# Mongo Db

1. Installation

<https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/>

The core components in the MongoDB package are: [mongod](https://docs.mongodb.com/manual/reference/program/mongod/" \l "bin.mongod" \o "bin.mongod), the core database process; [mongos](https://docs.mongodb.com/manual/reference/program/mongos/#bin.mongos) the controller and query router for [sharded clusters](https://docs.mongodb.com/manual/reference/glossary/" \l "term-sharded-cluster); and [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#bin.mongo) the interactive MongoDB Shell.

* [mongod](https://docs.mongodb.com/manual/reference/program/mongod/)
* [mongos](https://docs.mongodb.com/manual/reference/program/mongos/)
* [mongo](https://docs.mongodb.com/manual/reference/program/mongo/)

[mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) is the primary daemon process for the MongoDB system. It handles data requests, manages data access, and performs background management operations.

MongoDB needs data directory to store data. Default path is /data/db

When you start MongoDB engine, it searches this directory which is missing in your case. Solution is create this directory and assign rwx permission to user.

If you want to change the path of your data directory then you should specify it while starting mongod server like,

mongod --dbpath /data/<path> --port <port no>

This should help you start your mongod server with custom path and port

You can specify the installation location for the executable by modifying the INSTALLLOCATION value.

By default, this method installs all MongoDB binaries. To install specific MongoDB component sets, you can specify them in the ADDLOCAL argument using a comma-separated list including one or more of the following component sets:

| **Component Set** | **Binaries** |
| --- | --- |
| Server | mongod.exe |
| Router | mongos.exe |
| Client | mongo.exe |
| MonitoringTools | mongostat.exe, mongotop.exe |
| ImportExportTools | mongodump.exe, mongorestore.exe, mongoexport.exe, mongoimport.exe |
| MiscellaneousTools | bsondump.exe, mongofiles.exe, mongoperf.ex |

The operation should return test, which is the default database. To switch databases, issue the use <db>helper, as in the following example:

copy

copied

use <database>

To list the available databases, use the helper show dbs. See also [db.getSiblingDB()](https://docs.mongodb.com/manual/reference/method/db.getSiblingDB/" \l "db.getSiblingDB" \o "db.getSiblingDB()) method to access a different database from the current database without switching your current database context (i.e. db).

You can switch to non-existing databases. When you first store data in the database, such as by creating a collection, MongoDB creates the database. For example, the following creates both the databasemyNewDatabase and the [collection](https://docs.mongodb.com/manual/reference/glossary/#term-collection) myCollection during the [insertOne()](https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/" \l "db.collection.insertOne" \o "db.collection.insertOne()) operation:

copy

copied

use myNewDatabase

db.myCollection.insertOne( { x: 1 } );

The [db.myCollection.insertOne()](https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/" \l "db.collection.insertOne" \o "db.collection.insertOne()) is one of the [methods available in the mongo shell](https://docs.mongodb.com/manual/reference/method/).

* db refers to the current database.
* myCollection is the name of the collection.

If the [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#bin.mongo) shell does not accept the name of a collection, you can use the alternative [db.getCollection()](https://docs.mongodb.com/manual/reference/method/db.getCollection/" \l "db.getCollection" \o "db.getCollection()) syntax. For instance, if a collection name contains a space or hyphen, starts with a number, or conflicts with a built-in function:

copy

copied

db.getCollection("3 test").find()

db.getCollection("3-test").find()

db.getCollection("stats").find()

**cursor**

A pointer to the result set of a [query](https://docs.mongodb.com/manual/reference/glossary/#term-query). Clients can iterate through a cursor to retrieve results. By default, cursors timeout after 10 minutes of inactivity. See [Iterate a Cursor in the mongo Shell](https://docs.mongodb.com/manual/tutorial/iterate-a-cursor/#read-operations-cursors).

**daemon**

The conventional name for a background, non-interactive process.

**data directory**

The file-system location where the [mongod](https://docs.mongodb.com/manual/reference/program/mongod/" \l "bin.mongod" \o "bin.mongod) stores data files. The [dbPath](https://docs.mongodb.com/manual/reference/configuration-options/" \l "storage.dbPath" \o "storage.dbPath) option specifies the data directory.

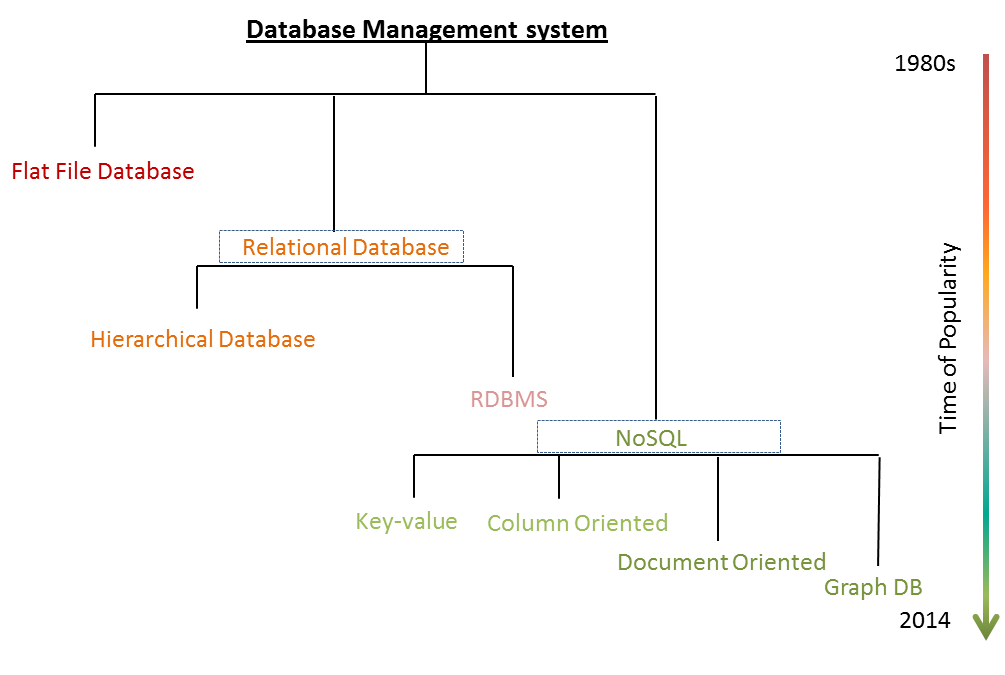
db.getCollection('ankitTask').find({"sysUnitId":"3216"})

**var** myCursor = db.users.find( { type: 2 } );

**while** (myCursor.hasNext()) {

print(tojson(myCursor.next()));

}



NoSQL databases offers many benefits, including:

* **Flexible Data Model.**Unlike relational databases, NoSQL databases easily store and combine any type of data, both structured and unstructured. You can also dynamically update the schema to evolve with changing requirements and without any interruption or downtime to your application.
* **Elastic Scalability.**NoSQL databases scale out on low cost, commodity hardware, allowing for almost unlimited growth.
* **High Performance.**NoSQL databases are built for great performance, measured in terms of both throughput and latency.

While MongoDB is great for application key-value store. I wouldn't recommend it as an analytics DB due to the following reasons:

* **No multi-document transaction**. This means it can’t guarantee consistency in some cases.
* **Can’t write queries that use more than one collection**. Aggregation framework only works on one collection at a time. Joining data has to be done programmatically and doesn't scale.
* **Nesting isn't always possible**, and there are no foreign key constraints to enforce consistency

Then two new Internet giants made breakthroughs, and developed their own distributed non-relational systems to help with this new onslaught of data: **MapReduce** ([published 2004](https://static.googleusercontent.com/media/research.google.com/en/archive/mapreduce-osdi04.pdf)) and **Bigtable** ([published 2006](https://static.googleusercontent.com/media/research.google.com/en/archive/bigtable-osdi06.pdf)) by Google, and **Dynamo** ([published 2007](http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf)) by Amazon. These seminal papers led to even more non-relational databases, including **Hadoop** (based on the MapReduce paper, [2006](https://en.wikipedia.org/wiki/Apache_Hadoop)), **Cassandra** (heavily inspired by both the Bigtable and Dynamo papers, [2008](https://en.wikipedia.org/wiki/Apache_Cassandra)) and **MongoDB** ([2009](https://en.wikipedia.org/wiki/MongoDB)). Because these were new systems largely written from scratch, they also eschewed SQL, leading to the rise of the NoSQL movement.

These NoSQL languages, being new, were also not fully developed. For example, there had been years of work in relational databases to add necessary features to SQL (e.g., JOINs); the immaturity of NoSQL languages meant more complexity was needed at the application level. The lack of JOINs also led to denormalization, which led to data bloat and rigidity.

Initially seduced by the dark side, the software community began to see the light and come back to SQL.

First came the SQL interfaces on top of Hadoop (and later, Spark), leading the industry to “back-cronym” NoSQL to “Not Only SQL” (yeah, nice try).

Then came the rise of NewSQL: new scalable databases that fully embraced SQL. **H-Store** [(published 2008](http://hstore.cs.brown.edu/papers/hstore-demo.pdf)) from MIT and Brown researchers was one of the first scale-out OLTP databases. Google again led the way for a geo-replicated SQL-interfaced database with their first **Spanner** paper [(published 2012](https://static.googleusercontent.com/media/research.google.com/en/archive/spanner-osdi2012.pdf)) (whose authors include the original MapReduce authors), followed by other pioneers like **CockroachDB** ([2014](https://en.wikipedia.org/wiki/Cockroach_Labs)).

At the same time, the **PostgreSQL** community began to revive, adding critical improvements like a JSON datatype (2012), and a potpourri of new features in [PostgreSQL 10](https://wiki.postgresql.org/wiki/New_in_postgres_10" \t "_blank): better native support for partitioning and replication, full text search support for JSON, and more (release slated for later this year). Other companies like **CitusDB** ([2016](https://www.citusdata.com/blog/2016/03/24/citus-unforks-goes-open-source/)) and yours truly (**[TimescaleDB](https://github.com/timescale/timescaledb" \t "_blank)**, [released this year](https://blog.timescale.com/when-boring-is-awesome-building-a-scalable-time-series-database-on-postgresql-2900ea453ee2)) found new ways to scale PostgreSQL for specialized data workloads.

<https://blog.timescale.com/why-sql-beating-nosql-what-this-means-for-future-of-data-time-series-database-348b777b847a>

aggregation:

SELECT hosting, SUM(hosting) AS total

FROM website

GROUP BY hosting

db.website.aggregate(

{

$group : {\_id : "$hosting", total : { $sum : 1 }}

}

);