A Security Model for Access Control in Graph-Oriented Databases

Claudia Morgado∗†, Gisele Busichia Baioco∗, Tania Basso*, Regina Moraes∗
∗ University of Campinas, Limeira, Brazil
gisele, taniabasso, regina {@ft.unicamp.br}
† Center for Research and Development in Telecommunications, Campinas, Brazil
cmorgado@cpqd.com.br

Abstract—Nowadays, organizations collect vast amounts of data for future analysis. Motivated by this amount of data and requirements of Web2.0, a plethora of non-relational databases (NoSQL) emerged in recent years. However, several security features in relational databases (e.g., access control) have been left in non-relational management systems to be developed by the application, which can raise security breaches. This paper proposes a security model, based on the use of metadata, to provide access control for NoSQL graph-oriented database management system. The goal is to support the development of applications that use graph-oriented database in preserving the integrity of stored data and protect them from non-authorized access. A case study was performed as proof of concept, where the model was instantiated and implemented for Neo4j database. Results showed that access restrictions were applied correctly, avoiding unauthorized access. A schema for Neo4j was provided, once it does not have a native one.

Keywords—Access Control, Graph-oriented Database, NoSQL Database, Security.

I. INTRODUCTION

Relational databases have been used for many decades. However, the recent advance in cloud computing and distributed web applications has created the need to store large amount of data in distributed databases that provide high availability and scalability. In recent years, an increasing number of companies have been adopting various types of the so-called non-relational databases, commonly known as NoSQL databases.

NoSQL databases take different approaches: wide-column stores (e.g., Cassandra); document stores (e.g., MongoDB); key-value stores (e.g. BerkeleyDB); and graph databases (e.g., Neo4j). Their primary advantage is that they handle unstructured data such as documents, e-mail, multimedia and social media efficiently. Although this type of database can handle high volumes of personal and sensitive information, up to now the majority of these systems provide poor privacy and security protection [1]. According to Zahid et al. [2], NoSQL databases were initially not designed by considering security as an important feature. Therefore, it has become the sole responsibility of NoSQL consumers to protect these databases by using third party tools and services.

Access control mechanisms ensure user confidentiality and provide data integrity, preventing data modifications by illegitimate users. While relational management systems implement their own access control mechanisms, non-relational management systems let them to be developed by applications.

This paper presents a security model for access control in NoSQL Graph-Oriented Databases Management System (GDBMS). The goal of the model is to guide the implementation of access control to graph-oriented databases and facilitate the development of different applications with data security, allowing them to implement their own access control when using this type of database. The model provides an architecture with authorization rules to control data access, similar to the one used by relational databases. It is based on metadata and supports DDL and DML operations.

As a case study, the proposed model was implemented for access control in the GDBMS Neo4j [3]. It was applied in a newspaper’s News website, where some authorized users can describe a short news report and give a title to it, and other users can comment on this news. The exclusion of comments is also controlled.

The paper brings the following contributions: (i) a model to guide the development, standardization, and evolution of access control systems for GDBMS; (ii) support to DDL and DML operations; (iii) a schema for graph-oriented database, based on structure of nodes; (iv) an access control mechanism for Neo4j; (v) a GDBMS catalog. The model and its respective mechanism contributed to a more protected GDBMS, avoiding unauthorized access to stored information. Furthermore, data become more consistent and homogeneous due to the predefined schema of nodes. Different user profiles have different access permissions and graph-like data visualization.

The paper is organized as follows. Section 2 introduces the background and related work. Section 3 presents the access control model and an example of its instance. Section 4 presents the case study, the results and discussions. Finally, Section 5 shows the conclusions and future works.

II. BACKGROUND AND RELATED WORK

To the best of our knowledge, there are no previous work that proposes access control models for graph-oriented databases. The access control must be done by the applications that use this type of database. Few works proposed access control models for other types of NoSQL databases.
Shermin [4] proposed a context-aware RBAC (Role-Based Access Control) model for NoSQL databases. However, it is not analyzed how the model can be implemented and integrated into a target NoSQL platform. Colombo and Ferrari [5] proposed an approach for the enhancement of the MongoDB RBAC model, including privacy concepts. However, the focus of this work is specifically the MongoDB, which is a document-oriented NoSQL database. Our work proposes a model for graph-oriented databases and the goal of the model is to guide the implementation of access control mechanisms for different databases in this category, not one in particular.

Accumulo [6] is a key-value store that supports cell-based access control policies where a cell represents a row and a column combination. In this same direction, Kulkarni [7] proposed an access control model which is integrated in the source code of Cassandra. Besides being specific for Cassandra database, this model is still limited because the permissions are only for reading and writing. Our model is more complete, providing permissions as alter and delete, as well as creation of structure of nodes.

Calil and Mello [8] proposed SimpleSQL, a specific solution for mapping a relational schema and some relational operations to SimpleDB, a document-oriented database. The solution is able to perform operations insert, update, delete and select. The fact of being based on few rules and having only operations of DML constitutes a limitation for this work. Furthermore, it was defined to document-oriented databases. Our model provides also operations of DDL and the definition of a schema for graph-oriented databases, based on structure of nodes. So, complementarily to described previous work ([5] [6] [7] [8]), the access control model presented in this article provides permissions based on groups. Users can be associated to groups and the permissions can be granted only once, to the user group.

III. THE SECURITY MODEL FOR ACCESS CONTROL

The goal of access control model we propose is to assure that only authorized users can access or modify the data managed by graph-oriented database management system (GDBMS). The idea is to create graphs as instances of the model in order to implement access control for the database. We based on the access control model of relational databases and adapted it to a graph context, using metadata. Figure 1 shows the proposed model.

We used the RDF (Resource Description Framework) language for representing the model because it is recommended by W3C and was originally designated to represent metadata [9]. The definition of the structure of node is given by the triple G = {N,L,A}, where N represents the node; L represents the identification of the type of the node (i.e., its label); and A is the collection of attributes of the node.

The model started with the need for creating user accounts and grant or revoke privileges for individual or group of users. So, similar to the access control models in relational DBMS, a Database Administrator (DBA) (also referred as Master) was defined.

The meta-node Meta-User represents the users of the system. Their data will be used to validate the permissions these users have in the access control to the GDBMS. The Meta-User is composed of the attributes Identifier (identifier of the user); Name (name of the user), Password (password of the user), Date (date of the creation of the user in the database), and Owner (user previously registered in the database who created the new user).

To facilitate the privilege management, we defined metadata for user groups. The meta-node Meta-Group represents a group of users that have the same permissions in the GDBMS. Their attributes are Identifier (identifier of the group) and Name (name of the group). So that a user can inherit the privileges of a user group, it must be associated to this group. This is represented through the Attachment relationship, which relates the Meta-User to the Meta-Group.

Still based on relational access control models, the operations of DDL and DML are represented by the meta-node Meta-Operation. It represents the types of permissions the users can have in the GDBMS. Meta-Operation is composed of the attributes Identifier (identifier of the operation) and Type (type of the operation, which can be Create, Alter, Drop - DDL operations, and Insert, Update, Delete, Select - DML operations).

The relationship between the meta-nodes Meta-User and Meta-Operation, called Permission_User in the model, represents the granting of permissions to users. When the model is instantiated, the nodes of the graph are related through edges in order to associate users to operations they are allowed to perform in the database. To revoke privileges, it is necessary remove the relationship (the edge in the instance). Similarly, the granting of privileges to the group is represented by the relationship Permission_Group, which associates Meta-Group with Meta-Operation.

Finally, it is necessary to represent the data to be persisted in the GDBMS. This is done through the meta-node Meta-Node. Similar to the relational model, this node represents the set of records in the database (also referred as tuples), i.e., a complete set of information (e.g., in a bookstore system, a meta-node could represent books, customers, orders). The attributes of Meta-Node are Label (identifies the label of the record, represented as instantiated nodes) and Owner (user who is responsible for the creation of the node - this will be discussed with more details later in this section).

In the relational model, records are composed of fields, each of which contains one item of information. Similarly, in our model, the record’s fields are represented by the meta-node Meta-Property (e.g., in a bookstore system, books records could have the fields title, author, publisher). Meta-Property is composed of the attributes Identifier (identifier
of the property), Name (name of the property), Type (data type of the property) and Mandatory (if the property is mandatory or not). The relationship Property associates the Meta-Node and Meta-Property meta-nodes. It relates the data, defining which properties belong to the node structure being instantiated.

The relationship Operation, which relates the Meta-Operation with the Meta-Node, establishes the permission a user has to manipulate certain node structure. It means that the permissions of the operation (Meta-Operation) granted to the user (Meta-User) or user group (Meta-Group) are associated to the node (Meta-Node). As Meta-Operation supports DDL and DML operations, from this association (Meta-User/Meta-Operation/Meta-Node or Meta-Group/Meta-Operation/Meta-Node), it is established that: (i) to create new structure of nodes, the user or group must be related to the operation, already predefined, with Type as Create. The user or group becomes the owner of this structure and gets all the permissions for manipulation of this node; (ii) to exclude a structure of nodes, the user or group must have permission of type Drop. Also, there can be no node from this structure instantiated in GDBMS; (iii) to modify the properties of the node structure the user or group must have permission of type Alter. A property cannot become mandatory, or have its data type modified, if there already exist nodes as part of the structure instantiated in the GDBMS; (iv) to instantiate nodes from particular structures, the user or group must have permission of type Insert associated to the structure of node; (v) to exclude a instantiated node, the user or group must have permission of type Delete; (vi) to modify instantiated nodes, the user or group must have permission of type Update and the restrictions about properties must be respected; (vii) the selection of instantiated nodes are related to permission of type Select.

It is important to mention that the Meta-User is not related directly to the Meta-Node because this relationship would give user the full control over the node, which is not desired. The idea is that users and groups have different permissions for different node structures. It means that several instances of Meta-Operation with the same Type (e.g., Insert) could be available in the graph, and each of this instance is related with its respective Meta-User/Meta-Group and Meta-Node instances (see Figure 2, the OPERATION nodes with identifiers 1 and 3).

It is also important to mention that, as the Meta-User is not related directly to the Meta-Node, we defined the attribute Owner to the Meta-Node to represent the user who created the node. Similar to denormalization in the relational model, this attribute was defined to guarantee the efficiency in the access rights over the node (full access), avoiding the cost to traversing the graph.

The instantiation of the model will form the structure of nodes of the GDBMS. This structure is the graph which represents the access control to information in the database. The nodes of the graph are instances of the model’s meta-nodes and the edges are instances of the model’s relationships. The data access restrictions are enforced according to the permissions established by the nodes structure, i.e., a user can have permissions to access certain structure of node and not have permissions to access other structure.

Figure 2 shows an example of instance of the access control model. The nodes are defined in capital letters according to the metadata (USER for users, OPERATION for operations, NODE for the structure of nodes, and PROPERTY for properties defined to the structure).

This example of instance represents the permissions a user has over a structure of node representing information of
books. So, two users are represented as USER node: User2 and User3. User2 has the attributes 2 (identifier), User2 (name), 123 (password), 06/11/2015 (date of creation), DBA (owner). User3 has the attributes 3 (identifier), User3 (name), 222 (password), 03/03/2016 (date of creation), DBA (owner).

The permissions are represented by the nodes with the label OPERATION. So, the first operation has the attributes 1 (identifier) and Insert (type). The second operation has the attributes 2 (identifier) and Alter (type). User2 has two access permissions associated to the schema of nodes. The both edges called Permission, which associates User2 with respective OPERATION nodes, represent the operations User2 can perform: Insert allows instantiating nodes from the structure and Alter allows modifying properties from the structure of nodes. Also, there is another OPERATION node for insert permission. It has the attributes 3 (identifier) and Insert (type). User3 is associated to this operation through the edge Permission. It means that the User3 has permission only for instantiating nodes from the structure. He/she cannot alter the structure of nodes.

The graph in the example also has the nodes with label NODE, which represents book and publisher information. The NODE representing Book has the attributes Book (la-
bel) and User1 (owner), where User1 is a user previously registered in the model and not represented here for sake of organization. Also, this node is composed of two properties, represented by the nodes with label PROPERTY and with the attributes 1 (identifier), Title (name), String (type), Y (mandatory) and attributes 2 (identifier), Author (name), String (type), N (mandatory). The NO\-DE representing Publisher has the attributes Publisher (label) and User1 (owner) and the PROPERTY with the attributes 1 (identifier), Name (name), String (type), Y (mandatory). The edges called Operation represent the access restrictions. In this case, they represent that new instance of nodes (Book) can be inserted (through operation Insert) and the structure of nodes can be modified (through operation Alter). Also, new instances of nodes (Publisher) can be inserted in the structure (but publisher nodes cannot be modified because there is no Operation edge connecting this node with the Alter operation).

Only users associated to the operations have the respective permissions. In Figure 2, User2 can insert instances and modify nodes in the structure because it is associated with the both operations (through Permission edges). However, he/she cannot perform any operation over the Publisher nodes. User3 has permission to insert new Book and Publisher instances of nodes in the structure. However, he/she cannot alter nodes in the structure.

IV. CASE STUDY

A case study was performed to better understand the potential of the proposed access control model. The goal is to evaluate, in practice, the feasibility of the model in terms of functionality and efficacy, i.e., if access control mechanisms can be implemented based on the model and if it can work correctly, avoiding unauthorized access.

The scenario we used to implement the access control mechanism is related to a newspaper’s news website. In modern news websites, news are posted and can be commented by the users. However, to post or to comment the news, the users must have permissions. The exclusion of comments is also controlled. We choose this area of application because it is simple and very common nowadays, and a graph-database is adequate to manage the amount of news and comments. This scenario is simpler than the example presented in Figure 2 because the focus here is the implementation. In Figure 2 we intended to show specific details of the model.

To implement the access control mechanism we chose the Neo4j GDBMS [10]. This choice was due to the Neo4j is open source, written using the Java programming language, and includes the ACID (Atomicity, Consistency, Isolation, Durability) properties. Furthermore, this GDBMS has few resources for security and access control, with a simple authentication control [11].

The Neo4j supports multiple query languages to manipulate graphs in the database (e.g., Cypher, Gremlin, etc.) [10]. We decided to use Cypher due to its simplicity and similarity to SQL (Structured Query Language) and SPARQL (Simple Protocol and RDF Query Language).

The Neo4j server’s functionality can be extended by adding plugins. Plugins are user-specified code which extend the capabilities of the database, nodes, or relationships. Our implementation was made as one Neo4j extension, configured at the plugin directory. We used the GDBMS Neo4j version 2.1.3 and Java, version 7.0. Figure 3 shows the data flow over the access control model implementation.

![Image 3](https://via.placeholder.com/150)

Figure 3. Data flow over the access control implementation

In Figure 3, System represents the applications that interact with the database, providing and retrieving data. Database is the data set, represented as graph structure. The Access Control Plugin is the implementation of the access control model. It is the schema of structure of nodes for access control, i.e., the graph which represents the data, users and permissions. This is implemented at the Plugin layer from Neo4j. For each commit transaction requested, the implementation in the Plugin layer is executed. If the user requesting a commit operation he/she is trying to perform over the data, if he/she has permission to the operations the commit is executed. Otherwise, i.e., if the user does not have permission to this operation, the rollback is executed, canceling the transaction.

The model was instantiated to the news context. Figure 4 shows, just for illustration, part of the graph representing the instance of the model for this scenario. The structure of nodes is composed of two main types of nodes: POST (represents the news reported) and COMMENT (represents the comments about news). We established a DBA, called master, which must be the first to access the GDBMS to create new users and grant permissions.

In the graph there are two users: User1 and User2. User1 is associated to the Create operation (identifier 3). This operation is associated to Post and Comment nodes, which
Figure 4. Instance of the model for the website news

represent, respectively, a news posted into the website and comments the users can write about the news. This association means that User1 can create posts in the website and comment the posted news (i.e., create the node structures). Also, this user can perform all DML operations over the posts and comments he/she created. In the example, User1 is the owner of the Post and the Comment nodes.

User2 is associated to two operations: Insert (identifier 1) and Select (identifier 2). These both operations are associated to the Post node. These association means that User2 is allowed to insert news (i.e., insert information of an instance) and visualize the news posted in the website. He/She is not allowed to comment news. Still in Figure 4, the Post node has three properties: ID (identification of the new), Title (title of the new) and New (text reporting the new). The Comment node has two properties: ID (identifier of the comment) and Text (text of the comment).

To implement the instance of the model correctly, some details were taken into account. We discuss them following.

The implementation provides validation of attributes.
When instantiating a node it is necessary to verify if all the mandatory attributes of a node are being provided. The operation is performed only if the attributes are according to the structure of nodes defined in the schema.

There is a limitation in Neo4j that had to be overcome in order to accurately follow the access control model. The Cypher does not differentiate DDL and DML operations. Thus, the same instruction is used to create a schema or instantiate a structure of node (the Create Cypher instruction). To distinguish which one of these both operations must be performed we evaluate the arguments of the Cypher instruction. If the goal of the instruction is to create a node in the structure with label METANODE, a new structure is being created. If the user who is trying to create this new structure has the Create permission and the new structure of node does not exist, this structure is created and the commit operation is performed. If the user does not have the Create permission or if the structure of nodes already exists in the database, its creation is not allowed and the rollback operation is performed. When the Cypher instruction does not mention the METANODE, it means that the node is being instantiated. If the user has Insert permission for the METANODE structure with the same label specified in the instruction and the attributes are respecting the rules defined in the model, the commit operation is performed. If the user does not have Insert permission, if the label does not exist in the structure or if the attributes do not fit the rules, the insertion of the node is not allowed and the rollback operation is performed.

According to the model, to alter the attributes in a structure of nodes, the user must have the alter permission. However, this only can be done if there is no node instantiated in the schema. The only alter operation allowed when there are instantiated nodes is to become an attribute as optional. We implemented this verification in order to better contemplate the model’s requirements.

In Neo4j it is not possible to exclude nodes associated to other nodes. To exclude a structure of nodes from the schema it is necessary, first, to exclude the association between the nodes, i.e., the edges. In the model, each node has its respective structures. So, in order to not have nodes without structure in the database, the access control mechanism verifies if there is no node instantiated.

After the implementation of the instance of the model, we performed some tests to evaluate it, just as a proof of concept. The goal is to create users and apply access restrictions in order to verify the efficacy of the implemented mechanism and respective model.

The first step was to connect to the database using the master user. Five new users were created for the tests. For each user, DML and DDL permissions were assigned. Table I shows these users and respective permissions.

<table>
<thead>
<tr>
<th>User</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>User1</td>
<td>Create, alter and drop structure of nodes in the GDBMS, i.e., POST and COMMENT nodes. To User2, only permissions of alter and drop were granted, and only for POST nodes. User3, User4, and User5 have no DDL permissions. Regarding DML permissions, User1 and User3 has permissions to insert, update, delete and select information in the GDBMS for POST and COMMENT nodes. User2 and User4 have the same permissions but only over POST. Over COMMENT User4 has no permission to delete. User5 is the one with more severe restrictions and can only consult POST and insert COMMENT related to the news.</td>
</tr>
</tbody>
</table>

In this scenario it was performed 26 test cases. Among them, 6 have their goal in the context of authentication (e.g., connect to the GDBMS with master user; create a new account and access the database with this new user; delete user and cannot login the bank database with him). Other 9 have their goal in the context of DDL permission (e.g., creation of a structure of node by a user with create permission; trying to create a structure of node by a user without create permission; exclusion of a structure of node by a user with drop permission). The remaining 11 test cases are in context of DML permission (e.g., instantiation of a node by a user with insert permission; trying to exclude a comment by a user without delete permission; manipulation of a node (insert, update, delete) by the owner of the node).

Due to space restrictions, we do not describe these tests cases here, but they can be found detailed at the website of our research group [12].

The test cases helped to refine the implementation and they were successfully performed. This allowed verifying that the access control model is feasible to be implemented and complete enough to deal with permission in the level of nodes.

The performance impact was not considered in this study. Our goal for now is only evaluate the model structure as a proof of concept, and that is why the instance was implemented as a plugin, which can raise some performance impact. If the instance is implemented at the core of Neo4j, we believe the performance impact will be very small. However, it requires the modification of the original database manager, and this is out of scope of this work. This must be done in a future work.

V. CONCLUSIONS AND FUTURE WORK

This paper proposed a security model for access control in graph-oriented databases. Instances of this model can be created, where metadata are related according to the permissions granted to users.

It was possible to adapt the concept of access control from relational database to graph-oriented database. It is complete enough once it supports DDL and DML operations. According to the experiments, the model was effective to provide access control, improving the security of this type of NoSQL database. In addition, the model guides the implementation
of access control to several graph-oriented databases. As the model is based on metadata, it smoothly adapts to graph-oriented structures. If the graph-oriented database has at least a structure of nodes (Meta-Node, Meta-Property) as metadata, it is possible to implement only the access control part of the proposed model (Meta-User, Meta-Group, Meta-Operation).

As Neo4j database does not have any metadata, nor even for the structure of nodes, the whole proposed model was implemented. Although the implementation for the Neo4j used a simple example, it considered all the operations (DDL and DML) and attributes provided by the model, dealing with specific characteristics of Neo4j. The access restrictions were applied correctly, avoiding unauthorized access.

As future work, we intend to implement the model in the core of Neo4j and evaluate the performance impact of the solution in order to evaluate its feasibility in this both context (as plugin and in the core database). We also intend to extend the access control to a finer granularity of access to information, restricting access in two levels: of relation and of properties. In the level of relation, the access restriction will be according to the properties of relationships between the database nodes. In the level of properties, the idea is to restrict access not only to the information represented by the whole node, but pieces of information represented by their attributes. Furthermore, cryptography can be used to improve the security of users, passwords and the data file from GDBMS.

ACKNOWLEDGMENT

This work has been partially supported by the projects EUBra-BIGSEA (www.eubra-bigsea.eu - Project 23614 - MCTI/RNP 3rd Coordinated Call) and ATMOSPHERE (https://www.atmosphere-eubrazil.eu/ - Horizon 2020 grant agreement No 777154 - MCTIC/RNP).

REFERENCES


