THE BEGINNER'S GUIDE TO MongoDB

Table of Contents

Introduction

This e-book is a general overview of MongoDB. It's intended to give you a basic understanding of the database. The first half of the book focuses on advantages and drawbacks, sample use cases, and alternate solutions for big data. The second half of the book focuses on the technical side of MongoDB.

In many sections, we link to published articles on BMC Blogs and our MongoDB Guide to provide you with more in-depth information, tutorials, and code.

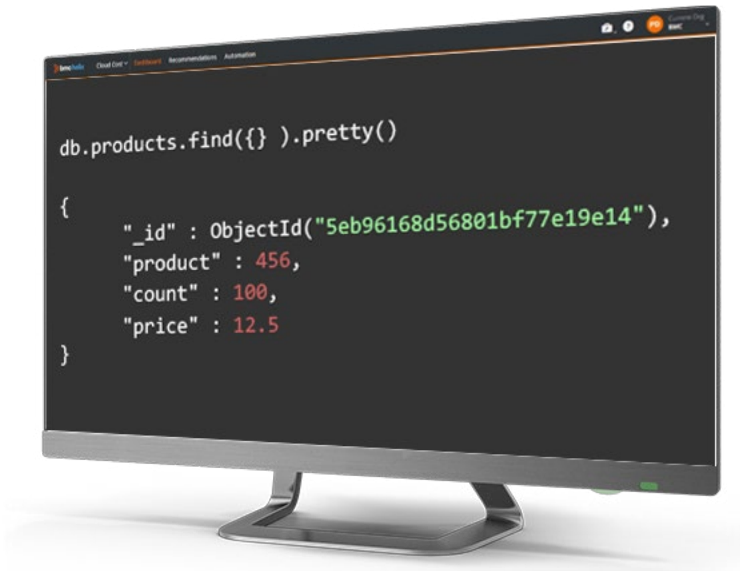
What's MongoDB?

MongoDB is a **JSON, NoSQL, big data** database. MongoDB is available to run in the cloud or on premises, with both free and pay-to-use versions.

First released in 2009, MongoDB solved a problem that most companies face: how to store data that varies in each record. This differs from the row-and-column format of traditional relational database management systems (RDBMS), where all records are assumed to be the same.

We illustrate these concepts with hands-on example in
MongoDB Overview: Getting Started with MongoDB

For example, here is a MongoDB **document.**
You list documents using the **find()** function:



```
db.products.find({} ).pretty()

{
        "_id" : ObjectId("5eb96168d56801bf77e19e14"),
        "product" : 456,
        "count" : 100,
        "price" : 12.5
}
```

In an RDBMS, that would look like this, using SQL (Structured Query Language):

```
select * from products;
```

id	product	count	price
2212121	456	100	12.5

Old Data vs Big Data

To understand big data and MongoDB, let's explain what we can call old data.

SQL databases and old data

Traditional SQL database vendors might use the label **big data** on their products to ride the big data wave, but what they're selling often isn't big data—it's old data. At its simplest, **old data** refers to RDBMS, SQL databases. These run on physical servers or mainframes. By far, the two most common SQL databases are Oracle and IBM's Db2®. SAP is also a major player thanks to their in-memory SQL database, SAP Hana.

Traditional SQL databases have **schema**, meaning there are fixed rules on how the data is structured. SQL data is organized in rows and columns and across tables or sheets. As you collect more data, your spreadsheet grows, but not every row or column applies to every item.

To simplify this problem, you may create different sheets for different data—but you're still stuck with too many items. This adds complexity, resulting in less flexibility to add features and inefficient computing when pulling data from a variety of locations.

SQL databases today

Today database sales like Oracle are declining, in part because open-source options such as PostgreSQL and MySQL are free. Despite these changes, many large companies continue to run SQL databases because they are excellent for handling heavy transactional data—like ERP systems for scheduling systems, inventory, sales, and order entry. SQL is easy to understand, and the volume of data in even the largest ERP systems is still relatively small.

For larger, more complex applications, like social media, survey systems, IoT, search engines, and geolocation, SQL is not a good fit: such highly variable data, **unstructured data**, does not fit into a rigid schema easily.

Old data, or structured data, are stored in RDMBS, SQL databases.

Unstructured data accounts for nearly of 90% data created today.

NoSQL databases and big data

That's where NoSQL databases come in. Big data, also known as **unstructured data**, is generally defined by a lack of **schema**—there's no rigid structure. Unlike old data, big data can scale almost without limit. (This second feature is because organizations like Yahoo, Google, and Stanford University have developed and given away software that lets systems run across two or more servers.)

That freedom from rigid rules makes sense in an application where not all the records require the same data. For example, in a sales system every customer has an address and taxpayer ID. But in a NoSQL database, you can still have an address and taxpayer ID—and you can add any other data.

With **structured data**, as in SQL databases, if you want to add a new column to a database table, you must rebuild the table and run a conversion. That's a lot of trouble.

With MongoDB you can add new fields at will and as much data to a single document as you need.

MongoDB uses JSON in place of a schema. Short for JavaScript Object Notation, JSON is completely free form, with no rigid rules as to structure. We say that it is **self-describing**. This means the field name and data value are both in the document, side-by-side. So, you can read what any field means just by looking at the name beside it.

With no schema in MongoDB, you can write:

```
{
anyThingYouWant":
"anyValueYouWAnt"
}
```

In a SQL database, the data **anyThingYouWant** would not fit because it's not in the schema. Only **fieldXXX** is there, so you would write something like:

```
create table blah:
( fieldname varchar(50)):
```

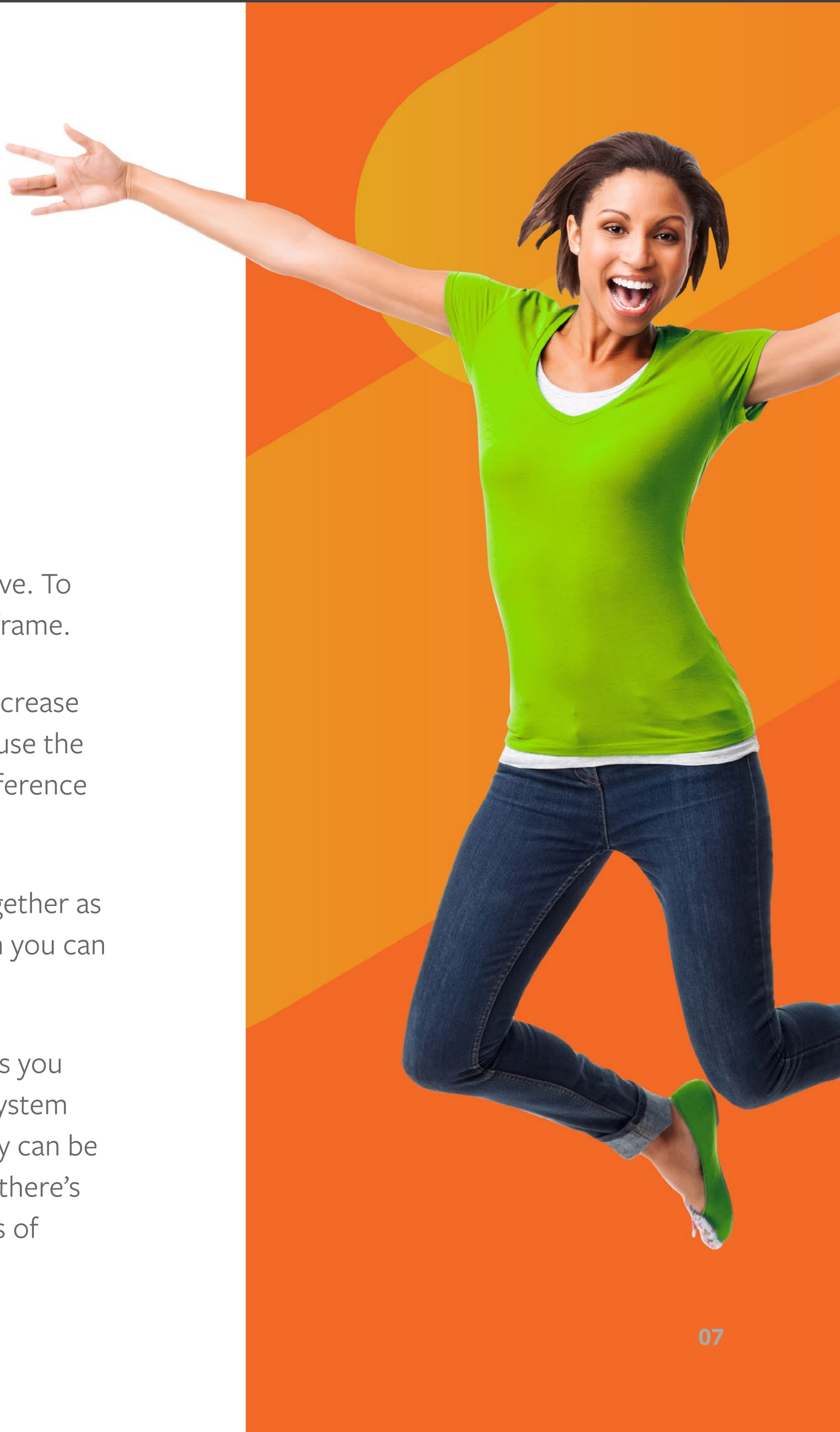
Learn more about data types and databases in:

Structured vs Unstructured Data

Big Data vs Analytics vs Data Science: What's The Difference?

SQL vs NoSQL Databases: What's The Difference?

MongoDB Benefits



Though SQL databases power countless companies, there are many benefits to using MongoDB.

Immensely scalable

SQL databases such as Oracle and Db2 are scalable, but this is expensive. To scale a mainframe, you must add costly memory or buy a bigger mainframe.

The NoSQL approach, however, is to use low-cost, PC hardware. To increase capacity, you simply add another server. Servers in giant data centers use the same PC architecture invented in the 1990s, and there's barely any difference between vendors. That's why it's called commodity hardware.

Recent changes in computing mean separate systems can function together as a single logical unit, replacing the need to buy one large machine when you can just have two cheap ones.

Open-source software, such as Apache Hadoop and Apache Spark, lets you scale data without expensive servers or mainframes. Hadoop is a file system and Spark is a database, but neither are limited to a single server—they can be tied into one logical unit across multiple servers. Using this approach, there's practically no limit to your application. Google, after all, uses hundreds of thousands of servers to support search, storage, and more.

Additional tools, like Apache Mesos and Kerberos, make it possible to spread memory across multiple machines. Think about that: memory is no longer limited to the size of one machine, overcoming a significant obstacle found with older systems.

MongoDB fits this model, too. You can increase MongoDB capacity by adding more servers to a cluster. Then fine tune the distribution by specifying what parts of the data to store in what servers using **sharding.** (See more in Clusters and Sharding.)

Increased speed

Old data databases are structured around the concept that data should exist in one place only. This concept, called **normalization**, means that data should not repeat in a database. This prevents the risk that one version of data is outdated, but it significantly slows the database speed.

Another reason for requiring normalization was cost. But cheap hardware means you no longer need to design for normalization—data can be located wherever you need it.

No more empty data

SQL stores empty data—metadata exists even if actual data does not. This wastes space and slows computing.

The obvious solution? Don't require all records to be the same length. MongoDB records are variable in length because there's no need to store empty columns, which also contributes to improved speed.

No expensive SQL operations

SQL databases use **view** and **join** operations. In a SQL database, you join two tables on some common element to achieve some goal. For example, if you want to show sales and prices together, you would join the sales and price tables by product number. Joining sets of data is an expensive operation: using significant cache, memory, and disk space.

To reduce computing time and resources, MongoDB does away with view and join operations altogether. Instead, you put related items together in a single document.

Easy development

The straightforward structure of MongoDB makes it easy for developers to learn and use. The mongo shell is an interactive JavaScript interface, which most programmers will appreciate, that allows for querying and updating data and performing administrative activities.

Despite this simple structure, its features and functions are robust enough to handle complex requirements, no matter the scale.

End-to-end security

MongoDB offers end-to-end security. Verify users via LDAP or AWS IAM, use the Hashicorp Vault to manage secrets, bring encryption keys with key management integrations, and establish network peering to cloud providers or use AWS PrivateLink.

MongoDB Use Cases

MongoDB is primarily used by companies seeking cost reduction and data optimization. Like any product, of course, MongoDB isn't perfect for every situation—it depends on your needs and expectations. Here are use cases when MongoDB may be just what you need:

IoT. The Internet of Things gathers metrics from devices, sensors, etc. This data must be free form as each device will capture different metrics. For example, in a preventive maintenance application a sensor can gather vibration. But an air quality application would gather particulate matter density. While these are widely different concepts, MongoDB lets you query them in a common way since the query language, like the database, is flexible.

Product catalogs. Products have different attributes. For example, a car has an engine size. A shirt can be made of silk, cotton, or polyester. MongoDB, as with any JSON database, lets you store objects whose attributes vary widely.

Geolocation operations. MongoDB supports the GeoJSON format, with points, polygons, and as built-in query methods to locate an item based on its proximity to a point on a map. (Learn how to query and work with geolocation data in MongoDB Geolocation Query Examples and Track Tweets by Geographic Location).

Alternative NoSQL Databases

MongoDB isn't the only NoSQL database on the market. Let's compare two other popular options, Elasticsearch and Cassandra.

Elasticsearch

Elasticsearch is very similar to MongoDB. Both are distributed JSON databases, but Elasticsearch is used primarily for performance monitoring. This is because Elastic has built parsers for hundreds of data sources to put disparate logs into JSON format. That enables searching differing data with a common query. Elasticsearch also has a graphical charting front end called Kibana.

Many companies use Elasticsearch the same way they would use MongoDB. It's the same kind of database with one notable difference: there is no interactive shell. Instead, you use JSON and HTTP to interact with it.

Elasticsearch is a good alternative for applications that don't use JavaScript. Learn more in our **Elasticsearch Guide.**

Cassandra

Unlike MongoDB, Apache Cassandra is the modern, highly scalable version of the relational database, albeit where data is grouped by column instead of rows, for fast retrieval. That does not, of course, fit all use cases. For many cases, keeping data together in rows works better.

Structure. Cassandra is a column-oriented database, whereas MongoDB stores records in JSON format. Still, Cassandra can support JSON in data fields.

Clustering. Cassandra has no configuration server to control the operation of other servers. Instead, a ring of servers each serve equal functions, but store different parts (i.e., shards) of the data.

Sharding. MongoDB and Cassandra both provide a fine level of control over sharding, but they do so differently.

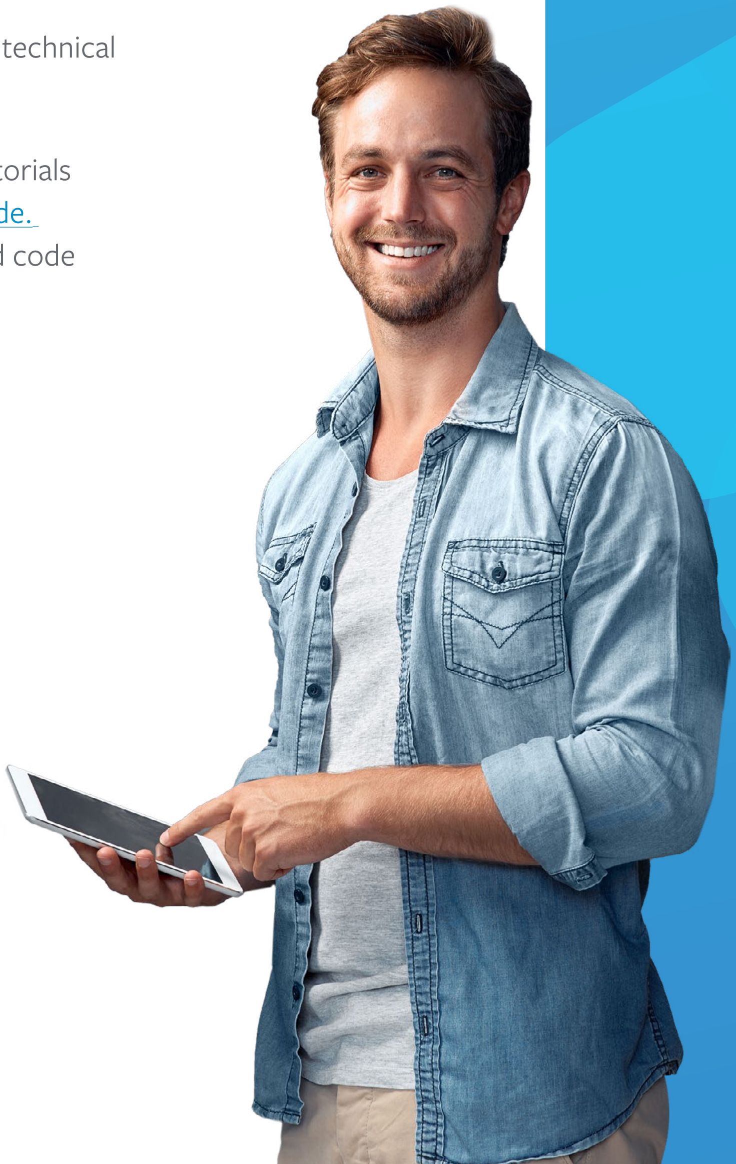
Replicating. Both MongoDB and Cassandra can replicate data, particularly useful for data consistency.

In MongoDB vs Cassandra: NoSQL Databases Compared we show the same operations in both databases to further illustrate how MongoDB works. Learn more about Cassandra in our **Cassandra Guide.**

Working in MongoDB

The second half of this e-book focuses on the technical side of MongoDB.

To complement this e-book, we have many tutorials and technical deep dives in our [MongoDB Guide.](#) We recommend you consult these for detailed code samples and hands-on demonstrations.



Installing MongoDB

If you're brand new to MongoDB, choosing and installing the software is your first step. MongoDB currently offers these options:

MongoDB Atlas is the cloud version, available as an on-demand, fully managed service that can run on AWS, Microsoft Azure, and Google Cloud Platform.

- The **Sandbox** version is free forever and a great place for training and ideating.
- The **Shared** version offers 5GB storage and shared RAM.
- The **Dedicated** version offers consistent performance, more advanced security, and unlimited scaling in dedicated clusters.

MongoDB runs on your server and requires a subscription.

- The **Community** version is feature rich and developer ready.
- The **Enterprise** version has more advanced features and increased performance.

In this guide, we use MongoDB Atlas, sandbox version.

Learn how in to install standalone MongoDB configurations and add and search data in [How To Install MongoDB on Ubuntu and Mac.]

Terminology

- MongoDB records are called **documents.** A document is the equivalent of a **row** in a SQL database. The document model maps to objects in your code, making it easier to work with your data.
- Each MongoDB database includes **collections**. A collection is a group of **documents**, like a table in an RDBMS database.
- Each collection and document have an **ObjectID**.

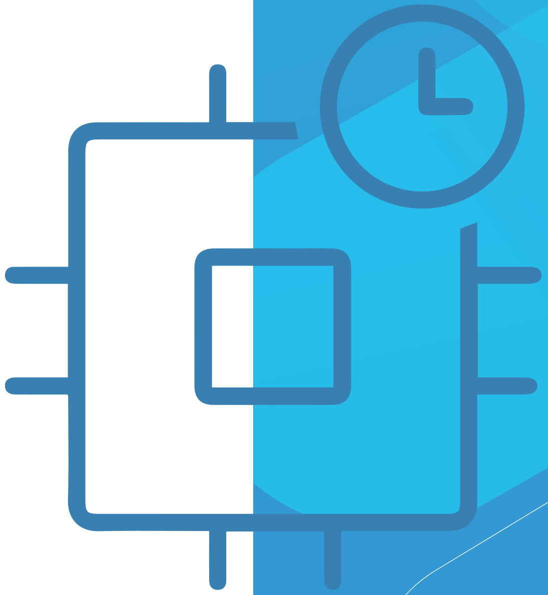
Memory Usage

MongoDB can quickly exhaust the memory on a server, so you'll need to know how to handle that. MongoDB is not an in-memory database, though you can configure it to run that way. But MongoDB makes liberal use of cache, meaning data records kept memory for fast retrieval.

Too much data in your MongoDB database will run your server out of memory. As the mongod daemon fills up its cache, the process will consume more and more memory. This can happen quickly—so quickly that you won't be able to shut down the mongod process because the bash shell will no longer respond. The solution is to add another node to your cluster.

In MongoDB Memory Usage, Management & Requirements, we illustrate:

- What a server looks like when it runs out of memory
- How to run queries and look at logs to prevent insufficient memory
- How to install free monitoring tools to help prevent this situation



Features and Functions

So, what can MongoDB do? Here are the main features, which we explain more fully in the following pages:

- Ad-hoc queries (Basic search functions)
- Indexing
- Real-time aggregation
- Clusters and Sharding
- Transactions

MongoDB also supports replication, load balancing, file storage, server-side JavaScript execution, and capped collections.

Ad-hoc queries (Basic searches)

MongoDB allows for a variety of search methods. Some ways of searching, or querying, in MongoDB are:

- By attribute
- Greater or less than
- Not equal to
- Projection, which returns or excludes only designated fields
- Regular expressions
- Elements in array

Get the code for all these and more in MongoDB Cheat Sheet.

Indexing

An **index** is a data structure that stores the location of a record on disk (or in memory or cached data). It tells the system from which address to find the data.

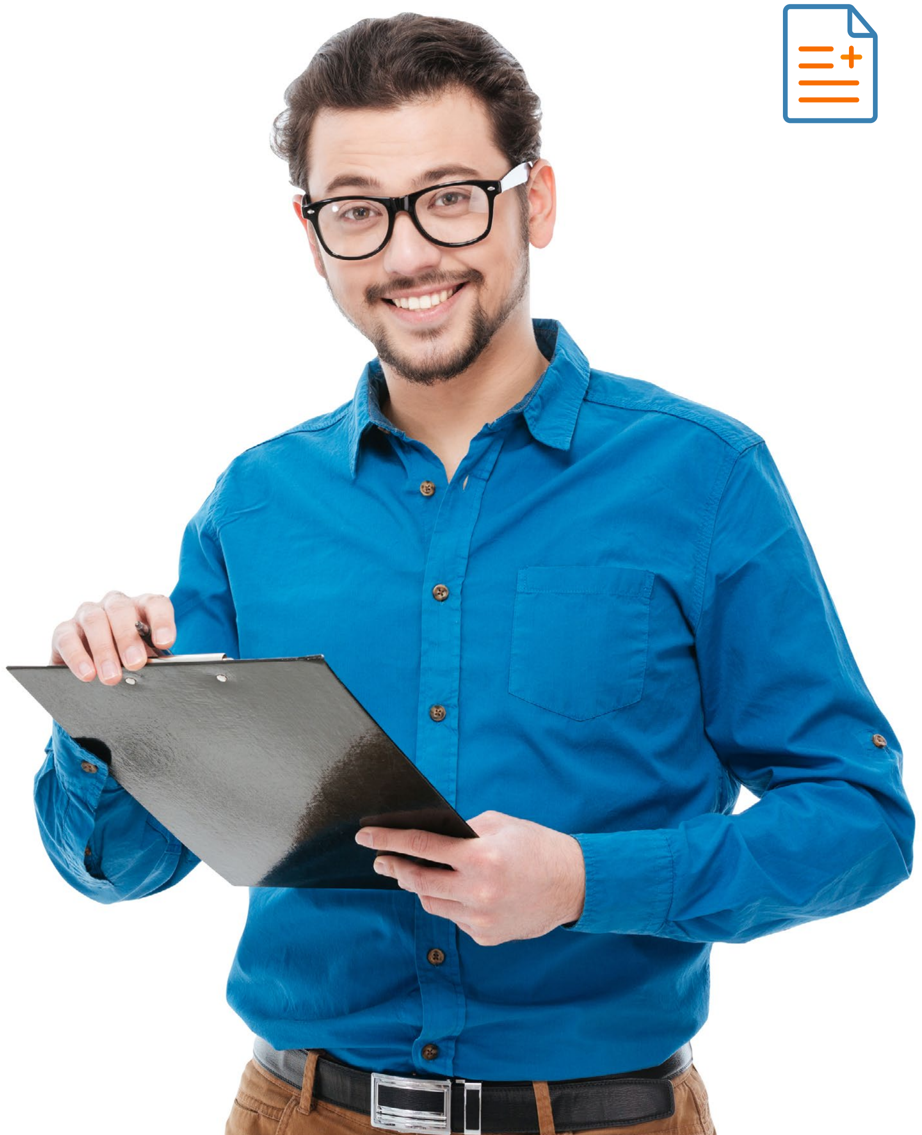
Creating appropriate indexes helps MongoDB maintain efficient computing. (Without indexes, MongoDB must scan every document, which slows the query significantly.)

For example, if you have a field product=456, and product is an **indexed field,** you can search for it quickly, because the computer knows that record is at, say, disk location FFFFFFX.

- A **single field index** lists data in ascending or descending order.
- A **sparse index** does not create an index when the document does not contain that field. This is to avoid needless index documents that have blank values.
- A **compound index** sorts fields inside other fields. For instance, indexing first on one attribute, then on a second.
- A **partial index** pulls documents that meet only a certain filter.

See more and get the code in [Introduction To MongoDB Indexes.]

Real-time aggregation



Aggregation operations group values from multiple documents together for processing and computing. You can also use aggregate functions to perform a variety of operations on the grouped data in order to return a single result.

In MongoDB you can perform aggregation in three ways:

- The aggregation pipeline
- The MapReduce function
- Single purpose aggregation

In MongoDB Aggregate Functions Explained, we use the WordCount program to illustrate aggregate functions.

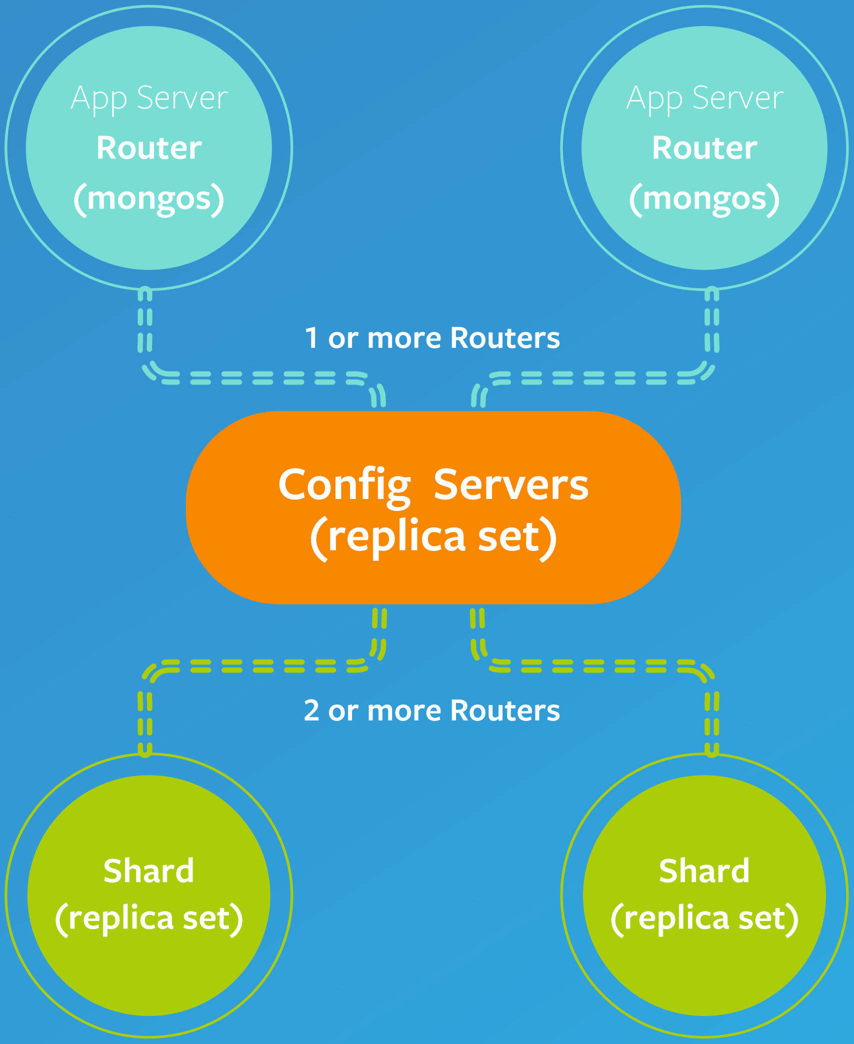
Clusters and Sharding

A **cluster** is how you build a distributed system, adding nodes when necessary as volume increases. **Sharding** means distributing data across a cluster. This is done by applying an algorithm across part or all of a document or index.

A cluster has three parts:

- Config server (which holds configuration information)
- Query router (aka mongos)
- Shard server (i.e., database)

This diagram shows how the mongos process runs as a router, meaning it tells clients where to look for data. Data is spread across the cluster based on **sharding**, the assignment of records to servers based on the hashed value of some index.



Transactions

A database **transaction** is any operation performed within a database, such as creating a new record or updating data within one. Changes made within a database need to be performed with care to ensure the data within doesn't become corrupted. The ACID concept—Atomicity, Consistency, Isolation, Durabillity—provides guidance on how to do this. (Learn more in ACID: Atomic, Consistent, Isolated & Durable)

Imagine if you have a sales order and inventory control system. Any sale you make should reduce on-hand inventory. So, what happens if the sales transaction works, but the inventory update fails? Then the database would no longer be consistent: the inventory would not match the sales.

The way to avoid that is to group the two transactions into one larger transaction. Learn how to install a transaction on a clustered MongoDB installation here
Introduction to MongoDB Transactions.

Author's bio

Walker Rowe is an American freelance tech writer and programmer living in Cyprus. He is also the founder of Hypatia Academy, an online school that teaches children computer programming. You can find him on his website and LinkedIn.

Editor's bio

Chrissy Kidd is a writer and editor in the technology sector, with more than 10 years of professional experience. She explains technical concepts, follows trends, and makes technology make sense to all of us. You can find her on LinkedIn.

About BMC
From core to cloud to edge, BMC delivers the software and services that enable over 10,000 global customers, including 84% of the Forbes Global 100, to thrive in their ongoing evolution to an Autonomous Digital Enterprise.

BMC—Run and Reinvent

www.bmc.com

523425