Mongo DB

MongoDB is a No SQL database. It is an open-source, cross-platform, document-oriented database written in C++.

Our MongoDB tutorial includes all topics of MongoDB database such as insert documents, update documents, delete documents, query documents, projection, sort() and limit() methods, create collection, drop collection etc. There are also given MongoDB interview questions to help you better understand the MongoDB database.

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.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

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 Installation and Setup Process (for Windows 32-bit):

Installation:

MongoDB Installation and Setup Process (for Windows):

Installation:

1. First install the MongoDB community edition server.

2. Then install MongoDB Compass if you want to work in MongoDB using GUI.

Setup:

1. First create a directory to save the MongoDB database files.

Let us create a directory at path C:\mongodb\data

2. After creating directory as shown in above step open command prompt window and type following command:

For Windows 32 bit

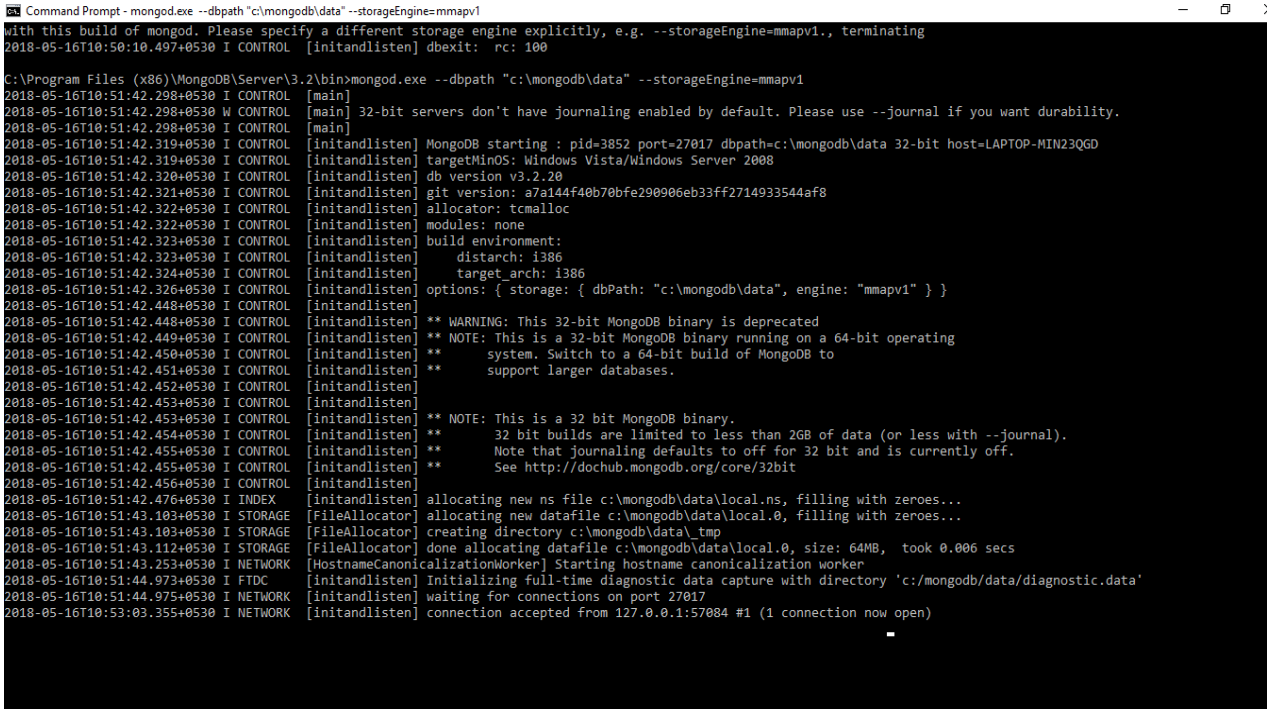
```
"C:\Program Files\MongoDB\Server\3.6\bin\mongod.exe" --dbpath "C:\mongodb\data"
```

For windows 64 bit

Now the MongoDB server will start and you can see at the end of command prompt window,

it will show "waiting for connections" message.

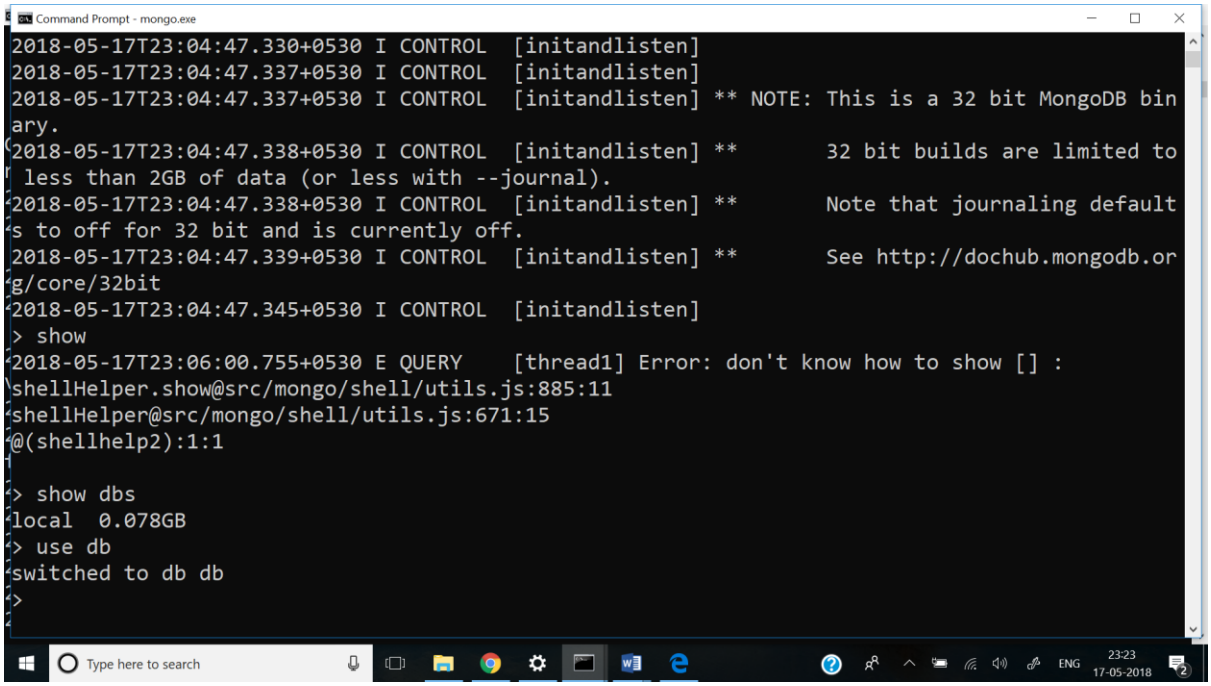
C:\Program Files (x86)\MongoDB\Server\3.2\bin\mongod.exe --dbpath "C:\mongodb\data" --storageEngine=mmapv1



3. Leave this command prompt running and open another command prompt and type following

command to connect to the server:

"C:\Program Files\MongoDB\Server\3.6\bin\mongo.exe"



> show dbs

local 0.078GB

> use db

switched to db db

> db.school.insert({"Name":"Naveen","Age":34,"E-Mail":"gupta.naveen1@gmail.com"})

WriteResult({ "nInserted" : 1 })

> db.school.find()

{ "_id" : ObjectId("5afbc6de87304fb7220142f7"), "Name" : "Naveen", "Age" : 34, "E-Mail" : "gupta.naveen1@gmail.com" }

An ObjectId is a 12-byte BSON type having the following structure:

- The first 4 bytes representing the seconds since the unix epoch
- The next 3 bytes are the machine identifier
- The next 2 bytes consists of process id
- The last 3 bytes are a random counter value.

MongoDB uses ObjectIds as the default value of _id field of each document, which is generated while the creation of any document. The complex combination of ObjectId makes all the _id fields unique.

//To Display name of the Collections

```
> show collections
school
system.indexes
>
> db.school.find().pretty()
{
        "_id" : ObjectId("5afbc6de87304fb7220142f7"),
        "Name" : "Naveen",
        "Age" : 34,
        "E-Mail" : "gupta.naveen1@gmail.com"
}
```

Ctrl+c to come out from mongoDB

Multiple Record Added

```
> db.school.insert([{"Name":"Naveen","Age":34,"E-Mail":"gupta.naveen1@gmail.com"},{"name":"nikunj","age":18,"E-mail":"raju@yahoo.com","Marks":40}])
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
> db.school.find().pretty()
```

```
{
	"_id" : ObjectId("5afbc6de87304fb7220142f7"),
	"Name" : "Naveen",
	"Age" : 34,
	"E-Mail" : "gupta.naveen1@gmail.com"
}
{
	"_id" : ObjectId("5afbc9505193d5efd84958ed"),
	"Name" : "Naveen",
	"Age" : 34,
	"E-Mail" : "gupta.naveen1@gmail.com"
}
{
	"_id" : ObjectId("5afbc9505193d5efd84958ee"),
	"name" : "nikunj",
	"age" : 18,
	"E-mail" : "raju@yahoo.com",
	"Marks" : 40
}
//how to drop a collection
>db.student.drop()

> db.createCollection("student")
{ "ok" : 1 }

> show collections
school
student
system.indexes
```

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

RDMS where clauses Equivalents in MongoDB:

Operations	Syntax	Example	RDBMS Equivalent
Equality	{Key:value}	db.student.find({"age":18}).pretty()	Where age=18
Less Than	{Key:{$lt:value}	db.student.find({"age":{$lt:18}}).pretty()	Where age<18
Less than Equals	{Key:{$lte:value}	db.student.find({"age":{$lte:18}}).pretty()	Where age<=18
Greater Than	{Key:{$gt:value}	db.student.find({"age":{$gt:18}}).pretty()	Where age>8
Greater Than Equals	{Key:{$gtt:value}	db.student.find({"age":{$gte:18}}).pretty()	Where age>=18
Not Equals	{Key:{$lne:value}	db.student.find({"age":{$ne:18}}).pretty()	Where age!=18

> db.school.find({"age":{$lte:34}}).pretty()

{

"_id" : ObjectId("5afe0f538e3142be05c15fc5"),

"name" : "nikunj",

"age" : 18,

"E-mail" : "raju@yahoo.com",

"Marks" : 40

}

OR Example

>db.mycol.find({$or:[{"age":"18"},{"marks": {$lte:50}]}).pretty()

And Example

>db.mycol.find({"age":"18","marks": "$gte:100"}).pretty()

MongoDB Update() Method

The update() method updates the values in the existing document.

> db.school.find().pretty()

{

"_id" : ObjectId("5afe0ee48e3142be05c15fc3"),

"Name" : "Naveen",

"Age" : 34,

"E-Mail" : "gupta.naveen1@gmail.com"

}

{

"_id" : ObjectId("5afe0f538e3142be05c15fc4"),

"Name" : "Naveen",

"Age" : 34,

"E-Mail" : "gupta.naveen1@gmail.com"

}

{

```
        "_id" : ObjectId("5afe0f538e3142be05c15fc5"),
        "name" : "nikunj",
        "age" : 18,
        "E-mail" : "raju@yahoo.com",
        "Marks" : 40
}
```

After Update query

```
> db.school.update({"name":"nikunj"},{$set:{"Name":"Rajiv"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

After this new field is being added because Name not exist as Field

```
> db.school.find().pretty()
{
        "_id" : ObjectId("5afe0ee48e3142be05c15fc3"),
        "Name" : "Naveen",
        "Age" : 34,
        "E-Mail" : "gupta.naveen1@gmail.com"
}
{
        "_id" : ObjectId("5afe0f538e3142be05c15fc4"),
        "Name" : "Naveen",
        "Age" : 34,
        "E-Mail" : "gupta.naveen1@gmail.com"
}
{
        "_id" : ObjectId("5afe0f538e3142be05c15fc5"),
        "name" : "nikunj",
        "age" : 18,
        "E-mail" : "raju@yahoo.com",
        "Marks" : 40,
```

```
        "Name" : "Rajiv"
}
Now data is changed
> db.school.update({"name":"nikunj"},{$set:{"name":"Rajiv"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.school.find().pretty()
{
        "_id" : ObjectId("5afe0ee48e3142be05c15fc3"),
        "Name" : "Naveen",
        "Age" : 34,
        "E-Mail" : "gupta.naveen1@gmail.com"
}
{
        "_id" : ObjectId("5afe0f538e3142be05c15fc4"),
        "Name" : "Naveen",
        "Age" : 34,
        "E-Mail" : "gupta.naveen1@gmail.com"
}
{
        "_id" : ObjectId("5afe0f538e3142be05c15fc5"),
        "name" : "Rajiv",
        "age" : 18,
        "E-mail" : "raju@yahoo.com",
        "Marks" : 40,
        "Name" : "Rajiv"
}
```

The remove() Method MongoDB's remove() method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

- deletion criteria: (Optional) deletion criteria according to documents will be removed.
- justOne: (Optional) if set to true or 1, then remove only one document.

```
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)


> db.school.remove({Name:"Rajiv"})
WriteResult({ "nRemoved" : 1 })
> db.school.find().pretty()
{
	"_id" : ObjectId("5afe0ee48e3142be05c15fc3"),
	"Name" : "Naveen",
	"Age" : 34,
	"E-Mail" : "gupta.naveen1@gmail.com"
}
{
	"_id" : ObjectId("5afe0f538e3142be05c15fc4"),
	"Name" : "Naveen",
	"Age" : 34,
	"E-Mail" : "gupta.naveen1@gmail.com"
}
```