Politecnico di Milano

Systems and methods for big and unstructured data

SMBUD project: first delivery

Group 24

Basso Paolo 10783951

Aiello Andrea 10863133

Borsatto Andrea 10628989

Cavalli Dario 10820532

Petriconi Emanuele 10577000

Academic Year 2021–2022

Contents

Chapter 1

Specifications

1.1 Specifications

The scope of this project is to design and store a graph data structure in Neo4J to effectively represent a contact tracing system with the aim of monitoring the spread of Covid-19 in a realistic scenario.
For each individual, their personal data must be recorded (full name, gender, email, telephone number, date of birth).
There are three types of Covid-19 tests available: serological, molecular and rapid. Every person can take multiple tests and for each of them, the test date and related result, published on the same day, must be stored. A person is considered *"positive"* when his last molecular test result is positive and textit"negative" if it has no tests, or the last molecular test result is negative. If a person is found positive by one of the other two tests, the individual is advised to take a molecular test.
A person is considered *"in contact"* only if it had a direct contact with a positive by one of the following scenarios:

- Same household members are always considered *"in contact"* between each other.
- **Explicit data collection**: the database must store a set of locations and track the people which visit that location. Each location has an average stay time, for example a cinema has an average stay of three hours while a barber shop only 30 minutes.
- **Contact tracing application and devices**: a person could get in touch directly with another person if the contact tracing app/device detects a close distance encounter.

If a person is found positive, then all the people that have been in contact with him in the last 3 days must be considered *"in contact"*. In order to reduce the size of the dataset, old contacts should be deleted after 21 days.
Furthermore, people can be vaccinated with one of the available vaccines on the market. The database keeps track of every type of vaccine along with other related information, such as their time coverage and minimum number of doses to be considered effective. For a multiple doses vaccine, the time coverage may vary (usually shorter for the first dose).

Chapter 2

ER diagram

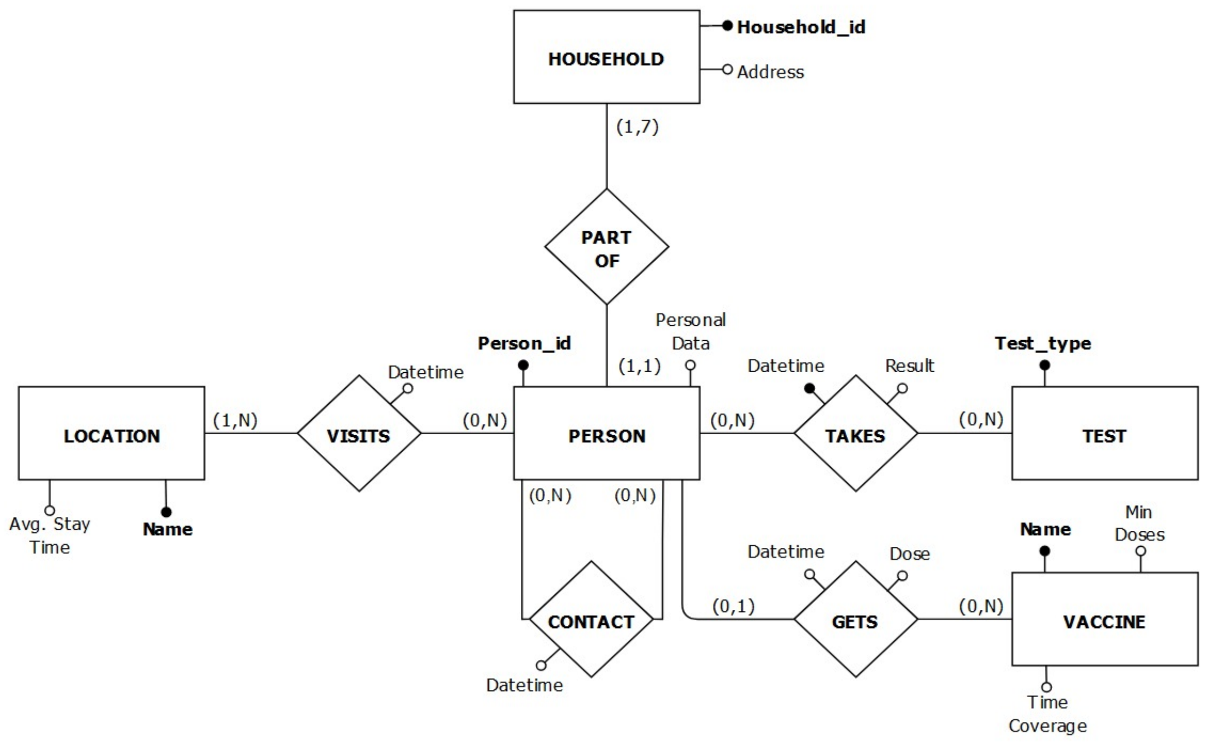


Figure 2.1: ER diagram

Chapter 3

Graph example

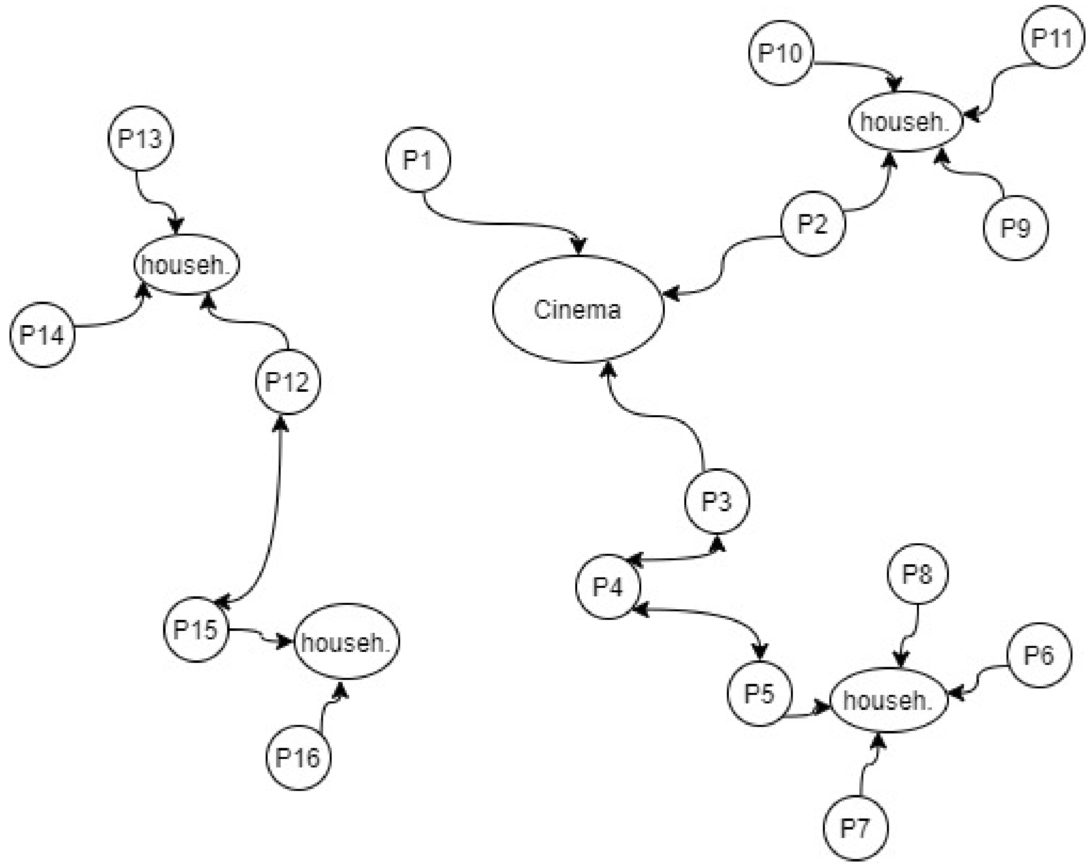


Figure 3.1: Graph example

Chapter 4

Model assumptions

4.1 Model assumptions

In our model we assume that 60% of the people have been vaccinated; we also assume that if a person is vaccinated then, the probability to be positive after a test is 20%, while if he or she is not vaccinated, the probability rises to 60%.
People that got the first dose of a vaccine that allows a second dose, have 50% probability to get also the second dose.
We decided to generate only 50 covid tests a day for a month (from 2021-10-14 to 2021-11-14) to reduce the dimension of the query built.

Chapter 5

Dataset generation

5.1 Dataset generation

We created a python script which uses a library that interacts with `https://randomuser.me/documentation` to generate random people data.
In the script we first generate the nodes for vaccines, locations and tests because they are fixed; then we generate the households and the people within the same household.
At this point we create associations between people and vaccines, the associations between couples of people that have been in direct contact with each other, associations between people and tests and finally people and locations.
All the data generated respect the above assumptions and refer to a time interval that goes from *2021-10-14* to *2021-11-14*.
The script writes the generated query in a file called *"Query.txt"* and this generation script is included in the zip file (it needs the pip package `https://pypi.org/project/randomuser/`).

5.1.1 Importing the dataset

The easy way to generate the dataset is to copy the content of *"Query.txt"* and paste it in the neo4j console. If you receive a timeout you should just retry the query since the already applied parts will be really fast. In our testing about three runs were necessary (depending on the setted timeout).
Another way to import the dataset is using the dump file we created and included in the zip.

Chapter 6

Queries

6.1 Queries

For some of these queries the official neo4j plugin APOC must be installed (`https://neo4j.com/developer/neo4j-apoc/#installing-apoc`). Sandbox environments on the neo4j website already have APOC installed.

1. **Find the people who had contact with a person tested positive on 2021-11-13 based on family, direct contacts and same location visits**.

```
MATCH (:Test {type:"Molecular"})-[:TAKES {result:"Positive"}]-(
    positivo:Person)
WITH collect(positivo) AS coll
MATCH (:Test {type:"Molecular"})-[testPositivo:TAKES {result:"
    Positive"}]-(positivo:Person)
WHERE date(testPositivo.datetime) = date("2021-11-13")
CALL {
        // Find people to test based on location
        // Only if they've visited the location the same time as
             the positive
        // and considering only locations visited since three
            days prior to the test by the positive
        WITH positivo, coll, testPositivo
        MATCH (positivo)-[wasat:VISITS]-(location:Location)-[
            wasatVis:VISITS]-(visitors)
        WHERE wasat.datetime <= wasatVis.datetime
        AND apoc.date.add(wasat.datetime.epochSeconds, "s",
            location.avg_stay_time, "hour") > wasatVis.datetime.
            epochSeconds
        AND apoc.date.add(wasat.datetime.epochSeconds, "s", 3, "
            day") > testPositivo.datetime.epochSeconds
        AND NOT visitors IN coll
        RETURN visitors AS to_test
        UNION

        // Find people to test based on contact
        // Only if they've had contact with the positive since
            three days prior to the test
        WITH positivo, coll, testPositivo
        MATCH (positivo)-[contatto:CONTACT]-(persona:Person)
        WHERE apoc.date.add(contatto.datetime.epochSeconds, "s",
            3, "day") > testPositivo.datetime.epochSeconds
        AND NOT persona IN coll
        RETURN persona AS to_test
        UNION

        // Find people to test based on same family
        WITH positivo, coll, testPositivo
        MATCH (positivo)
        MATCH (positivo)-[:PART_OF]-(:Family)-[:PART_OF]-(fam:
           Person)
        WHERE NOT fam IN coll
        RETURN fam AS to_test
}
RETURN DISTINCT to_test
```

2. **Percentage of vaccinated people with activated antibodies**.

```
MATCH (p:Person)
WITH count(*) AS pTot
MATCH (pv:Person)-[vdone:GETS]-(vacc:Vaccine)
WHERE
        CASE
        WHEN  (vacc.min_doses= 1)
                THEN date(vdone.datetime) < date()-duration({
                   Days:15})
                AND date(vdone.datetime) > date()-duration({Days
                   :365})
        WHEN (vacc.min_doses= 2 AND vdone.dose = 1)
                THEN date(vdone.datetime) < date()-duration({
                   Days:15})
                AND date(vdone.datetime) > date()-duration({Days
                   :vacc.coverage})
        WHEN (vacc.min_doses = 2)
                THEN date(vdone.datetime) > date()-duration({
                   Days:365})
        END
WITH count(distinct pv) AS pVacc, pTot
RETURN pVacc*100/pTot
```

3. **Average number of people that came in contact with a positive 3 days before the positive molecular test.**

```
MATCH (:Test {type: "Molecular"})-[testPositivo:TAKES {result:"
   Positive"}]-(positivo:Person)
CALL {
        WITH positivo, testPositivo
        MATCH (positivo)-[r:CONTACT]-(contacts:Person)
        WHERE date(r.datetime) >= date(testPositivo.datetime)-
           duration({Days:3})
        RETURN contacts
        UNION
        WITH positivo, testPositivo
        MATCH (positivo)-[rw1:VISITS]-(location:Location)-[rw2:
           VISITS]-(contacts:Person)
        WHERE rw1.datetime.epochSeconds < rw2.datetime.
           epochSeconds < apoc.date.add(rw1.date.epochSeconds, "
           s", location.avgST, "h")
        AND date(testPositivo.datetime)-duration({Days:3}) <
           date(rw1.datetime)
        RETURN contacts
        UNION
        WITH positivo
        MATCH (positivo)-[:PART_OF]-(:Family)-[:PART_OF]-(
           contacts:Person)
        RETURN contacts
}
WITH positivo, count(contacts) AS countCon
RETURN sum(countCon)*1.0/count(positivo)
```

4. **Most visited location by people found positive at most 14 days after the visit**.

```
MATCH (location:Location)-[visit:VISITS]-(person:Person)-[takes:
   TAKES{result:'Positive'}]-(test:Test{type:'Molecular'})
WHERE visit.datetime >= takes.datetime - duration({days: 14})
AND visit.datetime <= takes.datetime
RETURN location, count( person)
ORDER BY count(person) DESC
LIMIT 1
```

5. **Number of positive molecular tests per day in the last week (from 2021-10-20). If in a day there are 0 positive molecular tests then this is reflected in the result**.

```
MATCH (:Person)-[takes:TAKES]-(:Test{type:'Molecular'})
WHERE date(takes.datetime) > date("2021-10-20")-duration({Days
    :7})
WITH date(takes.datetime) AS dates
WITH collect(dates) AS dates

MATCH (:Person)-[takes:TAKES {result:'Positive'}]-(:Test{type:'
    Molecular'})
WHERE date(takes.datetime) > date("2021-10-20")-duration({Days
    :7})
WITH dates, date(takes.datetime) AS datesPositive
WITH dates, collect(datesPositive) AS datesPositive

CALL {
    WITH dates, datesPositive
        MATCH (person:Person)-[takes:TAKES {result:'Positive
            '}]-(:Test{type:'Molecular'})
        WHERE date(takes.datetime) IN dates
        RETURN date(takes.datetime) AS date, count(DISTINCT
           person) AS number_of_positives
UNION
        WITH dates, datesPositive
        MATCH (:Person)-[takes:TAKES]-(:Test{type:'Molecular'})
        WHERE date(takes.datetime) IN dates
        AND takes.result <> 'Positive'
        AND NOT date(takes.datetime) IN datesPositive
        RETURN date(takes.datetime) AS date, 0 AS
           number_of_positives
}
RETURN date, number_of_positives
ORDER BY date(date) DESC
LIMIT 7
```

6. **Given a location, count the total number of visits in the last 30 days**.

```
WITH "Cinema2" as locationName
MATCH (location:Location {name:locationName})-[visits:VISITS]-(
   person:Person)
WHERE date(visits.datetime) >= date("2021-11-13")-duration({Days
   : 30})
RETURN count(*)
```

Chapter 7

Commands

7.1 Commands

1. **Create a new test result**.

   ```
   // parameters: test type, personId, datetime, result
   WITH "Molecular" AS testType, 546 AS personId, "2021-11-05T00
      :00:00Z" AS testDate, "Positive" AS testResult
   MATCH (person:Person)
   WHERE id(person) = personId
   WITH person, testType, testDate, testResult
   CREATE (test {type:testType})<-[:TAKES {datetime:datetime(
      testDate), result:testResult}]-(person)
   ```

2. **Add a visit (a person visit a location)**.

   ```
   // parameters: location name, personId, datetime
   WITH "Cinema3" AS locationName,  546 AS personId,  "2021-10-05
      T00:00:00Z" AS visitDate
   MATCH (person:Person)
   WHERE id(person)=personId
   CREATE (person)-[:VISITS {datetime:datetime(visitDate)}]->(
      location:Location {name:locationName})
   ```

3. **Add a vaccination**.

   ```
   // parameters: vaccine name, personId, datetime, dose n
   WITH "Moderna" AS vaccineName, 546 AS personId,  "2021-10-05T00
      :00:00Z" AS vaccinationDate, 2  AS vaccinationDose
   MATCH (person:Person)
   WHERE id(person)=personId
   MERGE (person)-[gets:GETS]-(vaccine:Vaccine {name:vaccineName})
   ON CREATE
           SET gets.dose=1, gets.datetime=datetime(vaccinationDate)
   ON MATCH
           SET gets.dose=vaccinationDose, gets.datetime=datetime(
              vaccinationDate)
   ```

4. **Delete contacts older than 21 days**.

```
MATCH (:Person)-[contacts:CONTACT]-(:Person)
WHERE date(contacts.datetime) < date("2021-11-13")-duration({
    Days:21})
DELETE contacts
```

Chapter 8

UI implementation

8.1 UI implementation

As a possible implementation example we created an user interface which could be used to visualize the data present in the database. In particular, this application asks the credentials to connect to the database and which type of chart the user wants to see.
Currently the implementation is able to generate:

1. Vaccination ratio pie chart.
2. Top positive locations: the most visited location by people which then came out as positive less than 14 days after their last visit.
3. Number of positive molecular tests in the last 7 days bar chart.
4. Vaccination distribution pie chart.

The application is written in JavaScript and we used the official neo4j-driver to query the database.

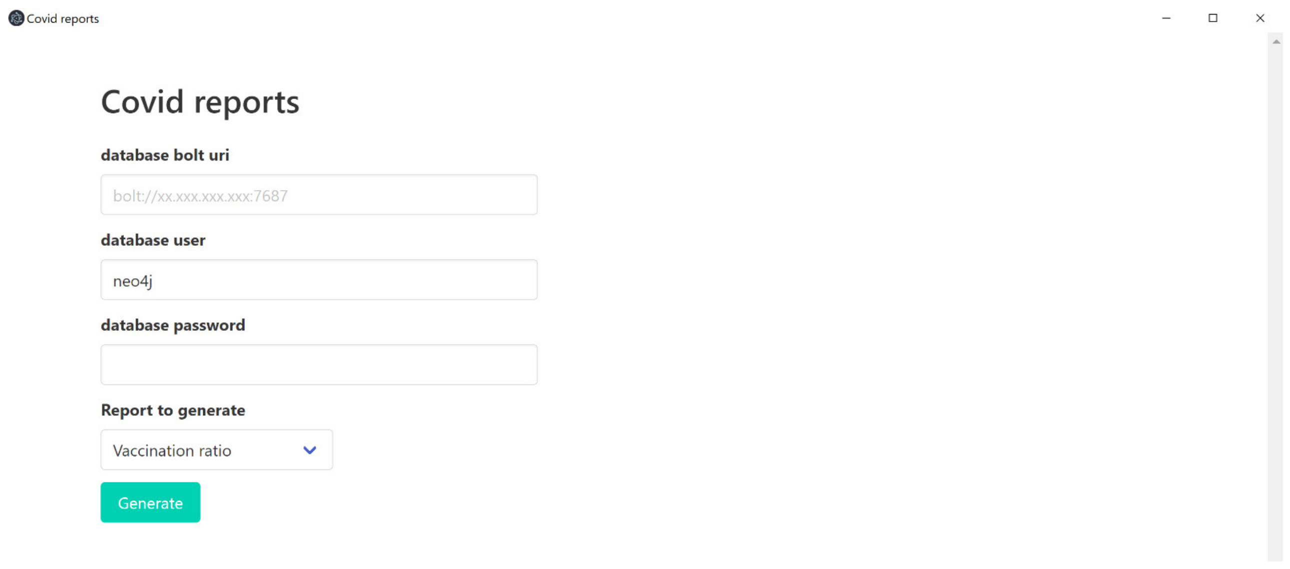
8.1.1 Installation

Windows 32/64bit
Inside the implementation folder there will be a zip for the implementation compiled for Windows 64 bit and another zip for Windows 32 bit.
Extract the corresponding zip and open the file smbud-neo4j.exe.
Mac
Unfortunately all members of this team do not use a Mac so we were not able to compile the implementation for it. However it is possible to run the application directly from the source code (see below).
Run from source

- Install Node.js if not already present in the system: `https://nodejs.org/en/download/`.
- Extract the zip containing the source code.
- Inside the source code folder run: npm instal.
- Inside the source code folder run: npm start.
- The UI should start.

Note: this method is for development purposes and the UI might be a bit slower than the compiled one.

8.1.2 ScreenShots



Covid reports

Covid reports

database bolt uri

bolt://xx.xxx.xxx.xxx:7687

database user

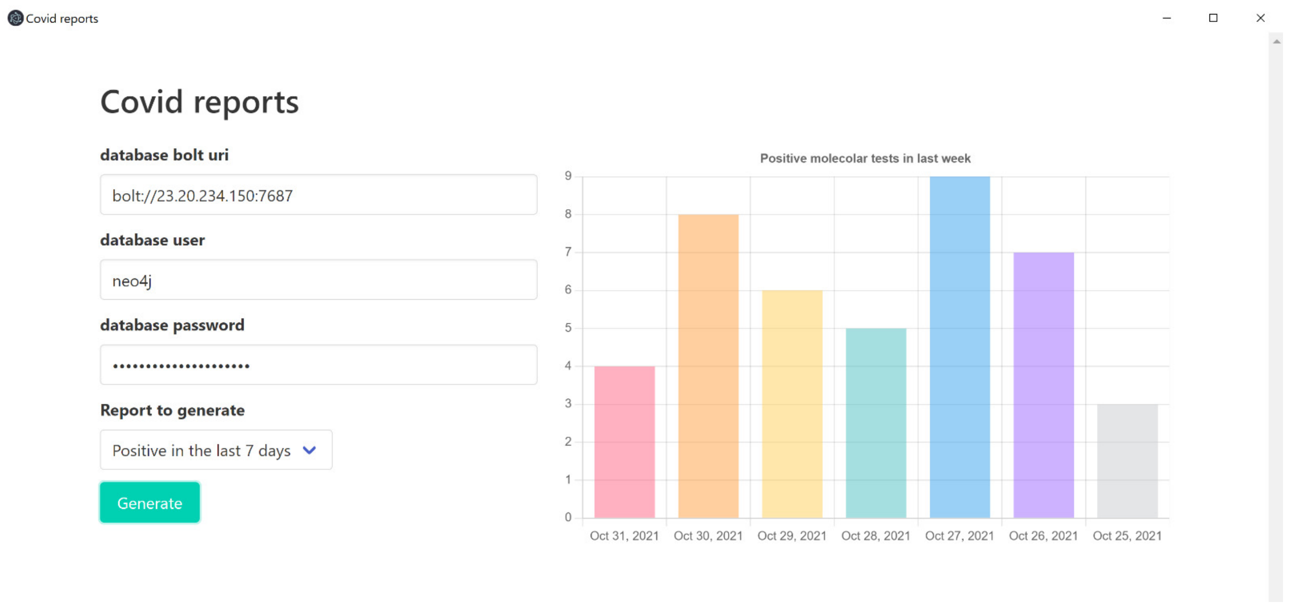
neo4j

database password

Report to generate

Vaccination ratio

Figure 8.1: First page



Covid reports

Covid reports

database bolt uri

bolt://23.20.234.150:7687

database user

neo4j

database password

••••••••••••••••••••

Report to generate

Positive in the last 7 days

Figure 8.2: Generated report