# Non-relational Database: Lab 2

Jordi Nadeu Ferran

## 1 Advanced queries in MongoDB

In this section we'll continue using the restaurants collection we imported in lab session 1 to answer the following questions:

1. **Return the list of boroughs ranked by the number of American restaurants in it. That is, for each borough, find how many restaurants serve American cuisine and print the borough and the number of such restaurants sorted by this number**

```
db.restaurants.aggregate([
    { $match: { cuisine: "American" }},
    { $group: { _id: "$borough", count: {$count: {}}}},
    { $sort: { count: 1 }}
])
```

2. **Find the top 5 American restaurants in Manhattan that have the highest total score. Return for each restaurant the restaurants' name and the total score.** *Hint: you can use $unwind.*

```
db.restaurants.aggregate([
    { $match: { borough: "Manhattan", cuisine: "American" }},
    { $unwind: "$grades" },
    { $group: {_id: "$name", totalScore: {$sum: "$grades.score"}}},
    { $limit: 5 }
])
```

3. **Consider a rectangle area on the location field, in which the vertices are [ -74, 40.5 ], [ -74, 40.7 ], [ -73.5, 40.5 ] and [ -73.5, 40.7 ]. Find the number of restaurants in this area that have received a grade score (at least one) more than 50.** *Hint: Use the $geoWithin and $box.*

```
// Create geospatial index
db.collection.createIndex({"address.coord":"2d"})
db.restaurants.aggregate([
    {
        $match: {
            "grades.score": { $gt: 50 },
            "address.coord": {
                $geoWithin: {
                    $box: [[-74, 40.5], [-73.5, 40.7]]}}
        }
    },
    { $count: "Number of restaurants" }
])
```

4. **Find the names of restaurants near Empire State Building (which is located at lat 0.7583243, lon -74.0179141) in a max distance of 1.5km.** *Hint: use $nearSphere.*

```
db.restaurants.createIndex({ "address.coord": "2dsphere" })

db.restaurants.find({
    location: {
        $nearSphere: {
            $geometry: {
                type: "Point",
                coordinates: [-74.0179141, 40.7583243]
            },
            $maxDistance: 1500
        }
    }
}, {name: 1})
```

## 2 Replication

Replication is a way of keeping identical copies of your data on multiple servers. Repli- cation keeps your application running and your data safe, even if something happens to one or more of your servers. In a production environment is recommended to have some sort of replication to minimize downtimes.

With MongoDB you set up replication by creating a replica set. A replica set is a group of servers with one primary (server taking writes) and multiple secondaries (servers that keep copies of the primary's data). If the primary crashes, the secondaries can elect a new primary from amongst themselves.

The most important concept about replica sets is that they are all about majorities:

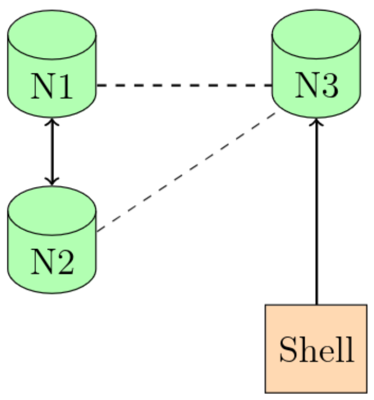
1. You need a majority of members to elect a primary
2. A primary can only stay primary as long as it can reach a majority
3. A write is safe when it's been replicated to a majority

Majority is defined to be "more than half of all members in the set". When a minority of members is available, all members will be secondaries.

```
db.coll.insertOne({count:1002})
```

- **Question 1: What is the result of the previous insert? Explain why this output occurs**

  Now there is only one mongod running. Let's call it Node 3. From now on we'll assume that all nodes are running but due to some network problems they are unreachable from our point of view. This means Node 1 and Node 2 are up and talking to each other and one of them is the primary of the set. Since Node 3 is isolated from the rest, it is in minority.



  Other clients are inserting new documents in N1 (the primary) but due to connectivity problems these changes are not replicated to N3.

*Answer:* The insert operation will fail because Node 3, being the only running node, cannot become the primary without a majority (at least two nodes). MongoDB replica sets require a majority of nodes to elect a primary and perform write operations.

- **Question 2: What is the output of db.coll.find().sort(count: -1).limit(1)?**

  *Answer:* The query returns a output with the highest count value document available on Node 3's dataset before the isolation occurred. Any updates made to the primary after the network partition are not visible to Node 3.

- **Question 3: Based on the described situation and the observed behavior, which two properties of the CAP Theorem are being fulfilled?**

  *Answer:* The two properties are being fulfilled are Consistency (C) and Partition Tolerance (P).

  In MongoDB's replica set, consistency is being ensured because every read operation on the replica set returns the most recent data that has been acknowledged by a majority of the nodes in our case.

  And for the Partition tolerance, the system continues to operate despite the network partition that isolates Node 3 from Nodes 1 and 2. The remaining nodes (Node 1 and Node 2) can still elect a primary, and the primary can continue accepting write.

- **Question 4: Let's say that in this specific case we need consistency and availability. In other words, we want to avoid reading stale data in case of network partition. How can we achieve that?** *Hint: check out all the options of find().*

  *Answer:* To achieve both Consistency and Availability while avoiding reading stale data in case of a network partition, you can adjust the behavior of MongoDB's find() operation by setting the read concern to "majority". This ensures that your application only reads data that has been written and acknowledged by a majority of the nodes in the replica set, thus preventing stale reads.

  We have to use readConcern("majority") and readPreference("primary") like the following example:

  ```
  db.coll.find({ }).readConcern("majority").readPreference("primary")
  ```