Proceedings of the 33rd European Modeling & Simulation Symposium (EMSS), 021
18th International Multidisciplinary Modeling & Simulation Multiconference

2724-0029 © 2022 The Authors.
doi: 10.46354/i3m.2022.emss.021

Towards Modelling Namespaces in Graph Databases

Andreas Pointner[1,*], Christoph Praschl[1] and Oliver Krauss[1]

[1]University of Applied Sciences Upper Austria, Hagenberg i. M., Austria

[*]Corresponding author. Email address: andreas.pointner@fh-hagenberg.at

Abstract

We present a novel approach to store data with different contexts inside a property graph model. We introduce namespaces, similar to namespaces in XML, and extend nodes and relationships with labels to assign them to a specific context, i.e. namespace. Individual properties of a node or relationship can also be put in a namespace. This work is specifically targeting the utilization in graph databases, with a reference implementation provided via the Neo4j database. In addition to the theoretical approach, an object to graph mapper for the programming language Java is implemented and used to evaluate the approach. As an evaluation example, a university organization is used, which is split into two domains. The experiments show, that information of different domains can be stored within the same model using namespaces. Thus, it is possible to reuse shared information over multiple contexts, which reduces data duplication in the graph database, as otherwise multiple nodes would be required.

Keywords: Model Extensions, Namespaces, Graph Database, Neo4j

1. Introduction

In this work, a combination of graph representations with the concept of domains and extensions is introduced, allowing to put (sub)graphs into specific contexts and providing views and query options on nodes and relationships in graphs. This work is specifically targeting the utilization in graph databases to enable these features and concepts in them.

Modern graph databases and graph APIs are based on data models defining nodes and relationships (or edges), consisting of properties with specific datatypes as discussed by Robinson (2015); Needham (2019); Hartig and Pérez (2018); Fernandes and Bernardino (2018). Some graph databases, like the *Ontotext* GraphDB introduced by de Leeuw et al. (2018), allow more rigorous modelling utilizing the *XML Resource Description Framework* (RDF) as defined in Klyne and Carroll (2004) and the *SPARQL Protocol and RDF Query Language* (SPARQL) as shown by Prud'hommeaux and Seaborne (2008). RDF enables rigorous modelling in graph databases by introducing XML namespaces and high level domain modelling. SPARQL enables querying by the namespace prefix defined in RDF. Although, these technologies allow using namespaces to put data into specific contexts, graph databases based on it often do not support putting nodes/relationships into multiple namespaces, but only one and for this reason limit the modelling possibilities.

This work introduces a method allowing to include data models in any graph database / modelling language with support for namespaces, such as RDF or other modelling options, thus enabling polymorphous nodes and relationships, as well as extensibility. Examples of where this approach can be useful are in the data mining domain, where these concepts allow enriching a base model in separate processing steps, or using views of the same data model for different purposes. Another application domain is the storage of complex data models in the health care domain, such as the Health Level 7 (HL7) Fast Healthcare Interoperability Resources (FHIR) standard as defined in HL7 International (2019).

The primary contributions presented in this work are:

- A concept to enrich a graph with namespaces, which allows grouping nodes and relationships into specific domains and extensions.
- A querying concept for the domain model.
- A proof of concept in the Neo4J graph database and its query language Cypher.
- A polymorphic object to graph mapping for the presented concept.

2. Background

An approach of namespaces in graph databases is presented based on a reference implementation using the graph database system (GDBS) Neo4j (2020b) and its query language Cypher, which is defined in Neo4j (2020a). Neo4j is a GDBS based on the labeled property graph data model defined by Anikin et al. (2019) and is implemented using the programming language Java. In this model, a graph is represented using a list of nodes and a list of relationships. Nodes are in turn the information entity saved in the database and take in a comparable role to relations in classic relational databases. Every node consists of an ID, its properties in the form of a list of key-value pairs and a list of labels. Labels are an additional meta information in the form of a set of strings, that can for example be used to represent a node's types. Relationships connect two nodes in a directed way and are semantically enriched. This means that Neo4j stores additional information for every relationship in the form of a label for its name/type and a list of key-value pairs to enhance the expressiveness of the relationship.

Cypher is used to manipulate the graph by inserting, deleting or updating nodes/relationships and by traversing the graph structure in a declarative way (comparable to SQL). Cypher also allows defining conditions based on the nodes' and relationships' properties and labels for the traversal of the graph.

3. Approach

In general, a graph G consists of a set of nodes V and a set of edges E, which results in an ordered pair (V, E). Thus, a graph can be defined as $G = (V, E)$. Furthermore, the condition that an edge E must have a source node and a target node applies. Thereby, the two functions $s, t : E \to V$ can be derived from it, as Bender and Williamson (2010) showed.

In this work, the definition of a namespace N is added, where every $n \in N$ is a tuple $n = (name, pred) \mid pred \in N \vee pred \in \{\epsilon\}$ containing the current namespace's name *name* and its parent namespace *pred* – respectively the empty (default) namespace if $pred \in \{\epsilon\}$ applies. This namespace definition leads to a namespace hierarchy, with the default namespace at the very beginning.

Based on this, the node definition is extended, so V is a set of nodes, where every node $v \in V$ is a tuple $v = (I, N_v, A) \mid N_v \subseteq N$. Such a tuple consists of an ID I, a set of namespaces N_v and a set of attributes A (properties) where every $a \in A$ is again a tuple $a = (n, val, type) \mid n \in N_v$ that defines the origin namespace n of the attribute, its actual value *val* and its data type *type*. With this definition, the property graph model is extended, so that every node is part of one or multiple namespaces. Finally, edges are assigned to namespaces as well, by adding a function $ns : E \to N$, that returns the namespaces to which the edge is assigned to.

Nodes in the namespace x define the subgraph $G' = (V', E') \mid \forall (I', N'_v, A') \in V' \wedge contains(x, N'_v) \wedge \forall e \in E'\ contains(x, ns(e))$ where *contains* is defined as followed: $contains(x, N) \Rightarrow N = \{\epsilon\} \vee x \in N \vee (x = (name_x, pred_x) \wedge pred_x \neq \epsilon \wedge contains(pred_x, N))$. This allows to define that a node is part of the subgraph G' if it is in the given namespace x or in any parent namespace along the hierarchy. The approach of subgraphs allows to query the model for different contexts by selecting nodes, edges and fields inside these elements only for specific namespaces.

4. Evaluation

A model of a university organization is used to evaluate the concept of namespaces in graph databases. The domain model is shown using the Enhanced Entity Relationship (EER) model as defined by Teorey et al. (1986) with Chen Notation as shown in Chen (1976) in Figure 1. EER is a visual notation for domain models and uses rectangles to describe entities (domain classes), attributes via ellipses, rhombus for relationships between entities and hexagons for the concept of generalization/specialization (inheritance).

The evaluation model is mainly based on persons in different roles. Every person is described by the main attributes `ID`, `FirstName` and `LastName`. In the example, a person can be a `Student` and/or an `Employee`, while employees are in turn separated into `Researchers`, `Tutors` or `Lecturers`. All of these roles consist of additional information like attributes as the `Semester` of a student or the `AreaOfExpertise` of a researcher, but also describe additional relationships as `teaches` between the entities `Student`, `Lecturer` and `Lecture`. The different components of the model can be separated into two organizational units: (I) Research and Development (R&D), as well as (II) Study Operation. Those units can be seen as different namespaces, with `Study Operation` as base namespace and `R&D` as extension. Those namespaces are shown with dashed bordered areas in Figure 1. In addition, the model highly depends on polymorphism, since e.g. a person may be a student and a tutor in the system at the same time. This information can be in turn represented via namespaces per role of a person and allows combining the person, the student and the tutor information in one single node in the database and like that allows minimizing storage space. With the presented concept it is still possible to extract

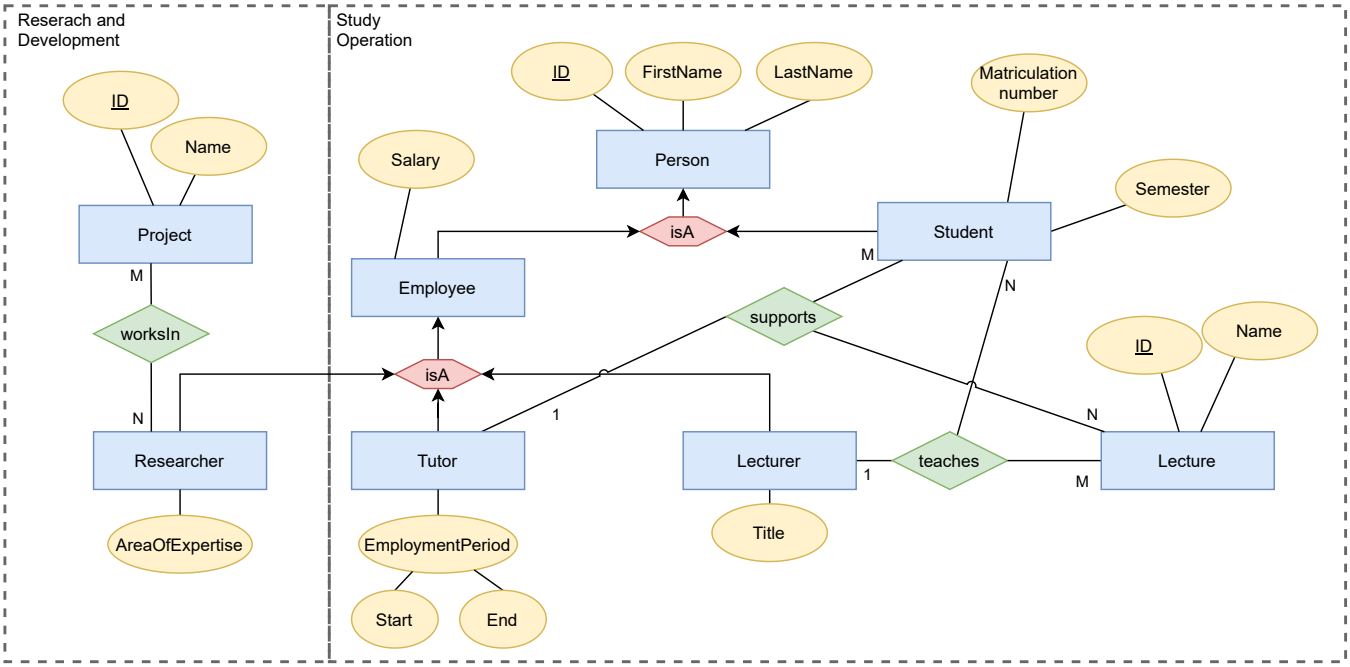

Figure 1. Enhanced Entity Relationship model showing different components of a university organization that is separated into two namespaces "Research and Development" and "Study Operation". Due to the support of polymorphism, a student may also be a tutor, or a lecturer may also work as a researcher.

only the required parts, when e.g. only the R&D related information or all tutors using the respective namespace information is desired.

The concept is tested based on a Java based object to graph mapper (OGM) implementation, that supports namespaces within a Neo4j graph database and with this to persist the presented university model with all its semantic information. This OGM prototype is open sourced by the authors Pointner et al. (2021) and publicly available on GitHub as well as on Maven Central. The implementation supports the concept of namespaces by utilizing labels in Neo4j's property graph model. This means, that a node or an edge can have multiple labels, that represent the various types which they belong to. In addition, the keys of each property within a node or edge are prefixed with the name of the namespace, allowing it to be mapped correctly. This also allows having properties of the same name with different purposes within the same element, but in a different namespace, and like this avoids clashing extensions. By default, the implementation maps the namespace to the fully qualified class name, which consists of the package name and the class name itself. This configuration can be overwritten by specifying custom annotations. The implementation uses Neo4j (2022) Java driver and allows querying the nodes and edges using Cypher. Using Java's reflection mechanism, the proposed implementation allows creating the correct objects at runtime and mapping the properties accordingly. To ensure that data is not lost when writing the object back to the database, it is required to have the value of all fields in all namespaces. To do so, all information that is present for a single element is stored in hidden fields in the object class, created via Java reflection. This only applies to properties and not to a node's relationships, which are only loaded if they are within the query results. Java does not support polymorphism. For this reason, it is not possible to have the same object in different namespaces. Because of this, multiple objects have to be used to access the specific information stored in the hidden fields. This requires a unique key to re-identify objects within different namespaces.

In order to be able to insert the nodes into the database, two parts are required within the object to graph mapping process: (1) the domain model in Java and (2) a repository, allowing to access the database. Both of these have been created for the whole domain model in order to test the defined scenario. A sample domain class is shown in Listing 1 and the creation of the corresponding repository is displayed in Listing 2.

Listing 1. Defining the person domain class and define the relevant annotation in order to be able to use it inside the object to graph mapper with namespaces.

```
package s;
@NodeEntity(label = "person")
public class Person {
  @Id
  private long id;
  private String firstName;
  private String lastName;
  /* Getter and setter omitted */
}
```

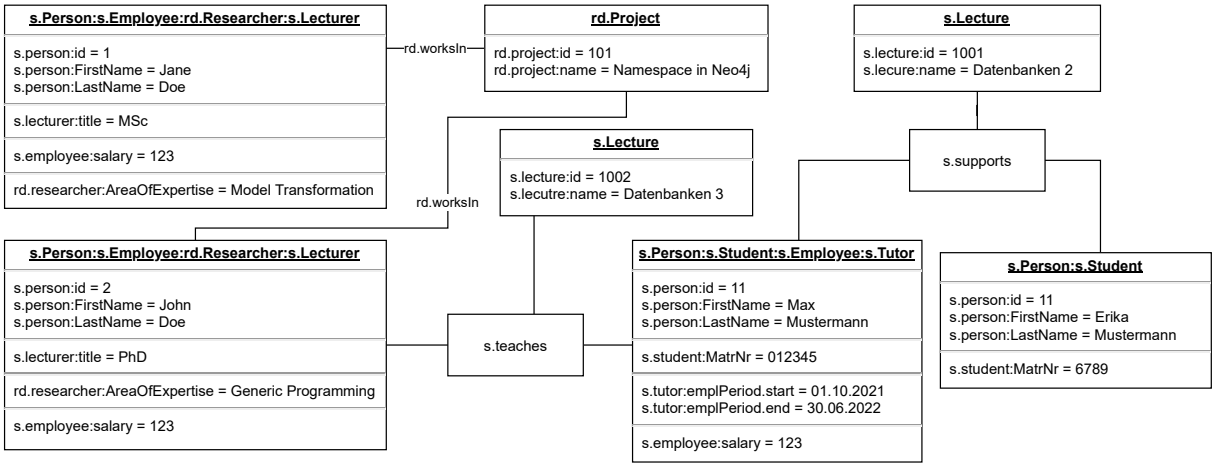

Figure 2. Graph representing the results of the evaluation example, consisting of vertices and edges with namespaces inside the Neo4j database. The namespaces are represented in the header of the object with the labels using the abbreviations "rd" for "Research and Development" and "s" for "Study Operation". In addition, every field has the namespace in prefix notation. The horizontal dividers are also indicating the namespaces.

Listing 2. Creates a new Neo4j repository for the Person class, which allows basic create, update and query operations on the database nodes.

```
/* Creation of transaction manager omitted */
TransactionManager tm = ...;
Neo4jRepository<Person, Long> persRepo = new
    ↪ NamespaceAwareReflectiveNeo4JNode
    ↪ RepositoryImpl(tm, Person.class);
```

Using the required repository objects, allows inserting a sample graph representing the domain model shown in Figure 1. This insert process results in an object graph that is in turn shown in Figure 2. Since Neo4j does not support hypergraphs it is necessary to create an intermediate node between ternary relationship, like "teaches" and "supports".

Next to inserting the data into the database, the OGM implementation enables to load different perspectives of the graphs using namespaces. As an example, a subgraph for the namespace "Study Operation" can be selected based on its entities *Lecture*, *Student* and *Lecturer* using the condition that the *Lecturer* with the name John Doe must be contained. To do so, the results are limited based on the main namespace "Study Operation" and the mentioned sub-namespaces *Lecture*, *Student* and *Lecturer* respectively. This query results in a Java object graph, which is visualized in Figure 3. This graph doesn't contain any explicit namespace information, as they are implicitly given via the Java domain class hierarchy and its packages.

In a second example, the namespace "Research and Development" is selected, and the entities are limited to *Project* as well as *Researcher*. Again, the condition that the *Researcher* with the name John Doe must be contained is used, other than that no further restrictions are applied. This results in the Java object graph show in Figure 4.

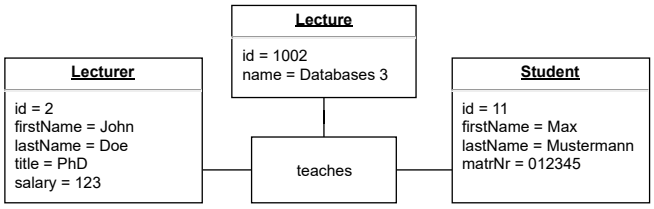

Figure 3. A subgraph when selecting the *Lecturer John Doe* with the *Lecture Database 3* and the *Student Max Mustermann* represented as Java objects.

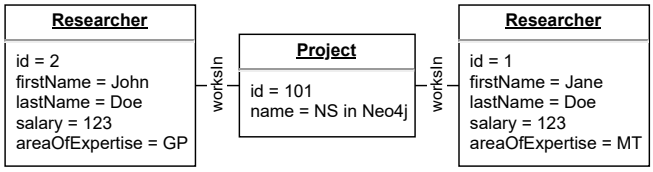

Figure 4. A subgraph when selecting the *Researcher John Doe* with all its dependencies inside the "Research and Development" represented as Java objects.

5. Related Work

The Extensible Markup Language (XML) as defined by Bray et al. (1997) provides with Bray et al. (1999) a namespace mechanisms, to avoid clashing element names between different documents or even within the same document and to clearly distinguish different element types. Namespaces are either defined via the `xmlns` attribute of a given element or via a prefix of the element name. Using such namespaces one is able to use polymorphism for an element, since every attribute is clearly assignable to its source class based on the given namespace and like this it is also possible to just retrieve the information that is associated with a given namespace. Similarly to this, the presented approach also uses a prefix to clearly identify objects and attributes.

The Resource Description Framework (RDF) as defined by Klyne and Carroll (2004) is an XML based information definition language. The language's basis are so-called semantic triples, that consists of a subject, a predicate and an object, where the subject is somehow related with the object via the predicate. Those semantic triples form a directed graph - the so called RDF model. Building on RDF, the Web Ontology Language (OWL) standard as defined by McGuinness et al. (2004) is used to describe ontologies. This is done by defining class hierarchies and object instances, as well as properties, that are added to a class using semantic triples. Due to this mechanism it is possible to define different properties along the class hierarchy and for this restricting the scope, in which attributes are available. Since RDF and in turn OWL are XML based, it is also possible to use namespaces for classes. This usage of namespaces in a graph based context and the limitation of properties to certain classes of a given namespace is comparable to the presented approach, that takes up this concept and brings it onto the level of graph databases.

Object/relational mapping (ORM) as defined by O'Neil (2008) has become an important concept in the field of object-orientated systems and represents an interface between object-orientated programming and relational databases. It is used to simplify the process of persisting and querying objects to/from a database, and with this to distribute information. Related to this, object to graph mappers (OGM) as shown in Dietze et al. (2016) are the equivalent for graph databases. The presented approach is tightly connected to OGM, since the presented approach uses this mechanism to map objects with their namespaces into graph databases.

In addition to OWL, there is also the SPARQL Protocol And RDF Query Language (SPARQL) as defined by Prud'hommeaux and Seaborne (2008) in the context of RDF. SPARQL is a query language used to retrieve and manipulate data stored as RDF files. Since OWL is based on RDF, SPARQL can be used to query ontologies and for these queries can filter information based on XML namespaces. Due to the usage of an ORM system, this mechanism of extracting certain data based on a give namespace is also possible with the presented approach.

Zhuge and Garcia-Molina (1998) present the concept of views for graph databases. This concept allows to dynamically provide certain excerpts of a database using a predefined query. Alternatively, so called Materialized Views as defined by Ross et al. (1996) extend the classic view concept by persisting the query result for later reuse to improve the performance. Consequently, views can be used as a mechanism for namespaces, where only associated attributes of a relation are returned and are for this comparable to the presented approach. Views require a manual selection of the attributes that belong to the namespace, while, in contrast our approach is able to automatically filter the required information based on the introduced namespace. Although the concept of views is widely used in the field of relational databases (since it is part of SQL as shown in for Standardization (2016)), it is not that common for graph databases. While e.g. ArangoDB defined by ArangoDB (2020) or OrientDB defined by OrientDB (2020) provide the possibility of views, the concept is not part of systems like Neo4j as defined by Robinson (2015), InfiniteGraph as defined by van der Lans (2010) or AllegroGraph as defined by Aasman (2006). In our work, an approach for namespaces in graph databases, but the concept itself can be also applied on different database systems.

Mennicke (2019) extends the Schema Graph as defined by Buneman et al. (1997) data model by concepts like key properties and modal specifications. In addition to that, Boyd and McBrien (2005), as well as HyperGraphDB as defined by Angles (2012) extend the mathematical model of Hypergraphs as defined in Bretto (2013) by constraint constructs, respectively by edges between more than two nodes. Comparable to these extensions, our work also introduces the concept of namespaces as extension to the property graph model used in graph databases like Neo4j. Furthermore, our namespace approach allows extending any model based on the property graph model by certain namespace-based groupings of information.

6. Conclusion

An approach is presented, on how to set (sub)graphs into a specific context, and the possibility to provide different views and query mechanisms. Namespaces are used in order to be able to extend domain objects for multiple different contexts.

In order to prove the correctness of our theoretical approach, a Java object graph mapper for the graph database Neo4j has been implemented and proved the concept with storing example objects of a university organization inside the database. A sub-graph is selected from the database and mapped that to the desired classes inside the Java model. With that, it is shown that the theoretical approach of using namespaces to store domain objects of different context within the same database works as desired. While, only the application in Java is shown, the approach can also be used in other programming languages.

For the future, we plan to evaluate our approach on a complex data model, namely the HL7 FHIR standard. The use-case in that scenario is based on a real world requirement. Each country, and often organizations, all have their own specifications based on HL7 FHIR. This is necessary because health care systems around the world have different requirements that have to be satisfied within the systems. With the presented approach, the shared data could be reused, and the models would be exchangeable between different systems.

In addition, a comparison of the presented object graph mapping approach to existing Neo4j object graph mappers like Spring Data Neo4j is planned. Furthermore, an evaluation of how the approach scales and performs in comparison to existing state-of-the-art frameworks should be carried out.

References

Aasman, J. (2006). Allegro graph: Rdf triple database. *Cidade: Oakland Franz Incorporated*, 17.

Angles, R. (2012). A comparison of current graph database models. In *2012 IEEE 28th International Conference on Data Engineering Workshops*, pages 171–177.

Anikin, D., Borisenko, O., and Nedumov, Y. (2019). Labeled property graphs: Sql or nosql? In *2019 Ivannikov Memorial Workshop (IVMEM)*, pages 7–13. IEEE.

ArangoDB (2020). Views | HTTP | ArangoDB Documentation. https://www.arangodb.com/docs/stable/http/views.html; accessed 10. May 2022.

Bender, E. A. and Williamson, S. G. (2010). *Lists, Decisions and Graphs*. S. Gill Williamson.

Boyd, M. and McBrien, P. (2005). Comparing and transforming between data models via an intermediate hypergraph data model. In Spaccapietra, S., editor, *Journal on Data Semantics IV*, pages 69–109, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bray, T., Hollander, D., Layman, A., and Tobin, R. (1999). Namespaces in xml. *World Wide Web Consortium Recommendation REC-xml-names-19990114. http://www. w3. org/TR/1999/REC-xml-names-19990114*.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (1997). Extensible markup language (xml). *World Wide Web Journal*, 2(4):27–66.

Bretto, A. (2013). Hypergraph theory. *An introduction. Mathematical Engineering. Cham: Springer*.

Buneman, P., Davidson, S., Fernandez, M., and Suciu, D. (1997). Adding structure to unstructured data. In Afrati, F. and Kolaitis, P., editors, *Database Theory — ICDT '97*, pages 336–350, Berlin, Heidelberg. Springer Berlin Heidelberg.

Chen, P. P. S. (1976). The entity relationship model toward a unified view of data. *ACM transactions on database systems (TODS)*, 1(1):9–36.

de Leeuw, D., Bryant, M., Frankl, M., Nikolova, I., and Alexiev, V. (2018). Digital methods in holocaust studies: The european holocaust research infrastructure. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 58–66.

Dietze, F., Karoff, J., Calero Valdez, A., Ziefle, M., Greven, C., and Schroeder, U. (2016). An open-source object-graph-mapping framework for neo4j and scala: Renesca. In *International Conference on Availability, Reliability, and Security*, pages 204–218. Springer.

Fernandes, D. and Bernardino, J. (2018). Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb. In *Proceedings of the 7th International Conference on Data Science, Technology and Applications*, DATA 2018, page 373–380, Setubal, PRT. SCITEPRESS.

for Standardization, I. O. (2016). Information technology database languages sql part 2: Foundation (sql foundation). Technical Report ISO/IEC 9075-2:2016(en), International Organization for Standardization.

Hartig, O. and Pérez, J. (2018). Semantics and complexity of GraphQL. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*. ACM Press.

HL7 International (2019). Profiling - FHIR v4.0.1. https://www.hl7.org/fhir/profiling.html; accessed 10. May 2022.

Klyne, G. and Carroll, J. J. (2004). Resource description framework (rdf): Concepts and abstract syntax. W3C Recommendation. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/; accessed 10. May 2022.

McGuinness, D. L., Van Harmelen, F., et al. (2004). Owl web ontology language overview. *W3C recommendation*, 10(10):2004.

Mennicke, S. (2019). Modal schema graphs for graph databases. In Laender, A. H. F., Pernici, B., Lim, E.-P., and de Oliveira, J. P. M., editors, *Conceptual Modeling*, pages 498–512, Cham. Springer Intern. Publishing.

Needham, M. (2019). *Graph algorithms : practical examples in Apache Spark and Neo4j*. O'Reilly, Beijing.

Neo4j (2020a). Cypher Query Language. https://neo4j.com/developer/cypher/; accessed 10. May 2022.

Neo4j (2020b). What is a Graph Database? https://neo4j.com/developer/graph-database/; accessed 10. May 2022.

Neo4j (2022). Neo4j driver. https://neo4j.com/docs/api/java-driver/current/org/neo4j/driver/Driver.html; accessed 10. May 2022.

O'Neil, E. J. (2008). Object/relational mapping 2008: Hibernate and the entity data model (edm). In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1351–1356, New York, NY, USA. Association for Computing Machinery.

OrientDB (2020). Create view - orientdb manual. http://orientdb.com/docs/last/sql/SQL-Create-View.html; accessed 10. May 2022.

Pointner, A., Praschl, C., and Krauss, O. (2021). Fhooeaist/aist-neo4j.

Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL Query Language for RDF. W3C Recommendation. http://www.w3.org/TR/rdf-sparql-query/; accessed 10. May 2022.

Robinson, I. (2015). *Graph databases : new opportunities for connected data*. O'Reilly, Sebastopol, CA.

Ross, K. A., Srivastava, D., and Sudarshan, S. (1996). Materialized view maintenance and integrity constraint checking: Trading space for time. *SIGMOD Rec.*, 25(2):447 to 458.

Teorey, T. J., Yang, D., and Fry, J. P. (1986). A logical design methodology for relational databases using the extended entity-relationship model. *ACM Comput. Surv.*, 18(2):197–222.

van der Lans, R. F. (2010). Infinitegraph: Extending business, social and government intelligence with graph analytics. Technical report, Technical report.

Zhuge, Y. and Garcia-Molina, H. (1998). Graph structured views and their incremental maintenance. In *Proceedings 14th International Conference on Data Engineering*, pages 116–125. IEEE.