1. MongoDB is a cross-platform, non-relational, "NoSQL" document oriented database that provides, high performance, high availability, and easy scalability.
2. The initial development of MongoDB began in 2007 when the company was building a platform as a service similar to window azure. MongoDB was developed by a NewYork based organization named 10gen which is now known as MongoDB Inc. It was initially developed as a PAAS (Platform As A Service). Later in 2009, it is introduced in the market as an open source database server that was maintained and supported by MongoDB Inc.
3. The first ready production of MongoDB has been considered from version 1.4 which was released in March 2010. MongoDB 4.0.9 - was the latest and stable version which was released on Apr 16, 2019. You can get more information at the official site mongodb.com;
4. So why MongoDB? MongoDB is an open source database. The chart shows the historical trend of the popularity of open source and commercial database management systems. So how can we see open source databases dominate in new approaches.
5. And MongoDb is a leader in popularity among nosql databases
6. There are several types of databases. Basically they are divided into sql and nosql. NOSQL databases use models such as key value (Memcache), document (MongoDb), wide column (Cassandra) etc. while relational databases use tables;
7. So let a look what is document oriented database, and we do this by compare relation database and mongodb.

Some benefits of this storage model are:

MongoDB works on concept of collection and document. MongoDB is a database that stores data records (documents) for use by an application. Mongo. This means Mongo stores all data associated within one record, instead of storing it across many preset tables as in a SQL database

Scalability: by default, non-relational databases are split (or "shared") across many systems instead of only one. This makes it easier to improve performance at a lower cost.

Flexibility: new datasets and properties can be added to a document without the need to make a new table for that data.

Replication: copies of the database run in parallel so if one goes down, one of the copies becomes the new primary data source.

## **Database**

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

## **Collection**

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

## **Document**

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

## **Sample Document**

Following example shows the document structure of a blog site, which is simply a comma separated key value pair.

**\_id** is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide \_id while inserting the document. If you don’t provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

Any relational database has a typical schema design that shows number of tables and the relationship between these tables. While in MongoDB, there is no concept of relationship.

## **Advantages of MongoDB over RDBMS**

* **Schema less** − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
* Structure of a single object is clear.
* No complex joins.
* Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
* Tuning.
* **Ease of scale-out** − MongoDB is easy to scale.
* Conversion/mapping of application objects to database objects not needed.
* Uses internal memory for storing the (windowed) working set, enabling faster access of data.

## **Why Use MongoDB?**

* **Document Oriented Storage** − Data is stored in the form of JSON style documents.
* Index on any attribute
* Replication and high availability
* Auto-sharding
* Rich queries
* Fast in-place updates
* Professional support by MongoDB

## **Where to Use MongoDB?**

* Big Data
* Content Management and Delivery
* Mobile and Social Infrastructure
* User Data Management
* Data Hub

MongoDB supports many datatypes. Some of them are −

* **String** − This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
* **Integer** − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
* **Boolean** − This type is used to store a boolean (true/ false) value.
* **Double** − This type is used to store floating point values.
* **Min/ Max keys** − This type is used to compare a value against the lowest and highest BSON elements.
* **Arrays** − This type is used to store arrays or list or multiple values into one key.
* **Timestamp** − ctimestamp. This can be handy for recording when a document has been modified or added.
* **Object** − This datatype is used for embedded documents.
* **Null** − This type is used to store a Null value.
* **Symbol** − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
* **Date**− This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
* **Object ID** − This datatype is used to store the document’s ID.
* **Binary data** − This datatype is used to store binary data.
* **Code** − This datatype is used to store JavaScript code into the document.
* **Regular expression** − This datatype is used to store regular expression.

Модели представления данных:  
-ориентированная

-реляционная

-на основе графов

now we're going to talk about MongoDB. MongoDB is pretty popular in corporate

environments. MongoDB is a popular choice in the corporate world in particular because it is built by an actual corporation that actually supports it as opposed to just being kind of out in the wild and open source. MongoDB s name comes from humongous data. Mongo humongous get it which is kind of weird because really what sets MongoDB apart is not the fact that it can handle big data but just it's document data model it's not going based data model which is very flexible and we'll talk about that more shortly so don't let the name fool you other no sequel databases do just as good of a job at managing

has a single master a single primary database that you have to talk to all the time to ensure consistency but if that master goes down it will result in a period of unavailability while a new primary database is put into place so the big thing that's different about MongoDB is that you can stick pretty much anything you want into MongoDB basically any JSON blob of data you can shove into a document in MongoDB it doesn't have to be structured you don't have to have the same schema across each document you can put whatever you want in there so here's an example of what an actual MongoDB document might look like let's say that we want to store blog posts in a MongoDB database well this is what it might look like and this is really what it looks like so MongoDB will automatically give you an underscore ID field let's just automatically appended to your document that contains some unique identifier for you and that's done because there is nothing in MongoDB that says that you have to have some unique field in your

document at all so within that document

02:12

we might have a title

02:13

the content of the blog post itself and

02:15

then we can have a comments field that

02:18

contains an array of other documents so

02:20

this is an example of an embedded

02:22

document where we have a document

02:24

representing a com comment that itself

02:27

contains the name email content and

02:28

rating and I could actually have

02:30

multiple of these embedded within this

02:32

blog post document so that's a little

02:35

concrete example of what a document

02:37

might look like in MongoDB like I said

02:41

no real schema is enforced in MongoDB at

02:43

all you can have different fields and

02:44

every document if you want to go

02:46

obviously not necessarily a good idea if

02:48

you want to actually do fast look ups

02:50

and database but you can you know you

02:53

don't have to have a single key value

02:55

like you would have to have in Cassandra

02:57

that's some unique identifier but you

02:59

can create indices in any fields that

03:00

you want you can also create indices on

03:03

combinations of fields so one nice thing

03:05

about MongoDB is that it's very flexible

03:07

in how you can index its data to achieve

03:10

fast lookups on whatever queries you

03:12

might be doing obviously if you want to

03:15

actually charge your MongoDB database

03:17

which is how they talk about actually

03:19

horizontally partitioning it so that you

03:21

have different ranges of data and on

03:23

different servers then you have to have

03:25

some unique index to do that shorting on

03:27

and we'll talk about that more in a bit

03:29

so with MongoDB you have a lot of

03:31

flexibility in what you can store in it

03:32

but with great power comes great

03:34

responsibility just because you can show

03:36

whatever you want into MongoDB doesn't

03:38

mean you should you still need to think

03:40

about what the queries are you're going

03:41

to be performing on this database and

03:44

design your database schema accordingly

03:46

so make sure that if there are think

03:49

about what indices you might need for

03:50

fast look ups for the queries you're

03:51

going to do at the end of the day it's

03:54

still a node sequel database so you

03:55

cannot do joins efficiently so you want

03:57

to make sure your schema is denormalized

03:59

as much as you can in MongoDB world we

04:03

talked about databases and collections

04:06

and documents instead of databases and

04:08

tables and rows so this kind of gets

04:10

away from the notion of there being some

04:11

sort of fixed schema which is kind of

04:13

implied in the words table and row so a

04:17

MongoDB database contains collections

04:19

any collection contains a collection of

04:22

documents so instead of tables

04:24

containing rows we have collections that

04:26

contain documents

04:27

conceptually you can think of them the

04:28

same way but just keep in mind that

04:31

collections can contain pretty much

04:32

anything and the main restriction here

04:34

is simply that you cannot move data

04:36

between collections across different

04:38

databases so if you do need to reference

04:41

data between different collections they

04:43

do need to be within the same database

04:46

so I can editorialize a little bit here

04:49

if you go to the MongoDB website you'll

04:51

see it's really aimed at more of a

04:52

corporate environment and kind of rubs

04:55

me the wrong way to be honest if you

04:56

look at the about MongoDB tab for

04:59

example it doesn't really tell you

05:00

anything concrete it says with MongoDB

05:03

these organizations move faster than

05:05

they could with relational databases at

05:06

one tenth of the cost with MongoDB you

05:08

can do things you could never do before

05:09

wow that sounds great too you know the

05:11

sort of CTO that hasn't written code in

05:13

20 years right but pathetical people

05:16

like you and me not really very helpful

05:18

kind of rubs me the wrong way but for

05:20

corporations this can be a good thing

05:21

you know you want to be able to pay for

05:23

a professional support and have

05:24

guarantees about support if you need it

05:26

so you know MongoDB has that sort of

05:28

service available to it and at the end

05:30

of the day it is still open source and

05:32

you can still get the documentation you

05:34

need as a developer if you just go

05:35

looking for it but and websites like

05:38

this just really bother me alright let's

05:41

talk about MongoDB architecture so the

05:44

first thing you need to understand with

05:45

MongoDB is what they call a replica sets

05:48

so like we said before MongoDB has a

05:51

single master architecture the idea

05:54

being that we want to have consistency

05:56

over availability but you can have these

05:59

secondary databases that maintain copies

06:01

over time from your primary database so

06:03

as rites happen to your primary database

06:04

those rights get replicated through an

06:06

operation log to any secondary nodes

06:09

that you might have attached to it so in

06:11

this talk in this diagram here we might

06:12

have a primary MongoDB server that your

06:16

application talks to you and maybe we

06:19

have a couple of secondary backup nodes

06:20

in one data center and a couple of

06:22

secondary backup nodes in some other

06:24

data center MongoDB will automatically

06:26

replicate those operations to those

06:29

secondaries so that in the event that

06:30

the primary goes down one of these

06:32

secondaries can take its place and the

06:35

right way that replication chain works

06:37

is kind of arbitrary

06:38

just tries to figure out which server

06:40

kind of talk to most quickly you know

06:42

where's it getting the fastest pain

06:44

times from so you know you don't

06:46

necessarily have this sort of structure

06:48

where you have a primary talking to a

06:49

secondary and another secondary

06:51

backing up from another secondary these

06:52

arrows could be pointing pretty much

06:54

anywhere in practice so the good thing

06:58

though is that if that primary does go

07:00

down a new secondary can be elected and

07:03

take its place within seconds so it

07:04

happens pretty quickly you're not

07:05

talking about massive amounts of

07:07

downtime in the event of a primary

07:08

failure but you do need to make sure you

07:11

get that primary back up online pretty

07:13

quickly because if your operation log

07:15

runs out of space during the time that

07:16

it's been down recovering that primary

07:18

is going to get a whole lot more

07:19

difficult so you know you need to make

07:22

sure that you're still you still have

07:24

some operational responsibilities to

07:26

actually get that back up and running

07:28

quickly and I want to stress again that

07:30

we haven't even talked about big data

07:32

yet what we're talking about here in

07:33

replica sets is just having a single

07:35

monolithic MongoDB server where all of

07:38

the data sits on that single server and

07:40

we're replicating that data to backup

07:41

servers okay so we're not talking about

07:43

big data yet we're just talking about

07:45

durability and actually having backup

07:47

copies of a single monolithic MongoDB

07:50

database here there are a lot of quirks

07:53

with MongoDB and it's a you know

07:55

something that it does get its share of

07:57

criticism for one thing is that you have

08:00

to have a majority of servers in your

08:02

set to agree on who the primary is so

08:05

you can't have an even number of servers

08:07

because you can't get a majority and

08:09

that implies that you need to have at

08:11

least three servers if you want to have

08:12

replication or some sort of durability

08:15

and I can get expensive right maybe it

08:17

doesn't make sense to actually have

08:18

three giant servers just to keep your

08:21

one monkey MongoDB instance reliable so

08:25

to get around that limitation they have

08:27

something called an arbiter node that

08:28

you can set up into place of a secondary

08:30

node where it's only job is to vote on

08:32

who the primary should be in the event

08:34

of a failure so that's an option but you

08:37

can only have one arbiter node in your

08:38

cluster so a little bit weird the other

08:41

thing is that your applications need to

08:43

know about at least a few servers in

08:45

your MongoDB cluster so it needs to know

08:47

about you know your current primary and

08:49

a few secondaries at least so it can

08:50

actually

08:51

ask one server who the primary is that

08:54

should be talking to so that means that

08:57

if you're going to be changing the

08:58

configuration of your servers or adding

09:00

more secondaries or removing secondaries

09:02

at the end of the day you need to push

09:04

that information all the way up to your

09:05

applications which can be kind of a pain

09:08

and again I want to stress that replica

09:10

sets only address durability we haven't

09:12

talked about scaling out to Big Data yet

09:13

if your if your replica set goes down

09:17

for whatever reason your your database

09:19

is down okay so there is a way to set

09:22

things up so that you can read from

09:23

secondaries but generally that's not

09:25

recommended so we're just talking about

09:28

durability here rather neat but one neat

09:31

thing about replica sets is that you can

09:32

set up something called a delayed

09:33

secondary and the idea there is that you

09:35

can set up a time delay between the

09:37

replication between your primary and a

09:39

specific secondary node and you can do

09:41

that as insurance against doing

09:42

something stupid so for example let's

09:45

say I set up a one hour delay between

09:47

primary and secondary replication and I

09:50

do something really dumb like

09:51

accidentally drop an entire database on

09:53

my MongoDB instance if I can catch that

09:55

quickly enough I can shut things down

09:57

and restore from that delayed secondary

09:59

to get back to where I was an hour ago

10:00

and restore that information relatively

10:03

quickly let's talk about Big Data that's

10:06

why we're here

10:07

so for actually scaling out data across

10:09

more than one server with MongoDB we

10:12

need to set up something called sharding

10:13

and the way sharding works is that we

10:16

actually have multiple replica sets

10:18

where each replica set is responsible

10:21

for some range of values on some indexed

10:23

value in my database so this in order to

10:27

get starting to work it requires that

10:29

you set up an index on some unique value

10:31

on your collection and that index is

10:35

used to actually balance the load of

10:38

information among multiple replicas sets

10:40

and then on each application server

10:43

whatever you're using to talk to MongoDB

10:46

you'll run a process called Mongo s and

10:48

Mongo s talks to exactly three

10:51

configuration servers that you have

10:52

running somewhere that knows about how

10:54

things are partitioned and then uses I

10:57

figure out which replicates that do I

10:58

talk to to get the information that I

11:00

want so let's take a look take a minute

11:03

to

11:04

and it's architecture here we can have

11:05

many application servers these might be

11:08

web servers on some big web app for

11:09

example where each process of your web

11:12

servers is running an instance of Mongo

11:15

s Mongo s has some communication with

11:17

three configuration servers you're

11:19

running somewhere

11:20

these can run on on top of other servers

11:23

you might have they'll have to do a

11:24

whole lot of work but you need to have

11:25

three of them and from there can figure

11:27

out which replicas set to talk to to

11:30

actually read or write the information

11:32

for a given say user ID or something

11:35

that you're indexing on and that

11:38

replicas set in turn can take care of

11:40

durability and actually backing that

11:42

data for that replicated replica set up

11:45

to a bunch of secondary nodes so they

11:46

can failover - now Mongo s is running

11:49

something called a balancer in the

11:52

background so over time if it finds that

11:55

it's actually doesn't have an even

11:56

distribution of values in whatever field

11:59

you're partitioning on it can rebalance

12:01

things across your replica sets in real

12:03

time over time so in this example we

12:06

might have replica set one that's set up

12:08

to handle user IDs you know from the

12:10

minimum value to user ID 1000 maybe

12:12

replica sets to is handling user IDs

12:15

1,000 to 5,000 and replica replicas set

12:18

3 might be handling user IDs 5,000 to

12:21

whatever the maximum value is so these

12:23

can change over time and get rebalanced

12:25

over time as the need arises so that is

12:30

how MongoDB handles big data you can see

12:32

it's actually pretty complicated but you

12:34

know if you actually to be fair if you

12:36

compare this to something like HBase

12:37

where you're using something like

12:38

zookeeper to maintain these sorts of

12:40

configuration it's not that different

12:43

charting itself has some quirks in

12:45

MongoDB so for example Auto sharding

12:48

where it's trying to rebalance thing

12:49

over time sometimes fails there is a

12:51

rather nasty failure mode called a split

12:53

storm where it simply cannot split

12:55

things quickly enough and it just keeps

12:57

trying to reel it things over and over

12:59

and over again and your entire cluster

13:01

goes down as a bad thing another failure

13:03

mode is if your Mongo s processes on the

13:05

front end get restarted too often things

13:07

will never rebalance so it actually

13:09

takes a look on each Mongo s process

13:11

over time to see how data is being

13:13

distributed throughout your cluster and

13:15

if you keep restarting it it basically

13:17

starts the clock restarts the count on

13:19

those things so if you are restarting

13:22

those processes too often and sometimes

13:24

depending on how you set up your web

13:26

server that might be pretty often that

13:27

might be the case things won't be

13:29

balanced properly so very easy to get

13:31

into a bad state got a better make sure

13:33

if someone's really keep an eye keeping

13:35

an eye on things from an administrative

13:36

standpoint you do need to have exactly

13:38

three config servers and if anyone goes

13:41

down your entire database goes down this

13:43

really isn't any different from HBase

13:44

where you have you know master nodes

13:46

that are maintained by a zookeeper

13:48

so again we're trading off intentionally

13:51

consistency for availability and the

13:55

other thing too is like I said before

13:56

even though MongoDB offers a very

13:58

loosely defined document model it

14:00

doesn't mean that your document model

14:01

should be loose if you're going to be

14:03

doing sharding and actually handling big

14:04

data you still need to think about

14:06

having some single primary key that is

14:08

unique to each document that you're

14:10

going to be starting on now let's kind

14:12

of talk a lot about the limitations of

14:15

MongoDB but there are some very neat

14:16

things about it too so again you know

14:19

the the big plus of MongoDB is that it's

14:22

not just a no sequel database but it can

14:24

store pretty much anything you want it

14:26

also has a shell that has a full

14:28

JavaScript interpreter so there's a lot

14:29

of power there you can do you can

14:31

actually run JavaScript functions across

14:34

your entire MongoDB database pretty

14:36

easily

14:36

it also supports many indices although

14:39

you're still discouraged from doing more

14:41

than two or three in a given collection

14:42

and you can only have one that's used

14:44

for sharding but you can actually set up

14:46

things like full-text indices for doing

14:48

efficient text searches across MongoDB

14:50

so again MongoDB is really a good choice

14:52

for things like storing you know big

14:54

documents of information or text you can

14:57

also have spatial indices regular you

14:58

can actually do searches across you know

15:00

latitudes and longitudes for example and

15:02

try to figure out what database objects

15:06

actually intersect a given position for

15:08

example which is kind of a neat feature

15:09

another thing about MongoDB that's worth

15:11

talking about is that they're kind of

15:13

trying to make MongoDB into a

15:14

replacement for Hadoop to some extent so

15:16

it actually has built-in aggregation

15:19

capabilities you can actually run

15:20

MapReduce code on MongoDB itself and it

15:24

actually has its own file system

15:25

built-in as well called grid FS that's

15:27

kind of like HDFS and some

15:29

then where it's storing documents within

15:31

Margo DB and actually chunking those

15:33

documents up kind of like HDFS does so

15:36

MongoDB is kind of value proposition is

15:39

in part the fact that for many

15:40

applications you might not need to do

15:41

but all MongoDB might be all that you

15:44

need but if you are integrating MongoDB

15:46

with Hadoop or smart or something like

15:47

that it's easy to do is we'll see in a

15:49

moment and the good thing is that it can

15:51

actually leverage some of these features

15:53

in MongoDB to do things more efficiently

15:55

so for example if we're tying MongoDB to

15:59

a spark data set and you're telling

16:01

spark to go perform some MapReduce EE

16:03

tasks on MongoDB that work might

16:06

actually get pushed down to MongoDB

16:08

itself so it might actually have to use

16:10

to do but all that can actually lead to

16:13

you know more efficient data analysis

16:15

and you might be able to get from other

16:16

no sequel solutions that are integrated

16:18

with something like a duper spark and

16:20

there is actually a sequel connector

16:22

available for MongoDB so you can

16:24

actually write full-blown sequel against

16:25

it if you want to but bear in mind it's

16:27

still not really a relational database

16:29

even if you have the ability of

16:31

executing sequel commands against it you

16:33

still can't do efficient joins and can't

16:35

deal with normalized data very

16:37

efficiently so with that we talked a lot

16:40

let's actually go play around with

16:42

MongoDB let's actually look at

16:43

integrating MongoDB with SPARC and get

16:45

some data into it and then we can play

16:47

around with the data in MongoDB and see

16:50

how it works from within the Mongo shell

16:51

so let's go have some fun