**What is MongoDB?**MongoDBis a NoSQL based query language having features mentioned below.

* It is schema less, meaning there is no pre-defined structure to the data that you insert.
* Instead of tables or entities, it has Collections. Each collection can have multiple documents.
* Instead of tuples or rows, it has Documents. Each document contains multiple key-value pairs (also known as JSON Objects).
* When creating relations, you don’t associate tables when querying data, but add data of related tables directly into the parent table at time of initialization.  
  For example:  
  In SQL, when making a 1-to-many relationship between Student and Course, we’d just pass foreign key of student to courses and then use it to retrieve shared data at time of retrieval.  
  Whereas in MongoDB, you’ll have to create an array of course objects(documents) in each document of Student.  
   **Example**

{// Student collection document(object)

**name**: "Shakir",

**courses**: [ //array of course documents(objects)

{

**code**: "XYZ",

**name**: "Advanced XYZ Course",

**Instructor**: "Dr. Instructor",

// other course details

},

{

**code**: "ABC",

**name**: "Introduction to ABC",

**instructor**: "Prof. Instructor",

// other course details

}

]

}

**How to start mongoDB on Lab PCs?**

1. Locate mongoDB folder (most likely in C drive).
2. Keep going into subfolders until you reach ***bin*** folder.
3. Open **mongod** file there.
4. If it throws an error showing:

* ***“Data directory C:\data\db not found”,*** just go to C directory and make data folder there and create db folder inside of it.
* ***“Data directory C:\data\db not empty”,*** just go to the specified folder and empty it.

1. Run **mongod**again. If it shows “waiting for connections” (meaning it’s working).
2. Now, run **mongo** file in the same **bin** folder without closing the other shell.
3. Once connection is established, you can use mongoDB commands.

Make sure to follow the right naming conventions when calling mongoDB functions as they are case sensitive.

Some common commands in MongoDB

1. *show dbs*
   1. displays a list of all available databases.
2. use ***<database name>***
   1. switches to the specific database (creates a new database and switches to it if doesn’t exist).
3. *show collections*
   1. displays a list of all available collections in the selected database.
4. db.createCollection(***<collection name>, options***)
   1. creates a new collection in the database selected by **query#2** above.
   2. The **options** parameter is optional and is used to define properties/constraints on the collection (mainly a JSON object).
5. db.***<collection name>***.insert({name: “Shakir”})
   1. inserts a document ***{name: “Shakir”}*** into the specified database. Here **name** is the key and ***Shakir*** is the value mapped against **name**.
6. *db.dropDatabase()*
   1. drops the database selected by using **query#2** above.
   2. If you want to delete new database named “**myDB**” then dropDatabase() command would be as follows:
      1. ***use myDB***
      2. ***db.dropDatabase();***
7. db.***<collection name>*.**drop()
   1. drops the specified collection.

**How to insert a document containing multiple key-value pairs (JSON object)?**

db.*<collection name>*.insert({  
 name: “Shakir”,

age: 24,  
 gender: “Male”,

classes: [“DB”, “DS”]   
})

**How to insert multiple documents in a single insert function?**

db.*<collection name>*.insert ([  
 {  
 name: “Shakir”,

age: 24,  
 gender: “Male”,

classes: [“DB”, “DS”]   
},  
{

name: “Hussain”,

age: 25,

gender: “Male”,

likes: 20   
}

])

**How to insert data of another collection(table) in a single insert function?**

db.*<collection name>*.insert ({  
 name: “Shakir”,

age: 24,  
 gender: “Male”,

classes: [“DB”, “DS”],  
 courses: [  
 {  
 courseTitle: “DB”,  
 message: “This is DB course”,  
 },

{

courseTitle: “DS”,  
 message: “This is DS course”,

}   
})

Similarly, you can add as many nested level collections as you want.  
  
**Retrieval Queries**

* db.***<collection name>***.find()
  + retrieves data from the specified collection.
* db.***<collection name>***.find().pretty()
  + retrieves data in a formatted order.
* You can learn the **AND**, **OR**, **Update**, and other commands from the other manual.

**Additional Queries**

Suppose you have a collection as shown below.

{\_id: ObjectId("6568a61cef6be7bb4cd8b46e"), a: 1},

{

\_id: ObjectId("6568ab92ef6be7bb4cd8b46f"),

name: 'Shakir',

age: 24,

gender: 'Male',

classes: [ 'DB', 'DS'],

courses: {title: 'DB Lab', message: 'This is DB Lab'}

}

**How to retrieve selected key-value pairs, and not all of them?**

* db.***<collection name>***.find({name: "Shakir"}, {name: 1, courses: 1})
  + This will retrieve just the name and courses array for document with name = Shakir.

**How to retrieve document based on data from nested document(object?)**

* db.***<collection name>.****find({'courses.title': "DB Lab"})*

**What if I had an array of nested documents?**

This statement inserts documents into the survey collection:

|  |
| --- |
| db.***<collection name>***.insert ( [ |
| {"\_id": 1, "results": [ {"product": "abc", "score": 10 }, |
| {"product": "xyz", "score": 5 }]}, |
| {"\_id": 2, "results": [ {"product": "abc", "score": 8 }, |
| {"product": "xyz", "score": 7 }]}, |
| {"\_id": 3, "results": [ {"product": "abc", "score": 7 }, |
| {"product": "xyz", "score": 8 }]}, |
| {"\_id": 4, "results": [ {"product": "abc", "score": 7 }, |
| {"product": "def", "score": 8 }]} |
| ] ) |

The following query matches only those documents where the results array contains at least one element with both product equal to "xyz" and score greater than or equal to 8:

|  |
| --- |
| db.<collection name>.find ({ |
| results: {$elemMatch: {product: "xyz", score: {$gte: 8} }} |
| }) |