No-SQL versus relational databases

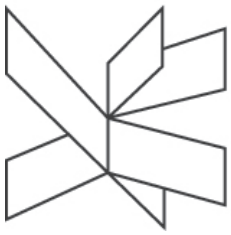
Course Assignment 2

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:

```
rs.initiate(
{   _id: "cfgrs",
    configsvr: true,
    members: [
        { _id : 0, host : "cfgsvr1:27017" },
        { _id : 1, host : "cfgsvr2:27017" },
        { _id : 2, host : "cfgsvr3:27017" }
    ]
}
```

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

First shard:

```
> mongo localhost:50001
```

And then:

```
rs.initiate(
{   _id: "shard1rs",
    members: [
        { _id : 0, host : "shard1svr1:27017" },
        { _id : 1, host : "shard1svr2:27017" },
        { _id : 2, host : "shard1svr3:27017" }
    ]
}
)
```

Second shard:

```
> mongo localhost:50004
```

And then:

```
rs.initiate(
{   _id: "shard2rs",
    members: [
        { _id : 0, host : "shard2svr1:27017" },
        { _id : 1, host : "shard2svr2:27017" },
        { _id : 2, host : "shard2svr3:27017" }
    ]
}
)
```

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

```
> mongo localhost:60000
```

And then:

```
sh.addShard("shard1rs/shard1svr1:27017,shard1svr2:27017,shard1svr3:27017")
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.

```
Books
{_id: ObjectId
isbn: str
title: str
format: str
price: float
units: int
pages: int
condition: str
avgReview: float
totalReviews: int
category: ObjectId
languages:[
    languageId: int
    ]
authors:[
    authorId: int
    ]
genres:[
    genreId: int
    ]
characters:[
    characterId: int
    ]
}
```

```
Customers
{customerId: int
name: str
email: str
shippingAddress: str
phone: str
orders:[
    orderId
    ]
}
```

```
Orders
{orderId: int
customerId: int
date: str
totalPrice: decimal
shippingPrice: decimal
orderItems:[
    orderItemId
    ]
}
```

```
OrderItems:
{orderId: int
items:[
    bookId: int
    name: string
    quantity: int
    unitPrice: decimal
    discount: decimal
    totalPrice: decimal
    ]
}
```

```
Characters
{_id
categoryId
name
books:[
    bookId
    ]
}
```

```
Genres
{_id
categoryId
name
```

```
books:[
    bookId
    ]
}
```

```
Categories
{_id
name
parentCategoryId
characters:[
    characterId
    ]
genres:[
    genreId
    ]
books:[
    bookId
    ]
}
```

```
Author
{_id
name
books:[
    bookId
    ]
}
```

```
Language
{_id
name
books:[
    bookId
    ]
}
```

3 Question 3

Design and create schemas for the collections in your model.

```
db.createCollection("books", {
    validator: {
      $jsonSchema: {
        bsonType: "object",
        required: ["isbn", "title", "format", "price", "units", "pages",
            "condition", "avgReview", "totalReviews", "category"],
        properties: {
          isbn: {
```

```
        bsonType: "string"
      },
      title: {
        bsonType: "string"
      },
      format: {
        bsonType: "string"
      },
      price: {
        bsonType: "decimal"
      },
      units: {
        bsonType: "int"
      },
      pages: {
        bsonType: "int"
      },
      condition: {
        enum: ["Good", "Bad"]
      },
      avgReview: {
        bsonType: "decimal"
      },
      totalReviews: {
        bsonType: "int"
      },
      category: {
        bsonType: "objectId"
      },
      languages: {
        bsonType: "array",
        uniqueItems: true,
        items: {
          bsonType: "objectId"
        }
      },
      authors: {
        bsonType: "array",
        uniqueItems: true,
        items: {
          bsonType: "objectId"
        }
      },
      genres: {
        bsonType: "array",
        uniqueItems: true,
        items: {
          bsonType: "objectId"
        }
      },
      characters: {
```

```
          bsonType: "array",
          uniqueItems: true,
          items: {
            bsonType: "objectId"
          }
        }
      }
    }
  }
})


db.createCollection("customers", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name", "email", "shippingAddress", "phone"],
      properties: {
        name: {
          bsonType: "string",
        },
        email: {
          bsonType: "string"
        },
        shippingAddress: {
          bsonType: "string"
        },
        phone: {
          bsonType: "string"
        },
        orders: {
          bsonType: "array",
          uniqueItems: true,
          items: {
            bsonType: "objectId"
          }
        }
      }
    }
  }
})


db.createCollection("orders", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["customerId", "date", "totalPrice", "shippingPrice",
          "lines"],
      properties: {
        customerId: {
```

```
            bsonType: "objectId",
          },
          date: {
            bsonType: "string"
          },
          totalPrice: {
            bsonType: "decimal"
          },
          shippingPrice: {
            bsonType: "decimal"
          },
          lines: {
            bsonType: "array",
            uniqueItems: true,
            items: {
              bsonType: "objectId"
            }
          }
        }
      }
    }
  })

  db.createCollection("orderItems", {
    validator: {
      $jsonSchema: {
        bsonType: "object",
        required: ["orderId", "lineItems"],
        properties: {
          orderId: {
            bsonType: "objectId",
          },
          lineItems: {
            bsonType: "array",
            uniqueItems: true,
            items: {
              bsonType: "object",
              required: ["bookId", "name", "quantity", "unitPrice", "discount",
                  "totalPrice"],
              properties: {
                bookId: {
                  bsonType: "objectId"
                },
                name: {
                  bsonType: "string"
                },
                quantity: {
                  bsonType: "int"
                },
                unitPrice: {
                  bsonType: "decimal"
```

```
            },
            discount: {
              bsonType: "decimal"
            },
            totalPrice: {
              bsonType: "decimal"
            },
          }
        }
      }
    }
  }
})

db.createCollection("characters", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["categoryId", "name"],
      properties: {
        categoryId: {
          bsonType: "objectId",
        },
        name: {
          bsonType: "string"
        },
        books: {
          bsonType: "array",
          uniqueItems: true,
          items: {
            bsonType: "objectId"
          }
        }
      }
    }
  }
})

db.createCollection("genres", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["categoryId", "name"],
      properties: {
        categoryId: {
          bsonType: "objectId",
        },
        name: {
          bsonType: "string"
        },
```

```
        books: {
          bsonType: "array",
          uniqueItems: true,
          items: {
            bsonType: "objectId"
          }
        }
      }
    }
  }
})

db.createCollection("categories", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name"],
      properties: {
        name: {
          bsonType: "string",
        },
        parentCategoryId: {
          bsonType: "objectId"
        },
        characters: {
          bsonType: "array",
          uniqueItems: true,
          items: {
            bsonType: "objectId"
          }
        },
        genres: {
          bsonType: "array",
          uniqueItems: true,
          items: {
            bsonType: "objectId"
          }
        },
        books: {
          bsonType: "array",
          uniqueItems: true,
          items: {
            bsonType: "objectId"
          }
        }
      }
    }
  }
})
```

```
db.createCollection("authors", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name"],
      properties: {
        name: {
          bsonType: "string",
        },
        books: {
          bsonType: "array",
          uniqueItems: true,
          items: {
            bsonType: "objectId"
          }
        }
      }
    }
  }
})

db.createCollection("languages", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name"],
      properties: {
        name: {
          bsonType: "string",
        },
        books: {
          bsonType: "array",
          uniqueItems: true,
          items: {
            bsonType: "objectId"
          }
        }
      }
    }
  }
})
```

4 Question 4

4.1 Modifying data

1. Sell a book to a customer.

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

```
    { $push: {lines: orderItems_ids.John3}}
  )
  db.customers.update(
    {_id: customers_ids.John },
    { $push: {orders: orders.John3}}
  )
  db.books.update(
    {_id: books_ids.Orwell },
    { $inc: {units: -1}}
  )
```

2. Change the address of a customer.

```
    db.customers.update(
    {name: "Pep"},
    {$set: {shippingAddress: "Kamtjatka 7, K13"}}
    )
```

3. Add an existing author to a book.

```
    db.books.update(
    {title: "Maus"},
    {$push: {authors: authors_ids.King}}
    )
```

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.

```
    db.books.aggregate([
  {
    $lookup: {
      from: "categories",
      localField: "category",
      foreignField: "_id",
      as: "oldCat"
    }
  },
  { $match: { "oldCat.name": "Soap Opera" } },
  {
    $set: {
      "category": {
        $first: "$oldCat.parentCategoryId"
      }
    }
  },
  { $unset: ["oldCat"] },
  {
    $merge: {
      into: "books",
      whenMatched: "replace",
      whenNotMatched: "discard"
```

```
    }
  }
])

db.characters.aggregate([
  {
    $lookup: {
      from: "categories",
      localField: "categoryId",
      foreignField: "_id",
      as: "oldCat"
    }
  },
  { $match: { "oldCat.name": "Soap Opera" } },
  {
    $set: {
      "categoryId": {
        $first: "$oldCat.parentCategoryId"
      }
    }
  },
  { $unset: ["oldCat"] },
  {
    $merge: {
      into: "characters",
      whenMatched: "replace",
      whenNotMatched: "discard"
    }
  }
])

db.genres.aggregate([
  {
    $lookup: {
      from: "categories",
      localField: "categoryId",
      foreignField: "_id",
      as: "oldCat"
    }
  },
  { $match: { "oldCat.name": "Soap Opera" } },
  {
    $set: {
      "categoryId": {
        $first: "$oldCat.parentCategoryId"
      }
    }
  },
  { $unset: ["oldCat"] },
  {
    $merge: {
```

```
      into: "genres",
      whenMatched: "replace",
      whenNotMatched: "discard"
    }
  }
])

db.categories.aggregate([
  {
    $lookup: {
      from: "categories",
      localField: "parentCategoryId",
      foreignField: "_id",
      as: "parentCat"
    }
  },
  { $match: { "parentCat.name": "Soap Opera" } }
  ,
  {
    $set: {
      "parentCategoryId": {
        $first: "$parentCat.parentCategoryId"
      }
    }
  }
  ,
  { $unset: ["parentCat"] },
  {
    $merge: {
      into: "categories",
      whenMatched: "replace",
      whenNotMatched: "discard"
    }
  }
])

db.categories.remove({ name: "Soap Opera" })
```

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

```
db.orders.insertOne (
    {
      customerId: customers_ids.Pep,
      date: "25/03/2022",
      totalPrice: NumberDecimal(49.95),
      shippingPrice: NumberDecimal(1.50),
      lines: []
    }
   )
   orders_ids["Pep2"] = db.orders.findOne({ customerId:
       customers_ids.Pep, date: "25/03/2022" })._id
```

```
db.orderItems.insertMany(
 {
   orderId: orders_ids.Pep2,
   lineItems: lineItems[
     {
       bookId: books_ids.Maus,
       name: "Maus",
       quantity: NumberInt(3),
       unitPrice: NumberDecimal(9.99),
       discount: NumberDecimal(0),
       totalPrice: NumberDecimal(29.97)
     },
     {
       bookId: books_ids.It,
       name: "It",
       quantity: NumberInt(2),
       unitPrice: NumberDecimal(9.99),
       discount: NumberDecimal(0),
       totalPrice: NumberDecimal(19.98)
     }
   ]
 }
)
orderItems_ids["Pep2"] = db.orders.findOne({ orderId: orders_ids.Pep2
    }).._id
db.orders.update(
 {__id: orders_ids.Pep2 },
 { $push: {lines: orderItems_ids.Pep2}}
)
db.customers.update(
 {_id: customers_ids.Pep2 },
 { $push: {orders: orders.Pep}}
)
db.books.update(
 {_id: books_ids.Maus },
 { $inc: {units: -3}}
)
db.books.update(
 {_id: books_ids.It },
 { $inc: {units: -2}}
)
```

4.2 Querying data

```
//Query data
//1
db.authors.aggregate([
  {$lookup: {
      from: "books",
      localField: "_id",
```

```
      foreignField: "authors",
      as: "AuthorsBooks"
  },
}
])

//2
db.orders.find({_id: ObjectId("623d937a60d37d56a344ad74")}, {totalPrice : 1})


//3
db.customers.aggregate([
  { $lookup : {from : "orders", localField : "orders" , foreignField : "_id",
     as : "ord_customer"} },
  { $match : { name : "John" } },
{$project: {"_id":1, "name":1,"totalValue": {$sum:
   "$ord_customer.totalPrice"}}}
])

//4
db.categories.aggregate([
  { $lookup : {from : "books", localField : "_id" , foreignField : "category",
     as : "books_cat"} },
  { $match : {$and : [{ name : { $ne : "Fiction"}},{ name : { $ne :
     "History"}}]}},
  { $project: {"Books":"$books_cat.title", name:1}}
])


//5
db.books.aggregate([
    {
      $unwind: "$genres"
    },
    {
      $group:
      {
        _id: "$genres",
        averagePagesByGenre: { $avg: { $sum: "$pages"}}
      }
    }
  ])

  //6
  db.categories.aggregate([
    {
      $lookup: {
        from: "categories",
        localField: "categoryId",
        foreignField: "_id",
        as: "parentCat"
```

```
    },
      $group: {
        _id: "$parentCat"
      }
    }
  ])

  //7
  db.books.aggregate([
    {
      $addFields: {
        arrayLength: {$size: '$authors'}
      },
    },
    {
      $match: {
        "arrayLength": {$gt: 1}
      }
    },
    {
      $group: {
        _id: "$isbn"
    }
    }
  ])

  //8
  db.orderItems.aggregate([
    {
      $unwind: "$lineItems"
    },
    {$group: {
      _id: "$lineItems.bookId",
      totalSold: { $sum: "$lineItems.quantity"}
    }},
    {$match: {totalSold: {$gt: 1}}},
     {$lookup:
        {
            from: "books",
            localField: "_id",
            foreignField: "_id",
            as: "Book"
        }
    },
    {$group:
        {
            _id: "$Book.isbn"
        }}
   ])

//9
```

```
db.orderItems.aggregate([
  {
    $unwind: "$lineItems"
  },
  {$group: {
    _id: "$lineItems.bookId",
    totalSold: { $sum: "$lineItems.quantity"}
  }}])

//10
db.orderItems.aggregate([
  {
    $unwind: "$lineItems"
  },
  {$group: {
    _id: "$lineItems.bookId",
    totalSold: { $sum: "$lineItems.quantity"}
  }},
  {$sort: {
    totalSold:-1
  }},
  {$limit: 10}
])

//11
db.genres.aggregate([
  {
      $lookup: {
          from: "books",
          localField: "_id",
          foreignField: "genres",
          as: "genBooks"
      }
  },
  { $project: { "_id": 1, "name": 1, "booksIds": "$genBooks._id", "genBooks":
      1 } },
  {
      $lookup: {
          from: "orderItems",
          localField: "booksIds",
          foreignField: "lineItems.bookId",
          as: "myOrderItems"
      }
  },
  {
      $addFields: {
          "myBooks": {
              $filter:
              {
                  input: "$myOrderItems.lineItems",
                  as: "li",
```

```
                cond: {
                    $in: ["$li.bookId", "$booksIds"]
                }
            }
        }
    }
  }
])

//12
db.categories.aggregate([
  {
      $graphLookup: {
          from: "categories",
          startWith: "$parentCategoryId",
          connectFromField: "parentCategoryId",
          connectToField: "_id",
          as: "catHierarchy"
      }
    },
    {
        "$match": {
           $or:
               [
                   { "$expr": { "$in": ["Science Fiction & Fantasy",
                       "$catHierarchy.name"] } },
                   { "$expr": { "$eq": [ "$name", "Science Fiction & Fantasy"]
                       } }
               ]
        }
    },
    {
        $group: {_id:"$_id"}
    },
    {$lookup: {
        from: "books",
        localField: "_id",
        foreignField: "category",
        as: "catBooks"
    }}
 ])


 //13
 db.categories.aggregate([
  {
      $graphLookup: {
          from: "categories",
          startWith: "$parentCategoryId",
          connectFromField: "parentCategoryId",
          connectToField: "_id",
```

```
            as: "catHierarchy"
        }
    },
    {
        "$match": {
            $or:
                [
                    { "$expr": { "$in": ["Science Fiction & Fantasy",
                        "$catHierarchy.name"] } },
                    { "$expr": { "$eq": [ "$name", "Science Fiction & Fantasy"] }
                        }
                ]
        }
    },
    {
        $group: {_id:"$_id"}
    },
    {$lookup: {
      from: "books",
      localField: "_id",
      foreignField: "category",
      as: "catBooks"
}},
{
    $group: {_id:"$catBooks.characters"}
}
])


//14
db.categories.aggregate([
  {
      $graphLookup: {
          from: "categories",
          startWith: "$_id",
          connectFromField: "_id",
          connectToField: "parentCategoryId",
          as: "catHierarchy"
      }
  },
  { $project: { "_id": 1, "name": 1, "cats": { $concatArrays: [["$_id"],
      "$catHierarchy._id"] } } },
  {
      $lookup: {
          from: "books",
          localField: "cats",
          foreignField: "category",
          as: "hierBooks"
      }
  },
  {$project: {"_id":1, "name":1, "totalBooks": {$size: "$hierBooks"}}}
```

```
252 ])
```

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.