DATA ANALYTICS REFERENCE DOCUMENT	
Document Title:	Document Title
Document No.:	1552560766
Author(s):	Gerhard van der Linde, Rita Raher
Contributor(s):	

REVISION HISTORY

Revision	Details of Modification(s)	Reason for modification	Date	By
0	Draft release	Document description here	2019/03/14 10:52	Gerhard van der Linde, Rita Raher

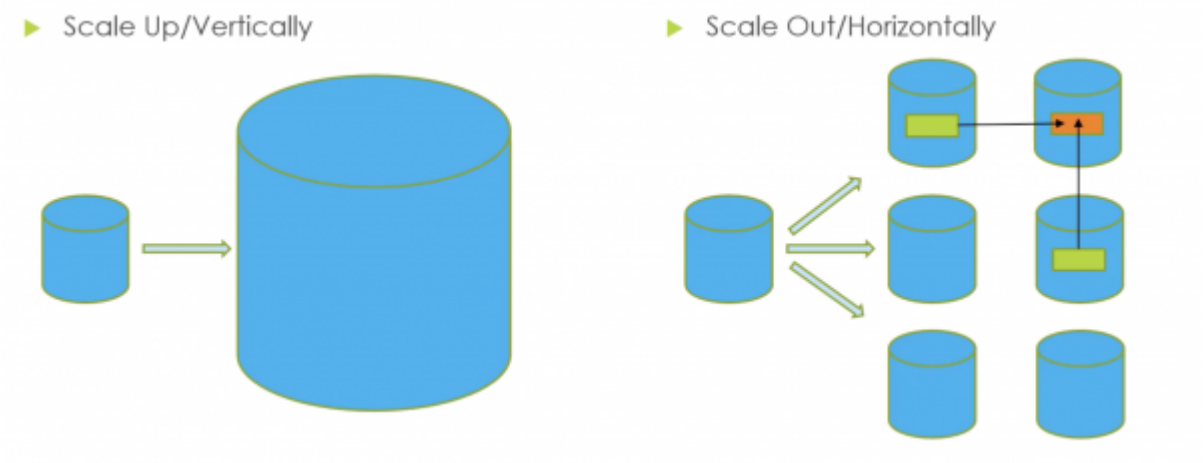
Topic 6 - MongoDB I

Why NoSQL Databases?

Scalability

Scale Up/Vertically: means moving the database to a bigger server.

Scale Out/Horizontally



Unstructured Data

- CustomerID INTEGER
- Name VARCHAR(20)
- Phone VARCHAR(20)
- Address VARCHAR(50)
- Email VARCHAR(50)
- Twitter VARCHAR(50)

CustomerID*	Name	Phone	Address	Email	Twitter
100	John	086 3304896	Tuam, Co. Galway	John@gmail.com	@John123
101	Alan	NULL	Athenry, Co. Galway	NULL	NULL
102	Mary	091 5688874	Galway, Co. Galway	Mary@yahoo.com	NULL
103	Tom	090 6458959	Athlone, Co. Westmeath	NULL	NULL
104	Alice	094 1245763	Castlebar, Co. Mayo	NULL	@AliceC1965

Add on new features later on like email and then twitter etc...

MongoDB

- Document Database
- Schemaless
- Horizontal Scalability Through sharding[1)]
- Duplication of data

JSON

- JSON - JavaScript Object Notation
- Lightweight data-interchange format
- Machine/Human readable
- Language independent
- JSON Structure
  - Name/Value pair
  - Ordered Lists

JSON Datatypes

Number

```
{
  "id" : 1
}
```

```
{
  "id" : 3.14
}
```

Note that there is no distinction between integer and floating point numbers.

String

```
{
  "id" : 1,
  "fname" : "John"
}
```

Boolean

```
{
  "reg" : "09-G-13"
  "hybrid" : false
}
```

Array

```
{
  "student" : "G00257854"
  "subjects" : ["Databases", "Java", "Mobile Apps"]
}
```

Object Document

```
{
  "student" : "G00257854"
  "address" : {
     "street" : "Castle Street"
     "town" : "Athenry"
     "county" : "Galway"
  }
}
```

JSON USES

```
{
  "type": "FeatureCollection",
  "totalFeatures": 28,
  "features": [
    {
      "type": "Feature",
      "id": "MINES_SiteLocation.fid-48245d43_1693a57810c_6682",
      "geometry": {
        "type": "MultiPoint",
        "coordinates": [
          [
            -6.78797543,
            54.15895923
          ]
        ]
      },
      "geometry_name": "Shape",
      "properties": {
        "Name": "Monaghan Pb",
        "Code": "MON",
        "SiteLocation": "Monaghan",
        "X_Centroid": 279389,
        "Y_Centroid": 323403,
        "CommodityProduced": "Pb(Zn-Ba-Ag),Sb",
        "URL": "No report available",
        "URLtext": "Link to More Information",
        "Description": "The Monaghan lead mines are made up of thı
      }
    },
    {
      "type": "Feature",
      "id": "MINES_SiteLocation.fid-48245d43_1693a57810c_6683",
      "geometry": {
```


https://data.gov.ie/dataset/mines-site-district/resource/8920e026-a3e7-4987-a9fe-

```
{
    status: "ok",
    totalResults: 10,
    articles: [
        {
            source: {
                id: "national-geographic",
                name: "National Geographic"
            },
            author: "Sarah Gibbens, Laura Parker",
            title: "Creatures in the deepest trenches of the sea are ea
            description: "In six of the ocean's deepest crevasses, scie
            shrimp-like creatures chomping on tiny bits of plastic.",
            url: https://www.nationalgeographic.com/environment/2019/02
            mariana-trench-eat-plastic.html,
            urlToImage:
            https://www.nationalgeographic.com/content/dam/environment/
            ,
            publishedAt: "2019-03-01T09:37:54.1275978Z",
            content: null
        },
        {
            source: {
                id: "national-geographic",
                name: "National Geographic"
            },
            author: "National Geographic Staff",
            title: "See the top 10 pictures entered in our Instagram co
            description: "Instagram users submitted more than 94,000 ph
```


https://newsapi.org/s/national-geographic-api

MongoDB, JSON and BSON

- JSON object = MongoDB document
- Internally, MongoDB represents JSON documents in binary-encoded format called BSON (Binary JavaScript Object Notation)
- BSON extends JSOM model to provide additional data types as well as indexes.

MongoDB Structures

Document  - slide 12....

A document is record in a MongoDB collection and the basic unit of data in MongoDB. Documents are analogous to JSON objects or records in an RDBMS.

```
{
    "_id"    : ObjectId("5919fecf0822ef8ecec132f8"),
    "name"   : "John",
    "house"  : 31,
    "street" : "Main St.",
    "town"   : "Athenry"
}
```

Collection

- A grouping of MongoDB documents.
- Collections are analogous to RDBMS tables.
- A collection exists within a single database.
- Collections do not enforce a schema. Documents within a collection can have different fields.
- Typically, all documents in a collection have a similar or related purpose.

```
{
    "_id"    : ObjectId("5919fecf0822ef8ecec132f8"),
    "name"   : "John",
    "house"  : 31,
    "street" : "Main St.",
    "town"   : "Athenry",
    "county" : "Galway"
}

{
    "_id"      : ObjectId("591a000f0822ef8ecec132f9"),
    "name"     : "Alan",
    "townland" : "Litirmor",
    "town"     : "Ballymurphy",
    "county"   : "Cork"
}
```

Database A number of databases can be run on a single MongoDB server.

MongoDB Commands

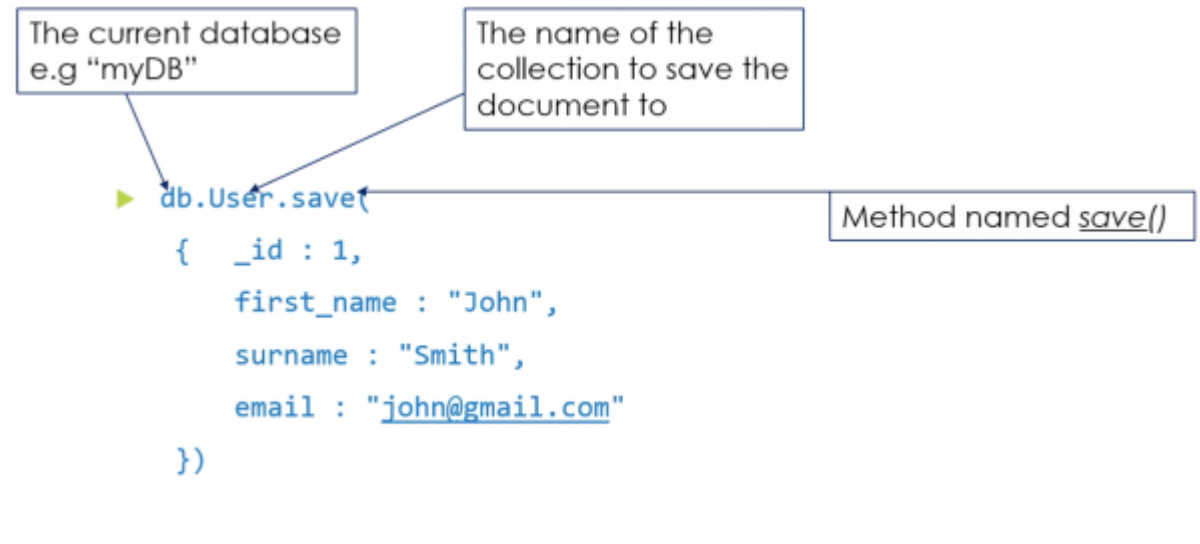
- show dbs - Show Databases
- use myDB - Switch to databases named “myDB” (If it doesn't exist, Mongo creates it)
- db - Show current Database.
- show collections - Show collections in the current database

MongoDB Rules for creating a Document

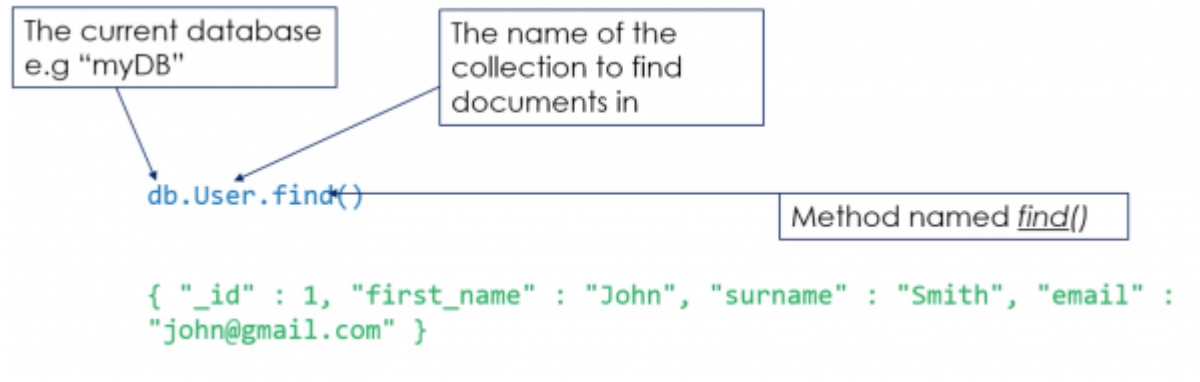
- Rules for MongoDB documents
  - A document must have an _id field. if one is not provided, it will be automatically generated

- The _id cannot be an array

Create a document - save()



Query the database - find()



pretty()

```
db.User.find().pretty()

{
        "_id" : 1,
        "first_name" : "John",
        "surname" : "Smith",
        "email" : "john@gmail.com"
}
```

```
db.User.find()
{ "_id" : 1, "first_name" : "John", "surname" : "Smith", "age" : 22, "email" : "john@gmail.com" }
{ "_id" : 2, "first_name" : "Sean", "surname" : "Williams", "age" : 30, "email" : "williamss@gmail.com" }
{ "_id" : 3, "first_name" : "Albert", "surname" : "O'Hara", "age" : 27, "email" : "al@hotmail.com", "twitter" : "@al1234" }
{ "_id" : 4, "first_name" : "Mary", "surname" : "Collins", "age" : 22 }
{ "_id" : 5, "first_name" : "Susan", "surname" : "Hanly", "age" : 18, "email" : "susie@hotmail.com", "twitter" : "@Susie2u" }
```

To find only documents where *age* is 22:

```
db.User.find({age:22})
```

```
{ "_id" : 1, "first_name" : "John", "surname" : "Smith", "age" : 22, "email" : "john@gmail.com" }
{ "_id" : 4, "first_name" : "Mary", "surname" : "Collins", "age" : 22 }
```

$and

To find only documents where *age* is 22 and *_id* is 1:

```
db.User.find({age:22, _id:1})
db.User.find({$and: [{age:22}, {_id:1}]})
```

```
{ "_id" : 1, "first_name" : "John", "surname" : "Smith", "age" : 22, "email" : "john@gmail.com" }
```

$or

To find only documents where *age* is 22 or 30

```
db.User.find({$or: [{age:22}, {age:30}]})
```

```
{ "_id" : 1, "first_name" : "John", "surname" : "Smith", "age" : 22, "email" : "john@gmail.com" }
{ "_id" : 2, "first_name" : "Sean", "surname" : "Williams", "age" : 30, "email" :
"williamss@gmail.com" }
{ "_id" : 4, "first_name" : "Mary", "surname" : "Collins", "age" : 22 }
```

$in

To find only documents where *age* is 22 or 30

```
db.User.find({age: {$in: [22, 30]}})
```

```
{ "_id" : 1, "first_name" : "John", "surname" : "Smith", "age" : 22, "email" : "john@gmail.com" }
{ "_id" : 2, "first_name" : "Sean", "surname" : "Williams", "age" : 30, "email" :
"williamss@gmail.com" }
{ "_id" : 4, "first_name" : "Mary", "surname" : "Collins", "age" : 22 }
```

Attribute

To find only documents that have a *twitter* attribute

```
db.User.find({twitter: {$exists:true}})
```

```
{ "_id" : 3, "first_name" : "Albert", "surname" : "O'Hara", "age" : 27, "email" : "al@hotmail.com",
"twitter" : "@al1234" }
{ "_id" : 5, "first_name" : "Susan", "surname" : "Hanly", "age" : 18, "email" :
"susie@hotmail.com", "twitter" : "@Susie2u" }
```

Attribute and age is greater than 20

To find only documents that have a *twitter* attribute and *age* is greater than 20

```
db.User.find({$and: [{twitter: {$exists: true}}, {age: {$gt: 20}}]})
```

```
{ "_id" : 3, "first_name" : "Albert", "surname" : "O'Hara", "age" : 27, "email" : "al@hotmail.com",
"twitter" : "@al1234" }
```

findOne()

To find only one document that has a *twitter* attribute

```
db.User.findOne({twitter: {$exists: true}})
```

```
{ "_id" : 3, "first_name" : "Albert", "surname" : "O'Hara", "age" : 27, "email" : "al@hotmail.com",
"twitter" : "@al1234" }
```

sort()

To sort all documents that have a *twitter* attribute alphabetically by *surname* and within *surname*, from oldest to youngest

```
db.User
.find({twitter: {$exists: true}}).sort({surname:1 , age:-1})
{ "_id" : 5, "first_name" : "Susan", "surname" : "Hanly", "age" : 18, "email" :
"susie@hotmail.com", "twitter" : "@Susie2u" }
{ "_id" : 3, "first_name" : "Albert", "surname" : "O'Hara", "age" : 27, "email" : "al@hotmail.com",
"twitter" : "@al1234" }
```

MongoDB -_id

- As previously described, the document ID (_id) attribute of a mongoDB document is the only mandatory part of a document.
- It can be any value, except an array.

```
db.Test.save({_id: 1, name: "John"})

db.Test.find({name: "John"})

{ "_id" : 1, "name" : "John" }
```

```
db.Test.save({name: "Billy"})

db.Test.find({name: "Billy"})

{ "_id" : ObjectId("591acc5612b8754878ffac81"),
  "name" : "Billy" }
```

more on save()

```
db.mydoc.find()
{ "_id" : 1, "name" : "Tom", "age" : 37 }
{ "_id" : 2, "name" : "Bill", "age" : 44 }
{ "_id" : 3, "name" : "Mary", "age" : 29 }
{ "_id" : 4, "name" : "Susan", "age" : 37 }

db.mydoc.save({_id:1, name:"John", age:24})

db.mydoc.find()

{ "_id" : 1, "name" : "John", "age" : 24 }
{ "_id" : 2, "name" : "Bill", "age" : 44 }
{ "_id" : 3, "name" : "Mary", "age" : 29 }
{ "_id" : 4, "name" : "Susan", "age" : 37 }
```

insert()

- Insert a document or documents into a collection.

```
db.mydoc.find()
{ "_id" : 1, "name" : "Tom", "age" : 37 }
{ "_id" : 2, "name" : "Bill", "age" : 44 }
{ "_id" : 3, "name" : "Mary", "age" : 29 }
{ "_id" : 4, "name" : "Susan", "age" : 37 }
db.mydoc.insert({_id: 1, name: "John", age: 24})
db.mydoc.find()

{ "_id" : 1, "name" : "Tom", "age" : 37 }
{ "_id" : 2, "name" : "Bill", "age" : 44 }
{ "_id" : 3, "name" : "Mary", "age" : 29 }
{ "_id" : 4, "name" : "Susan", "age" : 37 }
```

```
db.mydoc.find()
{ "_id" : 1, "name" : "Tom", "age" : 37 }
{ "_id" : 2, "name" : "Bill", "age" : 44 }
{ "_id" : 3, "name" : "Mary", "age" : 29 }
{ "_id" : 4, "name" : "Susan", "age" : 37 }
db.mydoc.insert([{_id: 5, name: "Sean", age: 54},{_id:6, name: "Luke"}])

db.mydoc.find()
{ "_id" : 1, "name" : "Tom", "age" : 37 }
{ "_id" : 2, "name" : "Bill", "age" : 44 }
{ "_id" : 3, "name" : "Mary", "age" : 29 }
{ "_id" : 4, "name" : "Susan", "age" : 37 }
{ "_id" : 5, "name" : "Sean", "age" : 54 }
{ "_id" : 6, "name" : "Luke" }
```

update()

- Modifies an existing document or docents in a collection
- Update (query, update, options)[2)]

Does not update Mary

```
db.mydoc.find()
{ "_id" : 1, "name" : "Tom", "age" : 37 }
{ "_id" : 2, "name" : "Bill", "age" : 44 }
{ "_id" : 3, "name" : "Mary", "age" : 29 }
{ "_id" : 4, "name" : "Susan", "age" : 37 }
db.mydoc.update( {$or: [{name:"Tom"}, {name:"Mary"}]}, {address: "Galway"})

db.mydoc.find()
{ "_id" : 1, "address" : "Galway" }
{ "_id" : 2, "name" : "Bill", "age" : 44 }
{ "_id" : 3, "name" : "Mary", "age" : 29 }
{ "_id" : 4, "name" : "Susan", "age" : 37 }
```

$set

```
db.mydoc.find()
{ "_id" : 1, "name" : "Tom", "age" : 37 }
{ "_id" : 2, "name" : "Bill", "age" : 44 }
{ "_id" : 3, "name" : "Mary", "age" : 29 }
{ "_id" : 4, "name" : "Susan", "age" : 37 }
```

```
db.mydoc.update( {$or: [{name:"Tom"}, {name:"Mary"}]}, {$set:{address: "Galway"}},  {multi:true})
```

```
db.mydoc.find()
{ "_id" : 1, "name" : "Tom", "age" : 37, "address" : "Galway" }
{ "_id" : 2, "name" : "Bill", "age" : 44 }
{ "_id" : 3, "name" : "Mary", "age" : 29, "address" : "Galway" }
{ "_id" : 4, "name" : "Susan", "age" : 37 }
```

```
db.emp.find()
{ "_id" : 1, "name" : "Tom", "experience" : 17 }
{ "_id" : 2, "name" : "Bill", "experience" : 3 }
{ "_id" : 3, "name" : "Mary", "experience" : 13 }
{ "_id" : 4, "name" : "Susan", "experience" : 5 }
```

```
db.mydoc.update( {experience: {$gt:20}}, {$set: {title:"Manager"}},  {multi:true, upsert:true})
```

```
db.mydoc.find()
{ "_id" : 1, "name" : "Tom", "experience" : 17 }
{ "_id" : 2, "name" : "Bill", "experience" : 3 }
{ "_id" : 3, "name" : "Mary", "experience" : 13 }
{ "_id" : 4, "name" : "Susan", "experience" : 5 }
{ "_id" : ObjectId("5c7bbf654be40b2777d5c006"), "title" : "Manager" }
```

deleteOne()

- Removes a single document from a collection

```
db.mydoc.find()
{ "_id" : 1, "name" : "Tom", "age" : 37 }
{ "_id" : 2, "name" : "Bill", "age" : 44 }
{ "_id" : 3, "name" : "Mary", "age" : 29 }
{ "_id" : 4, "name" : "Susan", "age" : 37 }
```

```
db.mydoc.deleteOne({age:{$lt:40}})
```

```
db.mydoc.find()
{ "_id" : 2, "name" : "Bill", "age" : 44 }
{ "_id" : 3, "name" : "Mary", "age" : 29 }
{ "_id" : 4, "name" : "Susan", "age" : 37 }
```

deleteMany()

```
db.mydoc.find()
{ "_id" : 1, "name" : "Tom", "age" : 37 }
{ "_id" : 2, "name" : "Bill", "age" : 44 }
{ "_id" : 3, "name" : "Mary", "age" : 29 }
{ "_id" : 4, "name" : "Susan", "age" : 37 }

db.mydoc.deleteMany({age:{$lt:40}})

db.mydoc.find()
{ "_id" : 2, "name" : "Bill", "age" : 44 }
```

Operators

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

Update Operators

https://docs.mongodb.com/manual/reference/operator/update/

Logical Query Operators

https://docs.mongodb.com/manual/reference/operator/query-logical/

Comparison Query Operators

https://docs.mongodb.com/manual/reference/operator/query-comparison/

Topic 7 - MongoDB II

More on find()

```
db.user.find()
```

```
{ "_id" : 1, "first_name" : "John", "surname" : "Smith", "age" : 22, "email" : "john@gmail.com" }
{ "_id" : 2, "first_name" : "Sean", "surname" : "Williams", "age" : 30, "email" : "williamss@gmail.com" }
{ "_id" : 3, "first_name" : "Albert", "surname" : "O'Hara", "age" : 27, "email" : "al@hotmail.com", "twitter" : "@al1234" }
{ "_id" : 4, "first_name" : "Mary", "surname" : "Collins", "age" : 22 }
{ "_id" : 5, "first_name" : "Susan", "surname" : "Hanly", "age" : 18, "email" : "susie@hotmail.com", "twitter" : "@Susie2u" }
```

To find only documents that have an email attribute and age is greater than 20

```
db.user.find({$and:[{email: {$exists:true}}, {age:{$gt:20}}]})
```

```
{ "_id" : 1, "first_name" : "John", "surname" : "Smith", "age" : 22, "email" : "john@gmail.com" }
{ "_id" : 2, "first_name" : "Sean", "surname" : "Williams", "age" : 30, "email" :
"williamss@gmail.com" }
{ "_id" : 3, "first_name" : "Albert", "surname" : "O'Hara", "age" : 27, "email" : "al@hotmail.com",
"twitter" : "@al1234" }
```

find(query, projection)

```
db.User.find()
```

```
{ "_id" : 1, "first_name" : "John", "surname" : "Smith", "age" : 22, "email" : "john@gmail.com" }
{ "_id" : 2, "first_name" : "Sean", "surname" : "Williams", "age" : 30, "email" : "williamss@gmail.com" }
{ "_id" : 3, "first_name" : "Albert", "surname" : "O'Hara", "age" : 27, "email" : "al@hotmail.com", "twitter" : "@al1234" }
{ "_id" : 4, "first_name" : "Mary", "surname" : "Collins", "age" : 22 }
{ "_id" : 5, "first_name" : "Susan", "surname" : "Hanly", "age" : 18, "email" : "susie@hotmail.com", "twitter" : "@Susie2u" }
```

Return only the email attribute of documents where age is greater than 18

```
db.User.find({age: {$gt: 20}}, {email:1})
```

```
{ "_id" : 1, "email" : "john@gmail.com" }
{ "_id" : 2, "email" : "williamss@gmail.com" }
{ "_id" : 3, "email" : "al@hotmail.com" }
{ "_id" : 4 }
```

Return only the first_name and surname attributes of all documents

```
db.User.find({}, {_id:false, first_name:1, surname:1})
```

```
{ "first_name" : "John", "surname" : "Smith" }
{ "first_name" : "Sean", "surname" : "Williams" }
{ "first_name" : "Albert", "surname" : "O'Hara" }
{ "first_name" : "Mary", "surname" : "Collins" }
{ "first name" : "Susan", "surname" : "Hanly" }
```

aggregate()

- Calculates aggregate values for the data in a collection
- db.collection.aggregate(pipeline, options)
  - pipeline stages
  - pipeline Operators

Example

```
{ "_id" : 1, "name" : "John", "age" : 23, "gpa" : 4.5, "sex" : "M" }
{ "_id" : 2, "name" : "Tom", "age" : 22, "gpa" : 3.5, "sex" : "M" }
{ "_id" : 3, "name" : "Mary", "age" : 24, "gpa" : 3.5, "sex" : "F" }
{ "_id" : 4, "name" : "Sarah", "age" : 22, "gpa" : 4, "sex" : "F" }
{ "_id" : 5, "name" : "Bill", "age" : 23, "gpa" : 3, "sex" : "M" }
```

Get the average gpa for all students

```
db.users.aggregate([{$group:{_id:null, Average{$avg:"$gpa"}}}])
```

$group same as Group by in MYSQL

Result:

```
{ "_id" : null, "Average" : 3.7 }
```

Get the Maximum GPA per age group

```
db.march8.aggregate([{$group:{_id:"$age", "Max GPA per Age":{$max:"$gpa"}}}])
```

```
{ "_id" : 24, "Max GPA per Age" : 3.5 }
{ "_id" : 22, "Max GPA per Age" : 4 }
{ "_id" : 23, "Max GPA per Age" : 4.5 }
```

To sort: $sort

```
db.march8.aggregate([{$group:{_id:"$age", "Max GPA per Age":{$max:"$gpa"}}}, {$sort:{_id:1}}])
```

```
{ "_id" : 22, "Max GPA per Age" : 4 }
{ "_id" : 23, "Max GPA per Age" : 4.5 }
{ "_id" : 24, "Max GPA per Age" : 3.5 }
```

Indexing

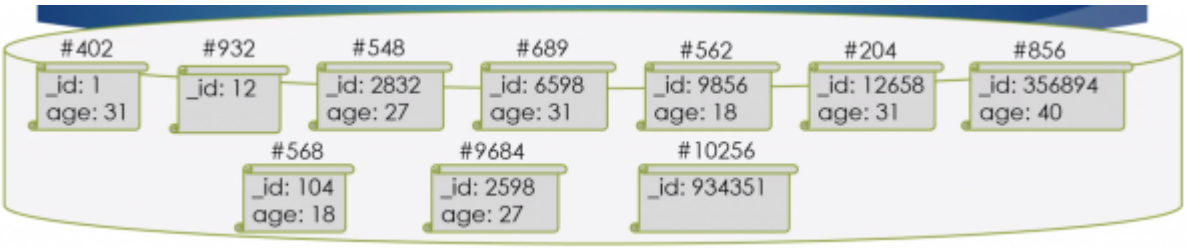
```
db.user.find()
```

Return all documents where age is greater than 18

Psuedo code example

```
for each document d in 'user'{
  if(d.age == 35){
    return d;
  }
}
```

- Indexes support the efficient execution of queries in MongoDB.
- Without indexes, MongoDB must perform a collection scan, i.e scan every document in a collection, to select those documents that match the query statement.
- Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form.
- Indexes hold mappings from field values to document locations.



age Index Table	
Value	Disk Location
18	#562, #568
27	#548, #9684
31	#402, #204, #689
40	#856

getIndexes()

- By default the only index on a document is on the _id field.
- To find the indexes on a collection:

```
db.collection.getIndexes()
```

Which returns information in the following format, detailing the index field (_id) and the order of the indexes(1 is ascending:-1 is descending):

```
"key":{

  "_id":1
}
```

createIndex()

- To create an index on a field other than **_id:**
- **db.collection.createIndex()**

```
db.User.find()
{ "_id" : 1, "first_name" : "John", "surname" : "Smith", "age" : 22, "email" : "john@gmail.com" }
{ "_id" : 2, "first_name" : "Sean", "surname" : "Williams", "age" : 30, "email" : "williamss@gmail.com" }
{ "_id" : 3, "first_name" : "Albert", "surname" : "O'Hara", "age" : 27, "email" : "al@hotmail.com", "twitter" : "@al1234" }
{ "_id" : 4, "first_name" : "Mary", "surname" : "Collins", "age" : 22 }
{ "_id" : 5, "first_name" : "Susan", "surname" : "Hanly", "age" : 18, "email" : "susie@hotmail.com", "twitter" : "@Susie2u" }
```

```
db.user.createIndex({age:1})
```

dropIndex()

- To drop an index on a field use:

```
db.collection.dropIndex()
```

- To drop the index on the age field we just created use:

```
db.collection.dropIndex({age:1})
```

- Note: The index on **_id** cannot be dropped

sort()

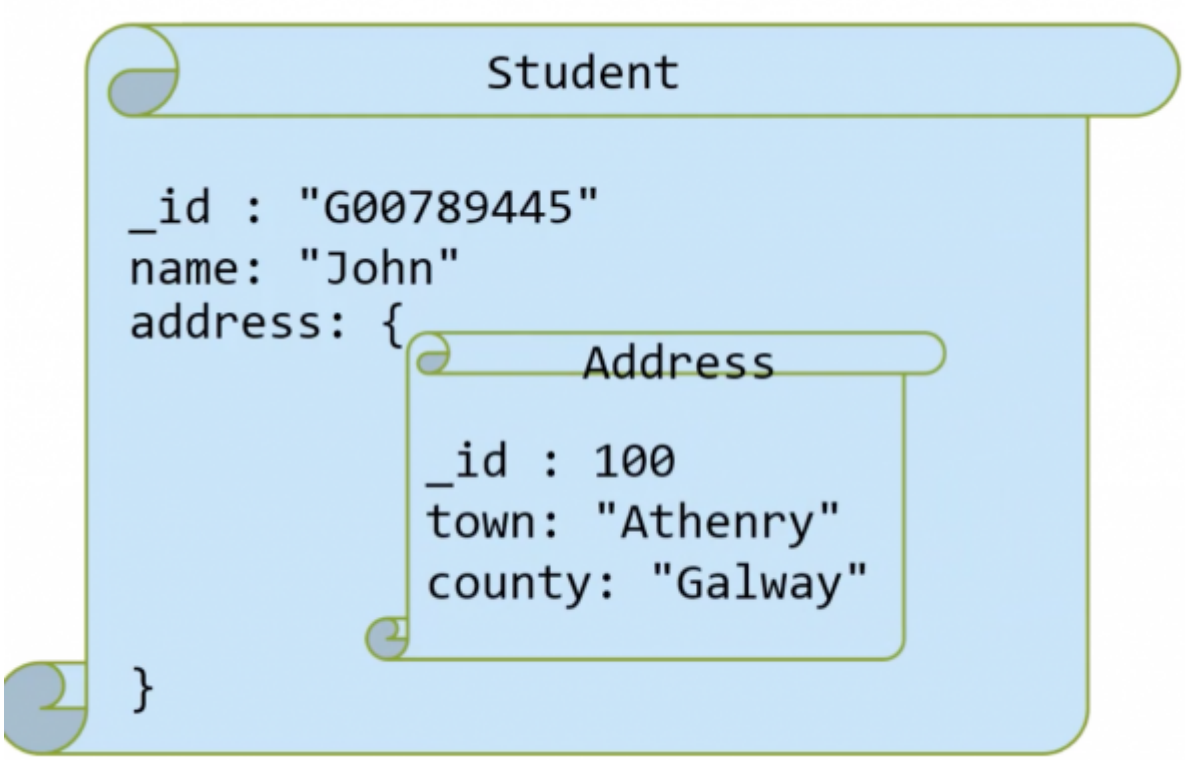
- When a sort() is performed on a field that is not an index, MongoDB will sort the results in memory.
- If the sort() method consumes more than 32MB of memory, MongoDB aborts the sort.
- To avoid this error, create an index supporting the sort operation.

__Relationships__ in MongoDB

- Modelling relationships between documents
  - One-to-One Relationships with Embedded Documents

- One-to-many Relationships with embedded Documents
- One-to-many relationships with document references

One-to-One relationships with embedded documents




```
db.student.save({_id:"G00789445",
                 name: "John",
                 address:{_id: 100,
                          town: "Athenry",
                          county:"Galway"}})
```

```
db.student.find({}, {address:1})
```

```
{ "_id" : "G00789445",
  "address" : {
    "_id" : 100,
    "town" : "Athenry",
    "county" : "Galway"
  }
}
```

- Show only the county field of documents that have an address field.

```
db.student.find({address:{$exists: true}}, {_id:0, "address.county":1})
```

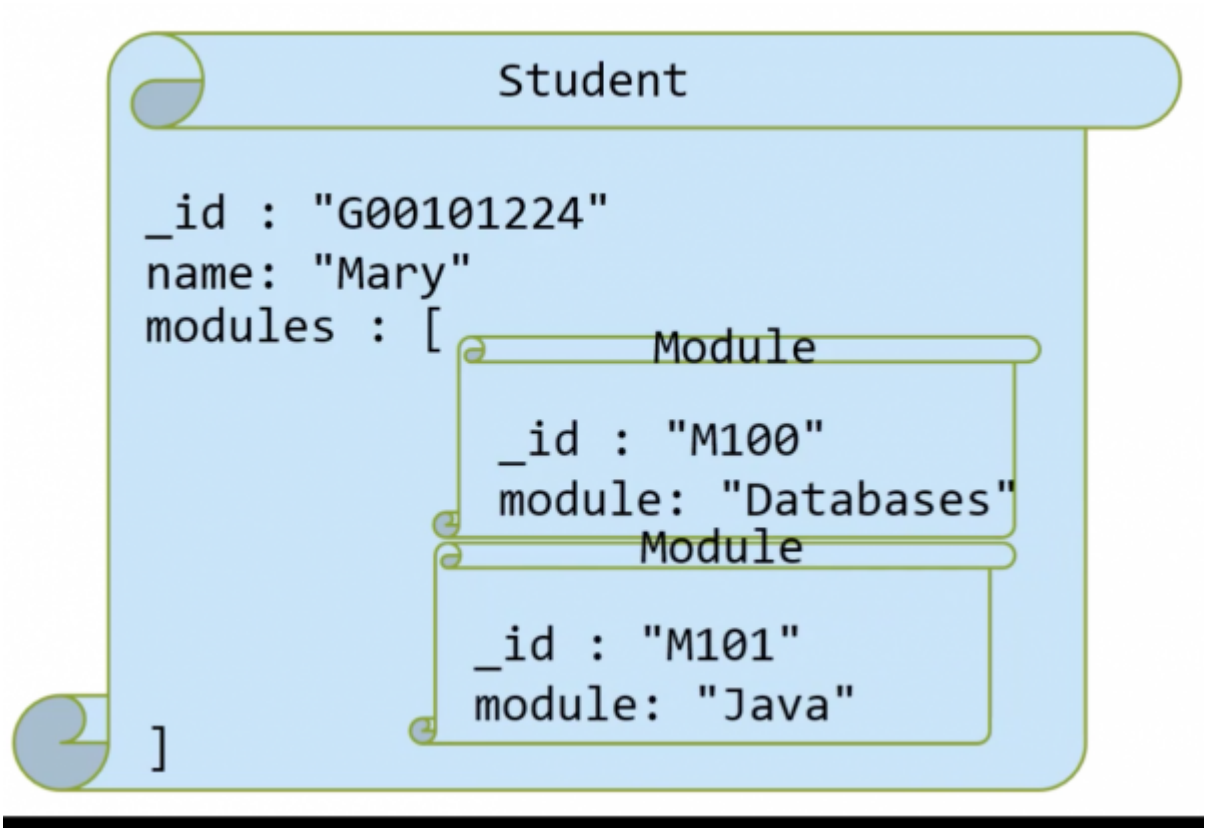
Note: Observe details in projection part of find, i.e. find(query, projection), {_id:0, "address.county":1}

{_id:0, "address.county":1}	Meaning
_id:0	Do NOT output **_id** field

{_id:0, “address.county”:1}	Meaning
“address.county”:1	Only output **county** field

```
{ "address" : { "county" : "Galway" } }
```

One-to-Many Relationships with Embedded Documents



Create the document with the relationships

```
db.student.save({_id:"G00101224",
                 name:"Mary",
                 modules:[{_id:"M100", module:"Databases"},
                          _id:"M101", module:"Java"}]})
```

Show the student's **_id** and **module** of all modules taken by student G00101224

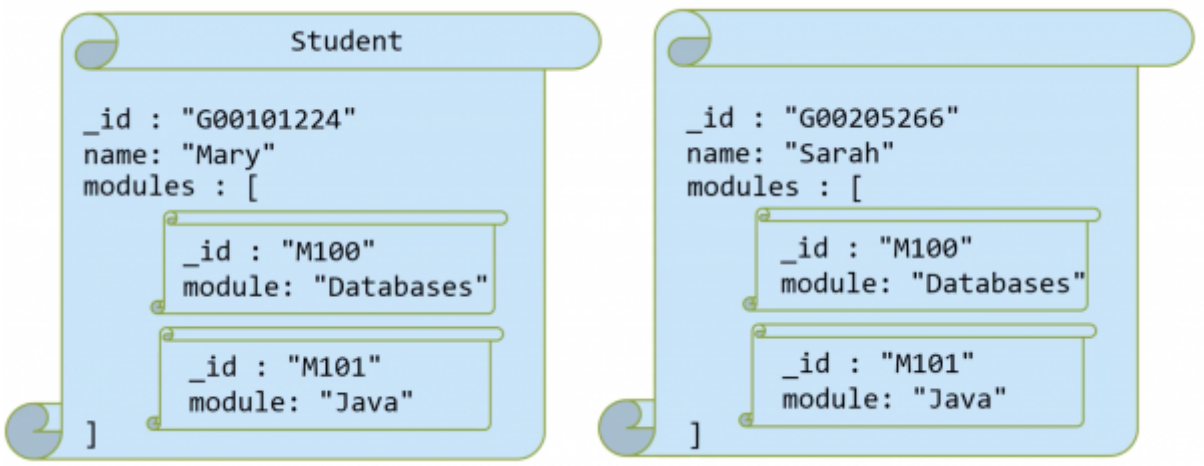
```
db.student.find({_id:"G00101224"}, {"modules.module":1})
```

projection - only show the **module** of the **modules field**

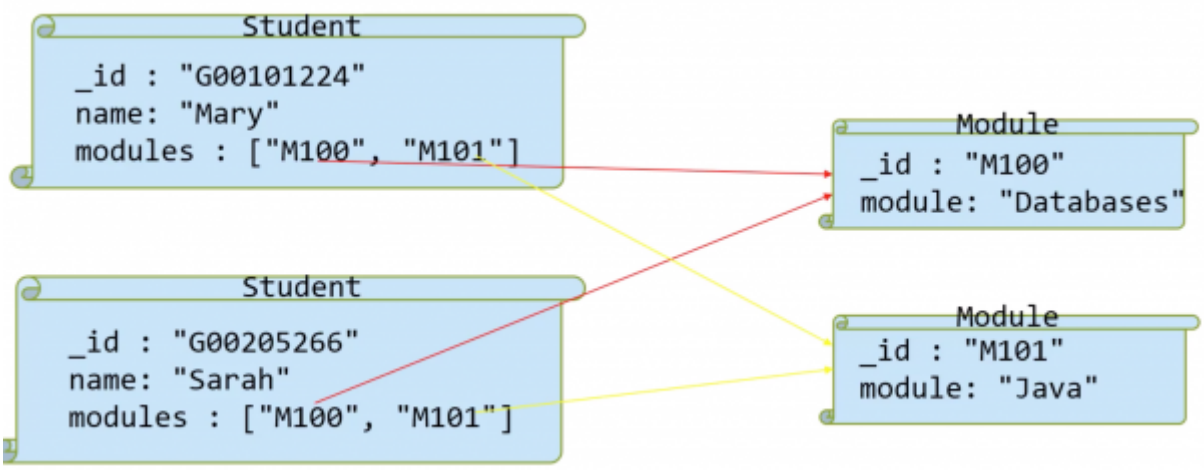
```
{"_id":"G00101224", "modules":[{"module":"Databases"},{"module":"Java"}]}
```

One-to-Many relationships with document References

In the example the document has only two field, but in reality it can be a very long document with much more information, so it makes sense to use relationships instead.



with referencing




```
//save the modules to the docs collection
db.docs.save({_id:"M100", module:"Databases"})
db.docs.save({_id:"M101", module:"Java"})
//save the students to the docs collection with references to the modules using the module _id
fields.
db.docs.save({_id:"G00101224", name:"Mary", modules["M100", "M101"]})
db.docs.save({_id:"G00205266", name:"Sarah", modules["M100", "M101"]})
```

$lookup

Using the $lookup *pipeline*... [3)]

Similar to a join in MySQL...

Performs a **left outer join** to an *unsharded* collection in the same database to filter in documents from the “joined” collection for processing. To each input document, the $lookup stage adds a new array field whose elements are the matching documents from the “joined” collection. The **$lookup** stage passes these reshaped documents to the next stage.

```
{
   $lookup:
     {
       from: <collection to join>,
       localField: <field from the input documents>,
       foreignField: <field from the documents of the "from" collection>,
       as: <output array field>
     }
}
```

from - specifies the collection in the same databases to perform the join with. The from collection cannot be *sharded*.

localField - The value to search for.

foreignField - The field to search for the value specified by localField.

as - The name of the output.

```
{ "_id" : "M100", "module" : "Databases" }
{ "_id" : "M101", "module" : "Java" }
{ "_id" : "G00101224", "name" : "Mary", "modules" : [ "M100", "M101" ] }
{ "_id" : "G00205266", "name" : "Sarah", "modules" : [ "M101" ] }
```

Return all documents including the complete referenced documents

```
db.docs.aggregate([{$lookup:{from:"docs", localField:"modules", foreignField: "_id",
as:"Details"}}])
```

```
{ "_id" : "M100", "module" : "Databases", "Details" : [ ] }

{ "_id" : "M101", "module" : "Java", "Details" : [ ] }

{ "_id" : "G00101224", "name" : "Mary", "modules" : [ "M100", "M101" ],
"Details" : [ { "_id" : "M100", "module" : "Databases" }, { "_id" :"M101", "module" : "Java" } ] }

{ "_id" : "G00205266", "name" : "Sarah", "modules" : [ "M101" ],
"Details" : [ { "_id" : "M101", "module" : "Java" } ] }
```

Embedded Documents vs Referenced Documents

Features of embedded Documents

- Better performance
- Atomic

Features of Referenced Documents

- Slower
- No repetition
- More complex relationships

MongoDB vs MySQL

Features of MongoDB

- Huge amounts of data
- Unstructured
- Doesn't really support relationships

Features of MySQL

- Very Stable
- Structured
- Integrity

1)

Sharding is a type of database partitioning that separates very large databases the into smaller, faster, more easily managed parts called data shards. The word shard means a small part of a whole.

2)

https://docs.mongodb.com/manual/reference/method/db.collection.update/#db.collection.update

3)

https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/