

1.ADD, UPDATE & DELETE

Let load the document:

- Download the student csv from this [link](#).
- Import the data to the collection created [link](#).
- You should be able to see the uploaded data in mongo compass.

Installation of Mongo Shell or Studio 3t:

- Mongo Shell download [link](#).
- All the work is expected to do it in mongo Shell not in mongo compass.

OR

- You can also install [Studio3T](#)
- Connect to mongoddb://localhost:27017

Few Commands to test after connections:

Command	Expected Output	Notes
Show dbs	Admin 40.00 KB Config 72.00 KB db 128.00 KB local 40.00 KB	All Databases are shown
Use db	Switched to db db	Connect and use db
Show collections	students	Show all tables
db.foo.insert({"bar":"baz"})		Insert a record to collection. Create Collection if not Exists
db.foo.batchinsert ([{"_id":0},{"_id":1}, {"_id":2}])		Insert more than one document
db.foo.find()		Print all rows
db.foo.remove()		Remove foo table

Documents, Collections, Database

Documents:

Records in a MongoDB database are called documents, and the field values may include number, strings, Booleans, arrays, or even nested documents.

EX:

```
{  
  Title: "Post Title 1",  
  Body: "Body of post.",  
  Category: "News",  
  Likes: 1,  
  Tags: ["news", "events"],  
  Date: Date()  
}
```

Database	humanResourcedb TutorialsTeacher.com		
Collections	employees	addresses	departments
Documents	{ "firstName": "John", "lastName": "King", "email": "john.king@abc.com", "salary": "33000" }	{ "street": "H12", "house": "33", "city": "New York", "country": "USA" }	{ "name": "Technical", "totalEmployees": "100", }

Mongo Database, Collection, and Document

Collections:

A collection in MongoDB is similar to a table in RDBMS. MongoDB collections do not enforce schemas. Each MongoDB collection can have multiple documents. A document is equivalent to row in a table in RDBMS.

To create a collection, use the `db.createCollection()` command. The following creates a new employees collection in the current database, which is `humanResourceDB` database created in the previous chapter.

EX:

```
humanResourceDB> db.createCollection("departments")
{ Ok: 1 }
humanResourceDB> db.createCollection("addresses")
{ ok: 1 }
```

create Collection

Use the `show collections` commands to list all the collections in a database.

EX:

```
humanResourceDB> show collection
addresses
departments
employees
humanResourceDB>
```

Show Collection

To delete a collection, use the `db.<collection-name>.drop()` method.

EX:

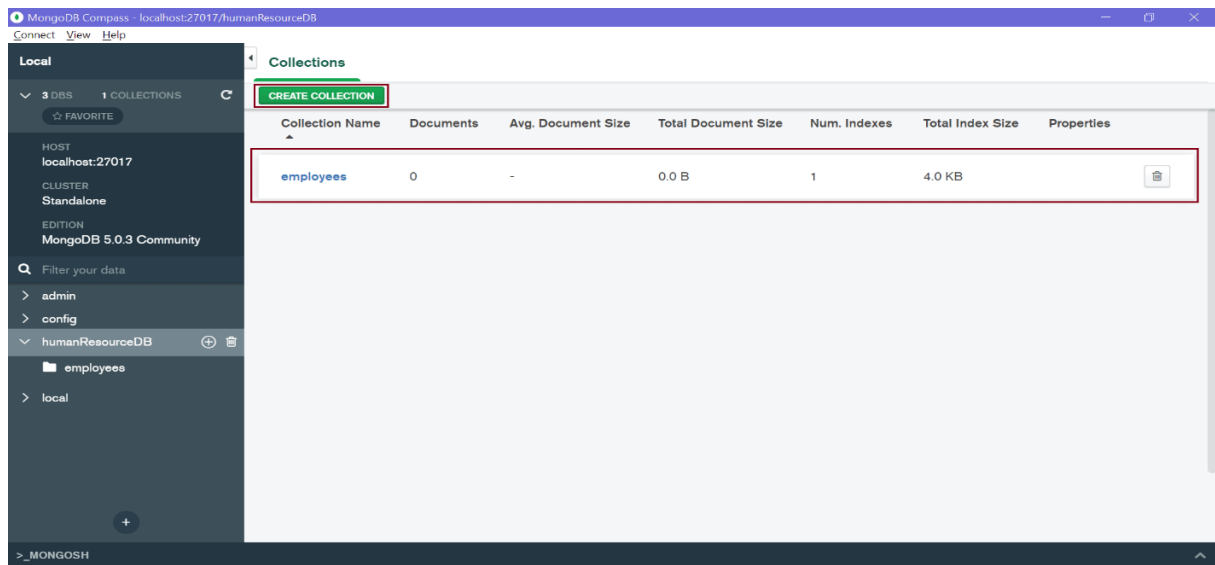
```
humanResourceDB> db.addresses.drop()
true
humanResourceDB> show collections
departments
employees
humanResourceDB>
```

Delete Collection

Create Collection in MongoDB Compass

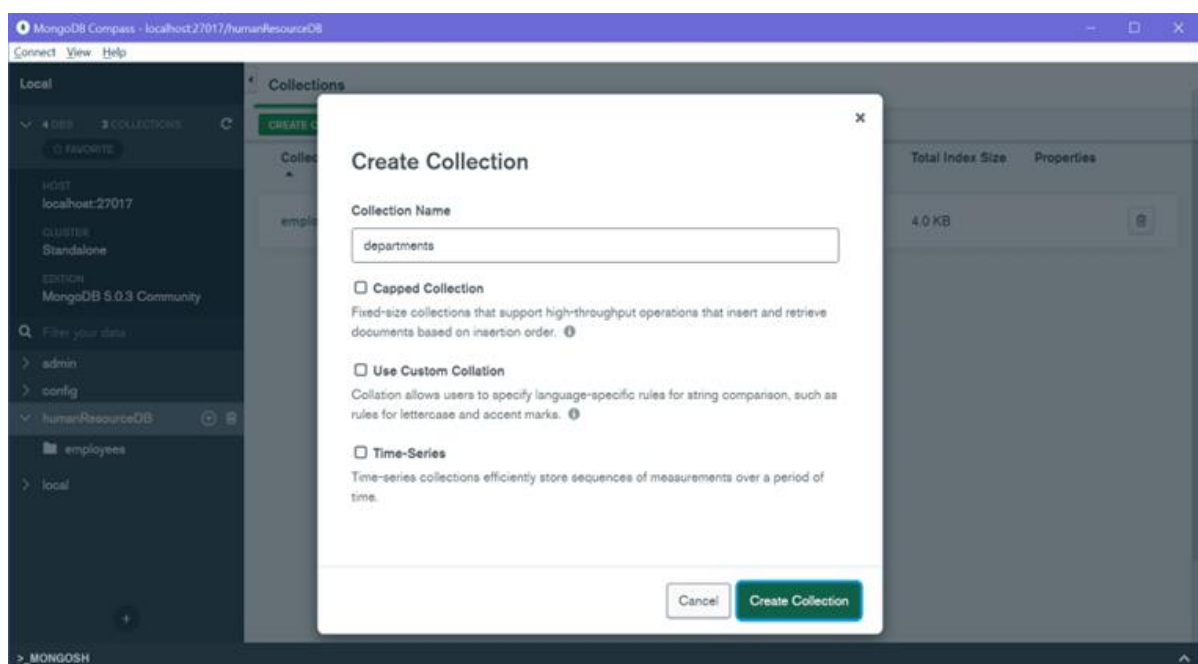
To create a new collection using MongoDB Compass, connect compass to your server and select the database.

Click on the "Create Collection" button to create a new collection, as shown below.



MongoDB-Collections

Enter the name of a collection, check appropriate checkbox and click on the **Create Collection** button to create it. Thus, you can create a new collection using MongoDB Shell mongosh or MongoDB Compass.



Database:

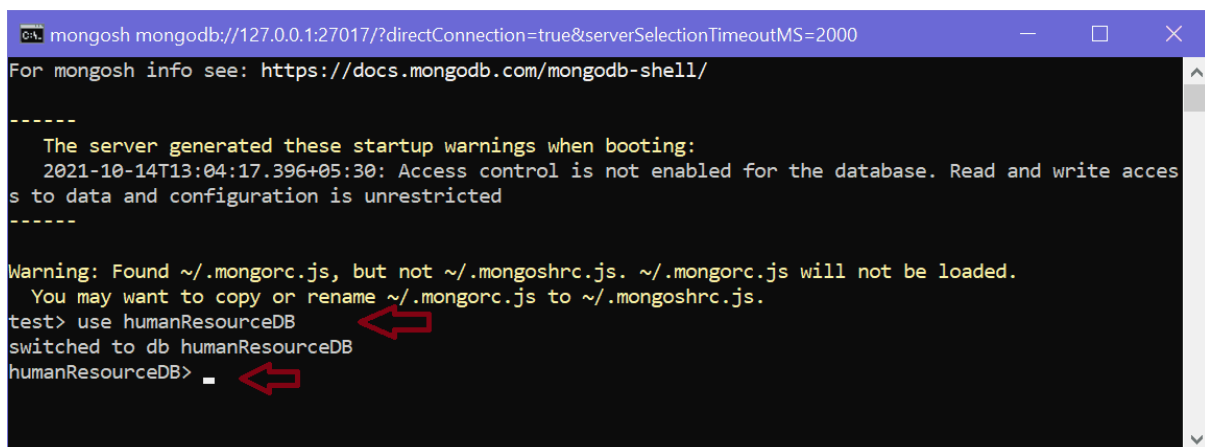
MongoDB provides the `use <database-name>` command to connect with the database. If the specified database name does not exist then it creates it and set it as a current database.

For example, the following command switch to the "humanResourcedb" database. If it does not exist then creates it.

EX: Switch or Create Database

```
use humanResourceDB
```

The following shows how to create or switch MongoDB database in MongoDB shell `mongosh`:



```
C:\> mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
For mongosh info see: https://docs.mongodb.com/mongodb-shell/

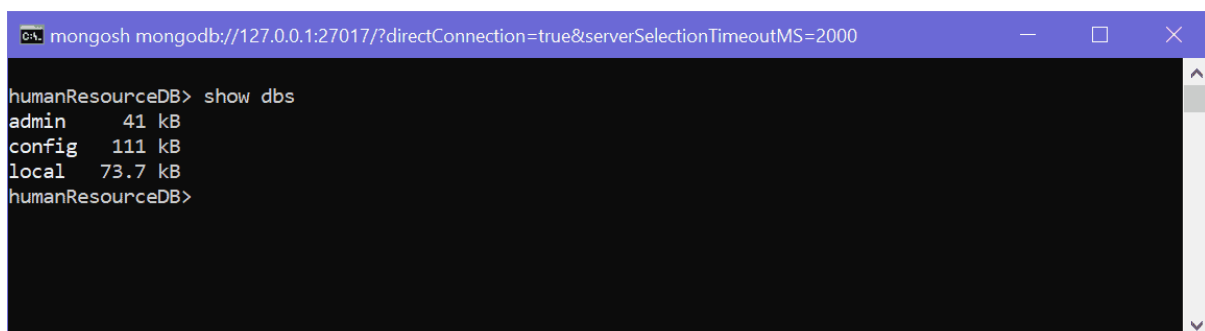
-----
The server generated these startup warnings when booting:
 2021-10-14T13:04:17.396+05:30: Access control is not enabled for the database. Read and write access
s to data and configuration is unrestricted
-----

Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be loaded.
You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.
test> use humanResourceDB
switched to db humanResourceDB
humanResourceDB> 
```

Create or Switch Database in MongoDB Shell

MongoDB will automatically switch to the newly created database. Notice that it prompts to `humanResourceDB>` now.

To check all the databases, use the "show dbs" command, as shown below.

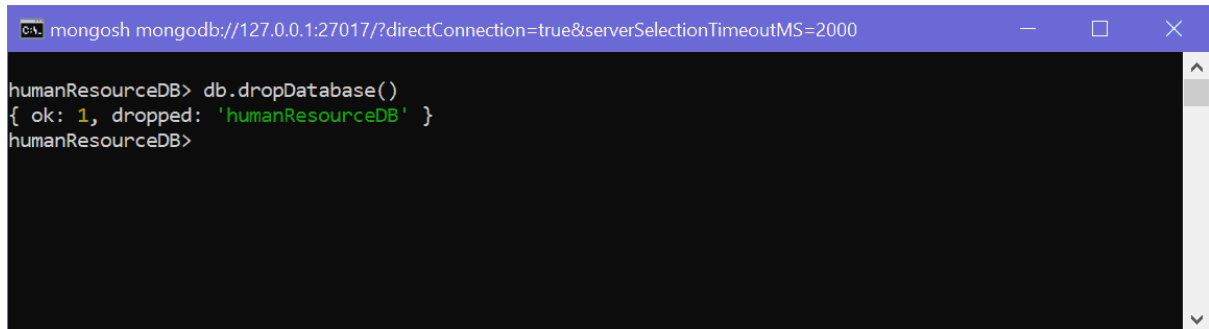


```
C:\> mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
humanResourceDB> show dbs
admin      41 kB
config    111 kB
local     73.7 kB
humanResourceDB> 
```

List Databases

As you can see above, the "admin", "config", and "local" are default databases. As of now, "humanResourceDB" is not visible. This is because there is no collection in it.

To delete a database, use the `db.dropDatabase()` method which deletes a current database.



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
humanResourceDB> db.dropDatabase()
{ ok: 1, dropped: 'humanResourceDB' }
humanResourceDB>
```

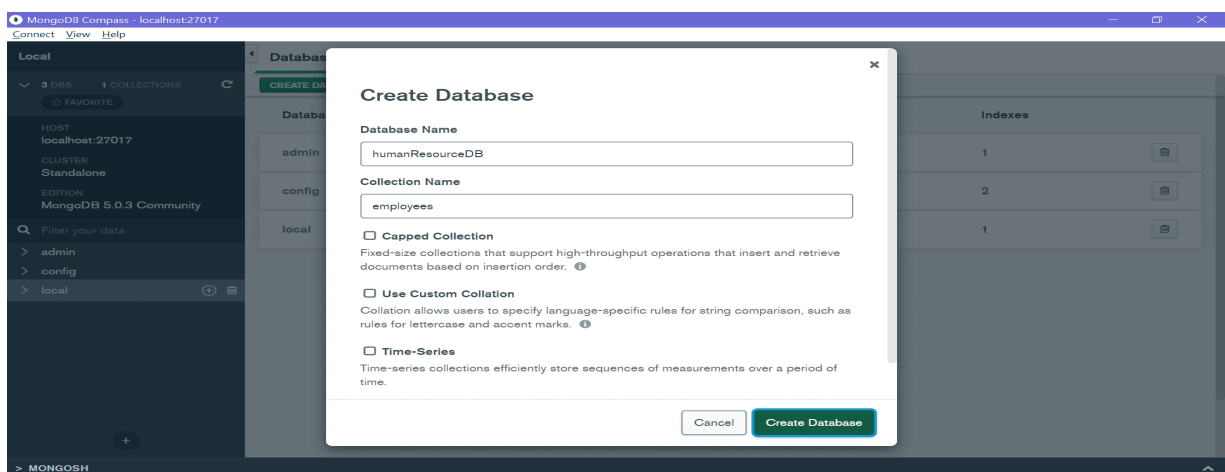
Delete Database

Above, `{ ok: 1, dropped: 'humanResourcedb' }` indicates that the database deleted successfully.

Note: Method names are case sensitive. So, executing `db.dropdatabase()` will throw an error.

Create Database using MongoDB Compass

You can create a new database using MongoDB Compass. For that, open Compass and connect with your local or remote database. Once it connects with the MongoDB server, click on the top "CREATE DATABASE" button which will open the popup window, as shown below.



MongoDB Compass-Create Database

Datatype:

MongoDB has a unique way of representing data types in which each data type is associated with an alias as well as a number that is usually used to search or find any specific record within a MongoDB database. MongoDB allows its users to implement different variations of data types:

Integer:

Integer is a data type that is used for storing a numerical value, i.e., integers as you can save in other programming languages. 32 bit or 64-bit integers are supported, which depends on the server.

Example:

```
db.TestCollection.insert({"Integer example": 62})
```

Output:

```
> use my_project_db
switched to db my_project_db
> db.TestCollection.insert({"Integer example": 62})
WriteResult({ "nInserted" : 1 })
> db.TestCollection.find()
{ "_id" : ObjectId("5d08724bce0d9292a5121a78"), "Integer example" : 62 }
>
```

Boolean:

Boolean is implemented for storing a Boolean (i.e., true or false) values.

Example:

```
db.TestCollection.insert({"Nationality Indian": true})
```

Output:

```
> db.TestCollection.insert({"Nationality Indian": true})
WriteResult({ "nInserted" : 1 })
> db.TestCollection.find()
{ "_id" : ObjectId("5d08724bce0d9292a5121a78"), "Integer example" : 62 }
{ "_id" : ObjectId("5d087412ce0d9292a5121a79"), "Nationality Indian" : true }
>
```

Double:

Double is implemented for storing floating-point data in MongoDB.

Example:

```
db.TestCollection.insert({"double data type": 3.1415})
```

Output:

```
> db.TestCollection.insert({"double data type": 3.1415})
WriteResult({ "nInserted" : 1 })
> db.TestCollection.find()
{ "_id" : ObjectId("5d08724bce0d9292a5121a78"), "Integer example" : 62 }
{ "_id" : ObjectId("5d087412ce0d9292a5121a79"), "Nationality Indian" : true }
{ "_id" : ObjectId("5d0874dcce0d9292a5121a7a"), "double data type" : 3.1415 }
```

Min/Max keys:

Min / Max keys are implemented for comparing a value adjacent to the lowest as well as highest BSON elements.

String:

String is one of the most frequently implemented data type for storing the data.

Example:

```
db.TestCollection.insert({"string data type" : "This is a sample message."})
```

Output:

```
> db.TestCollection.insert({"string data type" : "This is a sample message."})
WriteResult({ "nInserted" : 1 })
> db.TestCollection.find()
{ "_id" : ObjectId("5d08724bce0d9292a5121a78"), "Integer example" : 62 }
{ "_id" : ObjectId("5d087412ce0d9292a5121a79"), "Nationality Indian" : true }
{ "_id" : ObjectId("5d0874dcce0d9292a5121a7a"), "double data type" : 3.1415 }
{ "_id" : ObjectId("5d087547ce0d9292a5121a7b"), "string data type" : "This is sample message." }
>
```

Arrays:

Arrays are implemented for storing arrays or list type or several values under a single key.

Example:

```
var degrees = ["BCA", "BS", "MCA"]  
db.TestCollection.insert({" Array Example" : " Here is an example of array",  
" Qualification" : degrees})
```

Output:

```
> var degrees = ["BCA", "BS", "MCA"]  
> db.TestCollection.insert({" Array Example" : " Here is an example of array",  
" db.TestCollection.insert({" Array Example" : " Here is an example of array",  
" Qualification" : degrees})  
WriteResult({ "nInserted" : 1 })  
> db.TestCollection.find()  
{ "_id" : ObjectId("5d0a6bc0940707e25658e62c"), "Integer example" : 62 }  
{ "_id" : ObjectId("5d0a6c2e28323ade3e6287f9"), "Nationality Indian" : true }  
{ "_id" : ObjectId("5d0a6c3528323ade3e6287fa"), "double data type" : 3.1415 }  
{ "_id" : ObjectId("5d0a6c3d28323ade3e6287fb"), "string data type" : "This is a  
sample message." }  
{ "_id" : ObjectId("5d0a6ee828323ade3e628800"), " Array Example" : " Here is an  
example of array", " Qualification" : [ "BCA", "BS", "MCA" ] }  
>
```

Object:

Object is implemented for embedded documents.

Example:

```
var embeddedObject={"English" : 94, "ComputerSc." : 96, "Maths" : 80,  
"GeneralSc." : 85}  
db.TestCollection.insert({"Object data type" : "This is Object",  
"Marks" : embeddedObject})
```

Output:

```

> var embeddedObject = {"English" : 94, "ComputerSc." : 96, "Maths" : 80, "GeneralSc." : 85}
> db.TestCollection.insert({"Object data type" : "This is Object", "Marks" : embeddedObject})
WriteResult({ "nInserted" : 1 })
> db.TestCollection.find()
{ "_id" : ObjectId("5d0a6bc0940707e25658e62c"), "Integer example" : 62 }
{ "_id" : ObjectId("5d0a6c2e28323ade3e6287f9"), "Nationality Indian" : true }
{ "_id" : ObjectId("5d0a6c3528323ade3e6287fa"), "double data type" : 3.1415 }
{ "_id" : ObjectId("5d0a6c3d28323ade3e6287fb"), "string data type" : "This is a sample message." }
{ "_id" : ObjectId("5d0a6ee828323ade3e628800"), " Array Example" : " Here is an example of array", " Qualification" : [ "BCA", "BS", "MCA" ] }
{ "_id" : ObjectId("5d0a715828323ade3e628801"), "Object data type" : "This is Object", "Marks" : { "English" : 94, "ComputerSc." : 96, "Maths" : 80, "GeneralSc." : 85 } }
>

```

Symbol:

Symbol is implemented to a string and is usually kept reticent for languages having specific symbol type.

Null:

Null is implemented for storing a Null value.

Example:

```
db.TestCollection.insert({" EmailID " : null})
```

Output:

```

> db.TestCollection.insert({" EmailID " : null})
WriteResult({ "nInserted" : 1 })
> db.TestCollection.find()
{ "_id" : ObjectId("5d0a731828323ade3e628803"), " EmailID " : null }
>

```

Date:

Date is implemented for storing the current date and time as UNIX-time format.

Example:

```
var date=new Date()
```

```
var date2=ISODate  
  
var month=date2.getMonth()  
  
db.TestCollection.insert({"Date":date, "Date2":date2, "Month":month})
```

Output:

```
> var date=new Date()  
> var date2=ISODate()  
> var month=date2.getMonth()  
> db.TestCollection.insert({"Date":date, "Date2":date2, "Month":month})  
WriteResult({ "nInserted" : 1 })  
> db.TestCollection.find()  
{ "_id" : ObjectId("5d0a73fa28323ade3e628804"), "Date" : ISODate("2019-06-19T17:41:34.598Z"), "Date2" : ISODate("2019-06-19T17:41:45.071Z"), "Month" : 5 }  
>
```

Timestamp:

Timestamp stores 64-bit value, in which the first 32 bits are time_t value (seconds epoch) and the other 32 bits are ordinal to operate within a given second.

Binary data:

Binary data is implemented for storing binary data.

Object ID:

Object ID is implemented for storing the ID of the document.

Regular expression:

Regular expression is implemented for storing regular expression.

Code:

Code is implemented for storing JavaScript code for your MongoDB document.

In the next chapters, all data types and uses have been explained more deeply.