Barcelona School of Economics

# 23D020: Big Data Management for Data Science
# Lab 1: Document Stores

## Training

## A Documentation and Installation

- Check the MongoDB 'FAQ' page (`https://diligent-skirt-36b.notion.site/MongoDB-2f1db119176c4be7886edfac2062d3cc?pvs=4`)
- Check the MongoDB 'How To' page (`https://diligent-skirt-36b.notion.site/MongoDB-2f1db119176c4be7886edfac2062d3cc?pvs=4`)
- Check the MongoDB manual (`https://docs.mongodb.com/manual/`)
- Check the MongoDB Python manual (`https://docs.mongodb.com/drivers/pymongo/`)

  *Note: adapt the links to the version of MongoDB you are using, to avoid any version problems. This applies to all the links present in this document.*

## B Setting Up

### Installing MongoDB

Instructions on how to install the MongoDB Community Server, Mongo Shell and MongoDB Compass for Windows are provided in the 'How To' page: *see Section A.*

### Create a database and import data

From now on, we assume you have started the MongoDB Server and are using the Command Line (`cmd.exe`).

In this first exercise we will import a JSON sample dataset using the `mongoimport` command. Particularly, this dataset contains data about restaurants, and a sample document looks like the following:

```
{
  "address": {
     "building": "1007",
```

```
    "coord": [-73.856077, 40.848447 ],
    "street": "Morris Park Ave",
    "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
    { "date": {"$date": 1393804800000},"grade": "A", "score": 2},
    { "date": {"$date": 1378857600000},"grade": "A", "score": 6},
    { "date": {"$date": 1358985600000},"grade": "A", "score": 10},
    { "date": {"$date": 1322006400000},"grade": "A", "score": 9},
    { "date": {"$date": 1299715200000},"grade": "B", "score": 14}
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```


In order to populate the `test` database, you have to download the dataset and execute the `mongoimport` to insert the documents into the restaurants collection in the test database. If the collection already exists in the test database, the operation will drop the restaurants collection first.

```
https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset
    /primer-dataset.json
mongoimport --db test --collection restaurants --drop --file primer-
    dataset.json
```

The `mongoimport` connects to a `mongod` instance running on localhost on port number 27017. To import data into a `mongod` instance running on a different host or port, specify the hostname or port by including the `--host` and the `--port` options in your `mongoimport` command.

Use the mongo shell. Now, connect to the `mongod` instance using the `mongo` client by running the `mongo.exe` executable.

If you imported the data successfully, the `show dbs` command will show the newly created `test` database. Connect to the database with the command `use test`. After making the connection, you should be able to see the restaurants collection via the command `show collections`, and you should be able to explore it via `db.restaurants.find()`, which returns a cursor to the documents in the collection.

# C Training

- The MongoDB manual is the best reference for syntax and operation details: `https://docs.mongodb.org/manual/crud`.
- Some queries may require subqueries (not available in MongoDB), however you can store intermediate results using variables. Note that

the `find` method returns a cursor, and thus to use a variable in a sub-result you should use the `findOne` method which directly returns the first document of the result.

- To perform aggregations you may use the Aggregation Framework or MapReduce. However, using MapReduce on top of MongoDB is not advisable since it results in a significant performance degradation. Nevertheless, it is the only pure parallel approach, as the Aggregation Framework does pipelining. Read more in `https://docs.mongodb.org/manual/core/aggregation-introduction/`.
- Geospatial queries require indexes, read more about geospatial indexes in `https://docs.mongodb.org/manual/administration/indexes-geo/`.
- If you get many results you can use the `findOne` method to improve readability.

### C.1 Insert, Update and Delete operations:

**Q1:** Insert a document into the collection `restaurants` with the following information (with an empty list of grades):

- Name: La Llimona
- Address: Carrer de Constança, 8
- Zip code: 08029
- Coordinates: 2.136236, 41.387950
- Borough: Eixample
- Cuisine: Catalan

**Q2:** How is the document you just created uniquely identified?

**Q3:** Update the following information of the document you just inserted:

- Address: Carrer de Constança, 6
- Borough: Les Corts
- Zip code: 08014

**Q4:** Remove the document you just inserted.

**Q5:** Update the "name" field of the restaurant with the name "Gan Asia" to "Wild Asia".

**Q6:** For the restaurant you just updated, add a new grade to the "grades" array with a date of "2024-03-15", grade of "A", and score of 8.

**Q7:** Again for the restaurant you just updated, remove all the grades that are less than 8.

### C.2 Queries:

**Q1:** List all restaurants whose borough is "Manhattan".

**Q2:** List all restaurants with at least one grade with value C.

**Q3:** Retrieve the name and address of the restaurants that have a grade with a score less than 5.

**Q4:** Get the number of restaurants in "Manhattan" with some grade scored greater than 10.

**Q5:** Get the average score for each cuisine and sort them by the average score in descending order.

**Q6:** Get the number of distinct cuisine types for each borough.

**Q7:** Find the top three cuisines with the highest average scores, along with the count of restaurants for each cuisine.

### C.3 Geospatial queries

In this exercise, you are asked to import the dataset for Barcelona from OpenStreetMaps. There is already a package in JSON format that you can download from `https://s3.amazonaws.com/metro-extracts.mapzen.com/barcelona_spain.imposm-geojson.zip`. It contains several files for places like administrative regions, buildings, roads, etc.

The data is represented in the GeoJSON document format. A GeoJSON document contains a field with the geometric type of the document (e.g. point, line, polygon) and its coordinates. MongoDB has native support for GeoJSON documents and can perform geospatial queries on these documents using spatial indexes. To read more about geospatial queries in MongoDB and the GeoJSON document format, go to `https://docs.mongodb.com/manual/geospatial-queries/`.

The first task you must do is to load the following files into MongoDB:

```
barcelona_spain_admin.geojson
barcelona_spain_amenities.geojson
barcelona_spain_buildings.geojson
barcelona_spain_roads.geojson
```

Every file is organized as one JSON document with embedded documents representing the geographic entities (e.g. administrative regions, buildings, roads, etc). Before loading the files, you must convert each file to an array of JSON documents by deleting the following lines from the beginning:

```
1 {
2   "type": "FeatureCollection",
```

```
3  "crs": {"type": "name", "properties": {"name": "urn:ogc:def:crs:
       OGC:1.3:CRS84" }},
4
5  "features":
```

As well as the last closing bracket `}` from the end of the file. You must also use the option `--jsonArray` when importing the files with the command `mongoimport`. MongoDB will then automatically import each document in the JSON array as an independent document.

After loading the documents, we ask you to implement the following geospatial queries.

**Q1:** List the amenities (buildings/facilities) close (800 meters or less) to the "Barcelona Supercomuputing Center".

**Q2:** For each administrative region, list the number of buildings that are within its coordinates.

**Q3:** List the roads that span along 5 or more administrative regions.

**Q4:** Provide a ranking (sorted list) of most common amenity types in "Badalona"