Access Control for Database Management Systems

Eric McCandless, Swee K Chew, and Steven Cocke

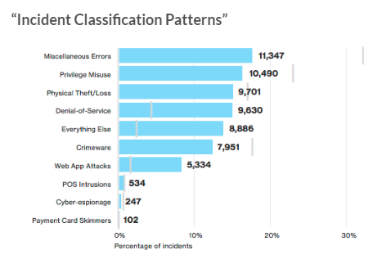
***Abstract*— We present the access control models that are used in relational and non-relational databases. For the relational database, we describe the access control models used in a MySQL database. For non-relational database, we primarily focus on the access controls implemented in MongoDB and Cassandra database programs.**

**In doing so, we discuss the importance of database access control in society today. We identify any commonalities between the three example databases, as well as any weaknesses found, in the context of providing security and confidentiality of the information stored in databases.**

***Keywords*— Access Control, Relational Database, Non-Relational Database, MySQL, MongoDB, Cassandra, Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC)**

# INTRODUCTION

One of the most important aspects of managing a Database Management System (DBMS) that stores sensitive information about individuals is protecting the data from unauthorized individuals. Data breaches have become far too common as more and more organizations are storing data to improve customer-focused business and services practices. Data breaches can occur for many different reasons, including miscellaneous errors, privilege misuse, and physical theft/loss.



**Figure 1**: 2016 Verizon Data Breach Investigations Report [1]

Administrators must design secure DBMS that include appropriate security policies and mechanisms to satisfy three basic security requirements: confidentiality, integrity and availability. This ensures that data are prevented from being improperly disclosed, modified, and denied [2].  Adding to the complexities in securing DBMS is that there are multiple types of relational and non-relational databases with different designs.

To protect sensitive information within DBMS, organizations implement security features, such as authorization, authentication, auditing and data encryption.

This paper focuses on DBMS access control, which is part of the authorization security feature. Access control is a common and effective method for protecting sensitive information. It is a tool that allows DBMS administrators to determine DBMS rights and privileges and enforce those policies.

The goal of this paper is to explore access control methods across three types of databases:  MySQL, MongoDB, and Cassandra. Furthermore, the paper will identify any commonalities in access control methods between the three databases and highlight potential weaknesses.

As a roadmap for this paper: In Section II, we discuss MySQL database for access control and the four models that have been used widely. In Section III, we review access control for the MongoDB database. In Section IV, we discuss access control for Cassandra. Sections V and VI include a comparison of the three access control methods, some of the weaknesses with access control methods, and conclusions.

1. MySQL Access Control

MySQL is a relational database management system based on Structured Query Language (SQL). The data in a MySQL database are stored in tables. A table consists of columns and rows. One of the advantages of using MySQL is that it is capable of replicating data and partitioning tables for better performance and durability.

In the framework of DBMS, two access control models, discretionary access control (DAC) and mandatory access control (MAC), were developed early on. The discretionary access control (DAC) allows users to grant accesses on the data for which they have administration authorization to other users. The creator of a relation in an SQL database becomes the owner and the owner has the ability to grant another user one or more privileges using the GRANT statement in SQL. Similarly, the owner can also use REVOKE statement to take away the privileges. The privileges that can be granted include insert, select, delete, update and many others.

The general SQL statement for a grant operation [2] is as follows:

GRANT privileges

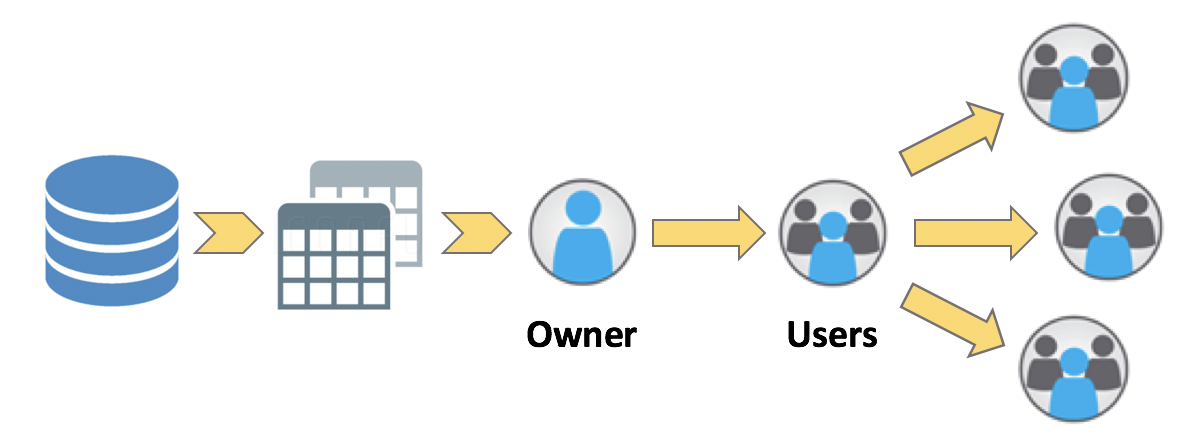
[ON relation]

TO users

[WITH GRANT OPTION]

The GRANT command can be applied to base relations as well as views. ON and WITH clauses in the brackets are optional. With the GRANT OPTION, the user, whom the owner grant privileges to, can further grant the same privileges to other users. Thus, there is no proper management of access rights under the DAC model.

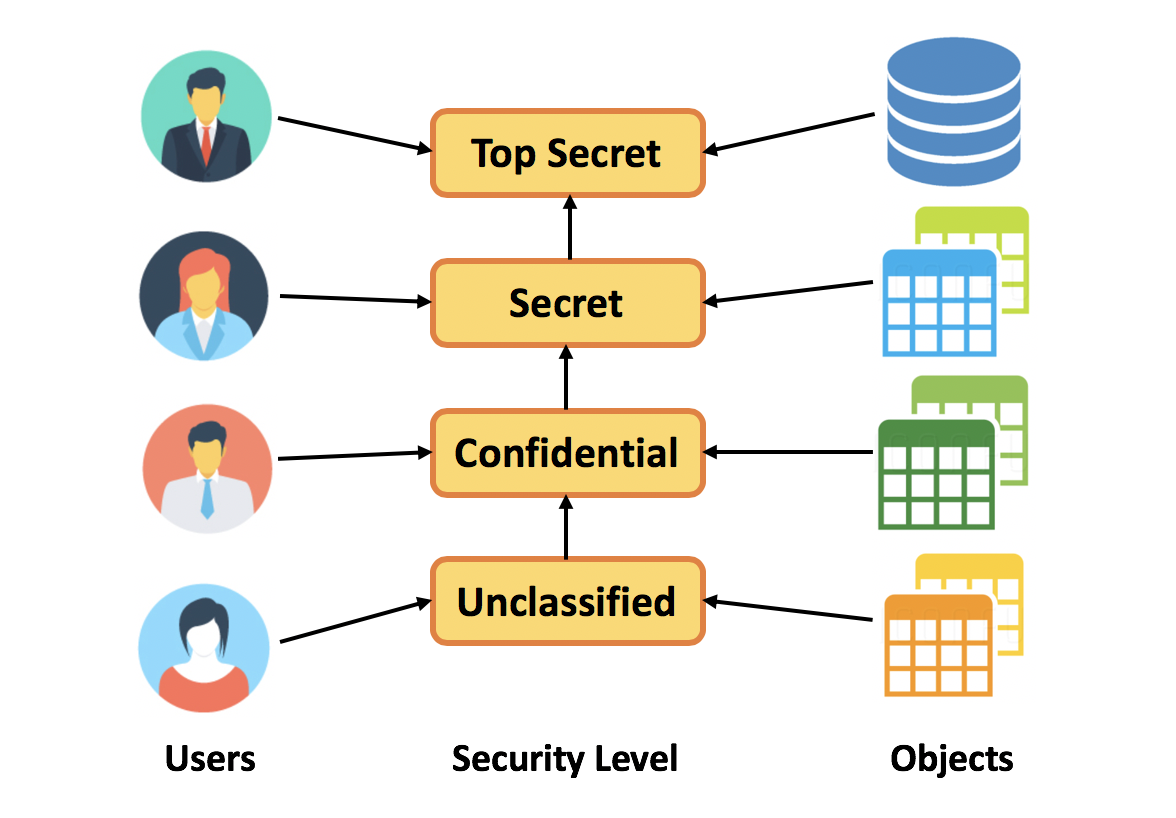
Figure 2 illustrates the DAC model used for access control. Owners and authorized users can modify access at anytime, thus it provides high flexibility. However, it has very weak security as the users continue to grant access to many other users who may or may not need access to the database or data tables that may contain sensitive information.



**Figure 2**. Discretionary Access Control (DAC)

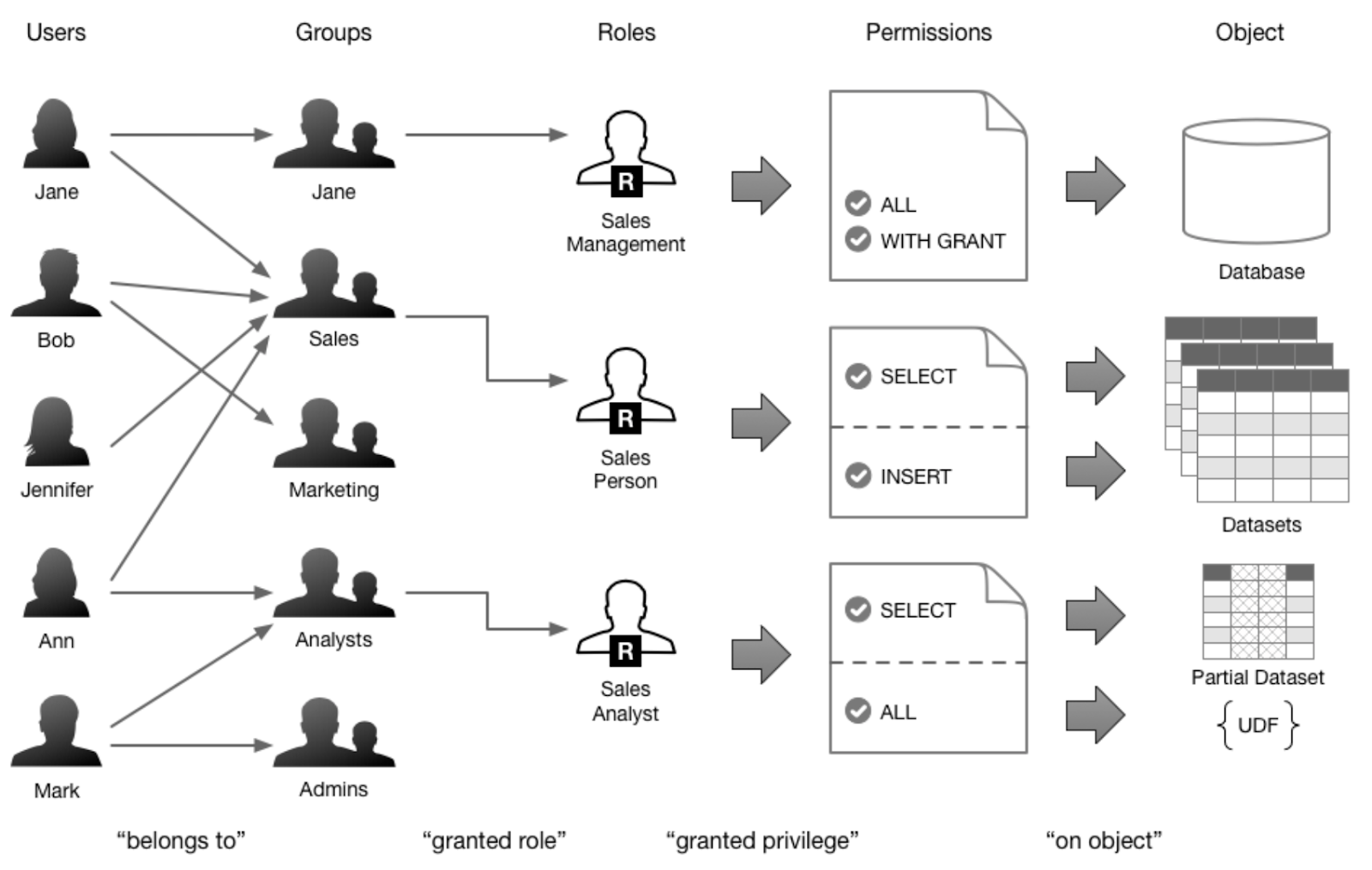
Another traditional access control model, MAC, regulates accesses to data using predefined classifications of users and data. Under this model, even the creator of a data object is not able to grant accesses to other users [2]. The criteria are defined and enforced by the system administrators or the operation systems. The elements under MAC consist of (1) Objects: the entities containing information to be protected, (2) Subjects: the entities requiring access to objects, and (3) Access modes: the types of operations subjects performed on objects, which includes read, write, and append [3]. Sensitivity levels are assigned to objects. An example of a set of security labels is {Top Secret, Secret, Confidential, Unclassified}. Then subjects are assigned clearance levels belonging to one of these classifications and they can perform operations that are allowed up to and including their assigned clearance levels. For an example, an individual with a “Secret” clearance cannot read “Top Secret” data but they can read objects that are allowed to subjects with “Confidential”, “Unclassified” or “Secret” security clearances.

Figure 3 demonstrates the MAC model in which each subject gets security clearance and each object has security classification. Only subjects of same or above security clearance can access objects of a security class. Although MAC is more secure than DAC, it is not a very flexible model and it limits user functionality.



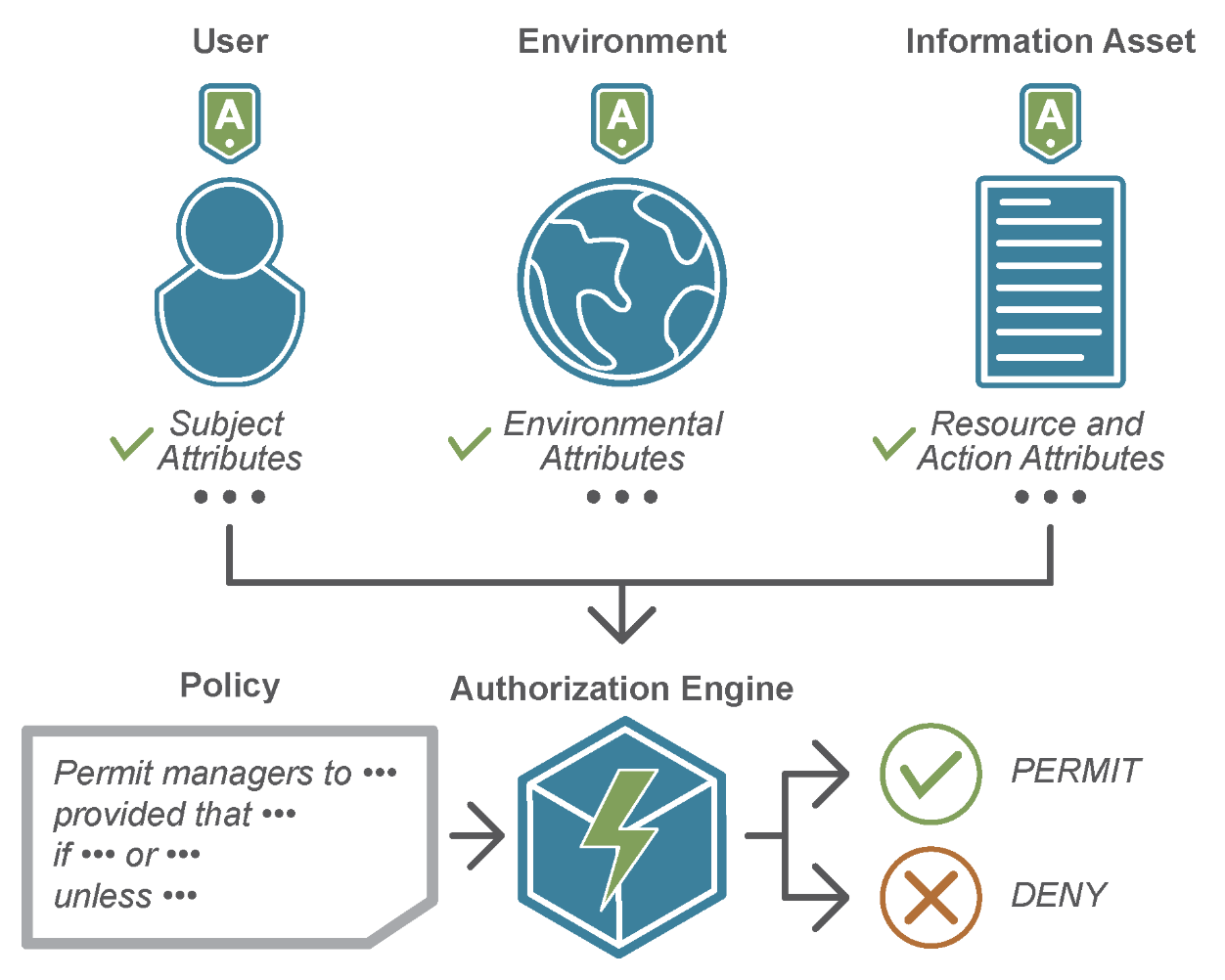
**Figure 3**. Mandatory Access Control (MAC)

In recent years, role-based access control (RBAC) is used to fill the gap between DAC and MAC and it presents a major advancement in efficiency from the traditional access control models. The administrative costs of maintaining access control lists or other similar data structures is a prevailing problem in traditional access control models. The number of grant and revoke operations to be performed become difficult to manage over time, especially with a highly dynamic user populations. Furthermore, end-users do not own the objects that they are allowed to access. In RBAC, access permissions are associated with roles instead of having discretionary access to enterprise objects. Users are assigned to be members of appropriate roles which have predefined associated permissions to perform certain operations on the objects [4]. Figure 4 illustrates the role-based access control model in DBMS. Since there will be far fewer roles than either users or permissions, RBAC adds simplicity to the access control management and reduces the administrative costs. However, RBAC model is unable to accommodate real-time context.



**Figure 4**. Role-Based Access Control (RBAC) [5]

Lastly, attribute-based access control (ABAC) model is introduced to provide most flexible and dynamic access control paradigm. The prior three access control models, DAC, MAC, and RBAC, are user-centric and have some disadvantages in which access control permissions are predefined and environment conditions, such as time of the day or user IP, are not taken into consideration. ABAC model, on the other hand, tries to address this by defining access control based on attributes, in order to provide dynamic, fine-grained and contextual access control model. ABAC controls access to objects by evaluating rules against the attributes of users, environment, resource and operations relevant to a request as illustrated in Figure 5. It enables more precise access control and creation of access rules without specifying individual relationships between each subject and each object. Under ABAC, access decisions can change between requests by simply altering attribute values, without making changes to the underlying subject/object relationships [6]. Thus, this provides a more dynamic, fine-grained access control management capability.

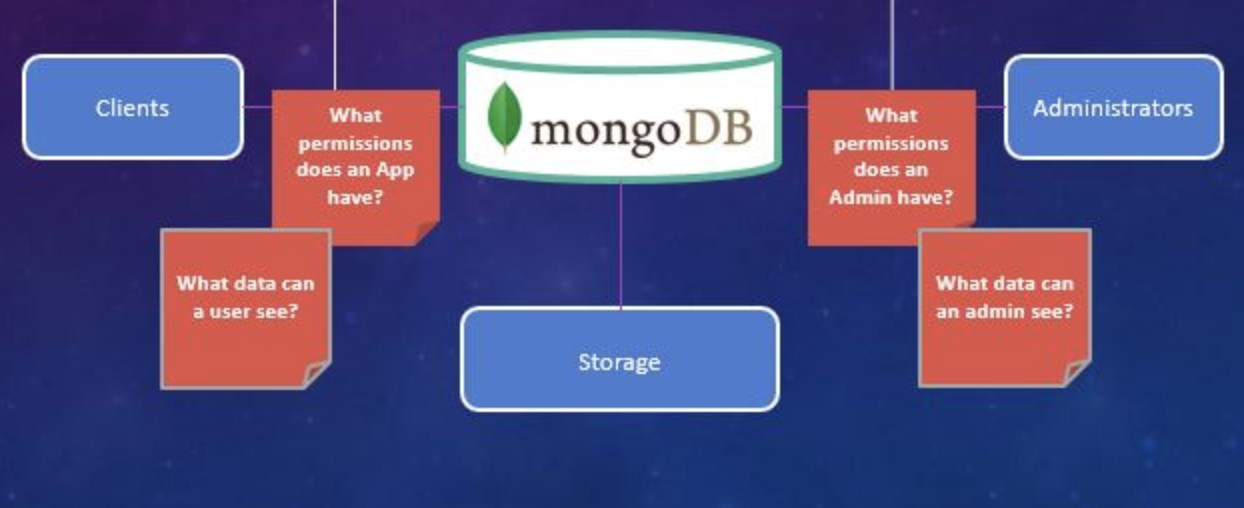


**Figure 5**. Attribute-Based Access Control (ABAC) [7]

1. MongoDB Access Control

MongoDB is a document-oriented database, as opposed to a relational one, that allows users to query and index information.  It uses JSON-like documents and non-structured query language.  One of the primary advantages of the database is that it allows for data to be split across multiple servers to increase storage capacity and scale [8].

MongoDB access control ensures that the people accessing the database are positively identified and can access, update, or delete the data they are given access to. It uses a role-based access control (RBAC) with a set of privileges.  A user is granted one or more roles that determine the user’s access to database.  A role grants privileges to perform the specified actions on resource.



**Figure 6**: MongoDB Access Control

Users have to enable authorization for access control of MongoDB.  Once access control is enabled, users must authenticate themselves [9].

Each MongoDB user should only be able to access data that is required for their role as determined by the data security manager.  Below are the steps to implement [10].

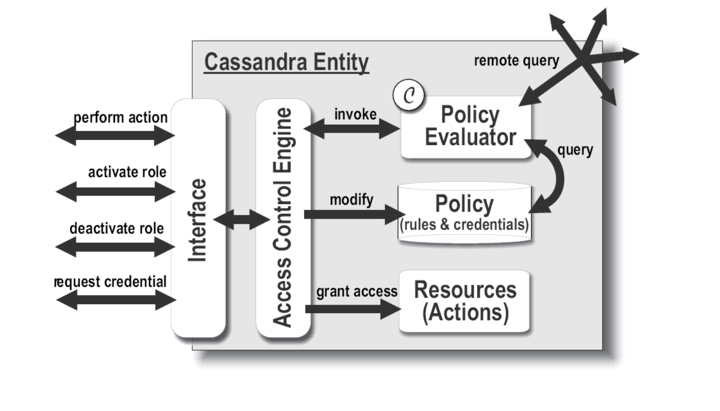
*1. Use createRole command to create a new role*  
use '<dbName>'  
db.createRole({  
role: "<roleName>",  
    privileges: [  
      { resource: { db : “<dbName>”, collection : “<collectionName>” },  
       actions: [ '<actionName>' ]  
      }  
   ],  
    roles: [ { role : '<fatherRoleName>', db : '<dbName>'} | '<roleName>' ]  
})

*2. Grant a role to a user*  
use '<dbName>'  
db.createUser({  
user: "<userName>",  
    pwd: "<password>",  
   roles: [ { role: "<roleName>", db: "<dbName>" } ]  
})

1. Cassandra Access Control

Apache Cassandra, originally developed by Facebook to power its inbox search feature, is a wide column store, distributed, NoSQL database management system [11].  A wide column store database refers to a database that has a two-dimensional key value store. It contains rows, columns, and tables, but the names of columns, as well as format, can vary between rows in the same table. The term distributed means that the system is designed to handle large amounts of data over various clusters/servers, where each node has the same role and can successfully handle any request. There is no master node within the Cassandra system. Each node stores different data; the data is not replicated on every node in a cluster, however, copies of one node’s data can be stored on another node as a fail-safe [11]. There is no single point of failure within the system. Cassandra is built to be both replicable and scalable, meaning that settings that can be configured can be deployed on several nodes or applied to clusters, as well as having read and write output scale linearly as new nodes are added to the system. Cassandra has its own query language, Cassandra Query Language (CQL), which is essentially an alternative to a Structured Query Language (SQL) [11]. CQL allows an interface to interact with the database system using a native syntax.

Within the database system, every node runs a Cassandra application that allows it to communicate with the other resources within the system [12]. Since there can be several nodes or clusters within an environment, access control needs to be defined for all users/applications consistently across the system. Cassandra is a role-based access control system (RBAC), which means that users/applications are assigned roles, and each role is defined in a certain way so that rules can be inferred about how that role is allowed to interact with the database (i.e. the actions they are allowed to perform) [12]. These inferred rules are gathered from policies within the system, which essentially state the permissions of different roles.  Some examples of access control within a database system include standard policies, role hierarchy and delegation, trust, and validity periods. Actions related to these topics can include things such as performing an action, role activation/deactivation, and credential requesting [12].



**Figure 7**: Cassandra Access Control Model [12]

When a user or application tries an action within the Cassandra system, a request to perform that action is sent. This request is received by the access control engine, which then calls upon policy evaluation engine. The policy evaluation engine’s responsibility is to check the requests against the stored policies, which contain user/application credentials, along with the definitions for what they are allowed to do within the system. Figure 7 is a diagram that shows this exact process. The result from the policy evaluation engine’s check is then returned to the access control engine and, based upon the result, either grants or denies the action [12].

1. Comparison of Access Control Methods

The common denominator, with regards to access control methods, between all three databases discussed is role-based access control. Each database has its own flavor of how RBAC is carried out, but RBAC remains the foundation for each system. MySQL, MongoDB, and Cassandra all control database access through the creation of users and groups. These users and groups are assigned roles, and these roles are then given specific privileges. Rules can then be inferred about how the user or group can interact with the database from the role(s) and privilege(s) assigned.

Some concerns with the RBAC model, include [13] [14]:

**Scalability:** If the organization’s data policy and framework is not strong, RBAC will require redesigns to get back on track, and/or introduce “work around” solutions.  This will increase the administrative workload and cost.

**Security:** If it is the practice of the organization to deal with security issues only as they become known, it can create major security issues.  If RBAC security administrators do not have intimate knowledge of how permissions are being granted, why they are being granted, and what operations are associated with those permissions and roles, it can lead to a very insecure database as users may gain access or continue to have access to data that they should not have access to.

**Logistics in Design:** RBAC may be difficult to implement if the infrastructure is large and pre-existing (e.g., organization is located at multiple locations).  Also, administrators will need to have intimate knowledge of the organization and will need the knowledge and skill to develop a hierarchical tree structure and develop roles.  Once the roles are developed, they need to be placed appropriately into the tree. This requires engineers and project managers along with discussions with the business side to understand their needs and organizational structure to ensure the database is functioning properly and according to the organization’s policies.

1. Conclusion

In conclusion, in designing a database management system, having an appropriate access control management properly set up is very important in safeguarding the data stored in databases. Access control method employed in any DBMS should be able to protect confidentiality, ensure data integrity, and restrict availability of sensitive information such as social security numbers, protected health information (PHI) and financial information. Only users who need to perform their job functions should have access to required databases or datasets. A system should also in place to revoke access to objects from users who move to another department or change roles within organization.

Furthermore, in today’s digital world, information systems are vulnerable to security breaches leading to data theft and unauthorized disclosures. Although several techniques, such as data encryption and digital signatures, are available to protect data, a truly comprehensive approach for data protection must include mechanisms for enforcing access control policies. Role-based access control (RBAC) is commonly used in relational and non-relational database management systems due to flexibility and scalability that it offers. An interesting direction for future work is exploring how RBAC policy implemented in a database server can be exploited by cyber criminals, and whether moving data from one server to another would accurately carry over the predefined roles and rules.

References

1. 2016 Data Breach Investigations Report (Rep.). (n.d.). Retrieved December 1, 2018, from Verizon website: https://www.verizonenterprise.com/resources/reports/rp\_DBIR\_2016\_Report\_en\_xg.pdf
2. Sandhu, R. (1994). Relational database access controls. Handbook of Information Security Management, Auerbach Publisher, Philadelphia, PA, 145-160.
3. Bertino, E., Ghinita, G., & Kamra, A. (2011). Access control for databases: concepts and systems. *Foundations and Trends® in Databases*, *3*(1–2), 1-148.
4. Ferraiolo, D., Cugini, J., & Kuhn, D. R. (1995, December). Role-based access control (RBAC): Features and motivations. In *Proceedings of 11th annual computer security application conference* (pp. 241-48).
5. Role-based Access Control. (2018, November 5). Retrieved December 1, 2018, from <https://docs.okera.com/authorization>.
6. Hu, V., Kuhn, D., & Ferraiolo, D. (2015). Attribute-based access control.(Report),48(2), 85–88. <https://doi.org/10.1109/MC.2015.33>.
7. Attribute Based Access Control (ABAC). (n.d.). Retrieved December 1, 2018, from <https://www.axiomatics.com/attribute-based-access-control/>.
8. What is MongoDB - Introduction to MongoDB Architecture & Features. (2018, November 06). Retrieved December 1, 2018, from https://intellipaat.com/blog/what-is-mongodb/
9. Intellipaat.com, Intellipaat. [online]. 2018. [Accessed 17 October 2018] Available from: <https://docs.mongodb.com/manual/core/authorization/>
10. Vaibhaw. (2017, November 17). MongoDB Basics: Configuring Role-Based Access Control (RBAC). Retrieved December 1, 2018, from https://scalegrid.io/blog/mongodb-basics-configuring-role-based-access-control-rbac/
11. Mitesh, Soni (2015, Jul.).  An Overview of Apache Cassandra*.* [Online]. Available:<https://opensourceforu.com/2015/07/an-overview-of-apache-cassandra/>
12. Becker, M., & Sewell, P. (2004). Cassandra: Distributed access control policies with tunable expressiveness. *Proceedings. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004. POLICY 2004.* doi:10.1109/policy.2004.1309162.
13. Weber, H. A. (2003, October 8). Role-Based Access Control: The NIST Solution (Rep.). Retrieved December 2, 2018, from SANS Institute InfoSec Reading Room website: <https://www.sans.org/reading-room/whitepapers/sysadmin/role-based-access-control-nist-solution-1270>.
14. Nyanchama, M., & Osborn, S. (1993). Role-based security: pros, cons, & some research directions. ACM SIGSAC Review, 11(2), 11-17.