

# A Comparative Analysis of Security and Privacy Challenges: Oracle vs NoSQL Databases

Group G04  
Aayush Kaffle, Manju Bhusal

April 30, 2024

## 1 Introduction

In this era of big data, it is very challenging for database systems to manage and process large volumes of data. Therefore, there is a need of a secure and reliable database for maintaining the infrastructures required to cope up with these advancements, regardless of whether it is a relational or NoSQL database. In the earlier days, relational databases were used by most organizations and were considered appropriate for structured data[4]. However, with the need for high flexibility and scalability, NoSQL came on the rise to manage both structured and non-structured data and most of companies shifted towards NoSQL databases with the purpose of enhancing and optimizing performance.

However, the concerns regarding security and privacy also increased with the shift from relational to NoSQL databases. It has been found that due to high scalability and flexibility, there are high chances of data breaches in NoSQL databases compared to relational ones. In order to identify the result, we conducted a comprehensive investigation into the security and privacy features of both relational and NoSQL database. We compared one relational database (Oracle) with the three distinct NoSQL databases (MongoDB, Cassandra, and Neo4j) to identify their strengths and weaknesses. It describes the key security and privacy features of four databases under authentication, authorization, encryption, auditing, and injection attacks.

To collect the information, the study reviewed different research, academic papers, blogs, and real-world case studies relating to security and privacy issues. It provided different information based on these issues under different time period and scenarios. Also, the paper reviewed the updated documentation of all four database to identify what mechanisms did these database implemented in terms of security and privacy. By examining and analyzing the information, the study compares among four databases and provides necessary recommendation for using appropriate databases.

In the proposal, we mentioned that we were going to use CouchDB as one of the NoSQL databases but later we planned to study Neo4j in order to introduce one graph database in our paper which in this case is Neo4j. Furthermore, we found an extensive research paper comparing the security features of graph databases like Neo4j.

This document is outlined in four other sections. Section 2 includes the core content of the study which discusses different topics of the study. Under section 2, we described related work or any previous work that was conducted on this topic. Also, we briefly mentioned the definition and features of security and privacy. Then, we gave an overview of Relational and NoSQL database and their features. Finally, we provided detailed information on security and privacy features for each of the relational (oracle), MongoDB, Cassandra, and Neo4j databases in the subsections. Section 3 provides a conclusion for all the information discussed on the core content. And finally, section 4 consists of references.

## 2 Core Content

In this section, we will compare relational and aforementioned NoSQL databases in terms of security and privacy.

### 2.1 Related Work

There are several research papers published about the comparison between NoSQL and relational databases in terms of characteristics, features, and their usage in the world of technology. In [1], the authors compare relational and NoSQL databases based on their scalability and performance, flexibility, query language, security and storage and access. They classify what database would be useful based on the needs of the data administrator. Another study was conducted to study the comparative analysis of security features and concerns in NoSQL databases [2],[3]. Similarly, [4] delves into the security and privacy implications of database systems in the context of Big Data. It evaluates security and privacy implementations in relational, NoSQL, and NewSQL database systems, with ongoing challenges and future directions.

Furthermore, the paper published [4], explores different NoSQL databases comparing them to relational databases, for handling large data volumes including the security challenges for each. The authors in [6] study the security capabilities of various NoSQL databases like Hadoop and MongoDB, examining their access control mechanisms, native security features, encryption capabilities, and adaptability.

### 2.2 Security and Privacy

Security is an important aspect of any database either relational or NoSQL; this helps to ensure data confidentiality, integrity, and availability. Security is the process of safeguarding information using different mechanisms that could be either stored in the database or transmitted across the network. Maintaining security helps to limit unauthorized users from accessing the database; however, it is a challenging task.

Privacy is defined as the preservation of users' sensitive information such as usernames, passwords, and credit card details through habits, profiling, tracing, or location services. The author in [3] suggests a policy enforcement framework with an encryption scheme based on privacy homomorphism (an encryption's ability to combine the ciphertexts so that they decrypt as if the plaintexts were combined). There are two types of attacks based on the privacy of data [4]:

(a) Correlation Attacks: Values in a dataset are linked with other information to create unique and informative entries. For example, if one database lists user information with medication prescriptions and another lists user information with pharmacies visited, then once these are linked the correlated database can contain information like which patient bought medication from which pharmacy [4].

(b) Identification Attacks: When an adversary attempts to find information by linking entries in a database. For example, if an employer searches for its employees in a pharmacy database, it may reveal the employee's medical history information.

Security and privacy depend on different factors such as encryption methods, auditing, access control mechanisms as well as implementation of privacy policies.

#### 2.2.1 Authentication

Authentication is the process of verifying the credentials of the users associated with the database before deciding to grant them access to the database. There are different mechanisms for authenticating the users. A typical authentication involves password-based authentication. Some databases have their own built-in authentication while some of them use other mechanisms such as digital certificates and integrated directory services[4].

### **2.2.2 Authorization**

Authorization is the process of providing the permission to the user to access the database they are related to after verifying their credentials. This will help to provide access control to the verified users based on the role or schema level which limits their actions.

### **2.2.3 Encryption**

Encryption is the process of converting the original file into the coded form to protect the data from unauthorized access. It is done to protect the sensitive data. In order to use the encrypted data or information, the user needs to decrypt it first to get it back in its original form. Encryption can be performed either at rest meaning encrypting data that are stored on disk or encrypting data at transit over the network.

### **2.2.4 Auditing**

Auditing is the process of monitoring tracking and recording the activities of the users and system within the database. It involves the process where audit trails are generated which logs different activities of user and system. These event logs are then evaluated for analyzing the security conditions. Other security mechanisms are used to mitigate the vulnerabilities and attacks, while this audit trail helps to find the root cause of the incident [4] and provides recommendations to mitigate those incidents.

### **2.2.5 Vulnerability to DoS / Injection**

Vulnerability to Denial of Service is the process by which attackers send many requests to the website making it overloaded, so that either it will provide a slow response or would not be able to provide service to the user who request service. Vulnerability to Injection is the process by which the attackers will inject harmful data into the command or query which causes data breaches. An example of Vulnerability to Injection is SQL Injection.

## **2.3 Relational vs NoSQL databases**

Relational databases use vertical scaling whereas NoSQL databases uses horizontal scalability of the system which is a cheaper solution than vertical scaling. In terms of schema, SQL (Relational) databases are based on static schema, each change in the database must be carefully made to avoid any errors whereas NoSQL databases can handle well-structured, semi-structured, and also unstructured schema. The relational database aims to reduce redundancy by normalization, which is splitting data into small logical tables [3]. Relational databases follow ACID (Atomicity, Consistency, Isolation, Durability) properties whereas NoSQL databases follow BASE (Basically Available, Soft state, eventually consistent) principles. Both properties follow the CAP (consistency, availability, and partition tolerance) theorem as shown in figure 1, which says that the data is consistent in every server, always accessible, and works fine despite network and machine failures. These are further described in the following table:

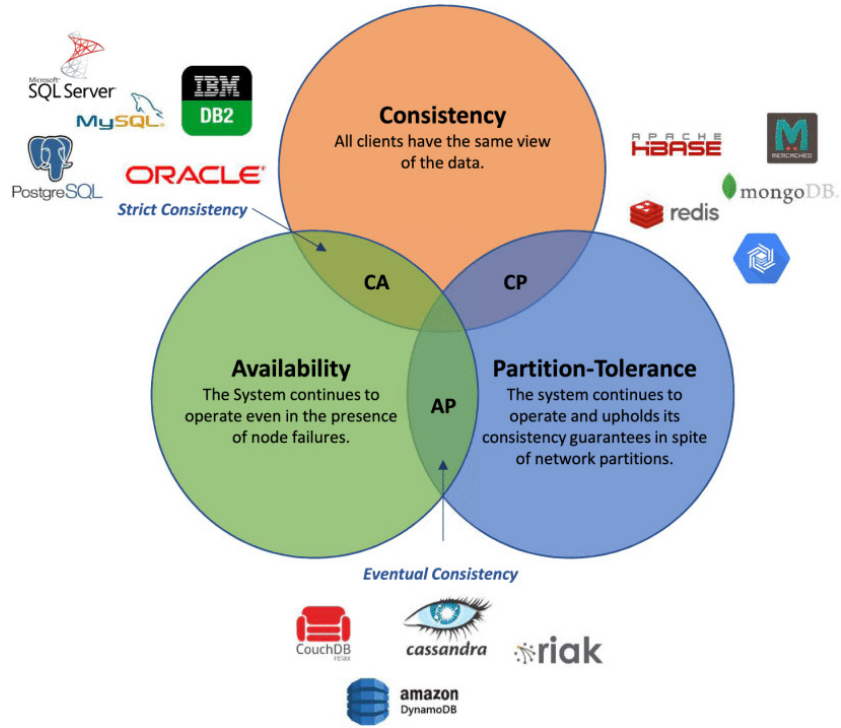


Figure 1: Database Systems by CAP Theorem [4]

Table 1: Comparison between SQL and NoSQL RDBMS [3]

Feature	SQL	NoSQL
Schema	structured	structured, semi-structured, and un-structured
Redundancy	avoided with normalization	replication management
Atomicity	guaranteed	guaranteed
Consistency	guaranteed	to be managed by developers
Isolation	guaranteed	fault tolerance guaranteed
Durability	guaranteed	data availability guaranteed
Scalability	vertical scaling	horizontal scaling
Query language	standard	custom

Based on how a database stores and retrieves data, NoSQL databases can be categorized into four types:

- a) **Key-value:** These are a type of NoSQL database where data are stored as a collection of key-value pairs. Each item in the databases has a pair consisting of a unique key with its corresponding value and thus can be implemented using a hash table where keys are indexes and a pointer to hold the actual data. Because of this, they are efficient for simple retrieval and storage operations making retrieval efficient with constant time. One weakness of such a data model could be in creating customized views of the data since this data model is schema-less [3].  
Redis is a key-value store NoSQL database and is widely adopted by companies such as Amazon, OpenAI, etc. Typical use cases are for storing database query results, web pages, cached objects, etc [3].
- b) **Column-oriented:** Column-oriented stores data in a column-wise architecture rather than the row-wise storage used in traditional relational databases. Every row has a row key and a row can contain many columns. A row can contain, super columns, which are the nested columns inside another column [2].  
Google introduced column-oriented schema in BigTable to handle the amount of data storage in Gmail, Google Maps, etc. Cassandra, HyperTable, and HBase are some databases that follow a column-oriented model [3].
- c) **Document-based:** A document-based NoSQL database stores data as a collection of documents. Each document is a self-contained unit of data that contains key-value pairs which makes it a flexible and schema-less data model. These can be used for content management software, and blog software, however, one downside is that they require a lot of relations and normalization [2].  
MongoDB, CouchDB, and RavenDB are some popular document-based NoSQL databases.
- d) **Graph databases:** Graph databases use graph structures for data storage, with nodes, edges, and properties representing entities, relationships, and attributes. They are optimized for traversing relationships which makes them efficient for queries that involve connected data. This model supports ACID properties and the rollback feature which ensures the consistency of data [2].  
Popular graph databases are Neo4j, RedisGraph, Titan, etc. In graph databases, a hashing technique is used to encrypt data organized within graphs [3]. However, if there is any change to the graph, the hash value must be re-calculated for the entire structure affecting its performance.

DATABASE	
KEY	VALUE
key_0	ID: id_0
	field_name_0: field_value_a
	field_name_1: field_value_b
key_1	field_name_2: field_value_c
	ID: id_1
	field_name_0: field_value_d
	field_name_1: field_value_e
	field_name_2: field_value_f

(a) Key-value

DATABASE	
ID: 0	
column-family: column-family_id_0	field_name_0: field_value_a field_name_1: field_value_b field_name_2: field_value_c
column-family: column-family_id_1	field_name_3: field_value_d field_name_4: field_value_e
ID: 1	
column-family: column-family_id_0	field_name_0: field_value_f field_name_1: field_value_g field_name_2: field_value_h
column-family: column-family_id_1	field_name_3: field_value_i field_name_4: field_value_j

(b) Column-oriented

DATABASE	
ID	DOCUMENT
id_0	field_name_0: field_value_a field_name_1: field_value_b field_name_2: field_value_c pointer to another document_2
id_1	field_name_0: field_value_d field_name_1: field_value_e field_name_2: field_value_f pointer to another document_3

DATABASE	
ID	DOCUMENT
id_2	field_name_3: field_value_g field_name_4: field_value_h
id_3	field_name_3: field_value_i field_name_4: field_value_j

(c) Document-based

DATABASE NODE_0	
ID	DOCUMENT
id_0	property_name_0: property_value_a property_name_1: property_value_b property_name_2: property_value_c
id_1	property_name_0: property_value_d property_name_1: property_value_e property_name_2: property_value_f

DATABASE NODE_1	
ID	DOCUMENT
id_2	property_name_3: property_value_g property_name_4: property_value_h
id_3	property_name_3: property_value_i property_name_4: property_value_j

(d) Graph databases

Figure 2: Data store for each NoSQL database type [3]

### 2.3.1 Relational (Oracle)

Oracle is one of the popular relational databases. Since, this database complies with ACID (Atomicity, Consistency, Isolation, and Durability) features, it can be advantageous for enterprises that demand high security and robustness. The latest version of Oracle is Oracle Database 23c, which is best suited for developers who are looking to create relational applications, documents, graphs, and new microservices [8]. The main security risk that NoSQL databases face are encrypted data storage, unauthorized exposure of both data and backup (or replicated data), and insecure communication over the network [3].

Oracle provides comprehensive security features such as authentication, authorization, and auditing. Moreover, Oracle offers fine-grained access control which helps to impose security more strictly [16]. The author [6] compared security maturity between the NoSQL database and two relational databases; Oracle and MySQL, and found that relational databases offer more in-built security features.

The Oracle database 23c offer following security features:

#### 1. Authentication:

Oracle 23c helps to secure the database by applying different authentication methods for verifying authorized users. Oracle can authenticate using different methods such as Centralized authentication and single sign-on, Kerberos, remote authentication dial-in user service (RADIUS), and certificate-based authentication. In addition to that Oracle Database supports length of passwords up to 1024 bytes which ensures secure authentication.

#### 2. Authorization:

The Oracle 23c database gives priority for schema level compared across systems while granting the permission. It helps to ensure tight security by blending the role-based access control with the schema level which limits user actions based on the roles and the schema where they are allowed to access. For example; Let us suppose one HumanResource Schema which contains different tables such as Employees, JobPosition, Address, and soon. Then if a user is assigned to select and update the HumanResource Schema, then the user is authorized to select and update the HumanResource Schema only, the user is not allowed to perform delete, insert, or other options unless stated.

#### 3. Encryption:

Oracle offers encryption at rest which requires decryption keys to access the stored data and this will be performed by Transparent Database Encryption. In addition to that, Oracle 23c supports a secure protocol, Transport Layer Security (TLS) 1.3, which helps to encrypt the data that communicates between client and database across the network.

#### 4. Auditing:

Oracle has enhanced the capability of auditing to track and record information regarding the activities of a database of the selected user. This database supports granular audit policies that align with regulatory requirements. This thus helps to prevent policy violations and unauthorized user from accessing the database resulting enhanced security of the database.

#### 5. SQL Firewall:

SQL firewall is included in the Oracle database to prevent SQL injection attacks, theft, or unauthorized access. This SQL firewall checks all incoming SQL queries and blocks the unauthorized SQL. It examines all the SQL regardless of local or network, or encrypted or clear.

### 2.3.2 MongoDB

MongoDB is one of the NoSQL databases that uses flexible, and document-oriented databases to manage data stored in JSON-like documents [16]. MongoDB being a schema-less database, gives the users the flexibility to add new fields or change the previous structure of document. MongoDB allows readily accessible embedded documents, it doesn't require the usage of joins, in contrast to relational databases. This enables MongoDB to organize data into hierarchical structures while making it accessible for querying and indexing as a result of which different applications can model data in a more natural way [2]. Additionally, MongoDB supports sharding, where data are distributed across several servers which helps to handle the load balancing, data distribution, and failover automatically. The common features of MongoDB includes ease of use, flexibility, sharding, high performance, horizontal scalable.

MongoDB is a popular database among NOSQL databases. Since many companies are using this database for storing data because of its flexibility, performance, and scalability, it is crucial to understand how secure this database is. There are different studies regarding the security features of MongoDB. Regarding the case study [16], The author concluded that MongoDB offers basic security features like authentication and role-based access control while suggesting the need for advanced security features.

The security features of MongoDB are as follows:

#### 1. Authentication:

MongoDB requires all clients to authenticate themselves to access the system. By default, the authentication mechanism for MongoDB is the Salted Challenge Response Authentication Mechanism (SCRAM). SCRAM complies with the IETF RFC 5802 standard. It helps in the two-way authentication between server and client as well as provides the iteration count of the use of hash function and random salts for each user. The SCRAM mechanisms support SCRAM-SHA-1 and SCRAM-SHA-256 for hashing function. In addition to SCRAM, MongoDB supports other authentication mechanisms. x.509 Certificate authentication is also provided by MongoDB for both client and internal authentication for which a secure Transport Layer Security/Secure Sockets Layer connection is required. Similarly, the other authentication mechanisms are Kerberos Authentication, LDAP Proxy Authentication, and OpenID Connect Authentication [9].

#### 2. Authorization:

MongoDB applies role based access control on which it provides one or more roles to a user for accessing database resources. To enable authorization, one should use `-auth` or the security. authorization settings. After enabling, users must authenticate themselves to access the system and resources. A user can have one or more roles which provides the privilege to perform actions. The roles can be either built-in or user-defined. MongoDB allows user to create and define new roles based on their need if the built-in roles don't provide sufficient privileges. Additionally, MongoDB supports LDAP (Lightweight Directory Access Protocol) authorization which helps LDAP servers to find the group they belong to through the LDAP server. Each group has the Distinguished Names and MongoDB maps them to their specific roles. MongoDB thus uses the mapped roles and their corresponding privileges to grant the user authorization.

#### 3. Encryption:

MongoDB provides different encryption methods for the security of the data. Queryable encryption provides a secure method of encrypting sensitive data from the client side and stored encrypted data on the server side too. Despite, this encryption, the client can still run many queries on the encrypted data which remains safe throughout every action. During all these activities, the server will not be aware of the encrypted data it is processing. The other feature is Client-Side Field Level Encryption where the client will encrypt data before sending it over the network to MongoDB so that even MongoDB will not have access to the client's unencrypted data. Another security feature, Encryption at Rest is offered by MongoDB Enterprise 3.2 available for the WiredTiger storage engine. Under this feature, the data files are allowed to be encrypted for securing data and in return can be decrypted by the parties who have that key to transform it into the original one.

#### 4. Auditing:

For Mongod and Mongos, MongoDB Enterprise facilitates an auditing mechanism that allows administrators

and users to track the activities for deployments with multiple users and applications. And you can go to `-auditDestination` to enable auditing. After enabling, the auditing system will record all the activities relating to the schema, replica set and shared cluster, authentication and authorization, as well as CRUD operations [9].

#### 5. Vulnerability to DoS/ Injection Attack:

MongoDB are vulnerable to injection attack as they do not enforce authentication by themselves. Therefore, there is a high chance of attackers accessing all data within the database[Evans].

### 2.3.3 Cassandra

Cassandra is an open-source, distributed, and decentralized NoSQL database management system designed to handle large amounts of data across many community servers providing high availability and scalability without affecting performance [1]. Some of the biggest companies that use Cassandra as their database are Twitter, Facebook, Netflix, etc.

NoSQL database provides complete security against attacks. In this section, we will look at how secure the Cassandra database management systems are against vulnerabilities to authentication, authorization, communication encryption, audition, and DoS/Injection attack criteria. Cassandra stores data without encryption and SQL language could be potentially exposed to injection attacks, to overcome this, Thrift APIs have been introduced in Cassandra [3].

#### 1. Authentication:

Users can have a pluggable authentication which can be enabled using the authenticator settings in the "Cassandra.yaml" file. Users can either remove the need for authentication or use the password authenticator, in which the usernames are hashed but unsalted MD5 passwords which are saved in the system's "auth.credentials table" [2]. When a server enables authentication, any connection attempts with wrong credentials will be rejected.

Moreover, there are three kinds of authentication available [3]:

- Internal authentication: the column-oriented database gives users access to verifying credentials such as username and password.
- External authentication: Another proper network protocol in charge to authenticate the user's identity using a ticketing system. For example, Kerberos authentication.
- Client-Server Encryption: Client-to-server or node-to-node encryption, with trusted certificate.

There is no protection available for newly added columns or column families, only existing column families are protected after each authorization.

#### 2. Authorization:

Authorization is also pluggable which can be enabled via the authorizer setting in the "Cassandra.yaml" file. There are two options for the authorization; "AllowAllAuthorizer" which is default and provides permission to all users irrespective of their roles, and "CassandraAuthorizer" which only allows access to the privileged administrators. [2]

#### 3. Communication encryption:

There are multiple levels of encrypting data like auxiliary encrypted mode of communication from the "client machine" to the "database cluster", and "node-to-node encryption." The first ensures that data in the flight is not compromised and the latter makes sure that data is secured as it is transferred between database cluster nodes which can be customized by changing settings in the server encryption options in the Cassandra.yaml file [2].



#### 4. Auditing:

Higher than 4.0 versions of Cassandra have an audit logging feature to log all incoming CQL command requests. Using audit logging, an administrator can log the query provided by the client along with the execution timestamp and all other attributes [2].

#### 5. Vulnerability to DoS/Injection Attack:

Cassandra allocates one thread per client which prevents attackers from making the Cassandra server allocate all its resources to fake connection attempts. Users can create user-defined functions (UDFs) functionality to perform custom processing of data in the database. Where these UDFs are enabled, there is an instance for the system to be vulnerable to a DoS (Denial of Service) attack [2]. Furthermore, the current feature does not time out idle connections, so any open connections consume a thread affecting the performance [3].

### 2.3.4 Neo4j

Neo4j is the most popular open-source graph-based database where data are represented as nodes, relationships between nodes, and their properties. Neo4j allows for ultra-fast queries, a deeper context for analysis, and easily modifiable data relationships [2].

We will examine the security of Neo4j based on authentication, authorization, communication encryption, auditing, and vulnerability to Injection attacks.

Neo4j defines a set of predefined roles; reader (read, change password), editor (modify, create nodes and relationships, delete, no control over another user), publisher (same as editor, and create labels, types of relationships), architect (same as publisher and manage database's indexes and constraints), and admin (perform any operation within the database) [3].

#### 1. Authentication:

Authentication is enabled by default and uses a username and password (encoded in SHA-256 format). Neo4j includes a "native auth provider" that keeps the users and their role information in the database. It also provides a "Single Sign-On" provider and "Custom-built" plugin auth providers for clients for special requirements that native or LDAP cannot handle [2].

#### 2. Authorization:

There is role-based access control (RBAC) which allows the administrator to create users and grant them specific roles in the database. However, it is impossible to have different security privileges on different instances of a cluster [2]. This means that the clients have the same privilege no matter what server they access inside a cluster. Neo4j can optionally be extended by writing custom code, using Cypher language which guarantees two concepts: sandboxing (isolation of some parts of the system to prevent the use of unsafe APIs), and white-lists (limiting some extensions' loading when a wider library is invoked).

#### 3. Communication encryption:

Even though Neo4j does not support encryption explicitly while data is at rest, it supports securing data-in-transit by using TLS/SSL technology which is implemented by Java Cryptography Extension (JCE), a digital certificate, and a set of configurations provided in neo4j.conf file [2]. This technology has two features: (i) the SSL framework must explicitly state which are the allowed encryption techniques; (ii) a certificate authority must be integrated within the system. [3]

#### 4. Auditing:

Neo4j has limited auditing in the open-source version, whereas there is an option for logging in to the Enterprise version. General root files are stored and can be configured via the "dbms.directories.logs" file. Audit logs also include security events such as login attempts, authorization failures, and security procedures that run towards the system database [2].

#### 5. Vulnerability to DoS/Injection Attack:

Neo4j includes prevention against cypher injection attacks by sending input as a parameter to the query. This lets placeholders be used for parameters, and their values are supplied at execution time which also improves the execution times. Since Neo4j uses Cypher (CQL) declarative graph query and Cypher is vulnerable to injection, it makes Neo4j vulnerable to injection attacks by using string concatenation [2].

## 2.4 Comparison of Databases

After reviewing the security and privacy features of the four databases, we have found that the relational (Oracle) database has high and robust security features compared to other NoSQL databases which is shown in Figure 3. Oracle tends to support all the different security features we have considered for this study in addition to complying with ACID features. While comparing among three NoSQL databases, Apache Cassandra is found to be more secure offering encryption at rest and in transit to secure the data from unauthorized users. In MongoDB, only the MongoDB Enterprise allows encryption at rest while Neo4j does not offer encryption at rest by itself, the third party can only employ the encryption mechanisms at rest. In addition encryption and auditing features are available for only the MongoDB enterprise version. Due to the lack of proper encryption mechanisms and auditing features in the other two NoSQL databases, Cassandra is considered to be more secure.





No.	Database & Vendor/Developer	Authentication	Authorization & Access Control	Encryption			Consistency Model	Auditing and Logging
				Data-at-Rest	Client-Server communication	Internode communication		
Relational Database Systems (RDBMS) - Oracle								
1	Oracle <i>by Oracle Corporation</i>	Yes. Implemented through OS or network-based authentication either using SSL, Kerberos, PKI or directory services.	Yes. Different system and object-level privileges. User roles and profiles with fine-grained access control.	Yes. Using Transparent Data Encryption (TDE) released with version 12c.	Yes. Using AEC and 3DES with different Helman key negotiation algorithm.		ACID	Yes. Different types of auditing mechanisms available.
								
Wide Column Stores - Cassandra								
2	Apache Cassandra <i>by Avinash Lakshman and Prashant Malik, and lately by Apache Software Foundation Project</i>	Yes, using password-based authentication, or external mechanism such as Kerberos or LDAP.	Yes, using role-based permissions.	Yes, using Transparent Data Encryption (TDE), however this supports in Datastax Enterprise version only.	Yes, using SSL		BASE	Available with Datastax Enterprise version.
								
Dcooment Stores - MongoDB								
3	MongoDB <i>by MongoDB Inc</i>	Yes, supports multiple authentication mechanisms including Salted Challenge Response Authentication Mechanism (SCRAM) and certificate based (x.509) authentication. MongoDB Enterprise supports LDAP and Kerberos.	Yes, role-based access control along with permissions.	Available with MongoDB Enterprise version only and it uses AES-256 CBC or AES-256 GCM mode.	Yes, using TLS/SSL with minimum of 128-bit key length for all connections.		ACID	Auditing is available only for MongoDB Enterprise and it has the features to audit and log DDLs, replica sets and shared cluster, and CRUD operations along with authentication and authorization.
								
Graph Database - Neo4j								
4	Neo4j <i>by Neo Technology</i>	Yes, using password-based, or LDAP based authentication,or Kerberos authentication and single sign-on.	Yes, using role-based access control and permissions.	No. Can be used with 3rd party volum encryption mechanisms.	Yes, using TLS/SSL		ACID	Yes, enabled with two types of logging for inspection of queries and security events
								

Figure 3: Security comparison for four databases [4].

### 3 Conclusion

This research delved deeper into a comprehensive analysis of the security and privacy of relational (Oracle) and NoSQL (MongoDB, Cassandra, Neo4j) databases regarding their security and privacy features. There are different security and privacy features for different databases. This paper reveals those nuanced considerations for data administrators and developers.

The relational database, Oracle, follows ACID properties and it provides robust security features in regard to authentication, authorization, encryption, and auditing in its open-source version. It offers comprehensive authentication including single sign-on, Transparent Database Encryption for data at rest, and TLS for secure communication. These features are crucial for companies to provide a high level of data security, however, there might be scalability and flexibility issues as relational databases such as Oracle is a non-distributed database.

MongoDB is a popular document-based NoSQL database that provides flexibility and scalability while having a secure mechanism through third-party authentication like Kerberos, LDAP, and X.509 certificates. Only the enterprise version provides an option to encrypt data at rest and in transit to ensure data security.

Cassandra is a distributed column-oriented NoSQL database, that also provides a robust authentication, access control, and communication encryption mechanism. The open-source version provides features like Pluggable/external authentication, role-based access control (least privileged access control), and SSL/TLS encryption for data transfer. The enterprise version further enhances these security measures by adding additional features like advanced auditing.

Neo4j is a graph-based NoSQL database that has its own unique security solutions such as role-based access control with predefined roles and SSL/TLS encryption. While authentication and authorization are well-supported in both versions, auditing may vary based on version.

I was able to get insights into the security of different databases behind modern web applications. I always wondered if the data we give to the third party application are secured enough that they cannot be intervened by someone in the middle. This research helped me understand different security features. Furthermore, this research helped me deepen my understanding of which database to choose if I were to design a web application for my project. What security aspect will I need to consider for choosing a database and what can be done to further enhance data confidentiality, integrity, availability, vulnerability, and mitigation strategies? These are the questions that I now have answers to through our research.

For someone working in this area, I think it is essential to stay updated with the latest security vulnerabilities, and best practices through the latest research papers. As security trends are evolving with every new web application, there is also a need for data administrators to be updated with the latest papers.

In conclusion, this research not only expanded my technical knowledge but also helped me to proactively take security measures for safeguarding sensitive data. It also serves as a reminder that database security is an ongoing journey of learning, adaptation, and improvement to mitigate modern security threats.

## References

- [1] Sahatqija, K., Ajdari, J., Zenuni, X., Raufi, B., & Ismaili, F. (2018). Comparison between relational and NoSQL databases. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 0216-0221). doi: 10.23919/MIPRO.2018.8400041
- [2] Ankomah, E., et al. (2022). A Comparative Analysis of Security Features and Concerns in NoSQL Databases. In E. Ahene & F. Li (Eds.), *Frontiers in Cyber Security. FCS 2022. Communications in Computer and Information Science*, 1726. Springer. doi: 10.1007/978-981-19-8445-7\_22
- [3] Sicari, S., Rizzardi, A., & Coen-Porisini, A. (2022). Security & privacy issues and challenges in NoSQL databases. *Computer Networks*, 206, 108828. doi: 10.1016/j.comnet.2022.108828
- [4] Samaraweera, G. D., & Chang, J. M. (2021). Security and Privacy Implications on Database Systems in Big Data Era: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 33(1), 239-258. doi: 10.1109/TKDE.2019.2929794
- [5] Mohamed, M., Altrafi, O., & Ismail, M. (2014). Relational Vs. NoSQL databases: A survey. *International Journal of Computer and Information Technology (IJCIT)*, 03, 598. URL: [https://www.researchgate.net/publication/263272704\\_Relational\\_Vs\\_NoSQL\\_databases\\_A\\_survey](https://www.researchgate.net/publication/263272704_Relational_Vs_NoSQL_databases_A_survey)
- [6] Srinivas, S., & Nair, A. (2015). Security maturity in NoSQL databases - are they secure enough to haul the modern IT applications? In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (pp. 739-744). doi: 10.1109/ICACCI.2015.7275699
- [7] Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., & Abramov, J. (2011). Security Issues in NoSQL Databases. In *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications* (pp. 541-547). doi: 10.1109/TrustCom.2011.70
- [8] Trivedi, D., Zavarsky, P., & Butakov, S. (2016). Enhancing Relational Database Security by Metadata Segregation. *Procedia Computer Science*, 94, 453-458. doi: 10.1016/j.procs.2016.08.070
- [9] Oracle. (2024). Retrieved from <https://www.oracle.com/>
- [10] MongoDB. (2024). Retrieved from <https://www.mongodb.com/>
- [11] Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Motwani, R., ... & Xu, Y. (2005). Two can keep a secret: A distributed architecture for secure database services. In *CIDR* (Vol. 2005, pp. 186-199). Retrieved from [https://theory.stanford.edu/~gagan/papers/storage\\_CIDR.pdf](https://theory.stanford.edu/~gagan/papers/storage_CIDR.pdf) (Accessed: April 30, 2024).
- [12] Byun, J. W., & Li, N. (2008). Purpose-based access control for privacy protection in relational database systems. *The VLDB Journal*, 17(4), 603-619. doi: 10.1007/s00778-006-0023-0 (Accessed: April 30, 2024).
- [13] Chauhan, D., & Bansal, K. (2017). Using the advantages of NoSQL: A case study on MongoDB. *International Journal on Recent and Innovation Trends in Computing and Communication*, 5, 90-93. Retrieved from [https://www.researchgate.net/publication/349110376\\_Using\\_the\\_Advantages\\_of\\_NoSQL\\_A\\_Case\\_Study\\_on\\_MongoDB](https://www.researchgate.net/publication/349110376_Using_the_Advantages_of_NoSQL_A_Case_Study_on_MongoDB) (Accessed: April 30, 2024).
- [14] Babitz, K. (2023). SQL vs NoSQL databases: Key differences and practical insights. DataCamp. Retrieved from <https://www.datacamp.com/blog/sql-vs-nosql-databases> (Accessed: April 30, 2024).
- [15] Author Unknown. (2023). Comparing database management systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others. Retrieved from <https://www.altexsoft.com/blog/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/> (Accessed: April 30, 2024).
- [16] Dutta, S. (2023). MongoDB vs Oracle: A comparative analysis of two leading database systems. sprinkle. Retrieved from <https://www.sprinkledata.com/blogs/mongodb-vs-oracle-a-comparative-analysis-of-two-leading-database-systems> (Accessed: April 30, 2024)