SMBUD - PROJECT WORK

Delivery #2

Academic year 2021/2022
Professor: Brambilla Marco

Group number: 4

Group members:
Elisa Mariani - 10632876
Pietro Guglielmo Moroni - 10625198
Giacomo Palù - 10682209
Francesco Panebianco - 10632465
Silvio Scanzi - 10622285

Politecnico di Milano

TABLE OF CONTENTS

1. Introduction 3
   1.1. Specifics
   1.2. Hypotheses and implementation choices
2. Conceptual model 4
   2.1. Structure of the documents
   2.2. ER model
3. Creation of the database 7
4. Queries 8
5. Commands 12
6. Application 15

1.Introduction

1.1. Specifics

The database records data related to the COVID19 pandemic, data that could be used to support a certification app. It includes two types of documents: personal certificates with information on vaccines/tests and fact sheets for each body authorized to perform vaccines/tests.
As regards the first type of document, the personal certificate, it contains all the information of a person relevant from a sanitary point of view: his personal data (with a unique fiscal code), his emergency contacts, all tests and vaccines he/she has done. In particular, for both vaccinations and tests several pieces of information are taken into account: the date, the institution where it was carried out, the name and ID of the health professionals who performed it. Then, for the tests, the result is also saved and for vaccinations, other details are also recorded: the dose number if there were complications, the brand, type, lot, vial number, and date of production of the vaccine.
On the other hand, for the second type of document concerning authorized entities, the name of the institution, its ID, its type, its location (city, address, latitude, longitude), the department in charge of vaccines/tests and its supervisor are registered.

1.2. Hypotheses and implementation choices

Several assumptions and choices have been made for the implementation of the database:

I. As long as vaccines and tests are concerned, the generated data cover the years 2020, 2021, 2022.
II. For vaccinations and tests, even though they are saved in the ISODate format, the time will be set to 00:00:00.000 because for DB interrogation purposes it was sufficient to know only the date.
III. A certificate is considered valid only if the person got vaccinated no more than 9 months(270 days) ago or took a test no more than 2 days ago.
IV. For the tests is registered only one operator for each, instead for vaccinations even more than one
V. The "****_id", "fiscal_code", "phone_number" fields (visible in the detailed diagram and in paragraph 2.1) are considered unique for the entity to which they refer to.
VI. If a patient has presented side effects (reported by the patient himself or by the treating doctor) in 48h after the vaccine the certificate must take it into account.

2. Conceptual model

2.1. Structure of the documents

Certificate document structure:

- _id
- registry
  - first_name
  - last_name
  - phone_number
  - birthday
  - gender
  - fiscal_code
- emergency_contacts []
  - first_name
  - last_name
  - phone_number
- vaccination_list []
  - vaccine
    - brand
    - type
    - lot
    - vial_id
    - production_date
  - date
  - side_effects
  - dose_number
  - institution_id
  - operators[]
    - first_name
    - last_name
    - operator_id
- tests[]
  - date
  - result
  - operator
    - first_name
    - last_name
    - operator_id
  - institution_id

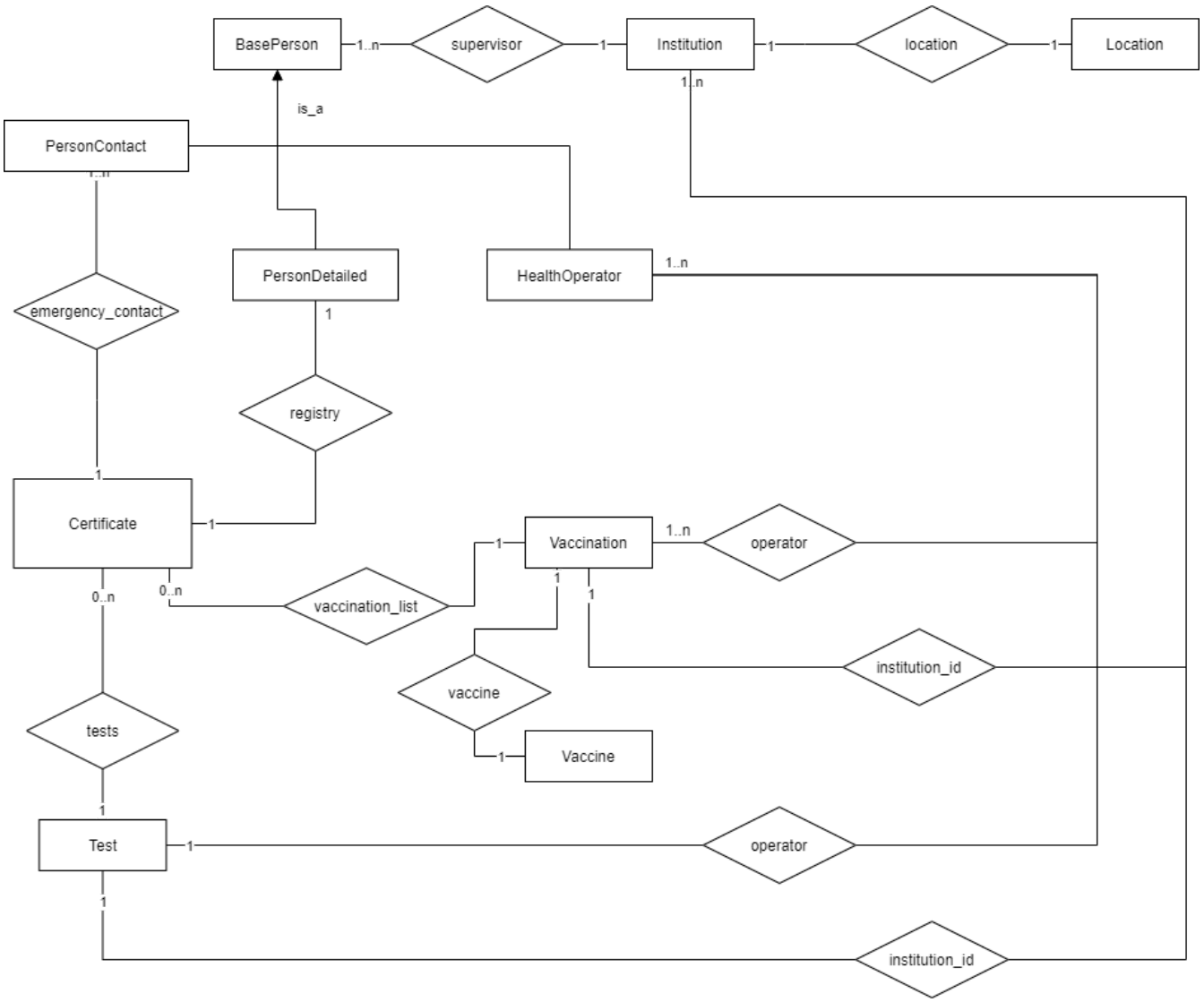
'*vaccination_list*','*emergency_contacts*', '*operators*' and '*tests*' are arrays of documents, *'registry', 'vaccine', 'operator' are embedded sub-documents.*
In particular, the field "result" is an integer: it is equal to '0' if the test is negative and '1' if the test is positive. Also the field "side_effects" is '1' if the patient had side effects after the vaccine and '0' otherwise.

Medical institutions document structure:

- name
- location
  - city
  - address
  - latitude
  - longitude
- entity_type
- supervisor
  - name
  - last_name
- vaccine_issuing_departement
- institution_id

'*location*' and '*supervisor*' are embedded sub-documents.

2.2. ER model



The above diagram displays how the different elements of the documents are related to each other. A more detailed representation of this diagram can be found in the file 'Detailed_ER_Schema'. Since the implemented DB is documental, all the documents will have an automatic 'primary key' field "_id" but in these diagrams it is not made explicit.

Since the implemented DB is a non relational one, the entities and the relationships between them don't have a direct correlation with the actual implementation of the documents. To simplify the mapping between the document format and the structure of the ER, relationships are named as the fields they will be turned on upon "translation" to the document format (e.g. Certificate will have a field called *emergency_contact* containing the subdocument with emergency contact details, while *tests* is a field containing a list of Test subdocuments). *institution_id* for Test and Vaccination is the exception, as it is translated as a reference to the *institution_id* contained in Institution.

In the DB, we want to register the data of multiple types of 'person' (health operators, emergency contact and the person to whom the certificate belongs): this concept is modeled in the diagram through a *is_a* hierarchy: the three types of 'person' cover *'BasePerson'*. A single certificate is related in different ways to these different people: there is a 'OneToOne' relationship between *'PersonDetailed'* and *'Certificate'* (in particular, this relationship is called *'registry'* meaning the personal data of the person) and a 'OneToMany' relationship between *'EmergencyContact'* and *'Certificate'*. As regards to the *'HealthOperators'* instead, they are related to *'Vaccination'* and *'Test'* in two different ways, respectively: for each vaccination multiple operators can be saved and for the tests only one each. Also for all the other relationships, the 'cardinalities' are reported: they show how many times a field is contained in a document (or embedded document).

3. Creation of the database

To populate the database, a similar approach to the previous delivery was chosen: entities and relationships from the ER model were modeled in *Python*, along with procedures to randomly generate them. Considering that the data format in this case is a well established standard (JSON), *Python* libraries could easily deal with the object to DB document representation. The idea in this case was to encapsulate the data types that had to be converted into documents into a base `DBDataType` that can be found in `utility.py`. All classes extending this base will inherit the `toStringDict()` method, which returns the JSON string corresponding to the object.

Modeling the dataset we had to make some decisions. Some of these were aimed at making the model flexible in case new laws were to introduce changes (e.g. we made the maximum number of vaccine doses per person a parameter), others at keeping our efforts and resources contained while still remaining in scope with the assignment (e.g. we decided not to track the healthcare personnel - `HealthServiceOperator`s in the codebase - among the population since it was not relevant for us to do so within the scope of the project).

The `main.py` script coordinates data generation in three steps:

1) Population is generated given `peopleCount`, referring to the size of the population, and `contactCount`, referring to the number of generated emergency contacts for vaccinated citizens).

2) Vaccination and testing hubs are generated given only a `hubCount` (number of hubs to be generated).

3) Certificates for the population are generated by passing the , the, and two self-explanatory parameters (`maxVaccineDosesPerPerson` and `maxTestsPerPerson`).

Finally, the method `collectionToFile` is called from the `DBJsonWriter` class in `utility.py` to save both the documents describing *hubs* and *certificates* in two separate files: respectively `medical_institutions_collection.json` and `certificates_collection.json`

4. Queries

Query #1. Verify the validity of the certificate of Raffaelino Boncompagni (fiscal code: BNCRFL33S23U169D). If it is valid return “VALID CERTIFICATE”, otherwise return nothing.

The query:

```
db.Certificates_data.find(
 {
  $and:[{"registry.fiscal_code":"RMTZTX01C59Q935D"},
      {$or:[
          {"vaccination_list":{$elemMatch:{date:{$gte:new Date(ISODate().getTime()-1000*3600*24*270)}}}},
          {"tests":{$elemMatch:{$and: [{date:{$gte:new Date(ISODate().getTime()-1000*3600*24*2)}},
                                       {date:{$lte:new Date(ISODate().getTime())}},
                                       {result:0}]
                                  }
                    }
          }
      ]}
  ]
 },{"RESULT:":"VALID CERTIFICATE"})
```

The outcome:

```
{ _id: ObjectId("61a12b02af7643168eb5f95d"),
  'RESULT:': 'CERTIFICATE IS VALID' }
```

In the query we want to find the certificate that is both valid and belongs to the person with the fiscal code BNCRFL33S23U169D, so we put in ‘AND’ these two conditions.
The validity constraint is created with an operator 'or': the person must have been vaccinated no more than 9 months ago or must have taken a negative swab no more than two days ago.
To compute the time intervals of 9 months and 2 days we multiplied the milliseconds needed to form the time period wanted. In particular, for the 9 months period we approximated 9 months with 270 days, that is the actual current validity period of the certificate.

Query #2. Extract the number of doses of vaccine given in every type of institution.

The query:

```
db.Certificates_data.aggregate([
 {
  $project:{
   "institutions_id":"$vaccination_list.institution_id"
  }
 },
 {
  $unwind:"$institutions_id"
 },
 {
  $lookup:{
   from: "Institutions_data",
   localField: "institutions_id",
   foreignField: "institution_id",
   as: "institution"
  }
 },
```

```
 {
  $unwind: "$institution"
 },
 {
  $group:{
   "_id":"$institution.entity_type",
   "count":{$sum:1}
  }
 },
 {
  $sort:{"count":-1}
 }
 ])
```

The outcome:

```
{ _id: 'Stadio', count: 204 }
{ _id: 'Complesso', count: 162 }
{ _id: 'Presidio Ospedaliero', count: 150 }
{ _id: 'Clinica', count: 124 }
{ _id: 'Istituto', count: 118 }
{ _id: 'Ospedale', count: 81 }
{ _id: 'Spazio', count: 71 }
```

In this query, first we project only the id of the institutions where a person got the doses of the vaccine, then we expand the array using 'unwind'. We then join every entry with the institution collection, via the 'institution_id' field, creating the field 'institution', an array containing only one element. We use 'unwind' on this field. We finally group by the type of institution, counting the vaccines that were made and sorting the result in descending order.

Query #3. Extract the mean age of the people that had a Pfizer vaccine (at least 1 dose) and reported side_effects. Extract also the age of the youngest person among those.
N.B. the age of people is computed with the birth date and years are considered to be of 365 days (leap years are not considered).

The query:

```
db.Certificates_data.aggregate([
   {$match:{
    "vaccination_list":{$elemMatch:{"$and": [{side_effects:1},{"vaccine.brand":"Pfizer"}]}}
   }},
   {$addFields:{
    "age":{$divide:[{$subtract:[ISODate(),"$registry.birthday"]},(365*24*60*60*1000)]}
   }},
   {$group:{
    "_id":true,
    "age_average_sideEffects_Pfizer":{"$avg": "$age" },
    "age_youngest":{$min: "$age"}
   }}
])
```

The outcome:

```
{ _id: true,
  age_average_sideEffects_Pfizer: 63.88516922020361,
  age_youngest: 1.1382929829718418 }
```

In this query we want first to extract all the certificates of people that have been vaccinated with 'Pfizer' vaccine and that had side effects. Then we add the field 'age', computed as the difference between the current date and the birth date, to each one of the extracted certificates. Lastly, we compute the average and the minimum age using the appropriate operators, respectively 'avg' and 'min'.

Query #4. Extract the percentage of people that got at least one positive test after a dose of vaccine.

The query:

```
db.Certificates_data.aggregate([
 {
  $group:{
   "_id":null,
   "total":{$sum: 1},
   "certificates":{$push:{tests:"$tests",
                          vaccination_list:"$vaccination_list",
                          registry:"$registry"}}
  }
 },
 {
  $unwind: "$certificates"
 },
 {
  $unwind: "$certificates.tests"
 },
 {
  $unwind: "$certificates.vaccination_list"
 },
 {
  $match:{"certificates.tests.result":1}
 },
 {
  $match: {$expr:{"$gte":["$certificates.tests.date", "$certificates.vaccination_list.date"]}}
 },
 {
  $group:{
   "_id":"$certificates.registry.fiscal_code",
   "total":{$first:"$total"}
  }
 },
 {
  $group:{
   "_id":null,
   "infected":{$sum:1},
   "total":{$first:"$total"}
  }
 },
 {
  $project:{
   "_id":0, "infected":1, "percentage":{$multiply:[{$divide:["$infected", "$total"]}, 100]}
  }
 }])
```

The outcome:

```
{infected:  260, percentage: 43.333333333333336}
```

In this query, first we count the number of certificates present in the database with the 'group' operation, keeping the relevant data with the 'push'. Then we have three 'unwind' operations that basically compose all the possible 'test-vaccination' pairs for each person. We filter, using 'match', the pairs that contain a positive test and in which the test was performed after the vaccination.
Then we group by the fiscal code of the person, for counting the person only once (we keep also the number of certificates calculated before). We count the remaining documents, getting in this way the number of people that got infected after the vaccination. Finally we project the result, displaying the percentage.

Query #5. Extract the top 5 institutions with the highest number of vaccinations made in the last year.

The query:

```
db.Certificates_data.aggregate([
 {$unwind: "$vaccination_list"},
 {$match:{
  $and:[{$expr:{$gt:["$vaccination_list.date",{$subtract:[ISODate(),365*24*60*60*1000]}]}},
      {$expr:{$gt:[ISODate(),"$vaccination_list.date"]}}]
  }},
  {$group:{
   "_id":"$vaccination_list.institution_id",
   "countPerInstitution": {$sum: 1}
  }},
  {$sort:{"countPerInstitution":1}},
  {$limit:5}
 ])
```

The outcome:

```
{ _id: 'ComplessoFiorilloSaccone01800',  countPerInstitution: 11 }
{ _id: 'IstitutoAdoratoAlbanese14245', countPerInstitution: 11 }
{ _id: 'ClinicaCesarinoBarone57673', countPerInstitution: 11 }
{ _id: 'IstitutoGerlandaD\'emilia88182',countPerInstitution: 10 }
{ _id: 'StadioIlvanaVandenbulcke76196', countPerInstitution: 9 }
```

In this query, first we deconstruct the array field 'vaccination_list' with the 'unwind'. What we have after this are different certificates for each vaccination taken by each person. Thus, the field 'vaccination_list' now contains one vaccination per certificate. With the 'match' we select only the certificates with the vaccination date that goes back 365 days at most. We can notice that the second constraint, the one that imposes that the date of the vaccine must be earlier to the current one, is necessary because we don't want to include future dates (that in the tested DB are present).
Then we group the certificates by the institutions where the vaccines were made and we count how many certificates are present per institution.
Lastly, we sort the result in descending order and we consider only the first five results.

Query #6. Extract the top 5 cities with the highest number of institutions.

The query:

```
db.Institutions_data.aggregate([
 {
  $group:{
```

```
    "_id":"$location.city",
    "number":{$sum:1}
  }
 },
 {
  $sort:{"number":-1}
 },
 {
  $limit:5
 }
 ])
```

The outcome:

```
{ _id: 'Napoli', number: 10 }
{ _id: 'Torino', number: 9 }
{ _id: 'Como', number: 7 }
{ _id: 'Catania', number: 7 }
{ _id: 'Firenze', number: 7 }
```

In this query, first we group by the city, getting the number of institutions located there. Then we sort the cities in descending order and we consider only the first five results.

5. Commands

Command #1. Add a positive test for the person with fiscal code "BNCRFL33S23U169D". The test was done on 20/11/2021 at 15:30, by the operator "OJS47153", Adriano Campanella, at the institute "PresidioOspedialieroMauricoBaiano36484".

The command:

```
db.Certificates_data.updateOne(
    {"registry.fiscal_code":"BNCRFL33S23U169D"},
    {$push: {"tests":{date:ISODate("2021-11-20T15:30:00"),
                      result:1,
                      operator:{"first_name":"Adriano","last_name":"Campanella",
                                "operator_id":"OJS47153"},
                      institution_id:"PresidioOspedialieroMauricoBaiano36484"}}}
 )
```

The outcome:

```
{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }
```

In this command, first we select the certificate of the wanted person using his fiscal code. Then, we add a document (embedded) in the array of document 'tests' using the '$push'. The

document inserted is structured in the same way as the already present ones: the fields are all ‘simple’ attributes except the ‘operator’ field that is an embedded document.

Before the command:

After the command:

Command #2. Report that Lavina De (fiscal code: “DXXLVN95P48Y013D”) had side effects after taking the vaccine on 02/11/2021.

The command:

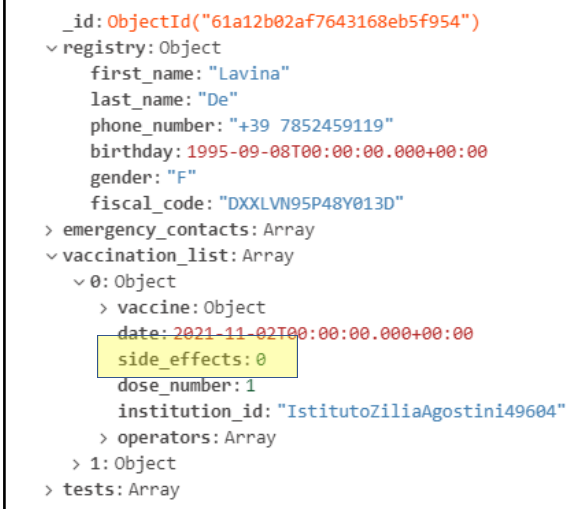
```
db.Certificates_data.updateOne(
 {"registry.fiscal_code":"DXXLVN95P48Y013D",
  "vaccination_list.date":ISODate("2021-11-02T00:00:00.000+00:00")
 },
 {$set:{"vaccination_list.$.side_effects":1}}
 )
```

The outcome:

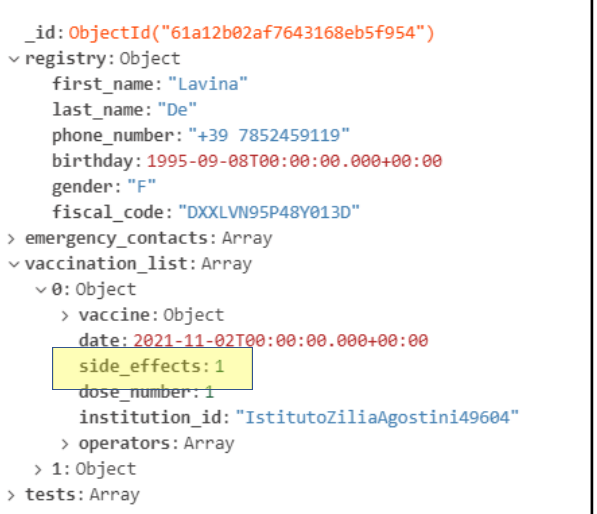
```
{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }
```

In this command, first we select the wanted person using his fiscal code. In this ‘filter’ we also select the vaccine we want to update using the date. Lastly, we set the field ‘side_effects’ to ‘1’ using the ‘$set’ operator. In particular, since we want to change a field of an embedded document that is part of an array of documents, we had to use the notation *“array.$.field”*.

Before the command:



After the command:



Command #3. Add the S.Chiara hospital in Trento as a new institution in the Institutions_data collection. It is located at the address "Largo Medaglie d'oro, 9" (46.05684702214164, 11.132842787177719), the supervisor is Mario Grattarola and the vaccines are issued in "Dipartimento Sanitario". As id, use "OspedaleSantaChiara99999".

The command:

```
db..insertOne({
  "name": "Ospedale Santa Chiara",
  "location":{
    "city": "Trento",
    "address":"Largo Medaglie d'oro, 9",
    "latitude":46.05684702214164,
    "longitude":11.132842787177719,
  },
  "entity_type":"Ospedale",
  "supervisor":{
    "first_name": "Mario",
    "last_name": "Grattarola"
  },
  "vaccine_issuing_department": "Dipartimento Sanitario",
  "institution_id": "OspedaleSantaChiara99999"
})
```

The outcome:

```
{ acknowledged: true,
  insertedId: ObjectId("61af2e104e899da2fe5c7989") }
```

In this command, we use the "InsertOne" command to specify all the requested fields and their values. The document inserted is structured in the same way as the already present ones: the fields are all 'simple' attributes except the 'location' and 'supervisor' fields that are two embedded documents.

After the command:

6. Application

The application has been developed to show how this database could be used in a real-world scenario. The application has two main purposes, firstly to generate QR codes which are the green pass of the individuals. Secondly, the application allows to update the database, inserting new data about swabs. Further information is provided in the user guide.