Introdução ao MongoDB

AULA PL06

Hugo Peixoto

2019 – 2020 Universidade do Minho

Conteúdo da UC

https://hpeixoto.me/class/nosql

- Introduction & Basics
- Schema Design
  - Data Modeling
- CRUD
  - Insert Document
  - Query Document
  - Update Document
  - Remove Document
- Install and First Commands

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

INTRODUCTION & BASICS

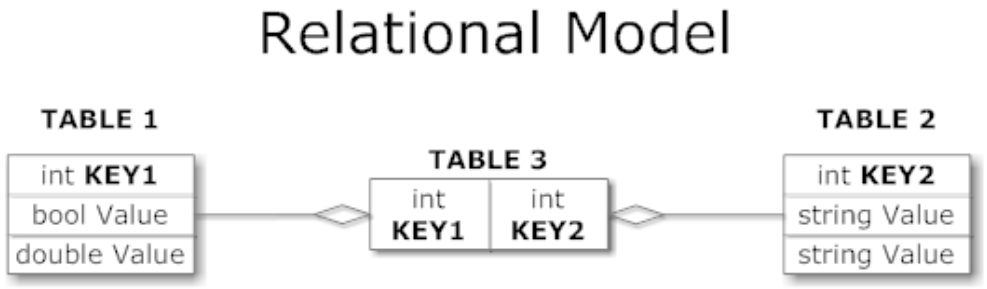

mongoDB

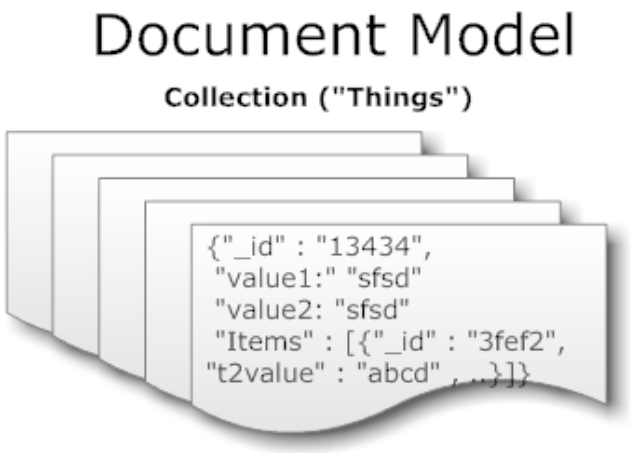
History

Hu**mongo**us DB

- Open-source
- Document-based
- High performance, high availability
- Automatic scaling

Relational Model vs Document Model





Motivations

Problems with SQL

- Rigid schema
- Not easily scalable (designed for 90's technology or worse)
- Requires unintuitive joins

Perks of mongoDB

- Easy interface with common languages (Java, Javascript, PHP, etc.)
- DB tech should run anywhere (VM's, cloud, etc.)
- Keeps essential features of RDBMS's while learning from key-value noSQL systems

In Good Company

Data Model

Document-Based (max 16 MB)

Documents are in BSON format, consisting of field-value pairs

Each document stored in a collection

Collections:

- Have index set in common
- Like tables of relational db's
- Documents do not have to have uniform structure

JSON

"JavaScript Object Notation"

Easy for humans to write/read, easy for computers to parse/generate

Objects can be nested

Built on:

- name/value pairs
- Ordered list of values

BJSON

"Binary JSON"

Binary-encoded serialization of JSON-like docs

Also allows "referencing"

Embedded structure reduces need for joins

Goals

- Lightweight
- Traversable
- Efficient (decoding and encoding)

BJSON

MongoDB represents JSON documents BSON behind the scenes.

BSON extends the JSON model to provide additional data types, ordered fields, and to be efficient for encoding and decoding within different languages.

Document Example

```
{
"_id" :    "37010"
"city" :   "ADAMS",
"pop" :   2660,
"state" :              "TN",
“councilman” : {
                          name: “John Smith”
                          address: “13 Scenic Way”
                        }
}
```

The _id Field

By default, each document contains an _id field. This field has a number of special characteristics:

- Value serves as primary key for collection.
- Value is unique, immutable, and may be any non-array type.
- Default data type is ObjectId, which is “small, likely unique, fast to generate, and ordered.” Sorting on an ObjectId value is roughly equivalent to sorting on creation time.

Field Types

Type	Description	Number
Double	used to store floating point values.	1
String	BSON strings are UTF-8. It is most commonly used datatype.	2
Object	used for embedded documents.	3
Array	used to store multiples of values and data types.	4
Binary data	used to store the binary data.	5
Undefined	used to store the undefined values.	6
Object id	used to store the document's id.	7
Boolean	used to store a boolean value.	8
Date	used to store the current date or time in UNIX time format.	9
Null	used to store a Null value.	10
Regular Expression	used to store regular expression.	11
JavaScript	used to store the javascript data without a scope.	13
Symbol	It is similar to the string data type, and not supported by a shell.	14
JavaScript (with scope)	It stores javascript data with a scope.	15
Timestamp	used to store a timestamp.	17
Min key	Min key compares the value of the lowest BSON element.	255
Max key	Max key compares the value against the highest BSON element.	127

mongoDB vs. SQL

mongoDB	SQL
Document	Tuple
Collection	Table/View
PK: _id Field	PK: Any Attribute(s)
Uniformity not Required	Uniform Relation Schema
Index	Index
Embedded Structure	Joins

SCHEMA DESIGN

RDBMS		MongoDB
Database	→	Database
Table	→	Collection
Row	→	Document
Index	→	Index
Join	→	Embedded Document
Foreign Key	→	Reference

Why databases exist in the first place?

Why can’t we just write programs that operate on objects?

- Memory limit
- We cannot swap back from disk merely by OS for the page-based memory management mechanism

Why can’t we have the database operating on the same data structure as in program?

- That is where mongoDB comes in

Mongo is basically schema-free

The purpose of schema in SQL is for meeting the requirements of tables and quirky SQL implementation

Every "*row*" in a database "*table*" is a data structure, much like a "struct" in C, or a "class" in Java. A table is then an array (or list) of such data structures

So what we design in mongoDB is basically same way how we design a compound data type binding in JSON

Data Modeling

Embedded Documents

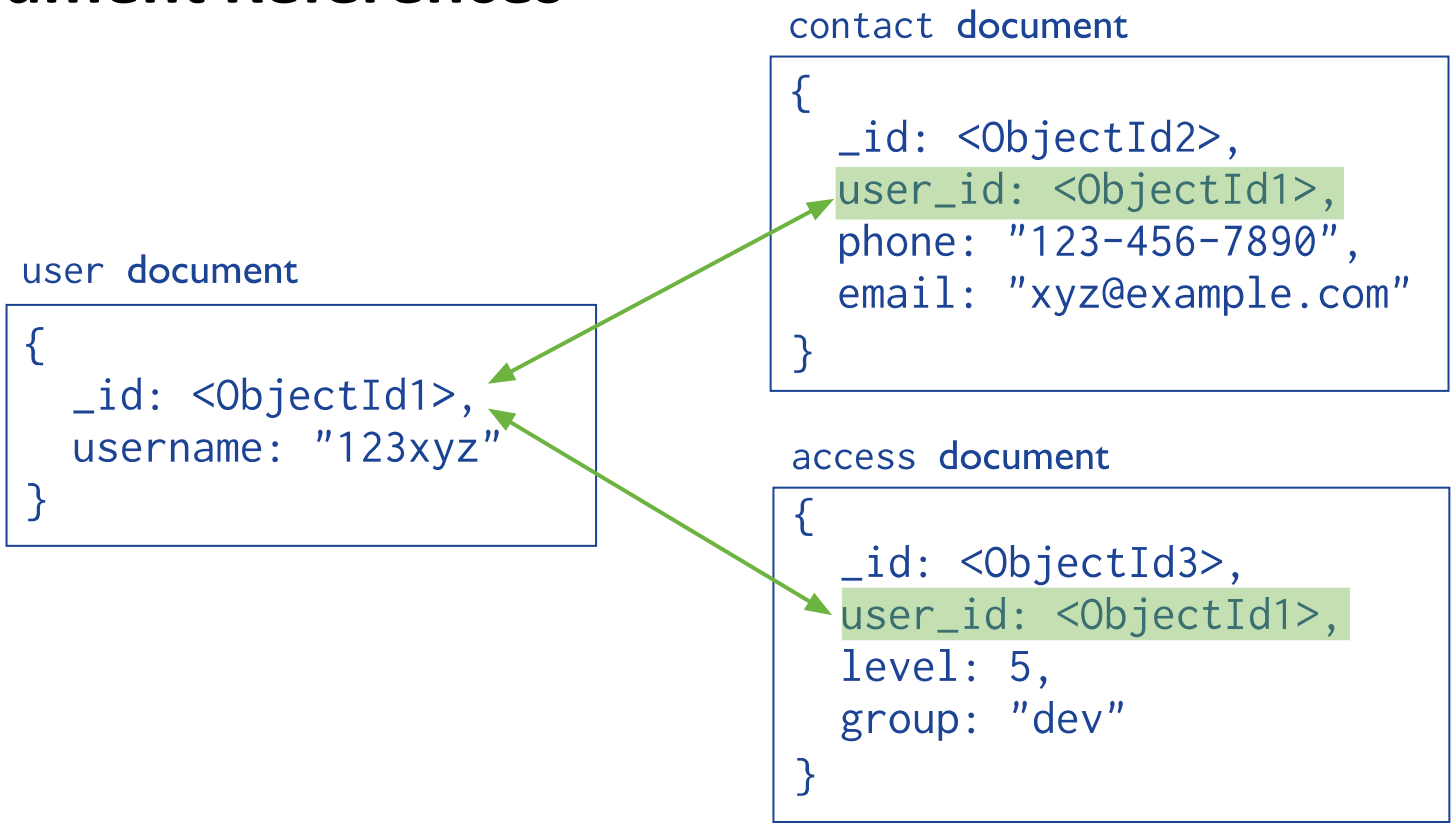
```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
             phone: "123-456-7890",
             email: "xyz@example.com"
           },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Embedded sub-document

Embedded sub-document

Data Modeling

Document References



Embedded vs Reference

Embedding is a bit like pre-joining data

Document level operations are easy for the server to handle

Embed when the “many” objects always appear with (viewed in the context of) their parents.

References are used when more flexibility is neededd

Embedded Documents

One-to-one relationships

```
{
  _id: "joe",
  name: "Joe Bookreader"
}

{
  patron_id: "joe",
  street: "123 Fake Street",
  city: "Faketon",
  state: "MA",
  zip: "12345"
}
```

```
{
  _id: "joe",
  name: "Joe Bookreader",
  address: {
          street: "123 Fake Street",
          city: "Faketon",
          state: "MA",
          zip: "12345"
        }
}
```

Embedded Documents

One-to-Many Relationships: One patron several addresses

```
{
  _id: "joe",
  name: "Joe Bookreader"
}
```

```
{
  patron_id: "joe",
  street: "123 Fake Street",
  city: "Faketon",
  state: "MA",
  zip: "12345"
}
```

```
{
  patron_id: "joe",
  street: "1 Some Other Street",
  city: "Boston",
  state: "MA",
  zip: "12345"
}
```

Embedded Documents

One-to-Many Relationships: One patron several addresses

```
{
  _id: "joe",
  name: "Joe Bookreader",
  addresses: [
            {
              street: "123 Fake Street",
              city: "Faketon",
              state: "MA",
              zip: "12345"
            },
            {
              street: "1 Some Other Street",
              city: "Boston",
              state: "MA",
              zip: "12345"
            }
          ]
 }
```

Referenced Documents

```
{
  title: "MongoDB: The Definitive Guide",
  author: [ "Kristina Chodorow", "Mike Dirolf" ],
  published_date: ISODate("2010-09-24"),
  pages: 216,
  language: "English",

  publisher: {
        name: "O'Reilly Media",
        founded: 1980,
        location: "CA"
      }
}
```

```
{
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English",

  publisher: {
        name: "O'Reilly Media",
        founded: 1980,
        location: "CA"
      }
}
```

Referenced Documents

```
{
  name: "O'Reilly Media",
  founded: 1980,
  location: "CA",
  books: [123456789, 234567890, ...]
}
```

```
{
  _id: 123456789,
  title: "MongoDB: The Definitive Guide",
  author: [ "Kristina Chodorow", "Mike Dirolf" ],
  published_date: ISODate("2010-09-24"),
  pages: 216,
  language: "English"
}
```

```
{
  _id: 234567890,
  title: "50 Tips and Tricks for MongoDB Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English"
}
```

What is bad about SQL

"Primary keys" of a database table are in essence persistent memory addresses for the object. The address may not be the same when the object is reloaded into memory. This is why we need primary keys.

Foreign key functions just like a pointer in C, persistently point to the primary key.

Whenever we need to deference a pointer, we do JOIN

It is not intuitive for programming and also JOIN is time consuming

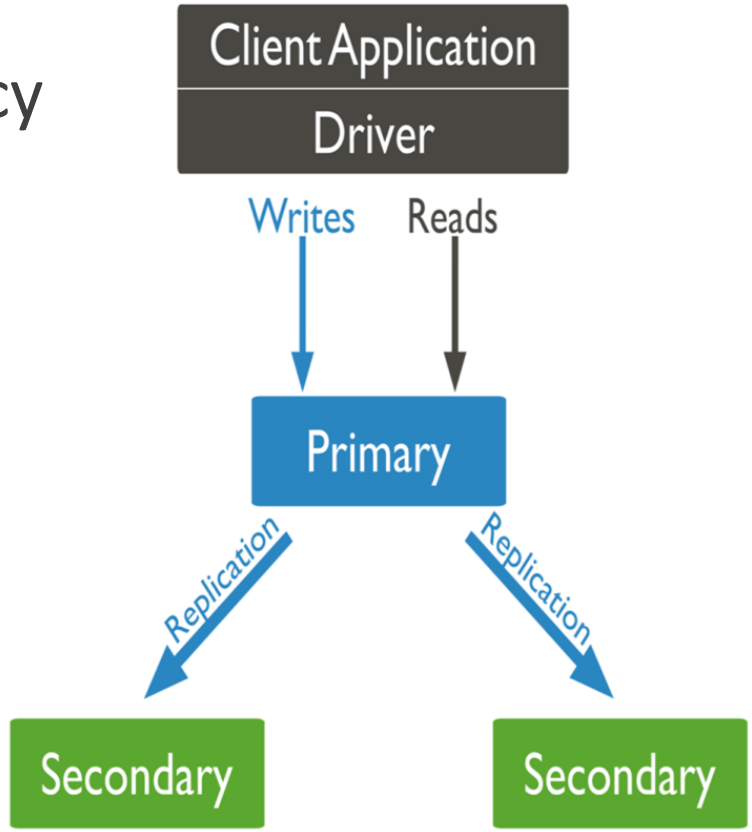
REPLICATION & SHARDING

Replication

What is replication?

Purpose of replication/redundancy

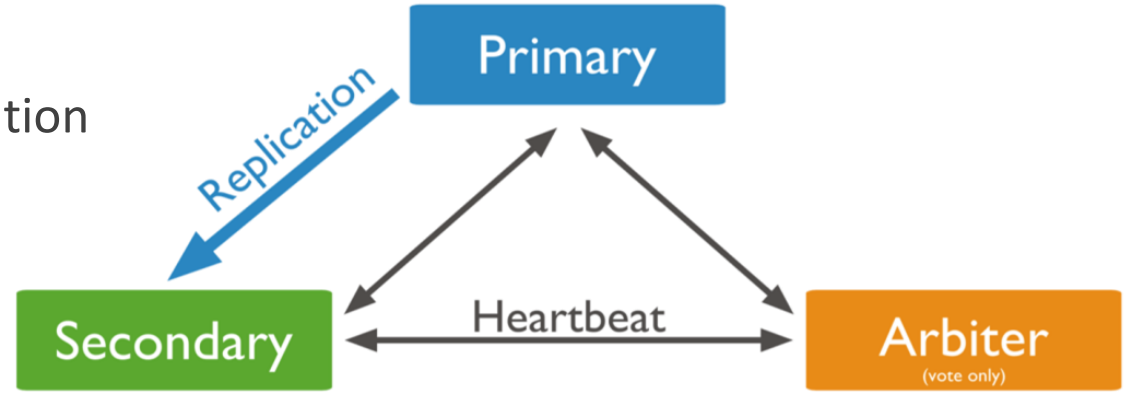
- Fault tolerance
- Availability
  - Increase read capacity



Replication in MongoDB

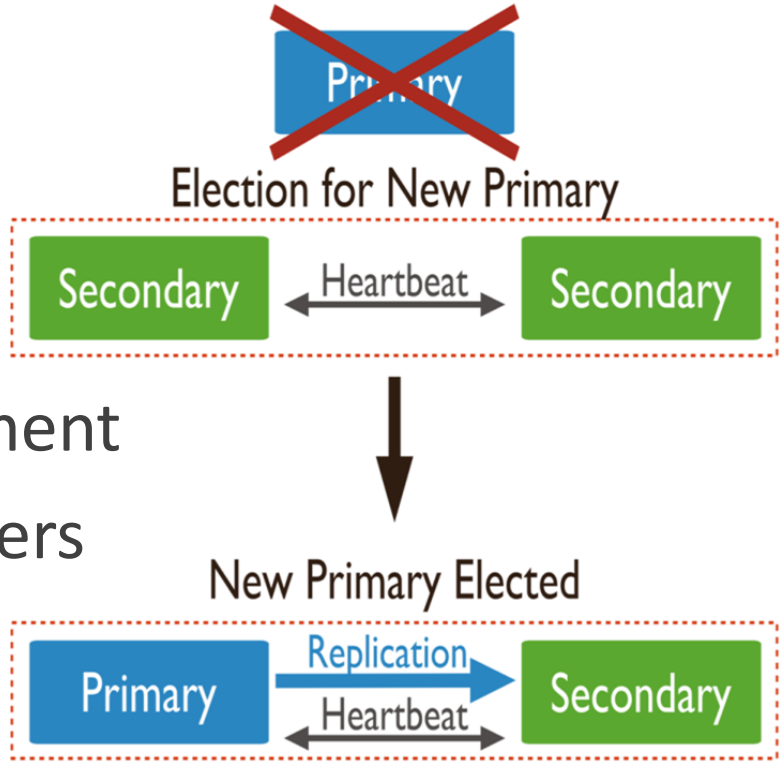
Replica Set Members

- Primary
  - Read, Write operations
- Secondary
  - Asynchronous Replication
  - Can be primary
- Arbiter
  - Voting
  - Can’t be primary
- Delayed Secondary
  - Can’t be primary



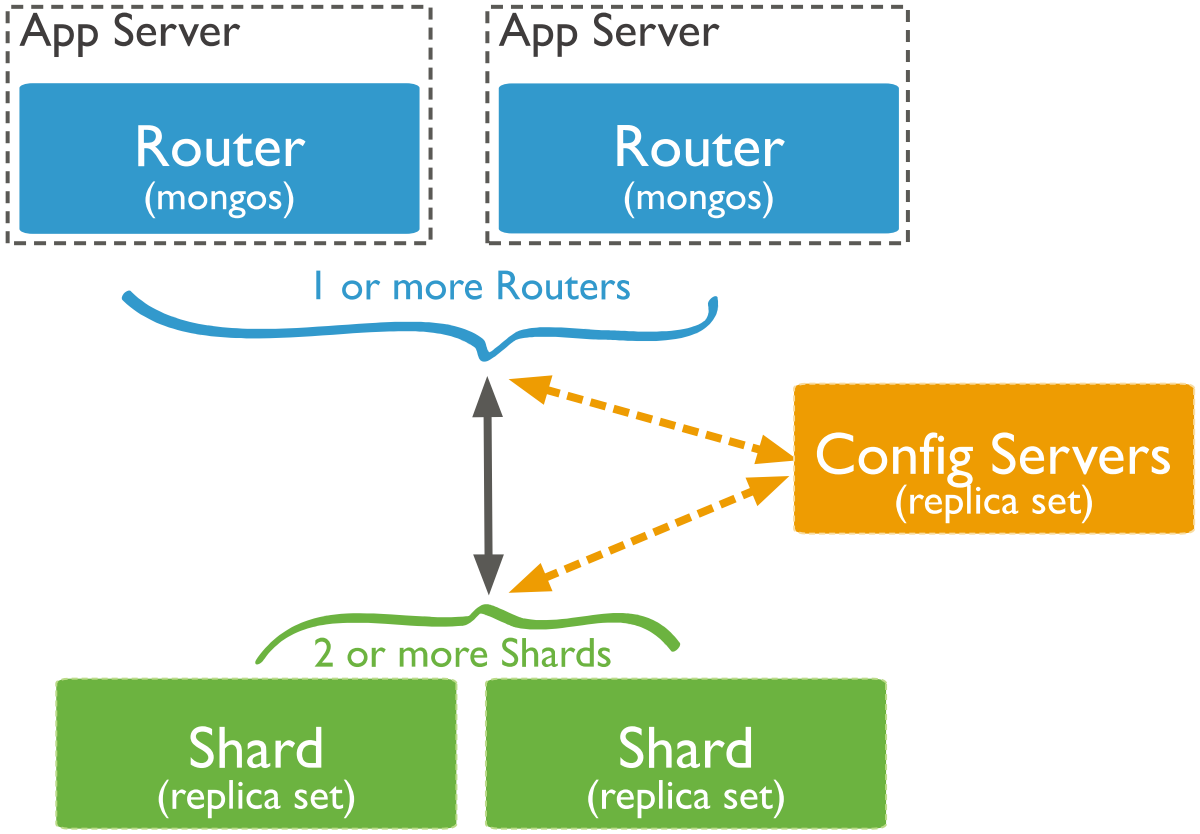
Replication in MongoDB

- Automatic Failover
  - Heartbeats
  - Elections
- The Standard Replica Set Deployment
- Deploy an Odd Number of Members
- Rollback
- Security
  - SSL/TLS



Sharding

A method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.


App Server
Router
(mongos)
App Server
Router
(mongos)
1 or more Routers
Config Servers
(replica set)
2 or more Shards
Shard
(replica set)
Shard
(replica set)

MongoDB Sharded Cluster

A MongoDB sharded cluster consists of the following components:

shard: Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set.

mongos: The mongos acts as a query router, providing an interface between client applications and the sharded cluster.

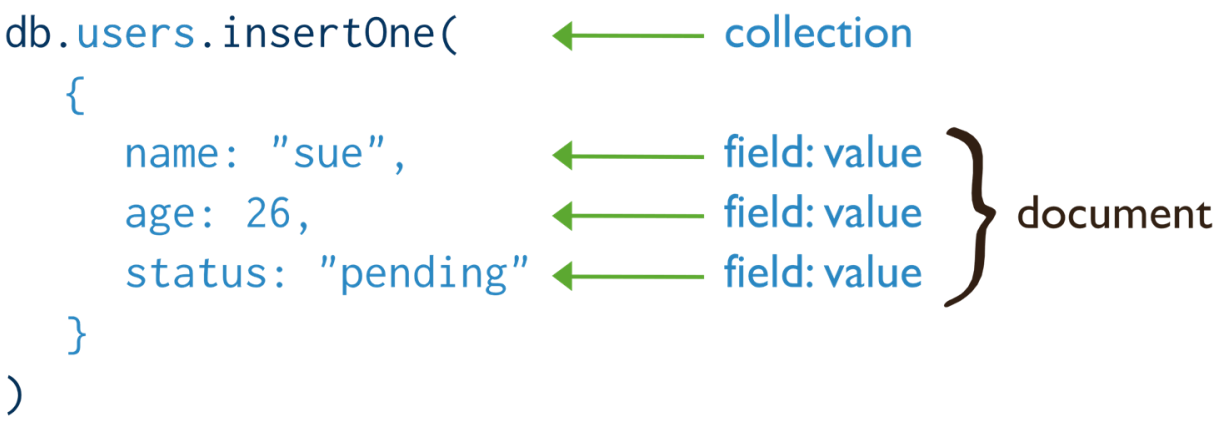
config servers: Config servers store metadata and configuration settings for the cluster. As of MongoDB 3.4, config servers must be deployed as a replica set (CSRS).

Create | Read | Update | Delete

CRUD

Insert Document | Query Document | Update Document | Remove Document

Insert Document



Insert Document

To insert documents into a collection/make a new collection:

mongo:

```
db.<collection>.insert(<document>)
```

sql:

```
INSERT INTO <table> VALUES(<attributevalues>);
```

Insert Document

Insert one document:

```
db.<collection>.insert({<field>:<value>})
```

Inserting a document with a field name new to the collection is inherently supported by the BSON model.

To insert multiple documents, use an array.

Insert Document

Insert One:

```
db.inventory.insertOne(
   { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }
)
```

Insert Many:

```
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" } }
])
```

Query Document

Done on collections.

Get all docs: `db.<collection>.find()`

Returns a cursor, which is iterated over shell to display first 20 results.

Add .limit(<number>) to limit results

```
SELECT * FROM <table>;
```

Get one doc: `db.<collection>.findOne()`

Query Document

```
db.inventory.find( { status: "D" } )
```

=

```
SELECT * FROM inventory WHERE status = "D"
```

Query Document

To match a specific value:

```
db.<collection>.find({<field>:<value>})
```

AND

```
db.<collection>.find({<field1>:<value1>,
                      <field2>:<value2>
                      })
```

sql:

```
SELECT * FROM <table> WHERE <field1> = <value1> AND <field2> =
<value2>;
```

Query Document

OR

```
db.<collection>.find({ $or:
[<field>:<value1><field>:<value2>]})
```

```
SELECT *
FROM <table>
WHERE <field> = <value1> OR <field> = <value2>;
```

Checking for multiple values of same field:

```
db.<collection>.find({<field>: {$in [<value>, <value>]}})
```

Query Document

Including/excluding document fields:

```
db.<collection>.find({<field1>:<value>}, {<field2>: 0})
```

```
SELECT field1 FROM <table>;
```

```
db.<collection>.find({<field>:<value>}, {<field2>: 1})
```

Find documents with or w/o field:

```
db.<collection>.find({<field>: { $exists: true}})
```

Query Document Query Selectors (comparison):

$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$in	Matches any of the values specified in an array.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$nin	Matches none of the values specified in an array.

https://docs.mongodb.com/

Query Document Query Selectors (logical):

$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
$not	Inverts the effect of a query expression and returns documents that do *not* match the query expression.
$nor	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

https://docs.mongodb.com/

Query Document Query Selectors (array):

$all	Matches arrays that contain all elements specified in the query.
$elemMatch	Selects documents if element in the array field matches all the specified $elemMatch conditions.
$size	Selects documents if the array field is a specified size.

https://docs.mongodb.com/

Update Document

```
db.collection.update(
   <query>,
   <update>,
   {
     upsert: <boolean>,
     multi: <boolean>,
     writeConcern: <document>,
     collation: <document>,
     arrayFilters: [ <filterdocument1>, ... ]
   }
)
```

Update Document

```
db.<collection>.update(
{<field1>:<value1>},                    //all which field = value
{$set: {<field2>:<value2>}},            //set field to value
{multi:true} )                          //update multiple docs
```

```
UPDATE <table> SET <field2> = <value2>
WHERE <field1> = <value1>;
```

Update Document

To remove a field

```
db.<collection>.update({<field>:<value>},
                       { $unset: { <field>: 1}})
```

Replace all field-value pairs

```
db.<collection>.update({<field>:<value>},
                       { <field>:<value>,
                        <field>:<value>})
```

*NOTE: This overwrites ALL the contents of a document, even removing fields.

Update Document

Syntax:

```
db.collection.update(filter, update, options)
```

Example:

Updates first document with this criteria:

```
db.students.update(
        {name: 'Andy Joya'},
        {$set: {age: 11, class: '5B'} }
)
```

Update multi:

```
db.students.update(
        {class: '5B'},{$set: {age: 11}}, {multi:true})
```

updateOne vs updateMany

updateOne

Syntax:

```
db.collection.updateOne(filter, update, options)
```

Example:

```
db.students.updateOne({name: 'Andy Joya'},{$set: {age: 11,
class: '5B'}})
```

updateMany

Syntax:

```
db.collection.updateMany(filter, update, options)
```

Example:

```
db.students.updateMany({age: 11},{$set: {class: '5C'}});
```

Remove Document

Remove all records where field = value

```
db.<collection>.remove({<field>:<value>})
```

```
DELETE FROM <table> WHERE <field> = <value>;
```

As above, but only remove first document

```
db.<collection>.remove({<field>:<value>}, true)
```

Remove Document

Syntax:

```
db.collection.remove(query, justOne)
```

justOne is Boolean: set to one removes only one document.

Example:

```
db.students.remove({'name': 'Andy Joya'})
```

INSTALL & FIRST COMMANDS

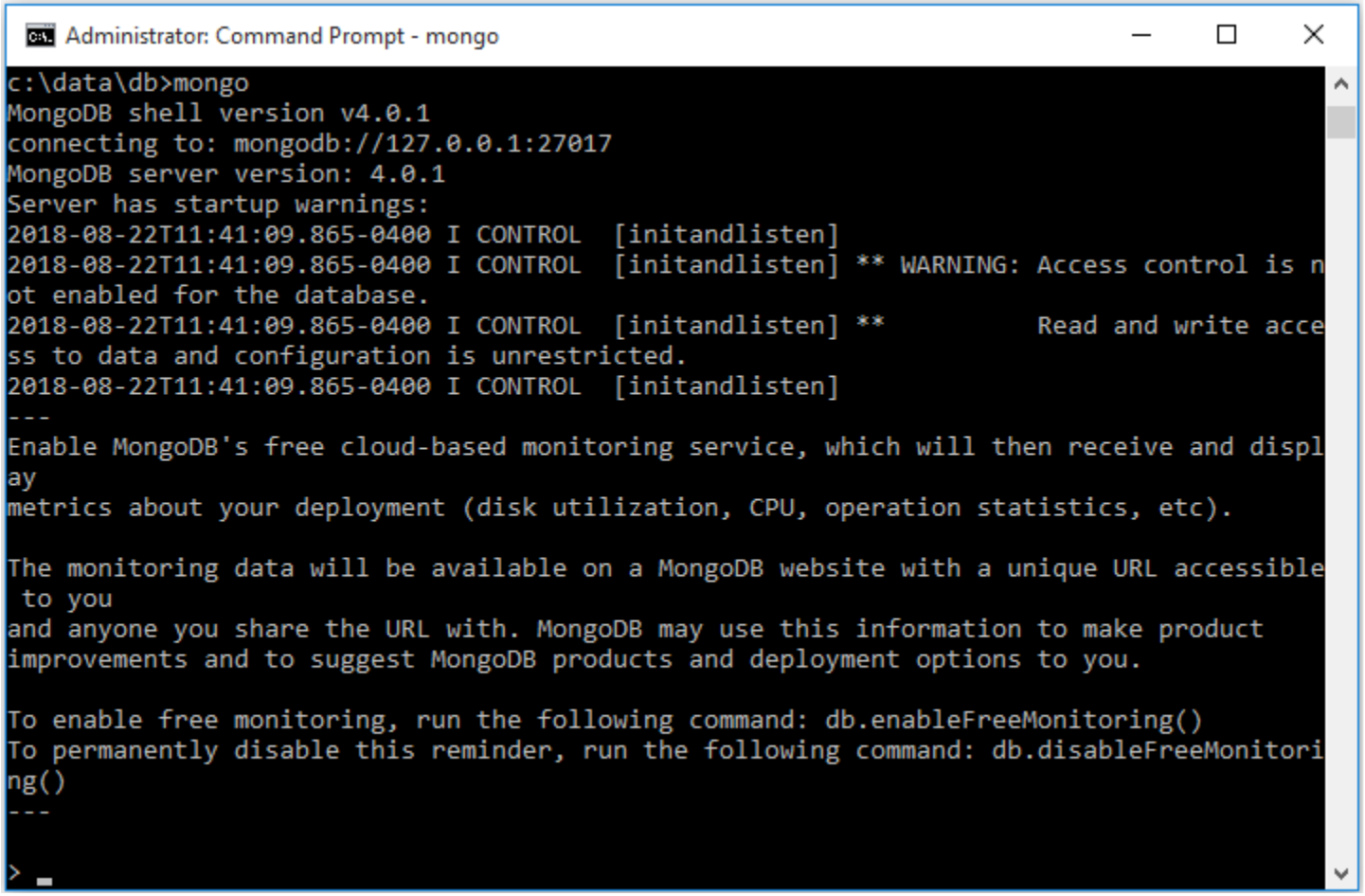
Install

Download

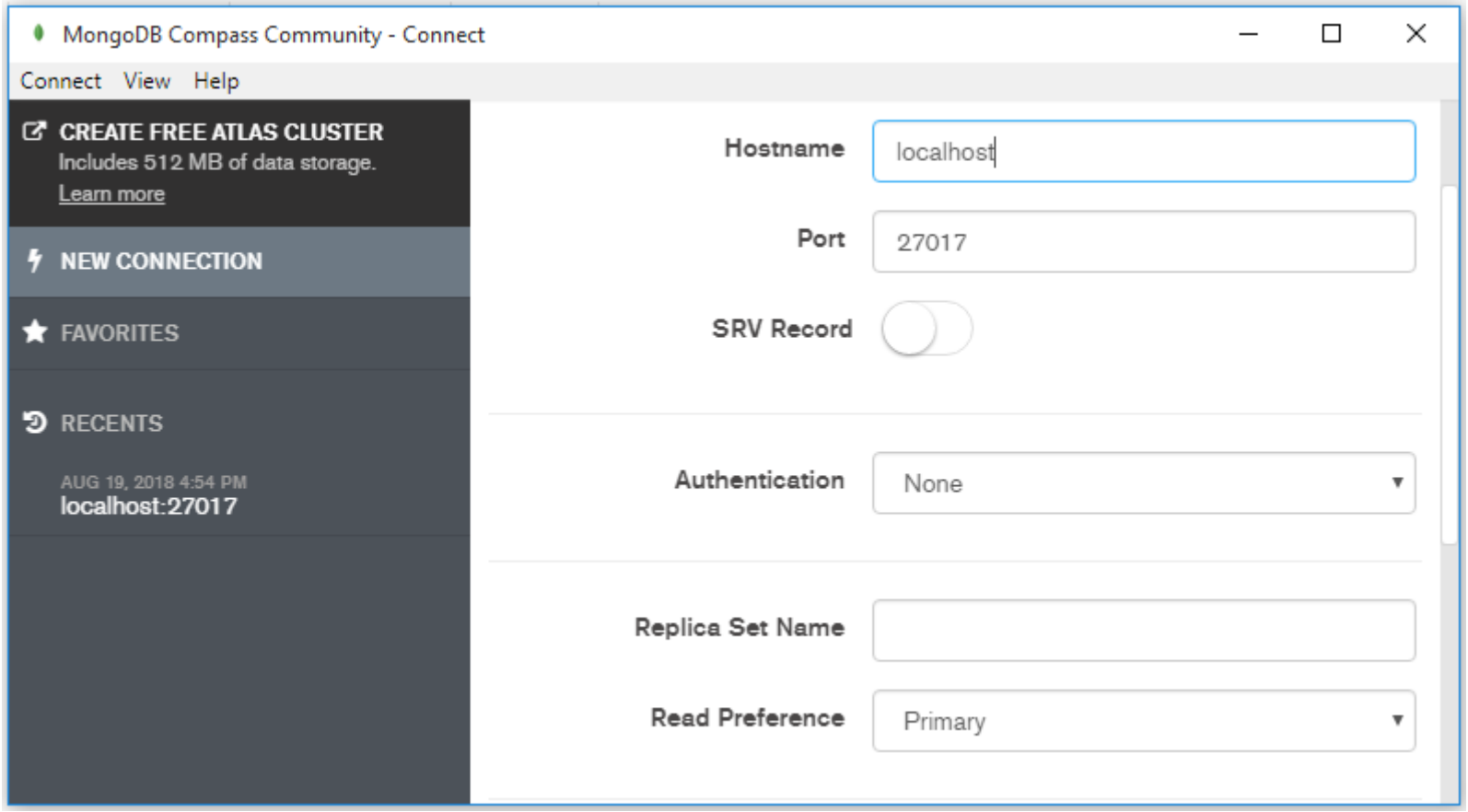
https://www.mongodb.com/download-center#community

Installation folder: “c:\mongodb”

MongoDB Shell



MongoDB Compass



Show Databases

> show dbs;

Create Database

- Syntax of Create Database

```
> use DATABASE_NAME;
```

- Example

```
> use school;
```

Show Collections

The mongodb show collections command returns all the collections of the current database:

```
> show collections;
```

Drop Database

The dropDatabase() command is used to drop an existing database.
Syntax of Drop Database

```
> DATABASE.dropDatabase();
```

Example

```
> newdb.dropDatabase();
```

MongoDB Delete Document

Syntax:

```
db.collection.remove(query, justOne)
```

justOne is Boolean: set to one removes only one document.

Example:

```
db.students.remove({'name': 'Andy Joya'})
```

MongoDB Create User

Criar User com roles: readWrite e dbAdmin:

```
> db.createUser({
        user:"aedb",
        pwd: "1234",
        roles: ["readWrite","dbAdmin" ]
 })
```

MongoDB Operators

Operators	Symbols
<	$lt
<=	$lte
>	$gt
>=	$gte

Syntax:

```
{field: {$lt: value} }
```

Example of $lt:

To get the students less than 11 years -

```
db.students.find({"age" : {"$lt" : 11}});
```

FE05

http://nicholasjohnson.com/mongo/course/workbook/

Introdução ao MongoDB

AULA PL06

Hugo Peixoto

2019 – 2020 Universidade do Minho