CS 207: Applied Database Practicum
Week 8

Varun Dutt

School of Computing and Electrical Engineering
School of Humanities and Social Sciences
Indian Institute of Technology Mandi, India

Scaling the Heights

NoSQL

NoSQL is basically a database used to manage huge sets of unstructured data. NoSQL provides following advantages:

- **Dynamic Schemas**: Adding new column is easy in NoSQL.
- **Sharding**: Large databases are partitioned into small, faster and easily manageable databases.
- **Replication**: If one DB server goes down, data is automatically restored using its copy created on another server in network.
- **Integrated Caching**: Many NoSQL databases have support for Integrated Caching, where in the frequently demanded data is stored in cache to make the queries fater.

MongoDB

- MongoDB is a NoSQL database written in C++ language.
- MongoDB is a document database. Each database contains collections which in turn contains documents. Each document can be different with varying number of fields.
- The rows (or documents as called in MongoDB) doesn't need to have a schema defined beforehand. Instead, the fields can be created on the fly.
- The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.

Common terms

Below are the a few of the common terms used in MongoDB

- **_id** – This is a field required in every MongoDB document. The _id field represents a unique value in the MongoDB document. The _id field is like the document's primary key. If you create a new document without an _id field, MongoDB will automatically create the field.
- **Collection** – This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS. A collection exists within a single database. Collections don't enforce any sort of structure.
- **Cursor** – This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.

Common terms

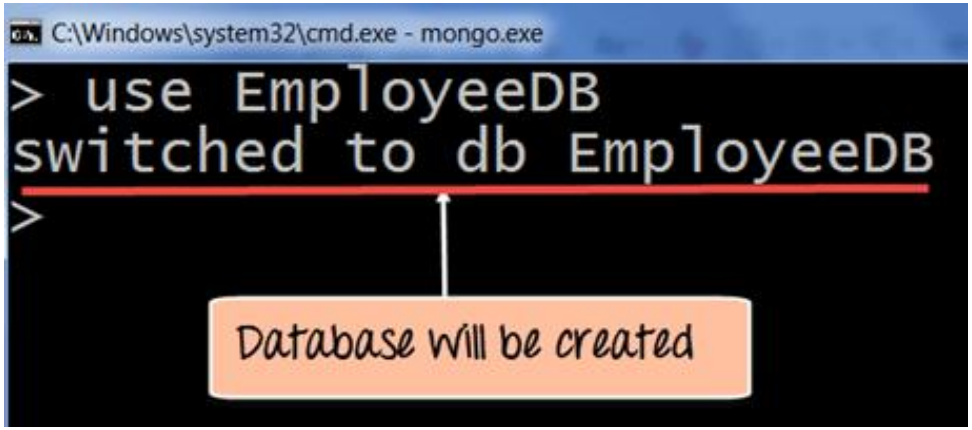
- **Database** – This is a container for collections like in RDMS wherein it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.
- **Document** - A record in a MongoDB collection is basically called a document. The document in turn will consist of field name and values.
- **Field** - A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.

Difference between MongoDB and RDBMS

RDBMS	MongoDB	Difference
Table	**Collection**	In RDBMS, the table contains the columns and rows which are used to store the data whereas, in MongoDB, this same structure is known as a collection. The collection contains documents which in turn contains Fields, which in turn are key-value pairs.
Row	**Document**	In RDBMS, the row represents a single, implicitly structured data item in a table. In MongoDB, the data is stored in documents.
Column	**Field**	In RDBMS, the column denotes a set of data values. These in MongoDB are known as Fields.
Joins	**Embedded documents**	In RDBMS, data is sometimes spread across various tables and in order to show a complete view of all data, a join is sometimes formed across tables to get the data. In MongoDB, the data is normally stored in a single collection, but separated by using Embedded documents. So there is no concept of joins in Mongodb.

Create Database

- The "use" command is used to create a database in MongoDB. If the database does not exist a new one will be created.



Creating a collection

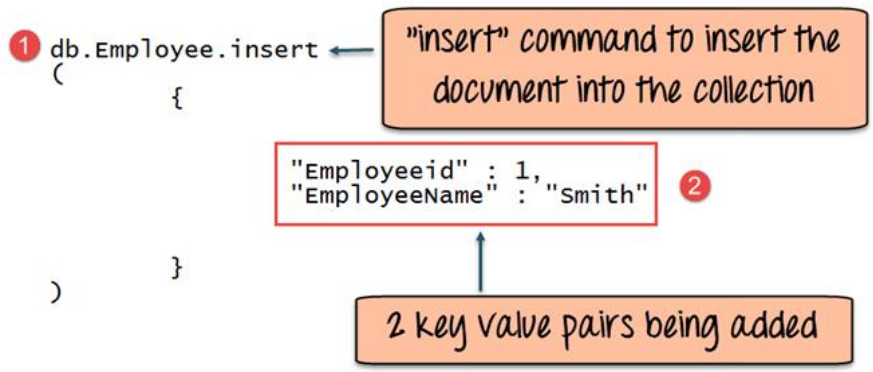
- The easiest way to create a collection is to insert a record (which is nothing but a document consisting of Field names and Values) into a collection. If the collection does not exist a new one will be created.

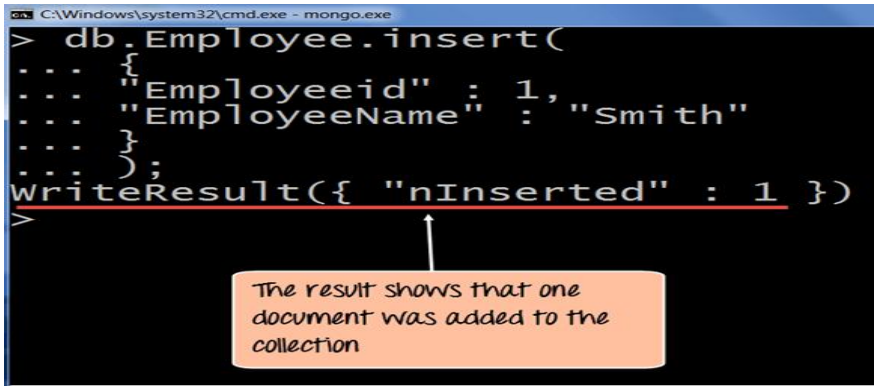
For eg.:

```
db.Employee.insert
(
        {
                "Employeeid" : 1,
                "EmployeeName" : "Martin"
        }
)
```

Adding documents to collection

- MongoDB provides the insert () command to insert documents into a collection.
- Step 1) Write the "insert" command
- Step 2) Within the "insert" command, add the required Field Name and Field Value for the document which needs to be created.





Add MongoDB Array

The "insert" command can also be used to insert multiple documents into a collection at one time.
Step 1) Create a JavaScript variable to hold the array of documents.
Step 2) Add the required documents with the Field Name and values to the variable.
Step 3) Use the insert command to insert the array of documents into the collection.

Add MongoDB Array

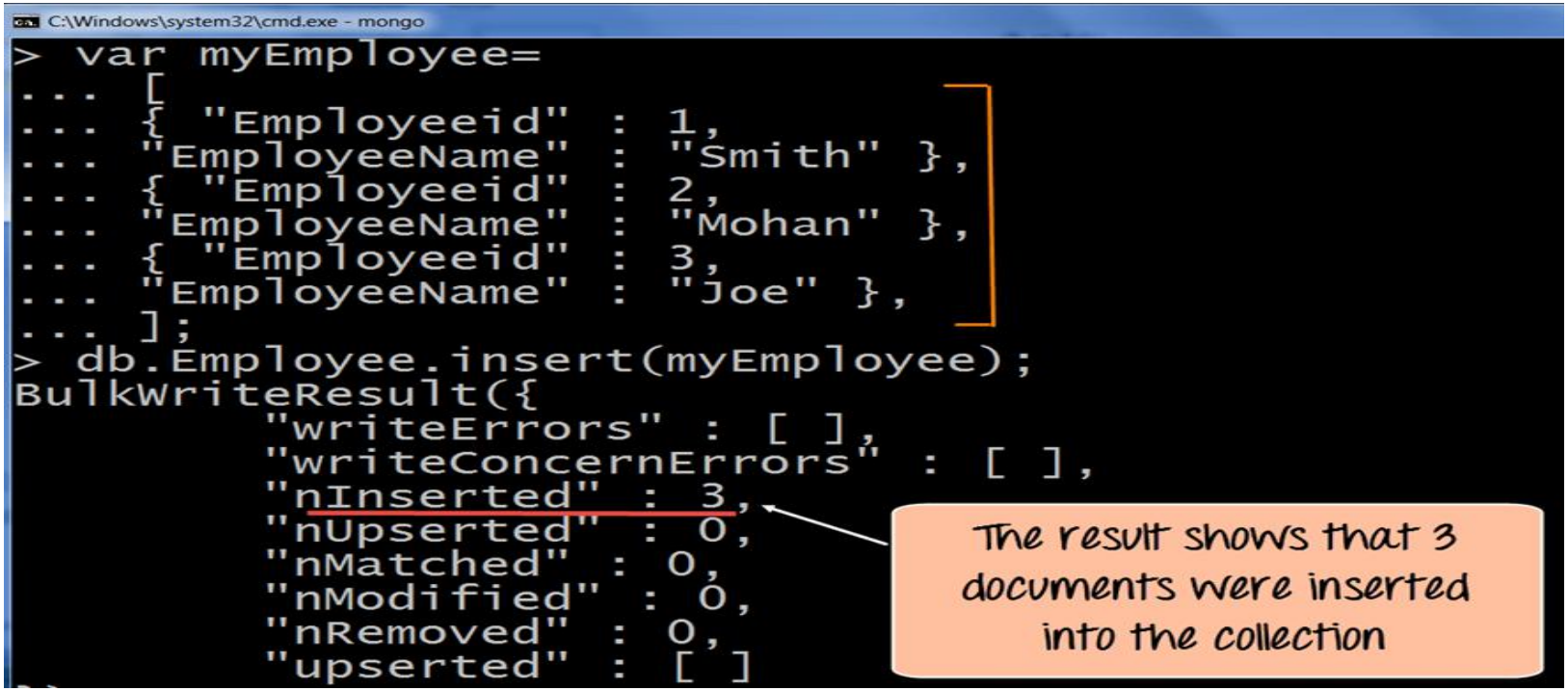
```
var myEmployee=
        [

                {
                        "Employeeid" : 1,
                        "EmployeeName" : "Smith"
                },
                {
                        "Employeeid"   : 2,
                        "EmployeeName" : "Mohan"
                },
                {
                        "Employeeid"   : 3,
                        "EmployeeName" : "Joe"
                },

        ];

        db.Employee.insert(myEmployee);
```

Add MongoDB Array



MongoDB multiple document Insertion example

```
>>> db.inventory.insertMany([
...     { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },
...     { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
...     { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm" } }
... ])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5bbae4a85e693eb83a1999ac"),
        ObjectId("5bbae4a85e693eb83a1999ad"),
        ObjectId("5bbae4a85e693eb83a1999ae")
    ]
}
>>>
```

Mongodb ObjectId()

- By default when inserting documents in the collection, if you don't add a field name with the _id in the field name, then MongoDB will automatically add an Object id field.
- MongoDB uses this as the primary key for the collection so that each document can be uniquely identified in the collection.
- If you want to ensure that MongoDB does not create the _id Field when the collection is created and if you want to specify your own id as the _id of the collection, then you need to explicitly define this while creating the collection.
- When explicitly creating an id field, it needs to be created with _id in its name.

Mongodb ObjectId()

For eg:

db.Employee.insert({_id:10, "EmployeeName" : "Smith"})

We are assuming that we are creating the first document in the collection and hence in the above statement while creating the collection, we explicitly define the field _id and define a value for it.

Printing in JSON format

JSON is a format called JavaScript Object Notation, and is just a way to store information in an organized, easy-to-read manner.

For eg:

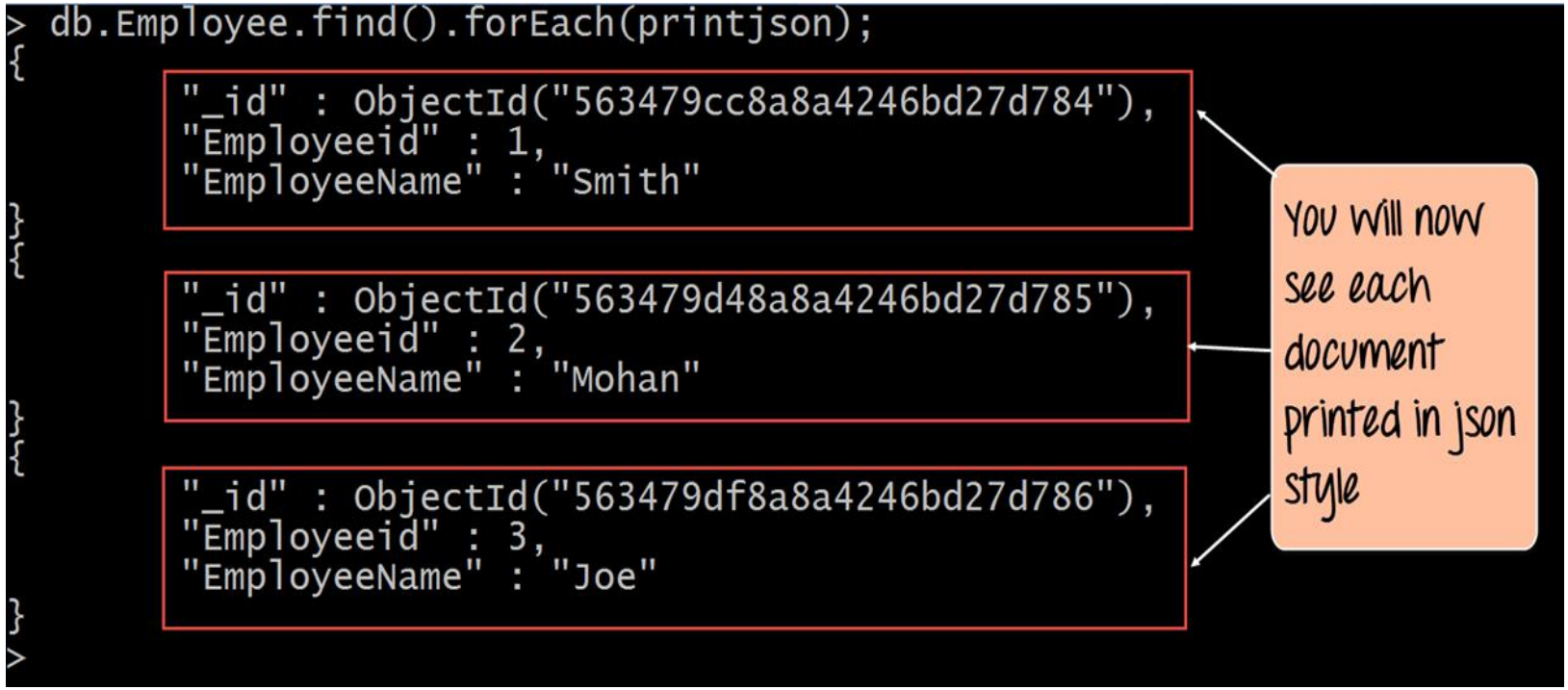
db.Employee.find().forEach(printjson)

Explanation:

The first change is to append the function called for Each() to the find() function. What this does is that it makes sure to explicitly go through each document in the collection. In this way, you have more control of what you can do with each of the documents in the collection.

The second change is to put the printjson command to the forEach statement. This will cause each document in the collection to be displayed in JSON format.

Printing in JSON format



MongoDB Query Document using find()

- The find command is an in-built function which is used to retrieve the documents in the collection.
- Eg: The output shows all the documents which are present in the collection.

```
> db.mydb.find({})
{ "_id" : ObjectId("5bba0b1dece76b6dcd622f33"), "id" : 4, "name" : "AJ" }
{ "_id" : ObjectId("5bba0ca0ece76b6dcd622f34"), "id" : 1, "name" : "Smith" }
{ "_id" : ObjectId("5bba0ca0ece76b6dcd622f35"), "id" : 2, "name" : "Mohan" }
{ "_id" : ObjectId("5bba0ca0ece76b6dcd622f36"), "id" : 3, "name" : "Joe" }
{ "_id" : 10, "name" : "Akul" }
>
```

- mydb is the collection name in the MongoDB database

Say we have a collection(table) called *inventory.*

copy

```
db.inventory.find( {} )
```

This operation corresponds to the following SQL statement:

copy

```
SELECT * FROM inventory
```

MongoDB Query Modifications using limit()

- Limit is a modifier used to limit the number of documents which are returned in the result set for a query. The following example shows how this can be done
  - Eg - db.mydb.find().limit(2).forEach(printjson);

```
> db.mydb.find().limit(2).forEach(printjson);
{ "_id" : ObjectId("5bba327dbf97c10766b12f67"), "id" : 4, "name" : "AJ" }
{ "_id" : ObjectId("5bba32a4bf97c10766b12f68"), "id" : 1, "name" : "Smith" }
>
```

- The output clearly shows that since there is a limit modifier, so at most just 2 records are returned as part of the result set based on the ObjectId in ascending order.

MongoDB Query Modifications using sort()

- By using sort() , one can specify the order of documents to be returned based on ascending or descending order of any key in the collection.
- E.g.

1. To print in descending order of id as key-
    - db.mydb.find().sort({id:-1}).forEach(printjson)
2. To print in asscending order of id as key-
    - db.mydb.find().sort({id:1}).forEach(printjson)

```
> db.mydb.find().sort({id:-1}).forEach(printjson)
{ "_id" : ObjectId("5bba327dbf97c10766b12f67"), "id" : 4, "name" : "AJ" }
{ "_id" : ObjectId("5bba32bbbf97c10766b12f6a"), "id" : 3, "name" : "Joe" }
{ "_id" : ObjectId("5bba32b3bf97c10766b12f69"), "id" : 2, "name" : "Mohan" }
{ "_id" : ObjectId("5bba32a4bf97c10766b12f68"), "id" : 1, "name" : "Smith" }
{ "_id" : ObjectId("5bba32ecbf97c10766b12f6b"), "id" : 0, "name" : "Aman" }
> db.mydb.find().sort({id:1}).forEach(printjson)
{ "_id" : ObjectId("5bba32ecbf97c10766b12f6b"), "id" : 0, "name" : "Aman" }
{ "_id" : ObjectId("5bba32a4bf97c10766b12f68"), "id" : 1, "name" : "Smith" }
{ "_id" : ObjectId("5bba32b3bf97c10766b12f69"), "id" : 2, "name" : "Mohan" }
{ "_id" : ObjectId("5bba32bbbf97c10766b12f6a"), "id" : 3, "name" : "Joe" }
{ "_id" : ObjectId("5bba327dbf97c10766b12f67"), "id" : 4, "name" : "AJ" }
>
```

- The output clearly shows that since there is a limit modifier, so at most just 2 records are returned as part of the result set based on the ObjectId in ascending order.

MongoDB Count() function

- MongoDB provides the count() function to know what is the count of documents in a collection as per the query fired.
- E.g. - db.mydb.count()

```
> db.mydb.find({});
{ "_id" : ObjectId("5bba327dbf97c10766b12f67"), "id" : 4, "name" : "AJ" }
{ "_id" : ObjectId("5bba32a4bf97c10766b12f68"), "id" : 1, "name" : "Smith" }
{ "_id" : ObjectId("5bba32b3bf97c10766b12f69"), "id" : 2, "name" : "Mohan" }
{ "_id" : ObjectId("5bba32bbbf97c10766b12f6a"), "id" : 3, "name" : "Joe" }
{ "_id" : ObjectId("5bba32ecbf97c10766b12f6b"), "id" : 0, "name" : "Aman" }
> db.mydb.count()
5
>
```

- The output clearly shows that 5 documents are there in the collection.

MongoDB Remove() function

- MongoDB provides the remove() function to remove documents from a collection. Either all of the documents can be removed from a collection or only those which matches a specific condition.
- E.g.

1. db.mydb.remove()

- If you just issue the remove command, all of the documents will be removed from the collection.

2. db.mydb.remove({id:2})

To remove the documents which have the id as 2.

```
> db.mydb.remove({id:2})
WriteResult({ "nRemoved" : 1 })
> db.mydb.find({});
{ "_id" : ObjectId("5bba327dbf97c10766b12f67"), "id" : 4, "name" : "AJ" }
{ "_id" : ObjectId("5bba32a4bf97c10766b12f68"), "id" : 1, "name" : "Smith" }
{ "_id" : ObjectId("5bba32bbbf97c10766b12f6a"), "id" : 3, "name" : "Joe" }
{ "_id" : ObjectId("5bba32ecbf97c10766b12f6b"), "id" : 0, "name" : "Aman" }
>
```

MongoDB Update() Document

- MongoDB provides the update() command to update the documents of a collection.
- To update only the documents you want to update, you can add a criteria to the update statement so that only selected documents are updated.
- The basic parameters in the command is a condition for which document needs to be updated, and the next is the modification which needs to be performed.
- **Step 1** Issue the update command .
- **Step 2** Choose the condition which you want to use to decide which document needs to be updated.
- **Step 3** Use the set command to modify the Field Name.
- **Step 4** Choose which Field Name you want to modify and enter the new value accordingly.

MongoDB Update() Document Example

- Lets say we want to update the document which has the id 1.

Code -

```
db.mydb.update(
{"id" : 1},
{$set: { "name" : "NewMartin"}});
```

Result -

```
> db.mydb.find({});
{ "_id" : ObjectId("5bba327dbf97c10766b12f67"), "id" : 4, "name" : "AJ" }
{ "_id" : ObjectId("5bba32a4bf97c10766b12f68"), "id" : 1, "name" : "Smith" }
{ "_id" : ObjectId("5bba32bbbf97c10766b12f6a"), "id" : 3, "name" : "Joe" }
{ "_id" : ObjectId("5bba32ecbf97c10766b12f6b"), "id" : 0, "name" : "Aman" }
> db.mydb.update(
... {"id" : 1},
... {$set: { "name" : "NewMartin"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.mydb.find({});
{ "_id" : ObjectId("5bba327dbf97c10766b12f67"), "id" : 4, "name" : "AJ" }
{ "_id" : ObjectId("5bba32a4bf97c10766b12f68"), "id" : 1, "name" : "NewMartin" }
{ "_id" : ObjectId("5bba32bbbf97c10766b12f6a"), "id" : 3, "name" : "Joe" }
{ "_id" : ObjectId("5bba32ecbf97c10766b12f6b"), "id" : 0, "name" : "Aman" }
>
```

Updating Multiple Values

- To ensure that multiple documents are updated at the same time in MongoDB you need to use the multi option because otherwise by default only one document is modified at a time.
- **Step 1** Issue the update command .
- **Step 2** Choose the condition which you want to use to decide which document needs to be updated.
- **Step 3** Choose which Field Name's you want to modify and enter their new value accordingly.

Updating Multiple Values Example

- Lets say we want to update the document which has the id “1” to id “10” and name to “NewSmith”. .
- Code -

```
db.mydb.update
(
    {
        "id" : 1
    },
    {
        $set :
        {
            "name" : "NewSmith",
            "id" : 10
        }
    }
)
```

- Result -

```
> db.mydb.find({});
{ "_id" : ObjectId("5bba327dbf97c10766b12f67"), "id" : 4, "name" : "AJ" }
{ "_id" : ObjectId("5bba32a4bf97c10766b12f68"), "id" : 1, "name" : "NewMartin" }
{ "_id" : ObjectId("5bba32bbbf97c10766b12f6a"), "id" : 3, "name" : "Joe" }
{ "_id" : ObjectId("5bba32ecbf97c10766b12f6b"), "id" : 0, "name" : "Aman" }
> db.mydb.update( { "id" : 1 }, { $set : { "name" : "NewSmith", "id" : 10 } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.mydb.find({});
{ "_id" : ObjectId("5bba327dbf97c10766b12f67"), "id" : 4, "name" : "AJ" }
{ "_id" : ObjectId("5bba32a4bf97c10766b12f68"), "id" : 10, "name" : "NewSmith" }
{ "_id" : ObjectId("5bba32bbbf97c10766b12f6a"), "id" : 3, "name" : "Joe" }
{ "_id" : ObjectId("5bba32ecbf97c10766b12f6b"), "id" : 0, "name" : "Aman" }
>
```

Updating Multiple Documents Example

- Consider the collection *inventory*:

```
>>> db.inventory.find( {} )
{ "_id" : ObjectId("5bbb0f0d14e381a1c506aefc"), "item" : "canvas", "qty" : 100, "size" : { "h" : 28, "w" : 35.5, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506aefd"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506aefe"), "item" : "mat", "qty" : 85, "size" : { "h" : 27.9, "w" : 35.5, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506aeff"), "item" : "mousepad", "qty" : 25, "size" : { "h" : 19, "w" : 22.85, "uom" : "cm" }, "status" : "P" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506af00"), "item" : "notebook", "qty" : 50, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "P" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506af01"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506af02"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, "w" : 30, "uom" : "cm" }, "status" : "D" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506af03"), "item" : "postcard", "qty" : 45, "size" : { "h" : 10, "w" : 15.25, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506af04"), "item" : "sketchbook", "qty" : 80, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506af05"), "item" : "sketch pad", "qty" : 95, "size" : { "h" : 22.85, "w" : 30.5, "uom" : "cm" }, "status" : "A" }
```

- For updating multiple documents we use db.collection.updateMany() method

e.g. we want to update all documents inside *inventory* whose *qty* is less than 50

Code:

```
db.inventory.updateMany(
  { "qty": { $lt: 50 } },
  {
    $set: { "size.uom": "in", status: "P" },
    $currentDate: { lastModified: true }
  }
)
```

Updating Multiple Values Example

- All records where the qty is less than 50 has been updated

```
>>> db.inventory.find( {} )
{ "_id" : ObjectId("5bbb0f0d14e381a1c506aefc"), "item" : "canvas", "qty" : 100, "size" : { "h" : 28, "w" : 35.5, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506aefd"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "in" }, "status" : "P", "lastModified" : ISODate("2018-10-08T08:02:56.622Z") }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506aefe"), "item" : "mat", "qty" : 85, "size" : { "h" : 27.9, "w" : 35.5, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506aeff"), "item" : "mousepad", "qty" : 25, "size" : { "h" : 19, "w" : 22.85, "uom" : "in" }, "status" : "P", "lastModified" : ISODate("2018-10-08T08:02:56.622Z") }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506af00"), "item" : "notebook", "qty" : 50, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "P" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506af01"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506af02"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, "w" : 30, "uom" : "cm" }, "status" : "D" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506af03"), "item" : "postcard", "qty" : 45, "size" : { "h" : 10, "w" : 15.25, "uom" : "in" }, "status" : "P", "lastModified" : ISODate("2018-10-08T08:02:56.622Z") }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506af04"), "item" : "sketchbook", "qty" : 80, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("5bbb0f0d14e381a1c506af05"), "item" : "sketch pad", "qty" : 95, "size" : { "h" : 22.85, "w" : 30.5, "uom" : "cm" }, "status" : "A" }
```

MongoDB BulkWrite example

- MongoDB provides clients the ability to perform write operations in bulk. Bulk write operations affect a *single* collection.
- **bulkWrite() Methods**

bulkWrite() supports the following write operations:

1. insertOne
2. updateOne
3. updateMany
4. replaceOne
5. deleteOne
6. deleteMany

MongoDB BulkWrite example

Consider the document *characters*

```
{ "_id" : 1, "char" : "Brisbane", "class" : "monk", "lvl" : 4 },
{ "_id" : 2, "char" : "Eldon", "class" : "alchemist", "lvl" : 3 },
{ "_id" : 3, "char" : "Meldane", "class" : "ranger", "lvl" : 3 }
```

We want to perform BulkWrite to this collection

MongoDB BulkWrite example

Code

```
>>> try {
...     db.characters.bulkWrite(
...         [
...             { insertOne :
...                 {
...                     "document" :
...                     {
...                         "_id" : 4, "char" : "Dithras", "class" : "barbarian", "lvl" : 4
...                     }
...                 }
...             },
...             { insertOne :
...                 {
...                     "document" :
...                     {
...                         "_id" : 5, "char" : "Taeln", "class" : "fighter", "lvl" : 3
...                     }
...                 }
...             },
...             { updateOne :
...                 {
...                     "filter" : { "char" : "Eldon" },
...                     "update" : { $set : { "status" : "Critical Injury" } }
...                 }
...             },
...             { deleteOne :
...                 { "filter" : { "char" : "Brisbane"} }
...             },
...             { replaceOne :
...                 {
...                     "filter" : { "char" : "Meldane" },
...                     "replacement" : { "char" : "Tanys", "class" : "oracle", "lvl" : 4 }
...                 }
...             }
...         ]
...     );
... }
... catch (e) {
...     print(e);
... }
```

Result:

```
{
    "acknowledged" : true,
    "deletedCount" : 0,
    "insertedCount" : 2,
    "matchedCount" : 0,
    "upsertedCount" : 0,
    "insertedIds" : {
        "0" : 4,
        "1" : 5
    },
    "upsertedIds" : {

    }
}
```

References

- https://www.guru99.com
- https://docs.mongodb.com/manual/

Acknowledgements: Thank you to Swapnil, Dheeraj, and Prajjwal for setting up the MongoDB environment in the lab.