SMBUD Delivery #1
2021/22

Prof. Brambilla Marco

Reale Simone	(Personal Code: 10616980)
Shalby Hazem	(Personal Code: 10596243)
Somaschini Marco	(Personal Code: 10561636)
Urso Giuseppe	(Personal Code: 10628602)
Vitobello Andrea	(Personal Code: 10608395)

14 November 2021

POLITECNICO
MILANO 1863

Contents

1 Problem specification

The project purpose is to Design a graph data structure in a NoSQL DB[1] for supporting a contact tracing application for COVID-19. The database must be designed to record the following:

- People and their connections
- Time and place of contacts between people
- Personal data of each person (including vaccines, tests, contagion date and place)

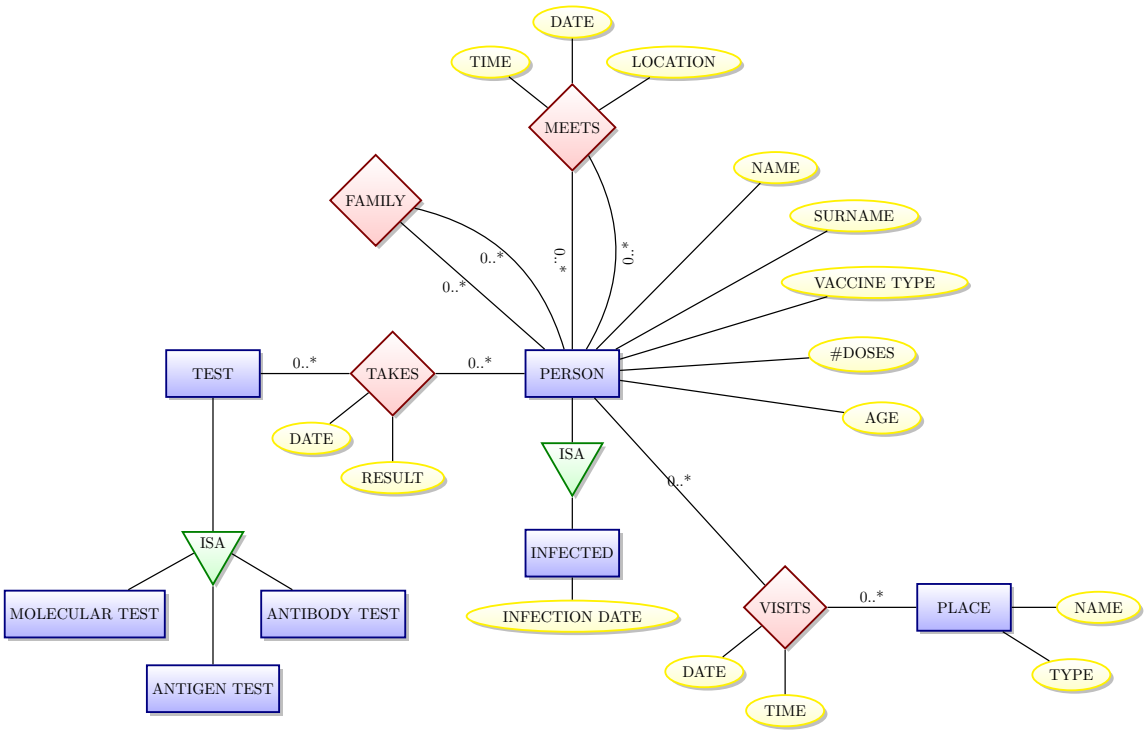
2 Hypothesis and assumptions

To design the DB structure the following assumptions on the data domain has been made:

- Vaccines are always the same for a certain person;
- The max number of doses is 3;
- Only people with age greater than 12 could be vaccinated;
- Only three type of covid test exists:
  - Molecular test;
  - Antigen test;
  - Antibody test.
- Only the date of contagion is known.
- Families are people with the same surname
- All contacts detected by contact tracing app or other devices is modelled by "MEETS" relationship (see ER), because the focus is on detecting possible contacts with infected people, while the application or the device that detected it is secondary and we have decided to not model it

[1] For this project we used Neo4j as graph database management system

3 ER diagram



The above ER diagram describe the model of the designed graph structure. In particular, each node could have the following label:

- **PERSON** which can be also **INFECT** in case of a positive Person.
- **TEST** which represents the test that a Person could take and in this case another label that describe the type of covid test is mandatory. In our system, as in the ER, three type of test are implemented: **MOLECULAR TEST**, **ANTIGEN TEST** and **ANTIBODY TEST**.
- **PLACE**

4 Data generation

The generation of data is done using a python script. For a better understanding of the data generation process go to Simulation section.

5 Queries

In this section we will discuss some of the most significant queries, but the database is designed in a way that facilitate any type of query.

5.1 Trend of the new infected patients over the last x days

```
MATCH (x:Infected)
RETURN x.p06_infectionDate, count(*)
```

The query simply displays the number of people infected for each day of the period analyzed in the simulation, whose length is defined by the parameters `start_date` and `end_date`.

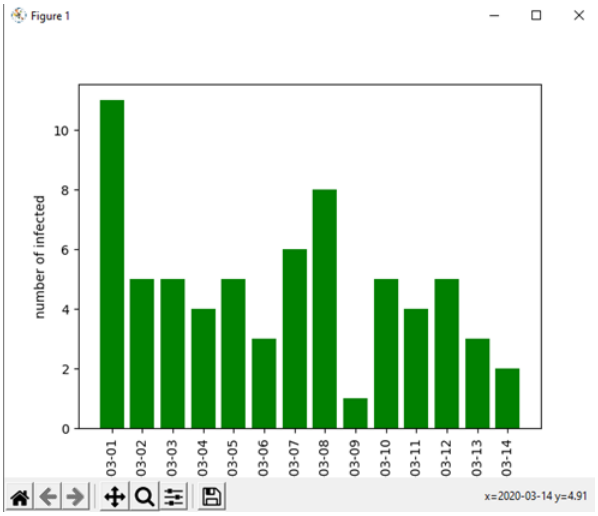


Figure 1: Example of the result of the above query plotted used the application described later

5.2 Locations associated to infected people

```
MATCH (x:Infected)-[r:VISITS]-(p:Place)
WHERE duration.inDays(x.p06_infectionDate, r.date).days <= 2
WITH count(r) as c, p.p02_type as type
RETURN type,c
```

The query aim is to find for each type of place, how many infected visited them in the last 2 days. The query is somehow pessimistic because it presume that all the infected people were already positive when they visited the Place.

5.3 People that went into contact with someone who got sick

The query's goal is to find those people that met or visited the same place as someone who afterwards, in a set interval of time, turned out to be infected. According to our design this query divides into two subqueries:

1. Here we want to check, for each infected individual, the places she visited the days prior the infection and match those people that visited that same place at the same date.

```
MATCH (a:Infected)-[v1:VISITS]-(p:Place)-[v2:VISITS]-(b:Person)
WITH duration.inDays(v1.date, a.p06_infectionDate).days AS DAYS_INTERVAL, b, v2, p
WHERE
NOT b:Infected AND
 v1.date = v2.date AND
 DAYS_INTERVAL > 0 AND
 DAYS_INTERVAL < $exposure_interval
RETURN b.p01_name AS Name, b.p02_surname AS Surname, v2.date AS DateOfContact,
p.p01_name AS PlaceOfContact
```

2. In this part we check for people that met and then one of them resulted positive.

```
MATCH (a:Infected)-[r:MEETS]-(b:Person)
WITH duration.inDays(r.date, a.p06_infectionDate).days AS DAYS_INTERVAL, b, r
WHERE
NOT b:Infected AND
DAYS_INTERVAL > 0 AND
DAYS_INTERVAL < $exposure_interval
RETURN b.p01_name AS Name, b.p02_surname AS Surname, r.date AS DateOfContact, r.place AS
PlaceOfContact
```

The parameter "exposure_interval" is how much back in time we search for potentially harmful contacts. The time granularity is of days and can be adjusted in the system files (by default is 5 days). The queries return the name and surname of the people potentially at risk of contagion, where and when the contact has occurred.

5.4 Find the average number of people met by infected ones, by kind of contact

```
CALL {
  MATCH (p:Infected)-[d2:VISITS]->(place:Place)<-[d1:VISITS]-(unfortunateSoul:Person)
  WHERE p.p06_infectionDate <= d1.date AND d1.date = d2.date
        AND NOT unfortunateSoul:Infected
  WITH count(unfortunateSoul) AS total1
  RETURN total1
}
CALL {
  MATCH (p2:Infected)-[d1:MEETS]->(unfortunateSoul:Person)
  WHERE p2.p06_infectionDate <= d1.date AND NOT unfortunateSoul:Infected
  WITH count(unfortunateSoul) AS total2
  RETURN total2
}
CAll{
MATCH (p3:Infected)-[d2:FAMILY_CONTACT]->(unfortunateSoul:Person)
WHERE NOT unfortunateSoul:Infected
WITH count(unfortunateSoul) AS total3
RETURN total3}
CALL{
MATCH (N:Infected)
RETURN count(N) AS totalInfected}
RETURN total1 * 1.0 / totalInfected, total2 * 1.0 / totalInfected, total3 * 1.0 /
↪  totalInfected
```

This query exploits subqueries to find different values in only one execution. In our database we can find three kind of relationships that connect people, and this query aims is to find the average for each of them; to do so, we look for the total number of contacts between an infected individual and another person, but since there can be three different ways two people can connect, it is necessary to divide the query into three parts. In addition to that, there is one more subquery that finds the total number of infected people so that the avereages can be computed. In fact, if the averages were computed in the queries specific to each relationship, they would only show the average values according to the number of infected that fulfills at least one such relationship.
Note that in this query only not infected people are considered while counting people met by infected ones, since the purpose of this query is to verify how the virus spreads to new, sane people.

5.5 Chart of number of people infected, subdivided by vaccine type (or no vaccine)

```
MATCH (x:Infected) RETURN x.p04_vaccine, COUNT(*)
```

This query gets as result, for each vaccine type (or no vaccine) the number of people actually infected / positive tested. Notice that if a person has not been vaccinated, the attribute `p04_vaccine` will be "no vaccine". After the execution of this query, a chart will be plotted to show how many people are infected for each vaccine type.

5.6 Determine most effective vaccine

The most effective vaccine is calculated with a (simplified) formula:

```
#of people infected with a certain vaccine / # total amount of people vaccinated
                          with the same certain vaccine
```

In this case, we use 2 queries to get the result:

- The query at the previous point provides the number of people infected per vaccine
- Another query to retrieve the total number of people vaccinated per vaccine type:

```
MATCH (x) RETURN x.p04_vaccine, COUNT(*)
```

Then, we simply calculate the ratio (via python code) using the results of the two queries and we take the absolute lowest ratio.

6 Commands

6.1 Add test

```
MATCH (n : Person{p01_name:$name, p02_surname:$surname}), (t:$test_type)
CREATE (n)-[r:TEST{date:$date,result:$test_result}]->(t)
FOREACH (p IN CASE WHEN r.result = true THEN[1] ELSE[] END | SET n.p06_infectionDate = r.date,
↪  n:Infected)
FOREACH (p IN CASE WHEN (r.result = false AND (NOT HAS(n.p06_infectionDate) OR
↪  n.p06_infectionDate < r.date)) THEN[1] ELSE[] END |  REMOVE n.p06_infectionDate,
↪  n:Infected);
```

The aim of the above command is to add the result of a covid test executed in a certain date to the corresponding person. To perform this action the command uses the following parameters:

- `$name`: the name of the patient;
- `$surname`: the surname of the patient;
- `$test_type`: the type of the test to add to the patient;
- `$date`: the date in which the test were executed;
- `$test_result`: the result of the test which is a boolean (true -¿ positive, false, negative).

Note that the command is designed in a way that allow the system to remain in a consistent state after the insertion of a new test record, and in particular two case are considered:

- if a person is tested positive, all the information about his infectivity state are update;
- if a positive person is tested negative, all the parameter about his infection period are resetted and this correspond in real life to the permission to end the quarantine period.

6.2 Add visit or meet relation

This command allows to add a contact occurred between two individuals, either because they were in the same monitored place in the same date or simply because they met at some point. Based on the previous cases the operations executed on the database are:

```
MATCH (a:Person {p01_name: $fn_A, p02_surname: $ln_A}), (b:Person {p01_name: $fn_B,
p02_surname: $ln_B}), (p:Place {p01_name: $place})
CREATE (a)-[:VISITS {date: date($date)}]->(p)<-[:VISITS {date: date($date)}]-(b)
```

and

```
MATCH (a:Person {p01_name: $fn_A, p02_surname: $ln_A}), (b:Person {p01_name: $fn_B,
p02_surname: $ln_B})
CREATE (a)-[:MEETS {date: date($date), place: $place}]->(b)
```

It is also possible to add the event of a single person visiting a monitored place (by leaving the second and third row empty). If this is the case then we execute the following command:

```
MATCH (a:Person {p01_name: $fn_A, p02_surname: $ln_A}), (p:Place {p01_name: $place})
CREATE (a)-[:VISITS {date: date($date)}]->(p)
```

6.3 Infect all families with at least 1 infected person

We know that the highest number of infections are related to family contacts, because it is easier to be infected when you use less cautions. So, we implemented a command to infect all families with at least 1 infected person, since this is a realistic scenario that appears quite frequently in real situations. The command is the following:

```
MATCH (x:Person:Infected)-[:FAMILY_CONTACT]-(y:Person)
SET y:Person:Infected SET y.p06_infectionDate=$date
```

Where `$date` is a parameter that (conventionally) is setted at `"2020-03-15"`, that is the last date in our simulation.

6.4 Add vaccine dose

```
MATCH (n:Person {p01_name : "input_name", p02_surname : "input_surname", p04_vaccine :
↪  "input_vaccine"})
SET n.p05_number_of_doses = 1 + n.p05_number_of_doses
UNION
MATCH (n:Person {p01_name : "input_name", p02_surname : "input_surname"})
WHERE n.p04_vaccine <> "input_vaccine"
SET n.p05_number_of_doses = 1 + n.p05_number_of_doses, n.p04_vaccine = "input_vaccine"
```

This simple command aims to add a single vaccine dose to a person's node. Two different scenarios are possible:

- if the objective is to add a dose of a certain vaccine to someone who has already had at least 1 dose of said vaccine, the number of doses will only increase by 1.
- if the objective is to add a dose of a certain vaccine to someone with no vaccine or with a different one, then the vaccine dose counter is increased by 1, while the vaccine type is overwritten as the latter is the most effective dose.

6.5 Simulation

The creation of the dataset is entrusted to a function called simulatePandemic, it outputs a perfectly coherent dataset directly available for consultation in neo4j.
On the first day (initial_date in config.py file) it starts by infecting the initial number of people given by the user, after that it starts inspecting the relations that they had and by doing that it finds the individuals that are at risk of infection.
Given a certain probability (it depends on multiple editable factors stored in conf.py) a person who has been in contact with a sick individual can be infected by the virus.
The set of ill people grows incrementally and so does the number of relations that need to be monitored by the application.

```
#initial date and final date
start_date = datetime.date(2020, 3, 1)
end_date = datetime.date(2020, 3, 15)
time_between_dates = end_date - start_date
days_between_dates = time_between_dates.days

#Exposure interval i.e. how many days before a positive test we check for trasmission of covid
exposure_interval = 10

vaccines = ('Pfizer', 'Astrazeneca', 'Moderna', 'Sputnik')
vaccine_probability = 0.7
probability_of_infection_with_vaccine = 0.3

#simulation parameters
proportion_of_people_initially_infected_no_vax = 0.6
proportion_of_people_initially_infected_1_vax = 0.2
proportion_of_people_initially_infected_2_vax = 0.1
proportion_of_people_initially_infected_3_vax = 0.07
proportion_of_people_initially_infected_4_vax = 0.03

type_of_places = ('Restaurant', 'Hospital', 'Theatre')
type_of_test = ("MOLECULAR_TEST", "ANTIGEN_TEST", "ANTIBODY_TEST")

proportion_n_of_people_n_of_place = 10 / 1
proportion_n_of_relationship_n_of_people = 1 / 1
#proportion for creating test connection with person
proportion_n_of_molecular_test_n_of_people = 1 / 10
proportion_n_of_antigen_test_n_of_people = 1 / 20
proportion_n_of_antibody_test_n_of_people = 1 / 50
#proportion of positivity per test
proportion_n_of_positive_n_of_molecular_test = 1 / 10
proportion_n_of_positive_n_of_antigen_test = 2 / 10
proportion_n_of_positive_n_of_antibody_test = 5 / 10
proportion_n_of_people_n_of_infected = 8 / 1
proportion_n_of_daily_test_n_of_people = 1 / 10
```

7 Application

To make all the above queries and commands easy to use for the users, we developed an application with the related GUI.
The application GUI is simply used to get the parameters necessary to run the query/command and to visualize the results of the queries.
For a better understanding of the application, a copy of the whole project is available on the Github repository of the project and also a small demo is available on YouTube link.