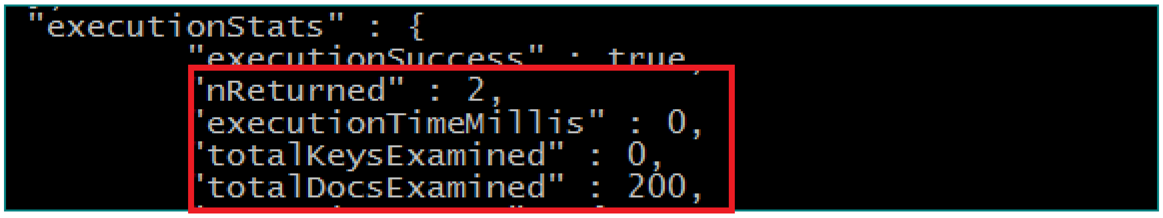
Performance

```
>mongoimport -d school -c students < students.json
connected to: 127.0.0.1
2014-05-13T07:12:09.293+0200 check 9 200
2014-05-13T07:12:09.294+0200 imported 200 objects

> use school
switched to db school

> db.students.findOne()
{
        "_id" : 0,
        "name" : "aimee Zank",
        "scores" : [
                {
                        "type" : "exam",
                        "score" : 1.463179736705023
                },
                {
                        "type" : "quiz",
                        "score" : 11.78273309957772
                },
                {
                        "type" : "homework",
                        "score" : 6.676176060654615
                },
                {
                        "type" : "homework",
                        "score" : 35.8740349954354
                }
        ]
}
```

db.students.find({name:"aimee Zank"}) : parcours l'intégralité de la collection

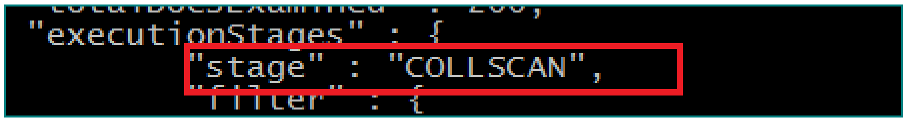
Tester avec

db.students.find({name:"aimee Zank"}).explain("executionStats")



```
"executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 2,
        "executionTimeMillis" : 0,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 200,
```

…



```
"executionStages" : {
        "stage" : "COLLSCAN",
        "filter" : {
```

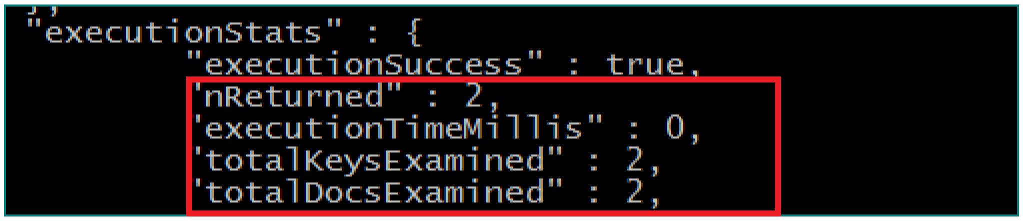
`db.students.findOne({name:"aimee Zank"})` : trouve le premier qui correspond, donc plus rapide

Création d'index

```
> db.students.ensureIndex({name:1})
{
	"createdCollectionAutomatically" : false,
	"numIndexesBefore" : 1,
	"numIndexesAfter" : 2,
	"ok" : 1
}
>db.students.find({name:"aimee Zank"})
```
: plus rapide

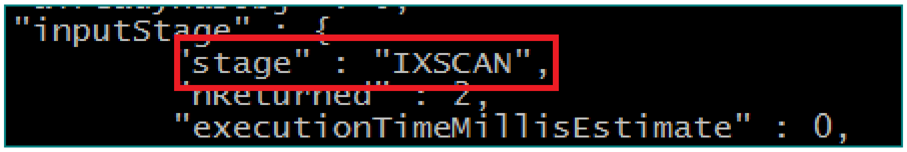
Tester avec :

db.students.find({name:"aimee Zank"}).explain("executionStats")



```
"executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 2,
        "executionTimeMillis" : 0,
        "totalKeysExamined" : 2,
        "totalDocsExamined" : 2,
```

…



```
"inputStage" : {
        "stage" : "IXSCAN",
        "nReturned" : 2,
        "executionTimeMillisEstimate" : 0,
```

Lister les index d'une collection

```
> db.students.getIndexes()
[
	{
		"v" : 1,
		"key" : {
			"_id" : 1
		},
		"name" : "_id_",
		"ns" : "school.students"
	},
	{
		"v" : 1,
		"key" : {
			"name" : 1
		},
            "name" : "name_1",
            "ns" : "school.students"
    }
]
```

Supprimer un index

```
> db.students.dropIndex({name:1})
{ "nIndexesWas" : 2, "ok" : 1 }
```

MultyKey index

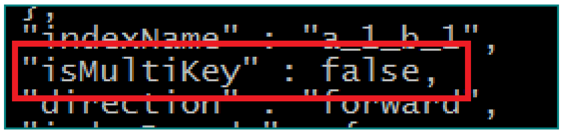
```
mongoimport -d school -c foo < foo.json
```

```
> db.foo.insert({a:1,b:1})
WriteResult({ "nInserted" : 1 })
```

```
> db.foo.ensureIndex({a:1,b:1})
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

```
> db.foo.find({a:1})
{ "_id" : ObjectId("5372490be54ca6f43c383250"), "a" : 1, "b" : 1 }
```

```
> db.foo.find({a:1}).explain("executionStats")
```

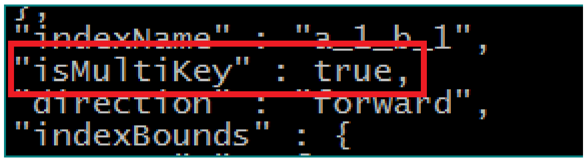


```
"indexName" : "a_1_b_1",
"isMultiKey" : false,
"direction" : "forward",
```

```
> db.foo.insert({a:[1,2,3],b:5})
WriteResult({ "nInserted" : 1 })
```

```
> db.foo.find({a:1})
{ "_id" : ObjectId("5372490be54ca6f43c383250"), "a" : 1, "b" : 1 }
{ "_id" : ObjectId("53724b1de54ca6f43c383251"), "a" : [ 1, 2, 3 ], "b" : 5 }
```

```
db.foo.find({a:1}).explain("executionStats")
```



```
"indexName" : "a_1_b_1",
"isMultiKey" : true,
"direction" : "forward",
"indexBounds" : {
```

```
> db.foo.insert({a:[1,2,3],b:[3,4,5]})
WriteResult({
	"nInserted" : 0,
	"writeError" : {
		"code" : 10088,
		"errmsg" : "insertDocument :: caused by :: 10088 cannot index parallel arrays [b] [a]" }
})
```

Option sur la création d’index : unique

```
> db.stuff.insert({thing:"pear"})
WriteResult({ "nInserted" : 1 })

> db.stuff.ensureIndex({thing:1})
{
	"createdCollectionAutomatically" : false,
	"numIndexesBefore" : 1,
	"numIndexesAfter" : 2,
	"ok" : 1
}

> db.stuff.insert({thing:"pear"})
WriteResult({ "nInserted" : 1 })

> db.stuff.insert({thing:"apple"})
WriteResult({ "nInserted" : 1 })

> db.stuff.getIndexes()
[
	{
		"v" : 1,
		"key" : {
			"_id" : 1
		},
		"name" : "_id_",
		"ns" : "test.stuff"
	},
	{
		"v" : 1,
		"key" : {
			"thing" : 1
		},
		"name" : "thing_1",
		"ns" : "test.stuff"
	}
]
```

```
> db.stuff.dropIndex({thing:1})
{ "nIndexesWas" : 2, "ok" : 1 }

> db.stuff.ensureIndex({thing:1},{unique:1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "ok" : 0,
        "errmsg" : "E11000 duplicate key error index: test.stuff.$thing_1  dup key: { : \"pear\" }",

        "code" : 11000
}

> db.stuff.find()
{ "_id" : ObjectId("53724e24e54ca6f43c383254"), "thing" : "pear" }
{ "_id" : ObjectId("53724e71e54ca6f43c383255"), "thing" : "pear" }
{ "_id" : ObjectId("53724e86e54ca6f43c383256"), "thing" : "apple" }
```

Enlever une collection contenant le mot “**pear**”

```
> db.stuff.ensureIndex({thing:1},{unique:1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}

> db.stuff.getIndexes()
[
        {
                "v" : 1,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "test.stuff"
        },
        {
                "v" : 1,
                "unique" : true,
                "key" : {
                        "thing" : 1
                },
                "name" : "thing_1",
                "ns" : "test.stuff"
        }
]

> db.stuff.insert({thing:"apple"})
WriteResult({
        "nInserted" : 0,
        "writeError" : {
                "code" : 11000,
                "errmsg" : "insertDocument :: caused by :: 11000 E11000 duplicate key error index: test.stuff.$thing_1 dup
key: { : \"apple\" }"
        }
})
```

Option sur la création d'index : sparse

```
> db.produits.insert({item:"polo shirt",size:"medium"})
WriteResult({ "nInserted" : 1 })
> db.produits.insert({item:"jeans",size:"32x32"})
WriteResult({ "nInserted" : 1 })
> db.produits.insert({item:"iphone"})
WriteResult({ "nInserted" : 1 })
> db.produits.insert({item:"DVI-to-VGA cable"})
WriteResult({ "nInserted" : 1 })
> db.produits.find()
{ "_id" : ObjectId("53733414a6d5dcf0dfba69a3"), "item" : "polo shirt", "size" : "medium" }
{ "_id" : ObjectId("5373343aa6d5dcf0dfba69a4"), "item" : "jeans", "size" : "32x32" }
{ "_id" : ObjectId("53733461a6d5dcf0dfba69a5"), "item" : "iphone" }
{ "_id" : ObjectId("5373348fa6d5dcf0dfba69a6"), "item" : "DVI-to-VGA cable" }
```

Problème si on veut poser un index unique sur « size »

```
> db.produits.ensureIndex({size:1},{unique:true})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "ok" : 0,
        "errmsg" : "E11000 duplicate key error index: test.produits.$size_1  dup key: { : null }",
        "code" : 11000
}
```

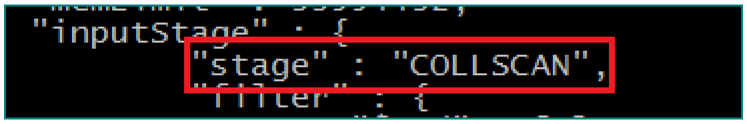
null car mongo considére que les deux derniers documents ont le champs « size » égal à null.

Solution : sparse

```
> db.produits.ensureIndex({size:1},{unique:true, sparse:true})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

```
> db.produits.find({size:"medium"}).explain("executionStats")
```

```
> db.produits.find().sort({size:1}).explain("executionStats")
```



```
"inputStage" : {
        "stage" : "COLLSCAN",
        "filter" : {
```

Pb : car il existe des documents qui n’ont pas le champ « **size** »

La commande : explain

>**mongoimport -d test -c foo < foo.json**
connected to: 127.0.0.1
2014-05-14T10:31:28.506+0200 check 9 10000
2014-05-14T10:31:28.507+0200 imported 10000 objects

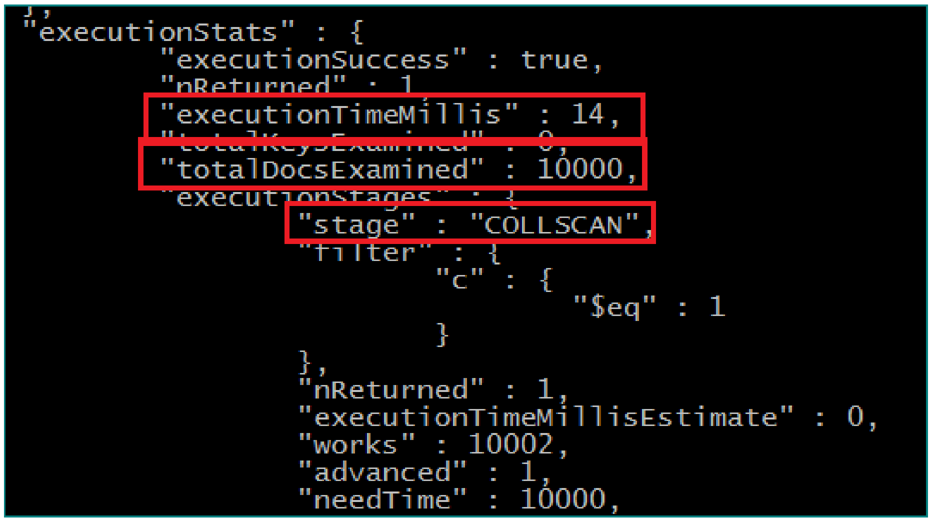
```
> db.foo.find()
{ "_id" : ObjectId("5373294e620bff2007825235"), "a" : 0, "b" : 0, "c" : 0 }
{ "_id" : ObjectId("5373294e620bff2007825236"), "a" : 1, "b" : 1, "c" : 1 }
{ "_id" : ObjectId("5373294e620bff2007825237"), "a" : 2, "b" : 2, "c" : 2 }
….
```

```
> db.foo.ensureIndex({a:1,b:1,c:1})
{
	"createdCollectionAutomatically" : false,
	"numIndexesBefore" : 1,
	"numIndexesAfter" : 2,
	"ok" : 1
}
```

Effectuons une requête qui n’utilise pas d’index

```
> db.foo.find({c:1})
{ "_id" : ObjectId("5373294e620bff2007825236"), "a" : 1, "b" : 1, "c" : 1 }
```

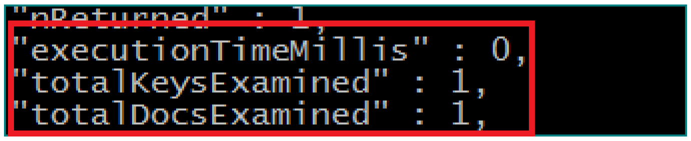
> **db.foo.find({c:1}).explain(“executionStats”)**



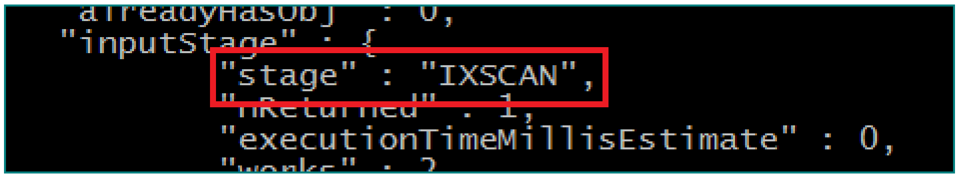
```
},
"executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 1,
        "executionTimeMillis" : 14,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 10000,
        "executionStages" : {
                "stage" : "COLLSCAN",
                "filter" : {
                        "c" : {
                                "$eq" : 1
                        }
                },
                "nReturned" : 1,
                "executionTimeMillisEstimate" : 0,
                "works" : 10002,
                "advanced" : 1,
                "needTime" : 10000,
```

Effectuons une requête qui utilise l’index

> **db.foo.find({a:1}).explain(“executionStats”)**



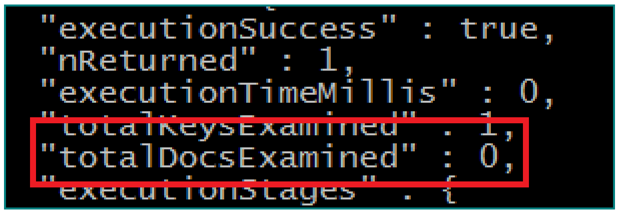
```
"nReturned" : 1,
"executionTimeMillis" : 0,
"totalKeysExamined" : 1,
"totalDocsExamined" : 1,
```



```
   "alreadyHasObj" : 0,
   "inputStage" : {
           "stage" : "IXSCAN",
           "nReturned" : 1,
           "executionTimeMillisEstimate" : 0,
           "works" : 2
```

> **db.foo.find({a:500},{a:1,b:1,_id:0})**
{ "a" : 500, "b" : 500 }

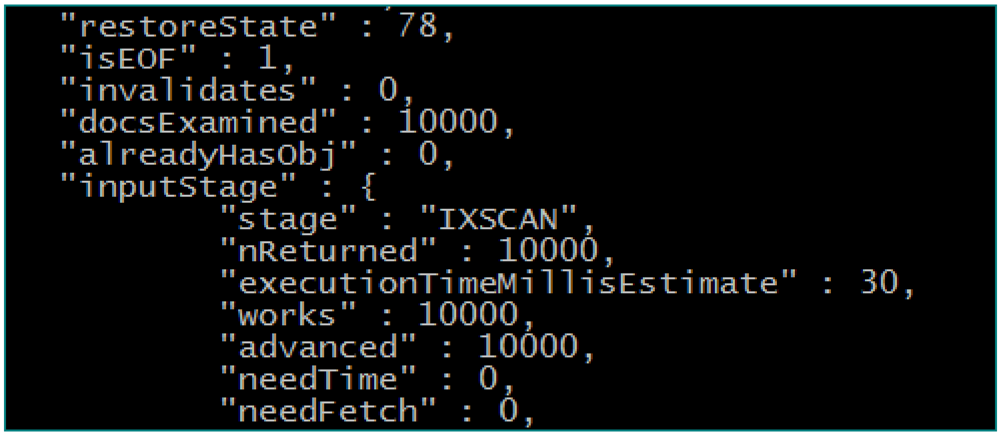
> **db.foo.find({a:500},{a:1,b:1,_id:0}).explain(“executionStats”)**



```
 "executionSuccess" : true,
 "nReturned" : 1,
 "executionTimeMillis" : 0,
 "totalKeysExamined" : 1,
 "totalDocsExamined" : 0,
 "executionStages" : {
```

Maintenant montrons que mongoDB peut utiliser l’index pour la partie tri et pas sur la partie de recherche

> **db.foo.find({$and:[{c:{$gt:250}},{c:{$lte:500}}]}).sort({a:1}).explain(“executionStats”)**



```
   "restoreState" : 78,
   "isEOF" : 1,
   "invalidates" : 0,
   "docsExamined" : 10000,
   "alreadyHasObj" : 0,
   "inputStage" : {
           "stage" : "IXSCAN",
           "nReturned" : 10000,
           "executionTimeMillisEstimate" : 30,
           "works" : 10000,
           "advanced" : 10000,
           "needTime" : 0,
           "needFetch" : 0,
```