end of the statement):

```
CREATE TABLE Employee (ID INTEGER PRIMARY KEY, Name VARCHAR(100), DEPT
VARCHAR(100)) AS NODE;
CREATE TABLE manages (SINCE date) AS EDGE;
```

Now, let's add a few employees:

```
INSERT INTO Employee VALUES (1, 'John', 'IT')
INSERT INTO Employee VALUES (2, 'Mark', 'IT')
INSERT INTO Employee VALUES (3, 'Jack', 'Finances')
```

Now let's specify one row for the "manages" table. Basically, we have to specify three values including $from_id, $to_id, and SINCE (we use subqueries to return the node id value for $from_id and $to_id columns):

```
INSERT INTO manages VALUES ((SELECT $node_id FROM Employee WHERE ID = 3),
       (SELECT $node_id FROM Employee WHERE ID = 1),'2020/01/01')
```

How do we find all of the employees who are managed by Jack?

```
SELECT e2.Name
FROM Employee e1, manages, Employee e2
WHERE MATCH (e1-(manages)-> e2) AND e1.name = 'Jack'
```

Here, we see that the SELECT statement is used to retrieve rows from the table. The main difference can be seen in the WHERE clause; the "MATCH (e1-(manages)-> e2)" part looks more like Cypher, than SQL.

GRAPH DATABASES IN ORACLE

In this section, we give a brief overview of Oracle's features and graph data model support. According to https://blogs.oracle.com/oraclespatial/graph-database-and-analytics-for-everyone, Oracle supports Property Graph database, PGX in-memory graph engine, PGQL graph query language, 50+ Graph algorithms, Support for graph visualization, SPARQL graph query language, Java APIs via open source Apache Jena, W3C standards support for semantic data, ontologies and inferencing, and RDF Graph views of relational tables. Due to space limitations we show one PGQL (property graph query language) example borrowed from https://pgql-lang.org/:

```
SELECT owner.name AS account_holder, SUM(t.amount) AS
total_transacted_with_Nikita
FROM MATCH (p:Person) -[:ownerOf]-> (account1:Account)
     , MATCH (account1) -[t:transaction]- (account2)
     , MATCH (account2:Account) <-[:ownerOf]- (owner:Person|Company)
WHERE p.name = 'Nikita'
GROUP BY owner
```

We see that the query looks like an SQL SELECT statement, almost like those found in SQL Server. However, in the FROM clause, we specify the nodes and the edges as well as their "connections." WHERE and GROUP BY clauses look like traditional SQL clauses.

DISCUSSION AND FUTURE RESEARCH

As is now obvious based on the discussion above, graph databases are interesting and important, but they do have some issues that are specific and one has to be aware of. In our previous research, we investigated some aspects of graph databases, like integrity constraint implementations (Rabuzin et al., 2016a) and (Rabuzin et al., 2016b). We have also implemented a visual interface (Gremlin By Example) that should make it easier for users to pose queries against the graph database (Rabuzin, Maleković, & Šestak, 2016). This interface is used to enable end users to pose queries against the Neo4j database in a manner similar to the way Query By Example is used to pose queries visually against MS Access. At this time, we are implementing a few other types of constraints given that many things that are supported in relational databases are still not supported for graph databases. One could also extend the research and include other GDBMSs and other query languages as well.

CONCLUSION

Graph databases are increasingly important. In this study, we had three goals: first we tried to investigate the importance of some of the other concepts that exist in graph theory, beyond those of edges and nodes. We tried to implement the presented concepts in Neo4j. Then we also investigated the graph database management systems that are popular and we used them to implement some examples. Finally, we also demonstrated how query languages look like in three different database systems. For that purpose, we have presented different types of graphs, including null graphs, directed graphs, cycle graphs, complete graphs, etc. We also investigated how these concepts could be implemented in Neo4j and described the repercussions of doing so. Then we showed how the graph data model is implemented in MS SQL Server 2019. The good thing is that one can use existing SQL-like statements to implement the graph database, but we do not have as many advanced features supported as are found in Neo4j. Oracle, on the other hand, supports more features than MS SQL Server and has more algorithms, but uses another query language (PGQL). Regarding the trends, we can see that existing purely graph database management systems have lost their popularity and many relational DBMS systems have turned multi-model. Finally, we can say that it will be interesting to see what the future will bring us.

REFERENCES

Chen, J., Song, Q., Zhao, C., & Li, Z. (2020). *Graph database and relational database performance comparison on a transportation network* doi:10.1007/978-981-15-6634-9_37

Gupta, S., Pal, S., & Chakraborty, M. (2020). *A study on various database models: Relational, graph, and hybrid databases* doi:10.1007/978-981-15-0361-0_11

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).

Holzschuher, F., & Peinl, R., (2013). *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).

Kudelić, R., (2016). *Monte-Carlo randomized algorithm for minimal feedback arc set problem*. Applied soft computing, 235 - 246. https://doi.org/10.1016/j.asoc.2015.12.018

Maleković, M., Rabuzin, K., & Šestak, M., (2016). *Graph Databases-are they really so new*. International Journal of Advances in Science Engineering and Technology. 4 (2016). Retrieved from http://bib.irb.hr/prikazi-rad?rad=843723

Rabuzin, K., Konecki, M., & Šestak, M., 2016a. Implementing CHECK Integrity Constraint in Graph Databases. *Proceedings of the 82nd IIER International Conference*. Retrieved from http://bib.irb.hr/prikazi-rad?rad=836861

Rabuzin, K., Maleković, M., & Šestak, M. (2016). Gremlin By Example. *International Conference on Advances in Big Data Analytics*, 144–149.

Rabuzin, K., Šestak, M., & Konecki, M., 2016b. Implementing UNIQUE Integrity Constraint in Graph Databases. *Multi-Conference on Computing in the Global Information Technology*. Retrieved from http://bib.irb.hr/prikazi-rad?rad=844504

Robinson, I., Webber, J., & Eifrem, E., 2013. *Graph Databases*. Information Management. https://doi.org/http://dx.doi.org/10.1016/B978-0-12-407192-6.00003-0

Wilson, J. R., 1996. *Graph Theory*. UK: Addison Wesley.