VIA University College

Authors: Xavier Nadal, Arnau Molins, Martí La Rosa

Question 1 - Set up database

There are 10 mongo instances.

1. 3 for the configuration servers, which are replicated.
2. 6 for the sharding cluster; 3 for each shard, which are replicated.
3. 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
```

```
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
```

```
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
```

```
sh.addShard("shard1rs/shard1svr1:27017,shard1svr2:27017,shard1svr3:2701
7")


sh.addShard("shard2rs/shard2svr1:27017,shard2svr2:27017,shard2svr3:2701
7")
```

Question 2 - Model database

```
Books
{
  _id: ObjectId
  isbn: str
  title: str
  format: str
  price: float
  units: int
  pages: int
  condition: str
  avgReview: float
  totalReviews: int
  category: ObjectId
```

For books,here 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.

we keep the same fields as the SQL version, but with the addition of arrays with the ids for

```
  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:
```

documents in other collections (languages, authors,genres,characters).

Customers only need to know about the id of their orders, so an array of ids for orders should be enough.

Orders have to keep the ids of the ordered items. In this case, OrderItems keeps multiple ordered items. The reason why we didn’t embed the ordered items in Orders is because we were worried about the 16 MB limit of each document. In the end we concluded that we

```
    [
      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:
```

shouldn’t worry about this size limit, but we kept the design.

OrderItems keep the ordered items of an order. Each ordered item references a book.

If a category has a parent, it will be specified in parentCategoryId, but there is no way to know if a category has subcategories, except for

```
    [
      bookId
    ]
}

Author
{
  _id
  name
  books:
    [
      bookId
    ]
}

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

graphLookup.

Question 3 - Create schemas

Check *collectionStructure.js* file for schema creation and *PoblateMongo.js* for populating the database.

Question 4 - Work with data

Check *modify.js* and *query.js* files for modify and query MongoDB commands.

VIA University College

Question 5 - Report

What were the decisions taken in the modeling?

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.