

Group Project Part 2 - MongoDB
of Systems and Methods for Big and Unstructured Data Course
(SMBUD)
held by
Brambilla Marco
Tocchetti Andrea

Group 14

Banfi Federico
10581441

Carotenuto Alessandro
10803080

Donati Riccardo
10669618

Mornatta Davide
10657647

Zancani Lea
10608972

Academic year 2021/2022



POLITECNICO
MILANO 1863

Contents

1	Problem Specification	3
2	Hypotheses	3
3	ER diagram	5
4	Dataset description	6
5	Queries and Commands	7
5.1	Queries	7
5.1.1	Find all the people with the Reinforced GreenPass . . .	7
5.1.2	Find the number of vaccinations ordered per month . .	9
5.1.3	Find the age average of unvaccinated people	10
5.1.4	Find all currently infected people	10
5.1.5	Find the number of tests per authorized body (JOIN)	12
5.2	Commands	14
5.2.1	Insert a new Authorized Body (CREATE)	14
5.2.2	Insert a test in a Certification (UPDATE)	15
5.2.3	Delete all the Authorized Bodies that test and are pharmacies (DELETE)	16
6	UI description & User Guide	18
6.1	UI description	18
6.2	User Guide	19
7	References & Sources	20

1 Problem Specification

During the sanitary emergency due to the Covid-19 pandemic, many programs and applications developed thanks to the use of Big Data proved to be particularly effective in different settings and scenarios, such as the hospital administration, the hospitalizations organization or the analysis of data relating to various clinical cases. Among the various areas that have been supported by these technologies there is also that which concerns the tracking of populations belonging to a given geographical area and the collection of all information regarding the tests carried out and the vaccination status.

Our project aims to create an information system suitable for this specific use case and to do this it is necessary to design a database that allows us to store large amounts of data derived from heterogeneous sources and to carry out targeted queries useful for different purposes. The NOSQL document-based approach, known for being based on managing data saved in JSON-like documents, is the optimal one in this case and MongoDB is the open source database management software that is best suited to accomplish this task thanks to the considerable scalability guaranteed by the automatic data sharding and ease of use thanks to the dynamic schemes developed starting from the archived documents.

2 Hypotheses

The way in which the database was structured and implemented is based on some hypotheses discussed in the design phase.

Considering a typical scenario in which each person can interact either with people who live with him (family members or roommates) or with other people who go to meet voluntarily or not in a limited closed place (for example a gym, a public place, a restaurant, etc.) or in an open place located in a generic position, it was assumed that the infection can occur if during the contact between the two subjects (one of which is potentially infected) the distance between them was particularly close and if the contact duration was at least 15 minutes. In addition, to check the state of health of each individual in relation to the pandemic situation, it is essential to consider whether they have been given a vaccine and/or whether they have recently undergone a test to check for any positivity.

Specifically, in order to simulate the pandemic situation in the most realistic way possible and to optimally manage the available data that an information system directly connected to an app supported by the database will have to process, the following assumptions were taken into consideration:

1. people can go to public places even without a green pass (therefore without a vaccine or a negative test carried out in the last 48 hours),
2. data relating to public places are sent to the system by the manager of the public place itself,
3. to record a greater number of relationships to be processed, while ensuring the consistency and meaningfulness of the stored data, it was decided to limit to an example study case in which the interactions present in the dataset are related to the single city of Milan and for a limited time interval, therefore:
 - the tests dates, the meetings between people and the visits to public places took place from the 15th of October to the 15th of November, in addition, to test the queries more easily, there is a specific date, i.e. the 31st of October, that expressly presents multiple recorded interactions,
 - vaccine administrations were all performed within the last year,
 - all public places are located in Milan,
 - meetings between people are geolocated within the city limits
4. with regard to the various connections that can exist between several people and the infections that can occur within the same family, it was preferred, for design purposes, to consider the concept of "same residential unit" (i.e. residence) rather than that of "family", since belonging to a given family unit does not necessarily imply a constant coexistence (e.g. workers who often travel for a job or off-site students) while a relationship of coexistence between several people, not necessarily related to each other, usually involves a close and inevitable contact,
5. concerning to meetings between two or more people, it was assumed that any contact tracing devices/ apps interact with the system, communicating this information for each meeting: person1, person2, time at which the contact took place, geographical point in where the contact took place,
6. a person is considered infected from the moment he undergoes a test and receives a positive result until he takes another test again but it is negative,
7. in the event that a person undergoes a test and receives a positive result, all people met in the 15 days prior to the test will receive an alert notification on the app ensuring the privacy of the infected person is safeguarded,
8. the possibility has also been envisaged in which a person already vaccinated may still be infected, in line with the cases that actually occurred

3 ER diagram

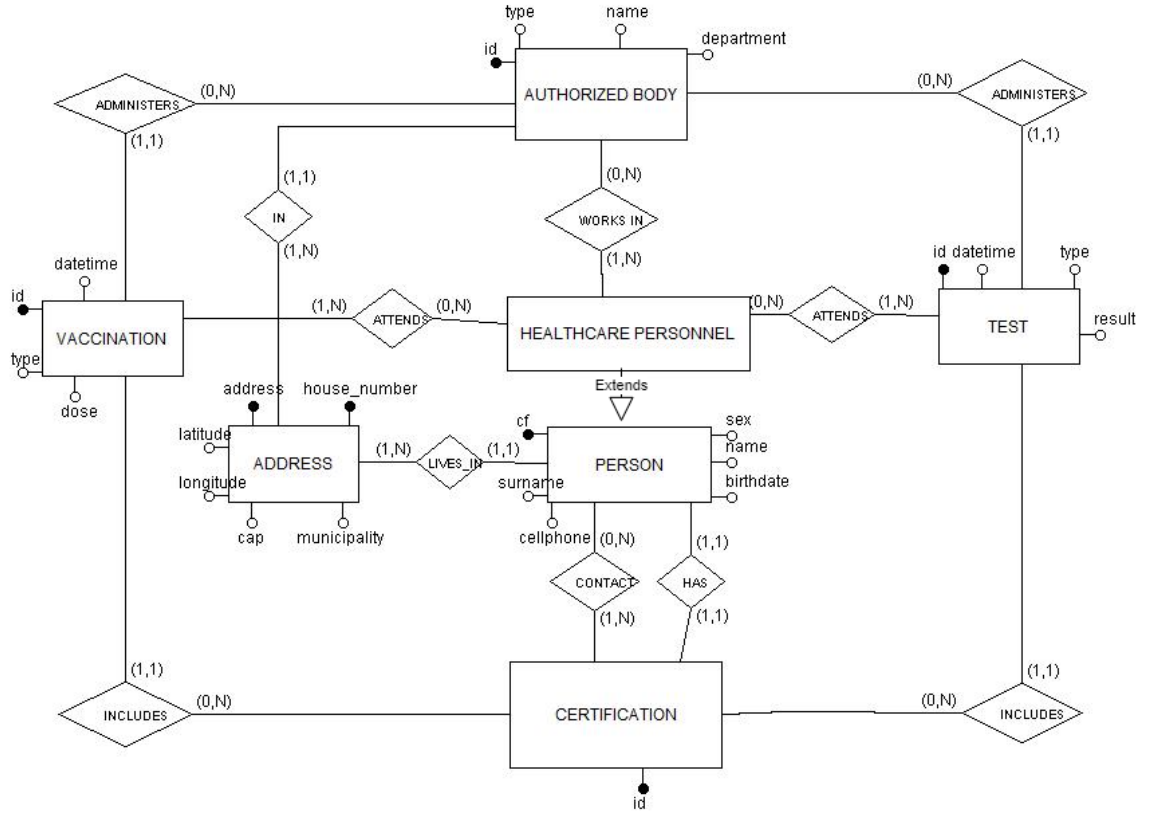


Figure 1: E-R Diagram

Starting from the considerations previously exposed regarding the implementation hypotheses, we have drawn an ER diagram (**Figure 1**) which includes 7 different entities and 4 many-to-many relationships described below in the logical model:

- **Address**(Address, HouseNumber, CAP, Latitude, Longitude, Municipality)
- **AuthorizedBody**(ID, Department, Name, Type)
- **Certification**(ID)
- **HealthcarePersonnel**(CF, Birthdate, Cellphone, Name, Sex, Surname)
- **Person**(CF, Birthdate, Cellphone, Name, Sex, Surname)
- **Test**(ID, Datetime, Result, Type)

- **Vaccination**(ID, Datetime, Dose, Type)
- **AttendsTest**(HealthcarePersonnel.CF, Test.ID)
- **AttendsVaccination**(HealthcarePersonnel.CF, Vaccition.ID)
- **Contact**(Certification.ID, Person.CF)
- **WorksIn**(AuthorizedBody.ID, HealthcarePersonnel.CF)

The **Person** entity describes each individual and their personal data, **HealthcarePersonnel** is a specialization of **Person** and includes all the people (doctors, nurses, pharmacists, etc.) who work (as indicated by the many-to-many **WorksIn** relationship) at an **AuthorizedBody**. Both the **Test** and **Vaccination** entities are linked to **HealthcarePersonnel** by the many-to-many **AttendsTest** and **AttendsVaccination** relationships, while the link with **AuthorizedBody** is given by the one-to-many **AdministersTest** and **AdministersVaccination**. Each **Person** can have one and only one (one-to-one relationship **Has**) **Certification**, that contains information regarding how many and which tests a person has undergone and whether or not he has received one or more doses of the vaccine, in fact, **Certification** is in a one-to-many relationship with both **Test** and **Vaccination**, through **IncludesTest** and **IncludesVaccination**, respectively. Furthermore, each **Certification** must also include for each **Person** a sort of "emergency contact" with its details as evidenced by the many-to-many **Contact** relationship. Finally, to save information relating to the geographical location of persons and authorized bodies, an **Address** entity was added with specific attributes, connected to **Person** and **AuthorizedBody** respectively by the one-to-many **LivesIn** and **In** relationships.

4 Dataset description

One of the most critical parts of working with Big Data is managing large amounts of data collected in large datasets. To test and simulate the use of the database for contact tracing activities, some sample datasets were generated, saved in .csv format and imported into Neo4j through the command: `LOAD CSV FROM "file: ///file.csv" AS`

Each dataset is divided into various fields that trace the structure of the tables expressed in the ER model, each one was generated randomly through Python scripts (you can find these ones into the folder "db") and to experiment and perform at best the possible tests on the queries and commands that can be executed thanks to Neo4j the number of entries foreseen for each dataset is in the order of magnitude of the hundreds, as a whole, starting exclusively from the data loaded from the datasets, the database provides:

Homes	101	Residence Relationship	300
People	300	Meetings	726
Public Places	20	Public Places Visits	600
Vaccine Types	4	Vaccinations	526
Test Types	2	Tests	450
TOTAL NODES	427	TOTAL RELATIONSHIPS	2602

5 Queries and Commands

The correct functioning of the information system involves the implementation of some essential commands and queries for the database in order to properly support the app and to ensure the right execution of searches among the data available for statistical or practical purposes.

First of all you need to load the .csv files in two separate collections (authorizedBodies and certifications), you can find them following the path "db/ab.csv" and "db/certification.csv"

When you are importing the data make sure to change the datatypes in:

- All the dates/datetime from String to Date
- In certifications test/vaccine.id_authorized_body from String to ObjectId
- In authorizedBodies _id from String to ObjectId

5.1 Queries

5.1.1 Find all the people with the Reinforced GreenPass

This query allows us to find all people in possession of a reinforced green-pass after the latest regulations, that is, all those who have been vaccinated in the last 9 months.

QUERY

```
db.certifications.find({
  "$and": [{"vaccination":{"$exists":true}},
  {"vaccination.datetime":{"$gte":new Date(
    ISODate().getTime() - 1000 * 3600 * 24 * 270)}}},
  {"vaccination.datetime":{"$lte":new Date(
    ISODate().getTime())}}}]
},{
  "person.name":1,
  "person.surname":1,
  "vaccination":1
})
```

The output is given by:

- The ObjectId of the certification document
- The name and the surname of the person
- The datetime of the vaccination

OUTPUT

```
{ _id: ObjectId("61af82fb2468d4592b37b6ee"),
  person: { name: 'Luca', surname: 'Colombo' },
  vaccination:
    [ { datetime: 2021-09-07T17:31:42.000Z },
      { datetime: 2021-10-05T17:31:42.000Z } ] }
{ _id: ObjectId("61af82fb2468d4592b37b70e"),
  person: { name: 'Lorenzo Federico Giuseppe',
            surname: 'Tomasi' },
  vaccination: [ { datetime: 2021-11-14T10:18:51.000Z } ] }
{ _id: ObjectId("61af82fb2468d4592b37b6e9"),
  person: { name: 'Vincenzo', surname: 'Maggiani' },
  vaccination:
    [ { datetime: 2021-10-23T15:51:37.000Z },
      { datetime: 2021-11-20T16:51:37.000Z } ] }
. . .
```


5.1.2 Find the number of vaccinations ordered per month

This query allows us to retrieve some statistical data on the number of the vaccinations per month.

QUERY

```
db.certifications.aggregate(  
  { $unwind : "$vaccination" },  
  { $group: {  
    _id: {month: { $month: "$vaccination.datetime" },  
    year: { $year: "$vaccination.datetime" } },  
    count: { $sum: 1 }  
  }  
}, {  
  "$sort": {count: -1}  
}  
)
```

The output is given by:

- The count of the vaccinations
- The months and year

OUTPUT

```
{ _id: { month: 11, year: 2021 }, count: 41 }  
{ _id: { month: 2, year: 2021 }, count: 36 }  
{ _id: { month: 10, year: 2021 }, count: 36 }  
{ _id: { month: 8, year: 2021 }, count: 36 }  
{ _id: { month: 4, year: 2021 }, count: 35 }  
{ _id: { month: 3, year: 2021 }, count: 32 }  
{ _id: { month: 12, year: 2021 }, count: 29 }  
{ _id: { month: 5, year: 2021 }, count: 27 }  
{ _id: { month: 7, year: 2021 }, count: 27 }  
{ _id: { month: 9, year: 2021 }, count: 26 }  
{ _id: { month: 1, year: 2021 }, count: 22 }  
{ _id: { month: 6, year: 2021 }, count: 14 }
```

5.1.3 Find the age average of unvaccinated people

This query also allows us to get a statistical information about the age of the unvaccinated people

QUERY

```
db.certifications.aggregate([
  { $match : {
    "person.birthdate" : { $exists : true},
    "vaccine":{"$exists":false}
  } },
  { $project : {"ageInMillis" : {$subtract :
    [new Date(), "$person.birthdate"] } } },
  { $project : {"age" : {$divide :
    ["$ageInMillis", 31558464000] } }},
  // take the floor of the previous number:
  { $project : {"age" : {$subtract : ["$age",
    {$mod : ["$age",1]]}}},
  { $group : { _id : true, avgAge : { $avg : "$age" } } },
  { $project: { "avgAge": { $round: ["$avgAge", 2] } } }
])
```

The output is given by:

- The age average

OUTPUT

```
{ _id: true, avgAge: 58.14 }
```

5.1.4 Find all currently infected people

This query allows us to find people currently infected, i.e. those whose last test is positive.

QUERY

```
db.certifications.aggregate([
  {$addFields : {test : {$reduce : {
    input : "$test",
    initialValue : {datetime :
      new ISODate('2000-01-01T00:00:00')},
    in : {$cond: [{gte : ["$$this.datetime",
      "$$value.datetime"]}, "$$this", "$$value"]}}}
  }},
  { $match: {"test.result": "Positive"}},
  {$project: {
    "person.name": "$person.name",
    "person.surname": "$person.surname",
    "person.codice_fiscale": "$person.codice_fiscale",
  }}
])
```

The output is given by:

- The ObjectId of the certification document
- The name and surname of the person
- The CF of the person

OUTPUT

```
{ _id: ObjectId("61af82fb2468d4592b37b71e"),  
  person:  
    { name: 'Adele',  
      surname: 'Mantovani',  
      codice_fiscale: 'MNTDLA62D60C059I' } }  
{ _id: ObjectId("61af82fb2468d4592b37b71b"),  
  person:  
    { name: 'Ferruccio',  
      surname: 'Meloni',  
      codice_fiscale: 'MLNFRC61D17B246A' } }  
{ _id: ObjectId("61af82fb2468d4592b37b7ab"),  
  person:  
    { name: 'Riccardo',  
      surname: 'Brumat',  
      codice_fiscale: 'BRMRCR65T22F356R' } }
```

5.1.5 Find the number of tests per authorized body (JOIN)

This query allows us to find the number of tests that each authorized body did, in this query we use the JOIN of the two document in order to retrieve the name and the type of the authorized body starting from the id in the certification document.

MongoDB is not optimized to perform this type of operation like relational databases, so the query is optimized to perform the join on as few elements as possible grouping the tests in pipeline before the join.

QUERY

```
db.certifications.aggregate([
  { $unwind : "$test" },
  {$group:{
    _id:"$test.id_authorized_body",
    count:{$sum:1}
  }},
  {
    "$lookup": {
      "from": "authorizedBodies",
      "localField": "_id",
      "foreignField": "_id",
      "as": "authorizedBodyInfo"
    }
  },
  {
    "$project":{_id:0,count:1,
      "authorizedBodyInfo.name":"$authorizedBodyInfo.name",
      "authorizedBodyInfo.type":"$authorizedBodyInfo.type"
    }
  }
])
```

The output is given by:

- The count of the tests done
- The name of the authorized body
- The type of the authorized body

OUTPUT

```
{ count: 17,
  authorizedBodyInfo: [ { name: [ 'Policlinico' ],
                        type: [ 'Hospital' ] } ] }
{ count: 21,
  authorizedBodyInfo: [ { name: [ 'Auxologico San Luca' ],
                        type: [ 'Hospital' ] } ] }
{ count: 25,
  authorizedBodyInfo: [ { name: [ 'San Raffaele' ],
                        type: [ 'Hospital' ] } ] }
{ count: 19,
  authorizedBodyInfo: [ { name: [ 'San Paolo' ],
                        type: [ 'Hospital' ] } ] }
{ count: 26,
  authorizedBodyInfo: [ { name: [ 'San Giuseppe' ],
                        type: [ 'Hospital' ] } ] }
{ count: 15,
  authorizedBodyInfo: [ { name: [ 'Ospedale Fiera' ],
                        type: [ 'Hospital' ] } ] }
. . .
```

5.2 Commands

5.2.1 Insert a new Authorized Body (CREATE)

In this command we insert in the `authorizedBodies` collection a new document.

COMMAND

```
db.authorizedBodies.insertOne({
  "address":{
    "address":"Via Roma",
    "cap":"24030",
    "geopoint":"45.0000 9.0000",
    "house":"13b",
    "municipality":"Bergamo",
  },
  "department":"covid test",
  "name":"Farmacia Comunale",
  "type":"pharmacy",
  "healthcarePersonnel":[
    {
      "cellphone":"+393476574382",
      "codice_fiscale":"DNTHGB87P15G764J",
      "name":"Carlo",
      "role":"pharmacist",
      "surname":"Merlutti",
    }
  ]
})
```

5.2.2 Insert a test in a Certification (UPDATE)

In this command we add to a given certification (via ObjectId) a new test as a subdocument in the ArrayList field test.

COMMAND

```
db.certifications.updateOne({
  "_id":ObjectId('61af82fb2468d4592b37b70e')
},{
  "$push":{
    "test":{
      "$each":[{
        "datetime":ISODate('2021-12-08T16:20:00'),
        "healthcarePersonnel":{
          "codice_fiscale":"PLCLRT48C02C224Y",
          "name":"Alberto Christian",
          "surname":"Paolucci"
        },
        "id_authorized_body":
          ObjectId('61acd3e25ca7e964a122def1'),
        "result":"Positive",
        "type":"Molecular"
      }]
    }
  }
})
```

5.2.3 Delete all the Authorized Bodies that test and are pharmacies (DELETE)

In this command we simulate a change of regulations where the pharmacies can no longer test people and therefore we delete this kind of authorized body from the collection.

COMMAND

```
db.authorizedBodies.deleteMany({
  "$and":[
    {"type":"pharmacy"},
    {"department":"covid test"}]
})
```


6 UI description & User Guide

6.1 UI description

The design of the information system ended with the development of *ContagionShield*, an application connected to the database with a simple GUI that is very intuitive and easy to use. The UI was created with the Python programming language by means of the libraries: *PySimpleGUI* for the creation of the graphical interface, *Py2neo* to work with Neo4j through the syntax offered by Python and *pandas* to manage the analysis and manipulation of data, you can find these ones into the folder "contagionshield".

As you can see from **Figure 3**, the main screen of the application is divided into two sections: one on the left that allows you to create customizable queries by choosing date, place, time interval and whether to display the vaccinated, non-vaccinated or with negative test, in **Figure 4** there is an example query with the results obtained; the other side section on the right (**Figure 5**) presents some predefined queries, some of them follow the ones specified and described in Chapter 5 of the report.

Finally, as shown in **Figure 6**, it is possible to execute some of the commands already presented in Chapter 5 by reaching the appropriate section of the application by means of the "Commands" button in the lower left corner of the main screen. The executable commands allow you to create new meetings between several people by indicating their social security numbers, the date and time, create new visits to public places by specifying who went to a given place and when and in the end you can also delete all the visits to public places recorded in the last year.

6.2 User Guide

In order to run *ContagionShield* it is necessary to verify some requirements and to perform some actions:

- At first you have to check if you have the right Python version installed by using the command: `python --version`, if not you could download it from the official website: <https://www.python.org/>
- Then make sure your local database is at `localhost:7687` and that it has "smbud" as password
- Install the required packages (in the "contagionshield" folder): `pip install -r requirements.txt`
- Finally make it run by navigating to the folder where you have been saved the materials, then into "contagionshield" folder and run: `python contagionshield.py`
- From there you can execute queries about the collected data

7 References & Sources

- [1] Course Slides
- [2] <https://pysimplegui.readthedocs.io/en/latest/call>
- [3] <https://py2neo.org/>
- [4] <https://neo4j.com/docs/cypher-manual/current/>
- [5] <https://neo4j.com/developer/python/>
- [6] <http://iniball.altervista.org/Software/ProgER>
- [7] <https://neo4j.com/developer/cypher/>
- [8] <https://pandas.pydata.org/docs/>