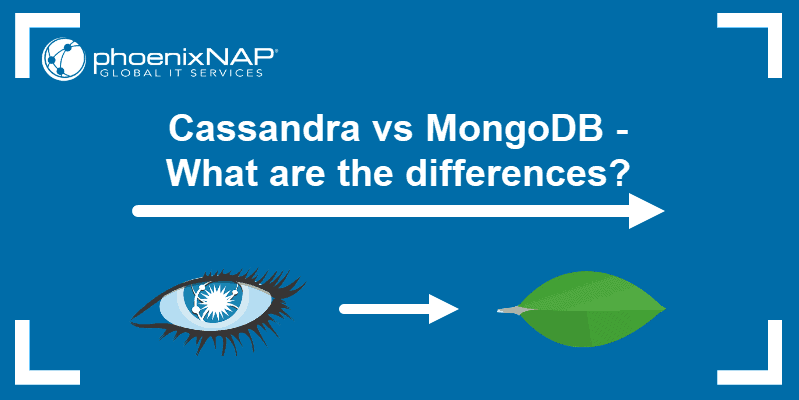
**Cassandra Vs MongoDB - What Are the Differences?**

**Introduction**

If you are searching for a NoSQL database, you probably came across Cassandra and MongoDB. Still, these two popular NoSQL choices have much less in common than expected.

**In this tutorial, we explain the similarities and differences between Cassandra and MongoDB.**



**Cassandra vs MongoDB: Similarities**

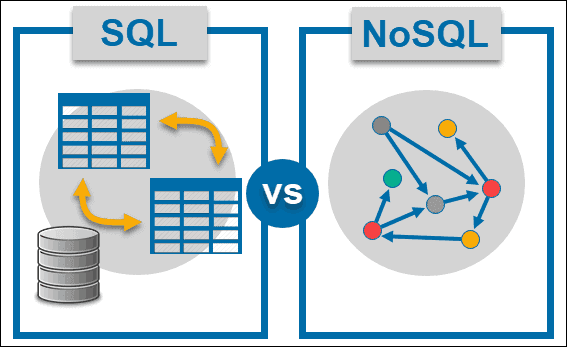
When making a comparison between two database systems, it is usually inferred there are shared similarities as well. Although they do exist, in regards to Cassandra and MongoDB, these similarities are limited.

**NoSQL Databases**

Most importantly, Cassandra and MongoDB are classified as NoSQL databases. NoSQL (Not only SQL) is a popular alternative to traditional databases. Unlike the relational databases we know, NoSQL can store large amounts of data without requiring a logical category or schema.

Since traditional databases weren’t able to handle a lot of unstructured data in real-time, NoSQL databases took up the challenge by scaling horizontally.

Accordingly, Cassandra was released in 2008, as one of these NoSQL databases. A year later, MongoDB was created.



**Open-Source Software**

Another commonality between these two is that they are free, open-source software. You can download the database packages, set up, and configure them at no expense.

Initially created by developers from Facebook, Cassandra is now under the ownership of the Apache project and part of its open-source community. On the other hand, MongoDB is one of the most popular database management systems in the world with a strong community of MongoDB developers.

**Cannot Replace RDBMS and ACID**

Bear in mind that neither Cassandra nor MongoDB can replace a traditional Relational Database Management System (RDBMS). If you need to store data using rows and columns, in a structured format, stick to one of the many available relational databases.

Additionally, if you need ACID-compliant databases, NoSQL is probably not the best solution. For database transactions that ensure atomicity, consistency, isolation, and durability, it is better to use relational databases, such as [MySQL or PostgreSQL](https://phoenixnap.com/kb/postgres-vs-mysql).

**Cassandra vs MongoDB: Differences**

**Data Availability**

One of the most significant differences between MongoDB and Cassandra is their strategy concerning data availability. This feature dependents on the number of master slaves in a cluster.

**MongoDB** has a single master directing multiple slave nodes. If the master node goes down, one of the slave nodes takes over its role. Although the strategy of automatic failover does ensure recovery, it may take up to a minute for the slave to become the master. During this time, the database isn’t able to respond to requests.

**Cassandra**, on the other hand, uses a different model. Instead of having one master node, it utilizes multiple masters inside a cluster. With multiple masters present, there is no fear of any downtime. The redundant model ensures high availability at all times.

**Conclusion:**[If your application requires high availability](https://phoenixnap.com/blog/what-is-high-availability) and depends on instant request responses, Cassandra is the more suitable option. Still, make sure you have the [server resources](https://phoenixnap.com/servers/dedicated) to facilitate such a setup. If a 30-40 second delay doesn’t affect your business, there is no need to prioritize high availability and burden the system with additional infrastructure.

**Scalability**

Scalability is a feature directly linked to the cluster model. Hence, Cassandra and MongoDB have significant differences between their writing scalabilities.

Only the master node can write and accept input. In the meantime, the slave nodes are only used for reads. Accordingly, as **MongoDB** has a single master node, it is limited in terms of writing scalability.

Having multiple master nodes increases **Cassandras** writing capabilities. It allows this database to coordinate numerous writes at the same time, all coming from its masters. Therefore, the more master nodes there are in a cluster, the better the write speed (scalability).

**Conclusion:**If you consider writing speed and scalability a priority, consider going with Cassandra.

**Data Model**

Now, let’s examine the data model of these two NoSQL databases.

**MongoDB’s** data model is categorized as object and document-oriented. This means it can represent any kind of object structures which can have properties or even be nested for multiple levels.

When it comes to Cassandra, there is a more traditional model. **Cassandra** has a table structure using rows and columns. Still, it is more flexible than relational databases since each row is not required to have the same columns. Upon creation, these columns are assigned one of the available [Cassandra data types](https://phoenixnap.com/kb/cassandra-data-types), ultimately relying more on data structure.

**Conclusion:**If you need a rich data model, MongoDB may be the better solution. Its unstructured architecture gives you more flexibility and the opportunity to arrange objects within the given hierarchy.

**Query Language**

Another distinguishing factor is whether you need a database that has query language support.

**MongoDB** uses queries structured into JSON fragments and does not have any query language support yet. If you or your team is used to SQL, this will be something to get used to. However, it is easy enough to manage.

Unlike MongoDB, **Cassandra** has its own query language called CQL (Cassandra Query Language). Its syntax is similar to SQL but still has some limitations. Essentially, the database has a different way of storing and recovering data due to it being non-relational.

**Conclusion:**If having query language support is a must, Cassandra can meet your needs.

**How are Queries Different?**

In the examples below you can see how queries in MongoDB differ from the ones used in Cassandra (while working in a demo *employee*table).

**Selecting records** from the employee table:

*MongoDB*

‘db.employee.find()’

*Cassandra*

‘SELECT \* FROM employee;’

**Inserting records** into the employee table:

*MongoDB*

‘db.employee.insert({ empid: '101', firstname: 'John', lastname: 'Doe', gender: 'M', status: 'A'})’

*Cassandra*

‘INSERT INTO employee (empid, firstname, lastname, gender, status) VALUES('101', 'John', 'Doe', 'M', 'A');’

**Updating records** in the employee table:

*MondgoDB*

'db.Employee.update({"empid" : 101}, {$set: { "firstname" : "James"}})'

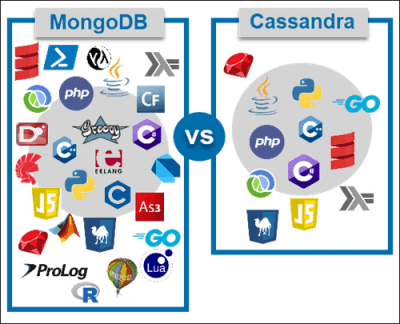
*Cassandra*

‘UPDATE employee SET firstname = ‘James' WHERE empid = '101';’

**Supported Programming Languages**

**MongoDB:**Actionscript, C, C#, C++, Clojure, ColdFusion, D, Dart, Delphi, Erlang, Go, Groovy, Haskell, Java, JavaScript, Lisp, Lua, MatLab, Perl, PHP, PowerShell, Prolog, Python, R, Ruby, Scala, Smalltalk

**Cassandra:**C#, C++, Clojure, Erlang, Go, Haskell, Java, JavaScript, Perl, PHP, Python, Ruby, Scala



**Conclusion:**The choice of programming language comes down to your experience, project requirements (i.e., amount of data and types of queries) and available frameworks. Generally, both MongoDB and Cassandra support a wide range of programming languages. It is believed that MongoDB works best with [Node.js](https://phoenixnap.com/kb/update-node-js-version), but it is very difficult to single out a particular programming language.

**Aggregation**

Deciding between MongoDB or Cassandra may also come down to whether or not you want a built-in aggregation framework.

**MongoDB** has a [built-in aggregation framework](https://docs.mongodb.com/manual/aggregation/). This feature allows it to retrieve data by utilizing an ELT multi-stage pipeline to transform the documents into aggregated results. However, such a framework is only efficient when working with small or medium-sized data traffic.

**Cassandra** has no aggregation framework and requires external tools like [Hadoop](https://phoenixnap.com/kb/install-hadoop-ubuntu), [Spark](https://phoenixnap.com/kb/install-spark-on-ubuntu) and others.

**Conclusion:**An in-house aggregation framework can only be found in MongoDB. If you are expecting small or medium-sized data traffic and do not want to involve external tools, MongoDB has the upper hand in this regard.

**Schema**

When it comes to the schema, you should decide whether you want a flexible database or a stationary one.

**MongoDB** is a database that does not require a schema, naturally making it more adaptable to changes. In its prior versions, the default configuration did not enforce any schema at all. Today, you can decide whether you want a schema or not. Such flexibility means the database can input documents of different structures and interpret them once in the software.

**Cassandra** is a much more stationary database. It facilitates static typing and demands the categorization and definition of columns beforehand.

**Conclusion:**If you need flexibility in terms of schema, MongoDB would probably suit you better. On the other hand, Cassandra should be sufficient if you are not expecting too much deviation in the data structure.

**Secondary Indexes**

The quality of secondary indexes determines how efficiently you can access records in the database. The extent to which these indexes are supported is not the same in MongoDB and Cassandra.

**MongoDB** has high-quality secondary indexes. Due to its flexible data model and secondary indexes, it can access any property of a stored object (even when it is nested).

Alternatively, **Cassandra** only has cursor support for the secondary index. Its queries are limited to single columns and equality comparisons.

**Conclusion:**The decision between the two depends on how you will query. If it is mostly by the primary index, Cassandra will do the job. If you need a flexible model with efficient secondary indexes, MongoDB would be a better solution.

**Performance**

There are a number of factors that impact the performance of these two types of databases.

Mainly, the **database model (or schema)** makes a big difference in performance quality as some are more suitable for MongoDB while others may work better with Cassandra.

What’s more, the **load characteristic** of the application your database needs to support also plays a crucial role. If you are expecting heavy load input, Cassandra, with its multiple master nodes, will give better results. With heavy load output both MongoDB and Cassandra will show good performance.

Finally, many consider MongoDB to have the upper hand when it comes to **consistency requirements**. Still, this may vary depending on the application. Also, you can manually configure Cassandra to meet the consistency standards you set.

**Conclusion**

After reading this article, you should have a better understanding of the difference between Cassandra and MongoDB. Ultimately, the decision between these two NoSQL databases will depend on your needs and the model your application requires.