



mongoDB

# The Humongous Database

Geoffrey Berard / [@geofberard](#)

[geoffrey.berard@gmail.com](mailto:geoffrey.berard@gmail.com)

[bit.ly/3oH4mMb](https://bit.ly/3oH4mMb) - pdf

# Plan (07/12/2022)

## I. MongoDB Presentation

- Schemaless
- BSon
- Document Model
- Performance

## II. MongoDB CRUD

- CREATE
- READ
- UPDATE
- DELETE

## III. Practical

# Plan (09/12/2022)

## VI. MongoDB Architecture

- Replica Set
- Sharding

## V. Software with MongoDB

- Architecture

## VI. Practical



# Presentation

07/12/2022

# MongoDB

## Technology

- Document oriented Data Base
- Non relational
- Dynamic : Schemaless
- No query language
- No transactions (by default)
- Atomic operation on document
- Scalable : Auto Sharding
- Made in C++

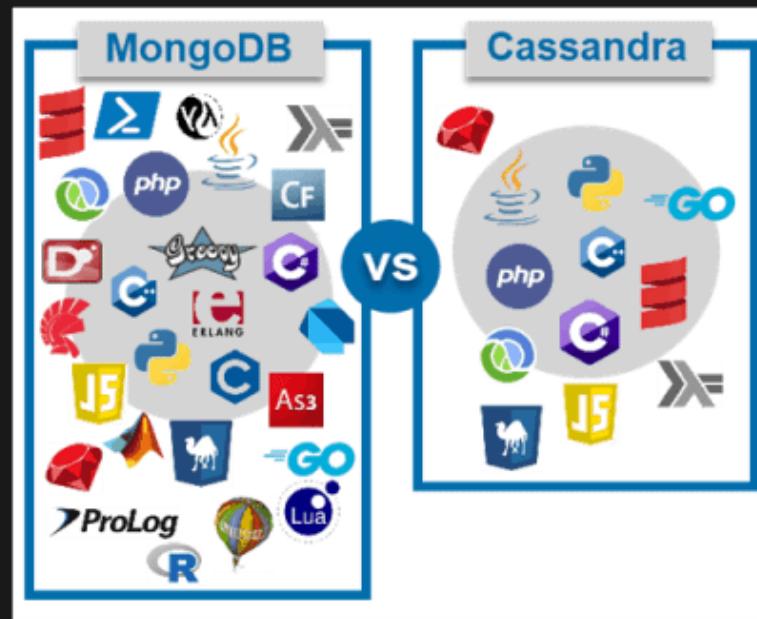
# MongoDB

## Functionnalities

- Text Search
- GeoSearch
- Aggregation
- Map-Reduce

# MongoDB

Supported languages



# JSON Document

```
{  
    "_id": 1,  
    "first_name": "Paul",  
    "surname": "McCartney",  
    "instruments": [  
        "Guitar",  
        "Bass guitar",  
        "Piano"  
    ],  
    "address": {  
        "street": "20 Forthlin Road",  
        "city": "LiverPool",  
        "zip": "United Kingdom"  
    }  
}
```

# JSON Document

Available Data Types :

- Array
- Object
- String
- Number
- Boolean
- Null

# Schemaless

Exemple RDBMS - Musiciens

<b>first_name</b>	<b>last_name</b>	<b>birthday</b>
Louis	Armstrong	4/08/1901

# Schemaless

Exemple RDBMS - Alter table

first_name	last_name	title	birthday
Louis	Armstrong		4/08/1901
Paul	McCartney	Sir	18/06/1942

# Schemaless

Exemple RDBMS - Holed Table

first_name	last_name	title	nickname	birthday
Louis	Armstrong			04/08/1901
Paul	McCartney	Sir		18/06/1942
Gordon Matthew Thomas	Summer		Sting	02/10/1951

# Schemaless

Exemple - MongoDB

```
[  
  {  
    "first_name": "Louis",  
    "surname": "Armstrong",  
    "birthday": "4 août 1901"  
  },  
  {  
    "first_name": "Paul",  
    "surname": "McCartney",  
    "title": "Sir",  
    "birthday": "18 June 1942"  
  },  
  {  
    "first_name": "Gordon Matthew Thomas",  
    "surname": "Sumner",  
    "age": 50  
  }]
```

# Schemaless

Be carefull

```
[  
 {  
   "first_name": "Louis",  
   "surname": "Armstrong",  
   "birthday": "4 août 1901"  
 },  
 {  
   "first_name": 42  
 }  
 ]
```

# Schemaless

Be carefull



# BSON

Binary Representation of JSON - 16MB Maximum

```
{  
  "hello": "world"  
}
```

Gives in BSON

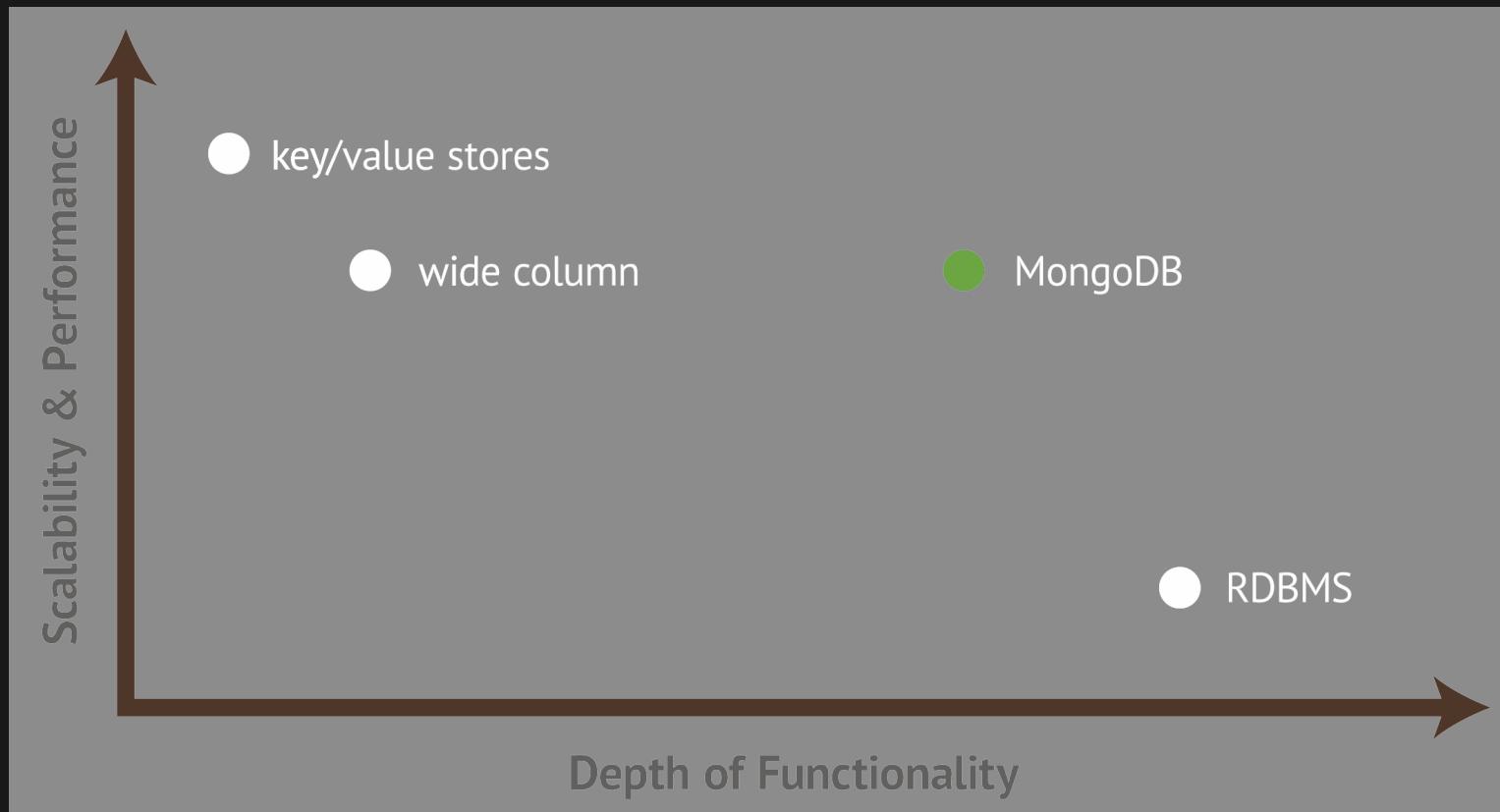
```
\x16\x00\x00\x00\x02hello\x00\x06\x00\x00\x00world\x00\x00
```

# BSON

## Enriched Types

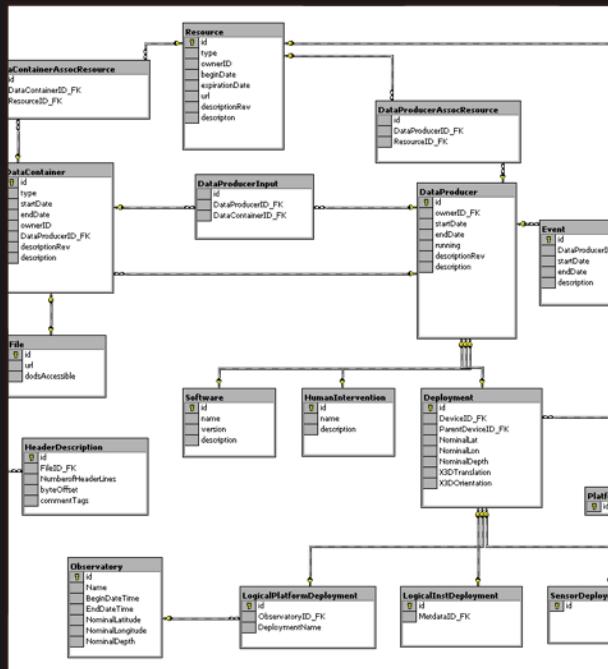
Type	Type
Double	Regular expression
String	JavaScript
Object	Symbol
Array	32-bit integer
Binary data	Timestamp
Object ID	64-bit integer
Boolean	Min key
Date	Max key
Null	

# Position



# Relational VS Document

## Models Differences



# Relational VS Document

## Relational Model

**PERSON**

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rome

NO RELATION

**CAR**

Car_ID	Model	Year	Value	Pers_ID
101	Bently	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

# Relational VS Document

## Document Model

```
{  
  "first_name": "Paul",  
  "surname": "Miller",  
  "city": "London",  
  "cars": [  
    {  
      "model": "Bentley",  
      "year": 1973,  
      "value": 100000  
    },  
    {  
      "model": "Rolls Royce",  
      "year": 1965,  
      "value": 330000  
    }  
  ]  
}
```

# Relational VS Document

## Alternative Document Model

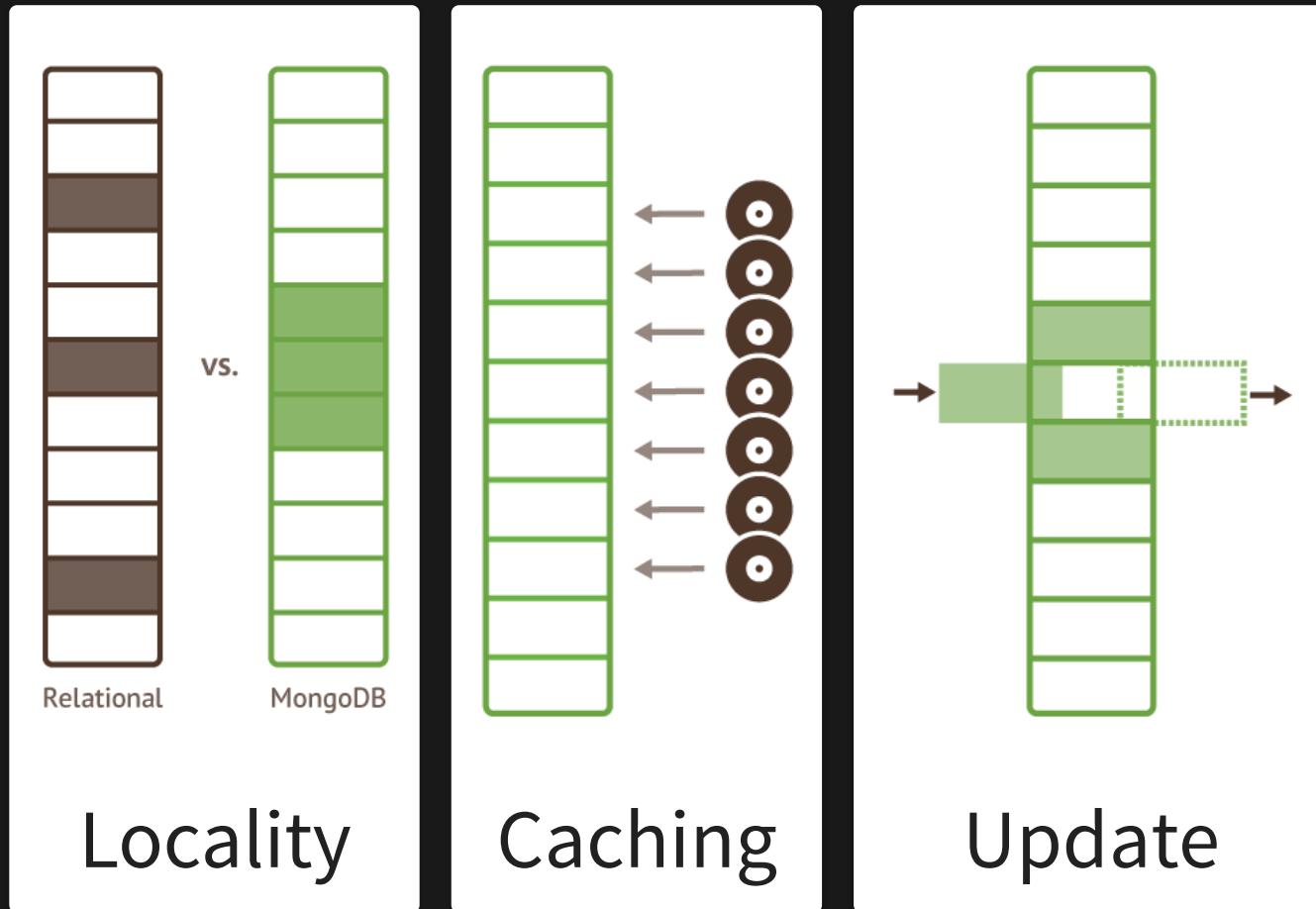
```
{  
  "model": "Bentley",  
  "year": 1973,  
  "value": 100000,  
  "owner": {  
    "first_name": "Paul",  
    "surname": "Miller",  
    "city": "London"  
  }  
}
```

# Relational VS Document

## Terminology

RDBMS	Mongo
Table, View	Collection
Row(s)	JSON Document
Index	Index
Join	Embedded Document
Partition	Shard
Partition Key	Shard Key

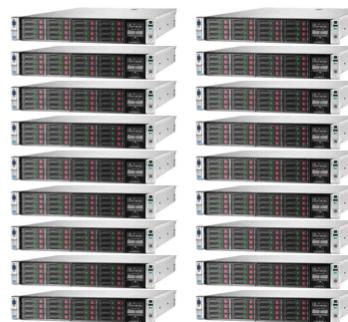
# Performances



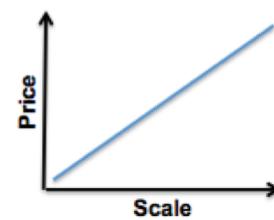
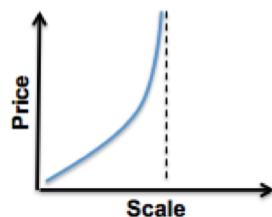
# Scalability



Vertical



Horizontal



# Benefits

## Efficient

- Super low latency
- Scale Easily

## Agility and flexibility

- Data models can evolve easily
- Companies can adapt to changes quickly

## Intuitive, natural data representation

- Developers are more productive
- Many types of applications are a good fit

## Reduces the need for joins, disk seeks

- Programming is more simple
- Performance can be delivered at scale



# Schemaless

Exemple RDBMS - Alter table

	<b>SQL</b>	<b>MongoDB</b>
Create	Insert	Insert
Read	Select	Find
Update	Update	Update
Delete	Delete	Remove

# Schemaless

Query language => methods of objects

# Identifier

```
{  
  "_id": 1,  
  "first_name": "Victor",  
  "surname": "Hugo",  
  "groups": [  
    "Writer",  
    "Painter"  
  ]  
}
```

- Unique
- Can't be changed

# Create

## Insert

```
> db.member.insert({new_document})
```

```
WriteResult({ "nInserted" : 1 })
```

Create the collection if necessary

```
> db.member.insert({first_name: "John", last_name: "Doe"})  
> db.member.insert({first_name: "Jean", last_name: "Dupont", city_of_birth: "Paris"})
```

# Create Index

Syntax ([Index types](#))

```
createIndex( { userid: index_type } )
```

```
> db.records.createIndex( { userid: 1 } )    // Ascending index
> db.records.createIndex( { userid: -1 } )   // Descending index
```

# Create

## mongoimport

Test Only

Supported file format

- Json
- CSV
- TSV

```
$ ./mongoimport --db test --collection zips --file ../../../../Downloads/zips.json
```

[Download zips.json](#)

# Read

## Find

## Returning all elements of a collection:

```
> db.member.find()
```

Returning only the first element:

```
> db.member.findOne()
```

# Read

## Find

Formating the result :

```
> db.member.find().pretty()
```

```
{
  "_id": "ObjectId(\"54853dd6dd8fc0fec931fcbe\")",
  "first_name": "John",
  "last_name": "Doe"
}
```

# Read

## Find

### Syntax

```
<collection>.find({query}, {keys_filter})
```

### Example

```
> db.zips.find({state:"NY"},{city:true, _id:false})
```

```
{ city: "FISHERS ISLAND" }
{ city: "NEW YORK" }
{ city: "NEW YORK" }
{ city: "NEW YORK" }
{ city: "GOVERNORS ISLAND" }
```

# Read

## Querying

Greater than (\$gt, \$gte)

```
{ pop : {$gt : 100000} }
```

Lower than (\$lt, \$lte)

```
{ pop : {$lte : 100} }
```

Regular Expression (\$regex)

```
{ city : {$regex : "^\wA"} }
```

# Read

## Querying

Value Exists (\$exists)

```
{ city : {$exists : true} }
```

Value Type (\$type) ([Type codes](#))

```
{ city : {$type : type_code} }
```

# Read

## Wrong Query

Overriding query property

```
{ pop: {$gt: 10000}, pop: {$lt: 50000}}
```

is equivalent to

```
{ pop: {$lt: 50000}}
```

# Read

## Array Query

### Natural

```
{ groups : "Painter" }
```

In (\$in)

```
{ groups : { $in : [ "Writer", "Sculptor", "Dancer" ] } }
```

All (\$all)

```
{ groups : { $all : [ "Painter", "Writer" ] } }
```

## Can return

```
{  
    "first_name": "Victor",  
    "surname": "Hugo",  
    "groups": [  
        "Writer",  
        "Painter"  
    ],  
    "address": {  
        "number": 6,  
        "street_name": "Place des Vosges",  
        "city": "Paris",  
        "zip": "75004"  
    }  
}
```

# Read

## Nested Document Query

```
{  
  "address": {  
    "number": 6,  
    "street_name": "Place des Vosges",  
    "city": "Paris",  
    "zip": "75004"  
  }  
}
```

or

```
{ "address.city" : "Paris" }
```

## Can return

```
{  
    "first_name": "Victor",  
    "surname": "Hugo",  
    "groups": [  
        "Writer",  
        "Painter"  
    ],  
    "address": {  
        "number": 6,  
        "street_name": "Place des Vosges",  
        "city": "Paris",  
        "zip": "75004"  
    }  
}
```

# Read

## Wrong Queries

Incomplete object description

```
{ address : { city : "Paris" }})
```

Document key in the wrong order

```
{
  "address": {
    "street_name": "Place des Vosges",
    "number": 6,
    "city": "Paris",
    "zip": "75004"
  }
}
```

# Read

## Combined Query

Natural

```
{ pop : { $gt : 100000, $lt : 2000000} }
```

Or (\$or)

```
{ $or : [ { state : "NY" } , { state : "NJ" } ] }
```

And (\$and)

```
{ $and : [ { state : "NY" } , { pop : { $gt : 50000} } ] }
```

## Query combinaison combinaison ???

```
{$and:[ {$and:[ {city:{regex: "^N"}}, {$or:[ {state: "NY"}, {state: "NJ"} ]} ]},  
{pop:{$gt:100000,$lt:150000}}]}
```

# Read

Query combinaison combinaison ???

```
{$and:[ {$and:[ {city:{regex: "^N"}},{$or:[ {state:"NY"},{state:"NJ"}]} ]},  
{pop:{$gt:100000,$lt:150000}}]}
```

```
{  
  $and: [  
    {  
      $and: [  
        { city: { regex: " ^N" } },  
        { $or: [{ state: "NY" }, { state: "NJ" } ] } ,  
      ],  
    },  
    { pop: { $gt: 100000, $lt: 150000 } },  
  ],  
};
```

## Query combinaison combinaison ???

```
{ $and: [ { $and: [ { city: { regex: "^\u004e" } }, { $or: [ { state: "NY" }, { state: "NJ" } ] } ] },
{ pop: { $gt: 100000, $lt: 150000 } } ] }
```

```
{
  $and: [
    {
      $and: [
        { city: { regex: "^\u004e" } },
        { $or: [ { state: "NY" }, { state: "NJ" } ] }
      ],
    },
    { pop: { $gt: 100000, $lt: 150000 } },
  ],
};
```

Cities starting with "N" in New York or New Jersey with a population between 100k and 150k inhabitants

# Read Count

```
> db.zips.count({find_query})
```

## Example

```
> db.zips.count({state:"NY"})
```

## Cursor

```
> cursor = db.zips.find({state:"MA"},{city:true, _id:false}); null;
```

Iterating over results :

```
> cursor.hasNext()    // > true
> cursor.next()       // > "AGAWAM"
> cursor.next()       // > "CUSHMAN"
```

# Read

## Operation on cursor

```
> cursor.sort({city : -1})      // Sort in reverse alphabetical order  
> cursor.limit(5)            // Limit the number of results to 5  
> cursor.skip(3)             // Skip 3 elements before returning the result
```

They can be combined

```
> cursor.sort({city : -1}).limit(5).skip(3)
```

# Update

## Syntax

```
<collection>.update( {find_query} , {update_query}, {update_params} )
```

## Example

```
update({surname : "Hugo"}, {surname : "Hugo", groups : [ "Writer", "Painter"]})
```

## On document

```
{  
  "first_name": "Victor",  
  "surname": "Hugo",  
  "address": {  
    "number": 6,  
    "street_name": "Place des Vosges",  
    "city": "Paris",  
    "zip": "75004"  
  }  
}
```

Will give

```
{  
  "surname": "Hugo",  
  "groups": [  
    "Writer",  
    "Painter"  
  ]  
}
```

# Update

Ajouter/Modifier des champs (\$set)

```
{$set : {groups : [ "Writer", "Painter"]}}
```

Supprimer des champs (\$unset)

```
{$unset : {groups : 1}}
```

Modify Key (\$unset)

```
 {$rename : {"oldName" : "newName"}}
```

# Update

Modify (number)

```
{$inc : {count : 1}}
```

```
{$mul : {price : NumberDecimal("0.5")}}
```

Compare to current value (number, date)

```
{$min : {bestPrice : 150}}
```

```
{$max : {highScore : 1275000}}
```

# Update

## Array manipulation

Change Value

```
{"groups.2" : "Poet"}
```

Add element (\$push)

```
{$push : {groups : "Poet"}}
{$pushAll : {groups : ["Poet","Politician"]}}
```

Remove element (\$pop)

```
{$pop : {groups : 1}}    // remove last element
{$pop : {groups : -1}}   // remove first element
```

# Update

## Array manipulation

Remove specific element (\$pull)

```
{$pull : {groups : "Poet"}}
{$pullAll : {groups : ["Poet", "Politician"]}}
```

Take array as a set (\$addToSet)

```
{$addToSet : {groups : "Poet"} } // Add "Poet"
{$addToSet : {groups : "Poet"} } // Do nothing because exists
```

# Update

Multiple update  
=>Upsert

```
{ upsert : true }      // As update third parameter
```

```
> db.member.update({surname : "Washington"},  
                  { $set : {groups : [ "Writer", "Painter" ]} },  
                  { upsert : true })
```

# Update

Update or insert

```
{ multi : true }      // As update third parameter
```

```
> db.member.update({},
  { $set : {title : "Mr"}},
  { multi : true })
```

```
WriteResult({ "nMatched" : 5, "nUpserted" : 0, "nModified" : 5 })
```

# Delete

## Syntax

```
<collection>.remove({find_query})
```

## Drop a collection

```
<collection>.drop()
```

# Debugging

```
> db.zips.explain().find({query});
```

or

```
> cur = db.zips.find({query}); null;  
> cur.explain()
```

# Debugging

## Example

```
> db.zips.explain().find({state:"MA"},{city:true, _id:false}).sort({city : -1}).li
```

```
{
  "queryPlanner": {
    "plannerVersion": 1,
    "namespace": "test.zips",
    "indexFilterSet": false,
    "parsedQuery": {
      "state": {
        "$eq": "MA"
      }
    },
    "serverInfo": {
      "host": "localhost",
      "port": 27017,
      "version": "4.0.4",
      "gitVersion": "f288a3bdf201007f3693c58e140056adf8b04839"
    }
  }
}
```

# Debugging

## Execution detail

```
> cur = db.zips.explain("executionStats").find({state:"MA"},{city:true, _id:false})
```

```
{
  "queryPlanner": {
    "namespace": "test.zips"
  },
  "executionStats": {
    "executionSuccess": true,
    "nReturned": 5,
    "executionTimeMillis": 20,
    "totalKeysExamined": 0,
    "totalDocsExamined": 29353
  },
  "serverInfo": {
    "host": "it-gbe",
    "port": 27017,
    "version": "3.2.11",
  }
}
```

# Advanced

## Distinct ([details](#))

```
> db.zips.distinct({field} , {search_query})
```

```
> db.zips.distinct("state" , {})  
[  
    "MA" ,  
    "RI" ,  
    "NH" ,  
    ...  
]
```

# Advanced

## Geospatial (details)

```
> db.zips.createIndex( { loc : "2d" } )  
> db.zips.find( { "loc": { $near : [ -112.416728, 37.781334 ] } } ).limit(5)
```

```
{ "_id" : "84759", "city" : "PANGUITCH", "loc" : [ -112.436886, 37.80777 ], "pop" : 1  
{ "_id" : "84710", "city" : "ALTON", "loc" : [ -112.548389, 37.469905 ], "pop" : 1  
{ "_id" : "84760", "city" : "PARAGONAH", "loc" : [ -112.773972, 37.89172 ], "pop" : 1  
{ "_id" : "84717", "city" : "BRYCE CANYON", "loc" : [ -112.074311, 37.608427 ], "pop" : 1  
{ "_id" : "84761", "city" : "PAROWAN", "loc" : [ -112.832251, 37.844861 ], "pop" : 1
```

# Advanced

## Aggregate (details)

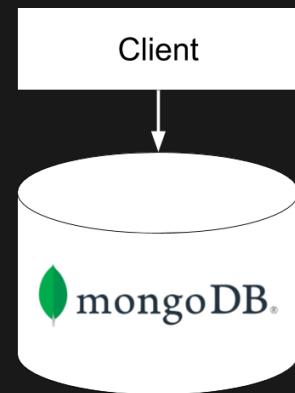
```
> db.zips.aggregate([{ $group: {group} } , { $match: {group} }])  
> db.zips.aggregate([{ $group: { _id: "$city", totalPop: { $sum: "$pop" } } }])
```

```
{ "_id" : "WRANGELL", "totalPop" : 2573 }  
{ "_id" : "SKAGWAY", "totalPop" : 692 }  
{ "_id" : "THORNE BAY", "totalPop" : 744 }  
...
```

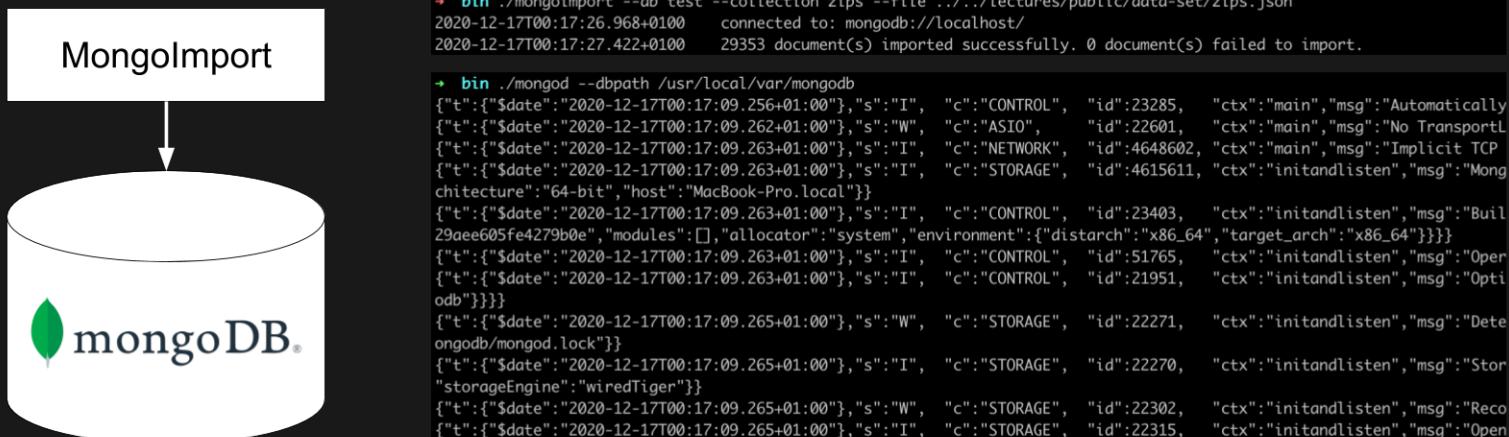


Practical - CRUD

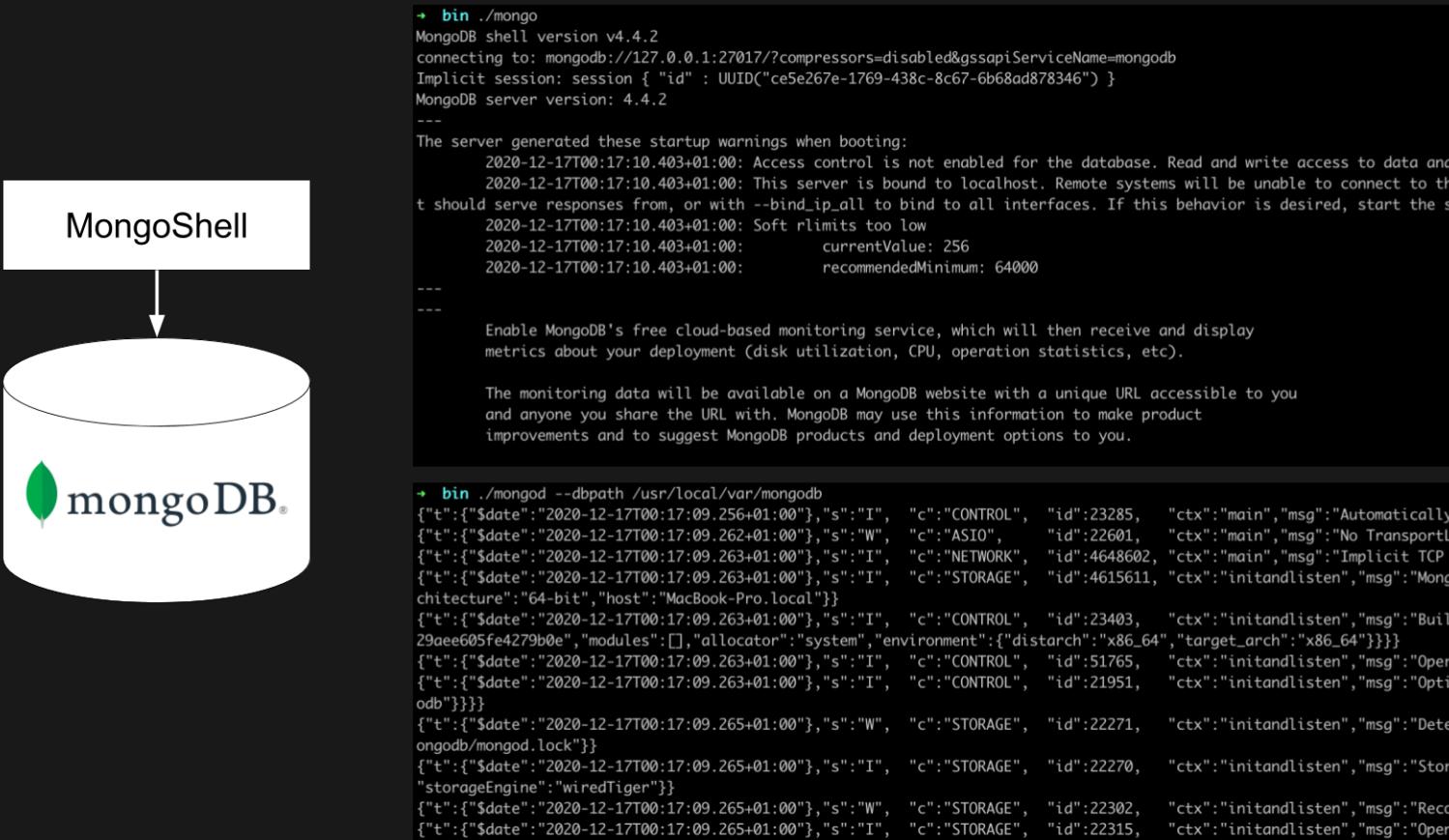
# Architecture



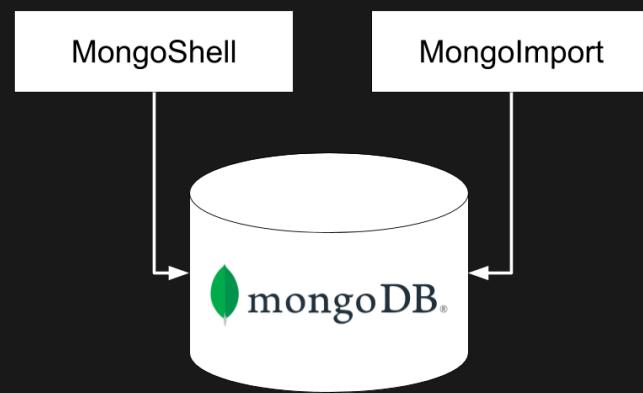
# Architecture



# Architecture



# Architecture



# Install MongoDB

## Run with Docker

Create container and start MongoDB Server

```
$ docker run --name mongo-server mongo
```

Restart server

```
$ docker start -i mongo-server
```

# Install MongoDB

In another shell, run MongoShell (client)

```
$ docker exec -it mongo-server mongosh
```

## Run with Docker

Import a json file in the container

```
$ docker cp <path>/file.json mongo-server:/tmp/<file>.json
```

Load it into the database

```
$ docker exec mongo-server mongoimport --db test --collection emails --file /tmp/<
```

# Install MongoDB

## Linux & MacOS

Documentation [here](#)

Download mongodb zip [here](#)

Unzip and go in the file

```
$ cd path_to_downloaded_file  
$ tar xvf mongodb-macos-x86_64-4.4.2.tgz
```

Create the storage directory

```
$ sudo mkdir -p /usr/local/var/mongodb  
$ sudo chmod 777 /usr/local/var/mongodb
```

# Install MongoDB

## Linux & MacOS

Go to the directory

```
$ cd mongodb-macos-x86_64-4.4.2/bin
```

Run MongoDB Daemon (server)

```
$ ./mongod --dbpath /usr/local/var/mongodb
```

Run MongoShell (client) in an other shell

```
$ ./mongo
```

# Install MongoDB Tools

## Linux & MacOS

Documentation [here](#)

Download zip and unzip it

```
$ unzip mongodb-database-tools-*.zip
```

Go to the bin directory

```
$ cd mongodb-macos-x86_64-4.4.2/bin
```

run mongoimport

```
$ ./mongoimport <your params>
```

# Install MongoDB

## Windows

Installation procedure is detailed [here](#)

Download mongodb zip [here](#)

Install it

Create the storage directory

```
$ cd C:\  
$ md "\data\db"
```

# Install MongoDB

## Windows

Go to the directory

```
$ cd "C:\Program Files\MongoDB\Server\4.2\bin\"
```

Run MongoDB Daemon (server)

```
$ mongod.exe --dbpath="c:\data\db"
```

Run MongoShell (client) in an other cmd

```
$ mongo.exe
```

# Install MongoDB Tools

Windows

Installation procedure is detailed [here](#)

# Mongo Shell

## Emails

- I.1. Import the [enron.json](#) in the collection "emails" [[Shell command](#)]
- I.2. What is the total amount of emails ? [[Query + Result](#)]
- I.3. What is the amount of emails in inbox ? [[Query + Result](#)]
- I.4. List the emails sent from domain yahoo.com [[Query](#)]
- I.5. How long took the last request [[Request + Time](#)]

# Mongo Shell

## Emails

- I.6. Add an index the right field to make the last request run faster [Index Query]
- I.7. How long took the last request with the index [Time]
- I.8. Find only dates and subjects of all messages sent by mike.mcconnell@enron.com [Query]
- I.9. Remove rosalee.fleming@enron.com from all the email recipient [Query]
- I.10. Add rob.bradley@enron.com as recipient  
to all emails sent by rosalee.fleming@enron.com [Query]

# Mongo Shell

## ZIP Codes

Import the [zips.json](#) in the collection "zips" [Shell command]

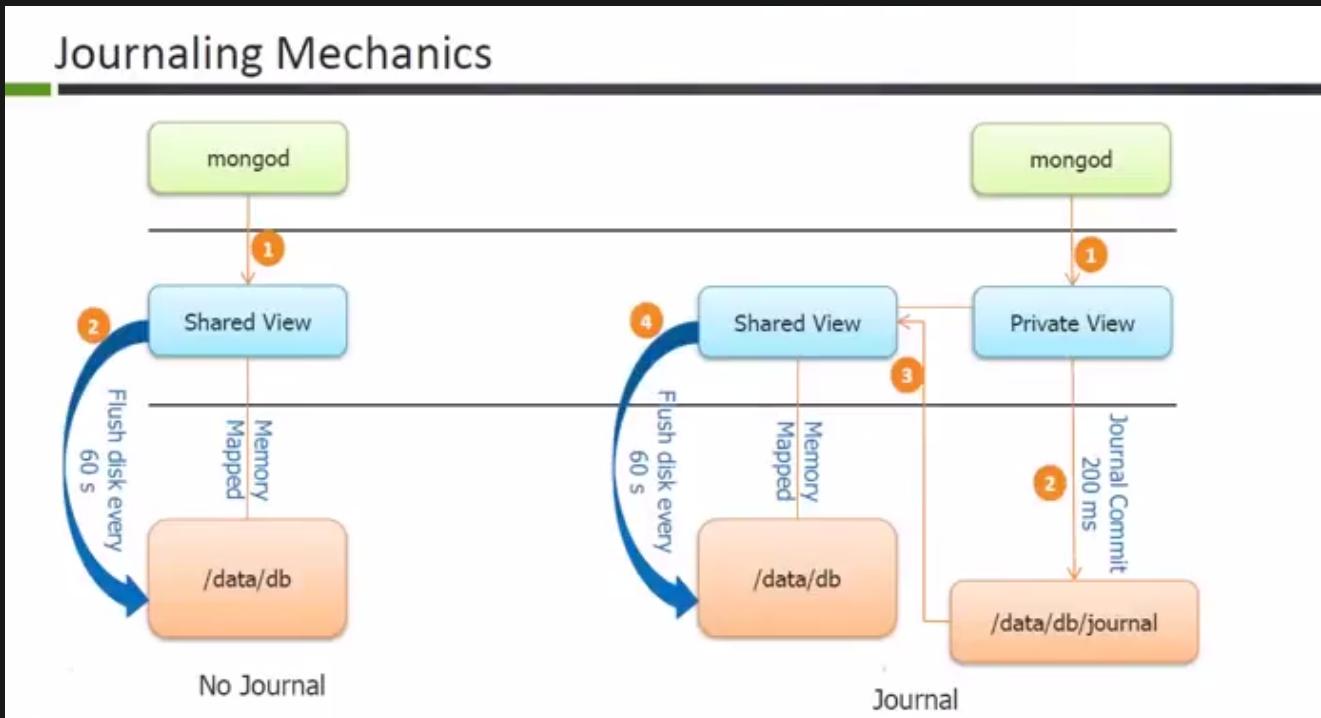
- II.1. List the 10 most populated zones in California and Louisiana [Query]
- II.2. Then most populated zones in California and Louisiana ranked 10 to 20 [Query]
- II.3. Add a field country with the value USA to all the zips [Query]
- II.4. List all zones with more than 100 000 inhabitants located on the east of meridian -110. [Query]
- II.5. What is the closest zones to coordinates -73.996705, 40.74838 [Query + Answer]
- II.6. The cities that are less than 5km away from -73.996705, 40.74838: [Query + Answer]
- II.7. The cities that have more than 500 000 inhabitants. [Query + Answer]



# Architecture

09/12/2022

# Journaling



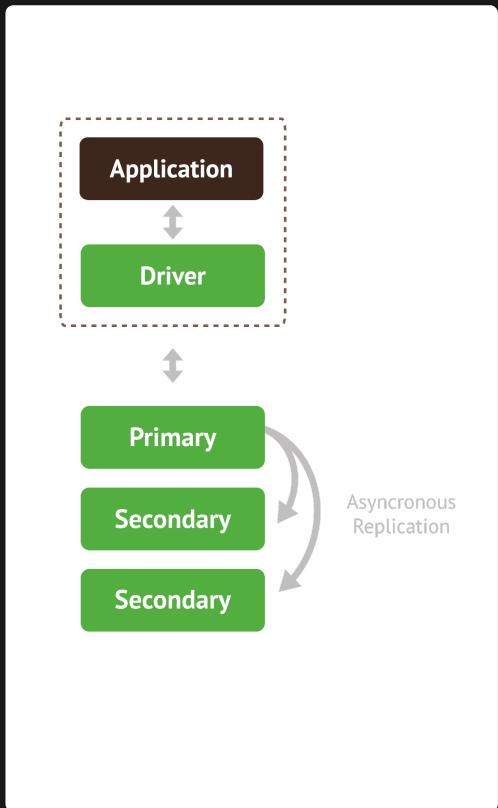
# Replication

## Replica Set

A group of mongod processes that contain the same data set.

Can be a standalone instance (not in production)

# Replica Set



Two copies or more  
Master / Slave  
Automatic Failover  
Purpose

- High Availability
- Data Recovery
- Maintenance

# Replica set

## Members

### Primary

- acceptes all writes and reads
- 1 primary per replica set

### Secondaries

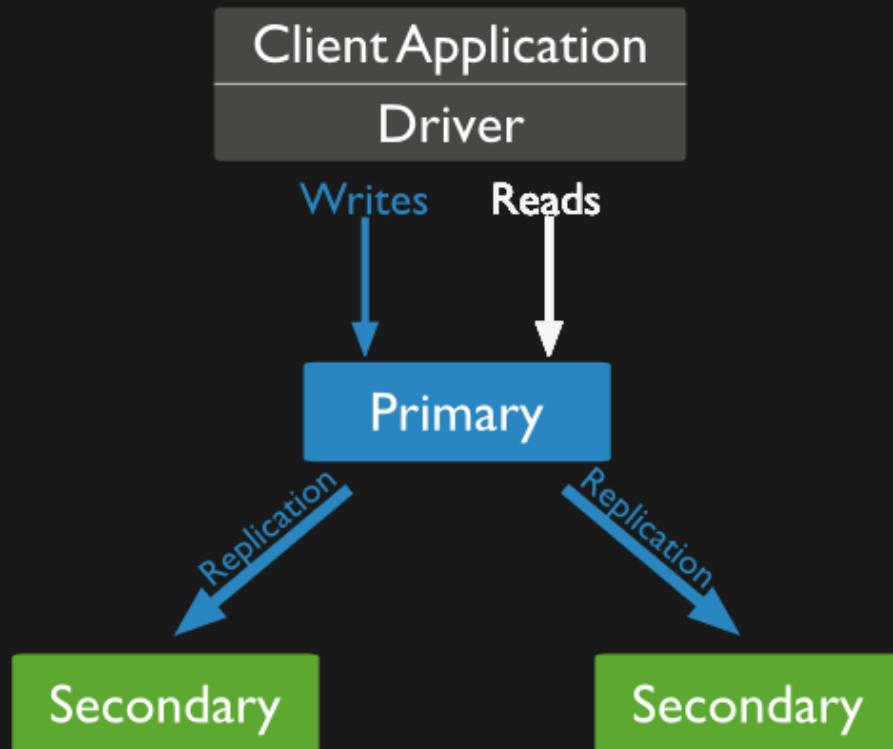
- replicates data
- can serve reads
- can be hidden (priority 0)

### Arbiters

- at most 1
- break ties

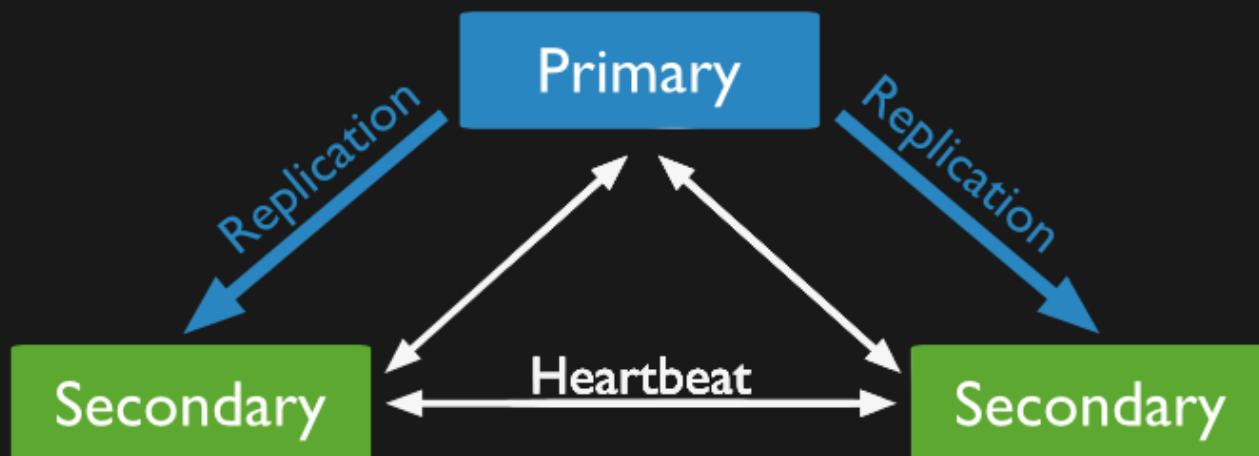
# Replica Set

Operation Management



# Replica Set

Synchronization



# Replica Set

## Journal vs Oplog

### Journal

- low level log of an operation for crash recovery (can be turned off)

### Oplog (similar to RDBMS binlog)

- stores (idempotent) high-level operation that modify the database
- kept on the master and used for replication

[Documentation here](#)

# Replica Set

MongoDB proposes a replica set tutorial in the [documentation](#)  
The full documentation is [here](#)

# Replica Set

## Write semantics

w:1

- Acknowledging the write after the local write

w:N

- Waiting for N members to write before acknowledging

w:majority

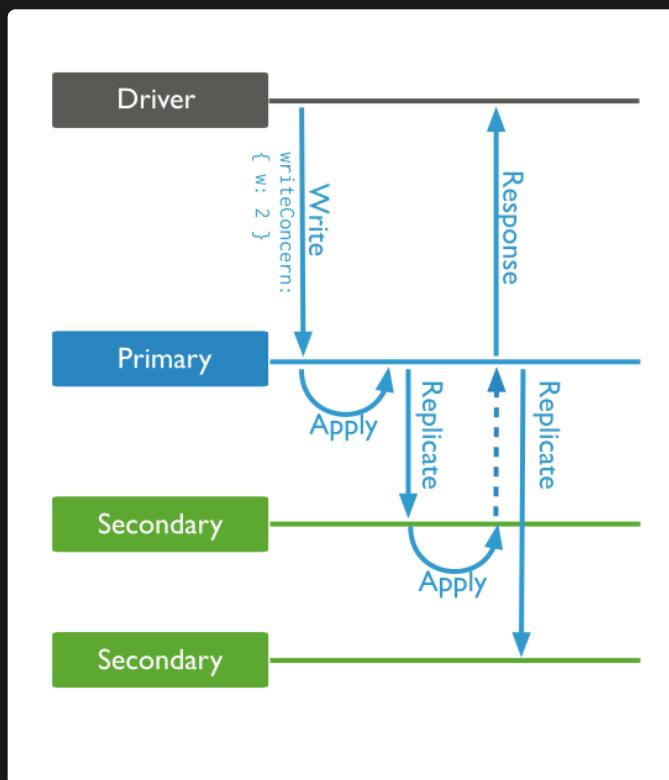
- Waiting for the majority of members to write before acknowledging

wtimeout (optional)

- Prevent blocking if write concern cannot be reached

# Replica Set

## Write semantics



# Replica Set

## Write semantics

At the query level

```
> db.products.insert(  
  { item: "envelopes", qty : 100, type: "Clasp" },  
  { writeConcern: { w: 2, wtimeout: 5000 } }  
)
```

Default value

```
> cfg = rs.conf()  
> cfg.settings = {}  
> cfg.settings.getLastErrorDefaults = { w: "majority", wtimeout: 5000 }  
> rs.reconfig(cfg)
```

# Replica Set

## Read preference

primary (default)

- read from the current replica set primary.

primaryPreferred

- read from primary (or secondary if no primary)

secondary

- read from secondary members

secondaryPreferred

- read from secondary (or primary if no primary)

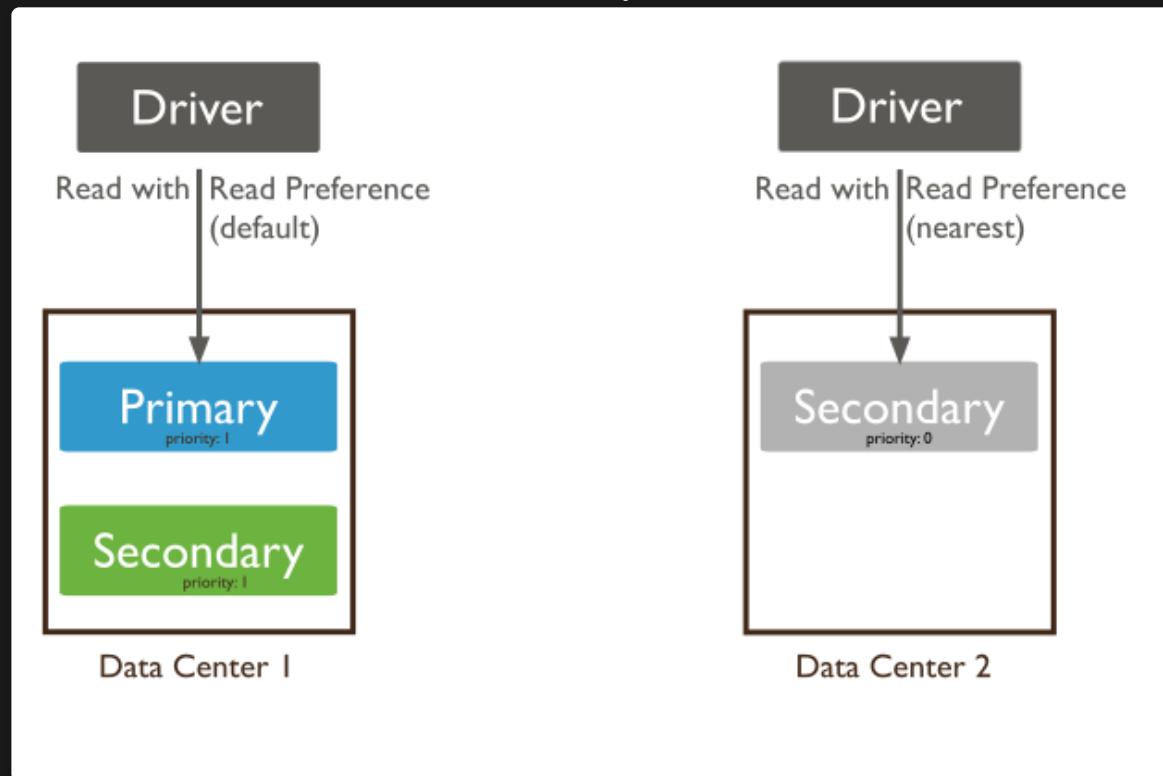
nearest

- read from the member with the least network latency

# Replica Set

## Read preference

### Example



# Replica Set

## Read preference

Usecase:

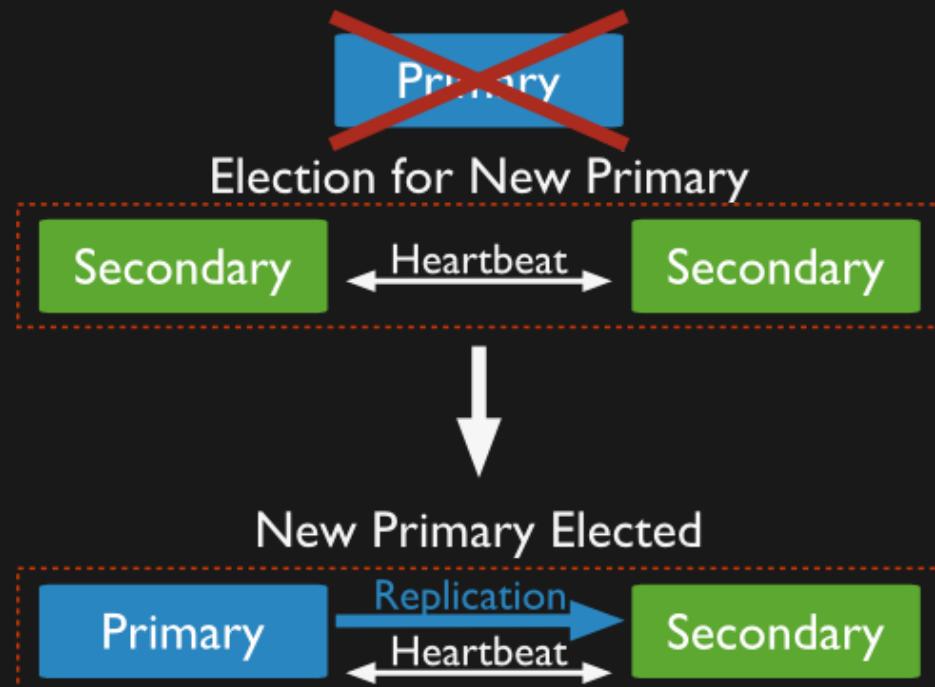
Maximize Consistency => primary

Maximize Availability => primaryPreferred

Minimize Latency => nearest

# Replica Set

Failover



# Replica set

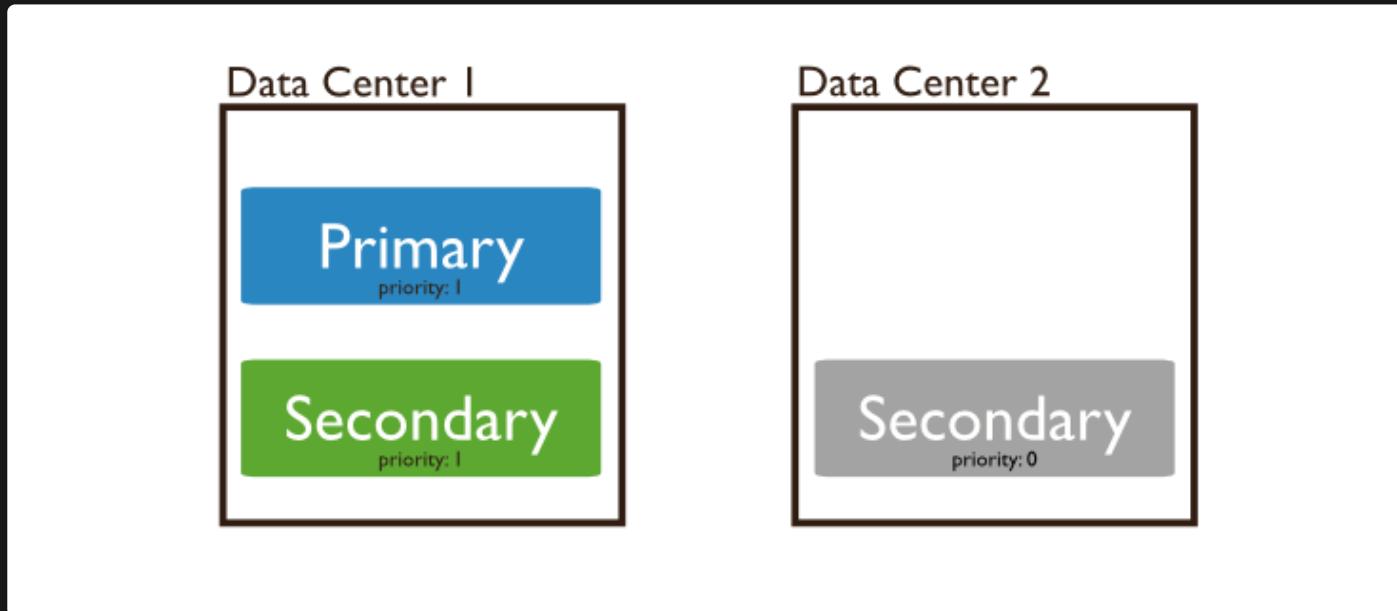
## Election strategy

- odd number of node
- cannot be elected with one vote
- member's priority

# Replica set

## Priority 0

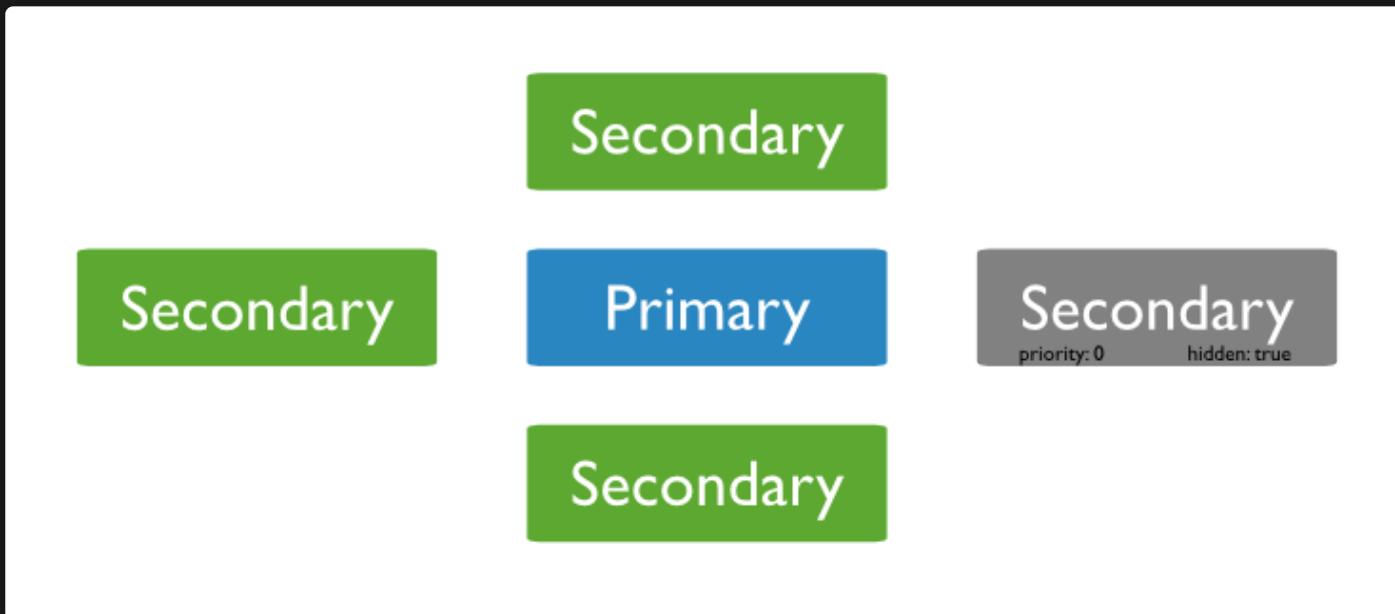
- cannot become primary
- cannot trigger elections
- can vote in elections
- copy of data + accepts reads



# Replica set

Hidden

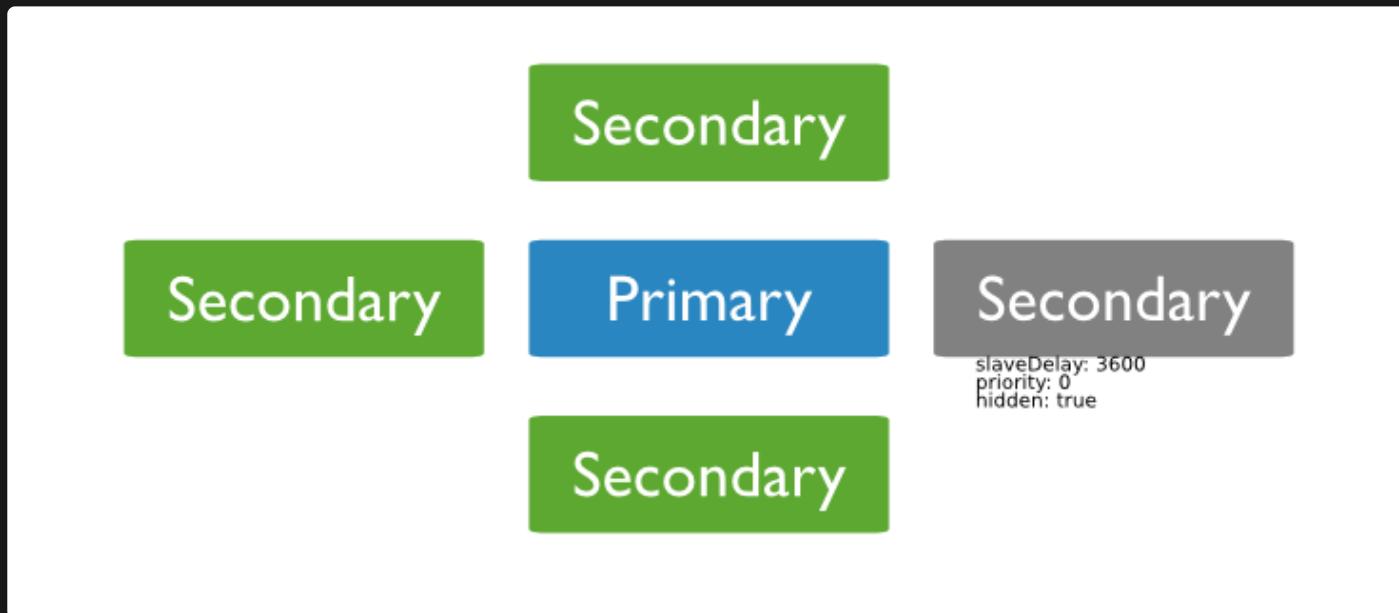
- Priority 0 members that don't accept reads



# Replica set

Delayed

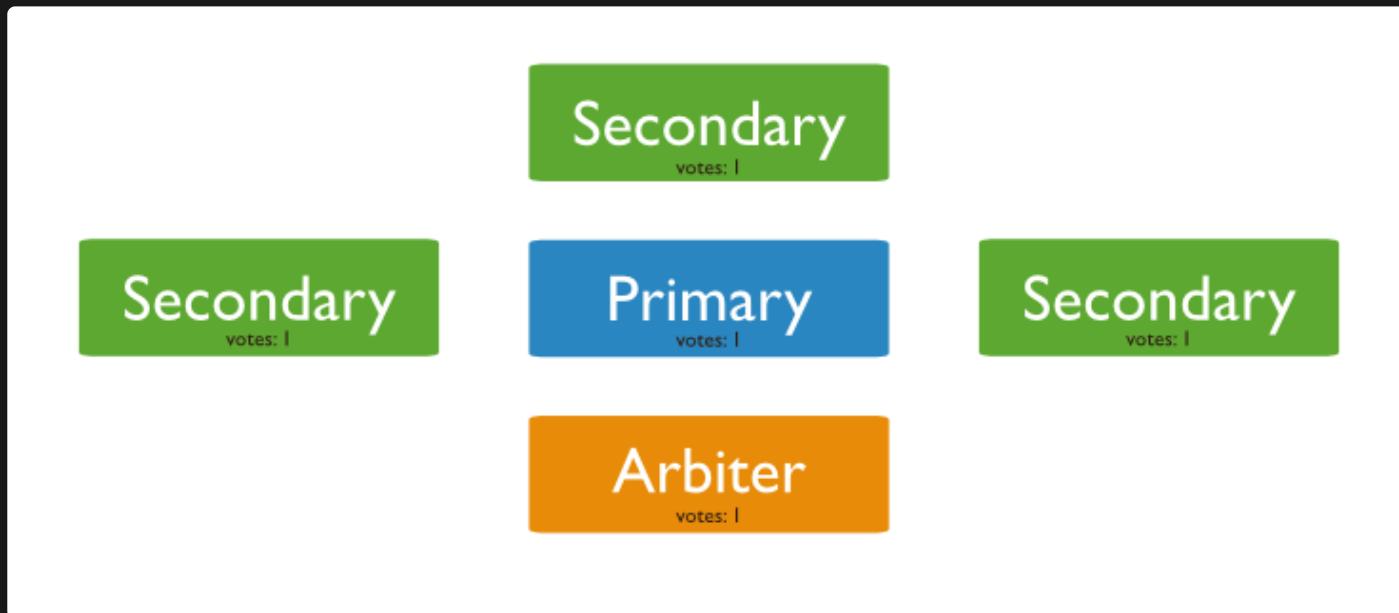
- must be priority 0
- must hidden



# Replica set

## Arbiters

- cannot be elected
- has no dataset



# Replica set

No primary:

- writes no longer possible
- reads still accepted

Fault tolerance:

Number of members that can become unavailable and still be able to elect a primary

Nb of members	Majority required	Fault tolerance
3	2	1
4	3	1
5	3	2
6	4	2

[Documentation here](#)

# Replica set

## Rollback

What if the primary is lost right after accepting write operation ?

Some operations may have not been replicated on secondaries

Issues

- some entries will be lost
- some client could have read those data before they were lost

Solution

- manually apply rollbacks
- change write configuration (semantics)

[Documentation here](#)

# Distribution

## Sharding

Splitting data set across multiple instances

# Sharding

## Config Server

Contains cluster metadatas

- 1 instance in dev, 3 in production
- Contains intervals definitions (chunks)
- Maintenance

# Sharding

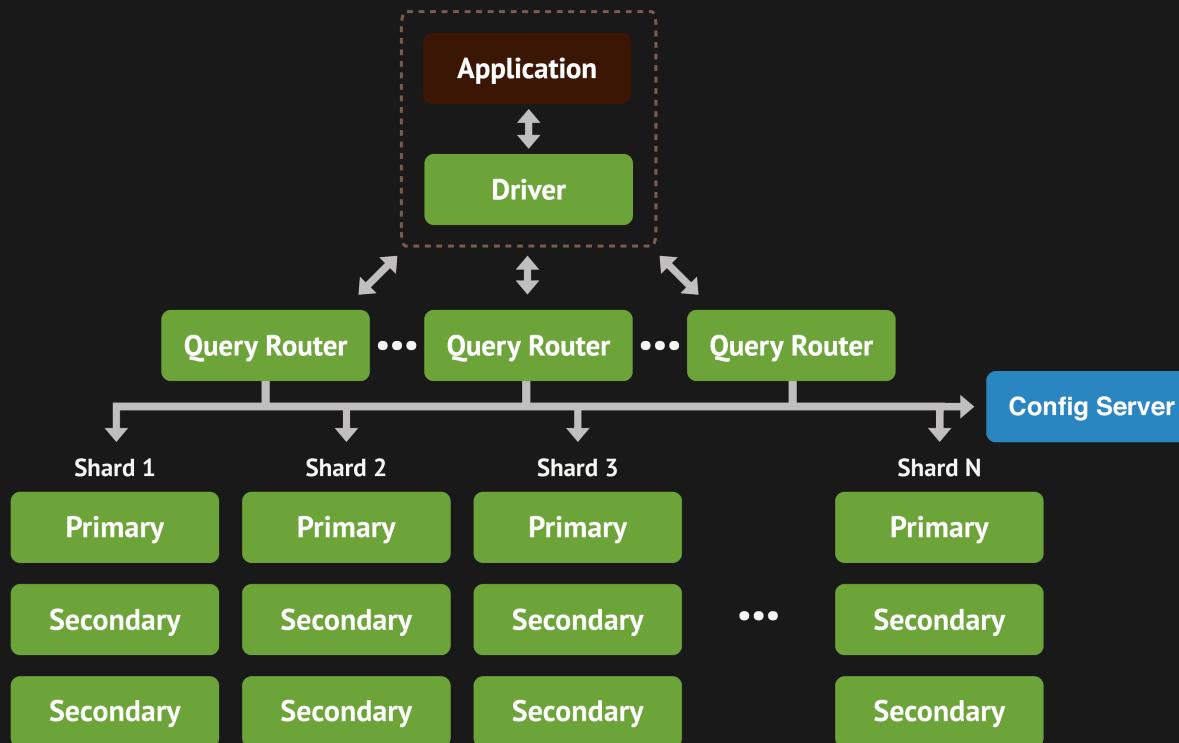
## Query Router (mongos)

Contains cluster metadatas

- Behaves identically to mongod
- Query router
- Load Balancer
- Dedicated or shared

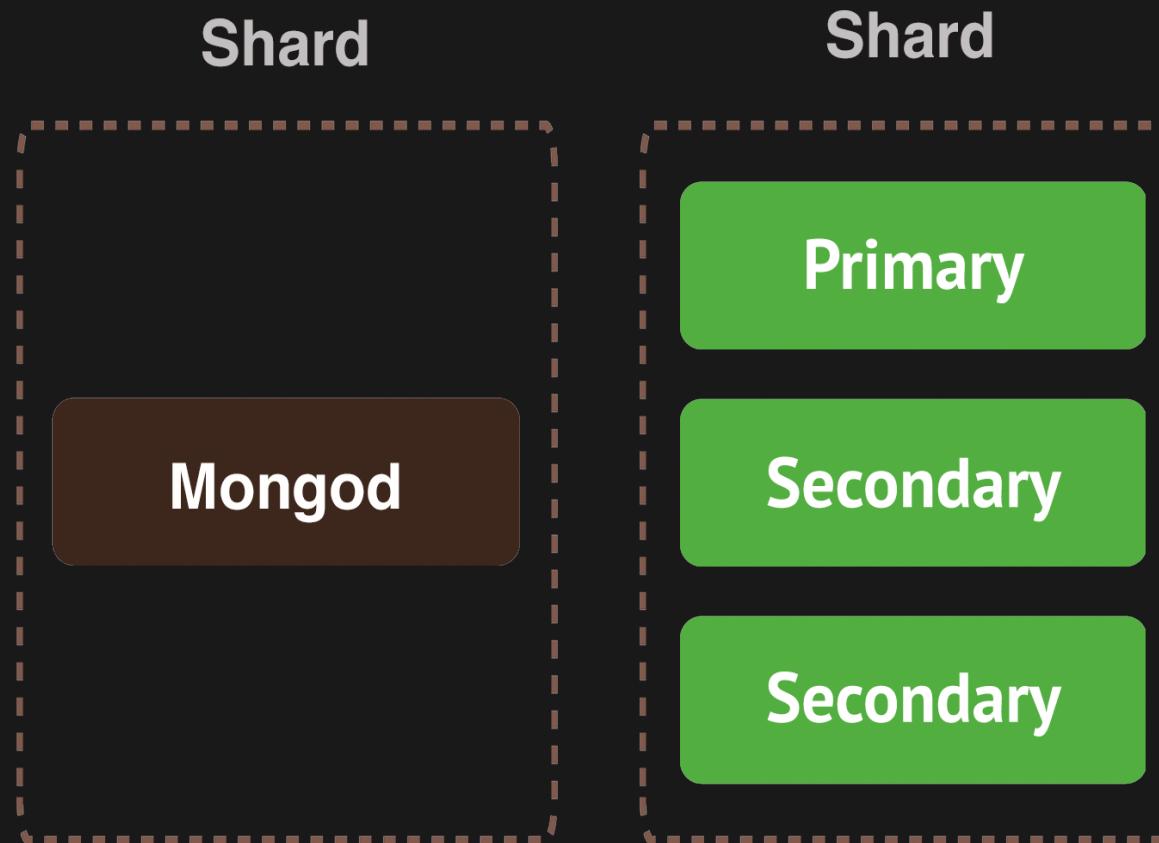
# Sharding

## Architecture



# Sharding

Shard = Cluster Node



# Sharding

Balancing



# Sharding

## Sharding Key

Requirement :

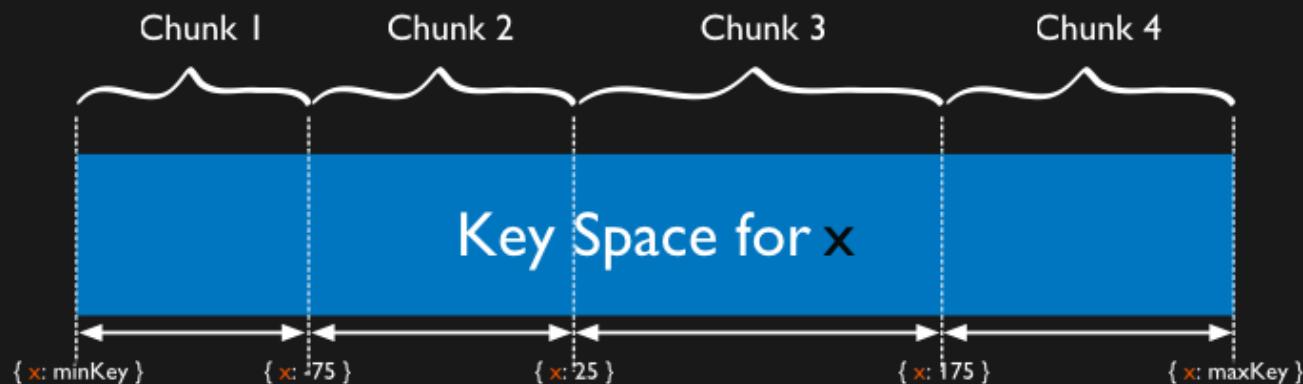
- Immutability (key/value)
- Big cardinality
- Distributed
- Should be indexed
- Limited to 512 octets

# Sharding

## Chunk

Requirement :

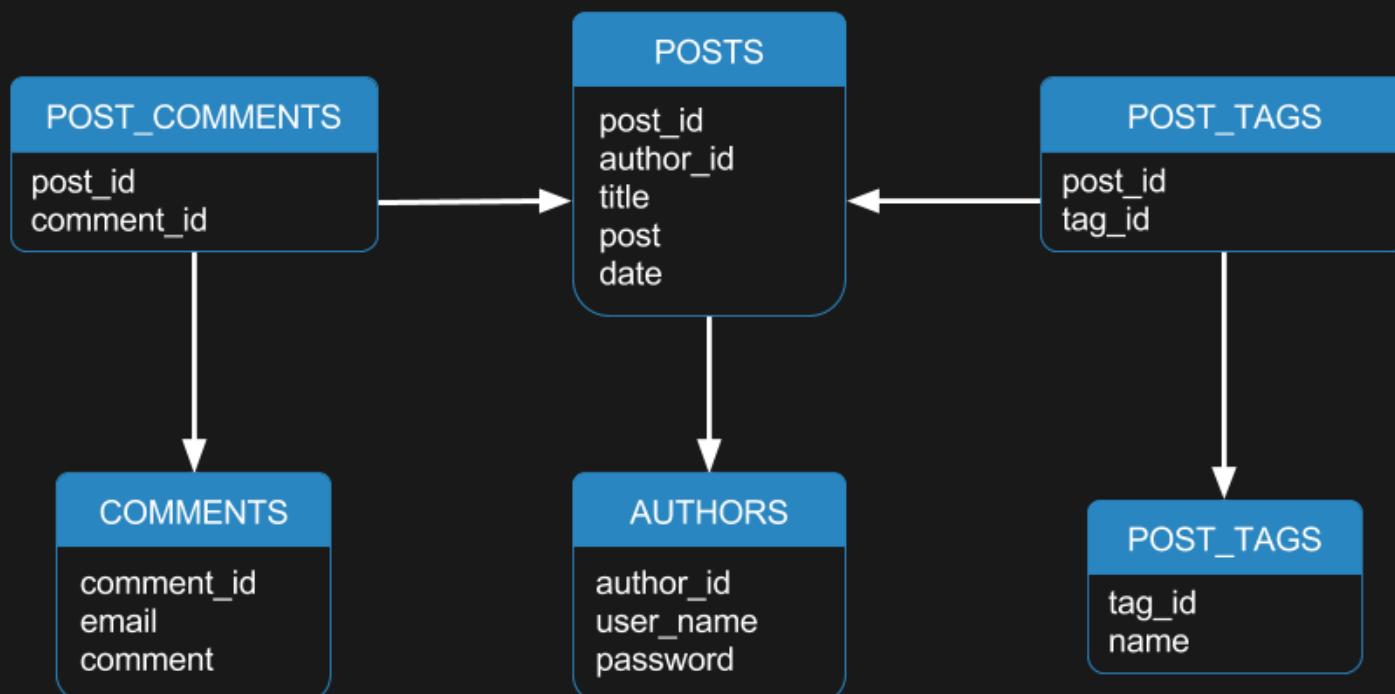
- Split if bigger than 64Mo
- Split & Moved
- Split only between 2 different values
- Moved Automatically





# Blog

Relational



# Blog Document Post

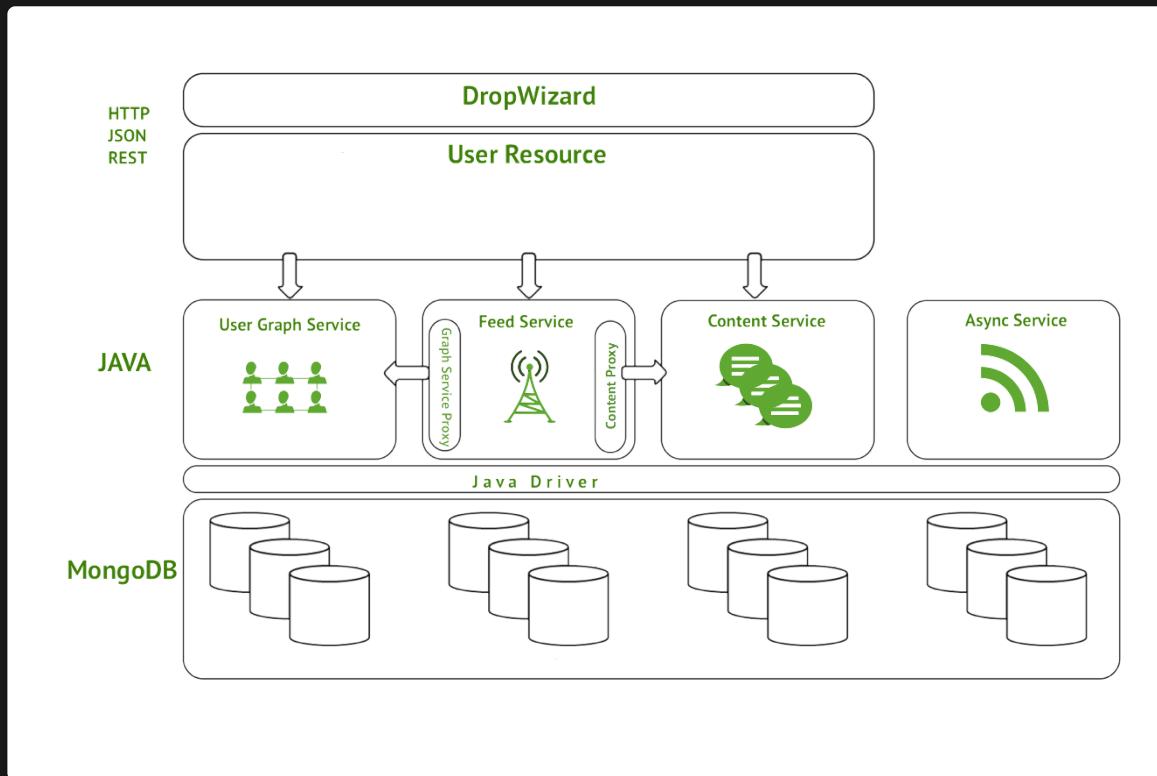
```
{  
  _id: ObjectId("54853dd6dd8fc0fec931fcbc"),  
  title: "Title",  
  body: "...",  
  author: "Author",  
  date: "Date",  
  comments: [  
    { name: "Observer", comment: "Comment", }  
  ],  
  tags: ["Course", "MongoDB"],  
};
```

## Author

```
{ _id: "UserName", email: "UserEmail" }
```

# Socialite

## Architecture



# Socialite

## Design 1

```
...
{ _id : "vHugo", email : "victor.hugo@gmail.com", follower:["gWashington"]}
{ _id : "gWashington", email : "george.washington@gmail.com"}
...
```

# Socialite

## Design 1



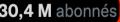
Barack Obama 

@BarackObama

Dad, husband, President, citizen.

Washington, DC  obamabook.com  Naissance le 4 août 1961

A rejoint Twitter en mars 2007

587,6 k abonnements  130,4 M abonnés

VS



obamabarack

@fapoon69

United States  A rejoint Twitter en septembre 2018

0 abonnement  0 abonné

Suivi par aucune des personnes que vous suivez

# Socialite

## Design 2

User collection

```
{ _id : "vHugo", email : "victor.hugo@gmail.com"}  
{ _id : "gWashington", email : "george.washington@gmail.com"}
```

Follower collection

```
{ _id : 1, _from : "gWashington", _to : "vHugo"}
```

# Socialite

## Design 3

### User collection

```
{ _id : "vHugo", email : "victor.hugo@gmail.com"}  
{ _id : "gWashington", email : "george.washington@gmail.com"}
```

### Follower collection

```
{ _id : 1, _from : "gWashington", _to : "vHugo"}
```

### Following collection

```
{ _id : 1, _from : "vHugo", _to : "gWashington"}
```



mongo**D**B

Practical - Applicative

# MongoTasks

## MongoDB based App

The screenshot shows a web application titled "MongoTasks" with a blue header bar. The header includes the app name, a search bar with placeholder text "Search...", and a set of icons for adding (+), refreshing (refresh), and other actions.

The main content area displays a vertical list of seven questions, each enclosed in a white rectangular box with a thin gray border:

- Do you do hallway usability testing ?
- Do new candidates write code during their interview ?
- Do you have testers ?
- Do you use the best tools money can buy ?
- Do programmers have quiet working conditions ?
- Do you have a spec ?
- Do you have an up-to-date schedule ?

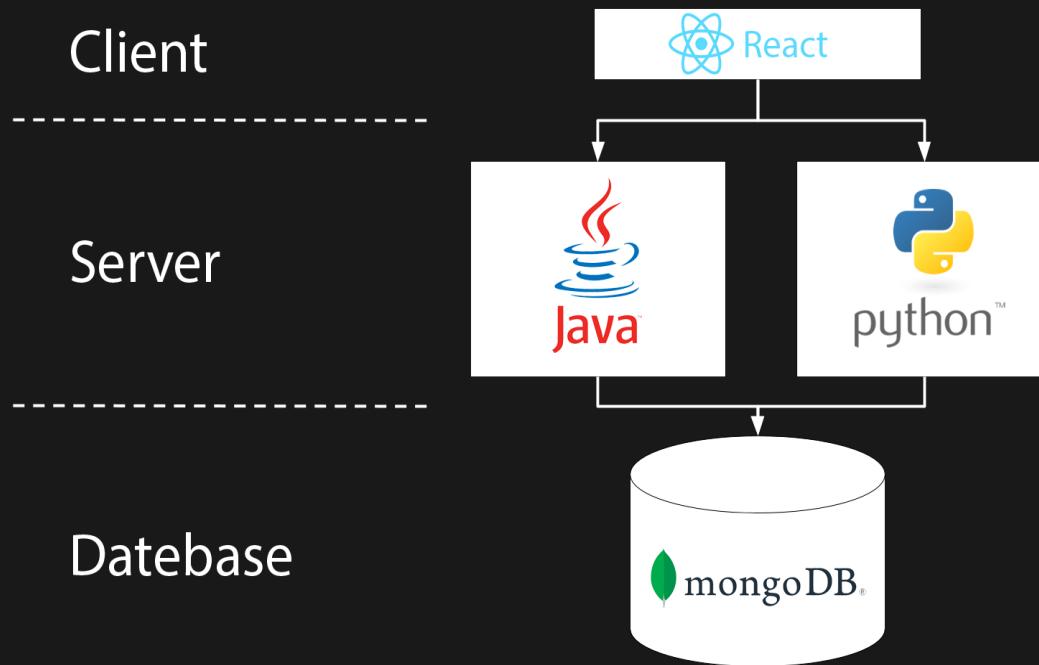
# MongoTasks



<https://github.com/geofberard/MongoTasks>

# MongoTasks

## Architecture



# Run MongoDB

Start server with docker

```
$ docker run --rm --name mongo-tasks -p 27017:27017 mongo
```

If needed, connect with mongo shell

```
$ docker exec -it mongo-tasks mongosh
```

# MongoTasks

## Python

Install Python ([documentation](#))

Install Git ([documentation](#))

Get source code

```
$ git clone https://github.com/geofberard/MongoTasks.git  
$ cd MongoTasks
```

Install python dependencie

```
$ pip3 install Django  
$ pip3 install pymongo
```

# MongoTasks

## Python

Start the server

```
$ cd src/main/python  
$ python3 manage.py runserver
```

```
→ MongoTasks git:(main) ✘ cd src/main/python  
→ python git:(main) ✘ python3 manage.py runserver  
Watching for file changes with StatReloader  
Performing system checks...  
  
System check identified no issues (0 silenced).  
  
You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.  
Run 'python manage.py migrate' to apply them.  
December 17, 2020 - 02:11:18  
Django version 3.1.4, using settings 'mongotasks.settings'  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CONTROL-C.
```

Then you can access the app on: <http://localhost:8000>

# MongoTasks

## Python

All changes need to be done in the file :

```
src/main/python/api/tasks_service.py
```

# MongoTasks

## Python

### CheatSheet

Each step of the practical is saved in a specific branch.

If you get stuck, you might jump to the next step (replace X) with :

```
$ git reset HEAD --hard  
$ git checkout step-X
```

All your changes will be lost but you will see a working implementation and you will be able to keep going.

# MongoTasks

## Python

### Create

You need to add a new document in the tasks collection

```
$ git checkout step-1
```

# MongoTasks

## Python

### Read

You need to return all the notes from the tasks collection

```
$ git checkout step-2
```

# MongoTasks

## Python

### Update

You need to update a document 'done' field in the tasks collection

```
$ git checkout step-3
```

# MongoTasks

## Python

### Delete

You need to remove a document from the tasks collection

```
$ git checkout step-4
```

# MongoTasks

## Python

### Full Implementation

```
$ git checkout complete
```

# MongoTasks

## Java

Install Java ([documentation](#))

Install Maven ([documentation](#))

Install Git ([documentation](#))

Get source code

```
$ git clone https://github.com/geofberard/MongoTasks.git  
$ cd MongoTasks
```

Install java dependencie

```
$ mvn clean install
```

# MongoTasks

## Java

Start the server

```
$ mvn clean install -Pstart
```

```
→ MongoTasks git:(main) ✘ mvn clean install -Pstart
[INFO] Scanning for projects...
[INFO]
[...]
[INFO]
[INFO] --- exec-maven-plugin:1.2.1:java (default) @ mongoTp ---
4 [com.mongodb.MongoTasksServer.main()] INFO spark.staticfiles.StaticFilesConfiguration - StaticResourceHandler configured with folder = /public
64 [Thread-1] INFO org.eclipse.jetty.util.log - Logging initialized @2706ms to org.eclipse.jetty.util.log.Slf4jLog
121 [Thread-1] INFO spark.embeddedserver.jetty.EmbeddedJettyServer - == Spark has ignited ...
121 [Thread-1] INFO spark.embeddedserver.jetty.EmbeddedJettyServer - >> Listening on 0.0.0:8000
124 [Thread-1] INFO org.eclipse.jetty.server.Server - jetty-9.4.31.v20200723; built: 2020-07-23T17:57:36.812Z; git: 450ba27947e13e66baa8cd1ce7e85a4461cac
142 [Thread-1] INFO org.eclipse.jetty.server.session - DefaultSessionIdManager workerName=node0
142 [Thread-1] INFO org.eclipse.jetty.server.session - No SessionScavenger set, using defaults
144 [Thread-1] INFO org.eclipse.jetty.server.session - node0 Scavenging every 600000ms
183 [Thread-1] INFO org.eclipse.jetty.server.AbstractConnector - Started ServerConnector@1d45b030{HTTP/1.1, {http/1.1}}{0.0.0.0:8000}
184 [Thread-1] INFO org.eclipse.jetty.server.Server - Started @2827ms
```

Then you can access the app on: <http://localhost:8000>

# MongoTasks

## Java

All changes need to be done in the file :

```
src/main/java/com/mongodb/TasksService.java
```

# MongoTasks

## Java

### CheatSheet

Each step of the practical is saved in a specific branch.

If you get stuck, you might jump to the next step (replace X) with :

```
$ git reset HEAD --hard  
$ git checkout step-X
```

All your changes will be lost but you will see a working implementation and you will be able to keep going.

# MongoTasks

## Java

### Create

You need to add a new document in the tasks collection

```
$ git checkout step-1
```

# MongoTasks

## Java

### Read

You need to return all the notes from the tasks collection

```
$ git checkout step-2
```

# MongoTasks

## Java

### Update

You need to update a document 'done' field in the tasks collection

```
$ git checkout step-3
```

# MongoTasks

## Java

### Delete

You need to remove a document from the tasks collection

```
$ git checkout step-4
```

# MongoTasks

## Java

Full Implementation

```
$ git checkout complete
```

# Bibliography

MongoDB consistency in real world

Read the documentation for the systems you depend on thoroughly–then verify their claims for yourself. You may discover surprising results!

— Kyle Kingsbury(Aphyr)

<https://aphyr.com/posts/322-jepsen-mongodb-stale-reads>

# Bibliography

MongoDB Manual

MongoDB University

JSon Specification

BSon Specification

Socialite