JSON-->10KB
BSON--->4 to 5 KB

Note: Efficient Storage and Extra data types are speciality of BSON over JSON.

While retrieving also, BSON will be converted to JSON by MongoDB server??

EJSON--->Extended JSON
At the time of retrieval BSON data will be converted to EJSON for understanding purpose.

Insertion of Document/Creation --->JSON to BSON
Read Operation/Retrieval Operation -->BSON to EJSON

Q. What data formats used in MongoDB?
3 formats: JSON,BSON,EJSON

MONGODB HANDS-0N

Creation of Database and Collection:

Database won't be created at the beginning and it will be created dynamically.
Whenever we are creating collection or inserting document then

database will be created dynamically.

```
> show dbs
```

```
admin   0.000GB
config   0.000GB
local  0.000GB
> use  durgadb
switched to db durgadb
> show dbs
admin   0.000GB
config 0.000GB
local   0.000GB
```

How to create collection:

```
-----------------------
db.createCollection("employees")


> show dbs
admin   0.000GB
config 0.000GB
local   0.000GB
> db.createCollection("employees")
{ "ok" : 1 }
> show  dbs
admin    0.000GB
config   0.000GB
durgadb 0.000GB
local    0.000GB
> show collections
employees
```

Q. How to drop collection?

db.collection.drop()

```
db.students.drop()

> show collections
employees
students
> db.students.drop()
true
> show collections
employees
```

so it is mandatory to be in the required DB to drop the collection

Q. How to drop database?

db.dropDatabase()

current database will be deleted.

```
> show  dbs
admin    0.000GB
config   0.000GB
durgadb 0.000GB
local    0.000GB
> db.dropDatabase()
{ "dropped" : "durgadb", "ok" : 1 }
> show dbs
admin   0.000GB
config  0.000GB
```

local 0.000GB

Sir, from anywhere like from another database can we drop other database??

Note: db.getName() to know current database.

Basic CRUD Operations in simple way:

----

1. C--->Create|Insert document

How to insert document into the collection

db.collection.insertOne()
db.collection.insertMany()
db.collection.insert()

db.employees.insertOne({eno:100,ename:"Sunny",esal:1000,eaddr:"Hyd"})

2. R--->Read / Retrieval Operation:

----

db.collection.find() --->To get all documents present in the given collection
db.collection.findOne() --->To get one document

eg: db.employees.find()

> db.employees.find()
{ "_id" : ObjectId("5fe16d547789dad6d1278927"), "eno" : 100, "ename" : "Sunny", "esal" : 1000, "eaddr" : "Hyd" }

```
{ "_id" : ObjectId("5fe16da07789dad6d1278928"), "eno" : 200, "ename"
: "Bunny", "esal" : 2000, "eaddr" : "Mumbai" }
{ "_id" : ObjectId("5fe16dc67789dad6d1278929"), "eno" : 300, "ename"
: "Chinny", "esal" : 3000, "eaddr" : "Chennai" }
{ "_id" : ObjectId("5fe16ddb7789dad6d127892a"), "eno" : 400, "ename"
: "Vinny", "esal" : 4000, "eaddr" : "Delhi" }

> db.employees.find().pretty()
{
        "_id" : ObjectId("5fe16d547789dad6d1278927"),
        "eno" : 100,
        "ename" : "Sunny",
        "esal" : 1000,
        "eaddr" : "Hyd"
}
{
        "_id" : ObjectId("5fe16da07789dad6d1278928"),
        "eno" : 200,
        "ename" : "Bunny",
        "esal" : 2000,
        "eaddr" : "Mumbai"
}
{
        "_id" : ObjectId("5fe16dc67789dad6d1278929"),
        "eno" : 300,
        "ename" : "Chinny",
        "esal" : 3000,
        "eaddr" : "Chennai"
}
{
        "_id" : ObjectId("5fe16ddb7789dad6d127892a"),
        "eno" : 400,
        "ename" : "Vinny",
```

```
        "esal" : 4000,
        "eaddr" : "Delhi"
}
```

3. U-->Update Operation:

---

db.collection.updateOne()
db.collection.updateMany()
db.collection.replaceOne()

Update Vinny salary as 10000
db.collection.updateOne()
db.employees.updateOne({ename: "Vinny"},{esal:10000})
if esal field is available then old value will be replaced with 10000.
If the field is not already available then it will be created.

```
> db.employees.updateOne({ename: "Vinny"},{esal:10000})
uncaught exception: Error: the update operation document must
contain atomic operators :
DBCollection.prototype.updateOne@src/mongo/shell/crud_api.js:565:19
@(shell):1:1
```

```
db.employees.updateOne({ename: "Vinny"},{$set: {esal:10000}})
```

```
> db.employees.updateOne({ename: "Vinny"},{$set: {esal:10000}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

> db.employees.updateOne({ename: "Vinny"},{$set: {esal:10000}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.employees.find().pretty()
```

```
{
        "_id" : ObjectId("5fe16d547789dad6d1278927"),
        "eno" : 100,
        "ename" : "Sunny",
        "esal" : 1000,
        "eaddr" : "Hyd"
}
{
        "_id" : ObjectId("5fe16da07789dad6d1278928"),
        "eno" : 200,
        "ename" : "Bunny",
        "esal" : 2000,
        "eaddr" : "Mumbai"
}
{
        "_id" : ObjectId("5fe16dc67789dad6d1278929"),
        "eno" : 300,
        "ename" : "Chinny",
        "esal" : 3000,
        "eaddr" : "Chennai"
}
{
        "_id" : ObjectId("5fe16ddb7789dad6d127892a"),
        "eno" : 400,
        "ename" : "Vinny",
        "esal" : 10000,
        "eaddr" : "Delhi"
}
```

Note: If anything prefixed with $ symbol, then it is predefined word in MongoDB.

4. D -->Delete:

---

db.collection.deleteOne()
db.collection.deleteMany()

db.employees.deleteOne({ename:"Vinny"})

Note: database and collection will be created dynamically whenever we are trying to insert documents.

```
> use  studentdb
switched to db studentdb
> db.students.insertOne({name:"Durga",rollno:101,marks:98})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5fe172617789dad6d127892b")
}
> show dbs
admin      0.000GB
config     0.000GB
durgadb    0.000GB
local      0.000GB
studentdb  0.000GB
> show collections
students
> db.students.find().pretty()
{
        "_id" : ObjectId("5fe172617789dad6d127892b"),
        "name" : "Durga",
        "rollno" : 101,
        "marks" : 98
}
```

CRUD

Capped Collections:

```
> use durgadb
> db.createCollection("employees")
```

```
db.createCollection(name)
db.createCollection(name,options)
   capped
   max 1000 documents--->1001 document
   size: 3736578 bytes only-->if space completed

db.createCollection("employees",{capped: true, size: 3736578, max:
1000})
  old documents will be deleted automatically.

on what basis old document will be deleted?
   based on timestamp
   oldest document will be deleted automatically.

If capped is true means that if size exceeds or maximum number of
documents reached, then oldest entry will be deleted automatically.
```

```
1. db.createCollection("employees") --->Normal Collection
2. db.createCollection("employees",{capped: true})--->Invalid
    "errmsg" : "the 'size' field is required when 'capped' is true",
```

3. db.createCollection("employees",{capped: true, size: 365675})--->valid

4. db.createCollection("employees",{size: 365675})--->invalid

"errmsg" : "the 'capped' field needs to be true when either the 'size' or 'max' fields are present"

5. db.createCollection("employees",{capped: true, size: 365675, max: 1000})--->valid

6. db.createCollection("employees",{capped: true, size: 365675, max: 1})

use case:

----

freshers jobs portal--->students registered

students collection---> 1 lakh

jobs collection---> jobs 100 jobs

Q. What is capped collection?

If size exceeds or maximum number of documents reached, then oldest entry will be deleted automatically, such type of collections are called capped collections.

CRUD

Inserting Documents in the collection(C--->Create):

----

db.collection.insertOne()

db.collection.insertMany()

db.collection.insert()

db.collection.insertOne():

To insert only one document.

db.employees.insertOne({...})

Argument is only one javascript object.

db.employees.insertOne({eno: 100, ename: "Sunny", esal: 1000, eaddr: "Mumbai"})

db.collection.insertMany():

To insert multiple documents

db.collection.insertMany([{..},{..},{..},{..}])

db.employees.insertMany([{eno: 200, ename: "Sunny", esal: 1000, eaddr: "Mumbai"},{eno: 300, ename: "Sunny", esal: 1000, eaddr: "Mumbai"}])

db.collection.insert():

To insert either a single document or multiple documents.

db.employees.insert({...})

db.employees.insert([{..},{..},{..},{..}])

db.employees.insert({eno: 700, ename: "Sunny", esal: 1000, eaddr: "Mumbai"})

db.employees.insert([{eno: 800, ename: "Sunny", esal: 1000, eaddr: "Mumbai"},{eno: 900, ename: "Sunny", esal: 1000, eaddr: "Mumbai"}])

Creating Document separately and inserting into collection:

```
var emp = {};
emp.eno = 7777;
emp.ename = "Bunny";
emp.esal = 777777;
emp.eaddr = "Hyderabad";
```

```
db.employees.insertOne(emp)
db.employees.insertMany([emp])
db.employees.insert(emp)
db.employees.insert([emp])
```

Will it make duplicate documents if we add same document multiple times in a collection?

_id

```
db.employees.insert({_id: "AAA-BBB", name: "Durga"})
db.employees.insertOne({_id: "AAA-BBB", name: "Ravi"})
```

What is capped collection?

insertOne() vs insertMany() vs insert()
How to create document before insertion

bit.ly/durgamongodb

Inserting Documents from java script file:

---

studentsdb --->database
students--->collection
in this collection we have to insert documents.

students.js:

```
db.students.insertOne({name: "Durga", rollno: 101, marks: 98 })
db.students.insertOne({name: "Ravi", rollno: 102, marks: 99 })
db.students.insertOne({name: "Shiva", rollno: 103, marks: 100 })
db.students.insertOne({name: "Pavan", rollno: 104, marks: 80 })
```

```
load("D:\students.js")
```

```
> show collections
> load("D:\students.js")
true
> show collections
students
> db.students.find().pretty()
{
        "_id" : ObjectId("5fe40341941e89a2bcd9f34b"),
        "name" : "Durga",
```

```
        "rollno" : 101,
        "marks" : 98
}
{
        "_id" : ObjectId("5fe40341941e89a2bcd9f34c"),
        "name" : "Ravi",
        "rollno" : 102,
        "marks" : 99
}
{
        "_id" : ObjectId("5fe40341941e89a2bcd9f34d"),
        "name" : "Shiva",
        "rollno" : 103,
        "marks" : 100
}
{
        "_id" : ObjectId("5fe40341941e89a2bcd9f34e"),
        "name" : "Pavan",
        "rollno" : 104,
        "marks" : 80
}
```

Q. Does the name of the javascript file has to be the same as collection name??
Not required.

MONGOIMPORT TOOL :

Inserting Documents from json file (mongoimport tool):

In json file, the data should be in array form.
Make sure the data should be of json only.

json vs javascript object:

In javascript object, quote symbols for keys are optional.
But in JSON, quote symbols are mandatory for keys.

db.collection.insertOne(javascript object)
Here quote symbols are optional

students.json:

```
[
  {
    "name": "Sunny",
    "rollno": 666
  },
  {
    "name": "Bunny",
    "rollno": 777
  },
  {
    "name": "Chinny",
    "rollno": 888
  },
  {
    "name": "Vinny",
    "rollno": 999
  },
  {
    "name": "Pinny",
    "rollno": 555
  }
]
```

mongod --->tool to start MongoDB Server
mongo --->tool to start MongoDB Shell

mongoimport: **--->tool to import documents from json file into MongoDB**

mongoimport: **is not available bydefault. We have to make it availablemanually.**

https://www.mongodb.com/try/download/database-tools

copy mongoimport.exe to the MongoDB bin folder

C:\Program Files\MongoDB\Server\4.4\bin

****

Note: mongoimport command should be executed from the command prompt but not from the shell.

Insert all documents from json file into MongoDB
database name: rahuldb
collection name: students9

from the command prompt, go to location where json file is available.

mongoimport --db databaseName --collection collectionName --file fileName --jsonArray

mongoimport --db rahuldb --collection students9 --file students.json --jsonArray

D:\>mongoimport --db rahuldb --collection students9 --file students.json --jsonArray
2020-12-24T08:57:34.007+0530 connected to: mongodb://localhost/

2020-12-24T08:57:34.191+0530 5 document(s) imported successfully.
0 document(s) failed to import.

mongoimport creates database and collection automatically if not available.

If collection already available then the new documents will be appended.

sir server must be in running naa while we are using mongodbimport?
Server must be in running state.

```
> show dbs
admin        0.000GB
config       0.000GB
dstudentsdb  0.000GB
durgadb      0.000GB
durgadb1     0.000GB
durgadb2     0.000GB
local        0.000GB
rahuldb      0.000GB
storedb      0.000GB
studentdb    0.000GB
> use  rahuldb
switched to db rahuldb
> show collections
students9
> db.students9.find().pretty()
{
        "_id" : ObjectId("5fe40aa643e59978520a102b"),
        "name" : "Vinny",
        "rollno" : 999
}
{
        "_id" : ObjectId("5fe40aa643e59978520a102c"),
        "name" : "Bunny",
        "rollno" : 777
```

```
}
{
        "_id" : ObjectId("5fe40aa643e59978520a102d"),
        "name" : "Chinny",
        "rollno" : 888
}
{
        "_id" : ObjectId("5fe40aa643e59978520a102e"),
        "name" : "Pinny",
        "rollno" : 555
}
{
        "_id" : ObjectId("5fe40aa643e59978520a102f"),
        "name" : "Sunny",
        "rollno" : 666
}
```

students.json:

```
----------
[
  {
    "name": "Dhoni",
    "rollno": 7777777
  }
]
```

mongoimport --db rahuldb --collection students9 --file students.json --jsonArray

This item it will perform append operation.

Note: Similarly, we can use mongoimport to import data from excel/csv files.

can i import from anyother RDBMS like oracle? Yes

How to insert documents from js file

How to insert documents from json file

How to insert documents from csv file| excel file

Nested Documents:(EMBEDDED DOCUMENTS)

Sometimes we can take a document inside another document, such type of documents are called nested documents or embedded documents.

```
employees:
{
   eno:100,
   ename:"durga",
   esal:1000,
   eaddr:"Hyderabad",
   hobbies: {h1:"Swimming",h2:"Reading"}

}
```

storedb-->database name

books--->collection

books.json

```
[
  {
    "title": "Python In Simple Way",
    "isbn": 12345,
    "downloadable": true,
    "no_of_reviews": 10,
    "author": {
                "name": "Daniel Kohen",
                 "callname": "Dan"
              }
  },
  {
    "title": "MongoDB In Simple Way",
    "isbn": 45678,
    "downloadable": false,
    "no_of_reviews": 5,
    "author": {
                "name": "Shiva Ramachandran",
                 "callname": "Shiva"
              }
  }
]
```

mongoimport --db storedb --collection books --file books.json --jsonArray

[

```
  {
    "title": "Linux In Simple Way",
    "isbn": 778899,
    "downloadable": true,
    "no_of_reviews":0,
    "author": {
                "name": "Shiva Ramachandran",
                 "callname": "Shiva",
                 "profile": {
                              "exp":8,
                               "courses":3,
                               "books":2
                            }
              }
  }
]
```

Note:

Inside Nested document, we can take another document also.

MongoDB supports upto 100 levels of nesting.

1. command name and purpose

db.collection.find().pretty()

db.collection.insertOne()

db.collection.insertMany()

db.collection.insert()

db.collection.updateOne({},{})

Arrays in Documents:

Any collection of items is called an array.
The items can be strings or numbers or objects.

A document can contain arrays also.

books.json:

```
[

  {
    "title": "Devops In Simple Way",
    "isbn": 112233,
    "downloadable": false,
    "no_of_reviews":20,
    "tags":["jenkins","git","CICD"],
    "languages":["english","hindi","telugu"],
    "author": {
                "name": "Martin Kohenova",
                 "callname": "Mart",
                 "profile": {
                              "exp":8,
                               "courses":3,
                               "books":2
                            }
              }
  }
]
```

mongoimport --db storedb --collection books --file books.json --jsonArray

s we use find() to fetch all documents but how to fetch a particularfield in the document ?

Basic idea about CRUD Operations.
C--->Create Operation|Insert Operation

ObjectId:

For every document, MongoDB Server will associate a unique id, which is nothing but ObjectId.

It is something like primary key in relational databases.
The ObjectId will be assigned to _id field.

"_id" : ObjectId("5fe6ad34b195d71b16a713c8")

ObjectId is not json type and it is of BSON type.

ObjectId is of 12 bytes.

1. The first 4 bytes represents the timestamp when this document was inserted.
2. The next 3 bytes represents machine identifier(host name)
3. The next 2 bytes represents process id.
4. The last 3 bytes represents some random increment value.

Why this lengthy ObjectId:

The only reason is uniqueness.

mobile number contains 10 digits
why 10 digits, just only one digit is enough???

To generate timestamp from ObjectId:

---

db.employees.find() --->List out all documents of employees collection
db.employees.find()[0] --->List out only first document of employees collection
db.employees.find()[0]._id --->ObjectId of first document
db.employees.find()[0]._id.getTimestamp() --->ObjectId of first document

```
> db.employees.find()[0]
{
        "_id" : ObjectId("5fe2b6fc9d0c84a052cb9745"),
        "eno" : 100,
        "ename" : "Sunny",
        "esal" : 1000,
        "eaddr" : "Mumbai"
}
> db.employees.find()[0]._id
ObjectId("5fe2b6fc9d0c84a052cb9745")
> db.employees.find()[0]._id.getTimestamp()
ISODate("2020-12-23T03:18:20Z")


{ "_id" : ObjectId("5fe6b3218c25aae60be989c0"), "A" : 100, "B" : 200 }
> db.employees.find({"B":200})
{ "_id" : ObjectId("5fe6b3218c25aae60be989c0"), "A" : 100, "B" : 200 }
> db.employees.find({"B":200})._id
> db.employees.find({"B":200})[0]
{ "_id" : ObjectId("5fe6b3218c25aae60be989c0"), "A" : 100, "B" : 200 }
> db.employees.find({"B":200})[0]._id
```

ObjectId("5fe6b3218c25aae60be989c0")
> db.employees.find({"B":200})[0]._id.getTimestamp()
ISODate("2020-12-26T03:50:57Z")

By using _id field, we can provide our own value as ObjectId. MongoDB server will generate default ObjectId iff we are not providing any _id field value.

If we provide our own value, it may not provide timestamp,machine identifier,process id etc. Hence it is not recommended to provide our own id.

Is it possible to have same _id for 2 documents?
Duplicate ObjectIds possible?
No chance at all, even if we provide value explicitly also.

db.employees.insertOne({_id:789, name:"Rahul"})
db.employees.insertOne({_id:789, name:"Viraj"})

"errmsg" : "E11000 duplicate key error collection: durgadb.employees index: _id_ dup key: { _id: 789.0 }",

ObjectIds are Immutable, ie once we creates/assigns ObjectId we cannot change its value.

{ "_id" : 789, "name" : "Rahul" }
db.employees.updateOne({_id: 789},{$set: {_id:9999}})

"errmsg" : "Performing an update on the path '_id' would modify the immutable field '_id'",

Q. We can use the same ObjectId is for other collection right?
Yes possible. Uniqueness is per collection not per database.

Q. _id: 100 and _id: "100" possible in same collction?
Yes because data types are different.

Q. Which of the following are TRUE?
A) We cannot store documents in collection without ObjectId.
B) _id field will be added automatically by MongoDB, if we are not providing that field explicitly.
C) ObjectId is not JSON type and it is of BSON Type.
D) Default ObjectId generated by MongoDB is of 12 bytes.
E) ObjectIds are unique.
F) ObjectIds are Immutable.
G) We cannot modify the value of ObjectId after creation.
H) Default ObjectId consists of timestamp,machine identifier, processid etc
I) The advantage of using default ObjectId is we can get several details like timestamp etc
J) If we provide our own ObjectId value, it may not generate timestamp,machine identifier,process id etc.
K) All of these.

Ans: K

insertOne()
insertMany()
insert()

- **Inserting documents from javascript file by using load() function**

- **Inserting documents from json file by using mongoimport tool**

Nested Documents
Arrays In Documents
ObjectId

Ordered Insertion
WriteConcern
Atomiticity

Ordered Insertion in Bulk inserts:

We can perform bulk inserts either by using insertMany() or insert() methods.

All documents present inside given array will be inserted into collection.

durgadb database

alphabets collection

some documents-->bulk insert

insert([{},{},{},{},{}])

db.alphabets.insertMany([{A:"Apple"},{B:"Banana"},{C:"Cat"}])

[{A:"Apple"},{B:"Banana"},{C:"Cat"}] ===>Array of Javascript objects

```
> db.alphabets.find()
{ "_id" : ObjectId("5fe7f8998a9854ae87538e18"), "A" : "Apple" }
{ "_id" : ObjectId("5fe7f8998a9854ae87538e19"), "B" : "Banana" }
{ "_id" : ObjectId("5fe7f8998a9854ae87538e1a"), "C" : "Cat" }
```

Default Behaviour of bulk inserts:

While performing bulk insert operation, if any document insertion fails then rest of the documents won't be inserted. i.e in bulk inserts, order is important.

The documents which are already inserted won't be rollbacked.

```
db.cars.insertMany([{_id: 100, M:"Maruti"},{_id: 100, A:"Audi"},{_id: 300, B:"Benz"}])

0
1
2


db.cars.find().pretty()


> db.cars.insertMany([{_id: 100, M:"Maruti"},{_id: 100, A:"Audi"},{_id: 300, B:"Benz"}])
uncaught exception: BulkWriteError({
        "writeErrors" : [
                {
                        "index" : 1,
                        "code" : 11000,
                        "errmsg" : "E11000 duplicate key error collection: durgadb.cars index: _id_ dup key: { _id: 100.0 }",
                        "op" : {
                                "_id" : 100,
                                "A" : "Audi"
                        }
                }
        ],
        "writeConcernErrors" : [ ],
        "nInserted" : 1,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
```

```
        "nRemoved" : 0,
        "upserted" : [ ]
}) :
BulkWriteError({
        "writeErrors" : [
                {
                        "index" : 1,
                        "code" : 11000,
                        "errmsg" : "E11000 duplicate key error collection:
durgadb.cars index: _id_ dup key: { _id: 100.0 }",
                        "op" : {
                                "_id" : 100,
                                "A" : "Audi"
                        }
                }
        ],
        "writeConcernErrors" : [ ],
        "nInserted" : 1,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
BulkWriteError@src/mongo/shell/bulk_api.js:367:48
BulkWriteResult/this.toError@src/mongo/shell/bulk_api.js:332:24
Bulk/this.execute@src/mongo/shell/bulk_api.js:1186:23
DBCollection.prototype.insertMany@src/mongo/shell/crud_api.js:326:5
@(shell):1:1

> db.cars.find().pretty()
{ "_id" : 100, "M" : "Maruti" }
```

While performing bulk insert operation, if any document insertion fails then rest of the documents won't be inserted. i.e in bulk inserts, order is important.
The documents which are already inserted won't be rollbacked.

We can customize this behaviour. We can customize in such a way that if one document insertion fails, still the remaining documents can be inserted.
For this we have to use ordered property.

> db.cars.insertMany([{},{},...{}],{ordered: false})
The default value for ordered is true.
if order is false==>still one document insertion fails, rest of the documents will be inserted without any problem.

db.cars.insert([{_id:200,I: "Innova"},{_id:200,R: "Ritz"},{_id:300,G: "Gitz"}],{ordered: false})

How many documents will be inserted in the collection:
Ans: 2

How to rollback already inserted documents in the case of any error in bulk inserts?

Q.

Ans: By using transactions

transaction: Either all or None

Transfer 10k from my account to sunny account
operation-1: debit 10k from my account
operation-2: credit 10k to sunny account

Q. What is the purpose of ordered propert in insert operation?
Q. While performing bulk insert operation by using either insertMany() or insert() method, if one document insertion fails, is rest of the documents will be inserted or not?

By default: No
But we can customize this behaviour by using ordered property.

Q1. Assume cars collection is empty.
db.cars.insert([{_id:200,I: "Innova"},{_id:200,R: "Ritz"},{_id:300,G: "Gitz"}],{ordered: true})
How many records will be inserted in the collection?

Ans: 1

Q2. Assume cars collection is empty.
db.cars.insert([{_id:200,I: "Innova"},{_id:200,R: "Ritz"},{_id:300,G: "Gitz"}])
How many records will be inserted in the collection?

Ans: 1

Q3. Assume cars collection is empty.
db.cars.insert([{_id:200,I: "Innova"},{_id:200,R: "Ritz"},{_id:300,G: "Gitz"}],{ordered: false})
How many records will be inserted in the collection?

Ans: 2

WriteConcern Property:

---

usecase-1:

Whenever we are performing insert operation, bydefault the shell/client will wait until getting acknowledgement. Server will provide acknowledgement after completing insert operation. This may reduce performance at client side.

```
> db.cars.insertOne({I:"Innova"})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5fe800f68a9854ae87538e1b")
}
```

We can customize this behaviour by using writeConcern propery.
db.collection.insertOne({},{writeConcern: {w:0}})
w:1===>It is the default value and client will wait until getting acknowledgment.
w:0===>It means client won't wait for acknowledgement.

```
> db.cars.insertOne({B:"BMW"},{writeConcern: {w: 1} })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5fe802f48a9854ae87538e1c")
}
> db.cars.insertOne({H:"Honda"},{writeConcern: {w: 0} })
{ "acknowledged" : false }
```

Even "acknowledged" : false, still document inserted.

```
> db.cars.find()
{ "_id" : ObjectId("5fe800f68a9854ae87538e1b"), "I" : "Innova" }
{ "_id" : ObjectId("5fe802f48a9854ae87538e1c"), "B" : "BMW" }
{ "_id" : ObjectId("5fe803248a9854ae87538e1d"), "H" : "Honda" }
```

If lakhs of records are required to insert, if one or two document insertion fails still no problem, but performance is important then writeConcern is the best choice.

usecase-2:

In Production , for every database we have to maintain cloned/replica database because

1. To handle Fail over situations
2. For Load Balancing Purposes

A single document is required to insert in multiple database instances like primary database, replica-1,replica-2 etc.

Diagram

After inserting how many instance, you are expecting acknowdgement, we can specify this by using writeConcern propery.

if w: 0 ===> No acknowledgement.
if w: 1 ===> Acknowledgement after inserting document in primary database.

if w: 2 ===> Acknowledgement after inserting document in primary database and replica-1.
if w: 3 ===> Acknowledgement after inserting document in primary database, replica-1 and replica-2.

```
db.cars.insertOne({A:"Audi"},{writeConcern: {w: 3} })
> db.cars.insertOne({A:"Audi"},{writeConcern: {w: 3} })
uncaught exception: WriteCommandError({
        "ok" : 0,
        "errmsg" : "cannot use 'w' > 1 when a host is not replicated",
        "code" : 2,
        "codeName" : "BadValue"
})
```

Note: To use 'w' > 1, replica copies should be available already.

Note: writeConcern is applicable for any write operation like insert,update and delete.

Using mongoimport — db OpenFlights — collection Airport — type csv — headerline — ignoreBlanks — file [local path]

Importing csv /excel file to mongoDB

1. A csv file with following data is getting imported

By using mongoimport --db myDb --collection myCollection --type csv --headerline --file emp.csv

eno,ename,esal,eadd
17325,rkg,100000,lucknow
gkr,17325,1000,lko

12345,yvan,15000,noida

2. An excel file with following data when converted to csv by changing Extension is not being imported by above command.

sl no name Emp number salary years of service desgnation
1 AAAAA 12345 10000 4.00 worker
2 BBBBB 12346 15000 8.00 worker
3 CCCCC 12347 20000 5.00 Manager
4 DDDDD 12348 25000 3.00 SM

The Big story of insert operation(C->Create Operation)
----
1. insertOne(),insertMany(),insert()
2. Insert documents from javascript file by using load()
3. Insert documents from json file by using mongoimport
3. Insert documents from csv file by using mongoimport
4. Nested Documents
5. Arrays in Documents
6. ObjectId
7. Ordered Insertion
8. WriteConcern
9. Atomicity

Bigger Doubt about Atomicity:

Q. Assume we have to insert a document where 100 fields are available, after inserting 50 fields if database server faces some problem then what will be happend?

Ans: Whatever fields already added will be rollbacked.

MongoDB Server stores either complete document or nothing. ie it won't store part of the document. ie CRUD operations are atomic at document level.

db.collection.insertMany([{},{},{},{}])

But while inserting multiple documents (Bulk Insertion), after inserting some documents if database server faces some problem, then already inserted documents won't be rollbacked. i.e atomicity bydefault not applicable for bulk inserts.

If we want atomicity for bulk inserts then we should go for transactions concept.
Transaction: Either All operations or None
eg:
trasfer 10k from my account to sunny account
operation-1: debit 10k from my account
operation-2: credit 10k to sunny account

CRUD Operations-->C

CRUD Operations--->R Operation/Read Operation/Retrieve Operation/Find Operation:

We can read documents from the collection by using the following find methods.

1. find({query}) --->Returns all matched documents based on query.
2. findOne({query}) --->Returns one matched document based on query.

The argument,query is a simple javascript object.