

---

No-SQL versus relational databases  
*Course Assignment 2*

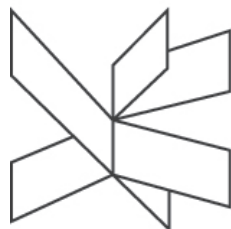
---

Jordi Rafael Lazo Florensa

27th March 2022

Software Technology Engineering

IT-NSQ1-S22



**VIA University  
College**

# 1 Question 1

Set up the database with a sharded cluster with at least two shards. At least one of the shards should be a replica set containing at least 3 nodes. There are 10 mongo instances.

- 3 for the configuration servers, which are replicated.
- 6 for the sharding cluster; 3 for each shard, which are replicated.
- 1 for mongos, the interface of the sharding cluster.

For starting all 10 services, you need to have installed docker-compose. Execute the following command for building and running the containers for each instance:

```
docker-compose -f docker-compose.yaml up -d
```

First, for configuring the replication of the configuration servers, access to one of the instances with:

```
> mongo localhost:40001
```

Being in the mongo cli, execute the following code for starting the replication:

---

```
1 rs.initiate(  
2 {   _id: "cfgrs",  
3     configsvr: true,  
4     members: [  
5         { _id : 0, host : "cfgsvr1:27017" },  
6         { _id : 1, host : "cfgsvr2:27017" },  
7         { _id : 2, host : "cfgsvr3:27017" }  
8     ]  
9 }
```

---

Secondly, start the replication of the first and second shard:

First shard:

```
> mongo localhost:50001
```

And then:

---

```
1 rs.initiate(  
2 {   _id: "shard1rs",  
3     members: [  
4         { _id : 0, host : "shard1svr1:27017" },  
5         { _id : 1, host : "shard1svr2:27017" },  
6         { _id : 2, host : "shard1svr3:27017" }  
7     ]  
8 }  
9 )
```

---

Second shard:

```
> mongo localhost:50004
```

And then:

---

```
1 rs.initiate(
2 {   _id: "shard2rs",
3     members: [
4         { _id : 0, host : "shard2svr1:27017" },
5         { _id : 1, host : "shard2svr2:27017" },
6         { _id : 2, host : "shard2svr3:27017" }
7     ]
8 }
9 )
```

---

Thirdly, enter to “mongos” sharding interface and add both created shards:

```
> mongo localhost:60000
```

And then:

---

```
1 sh.addShard("shard1rs/shard1svr1:27017,shard1svr2:27017,shard1svr3:27017")
2 sh.addShard("shard2rs/shard2svr1:27017,shard2svr2:27017,shard2svr3:27017")
```

---

## 2 Question 2

Design a MongoDB model for the bookstore model from the 1st course assignment. Make note of the choices you make and why. You can document the model using example documents.

---

```
1 Books
2 {_id: ObjectId
3  isbn: str
4  title: str
5  format: str
6  price: float
7  units: int
8  pages: int
9  condition: str
10 avgReview: float
11 totalReviews: int
12 category: ObjectId
13 languages:[
14     languageId: int
15 ]
16 authors:[
17     authorId: int
18 ]
19 genres:[
20     genreId: int
21 ]
22 characters:[
23     characterId: int
24 ]
25 }
```

---

```
1 Customers
2 {customerId: int
3 name: str
4 email: str
5 shippingAddress: str
6 phone: str
7 orders:[
8     orderId
9 ]
10 }
```

---

```
1 Orders
2 {orderId: int
3 customerId: int
4 date: str
5 totalPrice: decimal
6 shippingPrice: decimal
7 orderItems:[
8     orderItemId
9 ]
10 }
```

---

```
1 OrderItems:
2 {orderId: int
3 items:[
4     bookId: int
5     name: string
6     quantity: int
7     unitPrice: decimal
8     discount: decimal
9     totalPrice: decimal
10 ]
11 }
```

---

```
1 Characters
2 {_id
3 categoryId
4 name
5 books:[
6     bookId
7 ]
8 }
```

---

```
1 Genres
2 {_id
3 categoryId
4 name
```

```

5 books:[
6     bookId
7 ]
8 }

```

---

```

1 Categories
2 {_id
3 name
4 parentCategoryId
5 characters:[
6     characterId
7 ]
8 genres:[
9     genreId
10 ]
11 books:[
12     bookId
13 ]
14 }

```

---

```

1 Author
2 {_id
3 name
4 books:[
5     bookId
6 ]
7 }

```

---

```

1 Language
2 {_id
3 name
4 books:[
5     bookId
6 ]
7 }

```

---

### 3 Question 3

Design and create schemas for the collections in your model.

```

1 db.createCollection("books", {
2     validator: {
3         $jsonSchema: {
4             bsonType: "object",
5             required: ["isbn", "title", "format", "price", "units", "pages",
6                 "condition", "avgReview", "totalReviews", "category"],
7             properties: {

```

```

8         bsonType: "string"
9     },
10    title: {
11        bsonType: "string"
12    },
13    format: {
14        bsonType: "string"
15    },
16    price: {
17        bsonType: "decimal"
18    },
19    units: {
20        bsonType: "int"
21    },
22    pages: {
23        bsonType: "int"
24    },
25    condition: {
26        enum: ["Good", "Bad"]
27    },
28    avgReview: {
29        bsonType: "decimal"
30    },
31    totalReviews: {
32        bsonType: "int"
33    },
34    category: {
35        bsonType: "objectId"
36    },
37    languages: {
38        bsonType: "array",
39        uniqueItems: true,
40        items: {
41            bsonType: "objectId"
42        }
43    },
44    authors: {
45        bsonType: "array",
46        uniqueItems: true,
47        items: {
48            bsonType: "objectId"
49        }
50    },
51    genres: {
52        bsonType: "array",
53        uniqueItems: true,
54        items: {
55            bsonType: "objectId"
56        }
57    },
58    characters: {

```

```

59         bsonType: "array",
60         uniqueItems: true,
61         items: {
62             bsonType: "objectId"
63         }
64     }
65 }
66 }
67 }
68 })
69
70
71 db.createCollection("customers", {
72     validator: {
73         $jsonSchema: {
74             bsonType: "object",
75             required: ["name", "email", "shippingAddress", "phone"],
76             properties: {
77                 name: {
78                     bsonType: "string",
79                 },
80                 email: {
81                     bsonType: "string"
82                 },
83                 shippingAddress: {
84                     bsonType: "string"
85                 },
86                 phone: {
87                     bsonType: "string"
88                 },
89                 orders: {
90                     bsonType: "array",
91                     uniqueItems: true,
92                     items: {
93                         bsonType: "objectId"
94                     }
95                 }
96             }
97         }
98     }
99 })
100
101
102 db.createCollection("orders", {
103     validator: {
104         $jsonSchema: {
105             bsonType: "object",
106             required: ["customerId", "date", "totalPrice", "shippingPrice",
107                 "lines"],
108             properties: {
109                 customerId: {

```

```

109         bsonType: "objectId",
110     },
111     date: {
112         bsonType: "string"
113     },
114     totalPrice: {
115         bsonType: "decimal"
116     },
117     shippingPrice: {
118         bsonType: "decimal"
119     },
120     lines: {
121         bsonType: "array",
122         uniqueItems: true,
123         items: {
124             bsonType: "objectId"
125         }
126     }
127 }
128 }
129 }
130 })
131
132 db.createCollection("orderItems", {
133     validator: {
134         $jsonSchema: {
135             bsonType: "object",
136             required: ["orderId", "lineItems"],
137             properties: {
138                 orderId: {
139                     bsonType: "objectId",
140                 },
141                 lineItems: {
142                     bsonType: "array",
143                     uniqueItems: true,
144                     items: {
145                         bsonType: "object",
146                         required: ["bookId", "name", "quantity", "unitPrice", "discount",
147                             "totalPrice"],
148                         properties: {
149                             bookId: {
150                                 bsonType: "objectId"
151                             },
152                             name: {
153                                 bsonType: "string"
154                             },
155                             quantity: {
156                                 bsonType: "int"
157                             },
158                             unitPrice: {
159                                 bsonType: "decimal"

```



```

159         },
160         discount: {
161             bsonType: "decimal"
162         },
163         totalPrice: {
164             bsonType: "decimal"
165         },
166     }
167 }
168 }
169 }
170 }
171 }
172 })
173
174 db.createCollection("characters", {
175     validator: {
176         $jsonSchema: {
177             bsonType: "object",
178             required: ["categoryId", "name"],
179             properties: {
180                 categoryId: {
181                     bsonType: "objectId",
182                 },
183                 name: {
184                     bsonType: "string"
185                 },
186                 books: {
187                     bsonType: "array",
188                     uniqueItems: true,
189                     items: {
190                         bsonType: "objectId"
191                     }
192                 }
193             }
194         }
195     }
196 })
197
198 db.createCollection("genres", {
199     validator: {
200         $jsonSchema: {
201             bsonType: "object",
202             required: ["categoryId", "name"],
203             properties: {
204                 categoryId: {
205                     bsonType: "objectId",
206                 },
207                 name: {
208                     bsonType: "string"
209                 },

```

```

210         books: {
211             bsonType: "array",
212             uniqueItems: true,
213             items: {
214                 bsonType: "objectId"
215             }
216         }
217     }
218 }
219 }
220 })
221
222 db.createCollection("categories", {
223     validator: {
224         $jsonSchema: {
225             bsonType: "object",
226             required: ["name"],
227             properties: {
228                 name: {
229                     bsonType: "string",
230                 },
231                 parentCategoryId: {
232                     bsonType: "objectId"
233                 },
234                 characters: {
235                     bsonType: "array",
236                     uniqueItems: true,
237                     items: {
238                         bsonType: "objectId"
239                     }
240                 },
241                 genres: {
242                     bsonType: "array",
243                     uniqueItems: true,
244                     items: {
245                         bsonType: "objectId"
246                     }
247                 },
248                 books: {
249                     bsonType: "array",
250                     uniqueItems: true,
251                     items: {
252                         bsonType: "objectId"
253                     }
254                 }
255             }
256         }
257     }
258 })
259
260

```

```

261 db.createCollection("authors", {
262     validator: {
263         $jsonSchema: {
264             bsonType: "object",
265             required: ["name"],
266             properties: {
267                 name: {
268                     bsonType: "string",
269                 },
270                 books: {
271                     bsonType: "array",
272                     uniqueItems: true,
273                     items: {
274                         bsonType: "objectId"
275                     }
276                 }
277             }
278         }
279     }
280 })
281
282 db.createCollection("languages", {
283     validator: {
284         $jsonSchema: {
285             bsonType: "object",
286             required: ["name"],
287             properties: {
288                 name: {
289                     bsonType: "string",
290                 },
291                 books: {
292                     bsonType: "array",
293                     uniqueItems: true,
294                     items: {
295                         bsonType: "objectId"
296                     }
297                 }
298             }
299         }
300     }
301 })

```

---

## 4 Question 4

### 4.1 Modifying data

1. Sell a book to a customer.

---

```

1     db.orders.update(
2     {__id: orders_ids.John3 },

```

```

3     { $push: {lines: orderItems_ids.John3}}
4   )
5   db.customers.update(
6     {_id: customers_ids.John },
7     { $push: {orders: orders.John3}}
8   )
9   db.books.update(
10    {_id: books_ids.Orwell },
11    { $inc: {units: -1}}
12  )

```

---

2. Change the address of a customer.

```

1   db.customers.update(
2     {name: "Pep"},
3     {$set: {shippingAddress: "Kamtjatka 7, K13"}}
4   )

```

---

3. Add an existing author to a book.

```

1   db.books.update(
2     {title: "Maus"},
3     {$push: {authors: authors_ids.King}}
4   )

```

---

4. Retire the "Space Opera" category and assign all books from that category to the parent category. Don't assume you know the id of the parent category.

```

1   db.books.aggregate([
2     {
3       $lookup: {
4         from: "categories",
5         localField: "category",
6         foreignField: "_id",
7         as: "oldCat"
8       }
9     },
10    { $match: { "oldCat.name": "Soap Opera" } },
11    {
12      $set: {
13        "category": {
14          $first: "$oldCat.parentCategoryId"
15        }
16      }
17    },
18    { $unset: ["oldCat"] },
19    {
20      $merge: {
21        into: "books",
22        whenMatched: "replace",
23        whenNotMatched: "discard"

```

```

24     }
25   }
26   ])
27
28   db.characters.aggregate([
29     {
30       $lookup: {
31         from: "categories",
32         localField: "categoryId",
33         foreignField: "_id",
34         as: "oldCat"
35       }
36     },
37     { $match: { "oldCat.name": "Soap Opera" } },
38     {
39       $set: {
40         "categoryId": {
41           $first: "$oldCat.parentCategoryId"
42         }
43       }
44     },
45     { $unset: ["oldCat"] },
46     {
47       $merge: {
48         into: "characters",
49         whenMatched: "replace",
50         whenNotMatched: "discard"
51       }
52     }
53   ])
54
55   db.genres.aggregate([
56     {
57       $lookup: {
58         from: "categories",
59         localField: "categoryId",
60         foreignField: "_id",
61         as: "oldCat"
62       }
63     },
64     { $match: { "oldCat.name": "Soap Opera" } },
65     {
66       $set: {
67         "categoryId": {
68           $first: "$oldCat.parentCategoryId"
69         }
70       }
71     },
72     { $unset: ["oldCat"] },
73     {
74       $merge: {

```

```

75     into: "genres",
76     whenMatched: "replace",
77     whenNotMatched: "discard"
78   }
79 }
80 ])
81
82 db.categories.aggregate([
83   {
84     $lookup: {
85       from: "categories",
86       localField: "parentCategoryId",
87       foreignField: "_id",
88       as: "parentCat"
89     }
90   },
91   { $match: { "parentCat.name": "Soap Opera" } }
92   ,
93   {
94     $set: {
95       "parentCategoryId": {
96         $first: "$parentCat.parentCategoryId"
97       }
98     }
99   }
100   ,
101   { $unset: ["parentCat"] },
102   {
103     $merge: {
104       into: "categories",
105       whenMatched: "replace",
106       whenNotMatched: "discard"
107     }
108   }
109 ])
110
111 db.categories.remove({ name: "Soap Opera" })

```

---

5. Sell 3 copies of one book and 2 of another in a single order.

---

```

1 db.orders.insertOne (
2   {
3     customerId: customers_ids.Pep,
4     date: "25/03/2022",
5     totalPrice: NumberDecimal(49.95),
6     shippingPrice: NumberDecimal(1.50),
7     lines: []
8   }
9 )
10 orders_ids["Pep2"] = db.orders.findOne({ customerId:
    customers_ids.Pep, date: "25/03/2022" })._id

```

```

11 db.orderItems.insertMany(
12   {
13     orderId: orders_ids.Pep2,
14     lineItems: lineItems[
15       {
16         bookId: books_ids.Maus,
17         name: "Maus",
18         quantity: NumberInt(3),
19         unitPrice: NumberDecimal(9.99),
20         discount: NumberDecimal(0),
21         totalPrice: NumberDecimal(29.97)
22       },
23       {
24         bookId: books_ids.It,
25         name: "It",
26         quantity: NumberInt(2),
27         unitPrice: NumberDecimal(9.99),
28         discount: NumberDecimal(0),
29         totalPrice: NumberDecimal(19.98)
30       }
31     ]
32   }
33 )
34 orderItems_ids["Pep2"] = db.orders.findOne({ orderId: orders_ids.Pep2
35   })._id
36 db.orders.update(
37   {__id: orders_ids.Pep2 },
38   { $push: {lines: orderItems_ids.Pep2}}
39 )
40 db.customers.update(
41   {_id: customers_ids.Pep2 },
42   { $push: {orders: orders.Pep}}
43 )
44 db.books.update(
45   {_id: books_ids.Maus },
46   { $inc: {units: -3}}
47 )
48 db.books.update(
49   {_id: books_ids.It },
50   { $inc: {units: -2}}
51 )

```

---

## 4.2 Querying data

---

```

1 //Query data
2 //1
3 db.authors.aggregate([
4   {$lookup: {
5     from: "books",
6     localField: "_id",

```

```

7         foreignField: "authors",
8         as: "AuthorsBooks"
9     },
10 }
11 ])
12
13 //2
14 db.orders.find({_id: ObjectId("623d937a60d37d56a344ad74")}, {totalPrice : 1})
15
16
17 //3
18 db.customers.aggregate([
19     { $lookup : {from : "orders", localField : "orders" , foreignField : "_id",
20         as : "ord_customer"} },
21     { $match : { name : "John" } },
22     {$project: {"_id":1, "name":1,"totalValue": {$sum:
23         "$ord_customer.totalPrice"}}}
24 ])
25
26 //4
27 db.categories.aggregate([
28     { $lookup : {from : "books", localField : "_id" , foreignField : "category",
29         as : "books_cat"} },
30     { $match : {$and : [{ name : { $ne : "Fiction"}},{ name : { $ne :
31         "History"}}]}},
32     { $project: {"Books":"$books_cat.title", name:1}}
33 ])
34
35 //5
36 db.books.aggregate([
37     {
38         $unwind: "$genres"
39     },
40     {
41         $group:
42         {
43             _id: "$genres",
44             averagePagesByGenre: { $avg: { $sum: "$pages"}}
45         }
46     }
47 ])
48
49 //6
50 db.categories.aggregate([
51     {
52         $lookup: {
53             from: "categories",
54             localField: "categoryId",
55             foreignField: "_id",
56             as: "parentCat"
57         }
58     }
59 ])

```



```

54     },
55     $group: {
56         _id: "$parentCat"
57     }
58 }
59 ])
60
61 //7
62 db.books.aggregate([
63     {
64         $addFields: {
65             arrayLength: {$size: '$authors'}
66         },
67     },
68     {
69         $match: {
70             "arrayLength": {$gt: 1}
71         }
72     },
73     {
74         $group: {
75             _id: "$isbn"
76         }
77     }
78 ])
79
80 //8
81 db.orderItems.aggregate([
82     {
83         $unwind: "$lineItems"
84     },
85     {$group: {
86         _id: "$lineItems.bookId",
87         totalSold: { $sum: "$lineItems.quantity" }
88     }},
89     {$match: {totalSold: {$gt: 1}}},
90     {$lookup:
91         {
92             from: "books",
93             localField: "_id",
94             foreignField: "_id",
95             as: "Book"
96         }
97     },
98     {$group:
99         {
100             _id: "$Book.isbn"
101         }
102     }
103 ])
104 //9

```

```

105 db.orderItems.aggregate([
106   {
107     $unwind: "$lineItems"
108   },
109   {$group: {
110     _id: "$lineItems.bookId",
111     totalSold: { $sum: "$lineItems.quantity" }
112   }}})
113
114 //10
115 db.orderItems.aggregate([
116   {
117     $unwind: "$lineItems"
118   },
119   {$group: {
120     _id: "$lineItems.bookId",
121     totalSold: { $sum: "$lineItems.quantity" }
122   }},
123   {$sort: {
124     totalSold:-1
125   }},
126   {$limit: 10}
127 ])
128
129 //11
130 db.genres.aggregate([
131   {
132     $lookup: {
133       from: "books",
134       localField: "_id",
135       foreignField: "genres",
136       as: "genBooks"
137     }
138   },
139   { $project: { "_id": 1, "name": 1, "booksIds": "$genBooks._id", "genBooks":
140     1 } }},
141   {
142     $lookup: {
143       from: "orderItems",
144       localField: "booksIds",
145       foreignField: "lineItems.bookId",
146       as: "myOrderItems"
147     }
148   },
149   {
150     $addFields: {
151       "myBooks": {
152         $filter:
153           {
154             input: "$myOrderItems.lineItems",
155             as: "li",

```

```

155         cond: {
156             $in: ["$li.bookId", "$booksIds"]
157         }
158     }
159 }
160 }
161 }
162 ])
163
164 //12
165 db.categories.aggregate([
166     {
167         $graphLookup: {
168             from: "categories",
169             startWith: "$parentCategoryId",
170             connectFromField: "parentCategoryId",
171             connectToField: "_id",
172             as: "catHierarchy"
173         }
174     },
175     {
176         "$match": {
177             $or:
178             [
179                 { "$expr": { "$in": ["Science Fiction & Fantasy",
180                                     "$catHierarchy.name"] } },
181                 { "$expr": { "$eq": [ "$name", "Science Fiction & Fantasy"]
182                                     } }
183             ]
184         }
185     },
186     {
187         $group: {_id:"$_id"}
188     },
189     {
190         $lookup: {
191             from: "books",
192             localField: "_id",
193             foreignField: "category",
194             as: "catBooks"
195         }
196     }
197 ])
198
199 //13
200 db.categories.aggregate([
201     {
202         $graphLookup: {
203             from: "categories",
204             startWith: "$parentCategoryId",
205             connectFromField: "parentCategoryId",
206             connectToField: "_id",

```

```

204         as: "catHierarchy"
205     }
206 },
207 {
208     "$match": {
209         $or:
210         [
211             { "$expr": { "$in": ["Science Fiction & Fantasy",
212                                   "$catHierarchy.name"] } },
212             { "$expr": { "$eq": [ "$name", "Science Fiction & Fantasy"] } }
213         ]
214     },
215     {
216         $group: {_id:"$_id"}
217     },
218     {$lookup: {
219         from: "books",
220         localField: "_id",
221         foreignField: "category",
222         as: "catBooks"
223     }},
224     {
225         $group: {_id:"$catBooks.characters"}
226     }
227 ]})
228
229
230
231 //14
232 db.categories.aggregate([
233     {
234         $graphLookup: {
235             from: "categories",
236             startWith: "$_id",
237             connectFromField: "_id",
238             connectToField: "parentCategoryId",
239             as: "catHierarchy"
240         }
241     },
242     { $project: { "_id": 1, "name": 1, "cats": { $concatArrays: [["_id"],
243                                   "$catHierarchy._id"] } } },
244     {
245         $lookup: {
246             from: "books",
247             localField: "cats",
248             foreignField: "category",
249             as: "hierBooks"
250         }
251     },
252     {$project: {"_id":1, "name":1, "totalBooks": {$size: "$hierBooks"}}}

```

## 5 Question 5

Write a report on the experience gained by completing Question 1 through 4 above. The report should contain answers to the questions:

- What were the decisions taken in the modelling?  
One of the most important decisions that we have carried on with is the fact that we only store the Ids of the elements of another collection inside a collection, so we are not embedding. For example, in the books collection, we have arrays for saving the Ids of languages, authors, genres and characters, but to access the elements we have to access the correspondent collection. Another example of a decision is that for categories, we only have a field for parentCategories, and not for child categories.
- Why were these decisions taken?  
By only saving the Id as a key to access the other collections, the database becomes more structured and it costs less space. And by not keeping the child categories we have less redundancy.
- What were the consequences of these decisions?  
To access other collections, we have to make a lot of lookups to get the elements from there. Moreover, it was much harder to populate the database than we originally thought, as we had to take care of saving the ids of all the elements and store the ids in the correspondent collections accordingly. For accessing child categories, we have to use graphLookup.
- What were the difficult and easy parts of the exercise?  
As we didn't see non relational databases before, and more exactly mongodb, we can tell that the whole assignment was a challenge. However, we agree that the querying and populating part was the one that took us more time. We also had problems with the replication and sharding, as it was the first time we were communicating containers locally using docker-compose.
- How does that compare to relational databases?  
The fact of working with a no-sql database makes the navigation between collections different and sharing data more difficult. This makes the querying part less intuitive.
- What are the advantages and disadvantages of MongoDB over relational databases for this exercise?  
No-sql databases like MongoDB give us more versatility when data increases or changes, although we didn't include that much population in our collections. Two main disadvantages were that we required a lot of complex code for populating the database for testing purposes and that the complexity raised drastically for some queries depending on the data modeling.