Lecture-20
NoSQL –Mongodb

CS211 - Introduction to Database

MongoDB installation YouTube

https://www.youtube.com/watch?v=FwMwO8pXfq0

MongoDB official manual

https://docs.mongodb.com/manual/

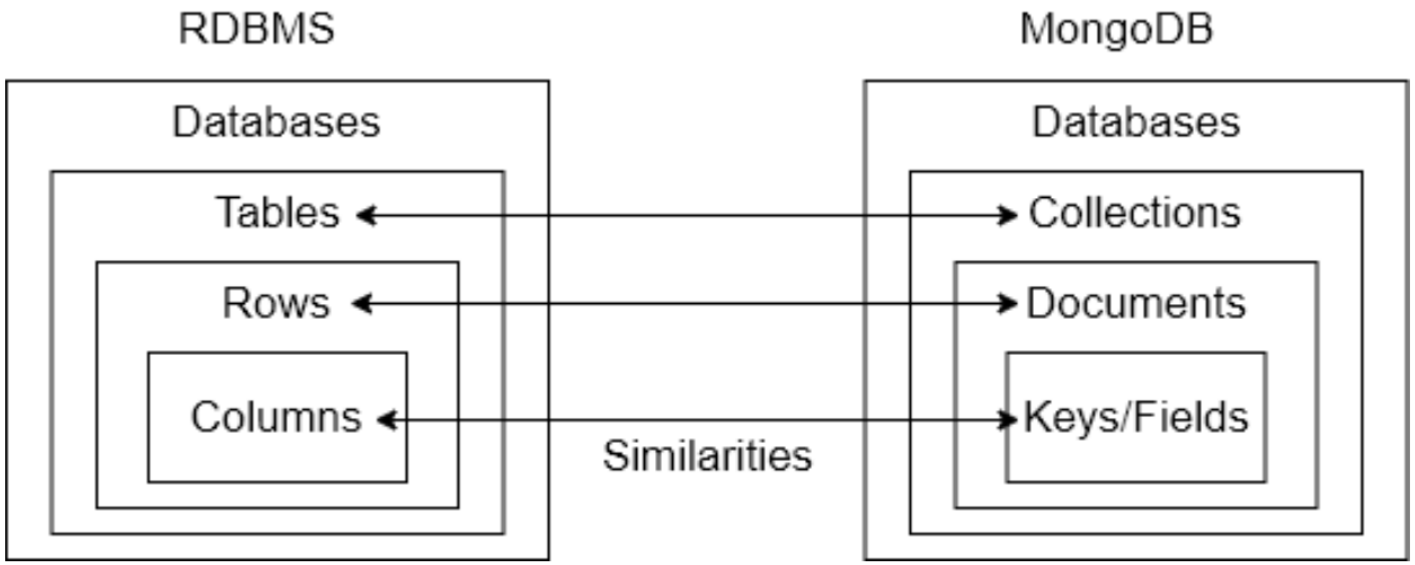
MongoDB tutorial

https://www.tutorialspoint.com/mongodb/index.htm

Mongodb - Overview

MongoDB stores data records as documents which are gathered together in collections. A database stores one or more collections of documents.

- Databases
- Collections
- Documents
- Datatypes



A Database contains collections, and a collection contains documents and the documents contain data, which are related to each other.

Restriction on database names

1. **Database Name Case Sensitivity**
   - Database names must differ on more than just case

     salesDB and **salesdb** names will clash for databases

2. **Restrictions on MongDB Database Names for Windows**
   - Database names cannot contain any of the following characters:

     /\. "$*<>:|?

3. **Restrictions on MongDB Database Names for Unix and Linux Systems**
   - Database names cannot contain any of the following characters:

     /\. "$

4. **Length of Database Names**
   - Database names cannot be empty and must have fewer than 64 characters.

Restriction on collection names

Collection names should begin with an underscore or a letter character, and cannot:

- contain the **$**
- be an **empty string** (e.g. "")
- contain the **null character**
- begin with the **system.** prefix (Reserved for internal use.)

Restriction on Field names

- Field names cannot contain the **null character**

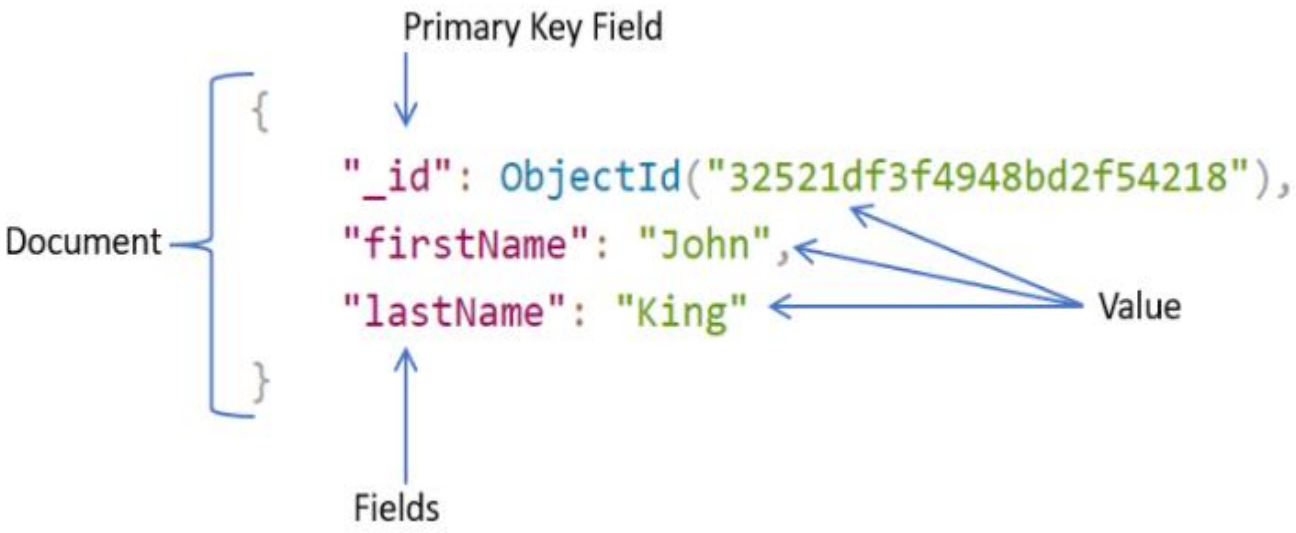
- The server permits storage of field names that contain **dots** (.) and **dollar** signs ($)

Restriction on _id names

- In MongoDB, each document stored in a collection requires a **unique _id field** that acts as a **primary key**. If an inserted document omits the _id field, the MongoDB driver automatically generates an **ObjectId** for the _id field.

- Its value must be **unique** in the collection.

- It is **immutable**, and may be of any type other than an array.



Databases

In MongoDB, databases hold one or more **collections of documents.**

- To show all the existing datbases, use the command:

  show dbs **OR** **show databases**

- To select a database to use, issue the command:

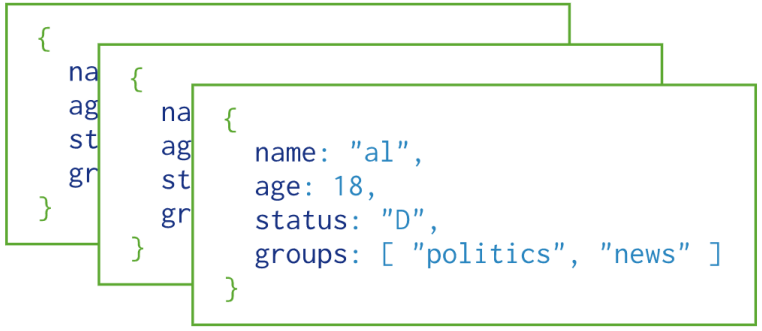
  use <database name>

- If a database does not exist, MongoDB creates the database when you first store data for that database.

Collections

It is a construct (container) that stores the documents

- **Dynamic schema** - documents can have different number of key-value pairs
- It is imperative that there be separate collections, to make querying, aggregation and indexing more efficient
- **It is similar to tables in a relational database**

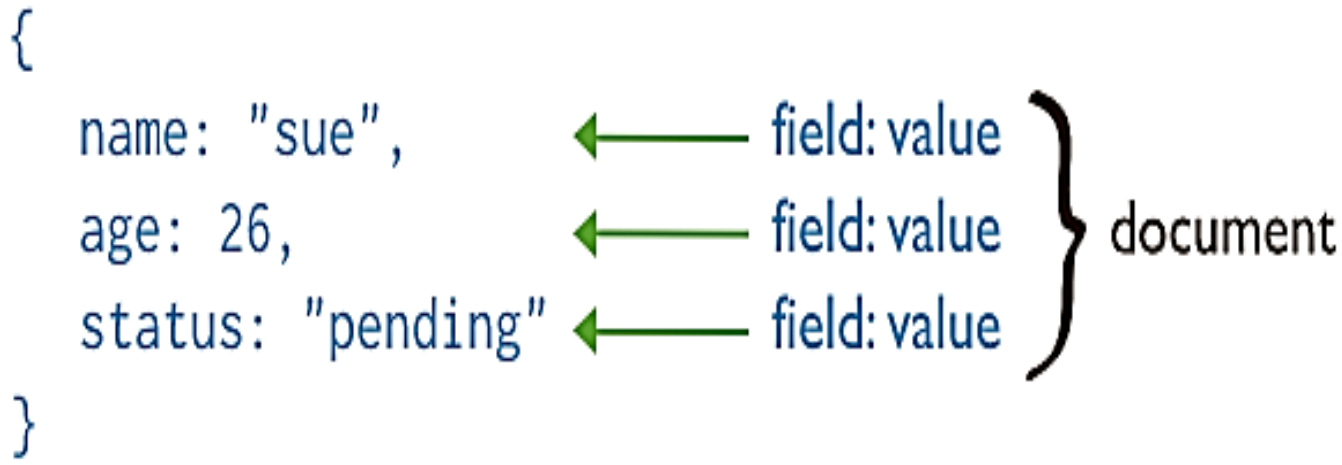


Collection

Documents

A document is a basic unit of data

- **Structure:** ordered set of **key-value pairs (**field-and-value pairs)
- **It is similar to a row in relational databases**
- The following example is a simple document:



Documents

The following example is a complex document:

```
{
  first_name: 'Paul',
  surname: 'Miller',
  cell: 447557505611,
  city: 'London',
  location: [45.123,47.232],
  Profession: ['banking', 'finance', 'trader'],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  ]
}
```

Fields

String

Number

Geo-Coordinates

Typed field values

Fields can contain arrays

Fields can contain an array of sub-documents

Datatypes

- **String** – This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer** – This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** – This type is used to store a boolean (true/ false) value.
- **Double** – This type is used to store floating point values.
- **Min/ Max keys** – This type is used to compare a value against the lowest and highest BSON (Binary Javascript Object Notation) elements.
- **Arrays** – This type is used to store arrays or list or multiple values into one key.
- **Timestamp** – ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object** – This datatype is used for embedded documents.

Datatypes

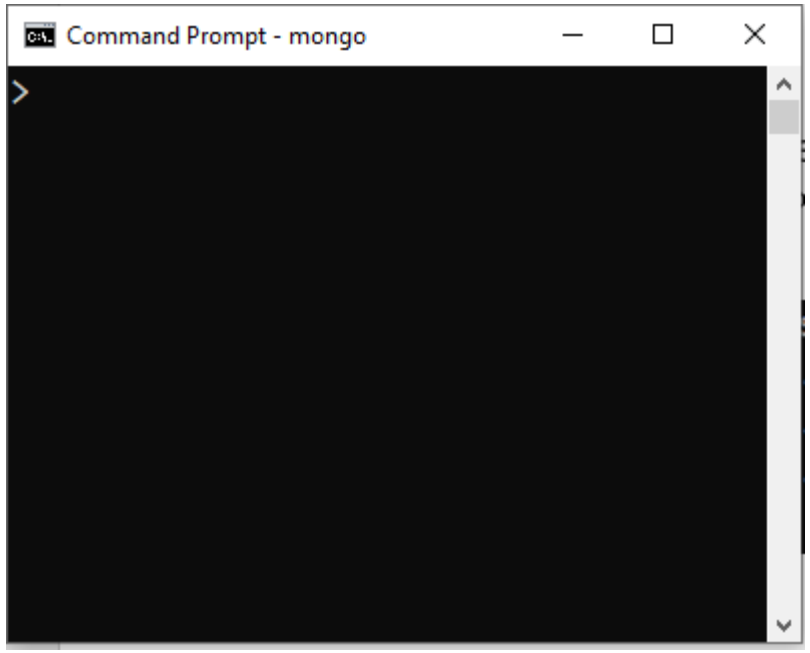
- **Null** – This type is used to store a Null value.
- **Symbol** – This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- **Date** – This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID** – This datatype is used to store the document’s ID.
- **Binary data** – This datatype is used to store binary data.
- **Code** – This datatype is used to store JavaScript code into the document.
- **Regular expression** – This datatype is used to store regular expression.

Mongodb – CRUD

CRUD: **Create Read Update and Delete – Mongo shell**

Mongo Shell

- The mongo shell is an interactive JavaScript interface to MongoDB.
- It is the command-line client of MongoDB.
- Mongo Shell connects to the MongoDB Server on the local host automatically.
- We can use the mongo shell to query and update data as well as perform administrative operations.



Database commands

- **Show existing databases**

  show databases OR show dbs

- **Accessing the existing database**

  use <database-name>

- **See the database we are currently working with**

  db

```
> show databases
admin    0.000GB
config   0.000GB
local    0.000GB
> use local
switched to db local
>
```

```
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
> use local
switched to db local
> db
local
>
```

Database commands

- **Creating a new database**

  use <new-database-name>

- ❑ If a database does not exist, MongoDB creates the database when you first store data for that database.
- ❑ To display database, we need to insert at least one document into it.

```
> use demoDB
switched to db demoDB
> db
demoDB
> show databases
admin    0.000GB
config   0.000GB
local    0.000GB
>
```

```
> use demoDB
switched to db demoDB
> db
demoDB
>
> show databases
admin    0.000GB
config   0.000GB
local    0.000GB
> db.my_collection1.insert({"name" : "John"})
WriteResult({ "nInserted" : 1 })
>
> show databases
admin    0.000GB
config   0.000GB
demoDB   0.000GB
local    0.000GB
>
```

Database commands

- **Dropping an existing database**

db.dropDatabase()

```
> use dummyDB
switched to db dummyDB
>
> db
dummyDB
>
> db.products.insert({"Item" : "Toy"})
WriteResult({ "nInserted" : 1 })
>
> show databases
admin     0.000GB
config    0.000GB
demoDB    0.000GB
dummyDB   0.000GB
local     0.000GB
>
> db.dropDatabase()
{ "ok" : 1 }
>
> show databases
admin    0.000GB
config   0.000GB
demoDB   0.000GB
local    0.000GB
>
```

Collection commands

- **Listing all collections in a database**

  show collections

```
> db
demoDB
> show collections
my_collection1
>
```

- **Inserting one document in a collection**

  db.my_collection1.insert({"name" : "Jerry"})

```
> use demoDB
switched to db demoDB
> db.my_collection1.insert({"name" : "Jerry"})
WriteResult({ "nInserted" : 1 })
>
```

Collection commands

- **Display all documents in a collection**

db.<collection_name>.find()

OR

db.<collection_name>.find().pretty()

```
> db
demoDB
>
> show collections
my_collection1
>
> db.my_collection1.find()
{ "_id" : ObjectId("620776c15799d2d256bb03bb"), "name" : "John" }
{ "_id" : ObjectId("62077a525799d2d256bb03bc"), "name" : "Jerry" }
>
```

Collection commands

- **Dropping a collection from a database**

db.<collection_name>.drop()

```
> use demoDB
switched to db demoDB
>
> show collections
my_collection1
my_collection2
>
> db.my_collection2.drop()
true
>
> show collections
my_collection1
>
```

Insert functions

Inserting a document directly into insert function as a parameter

```
> db
demoDB
>
> db.my_collection1.insert({"name" : "Eva"})
WriteResult({ "nInserted" : 1 })
>
> db.my_collection1.find()
{ "_id" : ObjectId("620776c15799d2d256bb03bb"), "name" : "John" }
{ "_id" : ObjectId("62077a525799d2d256bb03bc"), "name" : "Jerry" }
{ "_id" : ObjectId("6207838f5799d2d256bb03c2"), "name" : "Eva" }
>
```

Insert functions

Creating a document as a variable, then passing it as the insert function parameter

```
> var cust2 = {
... "cust_id":10,
... "name":"Bob",
... "street":"Moon lane",
... "city":"Los Angeles",
... "DOB":new Date("2012-09-25"),
... "phone": ["98777777", "98666666", "98555555"],
... "married":true
...   }
>
>
> db.my_collection1.insertOne(cust2)
{
        "acknowledged" : true,
        "insertedId" : ObjectId("62078d905799d2d256bb03c8")
}
>
```

Insert functions

Using find().pretty()

```
> db.my_collection1.find()
{ "_id" : ObjectId("620776c15799d2d256bb03bb"), "name" : "John" }
{ "_id" : ObjectId("62077a525799d2d256bb03bc"), "name" : "Jerry" }
{ "_id" : ObjectId("62078e445799d2d256bb03c9"), "cust_id" : 10, "name" : "Bob", "street" : "Moon lane", "city" : "Los
Angeles", "DOB" : ISODate("2012-09-25T00:00:00Z"), "phone" : [ "98777777", "98666666", "98555555" ], "married" : true
}
>
```

```
> db.my_collection1.find().pretty()
{ "_id" : ObjectId("620776c15799d2d256bb03bb"), "name" : "John" }
{ "_id" : ObjectId("62077a525799d2d256bb03bc"), "name" : "Jerry" }
{
        "_id" : ObjectId("62078e445799d2d256bb03c9"),
        "cust_id" : 10,
        "name" : "Bob",
        "street" : "Moon lane",
        "city" : "Los Angeles",
        "DOB" : ISODate("2012-09-25T00:00:00Z"),
        "phone" : [
                "98777777",
                "98666666",
                "98555555"
        ],
        "married" : true
}
>
```

Insert functions

Insert many documents into a collection: db.collection.insertMany()

- To insert more than one document
  1. Either group your documents in a list then assign it to a variable.
  2. Or directly type them into the function parameter as a list of comma separated documents.

- This function has an parameter name **ordered** which is used to define if the documents to be inserted are to be ordered or unordered. Its default value is **true**.

Insert functions

Insert many documents into a collection

```
> var custn=[
... {First_Name: "Sachin",
... Last_Name: "Sharma",
... Date_Of_Birth: "1995-09-26",
... e_mail: "radhika_sharma.123@gmail.com",
... phone: "9000012345"
... },
... {
... First_Name: "Rachel",
... Last_Name: "Christopher",
... Date_Of_Birth: "1990-02-16",
... e_mail: "Rachel_Christopher.123@gmail.com",
... phone: "9000054321"
... },
... {
... First_Name: "Fathima",
... Last_Name: "Sheik",
... Date_Of_Birth: "1990-02-16",
... e_mail: "Fathima_Sheik.123@gmail.com",
... phone: "9000054321"
... }
... ]
>
> db.my_collection1.insertMany(custn)
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("620792185799d2d256bb03ca"),
                ObjectId("620792185799d2d256bb03cb"),
                ObjectId("620792185799d2d256bb03cc")
        ]
}
```

Insert - summary

- The insert functions does little validation.
- It only checks for the basic document structure and then adds an object id (_id field) should there be none provided by the application.
- One of the basic document structure checks is that all documents size must be less than 16MB.
- It is also fairly easy to insert invalid data therefore only allow trusted sources such as applications to perform insert operations to the databases.

Delete functions

Deleting or removing documents, collections or databases are done via the **remove** or **drop** functions along with a **delete filter** to target the documents, collections or databases.

1. **db.collection.remove(delete filter**, 1**)** - removes the first instance of the document that matches the delete filter from the collection.

db.my_collection1.remove({"name" : "John"}, 1)

```
> db.my_collection1.find()
{ "_id" : ObjectId("62079a245799d2d256bb03de"), "name" : "John", "age" : 40 }
{ "_id" : ObjectId("62079a3f5799d2d256bb03df"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("62079a4d5799d2d256bb03e0"), "name" : "Eva", "age" : 36 }
{ "_id" : ObjectId("62079a535799d2d256bb03e1"), "name" : "John", "age" : 42 }
>
```

```
> db.my_collection1.remove({"name" : "John"}, 1)
WriteResult({ "nRemoved" : 1 })
>
> db.my_collection1.find()
{ "_id" : ObjectId("62079a3f5799d2d256bb03df"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("62079a4d5799d2d256bb03e0"), "name" : "Eva", "age" : 36 }
{ "_id" : ObjectId("62079a535799d2d256bb03e1"), "name" : "John", "age" : 42 }
>
```

Delete functions

2. db.collection.remove(delete filter) - removes all the instances of the document that matches the delete filter from the collection.

db.my_collection1.remove({"name" : "John"})

```
> db.my_collection1.find()
{ "_id" : ObjectId("62079a3f5799d2d256bb03df"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("62079a4d5799d2d256bb03e0"), "name" : "Eva", "age" : 36 }
{ "_id" : ObjectId("62079a535799d2d256bb03e1"), "name" : "John", "age" : 42 }
>
```

```
> db.my_collection1.remove({"name" : "John"})
WriteResult({ "nRemoved" : 2 })
>
> db.my_collection1.find()
{ "_id" : ObjectId("62079a4d5799d2d256bb03e0"), "name" : "Eva", "age" : 36 }
>
```

Delete functions

3. db.collection.remove({}) - removes all the documents from the collection.

db.my_collection1.remove({ })

```
> db.my_collection1.find()
{ "_id" : ObjectId("62079a3f5799d2d256bb03df"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("62079a4d5799d2d256bb03e0"), "name" : "Eva", "age" : 36 }
{ "_id" : ObjectId("62079a535799d2d256bb03e1"), "name" : "John", "age" : 42 }
>
```

```
> db.my_collection1.remove({})
WriteResult({ "nRemoved" : 3 })
>
> db.my_collection1.find()
>
```

Delete functions

- **db.<collection_name>.drop()** - delete the specified collection.

```
db.my_collection1.drop()
```

- **db.dropDatabase()** - delete the current database.

Update functions

db.<collection>.update(SELECTION CRITERIA, UPDATED DATA)

updates a single document in the collection

```
> db.my_collection1.find()
{ "_id" : ObjectId("620866e33dfcf7777cc9d5da"), "name" : "John", "age" : 40 }
{ "_id" : ObjectId("620866eb3dfcf7777cc9d5db"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("620866f83dfcf7777cc9d5dc"), "name" : "Eva", "age" : 39 }
{ "_id" : ObjectId("620867193dfcf7777cc9d5dd"), "name" : "John", "age" : 42 }
>
```

```
> db.my_collection1.update({"name":'John', "age":40},{$set:{"age":50}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.my_collection1.find()
{ "_id" : ObjectId("620866e33dfcf7777cc9d5da"), "name" : "John", "age" : 50 }
{ "_id" : ObjectId("620866eb3dfcf7777cc9d5db"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("620866f83dfcf7777cc9d5dc"), "name" : "Eva", "age" : 39 }
{ "_id" : ObjectId("620867193dfcf7777cc9d5dd"), "name" : "John", "age" : 42 }
>
```

Update functions

db.<collection>.update(SELECTION CRITERIA, UPDATED DATA, {multi : true})

updates multiple documents in the collection

```
> db.my_collection1.find()
{ "_id" : ObjectId("620866e33dfcf7777cc9d5da"), "name" : "John", "age" : 51 }
{ "_id" : ObjectId("620866eb3dfcf7777cc9d5db"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("620866f83dfcf7777cc9d5dc"), "name" : "Eva", "age" : 39 }
{ "_id" : ObjectId("620867193dfcf7777cc9d5dd"), "name" : "John", "age" : 42 }
```

```
> db.my_collection1.update({"name":'John'},{$set:{"age":80}}, {multi:true})
WriteResult({ "nMatched" : 3, "nUpserted" : 0, "nModified" : 2 })
>
>
> db.my_collection1.find()
{ "_id" : ObjectId("620866e33dfcf7777cc9d5da"), "name" : "John", "age" : 80 }
{ "_id" : ObjectId("620866eb3dfcf7777cc9d5db"), "name" : "John", "age" : 80 }
{ "_id" : ObjectId("620866f83dfcf7777cc9d5dc"), "name" : "Eva", "age" : 39 }
{ "_id" : ObjectId("620867193dfcf7777cc9d5dd"), "name" : "John", "age" : 80 }
>
```

Update functions

db.<collection>.replaceOne(filter, replacement, options)

replaces a single document in the collection

```
> db.my_collection1.find()
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e4"), "name" : "Eva", "age" : 46 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e5"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e6"), "name" : "Jessica", "age" : 48 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e7"), "name" : "John", "age" : 42 }
>
```

```
> db.my_collection1.replaceOne({ "name" : "Eva"}, { "name" : "Eden", "age" : "55" })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
> db.my_collection1.find()
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e4"), "name" : "Eden", "age" : "55" }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e5"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e6"), "name" : "Jessica", "age" : 48 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e7"), "name" : "John", "age" : 42 }
>
```

Update operators

- **$set** - sets value of the key or add the key-value pair if the pair is not in the document
- **$inc** - increments the value of the key by the specified amount
- **$mul** - multiplies the value of the key by the specified amount
- **$rename** - rename the key
- **$unset** - removes the key from the document

- Click here for more operators.

Update operator - $inc

```
> db.my_collection1.find()
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e4"), "name" : "Eden", "age" : 55 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e5"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e6"), "name" : "Jessica", "age" : 48 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e7"), "name" : "John", "age" : 42 }
>
```

```
> db.my_collection1.updateOne({"name" : "Eden"},{$inc : {"age" : 5}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
>
>
> db.my_collection1.find()
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e4"), "name" : "Eden", "age" : 60 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e5"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e6"), "name" : "Jessica", "age" : 48 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e7"), "name" : "John", "age" : 42 }
>
```

Array – Adding items ($push)

```
> db.my_collection1.find()
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e4"), "name" : "Eden", "age" : "60", "score" : [ 50, 60, 70 ] }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e5"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e6"), "name" : "Jessica", "age" : 48 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e7"), "name" : "John", "age" : 42 }
>
```

```
> db.my_collection1.updateOne({"name" : "Eden"} , { $push: {score:{$each:[10,20,30], $position:0}} })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
>
> db.my_collection1.find()
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e4"), "name" : "Eden", "age" : "60", "score" : [ 10, 20, 30, 50, 60, 70 ] }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e5"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e6"), "name" : "Jessica", "age" : 48 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e7"), "name" : "John", "age" : 42 }
>
```

Inserts the elements **[10,20,30]** in the array **scores** at position **0**

Array – Removing items ($pull)

```
> db.my_collection1.find()
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e4"), "name" : "Eden", "age" : "60", "score" : [ 10, 20, 30, 50, 60, 70 ] }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e5"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e6"), "name" : "Jessica", "age" : 48 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e7"), "name" : "John", "age" : 42 }
>
>
```

```
> db.my_collection1.update({ "name": "Eden" }, { $pull: { "score" : 50 }})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
>
> db.my_collection1.find()
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e4"), "name" : "Eden", "age" : "60", "score" : [ 10, 20, 30, 60, 70 ] }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e5"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e6"), "name" : "Jessica", "age" : 48 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e7"), "name" : "John", "age" : 42 }
>
```

Removes item **50** from the array **score**

Query functions

The query functions consists of only 2 functions, but it has a rich set of operators.

- **db.collection.findOne()** - returns the first occurrence of a single document matching the query criteria.
- **db.collection.find()** - returns a cursor object that points to all the documents that matches the query criteria.

```
db.collection.find(          ← Collection
  { age: { $gt: 40 } },      ← query criteria
  { name : 1, sex : 1 }      ← projection
).limit(2)                   ← cursor modifier
```

Projection parameter

The projection document accepts the key names and a Boolean value.

- Projection cannot have a mix of inclusion and exclusion values.
- That means all the keys-value pairs in the projection document has to consist fully of either keys that are to be returned (inclusion) or keys that we do not want (exclusion)
- The **_id** key is **exempted** from this rule.

```
# invalid, means we want '_id' and not 'MatrNum'
# change to {"MatrNum":1} to be valid
{"_id":1, "MatrNum":0}

# valid, means we want 'MatrNum' and 'name' and not '_id'
{"_id":0, "MatrNum":1, "name":1}
```

Query function & Projection parameter - example

```
> db.my_collection1.find()
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e4"), "name" : "Eden", "age" : 60, "score" : [ 50, 60, 70 ] }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e5"), "name" : "John", "age" : 41 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e6"), "name" : "Jessica", "age" : 48 }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e7"), "name" : "John", "age" : 42 }
>
```

```
> db.my_collection1.find( {age:{$gt:41}}, {name:1})
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e4"), "name" : "Eden" }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e6"), "name" : "Jessica" }
{ "_id" : ObjectId("62087d9d3dfcf7777cc9d5e7"), "name" : "John" }
>
```

```
> db.my_collection1.find( {age:{$gt:41}}, {"_id" : 0, "name" : 1}).limit(2)
{ "name" : "Eden" }
{ "name" : "Jessica" }
>
```

Query selectors - $in

- $in - a comparison operator that selects documents based on if the value in the key matches the value in the specified array.
- In the example below, the $in operator is used to find all the documents that have either a 5 or 15 value for the qty key

```
# document
{ _id: 1, item: "abc", qty: 15, tags: [ "school", "clothing" ], sale:
false }

db.collection.find( { qty: { $in: [ 5, 15 ] } } )
```

- Click here for more selectors.

Query selectors – Logical operators

- Logical operators - **$and** , **$or** , **$nor** and **$not**
- They are used to join multiple query conditions

```
db.collection.find( {
    $and: [
        { $or: [ { qty: { $lt : 10 } }, { qty : { $gt: 50 } } ] },
        { $or: [ { sale: true }, { price : { $lt : 5 } } ] }
    ]
} )
```

select all documents where the qty key has a value less than 10 or greater than 50 and is on sale or price is less than 5 .

Query selectors – $type

- Values can have different datatypes despite having the same key.

  Zipcode (integer, long, string)

- $type - querying by the datatype of the value in the key-value pairs

```
db.collection.find( { "zipCode" : { $type : "number" } } );
```