MongoDB CRUD Operations

In this document you can find sample CRUD operations for a MongoDB deployment. You can execute the commands from the command line after connecting to the MongoDB database.

Basic Commands

```
// List all available databases
show dbs

// Get the current database
db

// Connect to another database
use sample_airbnb

// Get Database collections
show collections
```

Create

insertOne

```
// Insert one document into the users collection
use test

db.users.insertOne({
    "firstName" : "John",
    "lastName" : "Smith",
    "average" : 8,
    "materials": [
        "NoSQL",
        "SQL",
        "Python"
    ],
    "address" : {
        "country": "France",
        "city": "Paris"
    }
})
```

insertMany

```
// Insert multiple documents into the users collection
use test
```

```
db.users.insertMany([{
        "firstName": "John",
        "lastName": "Flores",
        "average": 7,
        "materials": [
            "NoSQL",
            "SQL",
            "Java"
        ],
        "address": {
            "country": "France",
            "city": "Nice"
        }
    },
    {
        "firstName": "Jane",
        "lastName": "Smith",
        "average": 9,
        "materials": [
            "NoSQL",
            "SQL",
            "Python",
            "Java"
        ],
        "address": {
            "country": "Italy",
            "city": "Rome"
        }
    }
])
```

Read

findOne

```
// Find the first document in the users collection
use test

db.users.findOne()
```

```
// Find the first document where lastName is Smith
db.users.findOne({ lastName: "Smith" })

// Find the first document where lastName is Smith and firstName is Jane
db.users.findOne({ lastName: "Smith", firstName: "Jane" })
```

Projection

With projection, we can include or exclude certain fields

Inclusion Projection

For inclusion projection, we can choose the fields **to select** from the collection.

```
// Find only the firstName and lastName of the first document
Use test

db.users.findOne({}, { firstName:1, lastName:1 })

// _id field is always returned, unless we exclude it
db.users.findOne({}, { firstName:1, lastName:1, _id:0 })
```

Exclusion Projection

With exclusion projection, we choose the fields **not to select** from the collection.

```
// Find the first document, do not return the "materials" information
use test

db.users.findOne({}, { materials: 0 })
```

We cannot mix inclusion and exclusion in the same projection (**except for excluding _id**).

```
// Mixing inclusion and exclusion causes an error
use test

db.users.findOne({}, { materials:0, firstName:1  })
```

find

To find more than one result, we can use the “find” method. This method accepts the same arguments as the “findOne” method. The first parameter is the filter, and the second parameter is the projection.

```
// Find all the users that have lastName of smith
use test

db.users.find({ lastName: "Smith" })

// Find only firstName, lastName and average of users with firstName John
db.users.find({ firstName: "John" }, { firstName:1, lastName:1, average:1,
_id:0 })
```

Nested Objects

We can filter by fields inside nested documents using the dot notation.

```
// Find all users that live in France
use test

db.users.find({ "address.country": "France" })
```

We can filter by complete documents as well, as long as we use the complete document in the filter

```
// Querying by country alone returns no matches
use test

db.users.find({ address: { country: "France" } })

// Querying by country and city returns the results
db.users.find({ address: { country: "France", city: "Paris" } })
```

Ranges

To search in a range of values, we can use different operators.

$lt

```
// Find all users with average less than 8
use test

db.users.find({ average: { $lt: 8 } })
```

$lte

```
// Find all users with average less than or equal to 8
use test

db.users.find({ average: { $lte: 8 } })
```

$gt

```
// Find all users with average greater than 8
use test

db.users.find({ average: { $gt: 8 } })
```

$gte

```
// Find all users with average greater than or equal to 8
use test

db.users.find({ average: { $gte: 8 } })
```

$in

```
// Find all users with average is 7 or 9
use test

db.users.find({ average: { $in: [7,9] } })
```

$nin

```
// Find all users with average is not 7 nor 9
use test

db.users.find({ average: { $nin: [7,9] } })
```

Boolean Logic

$and

```
// Find all users with lastName Smith and firstName John
use test

db.users.find({ $and : [
        { firstName: "John" },
        { lastName: "Smith" }
    ]
})
```

$or

```
// Find all users with lastName Smith or firstName John
use test

db.users.find({ $or : [
        { firstName: "John" },
        { lastName: "Smith" }
    ]
})
```

We can also build complex logic using the different operators

```
// Find all users where:
// firtName is John and country is France or Italy
// or
// average is less than 5
use test

db.users.find({
    $or : [
        { $and : [
                {
                firstName: "John"
                },
                {
                    "address.country": { $in: [
                            "France",
```

```
                            "Italy"
                        ]
                    }
                }
            ]
        },
        {
            average : { $lt : 5 }
        }
    ]
})
```

Arrays

To find items with arrays that contain a specific value, we can pass the value in the query.

```
// Find all users where materials contain Python
use test

db.users.find({ materials: "Python" })
```

We can also look for exact matches of the array, by passing the complete array **in order** in the query.

```
// Find all users where materials is [ "NoSQL", "SQL", "Python" ]
use test

db.users.find({ materials: [ "NoSQL", "SQL", "Python" ] })

// If we change the order, we don't get any results
db.users.find({ materials: [ "NoSQL", "Python", "SQL"] })
```

$in

We can use $in to search for multiple elements in the array.

```
// Find all users that have Python or Java in materials
use test
```

```
db.users.find({ materials : { $in : [ "Java", "Python" ] } })
```

$all

We can use $all to search for arrays that contain all of the elements.

```
// Find all users that have Python and Java in materials
use test

db.users.find({ materials : { $all : [ "Java", "Python" ] } })
```

$size

We can use $size to filter by array length.

```
// Find all users that have 3 materials
use test

db.users.find({ materials : { $size : 3 } })
```

$elemMatch

The $elemMatch operator matches documents that contain an array field with at least one element that matches all the specified query criteria.

```
// Insert documents in scores collection
use test

db.scores.insertMany([
    { _id: 1, results: [82,85,88]},
    { _id: 2, results: [75,88,89]}
])

// Find all scores that have a result between 80 and 85
db.scores.find(
   { results: { $gte: 80, $lt: 85 } }
)
```

Without using $elemMatch, the results are wrong, since the second result doesn't contain a score between 80 and 85.
To get an accurate result, we should use $elemMatch.

```
// Find all scores that have a result between 80 and 85
use test

db.scores.find(
   { results: { $elemMatch: { $gte: 80, $lt: 85 } } }
)
```

We can also use $elemMatch with arrays of sub-documents.

```
//Insert documents in the survey collection
use test

db.survey.insertMany( [
   { "_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 } ] }
] )

// Find all surveys for product xyz that have a score >= 8
db.survey.find(
   { results: { $elemMatch: { product: "xyz", score: { $gte: 8 } } } }
)
```

Update

updateOne

Update the first document that matched the query. This method accepts two parameters:

1. Query: Which documents to update
2. Mutation: What to change in the documents

updateMany

Update all the documents that match the query. This method accepts the same parameters as updateOne.

Mutation

Mutation is an object describing the changes to make to each document.

Some of the supported mutation types are listed below.

$set

```
// Add a phoneNumber for the users
use test

db.users.updateMany({}, [ { $set: { phoneNumber: "0677445522" } } ])

db.users.find({}, { firstName:1, phoneNumber:1, _id:0 })
```

$unset

```
// Remove the phoneNumber field for users with firstName John
use test

db.users.updateMany({ firstName: "John" }, [ { $unset: [ "phoneNumber" ] } ])

db.users.find({}, { firstName:1, phoneNumber:1, _id:0 })
```

$inc

```
// Increment the average of all users by 2
use test

db.users.find({}, { firstName:1, lastName:1, average:1, _id:0 })

db.users.updateMany({}, { $inc: { average : 2 } })

db.users.find({}, { firstName:1, lastName:1, average:1, _id:0 })

// To decrement a value, we can increment by a negative value
```

```
// Decrement the average of all users by 1
db.users.updateMany({}, { $inc: { average : -1 } })

db.users.find({}, { firstName:1, lastName:1, average:1, _id:0 })
```

$mul

```
// Multiply the average of all users by 2
use test

db.users.find({}, { firstName:1, lastName:1, average:1, _id:0 })

db.users.updateMany({}, { $mul: { average : 2 } })

db.users.find({}, { firstName:1, lastName:1, average:1, _id:0 })

// To divide a value, we can multiply by a 1/value
// Divide the average of all users by 2
db.users.updateMany({}, { $mul: { average : 1/2 } })

db.users.find({}, { firstName:1, lastName:1, average:1, _id:0 })
```

$push

```
// Add "C" as a material for users
use test

db.users.find({}, { firstName:1, lastName:1, materials:1, _id:0 } )

db.users.updateMany({}, { $push: { materials : "C" } })

db.users.find({}, { firstName:1, lastName:1, materials:1, _id:0 } )
```

We can add $each to push multiple elements

```
// Add "C++" and "PHP" as materials for users
use test

db.users.find({}, { firstName:1, lastName:1, materials:1, _id:0 } )

db.users.updateMany({}, { $push: { materials : { $each: [ "C++", "PHP" ] }
} } )
```

```
db.users.find({}, { firstName:1, lastName:1, materials:1, _id:0 } )
```

$pop

```
// Remove the first element from the materials arrays
use test

db.users.find({}, { firstName:1, lastName:1, materials:1, _id:0 } )

db.users.updateMany({}, { $pop: { materials : -1 } } )

db.users.find({}, { firstName:1, lastName:1, materials:1, _id:0 } )

// Remove the last element from the materials arrays
db.users.updateMany({}, { $pop: { materials : 1 } } )

db.users.find({}, { firstName:1, lastName:1, materials:1, _id:0 } )
```

$pull

```
// Remove Python from the materials
use test

db.users.find({}, { firstName:1, lastName:1, materials:1, _id:0 } )

db.users.updateMany({}, { $pull: { materials : "Python" } } )

db.users.find({}, { firstName:1, lastName:1, materials:1, _id:0 } )
```

$pullAll

```
// Remove Java and C from the materials
use test

db.users.find({}, { firstName:1, lastName:1, materials:1, _id:0 } )

db.users.updateMany({}, { $pullAll : { materials : [ "Java", "C" ] } } )

db.users.find({}, { firstName:1, lastName:1, materials:1, _id:0 } )16
```

$addToSet

```
// Add NoSQL and SQL to the materials if they don't already exist
use test

db.users.find({}, { firstName:1, lastName:1, materials:1, _id:0 } )

db.users.updateMany({}, { $addToSet : { materials : { $each : [ "NoSQL",
"SQL" ] } } } )

db.users.find({}, { firstName:1, lastName:1, materials:1, _id:0 } )
```

replaceOne

There are cases where we don;t know the exact modification for a document, and we want to reset the complete document. In that case, we can use the replaceOne method. The method takes 2 parameters:

1. Query: Which document to replace
2. New document: The new version of the document

```
// Replace the details of the user with firstName "Jane"
use test

db.users.replaceOne({ firstName: "Jane" }, {
    firstName: 'Jane',
    lastName: 'Smith',
    average: 10,
    materials: [ 'SQL', 'NoSQL' ],
    address: { country: 'Italy', city: 'Venice' }
})
```

Delete

The delete methods accept a single parameter:

1. Query: Which document to delete.

deleteOne will delete the first document found, and deleteMany will delete all matching documents.

deleteOne

```
// Delete one user with average less than 9
use test

db.users.deleteOne({ average : { $lt: 9 } })
```

deleteMany

```
// Delete all users with NoSQL in the materials
use test

db.users.deleteMany({ materials : 'NoSQL'})
```