SMBUD 2021 - Project work 1

Aman Gabba
Andrea Cerasani
Giovanni Demasi
Pasquale Dazzeo
Vlad Marian Cimpeanu

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Problem Specification

The aim of this project was to design a 'query graph data structure' in Neo4j for supporting a contact tracing application for COVID-19. The database must register all the necessary information about the users including vaccines and Covid swabs in order to have a pandemic trend overview for a given country. The application using this database will be able to exploit all the data coming from tracking applications and from all the public facilities.

1.2 Hypothesis

The assumptions taken into account are the following:

- People belonging to the same family live in the same house if not explicitly specified. Using the concept of "house" instead of "family", it is offered the possibility to differentiate domicile from residence.
- All the personal data are verified by an authoritative figure, for instance the government.
- The domicile declaration is assumed to be truthful.
- All the data coming from public spaces are always considered true and complete.
- People always provide all the necessary information to the staff when they visit a certain public space.
- Every MEETS relationship is automatically added to the database by a tracing app when two mobile phones stay in the same range for more than 15 minutes. We assume that to every device corresponds only one person and viceversa.
- Relationships use the Data type instead of Timestamp (with the exception of Test relationship because could happen that a person due to a positive resulting rapid test do in the same day another test which results negative - the use of Timestamp is important to recognise false positive) to register the relations because of safety and simplicity reasons. The former reason allows people using the Database to trace contacts during all the day and not only during a range of time (for more control). The latter reason is required due to avoid mistakes by the staff of public facilities during the time registration (an error in time registration could lead to a wrong tracing).
- The designed model doesn't take into consideration the deaths, as the main focus of the project was the tracking of Covid infections. However in a more complete model that considers this aspect, a death node could

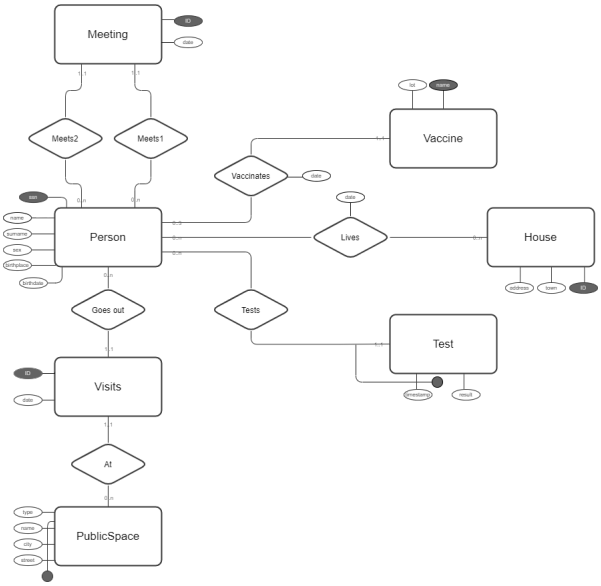
be used to represent it through a dead relationship that connect the dead person to the death node with a date attribute. This choice could be useful to analyze the death/infection ratio related to the Covid pandemic.

2 ER diagram

The designed ER diagram contains the following entities: Person, Public Space, Vaccine, House and Covid Test. The other classes have been introduced due to ER diagram correctness reasons, but they haven't been taken into account during the design of the Graph Database because the last one allows a more flexible design.

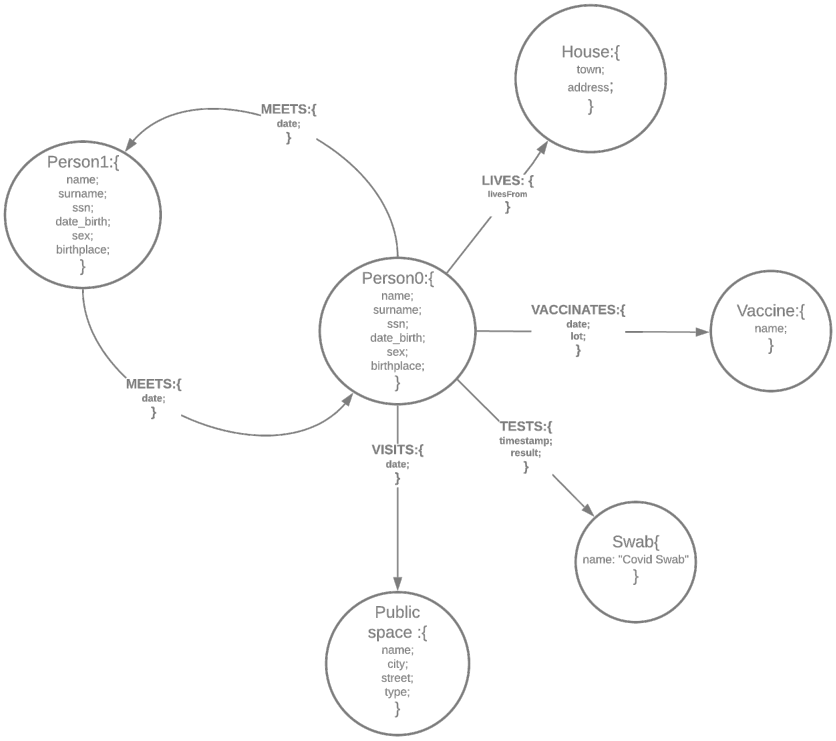
As said before the concept of 'Family' has been replaced with the concept of 'House' since it is more realistic and more useful for contact tracing.

The Person entity has SSN ('Codice Fiscale' in Italy) as primary key and all the relevant data. Every Vaccine has its name and lot. Houses are described by a unique ID, address and town. Covid tests are identified by the timestamp and they also have the result as attribute. Every Public Space has name, city and strees as primary key and also a type attribute to identify its category. Visits and Meeting entities, as said before, have been introduced due to correctness aspect but they are represented only as relationship in the graph diagram. Lives and Vaccines relationship are identified by date.



3 Graph diagram

The designed Graph diagram is shown below. Person, House, Vaccine, Public Space and Covid Swab are represented as nodes and they all have the same attributes shown in the ER above. The relationship of the database are Meets, Lives, Tests, Vaccinates and Visits, they all have date as attribute (excepted tests that has timestamp). Vaccinate also has the lot of the vaccine and Tests also has the result of the test.



4 Dataset description

The Dataset has been built by making a script in Python, using some useful packages like: random-italian-people that automatically generates people with random attributes; the official binary Neo4j driver for Python for the communication with the Database since it allows to make queries directly from Python.

We have also used this piece of code to generate amenities data. This query has been used in overpass turbo (a web-based tool for extracting OpenStreetMap data) selecting Roma, Napoli and Milano.

```
[out:csv("amenity","name", "addr:street", "addr:housenumber", "addr:city
    "; true; ",")];
// gather results
(
  //query part for: amenity= post_box
  node"amenity"="restaurant";
  node"amenity"="bar";
  node"amenity"="museo";
  node"amenity"="cinema";
);
// print results
out body;
>;
out skel qt;
```

5 Queries and Commands

5.1 Queries

The following query can be used to show the percentage of vaccinated people in an age range. X and Y must be two positive integers with X <= Y.

```
MATCH (sample: Person)
WHERE duration.between(date(sample.birthdate), date()).years >= X
      AND
      duration.between(date(sample.birthdate), date()).years <= Y
WITH sample AS sizeSample
MATCH (vaccinated:Person)-[:VACCINATES]->(vaccine: Vaccine)
WHERE duration.between(date(vaccinated.birthdate), date()).years >= X
      AND
      duration.between(date(vaccinated.birthdate), date()).years <= Y
RETURN
CASE
WHEN COUNT(DISTINCT vaccinated) = 0 THEN 0.0
WHEN COUNT(DISTINCT sizeSample) = 0 THEN 0.0
ELSE (COUNT(DISTINCT vaccinated) * 1.0 / COUNT(DISTINCT sizeSample) *
    1.0) * 100.0
END AS Percentage
```

Query that returns the infection ratio among all the tested people for each month.

```
MATCH ()-[t:TESTS]->()
WITH date(apoc.date.format(apoc.date.parse(t.timestamp, 'ms', 'yyyy-MM-dd
    '), 'ms', 'yyyy-MM-dd')).month AS month,
     date(apoc.date.format(apoc.date.parse(t.timestamp, 'ms', 'yyyy-MM-dd
    '), 'ms', 'yyyy-MM-dd')).year AS year, count(t) as all_tests
OPTIONAL MATCH ()-[t:TESTS]->()
WHERE date(apoc.date.format(apoc.date.parse(t.timestamp, 'ms', 'yyyy-MM-
    dd'), 'ms', 'yyyy-MM-dd')).month = month AND
    date(apoc.date.format(apoc.date.parse(t.timestamp, 'ms', 'yyyy-MM-dd
    '), 'ms', 'yyyy-MM-dd')).year = year AND
    t.res = 'Positive'
RETURN round((COUNT(t) * 1.0 / all_tests * 1.0) * 100.0 * 100.0) / 100.0
    AS ratio, month, year
ORDER BY year DESC, month DESC
```

The following query finds false negatives (for false positive it is necessary to switch pre with post in the last two lines before the Return statement).

```
MATCH (p: Person)-[pre: TESTS]->()<-[post:TESTS]-(p)
WHERE duration.between(datetime({ epochMillis: apoc.date.parse(pre.
    timestamp, 'ms', 'yyyy-MM-dd HH:mm:ss')}), datetime({ epochMillis:
    apoc.date.parse(post.timestamp, 'ms', 'yyyy-MM-dd HH:mm:ss')})).
    hours < 24
AND duration.between(datetime({ epochMillis: apoc.date.parse(pre.
    timestamp, 'ms', 'yyyy-MM-dd HH:mm:ss')}), datetime({ epochMillis:
    apoc.date.parse(post.timestamp, 'ms', 'yyyy-MM-dd HH:mm:ss')})).
    hours > 0
AND pre.res="Negative"
AND post.res="Positive"
RETURN p, pre, post
```

This query can be used to analyze the Vaccine efficacy computed as the ratio of the vaccinated who became positive over the total number of vaccinated people.

```
MATCH (p1:Person)-[:VACCINATES]->(v1:Vaccine {name: 'AstraZeneca'})
MATCH (p2:Person)-[:VACCINATES]->(v2:Vaccine {name: 'Moderna'})
MATCH (p3:Person)-[:VACCINATES]->(v3:Vaccine {name: 'Pfizer'})
MATCH (p4:Person)-[:VACCINATES]->(v4:Vaccine {name: 'Jensen'})
WITH COUNT(DISTINCT p1) AS vaccinated_astrazeneca, COUNT(DISTINCT p2) AS
    vaccinated_moderna, COUNT(DISTINCT p3) AS vaccinated_pfizer, COUNT(
    DISTINCT p4) AS vaccinated_jensen
OPTIONAL MATCH ()<-[t1:TESTS]-(p1:Person)-[v1:VACCINATES]->(vacc1:Vaccine
     {name: 'AstraZeneca'})
WHERE t1.res = 'Positive'
    AND date(apoc.date.format(apoc.date.parse(t1.timestamp, 'ms', 'yyyy-
    MM-dd'), 'ms', 'yyyy-MM-dd')) > date(v1.date)
OPTIONAL MATCH ()<-[t2:TESTS]-(p2:Person)-[v2:VACCINATES]->(vacc2:Vaccine
     {name: 'Moderna'})
WHERE t2.res = 'Positive'
    AND date(apoc.date.format(apoc.date.parse(t2.timestamp, 'ms', 'yyyy-
    MM-dd'), 'ms', 'yyyy-MM-dd')) > date(v2.date)
OPTIONAL MATCH ()<-[t3:TESTS]-(p3:Person)-[v3:VACCINATES]->(vacc3:Vaccine
     {name: 'Pfizer'})
WHERE t3.res = 'Positive'
    AND date(apoc.date.format(apoc.date.parse(t3.timestamp, 'ms', 'yyyy-
    MM-dd'), 'ms', 'yyyy-MM-dd')) > date(v3.date)
OPTIONAL MATCH ()<-[t4:TESTS]-(p4:Person)-[v4:VACCINATES]->(vacc4:Vaccine
     {name: 'Jensen'})
WHERE t4.res = 'Positive'
    AND date(apoc.date.format(apoc.date.parse(t4.timestamp, 'ms', 'yyyy-
    MM-dd'), 'ms', 'yyyy-MM-dd')) > date(v4.date)
WITH vaccinated_astrazeneca,vaccinated_moderna,vaccinated_pfizer,
    vaccinated_jensen, COUNT(DISTINCT p1) AS vaccinated_infected_astra,
```

```
    COUNT(DISTINCT p2) AS vaccinated_infected_moderna, COUNT(DISTINCT p3
    ) AS vaccinated_infected_pfizer, COUNT(DISTINCT p4) AS
    vaccinated_infected_jensen
RETURN (1- (toFloat(vaccinated_infected_astra)/toFloat(
    vaccinated_astrazeneca)))*100 AS AstraZenecaEfficacy,
    (1- (toFloat(vaccinated_infected_moderna)/toFloat(vaccinated_moderna)
    ))*100 AS ModernaEfficacy,(1- (toFloat(vaccinated_infected_pfizer)/
    toFloat(vaccinated_pfizer)))*100 AS PfizerEfficacy,(1- (toFloat(
    vaccinated_infected_jensen)/toFloat(vaccinated_jensen)))*100 AS
    JensenEfficacy
```

5.2 Commands

The following command can be used to add a vaccination. XXX is the fiscal code of the person receiving the vaccine (16 digits alphanumeric code), YYY is the name of the vaccine ('AstraZeneca', 'Jensen', 'Pfizer', 'Moderna'), ZZZ is the lot of the vaccine (it must must be in format AZ##, P##,J##,M##). WWW is the date of the vaccination and must be in the following format YYYY-MM-DD.

```
MATCH (person_to_vaccinate: Person),
      (vax: Vaccine)
WHERE person_to_vaccinate.ssn = 'XXX'
      AND
      vax.name = 'YYY'
CREATE (person_to_vaccinate)-[:VACCINATES {lot: 'ZZZ',
        date: 'WWW'}]->(vax)
```

The following command can be used to add a Covid Test. XXX is the fiscal code of the person tested (16 digits alphanumeric code), YYY is the result of the test ('Positive' or 'Negative'). WWW is the timestamp of the test and must be in the following format 'YYYY-MM-DD hh:mm:ss'.

```
MATCH (person_tested: Person),
      (test: Swab)
WHERE person_tested.ssn = 'XXX'
      AND
      test.name = 'Covid Swab'
CREATE (person_tested)-[:TESTS {res: 'YYY', timestamp: 'WWW'}]->(test)
```

6 References and Sources

Some CSV files have been used for the creation of amenities attributes, like a list of addresses of different cities.

7 Conclusion

Some interesting conclusions can be drawn from the development of this project:
Graph databases are, if well designed, easy to use and really scalable.
The creation of credible databases using Python could be really convenient to simulate a real situation and understand in advance which behaviours could be useful to manage the situation in exam.
Graph databases are more flexible than the classical relational databases since they allow to create same situations in an easier and more realistic way.