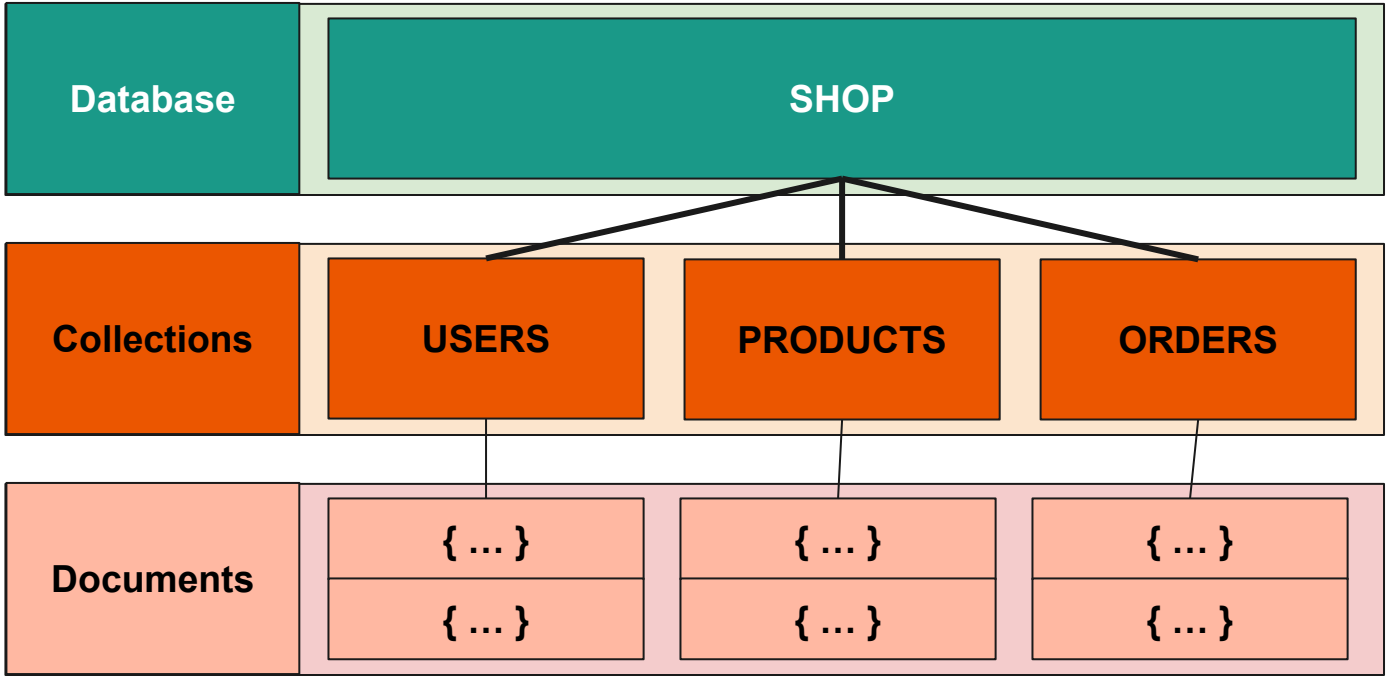
MongoDb Database

Humongous

Because it can store lots and lots of data

How it works

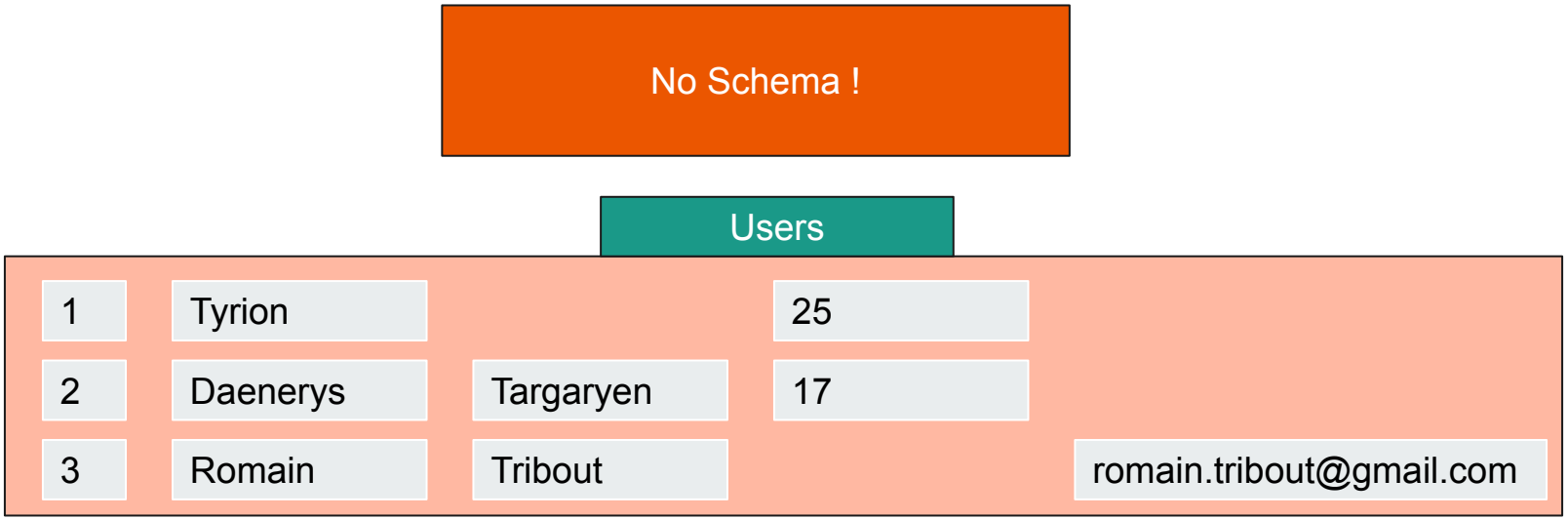

Database
SHOP
Collections
USERS
PRODUCTS
ORDERS
Documents
{ … }
{ … }
{ … }
{ … }
{ … }
{ … }

JSON (BSON) Data Format

```
{

        “name”: “Romain”,
        “age”: 31,
        “address”:
                {
                        “city”: “Lille”
                }
}
```

BSON Data Structure



Installing MongoDB

Install **mongoDb Community Server**

Windows

- store: ubuntu

Get started

First commands

Get all database in mongodb

show dbs

Use a specific database

use shop

Insert data et get data

Insert a data in a collection (ex collection products)

db.products.insertOne({ name: “A TV”, price: 399.99 })

Get all data in collection

db.products.find() or db.products.find().pretty()

Try it!

Insert new data in collection products (add a new field)

Show results in

db.products.find().pretty()

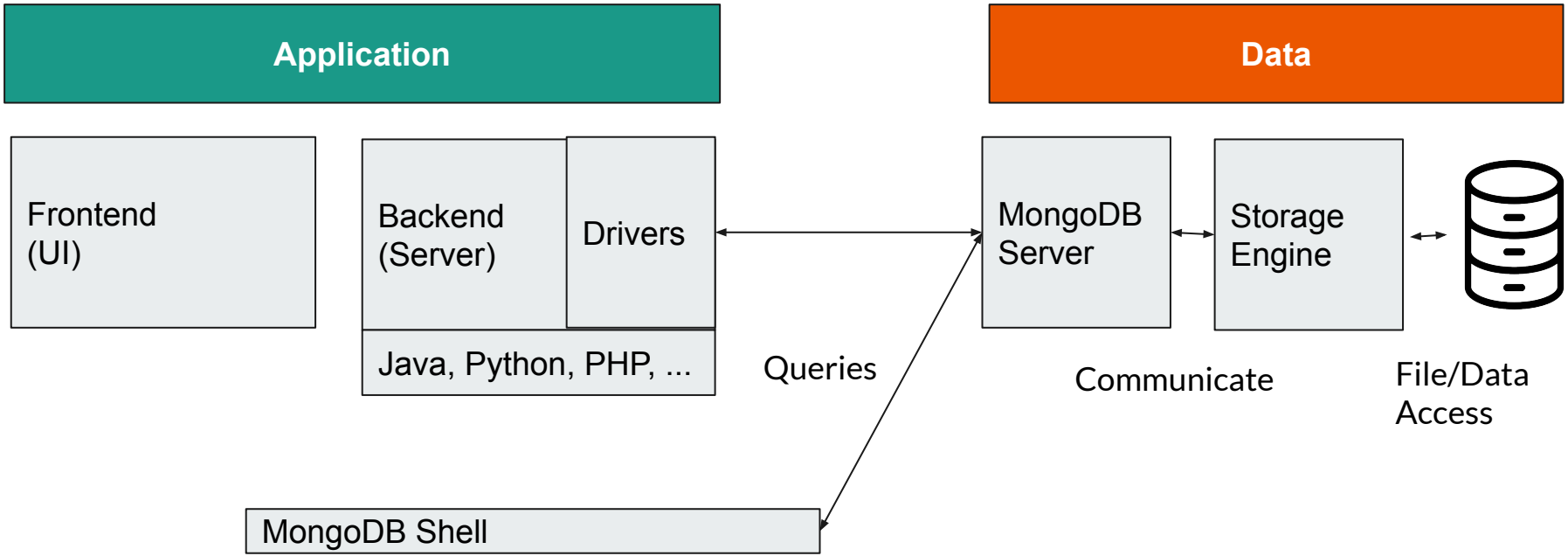
Shell vs Drivers

We use shell for this courses

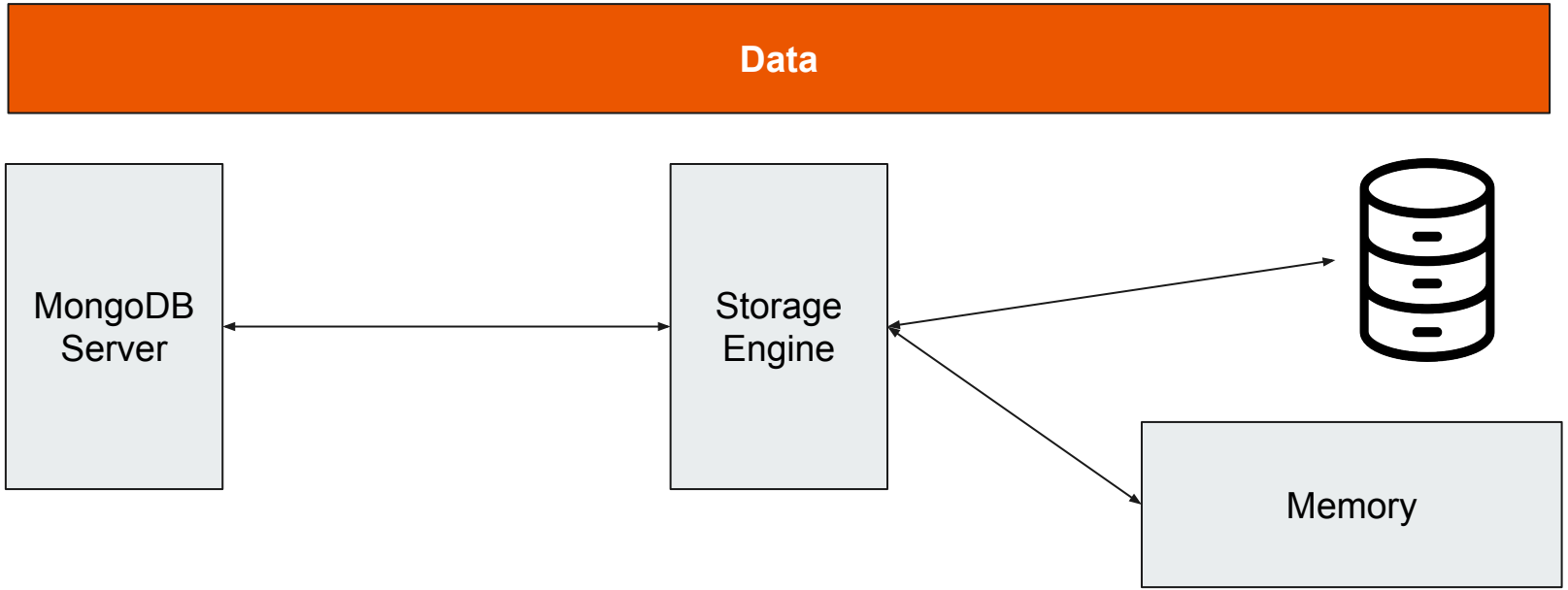
Every programming language have a driver for mongoDB

https://docs.mongodb.com/drivers/

Working with MongoDB



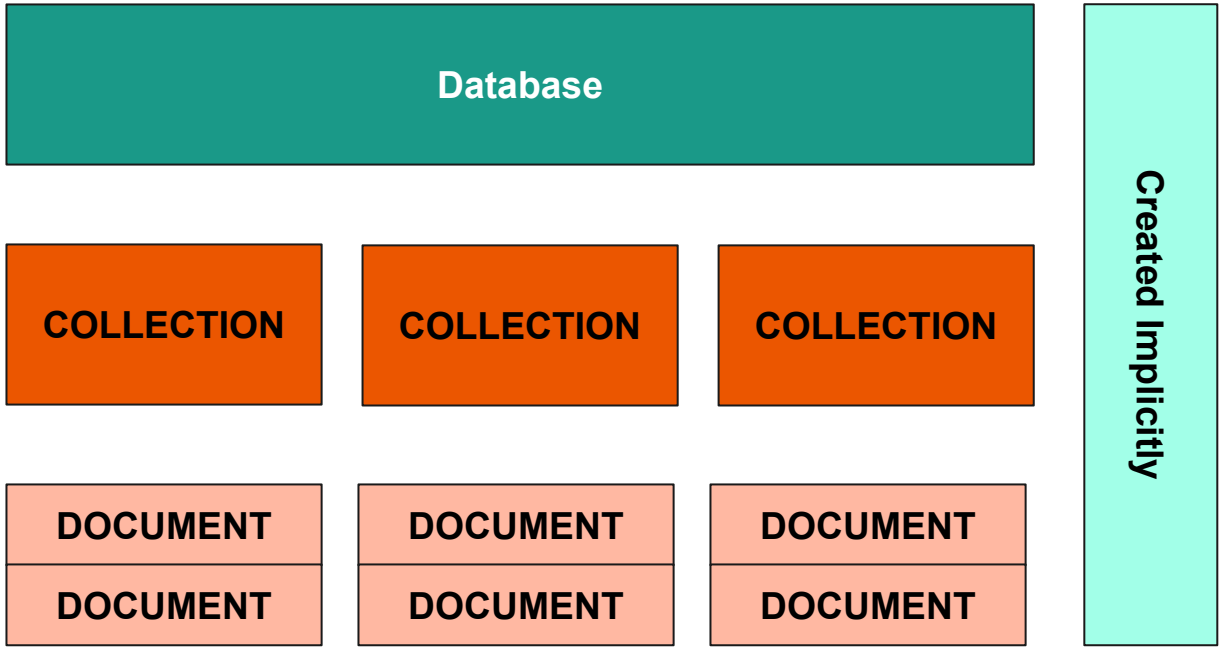
A Closer Look



Basic CRUD

Create Read Update Delete

Databases, Collections, Documents



Start mongodb server

Verify that your mongodb is running!

ONLINE Documentations

https://docs.mongodb.com/manual

Creating database and collections

Show databases:

show dbs

use vls

show dbs -> db “vls” is not here

Insert Data and show Dbs

db.stations.insertOne({
etat: “EN SERVICE”,
name: “HERON PARC”,
size: 16
})

MongoDb assign automatically an Id

show dbs

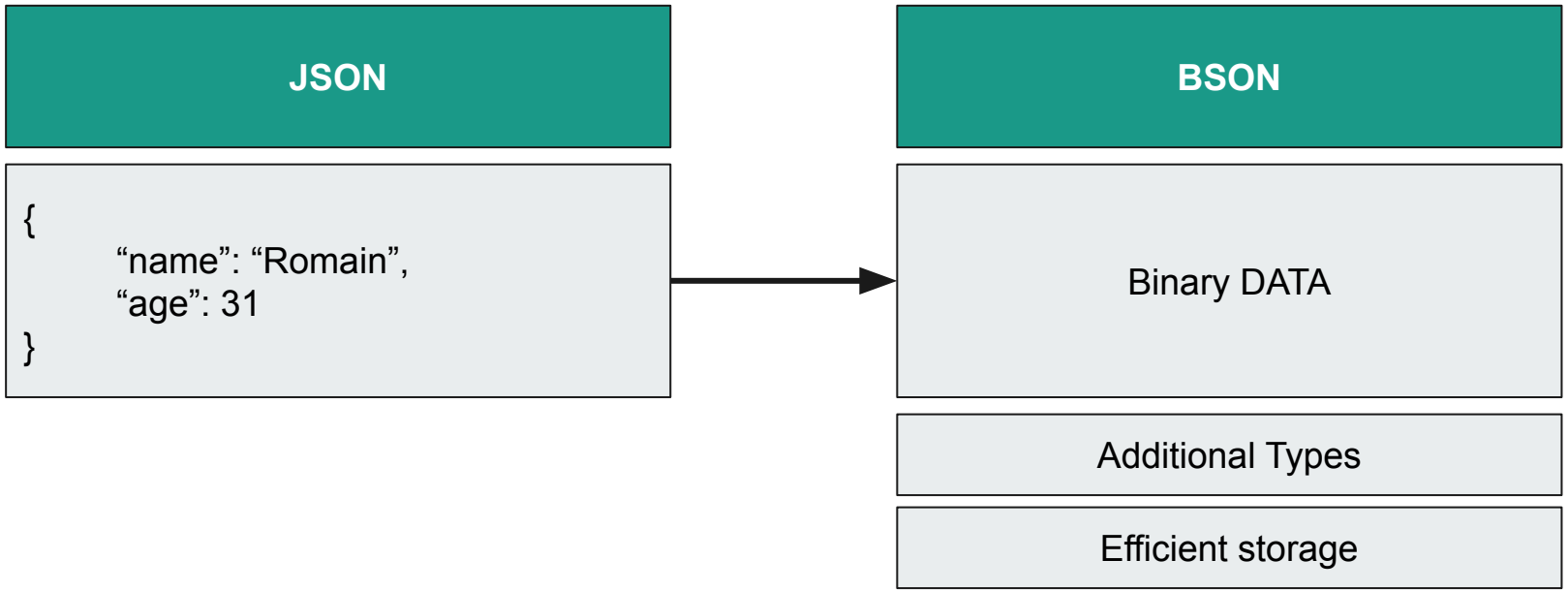

```
{
  "datasetid":"vlille-realtime",
  "recordid":"c8117f6ec1ca79c6731c9c35b9acc1a81e19bf0a",
  "fields":{
    "etat":"EN SERVICE",
    "etatconnexion":"CONNECTED",
    "nbvelosdispo":15,
    "nbplacesdispo":15,
    "commune":"VILLENEUVE D'ASCQ",
    "type":"AVEC TPE",
    "libelle":137,
    "datemiseajour":"2020-09-23T10:27:06+00:00",
    "localisation":[
      50.615868,
      3.126089
    ],
    "nom":"HERON PARC",
    "adresse":"40, RUE DE LA VAGUE",
    "geo":[
      50.615868,
      3.126089
    ]
  },
  "geometry":{
    "type":"Point",
    "coordinates":[
      3.126089,
      50.615868
    ]
  },
  "record_timestamp":"2020-09-23T10:28:06.036000+00:00"
}
```

Show data

db.stations.find().pretty()

"_id" : ObjectId("5f7084d80e23957f14e47f00")

JSON vs BSON


JSON
{
“name”: “Romain”,
“age”: 31
}
BSON
Binary DATA
Additional Types
Efficient storage

Set your own Id

db.stations.insertOne({ _id: “test_id” })

it works!

insert again

db.stations.insertOne({ _id: “test_id” })

throw error, because _id must be unique

CRUD Operations

Create

insertOne(data, options)

insertMany(data, options)

Read

find(filter, options)

findOne(filter, options)

Update

updateOne(filter, data, options)

updateMany(filter, data, option)

replaceOne(filter, data, option)

Delete

deleteOne(filter, option)

deleteMany(filter, option)


```
{
  "datasetid":"vlille-realtime",
  "recordid":"c8117f6ec1ca79c6731c9c35b9acc1a81e19bf0a",
  "fields":{
    "etat":"EN SERVICE",
    "etatconnexion":"CONNECTED",
    "nbvelosdispo":15,
    "nbplacesdispo":15,
    "commune":"VILLENEUVE D'ASCQ",
    "type":"AVEC TPE",
    "libelle":137,
    "datemiseajour":"2020-09-23T10:27:06+00:00",
    "localisation":[
      50.615868,
      3.126089
    ],
    "nom":"HERON PARC",
    "adresse":"40, RUE DE LA VAGUE",
    "geo":[
      50.615868,
      3.126089
    ]
  },
  "geometry":{
    "type":"Point",
    "coordinates":[
      3.126089,
      50.615868
    ]
  },
  "record_timestamp":"2020-09-23T10:28:06.036000+00:00"
}
```

Try commands

deleteOne

db.stations.deleteOne({ _id: “test_id”})

updateOne / updateMany

db.stations.updateOne({ size: 16 }, { $set: { to_delete: true }})

db.stations.updateMany({ etat: “EN SERVICE” }, { $set: { to_delete: true }})

deleteMany

db.stations.deleteMany({ to_delete: true })

Try: insertMany

Insert an array of JSON

```
db.stations.insertMany(
        [
                {
                        "name" : "Zac De La Buire",
                        "size" : 35,
                        "address" : {  "city" : "Lyon 3 Ème", "street" : "Face Au 106 Avenue Félix Faure"},
                        "tpe" : true,
                        "source" : {"id_ext" : "3036","dataset" : "lyon"},
                        "geometry" : {"type" : "Point","coordinates" : [ 4.8598633397749, 45.7537994582272)}
                },
                { … },
                { … }
        ]
)
```

Return all _id in the same order

Limits and summary

Database holds multiple collections and each collection can then hold multiple Documents
Databases and Collections are created “lazily”

Each document needs a unique ID
You may have embedded documents and array fields
one document max 16 mb

CRUD
Some operation have One and Many
find() return a cursor not data

Data Schemas & Data Modelling

Storing your Data Correctly

Schema-less Or not?

MongoDB enforces no schemas! Documents don’t have to use the same schema inside of one collection

Schema-less or Not?

db.products.insertOne({ name: “A book”, price: 12.99 })

db.products.insertOne({ title: “T-Shirt”, amount: 18.99})

It’s works !

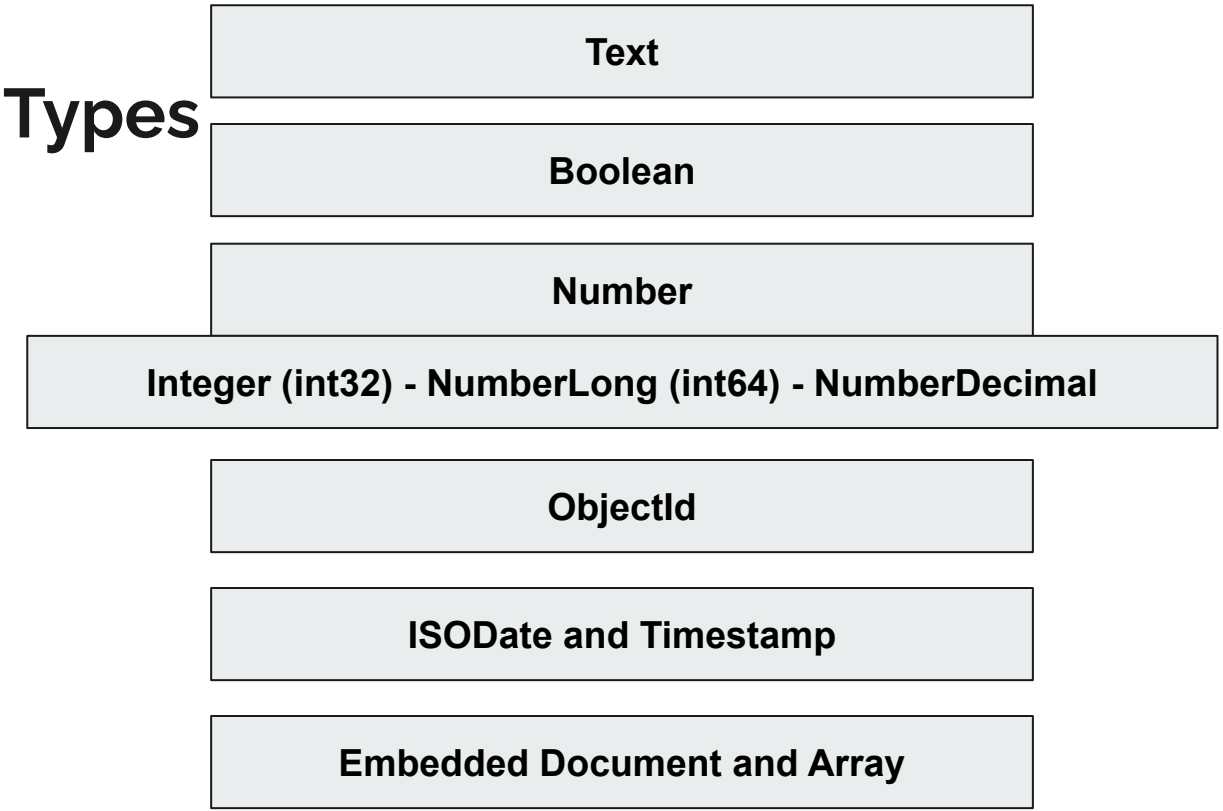
db.products.find({})

But that does not mean thant you can’t use some kind of schema!

To Chema Or not To Schema

chaos → SQL

Data Types


Text
Boolean
Number
Integer (int32) - NumberLong (int64) - NumberDecimal
ObjectId
ISODate and Timestamp
Embedded Document and Array

Relations

Relations - 2 Options

Embedded

users

```
{
    name: ‘Romain’,
    age: 31,
   address: {
        street: ‘bd vauban’,
        city: ‘Lille’
    }
}
```

Refrences

users

```
{
    name: ‘Romain’,
    fav_book: [ {...}, {....} ]
}
```

Lots of data duplications !

users

```
{
    name: ‘Romain’,
    fav_book: [ ‘id_book1’, ‘id_book2’ ]
}
```

book

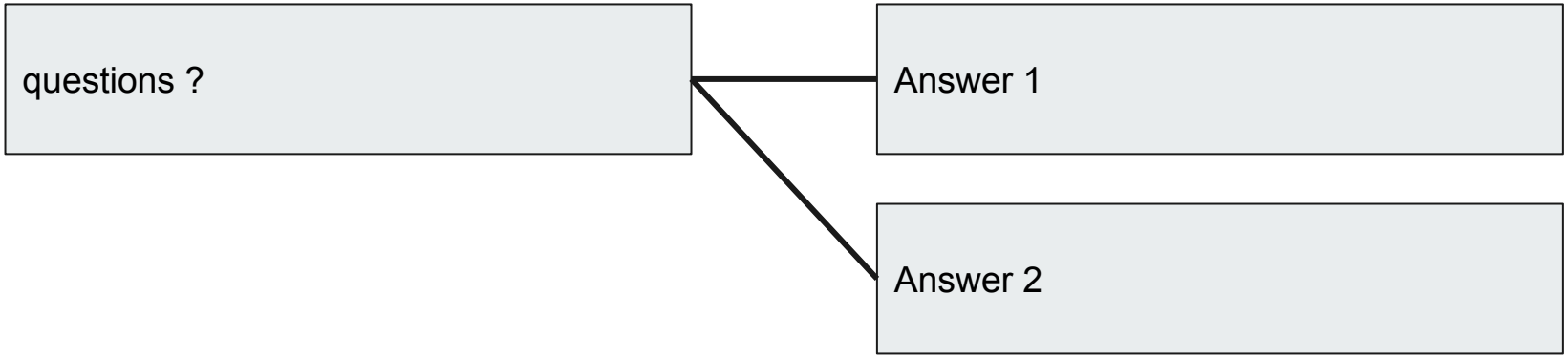
```
{
    _id: ‘id_book1’,
    name: ‘A good book’
}
```

Example 1 Patient <-> Disease Summary (1to1)

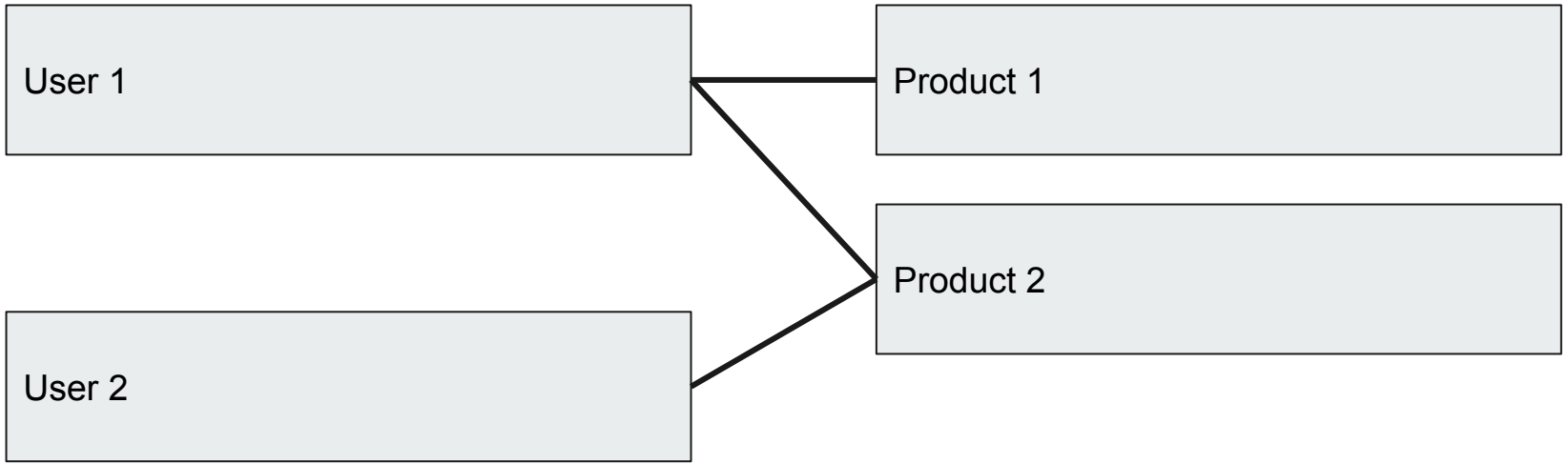
One patient has one disease summary, a disease summary belongs to one patient

Example 2 - Person <-> Car (1to1)

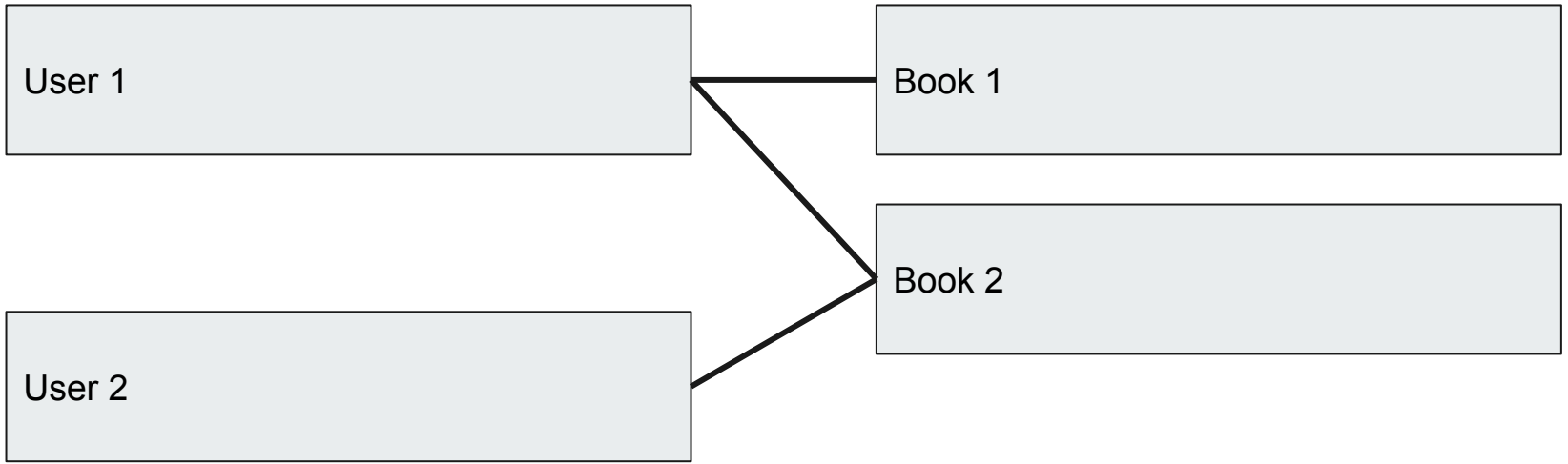
Example 3: Question Answers (1toN relations)



Example 4: Store (NtoN relations)



Example 5: Store (NtoN relations)



Relations

with two request

with $lookup in aggregate

Importer data tests

https://github.com/romaintribout

nosqlmongodbexample1

Download exemples_mongo.zip

mongorestore.exe --archive=test

Query and projection Operators

CRUD Operations

Create

insertOne(data, options)

insertMany(data, options)

Read

find(filter, options)

findOne(filter, options)

Update

updateOne(filter, data, options)

updateMany(filter, data, option)

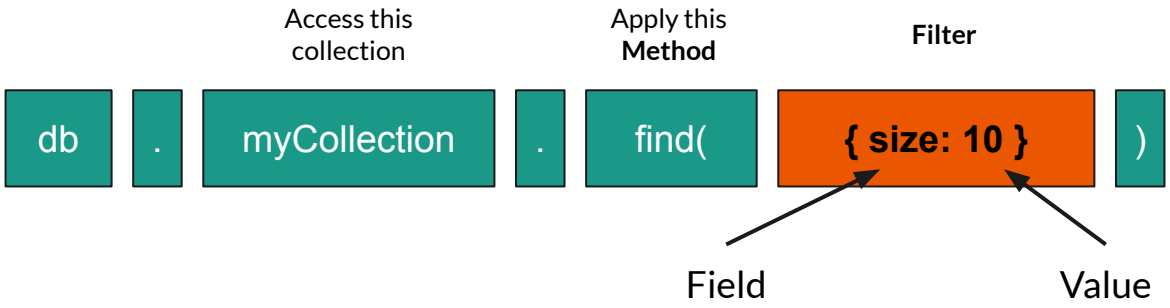
replaceOne(filter, data, option)

Delete

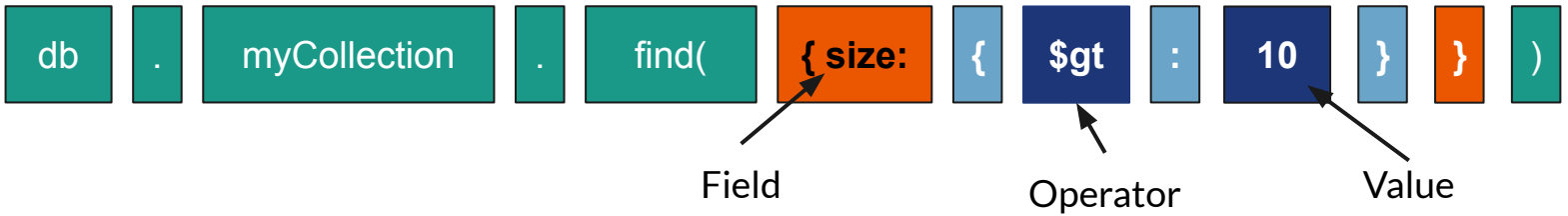
deleteOne(filter, option)

deleteMany(filter, option)

Methods, Filters & Operators



get all document where size equal 10



Understanding findOne() and find()

findOne()

- first match

find()

- all documents macth

Operators

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

Working with comparison operators

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

Example:

Find stations where *size* greater than or equal 10:

- *db.getCollection('stations').find({ **"size" : { "$gte": 10} })*

Find stations where *size* greater than or equal 10 and lower than or equal to 12:

- *db.getCollection('stations').find({ **"size" : { "$gte": 10, "$lte": 12} }*)

Find stations where *size* greater than or equal 10 without tpe:

- *db.getCollection('stations').find({ **"tpe": false, "size" : { "$gte": 10} }*)

Querying embedded Fields & Arrays

Pass a document like argument for filtering datas

Example:

- find stations where *city equal Roubaix*
  - *db.getCollection('stations').find({'address.city': "Roubaix"})*

Working with logical operators

$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

Example:

Find stations where *size* lower than 6 OR greater than or equal to 20 :

```
db.getCollection('stations').find({
        $or: [
                { 'size': { $lt: 6 } },
                { 'size': { $gte: 20 } },
        ]
})
```

Find stations whe size is lower than 6 and not equal to 0:

```
db.getCollection('stations').find({
        $and: [
                { 'size': { $ne: 0 } },
                { 'size': { $lt: 6 } }
        ]
})
```

Working with element operators

$exists	Matches documents that have the specified field
$type	Selects documents if a field is of the specified type

Example:

Find stations where *address not exist:*
db.getCollection('stations').find({'address': { "$exists": false } })

for $type

https://docs.mongodb.com/manual/reference/operator/query/type/#op._S_type

Working with evaluation operators

$expr	Allows use of aggregation expressions within the query language
$regex	Selects documents where values match a specified regular expression
$text	Performs text search. (need index)

Example: working with evaluation operators

Find stations where name contains “Flandres”:

db.getCollection('stations').find({ "name": { $regex: /Flandres/ } })

Find datas where bik_avaible is greater than stabd_available:

db.getCollection('data').find({ $expr: { $gt : ['$bike_available','$stand_available'] }})

Find stations where name contains word “wazemmes”: /!\ need index

db.getCollection('stations').find({ $text: { $search: "wazemmes" } })

Query Arrays

$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.

Example: Query Arrays

Find users with hobbies equal to sports

- *db.getCollection('users').find({ "hobbies.title": "Sports"})*

Find users with 3 hobbies

- *db.getCollection('users').find({ 'hobbies': { "$size": 3 } })*

Find users withs hobby Yoga at frequency 3

- db.getCollection('users').find({ hobbies: { $elemMatch: { "title": "Yoga", "frequency": 3 } } })

Find users withs movies drama and comedy

- *db.getCollection('users').find({ "movies": { $all: ['drama', 'comedy']} })*

Cursors: sorting / skip and limit

find() return a cursor. A cursor is a pointer on datas return by the query

SORT: db.collection().find().sort()
Example: db.getCollection('data').find({}).sort({"date": -1}) => sort by descending date

SKIP:db.collection().find().skip()
Example: db.getCollection('data').find({}).skip(10) => skip first 10 values

LIMIT:db.collection().find().limit()
Example: db.getCollection('data').find({}).limit(10) => limit to 10 values

All this operations can be chained: db.collection().find().skip().limit().sort()

Projection

db.collection.find(filter, projection)

With projection, you choose the fields to return => reduce the amount of datas, _id is return by default.

To remove _id, you need to set _id: 0 in projection option

Exemple:
just return the name, size and city of stations, without _id
db.getCollection('stations').find({}, { _id: 0, name: 1, size: 1, 'address.city': 1 })

Updating

Update operators

$set	Sets the value of a field in a document
$unset	Removes the specified field from a document.
$inc	Increments the value of the field by the specified amount.
$min	Only updates the field if the specified value is less than the existing field value.
$max	Only updates the field if the specified value is greater than the existing field value.
$mul	Multiplies the value of the field by the specified amount.
$rename	Renames a field.

Example: update operators

Set TPE to false for station id *5f75a5db0a6954d88662fc4a*

```
db.getCollection('stations').updateOne(
    { '_id': ObjectId("5f75a5db0a6954d88662fc4a") },
    {$set: {tpe: false}})
```

Increment size of 1 for station id *5f75a5db0a6954d88662fc4a*

```
db.getCollection('stations').updateOne(
    { '_id': ObjectId("5f75a5db0a6954d88662fc4a") },
    {$inc: {size: 1}})
```

Example: update operators

Remove TPE field for station id *5f75a5db0a6954d88662fc4a*

```
db.getCollection('stations').updateOne(
    { '_id': ObjectId("5f75a5db0a6954d88662fc4a") },
    {$unset: {tpe: “”}})
```

Update array operators

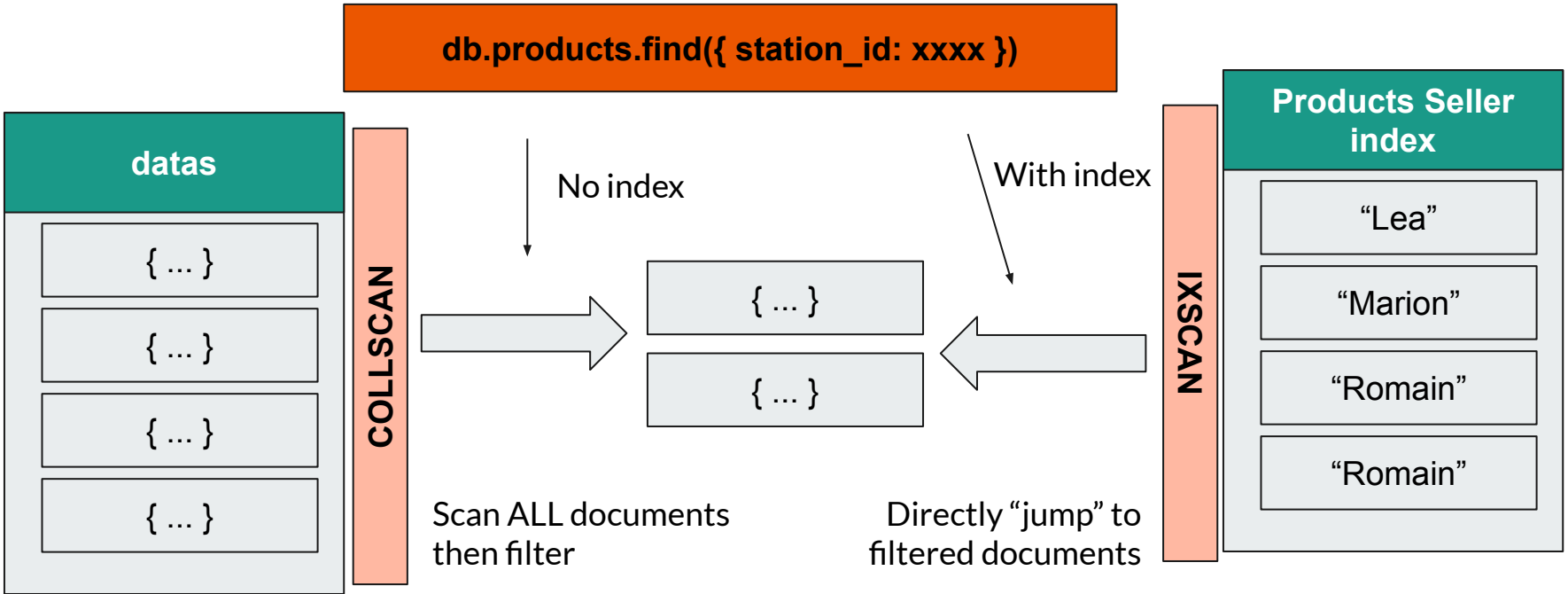
$	Acts as a placeholder to update the first element that matches the query condition.
$[]	Acts as a placeholder to update all elements in an array for the documents that match the query condition.
$pull	Removes all array elements that match a specified query.
$pop	Removes the first or last item of an array.
$push	Adds an item to an array.
$addToSet	Adds elements to an array only if they do not already exist in the set.

Example: update array operators

```
db.getCollection('users').updateMany(
        {'hobbies.title': 'Sports' },
        { '$set': { 'hobbies.$.title': 'Netflix' } }
)
```

Working with indexes

Why Indexes?


db.products.find({ station_id: xxxx })
datas
{ … }
{ … }
{ … }
{ … }
COLLSCAN
No index
{ … }
{ … }
With index
IXSCAN
Products Seller index
“Lea”
“Marion”
“Romain”
“Romain”
Scan ALL documents then filter
Directly “jump” to filtered documents

Don’t Use Too Many Indexes!

When insert data db work a lot for indexes data!

Lot of data for each index

Example: create index

```
db.collection.createIndex( { field: 1 } )

db.collection.getIndexes()
```

Options for create Index

Unique
db.collection.createIndex({field: 1}, {unique: true})

TTL only on date (Time To live)
db.collection.createIndex({field: 1}, {expireAfterSeconds: 10})

Geospatial Queries

Use GeoJSON to store data

```
{
        "type": "Feature",
        "geometry": {
                "type": "Point",
                "coordinates": [longitude, latitude]
        },
        "properties": {
                "name": "Dinagat Islands"
        }
}
```

Geospatial operators

$near	Returns geospatial objects in proximity to a point. Requires a geospatial index. The 2dsphere and 2d indexes support $near
$nearSphere	Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The 2dsphere and 2d indexes support $nearSphere
$geoInstersect	Selects geometries that intersect with a GeoJSON geometry. The 2dsphere index supports $geoIntersects.
$geoWithin	Selects geometries within a bounding GeoJSON geometry. The 2dsphere and 2d indexes support $geoWithin.

Example:

```
db.getCollection('stations').find({'geometry': {
        $near: {
                $geometry: {
                        type: "Point" ,
                        coordinates: [ 3.048567, 50.634268 ]
                },
                $maxDistance: 300,
                $minDistance: 0
        }
}})
```

You need to create a 2dsphere index

Exemple:

- db.stations.createIndex({ geometry: “2dsphere” })

Exemple: get stations in zone (polygon)

```
db.getCollection('stations').find({'geometry': {
        $geoWithin: {
                "$geometry": {
                        "type": "Polygon",
                        "coordinates": [ ....] }
                }
        })
})
```

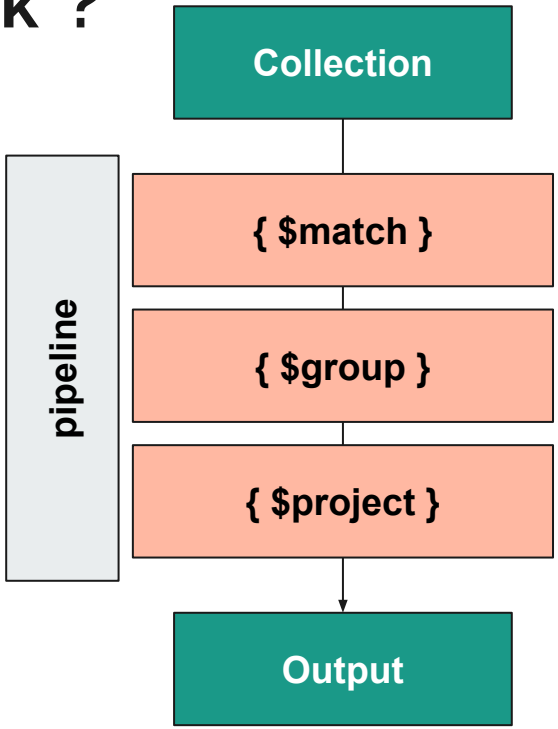
Using the aggregation Framework

What is the “Aggregation Framework”?

An alternative to the find method

You find with differents steps, and you can operate data transformations

All steps are in pipelines



Get started

db.collection.aggregate([{step1}, {step2}, {...}, {step n}])

Exemple:

```
db.getCollection('stations').aggregate([
        {
                $match: {
                        size: { $gt: 60 }
                }
        }
])
```

$group

```
db.getCollection('data').aggregate([
        {
                $group: {
                        _id: { station_id: "$station_id", dayOfYear: { $dayOfYear: "$date" } },
                        total_datas: { $sum: 1 },
                        last_date: { $max: "$date" },
                        first_date: { $min: "$date" }
                }
        }
])
```

$sort

```
db.getCollection('data').aggregate([
        {
                $group: {
                        _id: { station_id: "$station_id", dayOfYear: { $dayOfYear: "$date" } },
                        total_datas: { $sum: 1 },
                        last_date: { $max: "$date" },
                        first_date: { $min: "$date" }
                }
        },
        {
                $sort: { total_datas: -1  }
        }
])
```

$project

```
db.getCollection('data').aggregate([
        {
                $project: {
                        _id: 0,
                        date: 1,
                        station_id: 1,
                        size: { $sum: ["$bike_available", "$stand_available"] },
                        status: {
                                bikes: "$bike_available",
                                stands: "$stand_available"
                        },
                        info: { $cond: { if: "$available", then: "available", else: "not available" } }
                }
        }
])
```

$lookup

```
db.getCollection('data').aggregate([
        {
                $lookup: {
                        from: "stations",
                        localField: "station_id",
                        foreignField: "_id",
                        as: "station"
                }
        }
])
```

$unwind

```
db.getCollection('data').aggregate([
        {
                $lookup: {
                        from: "stations",
                        localField: "station_id",
                        foreignField: "_id",
                        as: "station"
                }
        },
        {
                $unwind: “$station”
        }
])
```

Project

Mongo Atlas

https://www.mongodb.com/cloud/atlas

Self-services Bicycle

Write 4 programs in python and mongo

(1) Get self-services Bicycle Stations (geolocations, size, name, tpe, available): Lille, Lyon, Paris and Rennes

(2) Worker who refresh and store live data for a city (history data)

Self-services Bicycle

(3) Client program:

- give available stations name next to the user lat, lon with last data (bikes and stand)

(4) Business program - for the city

- find station with name (with some letters)
- update / delete a station
- deactivate all station in an area
- give all stations with a ratio bike/total_stand under 20% between 18h and 19h00 (monday to friday)

Example to get vlille data

```
import requests
import json

def get_vlille():
    url = "https://opendata.lillemetropole.fr/api/records/1.0/search/?dataset=vlille-
realtime&q=&rows=3000&facet=libelle&facet=nom&facet=commune&facet=etat&facet=type&facet=etatconnexion"
    response = requests.request("GET", url)
    response_json = json.loads(response.text.encode('utf8'))
    return response_json.get("records", [])
```

Rules

Use Mongo Atlas

Create a git repository and send URL to romain.tribout@gmail.com with subject [ISEN/MONGO]

Optimize request mongo: (useful data)

Create requirements.txt file with your python lib