Authorization and Access Control for Different Database Models: Requirements and Current State of the Art

Aya Mohamed[1,2(✉)], Dagmar Auer[1,2], Daniel Hofer[1,2], and Josef Küng[1,2]

[1] Institute for Application-Oriented Knowledge Processing (FAW), Johannes Kepler University (JKU), Linz, Austria
{aya.mohamed,dagmar.auer,daniel.hofer,josef.kueng}@jku.at
[2] LIT Secure and Correct Systems Lab (SCSL), Linz Institute of Technology (LIT), Johannes Kepler University (JKU), Linz, Austria

Abstract. Traditional SQL-based data stores have been the market leaders for decades. However, they have drawbacks with today's massive and highly connected data due to their low flexibility in terms of data structures. NoSQL database models (i.e., key-value, column, document, and graph) are designed for unstructured data in large quantities. However, they currently lack fine-grained dynamic security support, with respect to authorization and access control, in contrast to relational database management systems. We define advanced authorization and access control requirements which are applicable for any database model regardless of the application and access control scenario. According to our discussion on existing access control features versus the requirements in the context of each database model, we conclude whether the requirements are satisfied or not, and provide a corresponding overview.

Keywords: Authorization · Access control · Requirements · Relational database model · NoSQL database models

1 Introduction

Today, NoSQL databases are increasingly used in business and security-critical domains, especially due to their ability to deal with big interconnected data [30]. Although NoSQL database systems have many advantages including scalability and availability, security features, especially authorization and access control to protect sensitive information, were not the primary focus and are not yet considered. Our initial motivation for addressing security issues in this context comes from our research on knowledge graphs, which rely on NoSQL data stores.

Authorization and access control are recognized as the most important security issues in big data [26]. Authorization is the specification of access rights in terms of who (subject) can perform which action on what (resource). Access control is crucial for internal as well as external security in enterprise systems to regulate and check the flow of information. It prevents access to data by unauthorized users.

T. K. Dang et al. (Eds.): FDSE 2022, CCIS 1688, pp. 225–239, 2022.
https://doi.org/10.1007/978-981-19-8069-5_15

In fact, there is no general solution that applies to all database models. This is because each database model has different access control requirements to protect information based on the kind of data (i.e., structured, semi-structured, or unstructured) regardless the underlying data store. Besides, the fine-grained access control (FGAC) solutions developed in the relational database management system (RDBMS) cannot be reused in non-relational data stores due to the schemaless nature of many NoSQL models.

The objective of this work is to guide researchers and practitioners in identifying authorization and access control requirements, features, and limitations of the selected database model. In this literature work, we aim to answer the following research questions:

RQ1 What are the general requirements to apply fine-grained dynamic authorization and access control in databases?
RQ2 Do these general requirements vary with the database model?
RQ3 Which requirements are satisfied by each database model?
RQ4 Are these requirements applicable to typical application scenarios for the various database systems with different models?

The remainder of this paper is organized as follows. We give an overview of related work in Sect. 2. In Sect. 3, we identify the general access control requirements used throughout this work. Furthermore, we discuss the relational and NoSQL database models (i.e., key-value, column, document, and graph) in Sects. 4 to 8 respectively. For each of these models, we give an overview followed by a discussion of the authorization and access control requirements ending with the features that are either supported in databases or published in research works. We then provide an overall discussion in Sect. 9. The paper concludes with a summary in Sect. 10.

2 Related Work

There are currently many survey works in the literature addressing the topic of security in databases, especially the NoSQL database management systems (DBMSs). For example, the work in Sicari et al. [25] is one of the very recent literature researches discussing security and privacy in the context of NoSQL database models selecting one for each category, i.e., Redis, Cassandra, MongoDB, and Neo4j for key-value, column, document, and graph respectively. They compare their security features and considerations with respect to encryption, authentication, authorization, and auditing. They also provide an overview of the RDBMS and compare it with the NoSQL in terms of schema, redundancy, atomicity, consistency, isolation, durability, scalability, and query language.

In Alotaibi et al. [3], access control models in different NoSQL databases are reviewed highlighting the lack of fine-grained access control. Dindoliwala and Morena [13] surveyed several NoSQL databases (i.e., MongoDB, Cassandra, GemStone, db4o, and Objectivity/DB) comparing the existing authentication, authorization, auditing, and data encryption features.

Moreover, the work presented in Zahid et al. [29] performs an assessment to evaluate the security of sharded NoSQL stores in Cassandra, MongoDB, CouchDB, Redis, and HBase. The assessment criteria are authentication, access control, secure configurations, data encryption, and auditing. The security features of the same databases are analyzed again in Dadapeer and Adarsh [12] along with defining the main security issues in NoSQL database.

In Sahafizadeh and Nematbakhsh [23], more NoSQL stores (i.e., HyperTable, Voldemort, DynamoDB, and Neo4j) are included in the comparison besides the five DBMSs used in [12,29]. For each database, a summary of whether the defined security features are supported or not is provided.

However, these works have a broader focus on security-related features with little consideration for authorization and access control. Furthermore, they select the popular NoSQL stores to compare and analyze the existing features. On the contrary, we focus on the database model rather than a specific DBMS.

3 Authorization and Access Control Requirements

Access control systems have to consider specific requirements to provide fine-grained dynamic access control, e.g., *attribute-based access control (ABAC)*. In this section, we define five authorization and access control requirements below.

R1 Authorization policies can be described at different levels of the data model hierarchy.
R2 Fine-grained authorization policies can be defined and enforced based on content.
R3 Custom authorization policies can be defined.
R4 Authorization is supported at different layers.
R5 Context information can be specified in authorization policies.

The significance of requirement *R1* originates from the demand to directly access data at different granularity levels in applications, i.e., from sets of data objects to portions of a single one. Thus, concise authorizations specification and mechanisms to control access at varying granularity levels of data are needed [5].

The requirement *R2* concerns the ability to formulate content-based access rules for specific parts of the data storing unit in a database model. For instance, a document is the unit of storing data in document-oriented databases consisting of fields with keys and values. In this case, only fields satisfying certain criteria specified in the authorization policy should be accessed. Defining and enforcing fine-grained authorization policies in NoSQL data stores is challenging due to the schema-free structure such that different data are stored in one huge database.

Requirement *R3* is about defining rules with custom attributes, roles, or labels based on the implemented access control model rather than only using the built-in ones. Regarding requirement *R4*, the access control system should also be able to support external authorization. External authorization is about specifying and enforcing the authorization policy independent of the application and the underlying DBMS. Generally, external authorization can be handled

either in the application, which is not scalable and hard to maintain, or using policy languages like the *eXtensible Access Control Markup Language (XACML)* as an external decision-making and enforcement mechanism.

In requirement *R5*, the context information in the authorization policy is associated with not only subject users requesting access, but also the resources to be accessed. Context-aware authorization policy addresses the dynamic varying user privileges affected by the frequent changes of context. Although the database model has no impact on the user context (e.g., time, location, and history), not all the database systems, especially non-relational ones, support context-based authorization policies and access control.

The identified requirements are not only implemented differently, but also the definition could vary from one database model to another due to the nature of the data stored in these systems, query language, and structure of the data model. For instance, the data model hierarchy, fine-grained level, and resource context are not the same for all database models. Requirements *R1*, *R2*, and *R5* are also identified as access control requirements for the access control model proposed in Kulkarni [16]. We additionally discuss the proposed requirements in the context of each database model along with the existing access control features in the upcoming sections.

4 Relational Data

The first relational database management system (RDBMS) evolved in the 1970s and is based on the rigid scientific fundamentals of the relational data model developed by Edgar F. Codd [30]. Figure 1 shows the relational model hierarchy. The data are stored in tables, also referred to as relations. The table columns and rows are called attributes and tuples respectively. The relation has a schema (i.e., metadata) which defines the attribute names along with their data type in addition to the instances representing the tuples at a given instant. The RDBMS relies on static schemas for controlling data type to maintain data integrity and evaluates each change to avoid introducing errors or maintenance issues. Furthermore, it supports only structured data and is managed by the common declarative query language *SQL*.

Access control approaches have been developed in relational systems ever since the first products emerged. In SQL'89, *discretionary access control (DAC)* is applied such that the relation creator in an SQL database becomes its owner. However, SQL'89 lacks control over who can create relations. The owner can then give access privileges (i.e., *SELECT*, *INSERT*, and *DELETE*) to other users using the *grant* operation in SQL which applies to base relations and views. The *DROP* relation privilege is not supported in SQL'89, but included in IBM DB2. In addition, the missing revoke operation is provided in SQL'92 to take away the previously granted privileges with the option of cascading revocation to revoke all the grants based on the revoked privilege. In general, DAC is prone to *Trojan Horse* attacks even if the relation access is strictly controlled. For instance, a user with *SELECT* privilege can violate these controls by creating a copy of the

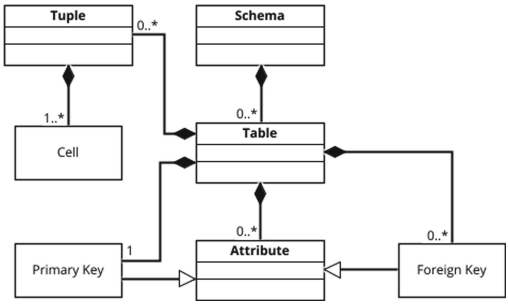

Fig. 1. Relational database model

relation. Another problem is related to access rights management such that the privileges for performing a particular task have to be explicitly granted to each user or group of users [24].

On the other hand, *mandatory access control (MAC)* is based on security labels associated with each data item and each user. Labels on data items and users are called security classification and clearance respectively. In relational databases, security classifications can be assigned to data at different levels of granularity. Assigning labels to entire relations or columns are coarse-grained. The finest granularity is at the tuple or element level. Nevertheless, secret data can be leaked using devious means of communication, i.e., covert channels [24].

Concerning the content-based fine-grained access control in requirement *R2*, there are two categories of enforcement mechanisms in the literature: view-based and query rewriting [9]. In view-based mechanisms, the resource authorized views are derived according to the specified authorization policy. Then, the access is granted to these views rather than the original data resource. On the other hand, the query rewriting enforcement approach intercepts the query submitted for execution to apply the authorization policy criteria.

In Bertino et al. [4], an access control model for relational databases supporting permission delegation and negative authorization is proposed. Moreover, *Oracle Virtual Private Database (VPD)* [7] enforces access control at the row level by appending the expressed content-based and context-based conditions in the authorization policy to the where clause of the SQL query. A FGAC at the cell level is introduced LeFevre et al. [17] using dynamically generated views nullifying the unauthorized cells. Last but not least, Agrawal et al. [2] proposed a language supporting grant command specification at the cell level [9].

The existing authorization and access control mechanisms are advanced and satisfy all the requirements. However, as the performance of relational databases degrades with joins, locks and impedance mismatch, non-relational databases emerged with various data storage models to address these limitations and handle large amounts of data [13]. In the following sections, we will explain each of the NoSQL models in detail.

5 Key-Value Data

The key-value model uses a hash table, and is the simplest one among all NoSQL models. It is powerful and efficient in storing schema-less data in the form of data values associated to keys which are used as indexes for quickly finding values in large data sets [3]. Data can be either stored as rows like structured data or JSON objects. *Redis*[1] and *Accumulo* are examples of native key-value databases.

Redis is an advanced open source data store where each key-value is a pair of binary strings for managing different types of binary data (e.g., XML documents, images, arrays, and bytes). It provides hashes to store and query the database objects. In Accumulo, data is stored in a distributed sorted map. The keys are logically divided into a row key to uniquely identify the row, a column, and an automatically generated timestamp used for versioning [19]. Each column is further divided into a family (i.e., the logical grouping of the key), a qualifier as a more specific key attribute, and a visibility tag which stores a logical combination of security labels.

According to the key-value model represented in Fig. 2, the hierarchy consists of a collection of records identified by their unique keys (cp. *R1*). Regarding requirement *R2*, the finest granularity in this model is the value of a particular key. If the value is an object with attributed values in the form of key-value pairs, access control should be applied at the field level.

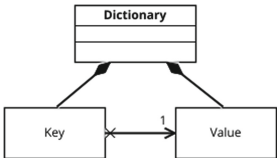

Fig. 2. Key-value database model

The Accumulo database applies fine-grained access control at the cell level such that security labels are assigned to key-value pairs as a new element to the key called *Column Visibility*. If these labels are satisfied at query time, the respective key and value will be included in the response of the user access request. On the contrary, Redis does not implement any access control mechanism [25]. The work in Moreno et al. [19] proposed a model design to describe who can access the values of specific data cells in a key-value database system. The labels may define rules for an access matrix, role-based access control (RBAC), or multilevel models.

[1] https://redis.io.

When matching the requirements in Sect. 3 with the available access control features, we find that the requirement *R1* is satisfied because the model hierarchy is simple (i.e., a table and key-value entries). Requirement *R2* is not satisfied for this model as the access to the fine-grained element is based on its security label or the assigned role rather than the content. We consider the key-value database model satisfying requirement *R3* because at least one database system has the option to specify custom authorization policies. For instance, custom security labels can be specified in the column visibility or each value during writing to Accumulo. Finally, no demonstration cases are presented for external authorization and context-based policy in key-value databases so far.

6 Column-Oriented Data

The column-oriented database is also referred to as column-family store. It is considered as an evolution of key-value stores where data are also represented as hash maps, but with more than one indexing level [1]. The meta model for column-oriented databases is illustrated in Fig. 3. Each column consists of a key and a value. The column-family is a set of rows equivalent to a table in the relational databases. A set of column-family is defined as a key space. This model is typically used in data mining and web applications because of its ability to deal with massive data and complex datasets in distributed systems. However, it is less flexible than key-value and document-oriented models because of the column-family that must define a schema at the application level. Examples of column-oriented databases are *Cassandra*[2] and *HBase*.

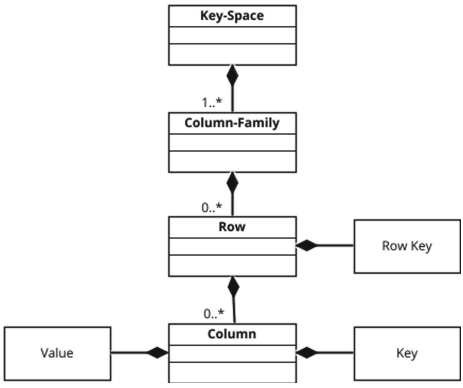

Fig. 3. Column-oriented database model

According to the column-oriented database model and requirement *R1*, authorization policies should specify constraints at the level of key space, column-family, row, and column. For requirement *R2*, FGAC should allow to limit access to a specific column values within a row.

[2] https://cassandra.apache.org.

Cassandra supports RBAC at the key space or column-family level according to the user role(s) and privileges. It also uses the GRANT/REVOKE security paradigm to manage permissions on database resources, which are assigned to roles [13]. The resource could be key space, role, table, index, or function. In Cassandra, access control at the object level is not available [12]. HBase enforces authorization using access control list (ACL). Kulkarni [16] proposed a fine-grained key-value access control (K-VAC) model where authorization policies can be specified at the level of column-family, key space, column, or row. However, this model is implemented as a library and restricted to specific databases, i.e., Cassandra and HBase.

Based on the research works and available access control features in column databases, it is possible to specify custom authorization policies at different levels (refer to *R3*). Additionally, the work in Kulkarni [16] claims to enforce content-based FGAC. It also provides three examples to apply context-based access control (*R5*) by describing user location and time of the day in the authorization policy. Since existing policy languages have no direct support for column-oriented data structures, this database model fails to meet requirement *R4*.

7 Document-Based Data

The document-based database model is the most commonly used NoSQL data model as it can manage structured, semi-structured (e.g., XML files), and unstructured (i.e., text) data. In this model, data is stored as schema-less documents with one or more fields as key-value pairs or nested documents as depicted in Fig. 4. Documents are analogous to records in the RDBMS and the term collection is equivalent to table, but without a pre-defined schema. Each document is identified by a unique key which is not only used to manipulate (i.e., insert, delete, and update) document data, but also for linking different documents. For fast data retrieval, indexing on specified fields can be added. Document data stores are typically used for blog software and content management systems due to their flexibility, high performance, and horizontal scalability. Examples are *MongoDB* and *Couchbase*.

MongoDB is a distributed general-purpose database that stores data in the form of JSON documents without schema definition. It has collections as an additional organization level for grouping similar documents and provides its own query language, i.e., *MongoDB Query Language (MQL)*. MongoDB is the first ranked NoSQL database due to its strengths including the support of all indexing techniques in relational databases for data sorting and faster searching [25].

The finest granularity in this database model is at the level of document fields (cp. *R1*), however, authorization policies should consider the rest of the hierarchy including document, collection, and up to the database level. Field-level policies control access to fields of a document with any structure level including a field of a document field (i.e., nested document) or an element of an array of fields.

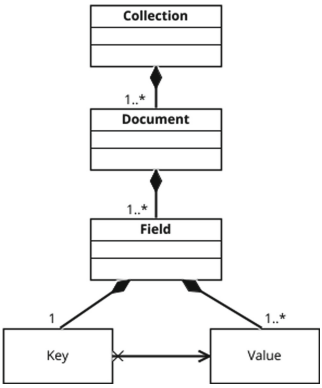

Fig. 4. Document-based database model

Document data stores support *role-based access control (RBAC)*. Couchbase implements 46 predefined roles with specific privileges on the entire collection[3]. Most of these roles are exclusive to the enterprise version; only three roles can be used in the community version. MongoDB has built-in and user-defined roles that grant privileges to perform the specified actions on a given resource (i.e., database, collection, set of collections, or cluster). Users have no access to the system if they are not assigned to at least one role. The first user created in the database should be a user administrator who has the privileges to manage other users[4]. Although MongoDB is adopted in many solutions due to its dynamic structure, there is no standardization of authorization and access control.

In Colombo and Ferrari [8], the RBAC model in MongoDB is enhanced to support purpose-based policy specification at the document level. However, the proposed approach is limited to MongoDB. Then, the same authors refined the granularity level of access control in MongoDB to support content-based and context-based policies at the field level [10]. They eventually generalized the concept to enforce fine-grained ABAC into document data stores at the document or field level without prior knowledge of the document structure [11]. In Kacimi and Benhlima [14], the work presents an architecture applied to MongoDB for purpose-based access control policies written in XACML.

According to the existing access control features for the document-based database model, we can say that at least one document data store satisfies each of the proposed requirements except *R4*. Although the work in [14] uses XACML in *access control as a service (ACCAAS)* with MongoDB, further implementation is needed to map the defined policy attributes to database values. Hence, the solution is application-specific, but it uses a policy language to express the authorization policy and decide whether the access request is authorized or not.

[3] https://docs.couchbase.com/server/current/rest-api/rbac.html.
[4] https://www.mongodb.com/docs/manual/core/authorization.

8 Graph-Structured Data

The data is stored in the form of graphs having object nodes as vertices connected by edges representing the relationships between them. A graph database has no predefined schema and can be seen as a special kind of document-oriented database where some documents act as the relationships connecting other documents [25]. It is scalable and uses shortest path algorithms for improving the efficiency of data queries, but is more complex to manage. There are different graph models, but the *property graph* (see Fig. 5) is the most common model in graph databases.

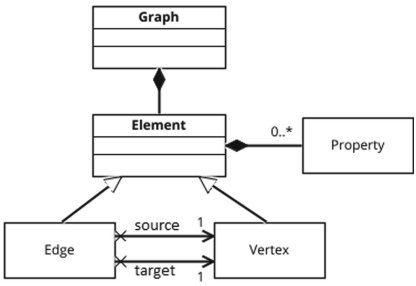

Fig. 5. Property graph database model

Graph databases are mainly used in recommendation systems and social networks, however, there is no standard language for inserting data and traversing graphs yet. Neo4j [21] is the top ranked native graph database[5] with its declarative query language *Cypher*.

Fine-grained access control in the context of graph-structured data is about protecting nodes and edges along with their properties (i.e., attributes). However, one of the key problems is how to describe the object of a permission [15]. This is because usually nodes and edges are not isolated and even contexts in the graph model can be used as an object of a permission, so protecting a single element ignoring the related surroundings results with allowing access to specific subjects although their connecting path could be unauthorized. In some cases, it is also desired to permit/deny specific connections to a particular object. Hence, a way to describe paths or subgraphs to which access rights apply is required. Another problem concerns permission propagation among objects.

Currently, native (e.g., Neo4j) and non-native (e.g., Microsoft Azure CosmosDB [28] and ArangoDB [22]) graph databases provide RBAC. Neo4j supports RBAC with predefined roles (i.e., reader, editor, publisher, architect, and admin) in addition to subgraph and property-level access control [6]. It also has a special database (i.e., system database) for storing the defined privileges.

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

A model-based approach is introduced in Morgado et al. [20] using metadata with authorization rules to control access in applications that use graph-oriented databases. It provides a predefined schema for the graph nodes and supports data definition language (DDL) and data manipulation language (DML) operations. This model only allows the specification of positive permissions that have to be defined for each node. The work did not show how the model handles conflicts. In Valzelli et al. [27], the authors proposed an initial solution towards protecting knowledge graphs. A knowledge graph contains all the world's main entities along with their relations. The work introduces a property graph model to specify open and closed policies using authorization edges between subjects (i.e., user and user group) and resources (or resource category). However, they focus only on DAC, MAC, and RBAC which are not sufficient to enforce FGAC. These works provide conceptual approaches that need to be implemented on top of the graph model.

Last but not least, the work in Mohamed et al. [18] tries to couple ABAC with a new declarative language for fine-grained, attribute-based authorization policy, named *XACML for Graph-structured data (XACML4G)* [25]. Even though additional path-specific constraints in terms of graph patterns can be described, the policy rules require specialized processing and the enforcement mechanism needs to be adapted to work in a specific graph data store.

Current access control in graph databases does not meet the requirement *R1*. For instance, the RBAC privileges in Neo4j are limited to reading/updating the database, managing resources (i.e., databases, users, roles, and privileges) as well as editing node labels, relationship types, and property names. Up to the best of our knowledge, content-based FGAC cannot be applied in the existing graph databases (*R2*). The achieved authorization in the enterprise edition of Neo4j is specifying privileges using static commands in terms of actions to be performed (e.g., traverse, read, and match) on particular node labels or relationship types within graphs. These privileges are then granted or denied to custom-defined roles. Requirement *R4* is satisfied since authorization policies can be specified and enforced in the database layer as well as externally (refer to [18]). Finally, existing graph databases and even the recent policy language *XACML4G* did not show rule specification and enforcement taking user context information, e.g., access time and location of the user, into account.

9 Discussion

In the previous sections, we provide the current state of the art of authorization and access control features supported within DBMSs or introduced by research works in relational as well as NoSQL database models. We now relate the requirements defined in Sect. 3 with the previously discussed features taking into consideration that the data model hierarchy in requirement *R1* is structured differently and the term FGAC in requirement *R2* is defined differently for each database model (see *RQ2*).

The relational model has the most sophisticated authorization and access control mechanisms in comparison to all NoSQL models. However, it is not scalable to deal with big interconnected data. The fine-grained content-based model

ABAC can be enforced in the access control systems for relational databases taking environmental conditions into account. Moreover, custom rules can be specified at different levels within the database (e.g., using views) or externally using a policy language such as XACML.

As opposed to relational databases, NoSQL databases trade consistency and security for performance and scalability [13]. The access control approaches available in the literature are specific to certain database models or even data stores. This is due to the lack of a reference model and multiple implementations of the same data model [9]. To address *RQ3*, we summarize the assessment of the requirements for each database model in Table 1.

Table 1. Requirements for each database model

Database model	R1	R2	R3	R4	R5
Relational	✓	✓	✓	✓	✓
Key-Value	✓	×	✓	×	×
Column	✓	✓	✓	×	✓
Document	✓	✓	✓	×	✓
Graph	×	×	✓	✓	×

The key-value model is the simplest with respect to structure and hierarchy, however, it has the least information security support, i.e., basic authorization using labelling, compared to the column, document, and graph models. The column-oriented and document-structured database models received the most attention among the NoSQL ones. The existing access control features together with the research works addressed most of the authorization and access control requirements. It is still challenging to meet the missing requirements and come up with an authorization policy language and enforcement model that fits within one or more non-relational database models. This is because of the lack of a common query language and consistent support that made it hard to switch from one NoSQL data store to another.

Regarding access control for graph databases, Neo4j is more advanced than other graph databases. However, the existing access control features still need to be enhanced to not only provide fine-grained access control for nodes and relationships on the attributes level, but also protecting the graph while traversing it. The proposed research works have two main drawbacks: (1) not generally applicable because graph data stores have different query languages, and (2) extra implementation is required upon changing or adding new policies.

For any application, there is an underlying database within a DBMS either having a specific database model or implementing several ones. For example, ArangoDB is a native multi-model database managing different data models (i.e., key-value, document, or graph) with one declarative query language. There is no access control solution that applies to all database models so far, but it is

possible to enforce an access control approach in different DBMSs with a specific model due to similar structure or a specific DBMS with different models because of the common query language. To answer *RQ4*, the defined requirements are generally applicable to typical application scenarios for various DBMSs even with different database models if structural differences are taken into account.

10 Conclusion

Access control ensures information security and protection by enforcing authorizations in terms of which users are permitted (or denied) to perform what operations on which organizational resources. Authorization and access control are open issues in the NoSQL data stores because these database models (i.e., key-value, column, document, and graph) are designed to focus on handling new data sets with less consideration on security. Unlike NoSQL models, the relational model has robust access control mechanisms to protect sensitive information. However, RDBMSs are inefficient in storing and handling big data.

The traditional relational model data is structured into tables with fixed schema where each data entry is equivalent to a row having values for the columns. On the other hand, non-relational models have different forms. Firstly, the key-value model is represented as a hash table with key-value entries. In the column-oriented model, records hold a collection of dynamic columns that are grouped into column families within a key space. The document-based model consists of collections having document entries in the form of key-value pairs or nested documents whereas the graph data structure is solely based on vertices and edges. In this paper, we address authorization and access control, with respect to requirements and features within existing DBMSs or research works, for all database models.

To answer our research questions, we start with defining five requirements to generally apply fine-grained dynamic authorization and access control for different database models regardless of the application scenario (refer to *RQ1* and *RQ4*). Due to the different hierarchy and fine-grained access control (FGAC) definition for each database model in requirements *R1* and *R2* respectively, we discuss them in the context of these database models along with an overview including the meta model structure, authorization and access control features ending with a summary matching the state-of-the-art features with the requirements (*RQ2*). The resource context in requirement *R5* is also different for some data structures (e.g., graph).

According to our results in Table 1, we indicate whether the requirements are satisfied or not for the relational as well as each of the NoSQL database models (*RQ3*). It can be concluded that the relational database model has advanced authorization and access control features while with the NoSQL models, we can only specify custom authorization policies on different levels, except for the graph model. There it is not yet possible to define policies for individual elements and their properties. However, a policy language for specific NoSQL models is only proposed for graph databases. Last but not least, FGAC based on content and context are currently supported by document and column models only.

As NoSQL data stores are increasingly used today, a lot of research and development have already been focusing on providing more sophisticated authorization and access control features. Still more research is needed for the NoSQL models to achieve a similar maturity level for authorization and access control to that in the relational model.

Acknowledgement. The research reported in this paper has been partly supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria. The work was also funded within the FFG BRIDGE project KnoP-2D (grant no. 871299).

References

1. Abadi, D.J., Boncz, P.A., Harizopoulos, S.: Column-oriented database systems. Proc. VLDB Endow. **2**(2), 1664–1665 (2009). https://doi.org/10.14778/1687553.1687625
2. Agrawal, R., Bird, P., Grandison, T., Kiernan, J., Logan, S., Rjaibi, W.: Extending relational database systems to automatically enforce privacy policies. In: 21st International Conference on Data Engineering (ICDE 2005), pp. 1013–1022 (2005). https://doi.org/10.1109/ICDE.2005.64
3. Alotaibi, A., Alotaibi, R., Hamza, N.: Access control models in NoSQL databases: an overview. JKAU **8**(1), 1–9 (2019)
4. Bertino, E., Samarati, P., Jajodia, S.: An extended authorization model for relational databases. IEEE Trans. Knowl. Data Eng. **9**(1), 85–101 (1997). https://doi.org/10.1109/69.567051
5. Bertino, E., Ghinita, G., Kamra, A.: Access Control for Databases: Concepts and Systems. Now Publishers Inc. (2011)
6. Borojevic, I.: Role-based access control in Neo4j enterprise edition (2017). https://neo4j.com/blog/role-based-access-control-neo4j-enterprise. Accessed Aug 2022
7. Browder, K., Davidson, M.A.: The virtual private database in oracle9ir2. Oracle Technical White Paper, Oracle Corporation 500(280) (2002)
8. Colombo, P., Ferrari, E.: Enhancing MongoDB with purpose-based access control. IEEE Trans. Dependable Secure Comput. **14**(6), 591–604 (2015). https://doi.org/10.1109/TDSC.2015.2497680
9. Colombo, P., Ferrari, E.: Fine-grained access control within NoSQL document-oriented datastores. Data Sci. Eng. **1**(3), 127–138 (2016)
10. Colombo, P., Ferrari, E.: Towards virtual private NoSQL datastores. In: 2016 IEEE 32nd International Conference on Data Engineering (ICDE), pp. 193–204 (2016). https://doi.org/10.1109/ICDE.2016.7498240
11. Colombo, P., Ferrari, E.: Towards a unifying attribute based access control approach for NoSQL datastores. In: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), pp. 709–720 (2017). https://doi.org/10.1109/ICDE.2017.123
12. Dadapeer, N.I., Adarsh, G.: A survey on security of NoSQL databases. Int. J. Innovative Res. Comput. Commun. Eng. **4**(4), 5250–5254 (2016)
13. Dindoliwala, V.J., Morena, R.D.: Survey on security mechanisms in NoSQL databases. Int. J. Adv. Res. CS **8**(5) (2017)
14. Kacimi, Z., Benhlima, L.: XACML policies into MongoDB for privacy access control. In: Proceedings of the Mediterranean Symposium on Smart City Application, SCAMS 2017. Association for Computing Machinery, New York (2017). https://doi.org/10.1145/3175628.3175646

15. Kalajainen, T., et al.: An access control model in a semantic data structure: case process modelling of a bleaching line. Department of CS and Engineering (2007)
16. Kulkarni, D.: A fine-grained access control model for key-value systems. In: Proceedings of the Third ACM Conference on Data and Application Security and Privacy, CODASPY 2013, pp. 161–164. Association for Computing Machinery, New York (2013). https://doi.org/10.1145/2435349.2435370
17. LeFevre, K., Agrawal, R., Ercegovac, V., Ramakrishnan, R., Xu, Y., DeWitt, D.: Limiting disclosure in hippocratic databases. In: 30th International Conference on Very Large Databases, VLDB Endowment, Toronto, Canada, pp. 108–119 (2004)
18. Mohamed, A., Auer, D., Hofer, D., Küng, J.: Extended authorization policy for graph-structured data. SN Comput. Sci. **2**(5), 1–18 (2021)
19. Moreno, J., Fernandez, E.B., Fernandez-Medina, E., Serrano, M.A.: A security pattern for key-value NoSQL database authorization. In: Proceedings of the 23rd European Conference on Pattern Languages of Programs, EuroPLoP 2018. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3282308.3282321
20. Morgado, C., Busichia Baioco, G., Basso, T., Moraes, R.: A security model for access control in graph-oriented databases. In: 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 135–142 (2018). https://doi.org/10.1109/QRS.2018.00027
21. Neo4j: Neo4j documentation (2022). https://neo4j.com/docs/. Accessed Aug 2022
22. Oasis: Access control in ArangoDB (2019). https://www.arangodb.com/docs/stable/oasis/access-control.html. Accessed Aug 2022
23. Sahafizadeh, E., Nematbakhsh, M.A.: A survey on security issues in big data and NoSQL. Adv. Comput. Sci. Int. J. **4**(4), 68–72 (2015)
24. Sandhu, R.: Relational database access controls. Handb. Inf. Secur. Manag. **95**, 145–160 (1994)
25. Sicari, S., Rizzardi, A., Coen-Porisini, A.: Security&privacy issues and challenges in NoSQL databases. Comput. Netw. **206**, 108828 (2022). https://doi.org/10.1016/j.comnet.2022.108828
26. Tankard, C.: Big data security. Netw. Secur. **2012**(7), 5–8 (2012). https://doi.org/10.1016/S1353-4858(12)70063-6
27. Valzelli, M., Maurino, A., Palmonari, M., Spahiu, B.: Towards an access control model for knowledge graphs (2021)
28. Weiss, T., et al.: Azure role-based access control in azure cosmos DB (2022). https://docs.microsoft.com/en-us/azure/cosmos-db/role-based-access-control. Accessed Aug 2022
29. Zahid, A., Masood, R., Shibli, M.A.: Security of sharded NoSQL databases: a comparative analysis. In: 2014 Conference on Information Assurance and Cyber Security (CIACS), pp. 1–8 (2014). https://doi.org/10.1109/CIACS.2014.6861323
30. Zugaj, W., Beichler, A.: Analysis of standard security features for selected NoSQL systems. Am. J. Inf. Sci. Technol. **3**(2), 41–49 (2019)