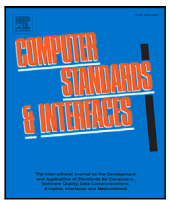
Computer Standards & Interfaces 88 (2024) 103801

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi



An ontology-based secure design framework for graph-based databases

Manuel Paneque [a,1,*], María del Mar Roldán-García [a,1], Carlos Blanco [b,1], Alejandro Maté [d,1], David G. Rosado [c,1], Juan Trujillo [d,1]

[a] ITIS Software, Department of Computer Science and Programming Languages, University of Málaga, Spain
[b] ISTR Research Group, Department of Computer Science and Electronics, University of Cantabria, Spain
[c] GSyA Research Group, Department of Information Technologies and Systems, University of Castilla-La Mancha, Spain
[d] Lucentia Research Group, Department of Software and Computing Systems, University of Alicante, Spain

ARTICLE INFO

Keywords:
Ontology
Security
Reasoning
Knowledge extraction
Healthcare

ABSTRACT

Graph-based databases are concerned with performance and flexibility. Most of the existing approaches used to design secure NoSQL databases are limited to the final implementation stage, and do not involve the design of security and access control issues at higher abstraction levels. Ensuring security and access control for Graph-based databases is difficult, as each approach differs significantly depending on the technology employed. In this paper, we propose the first technology-ascetic framework with which to design secure Graph-based databases. Our proposal raises the abstraction level by using ontologies to simultaneously model database and security requirements together. This is supported by the TITAN framework, which facilitates the way in which both aspects are dealt with. The great advantages of our approach are, therefore, that it: allows database designers to focus on the simultaneous protection of security and data while ignoring the implementation details; facilitates the secure design and rapid migration of security rules by deriving specific security measures for each underlying technology, and enables database designers to employ ontology reasoning in order to verify whether the security rules are consistent. We show the applicability of our proposal by applying it to a case study based on a hospital data access control.

1. Introduction

NoSQL databases have become the cornerstone of multiple business processes thanks to their focus on performance and high throughput. From Social Networks to Fraud Detection, NoSQL databases enable the scalability of applications to large volumes of data which would otherwise be impossible to exploit using relational technologies. However, this performance has unfortunately been achieved at the cost of other characteristics, as a result of which security and privacy have been relegated to a secondary place [1,2], despite the potential losses of economic and reputation resulting from information leaks.

Some of the different types of NoSQL databases are: (i) Key-Value stores, such as DynamoDB or Redis, in which data is accessed using unique keys; (ii) Columnar stores, such as Cassandra or HBase, in which keys are composed of combinations of columns, rows, and timestamps; (iii) Document-oriented databases, such as MongoDB or CouchDB, in which information is stored as documents in YAML (Yet Another Markup Language) or JSON (JavaScript Object Notation) formats, and (iv) Graph-based databases, such as Neo4J or GraphBase, which are widely known to perform better than the other three mentioned previously when accessing millions of pieces of data (nodes). To further complicate the secure design of NoSQL databases, even technologies that share the same NoSQL database type and model have differences as regards their implementation and the few security mechanisms that they provide.

In the last decade, industry and academia agreed that security should be incorporated from the earliest development stages by following what has been denominated as security by design [3], which combines principles from security and software engineering. This philosophy conflicts with the current state of NoSQL database design. To the best of our knowledge, there is currently no well-known systematic methodology or framework that supports the NoSQL database designer in producing a secure design at higher abstraction levels. This signifies that the database designer has to not only maintain in-depth knowledge of the underlying and specific database technology used, but also learn the concrete security mechanisms needed in order to properly implement all the security and access control issues required from the earliest stages.

* Corresponding author.
E-mail addresses: mpaneque@uma.es (M. Paneque), mrgarcia@uma.es (M.d.M. Roldán-García), carlos.blanco@unican.es (C. Blanco), amate@dlsi.ua.es (A. Maté), David.GRosado@uclm.es (D.G. Rosado), jtrujillo@dlsi.ua.es (J. Trujillo).
[1] The authors contributed equally to this manuscript.

https://doi.org/10.1016/j.csi.2023.103801
Received 1 February 2023; Received in revised form 9 July 2023; Accepted 15 October 2023
Available online 20 October 2023

In order to tackle this problem, in this paper, we propose the first NoSQL database design framework for NoSQL database types. Our proposal allows the extension of NoSQL database concepts with more specific ontologies, which capture the idiosyncrasy of each database type. These extensions in turn allow us to provide automatic derivation capabilities, thus facilitating the implementation of secure database designs in a rapid and error-free manner. More specifically, in this paper, we focus on the ontology for Graph-based databases and show the complete process from design to implementation. It is worth noting that covering all datastores requires the development of specific ontologies, each covering the particularities of each kind of datastore, and this is, therefore, not within the scope of this paper. In order to show the wide scalability of our approach we emphasize the implementation-independent property of our proposal throughout the paper. Furthermore, we also show the applicability of our approach by developing a complete case study regarding Graph databases, including model checks and transformation rules. To the best of our knowledge, this is the first work to show how security and access control mechanisms considered at the design phase can be derived semi-automatically in the final implementation, thus avoiding implementation details in the design phase.

Moreover, the integration of our approach with TITAN allows us to provide advanced reasoning and analytics to the organization implementing the database. TITAN [4] is a software platform with which to manage the entire life cycle of data science workflows, from deployment to execution, in the context of Big Data applications. TITAN uses semantics to deal with data and create interoperable components, improves the data analysis processes, and ensures efficient reuse and access to software components. As such, the NoSQL database designed with our approach not only implements security by design, but also offers more accessible access to other relevant components, such as algorithms and applications, thanks to TITAN. The core of TITAN is the BIGOWL ontology [5]. BIGOWL defines the set of metadata required in order to annotate Big Data analytic workflows, including algorithmic components, data sources, operation constraints, and execution planning.

Our NoSQL database design framework was integrated into TITAN by extending BIGOWL so as to enable the semantic description of the NoSQL database structure and security policies. Two new main classes were created, *Database* and *SecurityRule*. *Database* is a subclass of the *Data* class, and is related to the *SecurityRule* class. The ontology classes and properties describing a graph database were then defined, together with their specific security elements.

The significant advantages of our framework are: (i) the designer can incorporate security from the very beginning, using a high abstraction layer that hides the intricacies of particular technologies; (ii) the database designer can delay the selection of the database technology until the design requirements are clear; (iii) our approach incorporates the capability of deriving the implementation for specific target technologies, thus saving time, avoiding implementation errors, and relieving the designer from the task of having to know how a specific security policy should be implemented in a target technology, if this is even possible, and (iv) the incorporation of an ontology layer that allows us to support reasoning capabilities, thus offering the database designer an analysis of potential inconsistencies or the degree of accessibility of certain information. We wish to point out here that, to the best of our knowledge, our approach is the first to take great advantage of reasoning on ontologies in order to avoid ambiguity and ease the automatic generation of the implementation details. Furthermore, the ontologies have been built by following the FAIR (Findability, Accessibility, Interoperability, and Reusability) principles [6]. The objective of the FAIR Guiding Principles is to improve the Findability, Accessibility, Interoperability, and Reuse of digital assets. Since ontologies often result from research activities or are essential entities in many research areas, they should be treated like other research artifacts, such as data, software, methods, etc., and the FAIR principles must be applied to them. The compliance of our ontology with the FAIR principles has been assessed using the FOOPS! validator [7], a web service with which to detect best practices according to each FAIR principle, scoring the highest mark in 17 out of 24 tests.

In order to demonstrate the validity of our proposal, we have applied it to a case study focused on the healthcare domain. This domain is ideal owing to the fine-grained security aspects that need to be considered, especially as regards the accessibility of sensitive information that can be obtained directly and indirectly. It was not, unfortunately, possible to include all the material developed in this research for reasons of space. The full versions of all the material used in this paper, such as the ontology, the Graph database model, and the model transformations, can be found at https://github.com/ProyectoAether.

The remainder of this paper is structured as follows: Section 2 presents the background to and related work on ontologies and security in NoSQL databases. Section 3 details the key aspects required in order to implement security policies in Graph-oriented databases, while Section 4 presents our innovative framework for the secure design of NoSQL databases and their components. Section 5 presents the integration of our framework into TITAN [4], the ontological platform for interoperability across Big Data components, and Section 6 presents the case study focused on the healthcare domain. Finally, Section 7 presents our conclusions and outlines future work.

2. Background and related work

In this section, we start by providing background concepts in order to facilitate the reading of the paper and the understanding of our proposal. We then introduce the most closely related work as regards current proposals that attempt to model security issues at the early design stages. Finally, a literature review regarding the combination of ontologies, databases and security issues is conducted.

2.1. Background concepts

This section presents the background concepts of Semantic Web technologies for knowledge representation, structure, and reasoning.

- Ontology. According to [8,9], an ontology provides a formal representation of the real world. Ontologies are powerful tools with which to represent the knowledge from a particular domain. They define an explicit description of the concepts, relationships, attributes, and constraints of a field. Ontologies are used to allow people and organizations to communicate by providing the consensual terminology concerning a domain. One of the main features of ontologies is that they enable automated reasoning about data. Ontologies are part of the W3C (The World Wide Web Consortium) standard stack of the Semantic Web.[2] An ontology and a set of individuals form a knowledge base, which provides services with which to facilitate interoperability between multiple heterogeneous systems and databases.
- RDF. Resource Description Framework [10] is a W3C recommendation that defines a language with which to describe resources on the web. The RDF describes resources as triples, consisting of a subject, predicate and object, and can represent ontology individuals in the form of a graph. The term Knowledge Graph has recently emerged in order to refer to interlinked descriptions of entities while also encoding the semantics underlying the terminology employed. In the context of this work, RDF graphs are, therefore, knowledge graphs. The RDF Schema (RDFS) [11] describes the vocabularies used in RDF descriptions.

[2] https://www.w3.org/standards/semanticweb/

- OWL. The Ontology Web Language is used to define ontologies on the Web by extending the RDF and RDFS and adding a vocabulary. OWL is equivalent to very expressive DL (Description Logic), in which an ontology corresponds to a Tbox (Terminology box) [12]. This equivalence allows the language to exploit description logic research results. OWL provides two sublanguages: OWL Lite for simple applications and OWL-DL, which represents the subset of language equivalents to description logic, the reasoning mechanisms of which are quite complex. OWL-DL is a syntactic description that provides maximum expressiveness while retaining computational completeness and decidability [13]. The complete language is called OWL Full. OWL version 2 was published in 2009, and included new features and the definition of three new profiles: OWL 2 EL, OWL 2 QL, and OWL 2 RL [14], and a new syntax (OWL 2 Manchester Syntax). Moreover, some of the restrictions applicable to OWL DL have been relaxed, and as a result, the set of RDF Graphs that description logic reasoners can handle is slightly larger in OWL 2. Keeping in mind that OWL 2 is a popular expressive language with increased expressive power for properties, extended support for datatypes, simple meta-modeling capabilities, extended annotation capabilities, and keys, we used it to define our ontology. Moreover, OWL 2 specifies a precise mapping from ontology structures to RDF graphs, and it has an explicitly specified mapping from RDF graphs back to ontology structures.
- SPARQL (SPARQL Protocol and RDF Query Language) is a query language for easy access to RDF stores. It is the query language recommended by W3C [15] when working with RDF graphs [16], thus supporting queries and web data sources identified by URIs (Uniform Resource Identifier).
- BIGOWL is an ontology that defines the domain of workflows. The BIGOWL ontology focuses on: (1) the definition of the tasks that participate in the workflow; (2) the implementation of the components; (3) the configuration parameters; (4) the number of inputs and outputs of the task; (5) the specification of the data types of the inputs and outputs of each stage, and (6) the metadata associated with the workflow, such as the author and the number of tasks in the workflow. This ontology makes it possible to validate that the workflow is correct, since it is possible to verify that the type of the individuals in one output is compatible with the input of the next (or with one of the parents defined for that type). It is also possible to validate that all the necessary inputs of a component are connected to the output of others. Finally, the individuals that populate the ontology are defined following the structure specified and stored in an RDF repository.
- TITAN is a software platform with which to manage the entire life cycle of data science workflows, from deployment to execution, in the context of Big Data applications. Fig. 1 shows the architecture of TITAN (see [4] for a detailed description of the architecture). The architecture of the Core TITAN platform comprises a GUI (Graphical User Interface), a REST API (Representational State Transfer Application Programming Interface), and an orchestrator with distributed workers for the execution of the workflows. The TITAN GUI allows users to drag components to the design area, fill in the parameters and easily connect the components. The REST API requires RDF storage and database solutions for data persistence. Moreover, workflow workers also depend on remote data file storage in order to store component outputs. Finally, a message broker is used for distributing messages between the components.

In this work, we have defined an OWL2 ontology so as to represent the security domain in graph-oriented databases. This ontology extends the BigOWL ontology in order to model the concepts and relationships presented in a graph-oriented database, including the security policies. Both the structure and the security rules of a specific database are then codified in a knowledge graph according to the ontology. The knowledge graph is stored in an RDF repository with reasoning capabilities, thus allowing it to be queried using the SPARQL language. Finally, the TITAN platform provides the infrastructure required to implement the software component that is needed in order execute the proposed framework as a workflow.

2.2. Related work

Although security is considered an important aspect in most developments, it is very common for it to be dealt with at the end of the development process by adding the necessary security restrictions to a system that, since it is already being implemented, is not very flexible to change. The concept of security by design attempts to incorporate security needs into the development process as early as possible such that they can be considered in the design and subsequent implementation decisions, thus ensuring the development of more reliable final systems [17].

In this respect, there are proposals focusing on the development of secure systems that use notations for the specification of requirements and design (such as UML (Unified Modeling Language), i*, etc.) and extend them in order to represent security aspects. There are also complete development methodologies and processes that have been enriched with security activities [18–21]. Those that apply the model-driven development approach are particularly interesting, since they allow the automated attainment of models for specific platforms and final implementations in specific tools, initially considering the security aspects defined while saving development time and costs [22,23].

As occurs with any other software system, the development of NoSQL databases considers the stages of requirements specification and design, but takes into account the particularities of this type of systems. The relevant literature contains proposals that address the design of a NoSQL database type (columnar, document, etc.) [24–27] or that attempt to achieve a higher level of abstraction with concepts common to any NoSQL technology [28–30]. However, the proposals for the design of NoSQL databases do not incorporate non-functional requirements such as security until the final stages of implementation, which results in less secure and reliable systems [3,31].

There are limitations as regards the definition of security aspects in NoSQL database management tools at the implementation level [32–34] and works that provide advances for specific NoSQL technologies such as document [35] or columnar [36] or for specific database management tools, such as MongoDB or Cassandra [37].

With regard to graph-oriented databases, there are works focused on the access control system, such as [38], which proposes a reputation-based system, [39], which works on fine-grained security or [40], which extends Neo4J with a plugin in order to define user-based policies but without considering roles or fine-grained rules.

Our previous work has dealt with security by design in NoSQL databases but focuses on the document type. We proposed that the central point be a design-level metamodel that includes structural and security aspects specific to document databases, from which a MongoDB implementation can be generated automatically [41]. This is complemented with a modernization approach that uses domain ontologies to analyze the data in order to detect confidential information and recommends that the designer add the security policies needed to secure it [42].

The combination of ontologies and databases has been widely studied from different perspectives in the last two decades. For example, [43,44] presented surveys regarding mapping relational database contents onto ontologies in order to formally represent the content of the database and exploit the reasoning capabilities of the ontology in queries. In 2022, [45] introduced ontology learning from relational databases as an opportunity for semantic integration. Following

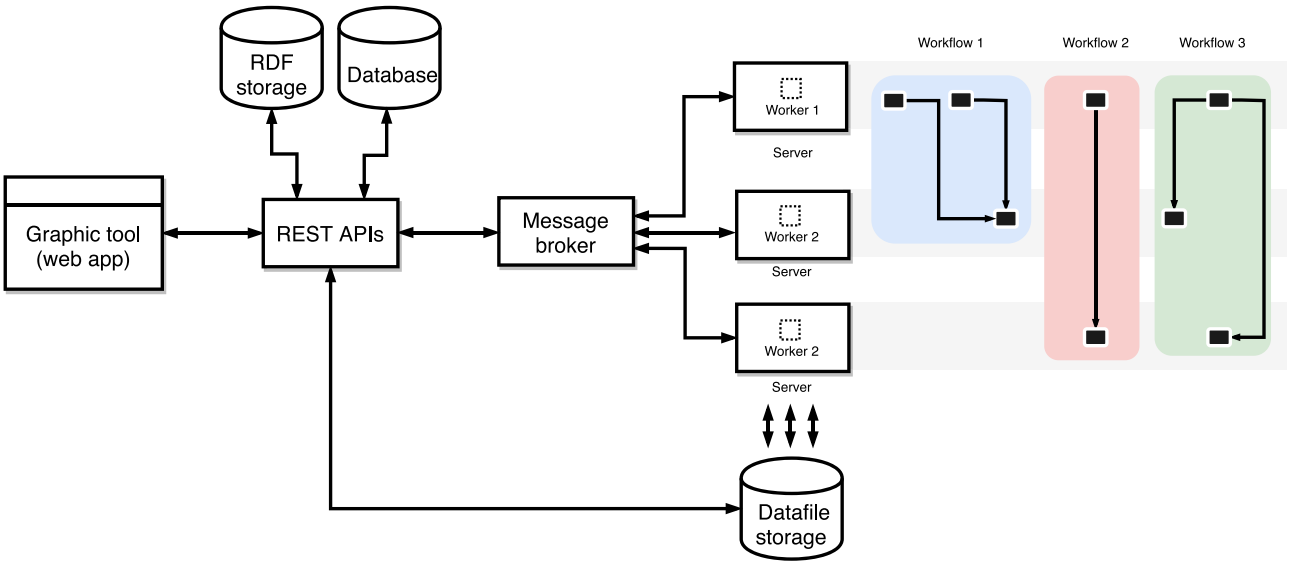


Fig. 1. Titan platform.

the same philosophy of previous works, [46,47] mapped the content of a MongoDB database onto an ontology, while [48] proposed an ontology-based semantic integration system for a column-oriented NoSQL data store. Finally, more recent works have mapped graph-oriented databases onto ontologies [49] and [50]. The objective of all of these approaches is to represent the content of the repository as an ontology. However, to the best of our knowledge, no ontologies represent the schema or structure of databases, and our proposal is, therefore, based on a representation of this nature.

The use of ontologies in cybersecurity is also a prominent area. The use of ontologies has specifically been proposed as a solution to various tasks, from modeling cyberattacks to easing auditors' or analysts' work. In this respect, OBAC [51] utilizes concepts and relations from the domain ontology of a data set to FAIR data secure access, and [52] proposes the use of semantic web technologies in order to implement control access in multi-domain environments. Another attractive research area is the security assessment domain. The most recent literature review on this topic is [53], which found that the objective of most of the ontologies identified is to describe the fields of Software Security and Software tests, including their various sub-domains, e.g. risk management, security policies, incident analysis, attack Patterns, performance tests, expert systems tests, etc.

Finally, the appearance of semantic web technologies has not been ignored in access control models for system authorization, such as RBAC (Role-Based Access Control) and ABAC (Attribute-Based Access Control). In [54], an ontology with which to extend XACML (Extensible Access Control Markup Language) [55], an XML (Extensible Markup Language) based general-purpose authorization policy model that is used to support RBAC, is described. The objective of ROWLBAC [56] is to bring formalism into policy languages by modeling RBAC in OWL. In ROWLBAC, entities (Users, Roles, Actions, etc.) are represented as OWL classes. An attempt to model ABAC concepts in OWL is carried out in [57]. In this case, complex elements of OWL, such as disjoint classes, are included in order to enrich the ontology.

To the best of our knowledge, our proposal complements all previous proposals, since it is based on an ontology that allows designers to formally represent the structure of NoSQL databases together with the security policies required. The use of an ontology allows both the database schema and security policies to be represented explicitly and formally, thus reducing ambiguity and providing a unifying framework for database developers. This representation additionally allows the schema to be processed by a computer, providing a mechanism with which to automate the database creation process. Although the problem of modifying the database schema or defining a new security rule is a limitation, it is not necessary to adjust the ontology, but only to add more individuals to it. Implementing our proposal as a workflow of the TITAN platform minimizes the impact of making a modification of this nature, since the process is fully automated. As previously-argued, and in order to show the applicability and scalability of our proposal, in this paper, we deploy our ontology for graph databases together with the security policies that can be applied to them. There is, to the best of our knowledge, no other ontology with similar features in literature.

3. Security policies in graph-oriented databases

When approaching security in NoSQL databases from a design point of view, it is necessary to, on the one hand, obtain the structural aspects specific to the type of database (graph-oriented, document, or columnar) and, on the other, define security policies related to them that can be independent and reusable for different types of NoSQL databases.

In order to assist in the design of security policies for graph-oriented databases, we present two metamodels: one with the structural concepts of graph-oriented databases (Fig. 2) and the other with the elements required in order to define security policies for databases, regardless of their type (Fig. 3).

The metamodel for graph-oriented databases makes it possible to establish all the necessary structural elements, signifying that a database has elements that can be Node or Relationship between two nodes. Both nodes and relationships can have an associated Field with an associated DataType.

Furthermore, the security metamodel allows the establishment of security policies in the database. This is done by allowing the definition of a set of security rules that grant or deny privileges (Create, Read, Update, Delete) regarding database elements to specific users. A condition that has to be satisfied can be specified for each privilege involved in the rule.

A role-based access control policy defines role hierarchies and associates users with them in order to group database users. This access control policy is that which is most widely used and supported by end tools. Security rules can be associated with database elements (SecurityRuleElement) or the fields of those elements (SecurityRuleField). In this paper, which is focused on graph-oriented databases, these rules refer to nodes and their relations or attributes. However, the security notation is independent of the type of NoSQL database that is applicable to other types (such as document and columnar), referring in this case to the specific elements of that type of database (documents, columns, etc.).

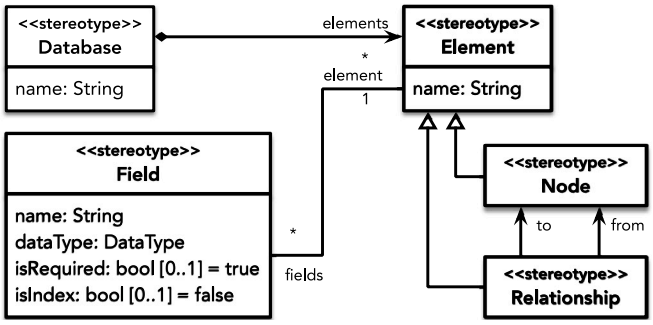


Fig. 2. Graph metamodel.

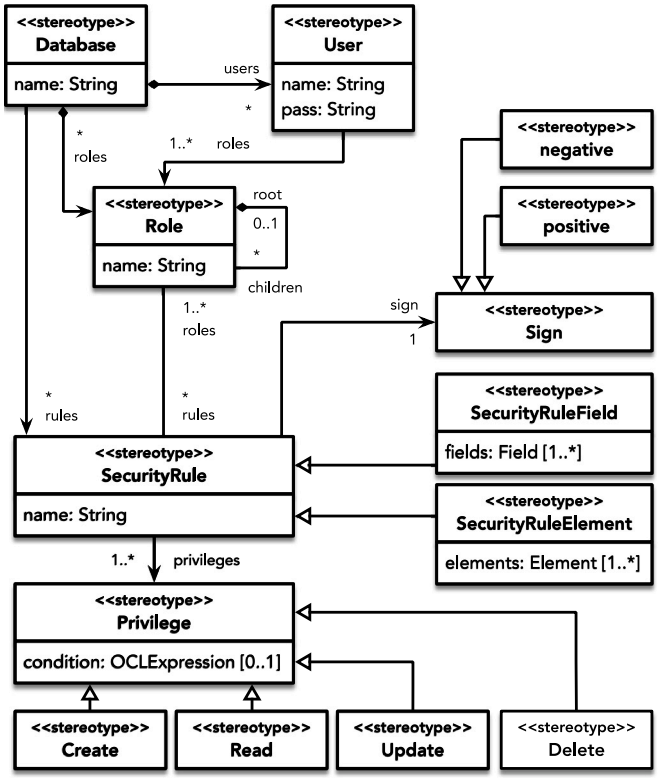


Fig. 3. Security metamodel.

4. An ontology-based framework for security in graph-oriented databases

In order to achieve the main objective of this work, it was first necessary to define an ontology that would generically encompass and represent the security domain in graph-oriented databases. The purpose of the proposed ontology is to enrich and facilitate the integration of information into existing and future platforms. In this respect, BIGOWL was extended in order to model both the structure (Fig. 2) and the security rules (Fig. 3) of a graph-oriented database. We specifically created a subclass of the BIGOWL class *Data* called *Database*, which contains a subclass called NoSQLDatabase. Finally, the class *GraphsDatabase* was included as a subclass of *NoSQLDatabase*. We should point out that our approach has been made scalable to any NoSQL database by adding the corresponding subclass and gathering the specific requirements of the database focused on.

The security ontology in graph databases was designed by following the well-known and widely-accepted "Ontology 101 development Process" methodology [58], which consists of the following seven steps:

1. **Determine the domain and scope of the ontology.** The application domain of the ontology defines aspects related to security and structure and the elements that define graph-oriented databases.
2. **Consider reusing existing ontologies.** As mentioned previously, we have considered reusability and integration with the BIGOWL [5] ontology. This ontology defines the *PrimitiveType* and *StructuredType* classes that have the generic *DataType* class as their parent; The type of a field is, therefore, a BigOWL *DataType*.
3. **Enumerate important terms in the ontology.** The essential terms in our ontology have been analyzed and extracted from the metamodels presented in Section 3. Examples of these terms are: *SecurityRule*, *Privilege*, *Role*, *User*, *ruleSign*, *RoleHasRules*, *UserHasRole*, etc. We shall, therefore, also define generic terms from graph-oriented databases, such as *Node*, *Relationship*, *Field*, and *DataType*.
4. **Define the classes and the class hierarchy.** The ontology classes are extracted from the relevant terms. We define classes and their hierarchy to the level of detail necessary to classify individuals. Fig. 4 shows the ontology classes. BIGOWL classes are marked in orange. The green tagged classes model the structure of a graph-oriented database, while the blue ones represent the security elements. The figures show that some of these classes are related by means of object properties or the is-a (subclass of) relationship.
5. **Define the properties of classes.** We have defined the object and data properties necessary to represent the connections required among the different individuals and store the information for each individual. Examples of properties are: *DatabaseHasRole*, which indicates the roles that exist in the database, *UserHasRole*, which relates users to the roles they have, *ruleSign*, which stores the sign of the rule, and *SecurityRuleDefineElements*, which links the security rule to the elements to which it is applied. Table 1 provides a description of the ontology properties of the security classes in Description Logic format. Examples of properties for the structural part of the graph databases are (see Table 2): *hasElements*, which relates the database to the elements it contains, *hasRelationshipTo* and *hasRelationshipFrom*, which are used to connect a relationship to the nodes, and *hasElementField*, which maps a field onto an element, and so on.
6. **Define the facets of the slots.** This step includes the definition of cardinality and value constraints. Value constraints are used to specify the data type of a property domain and range. For example, the range of the *fieldIsRequiered* property is restricted to Boolean. In contrast, the domain of the *hasRelationshipFrom* property is Relationship, and the range is limited to individuals in the *Node* class. The range of the *ruleSign* property is restricted to the values "+" or "-". In addition, the *HasSubRole* object property has been marked as transitive in order to allow the reasoner to infer the roles hierarchy.
7. **Create instances.** The ontology instances are generated from the metamodels, which are defined in XMI (XML Metadata Interchange) [59]. The metamodels are transformed into RDF through the use of mapping functions according to the specified classes and properties of the ontology, thus leading to the creation of a knowledge graph in RDF (see Section 4.2).

4.1. Ontology model

After applying the methodology described above, an ontology was developed. The ontology defines a total of 24 classes (that represent individuals with the same taxonomy), 12 object properties (that represent binary relationships between individuals), 7 data properties (attributes of individuals), and 112 constraint axioms. The ontology is, therefore, sufficiently large to allow complex reasoning.

The main classes, along with a brief description of each of them, are listed below:

- ***SecurityRule.*** This class represents the security rules in a generic database. As object properties have been defined: *SecurityRuleDefined* links the rule with the element of the database on which it is defined, *SecurityRuleContainsPrivilege* links the rule to the privilege that the rule grants or revokes. We have defined the *SecurityRuleElement* and *SecurityRuleField* subclasses in order to specify whether it is a rule that affects an element or a field in the case of a graph-oriented database. *SecurityRuleElement* and *SecurityRuleField* are disjoint classes, i.e., there is no rule for these two Classes.
- ***Privilege.*** This class describes the privileges associated with a security rule. Privileges are specialized into four sub-classes: Create, Read, Update, and Delete.
- ***Role.*** This class models the roles that are defined in a database. Roles group a set of rules that are applied to users. The creation of roles facilitates security management, since assigning individual rules to each user is unnecessary. The object properties of the class *Role* are *RoleHasRules*; this property links the role to the rules and *HasSubRole*, which allows the definition of hierarchies between roles.
- ***User.*** The *user* class represents the database management system users. The *UserHasRole* object property binds a user to its assigned roles.
- ***Database.*** This is the generic concept of a database. The database name is stored with the *databaseHasName* data property. *Database* is a subclass of the BIGOWL class *Data*. *NoSQLDatabases*, ultimately, *GraphsDatabases* are subclasses of *Database*. The *hasElement* object property relates an instance of a graph-oriented database to the elements of which it is composed.
- ***Element.*** According to graph theory, the *Element* class generically defines the common properties of the terms comprising a graph, i.e., Nodes and Relationships. The *elementHasName* data property stores the name of each element in the graph. The *hasElements* object properties relate a *GraphDatabase* to its elements. Furthermore, *hasElementField* links individuals from *Element* to individuals in the *NoSQLFieldGraph* class. *Node* and *Relationship* are specified as subclasses of *Element*; two object properties have been included for these classes: *hasRelationshipFrom* and *hasRelationShipTo*, which associate the nodes with the relationships.
- ***Field.*** This represents the concept of a data field in any database, regardless of its implementation. The *fieldHasName* and *isRequired* data properties store the name of the field and whether the field is required; With regard to object properties, *hasDatatype* connects the field with its data type. As subclasses, we defined the hierarchy *NoSQLField* and *NOSQLFieldGraph* in order to annotate the specific database type to which the field belongs.

The implementation of the ontology in OWL2 makes it possible to exploit the ontological language reasoning capabilities. Rule 1 shows a simple example of how reasoning can detect inconsistencies in the rule definition by detecting that a rule is defined as an Element and a Field rule.

Rule 1: Sample rule of reasoning.

```
OWL 2 Semantic rule:
Let C be a class, P a property, and x and y individuals.
If Domain(P,C) and P(x,y) then C(x) (sr1)

Ontology layer:
Domain(SecurityRuleDefinedField, SecurityRuleField)
DisjointClasses(SecurityRuleDefinedField,
                SecurityRuleDefinedElement) (ax1)

Individuals:
SecurityRuleElement(rule1)SecurityRuleDefinedField(rule1,
    field1)
—> SecurityRuleField(rule1) (by sr1)Inconsistence (by ax1)
```

4.2. Knowledge graph creation

After the development of the ontology, the metamodels were translated to the RDF in order to build a knowledge graph according to the ontology structure. Fig. 5 shows the infrastructure designed for this transformation. A set of specific mapping functions was developed. These functions produce the individuals from the XMI file by following the ontology class hierarchy. The ontology object and data properties make it possible to relate individuals to each other. The metamodels are specified in XML. Function 2 shows an extract of the function that creates the RDF triples for the privileges.

Function 2: Parse privileges.

```
def parse_privileges(rule, privilege):
    id_privilege = hashlib.sha224(privilege["@xmi:id"])
    uri_privilege = URIRef(sec + id_privilege)

    rdf_graph.add(
    (uri_privilege,RDF.type,sec[privilege["@xsi:type"]]))

    rdf_graph.add(
    (rule,sec.SecurityRuleContainsPrivilege,uri_privilege))

    if "@condition" in privilege:
        rdf_graph.add(
        (uri_privilege, sec.privilegeCondition,
            Literal(privilege["@condition"])))
```

Once the knowledge graph has been created, it is stored in an RDF Stardog[3] repository with persistent and reasoning capabilities. At this point, it is now possible to query the knowledge graph from a SPARQL Endpoint. Query 1 shows an example of a SPARQL query in the RDF repository. It retrieves details of a specific rule (sign, role, privilege, element name, action name, and condition).

Query 1: Obtain details of security rule elemement.

```
PREFIX sec: <https://w3id.org/OntoSecurityGraphDB/>
PREFIX db: <https://w3id.org/OntoGraphDB/>
SELECT ?sign ?RoleName ?privilege ?elementName
       ?condition
WHERE{
?role rdf:type sec:Role.
?role sec:roleName ?RoleName.
?role sec:RoleHasRules ?rule.
?rule sec:ruleName ?ruleName.
    FILTER(?ruleName = "rule_name")
OPTIONAL{
  ?rule sec:ruleSign ?sign.}
OPTIONAL{
  ?rule sec:SecurityRuleContainsPrivilege ?priv_uri.
  ?priv_uri rdfs:label ?privilege.
    OPTIONAL{
      ?priv_uri sec:privilegeCondition ?condition.} }
?rule sec:SecurityRuleDefineElements ?element.
?element db:elementHasName ?elementName.
?element rdf:type ?ty FILTER(?ty!=owl:NamedIndividual).}
```

We have also defined an API that wraps the SPARQL queries. The objective of this API is to facilitate the extraction of the necessary data from the knowledge graph in order to generate the security policies for the different graph database implementations. The implemented calls return the roles, their direct children and their descendants; the users, the roles assigned to a user; the security rules defined for elements; the rules defined for fields, etc. The querying of the knowledge graph makes it possible to obtain all the data needed in order to create the security policies in a final graph-oriented database.

[3] https://www.stardog.com/

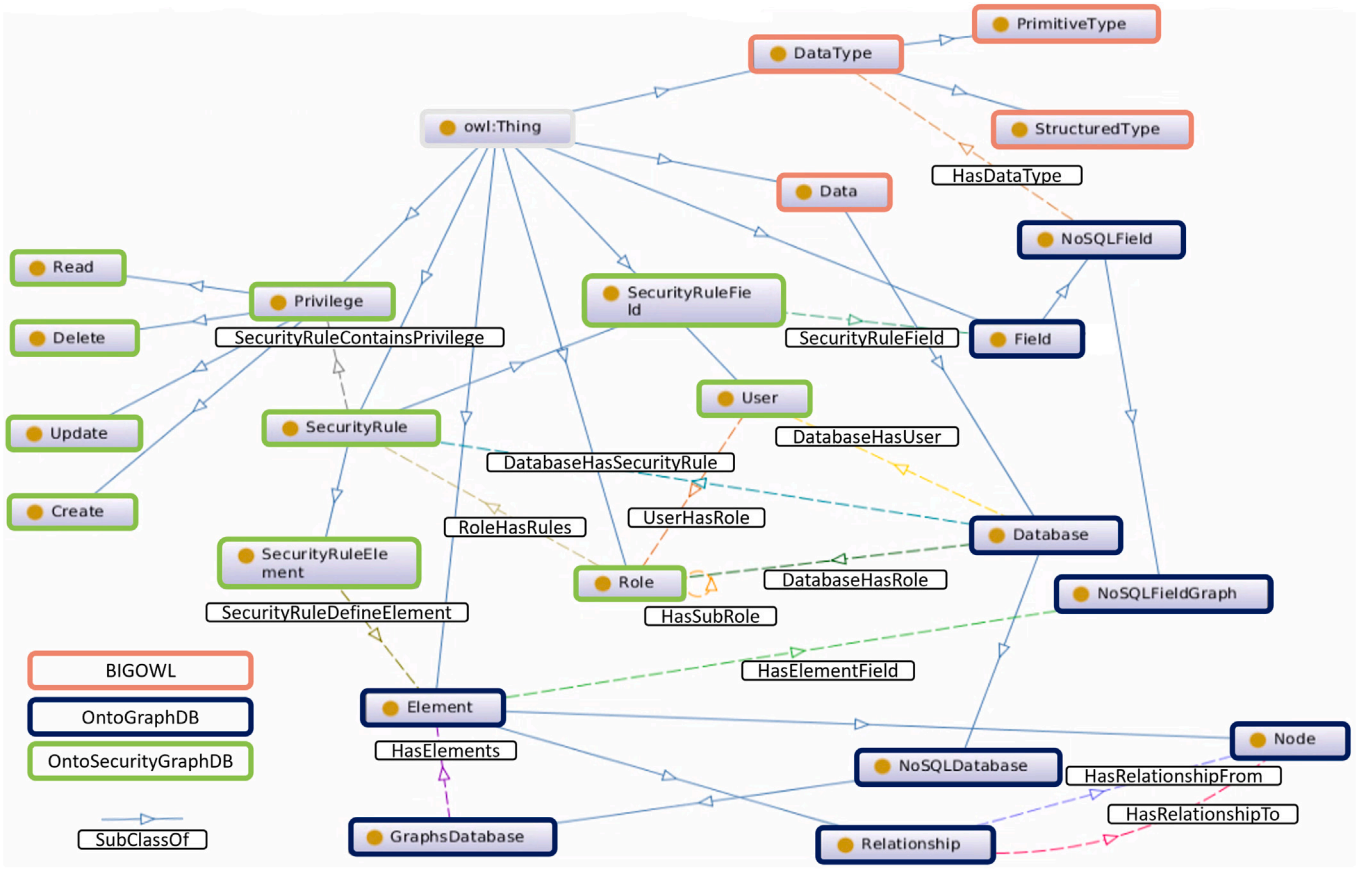


Fig. 4. General overview of security graph ontology.

Table 1
Data properties and object properties of the ontology defined for No-SQL security.

Object properties	Description logics
DatabaseHasRole	∃ DatabaseHasRole Thing ⊑ Database ⊤ ⊑ ∀ DatabaseHasRole Role
DatabaseHasSecurityRule	∃ DatabaseHasSecurityRule Thing ⊑ Database ⊤ ⊑ ∀ DatabaseHasSecurityRule SecurityRule
DatabaseHasUser	∃ DatabaseHasUser Thing ⊑ Database ⊤ ⊑ ∀ DatabaseHasUser User
HasSubRole	TransitiveProperty HasSubRole ∃ HasSubRole Thing ⊑ Role ⊤ ⊑ ∀ HasSubRole Role
RoleHasRules	∃ RoleHasRules Thing ⊑ Role ⊤ ⊑ ∀ RoleHasRules SecurityRule
SecurityRuleContainsPrivilege	∃ SecurityRuleContainsPrivilege Thing ⊑ SecurityRule ⊤ ⊑ ∀ SecurityRuleContainsPrivilege Privilege
SecurityRuleDefineElements	⊑ SecurityRuleDefined ∃ SecurityRuleDefineElements Thing ⊑ SecurityRuleElement ⊤ ⊑ ∀ SecurityRuleDefineElements Element
SecurityRuleDefineField	⊑ SecurityRuleDefined ∃ SecurityRuleDefineField Thing ⊑ SecurityRuleField ⊤ ⊑ ∀ SecurityRuleDefineField Field
SecurityRuleDefined	∃ SecurityRuleDefined Thing ⊑ SecurityRule
UserHasRole	∃ UserHasRole Thing ⊑ User ⊤ ⊑ ∀ UserHasRole Role
Data properties	**Description logics**
privilegeCondition	∃ privilegeCondition Datatype Literal ⊑ Privilege ⊤ ⊑ ∀ privilegeCondition Datatype string
roleName	∃ roleName Datatype Literal ⊑ Role ⊤ ⊑ ∀ roleName Datatype string
ruleName	∃ ruleName Datatype Literal ⊑ SecurityRule ⊤ ⊑ ∀ ruleName Datatype string
ruleSign	∃ ruleSign Datatype Literal ⊑ SecurityRule ⊤ ⊑ ∀ ruleSign {"+"string"} ⊔ {"-"string}
userName	∃ userName Datatype Literal ⊑ User ⊤ ⊑ ∀ userName Datatype string
userPass	∃ userPass Datatype Literal ⊑ User ⊤ ⊑ ∀ userPass Datatype string

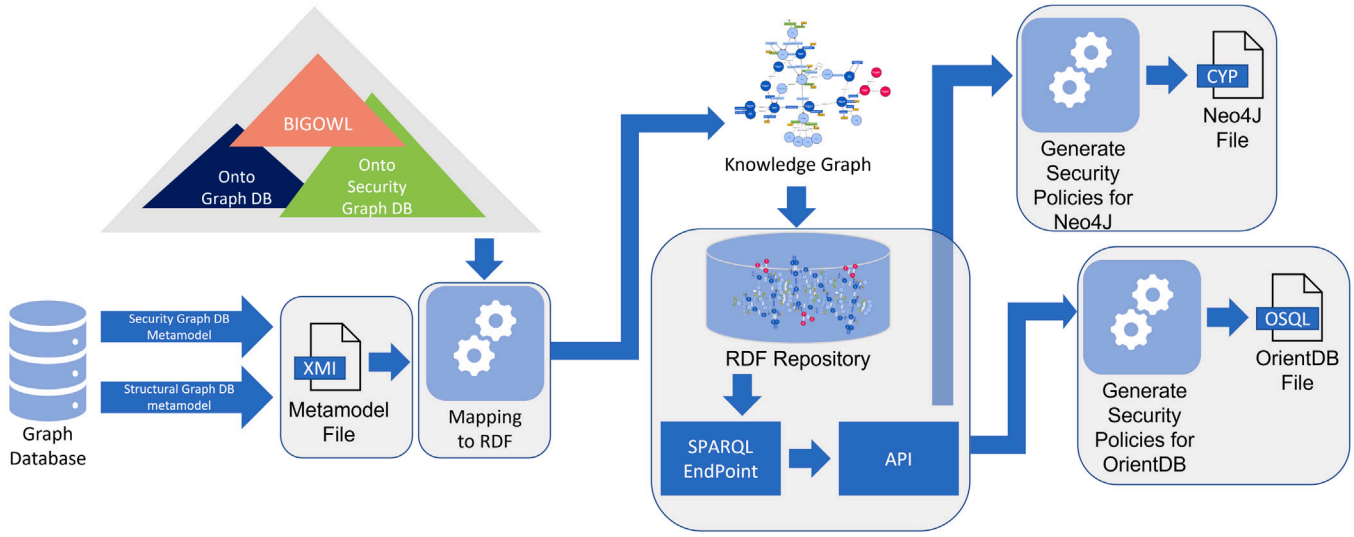


Fig. 5. Infrastructure used to create the knowledge graph.

Table 2
Data properties and object properties of the ontology defined for graph-oriented databases.

Object properties	Description logics
hasDatatype	∃ hasDatatype Thing ⊑ NoSQLField ⊤ ⊑ ∀ hasDatatype DataType
hasElementField	∃ hasElementField Thing ⊑ Element ⊤ ⊑ ∀ hasElementField NoSQLFieldGraph
hasElements	∃ hasElements Thing ⊑ GraphsDatabase ⊤ ⊑ ∀ hasElements Element
hasRelationshipFrom	∃ hasRelationshipFrom Thing ⊑ Relationship ⊤ ⊑ ∀ hasRelationshipFrom Node
hasRelationshipTo	∃ hasRelationshipTo Thing ⊑ Relationship ⊤ ⊑ ∀ hasRelationshipTo Node
Data properties	**Description logics**
databaseHasName	∃ databaseHasName Datatype Literal ⊑ Database ⊤ ⊑ ∀ databaseHasName Datatype string
elementHasName	∃ elementHasName Datatype Literal ⊑ Element ⊤ ⊑ ∀ elementHasName Datatype string
fieldIsIndex	∃ fieldIsIndex Datatype Literal ⊑ NoSQLFieldGraph ⊤ ⊑ ∀ fieldIsIndex Datatype boolean
fieldHasName	∃ fieldHasName Datatype Literal ⊑ NoSQLField ⊤ ⊑ ∀ fieldHasName Datatype string
fieldIsRequired	∃ fieldIsRequired Datatype Literal ⊑ NoSQLField ⊤ ⊑ ∀ fieldIsRequired Datatype boolean

One of the great advantages of developing an ontology is the ability to use reasoning capabilities. For example, the fact of having declared the *HasSubRole* object property as transitive allows us to obtain all the descendants of a specific role using the reasoner when running Query 2; if the reasoner is not employed, the query returns only its direct children. In Query 3, the reasoner infers that a Graph Database is a Database and then returns the security rules defined for that database. Without the reasoner, the query obtains no results. Note that we could easily extend the ontology to include other types of databases.

Another advantage is that the developer can model the security using Graph DB ontology concepts such as Element or Field without having yet selected the particular graph database technology to be used. This makes it possible to delay the selection if necessary until requirements such as scalability, server-side programming, or ACID (Atomicity, Consistency, Isolation, Durability) properties, among others, have been clarified or until security policies need to be implemented.

Query 2: Obtain all children of a role.

```
PREFIX sec: <https://w3id.org/OntoSecurityGraphDB/>
SELECT ?RoleName ?subRoleName
WHERE{
?role rdf:type sec:Role.
?role sec:roleName ?RoleName. FILTER(?RoleName = "rol")
    OPTIONAL{
            ?role sec:HasSubRole ?subRole.
            ?subRole sec:roleName ?subRoleName.
    }
}
```

Query 3: Obtain all security rules elements.

```
PREFIX sec: <https://w3id.org/OntoSecurityGraphDB/>
PREFIX db: <https://w3id.org/OntoGraphDB/>
SELECT ?ruleName
WHERE{
      ?dataBase rdf:type db:Database.
      ?dataBase db:databaseHasName ?DatabaseName.
          FILTER(?DatabaseName = "db_name")
      ?dataBase sec:DatabaseHasSecurityRule ?rule.
      ?rule rdf:type sec:SecurityRuleElement.
      ?rule sec:ruleName ?ruleName.
}
```

5. Implementation

We have implemented our proposal using TITAN as a workflow orchestration and execution platform. The core of TITAN is the BIGOWL ontology, which allows us to define different types of components and their inputs, outputs, and parameters as RDF triples. This platform makes it possible to compose and execute workflow instances by means of a web interface and API REST. One of TITAN's features is that of listing the catalog of available components and their inputs and outputs. The list of components is obtained through the use of calls to the API REST, which transforms the SPARQL queries into JSON. The workflow is executed once the compatibility between components has been designed and validated. Each execution is semantically annotated as a task type in the TITAN RDF repository. A task is a concrete component instance with its parameters instantiated for that execution. Finally, the API REST allows users to check the execution status and download the files generated in each component.

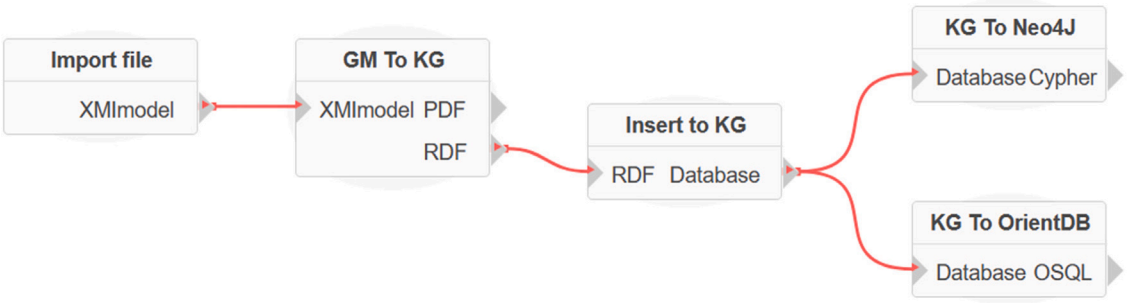


Fig. 6. Workflow components.

The workflow has been implemented in 5 interconnected components. Fig. 6 shows the workflow diagram. The functionality of each component is detailed below.

- ***Import file.*** This component is the user's data entry point to the workflow. The component has the URL of the file that contains the metamodel as a parameter. Its output is the address in the storage system used by TITAN.
- ***Graph model to knowledge graph.*** This component translates the metamodels into RDF triples. The component outputs are a file with the triples in RDF and a PDF file with a representation of the knowledge graph.
- ***Insert into knowledge graph.*** This component receives the file with RDF triples that contain the information regarding the individuals and the ontologies defined. Triples are stored in an RDF repository database. The component returns the knowledge graph identifier as output, thus allowing other components to use it.
- ***Knowledge graph to Neo4J*** and ***Knowledge graph to OrientDB.*** These components extract the information from the knowledge graph and transform it into the definition language used by Neo4J and OrientDB for security policies, respectively. The implementation of these two components is described in the following section.

5.1. Implementation of security policies in database management systems

This section deals with the implementation of security policies defined at the design level in final graph-oriented database management tools. Although our proposal is applicable to any tool, in this work, we have focused on Neo4J and OrientDB because Neo4J is considered a reference in this type of systems[4] and OrientDB includes some interesting security functionalities (that Neo4J does not allow) such as representing security policies with associated conditions.

We could apply our proposal to other graph-oriented database management tools (such as JanusGraph, NebulaGraph, Memgraph, TigerGraph, etc.). However, they have not been considered in this work, since they are less interesting from a security point of view, as they usually have basic RBAC systems and limited functionalities with which to establish security policies.

First, we analyze the features offered by Neo4J for the definition of security policies. Table 3 shows the concrete syntax in EBNF (Extended Backus–Naur Form).

It will be noted that a security policy includes information related to the authorization sign, privileges, elements, and roles it affects. In the specific case of Neo4J, it is important to mention that the classic privileges (creation, reading, modification, and deletion) are specialized into more types of privileges and are simultaneously grouped in a hierarchical manner. The read permissions refer to access to the value (Read) or to being able to reach the element (Traverse), and have a privilege that groups the two previous ones (Match). With regard to writing, there is a privilege that groups the others (Write), which in turn specializes in creation (Create), deletion (Delete), modification (Set Property), and label management (set and delete label).

Table 3
Syntax used to specify security policies in Neo4J.

$< securitypolicy > ::= < sign >< privilege >$
$[< properties >]ON < graph >< entity > TO < role >$
$< sign > ::= GRANT \mid DENY$
$< privilege > ::= MATCH \mid TRAVERSE \mid READ \mid$
$WRITE \mid CREATE \mid DELETE \mid SETPROPERTY \mid$
$SETLABEL \mid DELETELABEL$
$< properties > ::= \}\}\{''* \mid < nameList > \}\}\}''$
$< graph > ::= DEFAULTGRAPH \mid$
$(GRAPH[S](* \mid < nameList >))$
$< entity > ::= < entityType > (* \mid < nameList >)$
$< entityType > ::= ELEMENT[S] \mid NODE[S] \mid$
$RELATIONSHIP[S]$
$< roles > ::= < nameList >$
$< nameList > ::= < name > [\}\},'' < name >] *$

The structural elements are also categorized in hierarchies, signifying that we can refer to nodes (Node), relationships (Relationship), or both, which are denominated as elements (Element). In this respect, if we wish to establish a fine-grained authorization for specific properties, we should indicate the names of these properties and the names of the elements to which they belong.

Table 4 shows what the syntax with which to implement security policies in the other graph-oriented database management system considered, OrientDB, would be.

Its structure is similar to the syntax in Neo4J, and it allows the definition of security policies by indicating the sign, privileges, elements, and roles affected. However, both tools differ as regards the number of policies that must be defined in order to specify the same security constraint. In OrientDB, it is possible to group several privileges (create, read, etc.) in a single security policy, while in Neo4J, it is necessary to define one policy per privilege. Furthermore, OrientDB has the requirement that each policy should affect only one role, while Neo4J allows it to specify a list of roles. The most interesting aspect of OrientDB is that, unlike Neo4J, it allows each privilege to be associated with a condition (sqlPredicate) that must be fulfilled. For example, this would make it possible to represent policies that allow users to query or edit their own data. Let us imagine a medical system in which a user of a patient role can query a hypothetical patient node (with its attributes), but only the instance that refers to its own data.

Having analyzed the target tool, we now show the development of a set of transformations. This starts with the design level specification and leads to the automatic generation of the corresponding implementation of the security policies.

These transformations have been implemented as Python scripts packaged in a Docker container, and make use of the API developed in this work to query the necessary semantic resources (graph-oriented databases structure, security model for NoSQL databases, and the information of the particular design model to be transformed).

[4] https://db-engines.com/en/ranking/graph+dbms

Table 4
Syntax used to specify security policies in OrientDB.

< *securitypolicy* >::=< *sign* >< *listOf Privileges* >
ON < *element* > *TO* < *role* >
< *sign* >::= *GRANT* | *REVOKE* < *privilege* >::=< *action* > [= (< *sqlPredicate* >)]?, < *action* >::= *CREATE* | *READ* | *BEFOREUPDATE* | *AFTERUPDATE* | *DELETE* | *EXECUTE* < *element* >::= *database.class.*[∗ | < *name* >]

6. Validation: Hospital use case

Our framework is applicable to the secure design of any graph-oriented database. The following is an example of a validation, in which our proposal has been applied to the health domain, and more specifically to the management of diagnoses involving patients, doctors and associated treatments. The following illustrates how a designer would apply our proposal to the secure design of this database. These steps could be extrapolated to the design of any other graph-oriented database.

First, the designer models the graph-oriented database at a high level. This includes the necessary structural aspects (Nodes, Relationships, Fields, etc.), along with the security policies needed in order to satisfy the security requirements of the system.

This task is carried out without taking into account specific characteristics of the database management system in which the system is eventually implemented, since this is an aspect that our proposal generates automatically.

In this example, the security policies of the system are generated by considering Neo4J as the target platform, and we eventually check how the implementation generated satisfies the security requirements established.

We begin with the structural part (Fig. 7), and the following nodes, relationships and properties are then defined. The administration staff (AdmissionStaff node) is in charge of registering patients (Patient node) and maintaining their associated information (name, address, social security number, etc.). Doctors (Doctor node) have an associated specialty. They are in charge of diagnosing patients, signifying that patients have associated diseases (Disease nodes) diagnosed on a certain date by a specific doctor. Each disease additionally has a series of possible treatments associated with it (Treatment node), and the doctor selects one of them as the current treatment that a particular patient suffering from that disease is following.

With regard to the security aspect, the decision is first made to define a role for each type of user that will be able to interact with the system: administration staff (RoleAdmissionStaff), patients (RolePatient), and doctors (RoleDoctor). A set of authorizations is then established for each of these roles, which limit their privileges according to the security policy sought. In order to provide more details on this, the following security rules are defined:

- The patient role (Fig. 8) has a positive authorization defined for query doctors.
- The admission staff role (Fig. 9 has several authorizations associated with it in order to grant privileges regarding nodes and relationships. On the one hand, it is necessary to read, create, and update (not delete) privileges for patients and for the relationship that indicates that a certain admission worker has registered a certain patient. On the other hand, the admissions staff is granted read privileges concerning admissions staff.
- The doctor role (Fig. 10) presents two authorizations similar to those described above, which grant several privileges to nodes and relationships. Nevertheless, it also defines two fine-grained authorizations for fields. The first rule establishes a negative authorization that withdraws read permission for the patient's social security number (which had full read access). The second rule also refines access to patient information, this time withdrawing the privilege to read their addresses, but only for those patients who are underage.

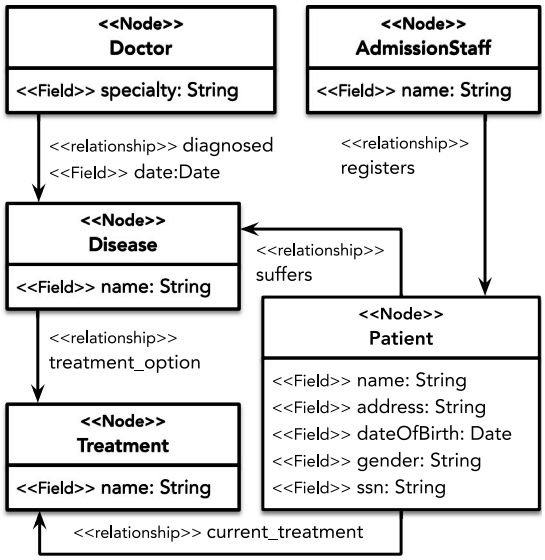


Fig. 7. Use case: structural aspects.

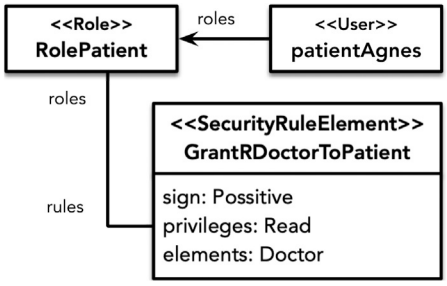


Fig. 8. Use case: role patient.

We execute the TITAN workflow with the Metamodels shown in Figs. 7, 8, 9, 10 as input. The knowledge graph is created and stored in the Stardog RDF repository.

Figs. 11 and 12 show part of this knowledge graph, specifically the structural aspects of the database and the patient role, respectively. The pink nodes in the knowledge graph represent individuals, while the blue nodes represent ontology classes. The square nodes symbolize data values, and the arcs are ontology objects and data properties.

The security policy components can, therefore, extract data by means of the REST API, which encapsulates the SPARQL queries. For example, Query 3 makes it possible to obtain all the rules defined in the database. Table 5 displays the query result. Once all the rule names have been obtained, Query 1 is executed in order to retrieve the rule details. Table 6 shows the details of the rule *GrantRNodesToDoctor*, which authorizes the RoleDoctor role to read several nodes. Note that it completely affects the Patient, Disease, Doctor, and Treatment nodes, without having any condition to evaluate (which is an optional field of the security rules).

The last step of our proposal consists of the automated generation of the implementation of the security policies defined in a specific graph-oriented database manager. For this case study, we generate the implementation of the security policies specified in the model for the Neo4J and OrientDB tools.

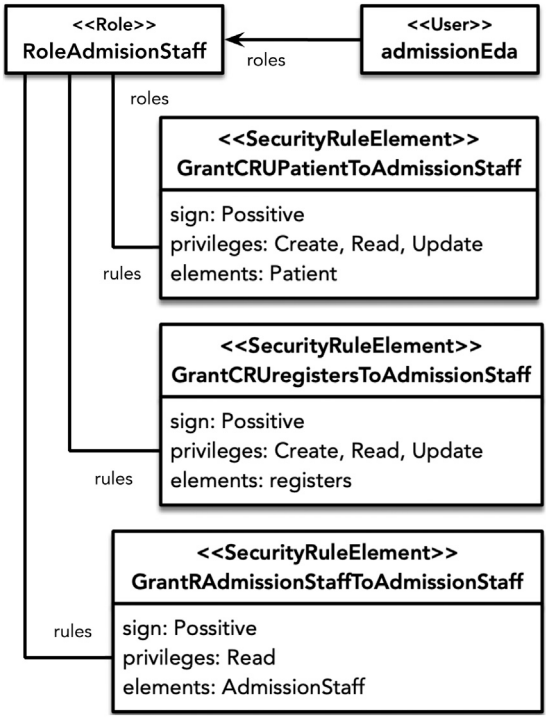


Fig. 9. Use case: role admission staff.

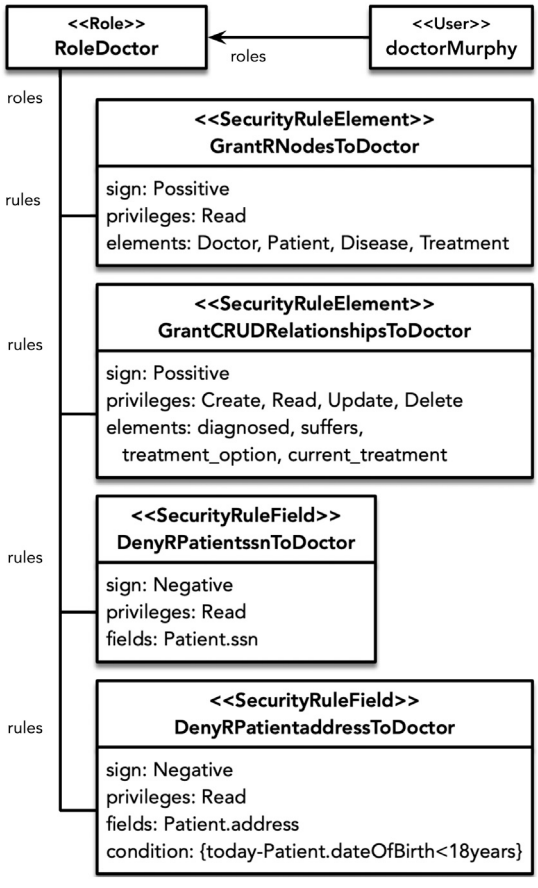


Fig. 10. Use case: role doctor.

Table 5
Query 3 results.

ruleName
GrantCRUregistersToAdmissionStaff
GrantCRUPatientToAdmissionStaff
GrantRNodesToDoctor
GrantRAdmissionStaffToAdmissionStaff
GrantCRUDRelationshipsToDoctor
GrantRDoctorToPatient

Table 6
Query 1 results.

sign	RoleName	privilege	elementName	condition
+	RoleDoctor	Read	Patient	
+	RoleDoctor	Read	Disease	
+	RoleDoctor	Read	Doctor	
+	RoleDoctor	Read	Treatment	

Listing 4: GrantCRUPatientToAdmissionStaff.

```
# Neo4J
GRANT CREATE ON GRAPH Hospital NODE Patient
 TO RoleAdmisionStaff;
GRANT MATCH ON GRAPH Hospital NODE Patient
 TO RoleAdmisionStaff;
GRANT SET PROPERTY {*} ON GRAPH Hospital
 NODE Patient TO RoleAdmisionStaff;
# OrientDB
GRANT SET CREATE, READ, AFTER UPDATE ON
  database.class.Patient TO RoleAdmisionStaff;
```

Listing 5: DenyRPatientaddressToDoctor.

```
# Neo4J
DENY MATCH address ON GRAPH Hospital
 NODE Patient TO RoleDoctor;
# OrientDB
REVOKE SET READ = (today—Patient.dateOfBirth
 < 18 years) ON database.class.address
 TO RoleDoctor;
```

Listings 4 and 5 provide an example of how several security rules have eventually been implemented. On the one hand, we show the "GrantCRUPatientToAdmissionStaff" rule, which grants creation, reading, and modification privileges for the Patient element for the RoleAdmissionStaff role. This rule is transformed into several security policies (one per privilege) in Neo4J, and into a single policy in OrientDB (Listing 4).

On the other hand, we show the "DenyRPatientaddressToDoctor" rule, which denies reads as regards the address attribute of Patient to the Doctor role, but includes a condition indicating that it will affect only those patients under the age of 18. The setting of these conditions is not supported by Neo4J but is supported by OrientDB, and the transformations will, therefore, act by following different strategies. In Neo4J, a conservative strategy is chosen in which the address of all Patient instances is hidden, while in OrientDB the condition is indicated in the security policy (Listing 5).

7. Conclusions

NoSQL databases are characterized by the fact that they provide high performance and flexibility while considering security to be a 'second-class citizen'. The lack of standardized security mechanisms, even in technologies that share the same NoSQL database type, and the absence of a high-abstraction secure methodology make it challenging

Fig. 11. Fragment of the knowledge graph representing the Structural aspects.

to create secure designs. In this context, the database designer has to be knowledgeable as regards the particular low-level mechanisms offered by each technology, limitations, and workarounds in order to achieve a secure implementation.

In order to tackle this problem, in this paper we present the first high-abstraction security framework for the secure design of NoSQL databases. One of the key aspects of our approach is the proposal of an ontology that allows designers to simultaneously model the high level concepts and structures of NoSQL databases together with the security policies required in the same design phase. Another great advantage of our framework is that, thanks to the ontological layer, designers can implement ontology rules in order to analyze and detect potential errors and overlooked issues. Furthermore, our framework makes it possible to automatically derive the implementation of the secure NoSQL databases regardless of the concrete technology that will be used in the future, thus saving time and avoiding the errors introduced by ad-hoc implementations. Since the implementation of security mechanisms requires fine-grained concepts that vary from one NoSQL database type to another, we have focused on the extended model for Graph-based databases. In order to show the applicability of our proposal, we have applied it in a case study in the Healthcare domain by designing a secure Graph-based database and implementing it in two popular technologies, Neo4J and OrientDB.

We would also like to stress here that our framework has been integrated into the well-known TITAN framework, thus allowing us to easily integrate the implemented database into more complex and huge data science workflows and interact with data science algorithms and data transformation steps. In other words, our framework is not only another theoretical framework, but is also a framework that is ready to be used and embedded in complex data science workflows and processes.

Our immediate future work will be focused on exploring the comprehensive reasoning capabilities that can be implemented from our ontology, thus allowing database designers to better evaluate potential information leaks and undesired data access, along with other design errors that could be detected before generating the final NoSQL database implementation. In the long term, we plan to support all kinds of NoSQL datastores, which will require the analysis and development of specific ontologies for each of them.

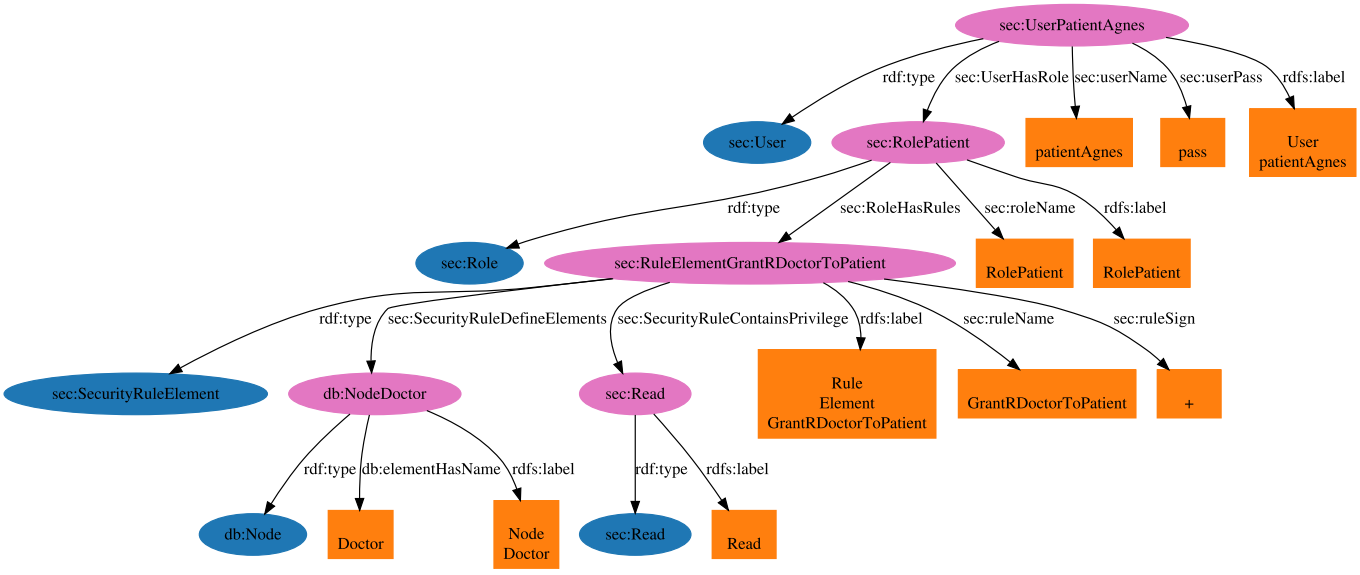


Fig. 12. Fragment of the knowledge graph representing the Patient role.

CRediT authorship contribution statement

Manuel Paneque: Conceptualization, Methodology, Software, Validation, Writing – original draft. **María del Mar Roldán-García:** Supervision, Conceptualization, Methodology, Validation, Writing – original draft. **Carlos Blanco:** Conceptualization, Software, Validation, Writing – original draft. **Alejandro Maté:** Conceptualization, Funding acquisition, Writing – original draft. **David G. Rosado:** Conceptualization, Software, Writing – original draft. **Juan Trujillo:** Conceptualization, Supervision, Funding acquisition, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors are unable or have chosen not to specify which data has been used.

Acknowledgments

This work has been developed within the AETHER-UA (PID2020-112540RB-C43), AETHER-UMA (PID2020-112540RB-C41) and AETHER-UCLM (PID2020-112540RB-C42), ALBA (TED2021-130355B-C31, TED2021-130355B-C33), PRESECREL (PID2021-124502OB-C42) projects funded by the "Ministerio de Ciencia e Innovación, Spain", Andalusian PAIDI program with grant (P18-RT-2799) and the BALLADER Project (PROMETEO/2021/088) funded by the "Consellería de Innovación, Universidades, Ciencia *y* Sociedad Digital", Generalitat Valenciana, Spain. Funding for open access charge: Universidad de Málaga / CBUA.

References

[1] N. Kshetri, Big data's impact on privacy, security and consumer welfare, Telecommun. Policy 38 (11) (2014) 1134–1145.
[2] B. Thuraisingham, Big data security and privacy, in: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, 2015, pp. 279–280.
[3] M. Kantarcioglu, E. Ferrari, Research challenges at the intersection of big data, security and privacy, Front. Big Data 2 (2019) 1, http://dx.doi.org/10.3389/fdata.2019.00001.
[4] A. Benítez-Hidalgo, C. Barba-González, J. García-Nieto, P. Gutiérrez-Moncayo, M. Paneque, A.J. Nebro, M. del Mar Roldán-García, J.F. Aldana-Montes, I. Navas-Delgado, TITAN: A knowledge-based platform for Big Data workflow management, Knowl.-Based Syst. 232 (2021) 107489.
[5] C. Barba-González, J. García-Nieto, M. del Mar Roldán-García, I. Navas-Delgado, A.J. Nebro, J.F. Aldana-Montes, BIGOWL: Knowledge centered big data analytics, Expert Syst. Appl. 115 (2019) 543–556, http://dx.doi.org/10.1016/j.eswa.2018.08.026, URL https://www.sciencedirect.com/science/article/pii/S0957417418305347.
[6] M. Poveda-Villalón, P. Espinoza-Arias, D. Garijo, O. Corcho, Coming to terms with FAIR ontologies, in: C.M. Keet, M. Dumontier (Eds.), Knowledge Engineering and Knowledge Management, Springer International Publishing, Cham, 2020, pp. 255–270.
[7] D. Garijo, O. Corcho, M. Poveda-Villalón, FOOPS!: An ontology pitfall scanner for the FAIR principles, in: International Semantic Web Conference (ISWC) 2021: Posters, Demos, and Industry Tracks, in: CEUR Workshop Proceedings, vol. 2980, CEUR-WS.org, 2021, http://ceur-ws.org/Vol-2980/paper321.pdf.
[8] N.F. Noy, D.L. McGuinness, et al., Ontology Development 101: A Guide to Creating Your First Ontology, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, Stanford, CA, 2001.
[9] N. Guarino, et al., Formal ontology and information systems, in: Proceedings of FOIS, Vol. 98, 1998, pp. 81–97.
[10] B. McBride, The resource description framework (RDF) and its vocabulary description language RDFS, in: Handbook on Ontologies, Springer, 2004, pp. 51–65.
[11] S. Staab, R. Studer, Handbook on Ontologies, Springer Science & Business Media, 2013.
[12] T.R. Gruber, et al., A translation approach to portable ontology specifications, Knowl. Acquis. 5 (2) (1993) 199–220.
[13] D.L. McGuinness, F. Van Harmelen, et al., OWL web ontology language overview, W3C Recomm. 10 (10) (2004) 2004.
[14] W3C OWL Working Group, OWL 2 web ontology language: Document overview, 2019, http://www.w3.org/TR/owl2-overview/. (Online; accessed 5 April 2018).
[15] S. Harris, A. Seaborne, E. Prud'hommeaux, SPARQL 1.1 query language, W3C Recomm. 21 (10) (2013).
[16] E. Prud, A. Seaborne, et al., SPARQL query language for RDF, W3C Recomm. (2006).

[17] A. van den Berghe, R. Scandariato, K. Yskout, W. Joosen, Design notations for secure software: a systematic literature review, Softw. Syst. Model. 16 (2017) 809–831, http://dx.doi.org/10.1007/s10270-015-0486-9.

[18] O. M.Surakhi, A. Hudaib, M. AlShraideh, M. Khanafseh, A survey on design methods for secure software development, Int. J. Comput. Technol. 16 (2017) 7047–7064, http://dx.doi.org/10.24297/ijct.v16i7.6467.

[19] J. Viega, Security in the software development lifecycle: an introduction to {CLASP}, the comprehensive lightweight application security process, Secure Software, Inc., McLean, Virginia, USA, 2005, White Paper.

[20] M. Howard, S. Lipner, The Security Development Lifecycle: Sdl: A Process for Developing Demonstrably more Secure Software, Microsoft Press, 2006, p. 352.

[21] G. McGraw, Software security: building security in, in: Proceedings - International Symposium on Software Reliability Engineering, ISSRE, Addison-Wesley Professional, 2006, p. 6, http://dx.doi.org/10.1109/ISSRE.2006.43.

[22] O. Masmali, O. Badreddin, Model driven security : A systematic mapping study, Softw. Eng. 7 (2019) 30–38.

[23] A. Mashkoor, A. Egyed, R. Wille, Model-driven engineering of safety and security systems: A systematic mapping study, 2020, arXiv preprint arXiv:2004.08471.

[24] H. Olivera, M. Holanda, F. Guimaraes, Data modeling for NoSQL document-oriented databases, in: Annual International Symposium on Information Management and Big Data, Vol. 1478 CEUR Workshop Proceedings, SIMBig, 2015, pp. 129–135.

[25] M. Chevalier, M. El Malki, A. Kopliku, O. Teste, R. Tournier, Implementation of multidimensional databases with document-oriented NoSQL, in: S. Madria, T. Hara (Eds.), Big Data Analytics and Knowledge Discovery: 17th International Conference, DaWaK 2015, Valencia, Spain, September 1-4, 2015, Proceedings, Springer International Publishing, 2015, pp. 379–390, http://dx.doi.org/10.1007/978-3-319-22729-0_29.

[26] M. Chevalier, M.E. Malki, A. Kopliku, O. Teste, R. Tournier, Implementation of multidimensional databases in column-oriented nosql systems, in: M. Tadeusz, P. Valduriez, L. Bellatreche (Eds.), Advances in Databases and Information Systems: 19th East European Conference, ADBIS 2015, Poitiers, France, September 8-11, 2015, Proceedings, Springer International Publishing, 2015, pp. 79–91, http://dx.doi.org/10.1007/978-3-319-23135-8_6.

[27] Y. Li, P. Gu, C. Zhang, Transforming UML class diagrams into hbase based on meta-model, in: Proceedings - 2014 International Conference on Information Science, Electronics and Electrical Engineering, Vol. 2, ISEEE 2014, 2014, pp. 720–724, http://dx.doi.org/10.1109/InfoSEEE.2014.6947760.

[28] D. Ruiz, S. Morales, J. Molina, Inferring versioned schemas from NoSQL databases and its applications, in: International Conference on Conceptual Modeling, ER, 2015, pp. 467–480.

[29] F. Bugiotti, L. Cabibbo, P. Atzeni, R. Torlone, Database design for NoSQL systems, 8824, 2014, pp. 223–231, http://dx.doi.org/10.1007/978-3-319-12206-9_18,

[30] S. Banerjee, A. Sarkar, Modeling NoSQL databases: from conceptual to logical level design, in: 3rd International Conference Applications and Innovations in Mobile Computing, AIMoC 2016, Kolkata, India, February, 2016, pp. 10–12.

[31] D.B. Rawat, R. Doku, M. Garuba, Cybersecurity in big data era: From securing big data to data-driven security, IEEE Trans. Serv. Comput. (2019) 1, http://dx.doi.org/10.1109/tsc.2019.2907247.

[32] N. Gupta, R. Agrawal, Chapter four - NoSQL security, in: P. Raj, G.C. Deka (Eds.), A Deep Dive into NoSQL Databases: The Use Cases and Applications, Vol. 109, Elsevier, 2018, pp. 101–132, http://dx.doi.org/10.1016/bs.adcom.2018.01.003, URL https://www.sciencedirect.com/science/article/pii/S0065245818300032.

[33] B. Saraladevi, N. Pazhaniraja, P.V. Paul, M.S.S. Basha, P. Dhavachelvan, Big data and Hadoop-A study in security perspective, 50, 2015, http://dx.doi.org/10.1016/j.procs.2015.04.091,

[34] I.L. Solsol, H.F. Vargas, G.M. Díaz, Security mechanisms in NoSQL dbms's: A technical review, in: F.R. Narváez, D.F. Vallejo, P.A. Morillo, J.R. Proaño (Eds.), Springer International Publishing, Cham, 2020, pp. 215–228,

[35] D. Pasqualin, G. Souza, E.L. Buratti, E.C. de Almeida, M.D.D. Fabro, D. Weingaertner, A case study of the aggregation query model in read-mostly nosql document stores, in: Proceedings of the 20th International Database Engineering & Applications Symposium on, IDEAS '16, ACM Press, New York, New York, USA, 2016, pp. 224–229, http://dx.doi.org/10.1145/2938503.2938546.

[36] G. Weintraub, E. Gudes, Data integrity verification in column-oriented NoSQL databases, 10980 LNCS, Springer Verlag, 2018, pp. 165–181, http://dx.doi.org/10.1007/978-3-319-95729-6_11.

[37] W. Zugaj, Analysis of standard security features for selected NoSQL systems, Am. J. Inf. Sci. Technol. 3 (2019) 41, http://dx.doi.org/10.11648/j.ajist.20190302.12.

[38] S. Telghamti, L. Derdouri, Towards a trust-based model for access control for graph-oriented databases, in: 2021 International Conference on Theoretical and Applicative Aspects of Computer Science, ICTAACS, 2021, pp. 1–3, http://dx.doi.org/10.1109/ICTAACS53298.2021.9715180.

[39] M. Valzelli., A. Maurino., M. Palmonari., A fine-grained access control model for knowledge graphs, in: Proceedings of the 17th International Joint Conference on e-Business and Telecommunications, SECRYPT, SciTePress, 2020, pp. 595–601, http://dx.doi.org/10.5220/0009833505950601.

[40] C. Morgado, G.B. Baioco, T. Basso, R. Moraes, A security model for access control in graph-oriented databases, in: 2018 IEEE International Conference on Software Quality, Reliability and Security, QRS, 2018, pp. 135–142, http://dx.doi.org/10.1109/QRS.2018.00027.

[41] C. Blanco, D. García-Saiz, D.G. Rosado, A.S.-O. Parra, J. Peral, A. Maté, J. Trujillo, E. Fernández-Medina, Security policies by design in NoSQL document databases., J. Inf. Secur. Appl. 65 (2022) 103120, http://dx.doi.org/10.1016/j.jisa.2022.103120.

[42] A. Maté, J. Peral, J. Trujillo, C. Blanco, D. García-Saiz, E. Fernández-Medina, Improving security in NoSQL document databases through model-driven modernization, Knowl. Inf. Syst. (2021) http://dx.doi.org/10.1007/s10115-021-01589-x.

[43] N. Konstantinou, D.-E. Spanos, N. Mitrou, Ontology and database mapping: A survey of current implementations and future directions, J. Web Eng (JWE) 7 (2008) 1–24.

[44] P. Mayadewi, B. Sitohang, F. Azizah, Scheme mapping for relational database transformation to ontology: A survey, 2017, pp. 1–6, http://dx.doi.org/10.1109/ICODSE.2017.8285866.

[45] C. Ma, B. Molnár, Ontology learning from relational database: Opportunities for semantic information integration, Vietnam J. Comput. Sci. 9 (2022) 31–57, http://dx.doi.org/10.1142/S219688882150024X.

[46] H. Abbes, F. Gargouri, M2Onto: An approach and a tool to learn OWL ontology from MongoDB database, in: A.M. Madureira, A. Abraham, D. Gamboa, P. Novais (Eds.), Intelligent Systems Design and Applications, Springer International Publishing, Cham, 2017, pp. 612–621.

[47] H. Abbes, S. Boukettaya, F. Gargouri, Learning ontology from big data through MongoDB database, in: 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications, AICCSA, 2015, pp. 1–7, http://dx.doi.org/10.1109/AICCSA.2015.7507166.

[48] V.K. Kiran, R. Vijayakumar, Ontology based data integration of NoSQL datastores, in: 2014 9th International Conference on Industrial and Information Systems, ICIIS, 2014, pp. 1–6, http://dx.doi.org/10.1109/ICIINFS.2014.7036545.

[49] S. Ferilli, Integration strategy and tool between formal ontology and graph database technology, Electronics 10 (21) (2021) http://dx.doi.org/10.3390/electronics10212616, URL https://www.mdpi.com/2079-9292/10/21/2616.

[50] N. Fathy, W. Gad, N. Badr, M. Hashem, ProGOMap: Automatic generation of mappings from property graphs to ontologies, IEEE Access 9 (2021) 113100–113116, http://dx.doi.org/10.1109/ACCESS.2021.3104293.

[51] C. Brewster, B. Nouwt, S. Raaijmakers, J. Verhoosel, Ontology-based access control for FAIR data, Data Intell. 2 (1–2) (2020) 66–77, http://dx.doi.org/10.1162/dint_a_00029, arXiv:https://direct.mit.edu/dint/article-pdf/2/1-2/66/1893368/dint_a_00029.pdf.

[52] N. AbdulKadhim, M. Al-Wahah, Semantic-based multi-domain data access authorization, J. Phys. Conf. Ser. 1818 (1) (2021) 012211, http://dx.doi.org/10.1088/1742-6596/1818/1/012211.

[53] F. Rosa, M. Jino, A survey of security assessment ontologies, ISBN: 978-3-319-56535-4, 2017, pp. 166–173, http://dx.doi.org/10.1007/978-3-319-56535-4_17.

[54] R. Ferrini, E. Bertino, Supporting RBAC with xacml+owl, in: Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, SACMAT '09, Association for Computing Machinery, New York, NY, USA, 2009, pp. 145–154, http://dx.doi.org/10.1145/1542207.1542231.

[55] T. Moses, Extensible access control markup language (XACML) version 1, 2005, OASIS Standard.

[56] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, B. Thuraisingham, ROWLBAC: representing role based access control in OWL, in: Proceedings of ACM Symposium on Access Control Models and Technologies, SACMAT, 2008, pp. 73–82, http://dx.doi.org/10.1145/1377836.1377849.

[57] N. Sharma, A. Joshi, Representing attribute based access control policies in OWL, 2016, pp. 333–336, http://dx.doi.org/10.1109/ICSC.2016.16.

[58] N.F. Noy, D.L. McGuinness, Ontology Development 101: A Guide to Creating Your First Ontology, Technical Report, 2001, URL http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html.

[59] Object Management Group, et al., OMG xml metadata interchange (XMI) specification. Version 2.0, 2003, URL https://www.omg.org/spec/XMI/2.0/.