# MongoDB Course by Amigos Code

https://www.amigoscode.com/courses/mongodb - $34.99 (Official AmigosCode Website)

https://github.com/amigoscode/mongodb-course

https://tut4it.com/ - Search Amigoscode, download the course

## MongoDB

When we use the mongosh shell to show all the available db's

```
show dbs;
admin   40.00 KiB
config  72.00 KiB
local   40.00 KiB
```

These are all the default databases that come with MongoDB.

First, to create our own database or switch to database if exists, command use <database_name>

Database > collections > documents

To get the Current database Name: -

```
db.getName();
mongoamigoscode
```

To create a Collection inside the database, use this Command: -

```
db.createCollection('Hello Course');
{ ok: 1 }
show dbs;
admin          40.00 KiB
config         96.00 KiB
local          40.00 KiB
mongoamigoscode   8.00 KiB
```

To drop/delete a database, use this Command: -

```
db.dropDatabase();
{ ok: 1, dropped: 'mongoamigoscode' }
```

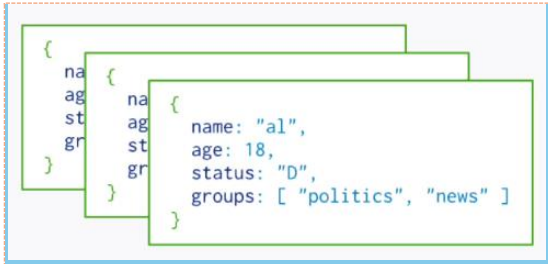
Returns the JSON Object.

To return the current database connection.

```
db.getMongo();
mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.2.5
```

COLLECTIONS

Mongo stores documents (rows) in collections (table)




```
db.createCollection(name, {size: ....,
capped: ...., max: ....})
db.createCollection("person");
{ ok: 1 }
show collections;
person
```

For showing Collection Stats: -

```
db.<collectionName>.stats();
db.person.stats();
```

To drop the collection in a database:-

```
db.<collectionName>.drop();
db.person.drop();
true
db.createCollection( "person", { capped: true, size: 6142800, max: 30000});
{ ok: 1 }
```

## DOCUMENTS

MongoDB stores data records as BSON Documents. BSON is a binary representation of JSON Documents.

```
{
  name: "sue",                    ←— field: value
  age: 26,                        ←— field: value
  status: "A",                    ←— field: value
  groups: [ "news", "sports" ]    ←— field: value
}
```

## CREATE A DOCUMENT IN MONGODB

```
db.createCollection("person");
db.person.insert(person);
db.person.insertMany(person);
db.person.countDocuments();     1
db.person.estimatedDocumentCount();          1
db.person.find().pretty();
[
 {
   _id: ObjectId('662a38bb67093784d646b799'),
   firstName: 'Robby',
   lastName: 'Deaconson',
   email: 'rdeaconson0@hubpages.com',
   gender: 'M',
   country: 'Indonesia',
   isStudentActive: true,
   favouriteSubjects: 'Fashion Design',
   totalMoneySpentOnBooks: '$182.32'
 }
]
```

## OBJECT ID's

ObjectId values are 12 bytes in Length.

_id: ObjectId('662a38bb67093784d646b799'), MongoDB auto generates the ID's when we insert documents in a collection.

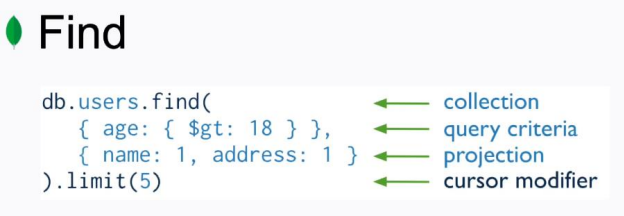
```
db.person.find({}, {_id: 1}).pretty();
[ { _id: ObjectId('662a38bb67093784d646b799') } ]
ObjectId('662a38bb67093784d646b799').getTimestamp();
ISODate('2024-04-25T11:04:27.000Z')
```

## QUERYING WITHIN MONGODB

```
db.person.insertMany(person);
{
 acknowledged: true,
 insertedIds: {
  '0': ObjectId('662a8156c03e42a61174f328'),
  '1': ObjectId('662a8156c03e42a61174f329'),
  '2': ObjectId('662a8156c03e42a61174f32a'),
  '3': ObjectId('662a8156c03e42a61174f32b'),
  '4': ObjectId('662a8156c03e42a61174f32c'),
  '5': ObjectId('662a8156c03e42a61174f32d'),
  '6': ObjectId('662a8156c03e42a61174f32e'),
  '7': ObjectId('662a8156c03e42a61174f32f'),
  '8': ObjectId('662a8156c03e42a61174f330'),
  '9': ObjectId('662a8156c03e42a61174f331'),
  '10': ObjectId('662a8156c03e42a61174f332'),
```

```
  '11': ObjectId('662a8156c03e42a61174f333')
 }
}
db.person.findOne({studentId: 'b372a524-567f-4004-b849-c4dceae99a03'});
{
  _id: ObjectId('662a416248a253440246b799'),
  studentId: 'b372a524-567f-4004-b849-c4dceae99a03',
  firstName: 'Robby',
  lastName: 'Deaconson',
  email: 'rdeaconson0@hubpages.com',
  gender: 'M',
  country: 'Indonesia',
  isStudentActive: true,
  favouriteSubjects: 'Fashion Design',
  totalMoneySpentOnBooks: '$182.32'
}
```

FIND Function




```
db.person.find({gender : {$eq: 'M'}}, {firstName : 'Deaconson', country:
'Australia'}).limit(5);
db.person.find({gender : {$eq: 'M'}}, {firstName : 'Deaconson', country:
'Australia'}).limit(5).count();        5

db.person.find({firstName: 'Fabe'}).pretty();
{
  _id: ObjectId('662a8156c03e42a61174f32a'),
  firstName: 'Fabe',
  lastName: 'Hawick',
  email: 'fhawicke@psu.edu',
  gender: 'M',
  country: 'Greece',
  isStudentActive: true,
  favouriteSubjects: 'Art',
  totalMoneySpentOnBooks: '$227.01'
}
db.person.find({firstName: 'York'}, {gender: 0, firstName: 0, lastName:
0}).pretty();
Basically '0' means exclude.
Resultant is as follows: -
{
  _id: ObjectId('662a8156c03e42a61174f32f'),
  email: 'yalfordj@fema.gov',
  country: 'France',
  isStudentActive: true,
  favouriteSubjects: 'Horticulture',
  totalMoneySpentOnBooks: '$386.31'
}
```

QUERY SELECTORS

# Query Selectors

| Operation | Syntax | SQL Equivalent |
|---|---|---|
| **Equals** | {key:{$eq: value}} | where column **==** value |
| **Not Equals** | {key:{$ne: value}} | where column **!=** value |
| **Less Than** | {key:{$lt: value}} | where column **<** value |
| **Less Than Equals** | {key:{$lte: value}} | where column **<=** value |
| **Greater Than** | {key:{$gt: value}} | where column **>** value |
| **Greater Than Equals** | {key:{$gte:value}} | where column **>=** value |
| **Values In Array** | {key:{$in:[v1, v2, v3]}} | where column **in** (v1,v2,v3) |
| **Values Not In Array** | {key:{$nin:[v1, v2, v3]}} | where column **not in** (v1,v2,v3) |
| **And** | {$and: [{k: v}, {k: v} ]} | where column == x **and** column == y |
| **Or** | {$or: [{k: v}, {k: v} ]} | where column == x **or** column == y |
| **Not** | {$not: [{k: v}, {k: v} ]} | where not condition |

mongoDB

```
db.person.find({totalMoneySpentOnBooks : {$eq: "$880.36"}}).pretty();
{
  _id: ObjectId('662a808548a253440246b7a3'),
  firstName: 'Feodora',
  lastName: 'Smeeth',
  email: 'fsmeethb@hibu.com',
  gender: 'F',
  country: 'Albania',
  isStudentActive: true,
  favouriteSubjects: 'Military Science',
  totalMoneySpentOnBooks: '$880.36'
}
db.person.find({totalMoneySpentOnBooks : {$eq: "$880.36"}}).pretty();
```

$eq – Equals

$ne – Not Equals

```
db.person.find({totalMoneySpentOnBooks : {$ne: "$880.36"}}).pretty().count();
23
db.person.find({totalMoneySpentOnBooks : {$ne: "$880.36"}}, {firstName:
1}).pretty();
```

```
{
  _id: ObjectId('662a38bb67093784d646b799'),
  firstName: 'Robby'
}
db.person.find({totalMoneySpentOnBooks : {$ne: "$880.36"}}, {firstName:
1}).pretty().count();      23
db.person.find({totalMoneySpentOnBooks : {$ne: "$880.36"}}, {firstName: 1,
totalMoneySpentOnBooks: 1}).pretty();
{
  _id: ObjectId('662a38bb67093784d646b799'),
  firstName: 'Robby',
  totalMoneySpentOnBooks: '$182.32'
}
db.person.find({totalMoneySpentOnBooks : {$lt: "$200"}}, {firstName: 1,
totalMoneySpentOnBooks: 1}).pretty();
{
  _id: ObjectId('662a38bb67093784d646b799'),
  firstName: 'Robby',
  totalMoneySpentOnBooks: '$182.32'
}
```

$lt – Less Than

$gt – Greater Than

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

```
db.person.find({}, {favouriteSubjects: 1}).pretty();
{
  _id: ObjectId('662a38bb67093784d646b799'),
  favouriteSubjects: 'Fashion Design'
}
db.person.find({favouriteSubjects: {$all : ["Drama"]}}, {favouriteSubjects:
1}).pretty();
{
  _id: ObjectId('662a808548a253440246b79d'),
  favouriteSubjects: 'Drama'
```

```
}
db.person.find({favouriteSubjects: {$in : ["Drama"]}}, {favouriteSubjects:
1}).pretty();
Exact same result as above query
db.person.find({favouriteSubjects: {$nin : ["Drama", "Horticulture"]}},
{favouriteSubjects: 1}).pretty();
$all – Include all the values
$in
$nin – Not in. Don't include the search value
db.person.find({favouriteSubjects: {$nin : ["Drama", "Horticulture"]}},
{favouriteSubjects: 1}).pretty().count();
```

## UPDATING THE DATA FROM A QUERY IN MONGODB

To select only the firstName and lastName

```
db.person.find({}, {firstName: 1, lastName:1}).pretty();
{
  _id: ObjectId('662a38bb67093784d646b799'),
  firstName: 'Robby',
  lastName: 'Deaconson'
}
db.person.find({}, {firstName: 1, lastName:1}).pretty().count();        24
db.person.updateOne({_id: ObjectId('662a8156c03e42a61174f32e')}, {$set:
{firstName: 'PrasanthiBHima'}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
db.person.updateOne({_id: ObjectId('662a8156c03e42a61174f32e')}, {$unset:
{lastName:1}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
$set – Rewriting the existing value
$unset – Remove the field
{
  _id: ObjectId('662a8156c03e42a61174f32e'),
  firstName: 'PrasanthiBHima',
  gender: 'F'
}
db.person.find({}, {totalMoneySpentOnBooks:1}).pretty();
{
  _id: ObjectId('662a38bb67093784d646b799'),
  totalMoneySpentOnBooks: '$182.32'
}
$inc – (Increment function). To increment the Numeric Values like age, price etc.
Update deprecated, use updateOne or updateMany.
While using update operation, use $pull only on the array values. We cannot apply
$pull to non-array value.
Cannot apply $pull to a non-array value
While using update operation, use $push for adding value to the array. We cannot
apply $push to non-array value.
The field 'favouriteSubjects' must be an array but is of type string in document
{_id: ObjectId('662a8156c03e42a61174f32f')}
```

## DELETE OPERATION FROM A COLLECTION

```
db.person.deleteOne({_id: ObjectId('662a38bb67093784d646b799')});
{
  acknowledged: true,
  deletedCount: 1
}
```

```
With delete Operation, one should be careful with the Query Criteria and Filter.
db.person.deleteMany({gender: 'M'});
{
  acknowledged: true,
  deletedCount: 14
}
Query Criteria - Gender with M, all documents deleted.
Basically, the delete operation deletes every document if you don't include
filter.
```

## CURSOR MODIFIER

When you use the find() method, in reality what it gives you back is a Cursor.

```
var cursor = db.person.find();

cursor.count();

209

cursor.hasNext();          has a next element, returns true if present/iterates

True       In reality, when we use the find() methods it only gives us the first
     20 documents. If we want more documents, we goof and use the
     iterator.

cursor.hasNext(); can also be false if the iterator has no next document.

cursor.next();  Actually, gives us the whole document/object if the iterator
           contains the next document/object. Can also give us false, if
     the iterator has no next document/object.

cursor.count(); Gives us the Number of documents present in a collection.

db.person.find({}, {_id: 1});          After 20 documents.

Type "it" for more. Here "it" is the Iterator
```

## LIMITING THE RESULTS, SORTING, SKIPPING

```
db.person.find({}, {firstName: 1, country: 1}).pretty().limit(5);

Only the first 5 documents.

If you want to skip any document, we have a function called skip(), it will
accept integer/numeric value.

db.person.find({}, {firstName: 1, country: 1}).pretty().limit(5).skip(1);

The sort() function accepts an object.
db.person.find({}, {firstName: 1, country: 1}).pretty().limit(5).sort({firstName:
1});
```

```
.sort({firstName: 1}); --> Ascending Order Sorting
db.person.find({}, {firstName: 1, country: 1}).pretty().limit(5).sort({firstName:
-1});
.sort({firstName: -1}); --> Descending Order Sorting
Another Example: -
db.person.find({}, {firstName: 1, country: 1}).pretty().limit(5).sort({firstName:
-1, country: 1});
At the end you can use the skip() method too for skipping the documents.
```

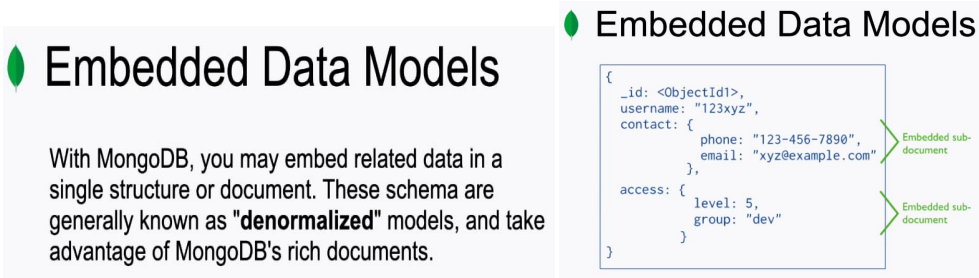
## CURSOR.FOREACH(), CURSOR.MAP()

curosr.forEach() method. It iterates the cursor to apply a JavaScript function for each document.

```
db.person.find().forEach(function(person) {print(person)});

db.person.find().forEach(function(person) {print(person._id)});

db.person.find().forEach(function(person) {print(person.gender)});

db.person.find().forEach(function(person) {print("Gender - " + person.gender)});

db.person.find().forEach(function(person) {print("Gender - " + person.gender + " "
+ person.firstName)});
```

```
Gender - F Xylina
Gender - F Myrtie
Gender - F Austin
Gender - F Rochell
Gender - F Feodora
```

## EMBEDDED DATA MODELS, NORMALIZE DATA MODELS

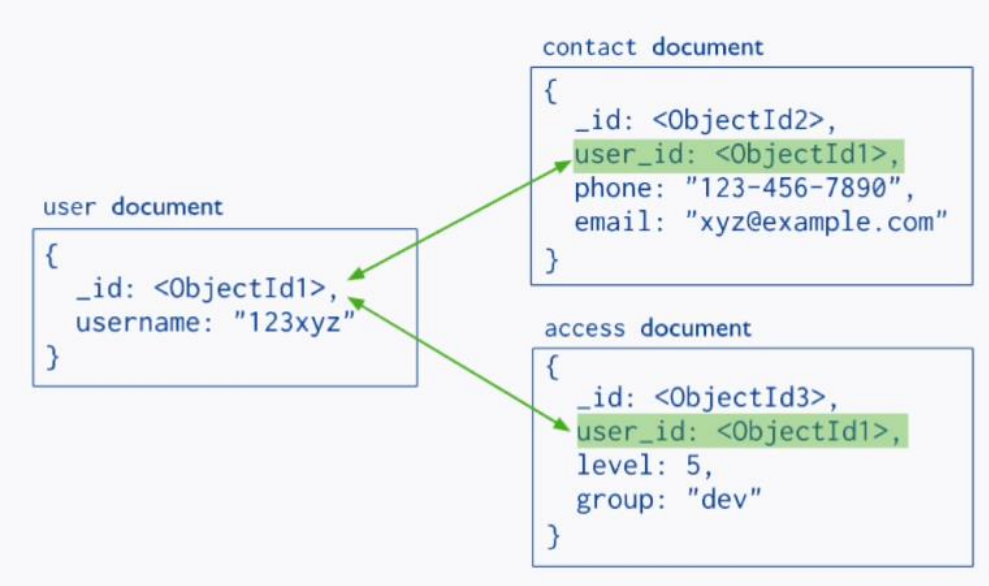



Relations like OneToOne, OneToMany use Embedded Data Models.

Better performance for Read Operations.

# Normalised Data Models

Normalized data models describe relationships using references between documents.




For normalized Data Models ManyToMany Relationships, Large Hierarchical Data Sets. This would result in duplication of data but would not provide sufficient read performance advantages.

## VALIDATION's

Schema validations in MongoDB.

```
db.person_with_validation.insertOne(person);
MongoServerError: Document failed validation
db.person_with_validation.insertOne(person);
{
  acknowledged: true,
  insertedId: ObjectId('662c9c72861aa963a1e4fc92')
}
```

With Validations Successful

## INDEXING IN MONGODB

Index allows you to speed up your queries.

db.person.find().explain("executionStats");

```
db.person.find({firstName : 'Xylina'}).explain("executionStats");
Result: -
  executionStats: {
    executionSuccess: true,
    nReturned: 2,
    executionTimeMillis: 16,
    totalKeysExamined: 0,
    totalDocsExamined: 209,
    executionStages: {
      stage: 'COLLSCAN',
      filter: {
        firstName: {
          '$eq': 'Xylina'
        }
      },
      direction: 'forward',
      docsExamined: 209
    }
  },            --> Examined 209 Documents, the key doesn't have an index.
In real time projects we will be in trouble, if no index is present

db.person.find({_id:
ObjectId('662a808548a253440246b79a')}).explain("executionStats");
executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 0,
    totalKeysExamined: 1,
    totalDocsExamined: 1,
    executionStages: {
      keysExamined: 1,
      docsExamined: 1
    }
  },

db.person.getIndexes();
[ { v: 2, key: { _id: 1 }, name: '_id_' } ]
```

```
db.person.getIndexKeys();
[ { _id: 1 } ]
```

TO CREATE AN INDEX IN MONGODB

```
db.person.createIndex({firstName: 1});
firstName_1
db.person.getIndexes();
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { firstName: 1 }, name: 'firstName_1' }
]
db.person.getIndexKeys();
[ { _id: 1 }, { firstName: 1 } ]
db.person.find({firstName : 'Xylina'}).explain("executionStats");
executionStats: {
    executionSuccess: true,
    nReturned: 2,
    totalKeysExamined: 2,
    totalDocsExamined: 2,
    executionStages: {
      stage: 'FETCH',
      nReturned: 2,
      docsExamined: 2, --> Before, the docsExamined was at 209, after creating an
                           index on the firstName the docsExamined was at 2
```

To drop Indexes Command is

```
db.person.dropIndex({firstName: 1});   * 1--> Ascending Order, -1--> Descending
                                                                  Order *
{ nIndexesWas: 2, ok: 1 }        * The _id by default comes with an Index *
```

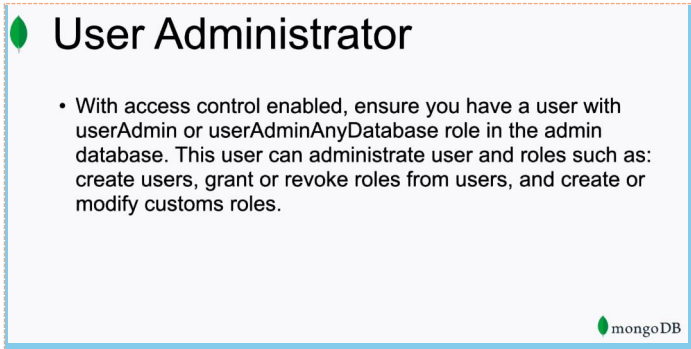
DATABASE ADMINISTRATION IN MONGODB

Mongo Configuration

Authentication --> Who can Login

Authorization --> What the logged in user can do.

RBAC --> Role Based Access Control




show dbs;

Use admin;

```
db.createUser({user: "UserAdmin", pwd: passwordPrompt(), roles: [{role:
"userAdminAnyDatabase", db: "admin"}, "readWriteAnyDatabase"]});
{ ok: 1 } --> Password: root@12345
{
  _id: 'admin.UserAdmin',
  userId: UUID('aa404dd9-a6f0-4cf5-a821-b7c60641e53a'),
  user: 'UserAdmin',
  db: 'admin',
  credentials: {
    'SCRAM-SHA-1': {
      iterationCount: 10000,
      salt: 'G1OJ0cWuoMwyit3wu/XXGw==',
      storedKey: '7f5YTJZyltxOBygZE/YtEa1HYKk=',
      serverKey: 'RiNzBFQ/2wapLDgS66jqSInG86U='
    },
    'SCRAM-SHA-256': {
      iterationCount: 15000,
      salt: 'Qp6Mxdzx4y+u6Q6i8RH2WOhX5CXSHmArrpv7Qw==',
      storedKey: 'ZLVSSAK6r0p3ghcvZJAWAQDa0iFmQCPJ5FY6d80o484=',
```

```
      serverKey: 'EG+xxBjABB8s9IoXfl54MGYbSA7tEwoisext1Sd/JnY='
    }
  },
  roles: [
    {
      role: 'readWriteAnyDatabase',
      db: 'admin'
    },
    {
      role: 'userAdminAnyDatabase',
      db: 'admin'
    }
  ]
}
```