MongoDB is an open-source document NoSQL database. It provides high performance, high availability, and easy scalability. MongoDB works on the concept of document, collection and databases. A **document** is a set of key-value pairs. Documents have a dynamic schema (i.e., 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). A **collection** is a group of MongoDB documents. It is the equivalent of tables in relational databases. Typically, all documents in a collection are of similar or related purpose. A **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. Please refer to Week 8 recorded lesson for further details. Table 1 shows the conceptual mapping of RDBMS terminology with Document databases/MongoDB. MongoDB supports dynamic queries on documents using a **document-based query language** that's nearly as powerful as SQL.

RDBMS	Document Databases (MongoDB)
Database	Database
Table	Collection
Row	Document
Column	Field/Attribute/Key-Value (note: case-sensitive Refer to 6A query document)
Table join	Embedded Documents
Primary Key	Primary Key (Default key ***_id***[1] provided by MongoDB itself)

Table 1: Conceptual mapping of terminologies - RDBMS and DocumentDB (Ack: Tutorialspoint.com).

1. Configuration (one time procedure)

- Login through Exeter Virtual Device (WVD)
- Locate S:\ Drive
- Copy the following two batch files (***mongod_use this.bat*** and ***mongo_use this.bat***) from the BEMM459 Github repository (under Week 8) to S:\ drive[2].
- First, create a folder "MongoDB" in S:\ drive. Next, under "S:\MongoDB" create a folder called "Data" (this is the folder that will store the database files). See Figure 1.
- Start MongoDB server (*mongod_use this.bat*)
- Start MongoDB client (*redis-server_use this.bat*)
- Enter the command db.stats() in mongo client (mongo.exe) to test connectivity with the server (mongod.exe). The output of the command will show the database name, number of collections and documents in the database.
- Enter the command db.help() in mongo client to display a list of commands.

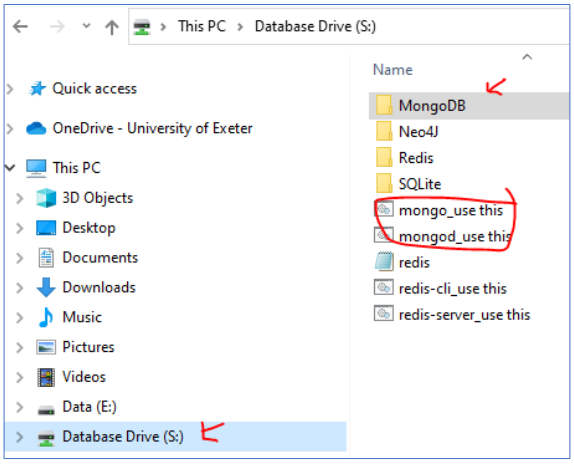


Figure 1: Screenshot of MongoDB configuration over Exeter WVD.

[1] ***_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.

[2] *Note: MongoDB NoSQL database software is installed in E:\ drive (this is non-persistent). Only S:\ drive (and Windows One Drive) is persistent storage and you must save all files in S:\ drive. Otherwise, you will lose your work between WVD logins.*

2. Database commands

MongoDB commands are shown in brown font. Examples are shown in green font.

> db
Displays the name of currently selected database.

> use **<database_name>**
Creates a new database if it doesn't exist, otherwise it will return the existing database.

> show dbs
Displays the list of databases. Default db is test. If you create a new database then you need to create a new collection or insert atleast one document into the new database, for this to be shown using the 'show dbs' command.

> db.dropDatabase()
This will delete the selected database (the database you have selected using 'use <database_name>' command). If you have not selected any database, then it will delete default 'test' database.

3. Collections

> db.createCollection(name, options)
Creates a new collection. The first parameter is the name of the collection (string); the second parameter is optional and specifies options on memory size and indexing.

```
> db.createCollection("students")
```

> show collections
Show the collections that are present in the selected database. **Note:** In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document using command db.<collection_name>.insert().

```
> db.test.insert({"name" : "This is a test"})
> show collections
```

> db.**<collection_name>**.drop()
The method will return true, if the selected collection is dropped successfully, otherwise it will return false.

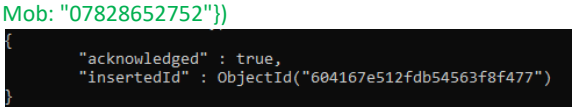
4. Data Types

Data Types	Description
String	This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
Integer	For number
Double	Floating point
Boolean	True/False
Arrays	To store arrays or list or multiple values into one key
Timestamp	Current timestamp (activity logs)
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	This datatype is used for embedded documents
Object ID	This datatype is used to store the document's ID
Null	Null values

5. Create Documents

> db.**<collection_name>**.insertOne(document)
If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.

```
>db.students.insertOne({
        First_Name: "Hello",
        Last_Name: "World",
        Age: 27,
        E_mail: "Hello.World@Exeter.ac.uk",
        Mob: "07828652752"})
```



Note: In the inserted document, if we don't specify the **_id parameter**, then MongoDB assigns a unique ObjectId for this document (the screenshot above acknowledges the creation of a key by MongoDB). The **_id** is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows: _id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer).

```
>db.students.insertOne({
        _id:346,
        First_Name: "Lost",
        Last_Name: "World",
        Age: 37,
        E_mail: "Lost.World@Exeter.ac.uk",
        Mob: "07828653123"})
{ "acknowledged" : true, "insertedId" : 346 }
```

Note: In the inserted document, if we don't specify the **_id parameter**, then MongoDB assigns a unique ObjectId for this document. If **_id exists**, then reports duplicate ("errmsg" : "E11000 duplicate key error collection").

> db.**<collection_name>.**insert(document)

If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.

```
> db.modules.insert({
_id : 1,
module: "BEMM459",
module_name: "Databases",
module_leader: "Nav",
keywords: ["SqlLite", "Redis", "MongoDB", "Neo4J"]})
```

Note: In the inserted document, if we don't specify the **_id parameter**, then MongoDB assigns a unique ObjectId for this document. If **_id exists**, then reports duplicate ("errmsg" : "E11000 duplicate key error collection").

```
> db.modules.insert([
{
_id : 5,
module: "BEMM115",
module_name: "Programming",
module_leader: "Nav",
keywords: ["C", "VB", "Java", "Python"]
},
{
_id : 6,
module: "BEMM457",
module_name: "Topics in Business Analytics",
module_leader: "Staff X1",
keywords: ["Big Data", "Data Science", "Cloud Computing"]
},
{
_id : 7,
module: "BEMM460",
module_name: "Statistics and Mathematics for Business Analytics",
module_leader: "Staff S1",
keywords: ["R", "SPSS"]
}]
)
```

Note: Passing an array of documents into the insert() method.

> db.**<collection_name>.**insertMany(document array)

You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

```
> db.modules.insertMany([
{
_id : 8,
module: "BEMM115_J",
module_name: "Programming ** Second Intake**",
module_leader: "Nav",
keywords: ["C", "VB", "Java", "Python"]
},
{
_id : 9,
module: "BEMM457_J",
module_name: "Topics in Business Analytics ** Second Intake**",
module_leader: "Staff X1",
keywords: ["Big Data", "Data Science", "Cloud Computing"]
},
{
_id : 10,
module: "BEMM460_J",
module_name: "Statistics and Mathematics for Business Analytics ** Second Intake**",
module_leader: "Staff S1",
keywords: ["R", "SPSS"]
}]
)
```

6. Query Documents

> db**.<collection_name>.**find()
This method retrieves all the documents in the collection.

```
> db.modules.find()
```

> db. **.<collection_name>.**find().pretty()
Pretty() displays the results in a formatted way.

```
> db.modules.find().pretty()
```

> db. **.<collection_name>.**findOne()
To return only one document use findOne(). The document is formatted.

```
> db.modules.findOne()
```

6A. Operators

Operators (SQL equivalents)	Syntax	Example
Equal to **(=)**	{<key>:{$eg;<value>}}	db.**modules**.find({"keywords":"C"}).pretty() db.**students**.find({"Age":33}).pretty()
Less than **(<)**	{<key>:{**$lt**:<value>}}	db.**students**.find({"Age":{**$lt:**35}}).pretty()
Less than or equal to **(<=)**	{<key>:{**$lte**:<value>}}	db.**students**.find({"Age":{**$lte:**35}}).pretty()
Greater than **(>)**	{<key>:{**$gt**:<value>}}	db.**students**.find({"Age":{**$gt:**35}}).pretty()
Greater than or equal to **(>=)**	{<key>:{**$gte**:<value>}}	db.**students**.find({"Age":{**$gte:**35}}).pretty()
Not equals **(!-)**	{<key>:{**$ne**:<value>}}	db.**students**.find({"Age":{**$ne:**35}}).pretty()
Values in an array **IN [..,..]**	{<key>:{**$in**:[<value1>, <value2>,……<valueN>]}}	db.**students**.find({"Age":{**$in:**[32, 35, 37]}}).pretty() db.**modules**.find({"keywords":{$in:["R", "VB", "Data Science", "AnyLogic"]}}).pretty()
Values in an array **NOT IN [..,..]**	{<key>:{**$nin**:<value>}}	db.**students**.find({"Age":{**$nin:**[32, 35, 37]}}).pretty() db.**modules**.find({"keywords":{**$nin**:["R", "VB", "Data Science", "AnyLogic"]}}).pretty()

Note: attributes are case sensitive ('A' and 'a' are different).

6A. AND

> db.**<collection_name>.**find({ **$and**: **[** {<key1>:<value1>}, { <key2>:<value2>} **]** })
Use the $and keyword.

```
>db.students.find({$and:[{"Age":{$nin:[32, 35, 37]}}, {"Mob":"07828653123"}]}).pretty()

>db.students.find({"Age":{$nin:[32, 35, 37]}, "Mob":"07828653123"}).pretty()
```
Alternative method: without using AND – refer to 6C.

6B. OR
>db.**<collection_name>**.find({ **$or**: **[{**<key1>: value1}, {<key2>:value2} **]** })
Use the $or keyword.

```
>db.students.find({$or:[{"Age":{$nin:[32, 35, 37]}}, {"Mob":"07828653123"}]}).pretty()
```

6C. use of OR and AND
> db.**<collection_name>.**find({<key1>:<value1>, **$or**: **[** {<key1>: value1}, {<key2>:value2} **] }**)
Use of 'and' and 'or' together. Note that there are two way of including the logical operator 'and' – refer to 6A.

```
>db.students.find({"_id":{$gt:3}, $or:[{"Age":{$nin:[32, 35, 37]}}, {"Mob":"07828653123"}]}).pretty()
```

6D. NOT IN
> db.**<collection_name>.**find({**$NOT**: **[**{key1: value1}, {key2:value2}**]}**)
Use the $not keyword.

```
>db.students.find({"_id": { $not: {$gt:3} } }).pretty()
```

6E. Projection
> db.**<collection_name>**.find({},{KEY:1})
Projection means selecting only the necessary data rather than selecting whole of the data of a document. MongoDB's find() method **accepts second optional parameter that is list of fields that you want to retrieve**. In MongoDB, when you execute find() method, then it displays all fields of a document (default behaviour). To limit this, you need to set a list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the fields.

```
>db.students.find({},{"First_Name":1,_id:0}).pretty()
```
Note that the *_id* field is always displayed while executing find() method, if you don't want this field, then you need to set it as 0.

6F. Limiting records
> db.**<collection_name>**.COLLECTION_NAME.find().limit(NUMBER)
To limit the records in MongoDB, you need to use limit() method. The method accepts one number type argument, which is the number of documents that you want to be displayed.

```
> db.students.find({},{"First_Name":1,_id:0}).limit(2)
```

>db.**<collection_name>.**find().limit(NUMBER).skip(NUMBER)
Skip() method accepts number type argument and is used to skip the number of documents. The default value of skip is 0.

```
> db.students.find({},{"First_Name":1,_id:0}).limit(1).skip(1)
```

6G. Sorting records
> db.**<collection_name>.**find().sort({KEY:1})
Accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. **1 is used for ascending order** while **-1 is used for descending order**. By default, the sort() method will display the documents in ascending order.

```
>db.students.find({},{"First_Name":1,_id:0}).sort({"First_Name":-1})
>db.students.find({},{"First_Name":1,_id:0}).sort({"First_Name":1})
```

7. Update Documents

>db.**<collection_name>.**update(SELECTION_CRITERIA, UPDATED_DATA)
The update() method updates the values in the existing document. By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

```
>db.students.update({'First_Name':'Redis'},{$set:{'title':'This is a new attribute added'}})

>db.students.update({'First_Name':'Redis'},{$set:{'First_Name':'This record is now updated'}})

>db.students.update({'First_Name':'This record is now updated'},{$set:{'Last_Name':'Now the last name is updated'}})

>db.students.update({"Mob":"07828653123"}, {$set:{"E_Mail":"Email attribute updated for all records matching the mobile number"}},{multi:true})
```

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

>>db.**<collection_name>.**save({_id:ObjectId(),NEW_DATA})
The save() method replaces the existing document with the new document passed in the save() method.

```
>db.students.save(
{
"_id":5,
"First_Name": "Replacing ..",
"Last_Name": "Using Save ...",
Age: 21,
E_mail: "RUS@Exeter.ac.uk",
Mob: "Replaced record"
})
As
```

8. Delete Documents

> db.**<collection_name>.**remove(DELLETION_CRITTERIA)
Remove() method is used to remove a document from the collection. remove() method accepts two parameters – both of which are optional. One is deletion criteria and second is justOne flag (if set true or 1, then removes only one document).

```
> db.students.remove({"First_Name":"Sql"})
```

> db.**<collection_name>.**remove(DELETION_CRITERIA,**1**)
If there are multiple records and you want to delete only the first record, then set justOne parameter in remove() method.

> db.**<collection_name>.**remove(**{}**)
If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection.

9. Exit Mongo Client

> exit

References:

- *https://docs.mongodb.com/manual/reference/mongo-shell/*
- *https://www.tutorialspoint.com/*

/End