POLITECNICO
MILANO 1863

System and Methods for Big and Unstructured Data

Course project 2021 - First part

Contact Tracing System

Authors

Marcello Bavaro - 10597421

Francesco Puddu - 10583315

Filippo Rescalli - 10609906

Elena Righini - 10796621

Simone Sarti - 10580595

Table of contents

1 Overview

The goal of this project is to build an information system for the management of data related to a pandemic (like the COVID-19 one) in a given country.

In this document we describe the first part, which consists in a data architecture to support contact tracing with the aim of monitoring the diffusion of the disease. In particular, we will leverage a graph database built with Neo4j.

The main purpose of the system is to notify users which came into contact with someone that was tested positive for the virus. Such contact can take place:

- **When people live together:** all the people who live with the infected subject should be notified.
- **In public places (restaurants, bars ...):** all the people who were in the same location at the same time as the infected subject should be notified. In these places, presence is registered. We keep track of both the time and place of the contact.
- **Other locations:** people who met the infected subject in a place where no bookkeeping is done (like a park) should be notified after the contact is registered by a contact tracing app. In this case, we will only keep track of the time of the contact (we don't want the contact tracing app to constantly know the position of the owner of the device)

The system described in the following sections will also keep track of vaccines and manage tests.

2 Assumptions

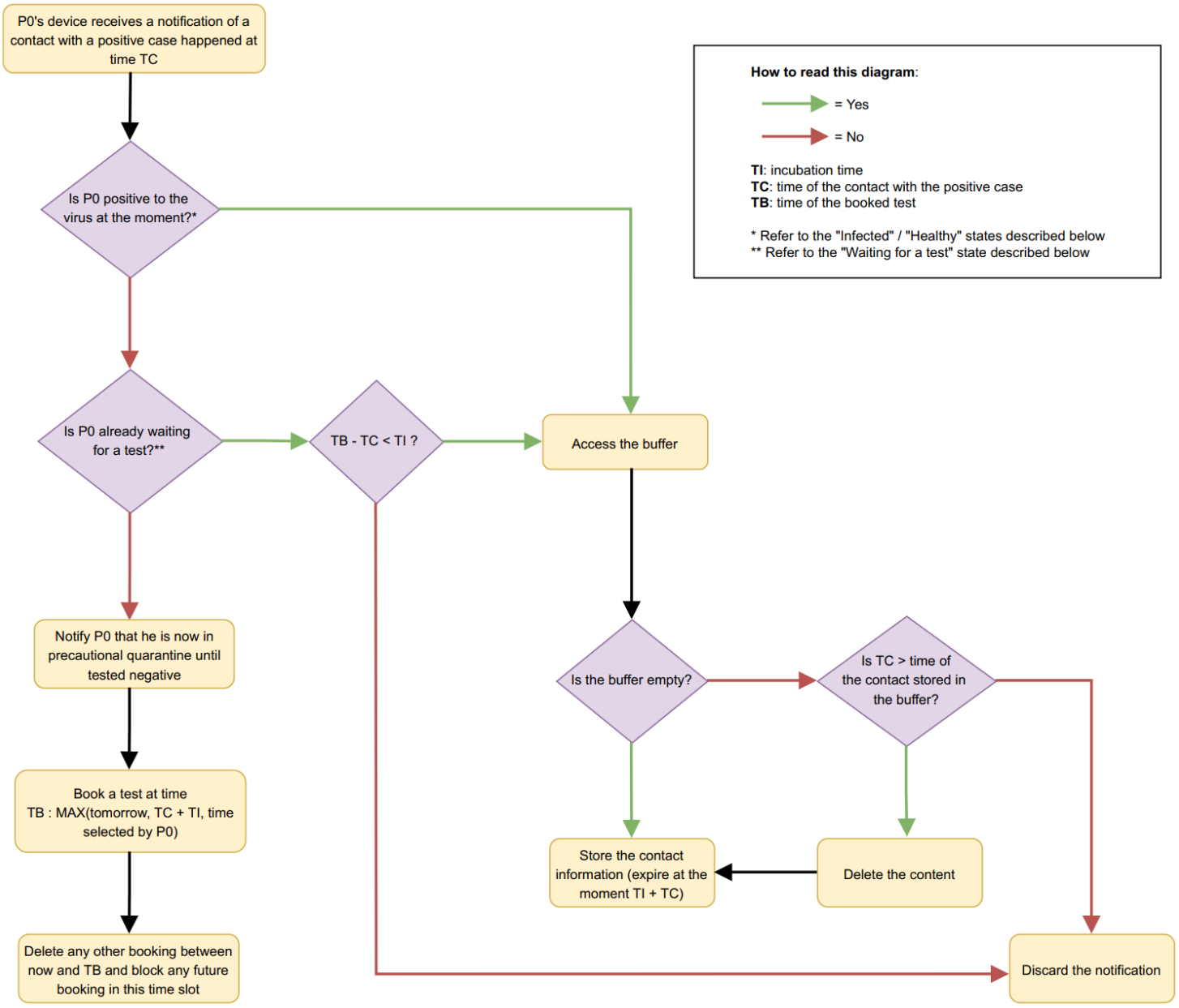
For our system to work properly, we need to make some assumptions:

- The national health system will provide us with data about the vaccines and tests performed
- Public places (like restaurants) will provide use with the log of visits (date and time of arrival and departure for each person)
- People living together are always considered as being in contact
- A person is treated as infected after exactly one positive test, we won't deal with testing errors
- A person who receives a notification of exposure will go and get tested
- A person that tested positive is quarantined and cannot go out. Also, a person that was notified is in preventive quarantine and cannot go out.

3 Dynamics of the System

To better describe our idea, we'll refer to an exemplary situation in which P0 is a person who contracted the virus.
One day, P0 decides to get tested for the virus, and the result is positive. Just after the result is known, the database is queried to discover which people came into contact with P0, and in which circumstances, in a proper time window (our contact tracing window).
Once we have the list of people to notify and the corresponding contact event, there are a variety of cases to manage depending on the state of each one.
The flow of events, starting from the reception of the notification on P1's personal device, is summed up in the diagram below.
The reader can refer to the following pages for a more detailed explanation.

3.1 Handling a notification



The management of the notification derived from contact tracing depends on the current state of the receiver:

- If the receiver is in the "healthy" state, the system will proceed with booking a test and informing the subject that he/she is now in precautional quarantine (corresponding to the "waiting for a test" state in our model). The details of the booking system will be discussed later in the document.
- If the receiver is in the "infected" state there is no need to notify him/her of a potential contagion, but (if the contact is recent enough to be relevant) we temporarily* store the contact information. This data will eventually be used to validate a negative test: in our model, such result is considered to be valid only if no contact with a positive person happened recently**.
  Note that, under our assumptions, an infected person should be quarantined and thus isolated. Nevertheless, we modeled the eventuality of an infected person receiving visits (that is typically the case for fragile subjects).
- If the receiver is in the "waiting for a test" state, our system once again needs to decide whether the contact data is relevant to the subject or not. To validate an eventual future negative result of a test, we need to buffer the data related to the most recent contact happened shortly before the moment of the test**. Consequently, when a new notification is received, the system will discard it only if a more recent contact is already known.

* the expiration date is set to $T_i + T_c$ to target the moment in which the contagion is detectable
** the system needs to exclude the possibility of the virus being still incubating

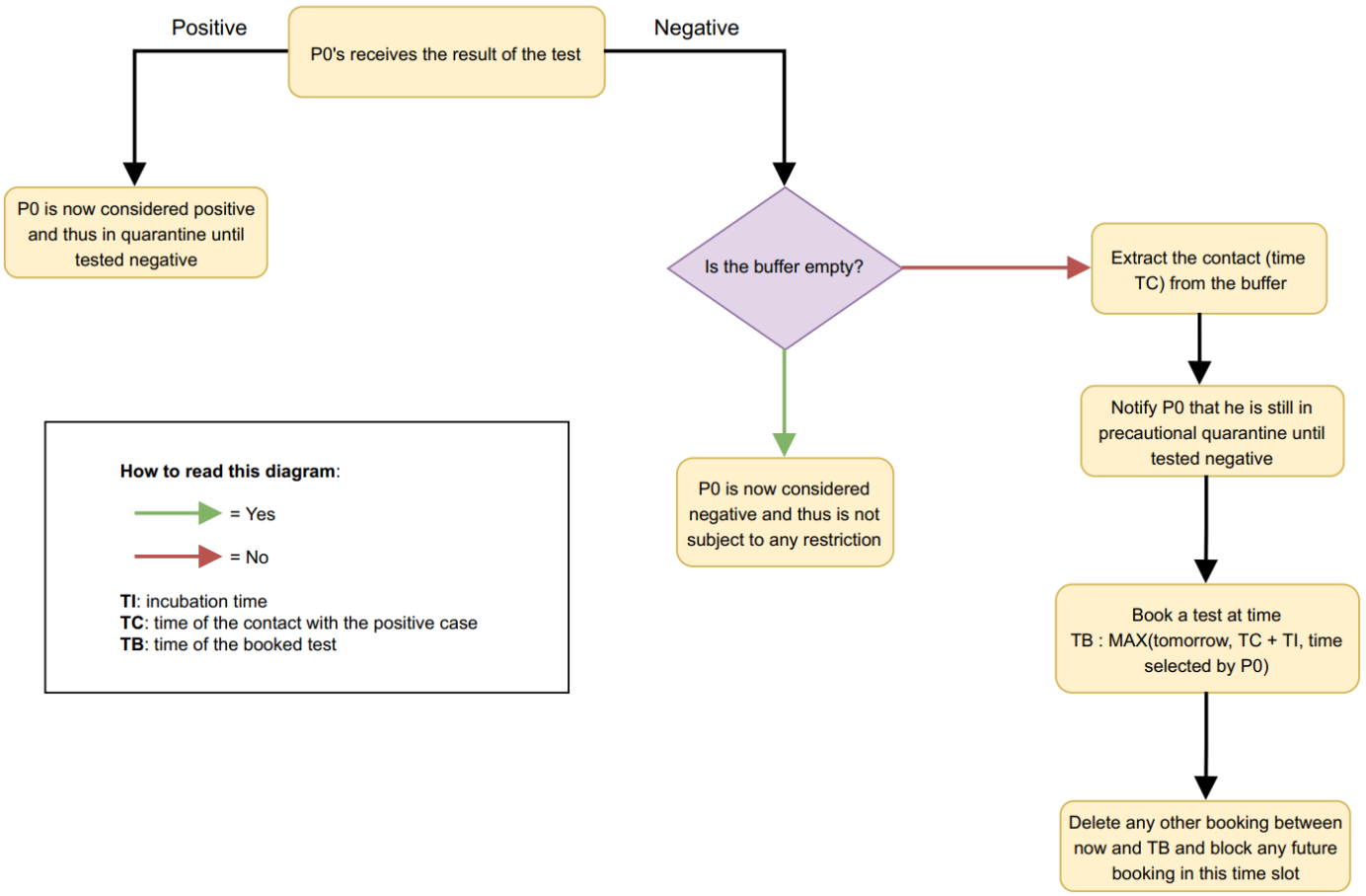
3.2 Booking a test

When someone in the "healthy" state has to book a test as a consequence of a contact happened at time T_c , the time selected for it will be the maximum between:

- $T_i + T_c$: this is because the virus may not be detectable before the incubation time T_i has passed, it makes no sense to get tested in this time window
- The day after the reception of the notification : this deals with the case in which the contact is discovered very late, so we overshot the date from the case above
- Date of choice : we leave the subject the freedom to choose a date after the previous two. Note that this means a longer precautional quarantine, but still gives more flexibility to the subject

In principle, users can also voluntarily book tests whenever they want, with the only exception of the time window corresponding to the precautionary quarantine. Moreover, if any test was previously booked for that period, the booking is canceled.

3.3 Handling a test result

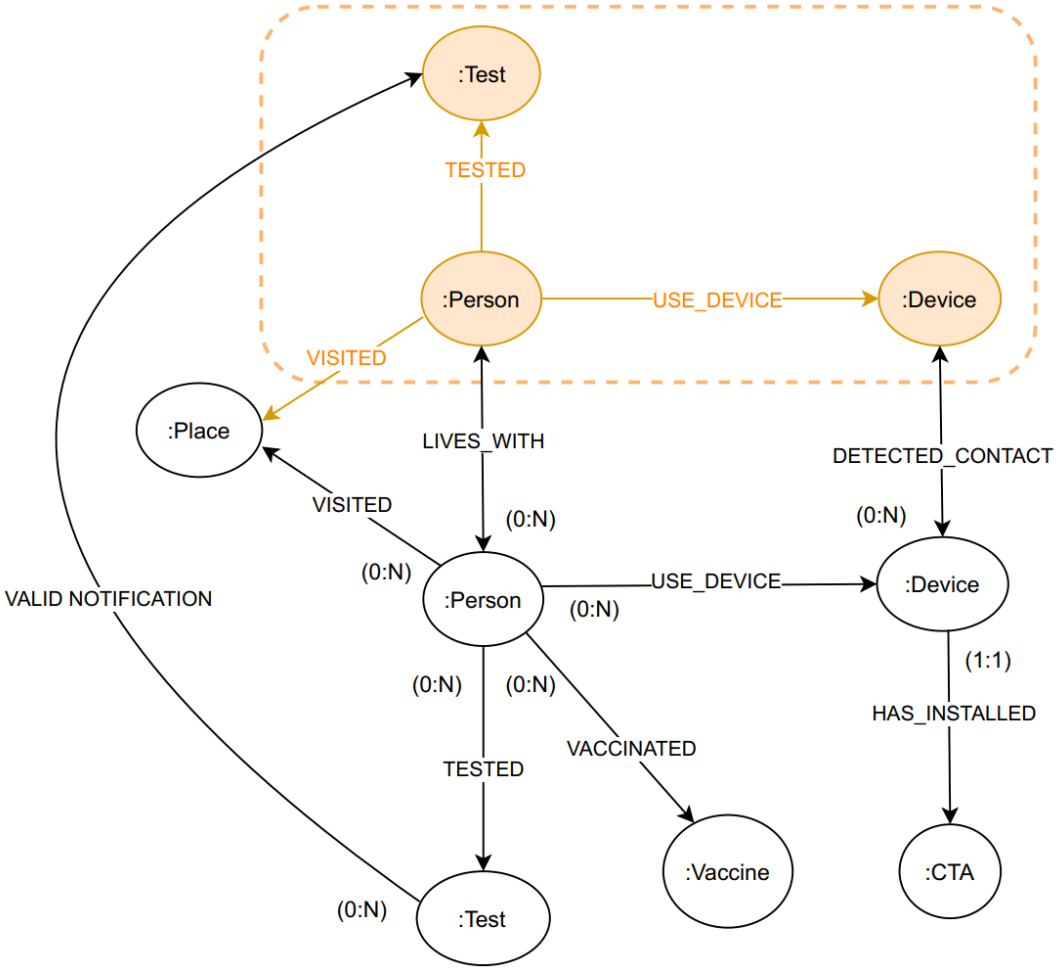


When the result of a test is published by the national health system, our contact tracing system will respond to a positive case by informing the subject that he/she is now in quarantine and sending notifications to the recent contacts. The booking of a future test to prove the recovery is considered to be a responsibility of the subject.

In case of a negative result, the system will check if any contact data is still in the buffer described previously. If that is the case, the subject will be forced to book a new test.

4 Conceptual level diagram

The diagram below shows the structure of our data. Please note that, to give the correct perspective on some of the relations, the entities in the coloured box are supposed to refer to a different person with respect to the others.



The attributes of our nodes are:

- Person: Fiscal Code (CF), name, last name, birthdate, phone, email, address
- Place: ID, name, address, phone, email
- Test: ID, date, place, result
- Vaccine: ID, date, place, pharmaceutical company
- Device: ID
- CTA (Contact Tracing App): name, state

The attributes of our relations are:

- VISITED: timestamp of arrival, timestamp of departure
- VACCINATED:
- VALID_NOTIFICATION: type of contact, date of contact, place of contact, date of notification

All the relationships not listed here do not have attributes attached.

5 Example dataset

We developed a python module to generate a dataset to exemplify our data structure. The user can set various parameters such as how many people, places and cities have to be generated, but also the probability that a test is positive/negative.
The basic information (people, places and cities) has been generated by the third-party open tool Mockaroo and then elaborated by us mainly using the python library pandas.

In order to obtain a number of nodes in the order of hundreds we set the parameters in the following way:

```
num_people = 50
num_places = 20
num_cities = 10
```

Moreover, we assume that a person will visit at most 10 places and that a test is positive with a probability of 20%, quite unrealistic, but useful for the goal of testing our queries.
The total number of nodes in our DB, so considering also tests, vaccines and devices, is in the order of hundreds as required in the assignment.
A particular database called db3.txt has been created in order to stress the third query, in fact we reduce the time interval in which contacts between devices and visits to a certain place can be generated. This increases the probability of satisfying the query.

The code of the dataset generator is available on GitHub. Please note that the repository also includes a requirements.txt file for the dependencies and a folder containing the final databases generated.

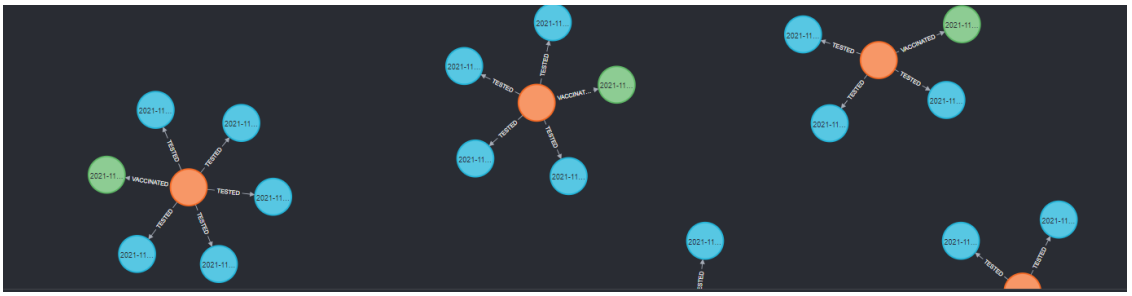
6 Queries and commands

This section provides the results of our study of the interaction with our data structure.

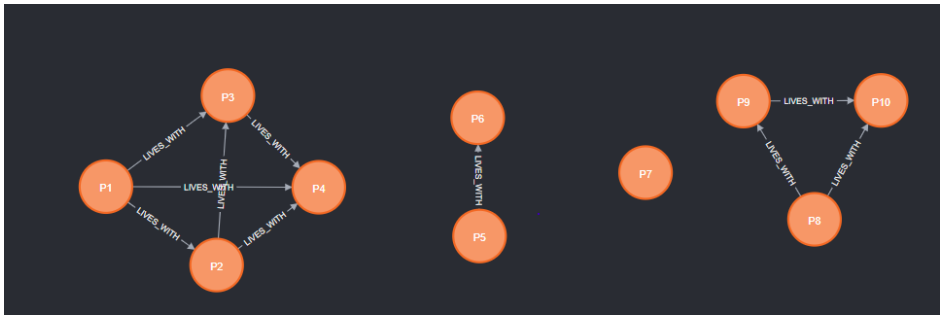
6.1 Test data for the queries:

All the following databases will be delivered together with the project to show that the results of the queries below are truthful and reproducible (beware that some of them compare a date with the current date, therefore, as the current date changes, the results will change as well).
Please note that all of the queries have also been tested on the example dataset described in the previous section.

- **Test_DB1:**
  This database is composed of a set of people, each one with a series of tests (ranging from a single one to 5, some positive and one negative), plus a vaccination. Both tests and vaccinations have associated dates as specified in the contact tracing database schema.

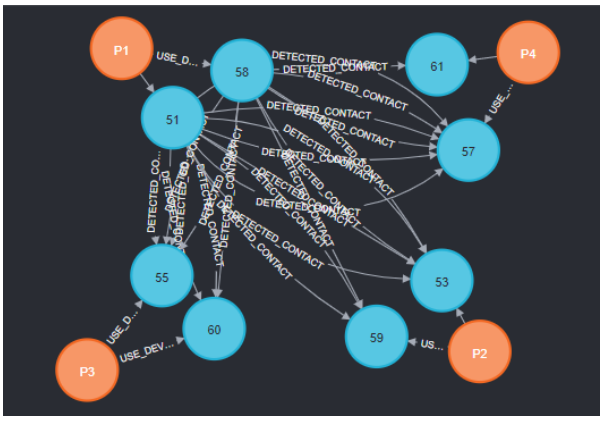


- **Test_DB2:**
  This database is composed of a set of people connected by relationships of the type LIVES_WITH. Relationships are implemented in a monodirectional way because, from what we found, it seems that in neo4j it's best practice to implement bidirectional relationships as monodirectional, and perform bidirectional queries instead

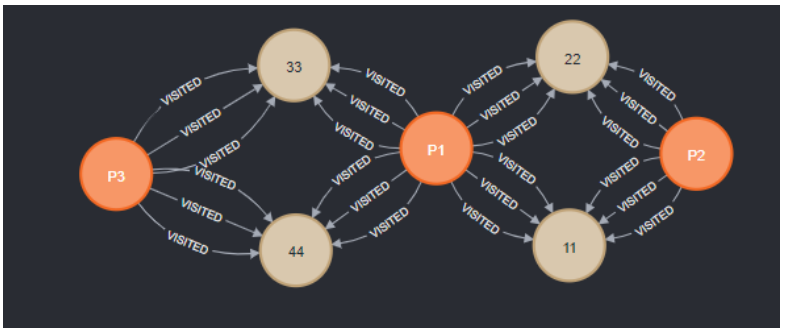


- **Test_DB3:**
  This database is composed of a set of four people, each with 2 devices. Person P1 has multiple contacts with all the devices belonging to other people. The contacts are registered as a DETECTED_CONTACT relationship, with a certain date. Multiple contacts can happen in a single day, a device could capture a contact that another didn't detect, and the time frame

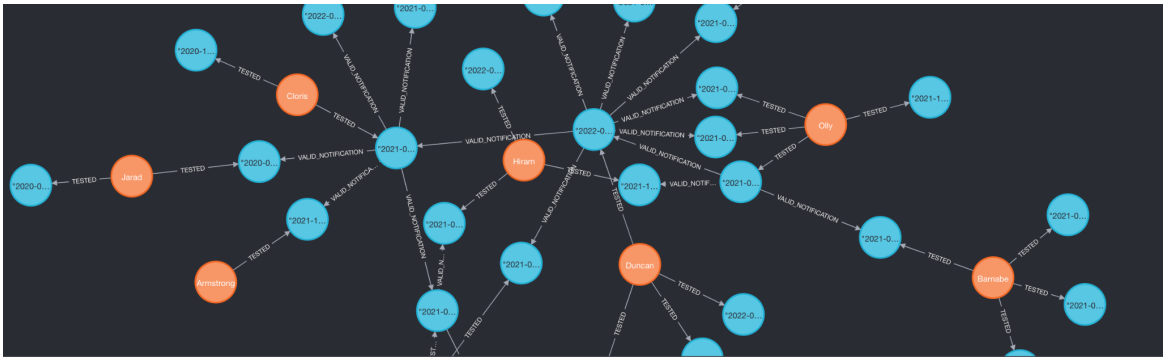
considered is larger than the one useful to perform the contact tracing. As for the previous test database, the DETECTED_CONTACT should be a bidirectional relationship, but the DB is built using the monodirectional version



- **Test_DB4:**
  This database contains 3 people and 4 places, and it is constructed around many different time intervals in which person P1 could have come across the other two people in those places (but one may have left before the other arrived, no contact there). Here as well the contact may be older than those of interest for the contact tracing.



- **Test_DB5:**
  This database is composed of a set of people and tests with TESTED and VALID_NOTIFICATION relationships linking them. There's a total of 20 VALID_NOTIFICATION relationship, the tests generated following these relationships are 8 positives and 12 negatives



6.2 Queries

1. **List of people that got infected after being vaccinated**

```
MATCH
(p:Person)-[:VACCINATED]->(va:Vaccine),(p:Person)-[:TESTED]->(tt:Test)
WHERE
va.date < tt.date AND
tt.result="positive"
RETURN DISTINCT p
```

Description

First the query matches each person with the vaccines he/she got and with the tests he/she was subjected to, then the result is filtered considering only those people for whom the date of the positive tests were subsequent to the date of vaccination. Finally we return the list of people satisfying those conditions.

Result on "Test_DB1"

In this case the people that satisfy the conditions should be P5,P6,P8,P9,P10 and P11, and those people are correctly returned by the query.

Result on the example database (section 5)


```
"p"
{"birthday":"1950-04-19","phoneNumber":"+48 730 565 6001","address":"9
127 Golf Course Court","CF":"JANEVE50S04H433L","mail":"jeverettd@newsv
ine.com","surname":"Everett","name":"Janos"}
{"birthday":"1937-06-20","phoneNumber":"+86 586 879 6128","address":"4
 School Place","CF":"DELFAI37C06P201A","mail":"dfairhead1f@naver.com",
"surname":"Fairhead","name":"Delmor"}
{"birthday":"1974-05-05","address":"185 West Point","phoneNumber":"+86
 209 529 6816","CF":"DONATT74S05D460S","mail":"dattridgex@blogspot.com
","surname":"Attridge","name":"Donnell"}
{"birthday":"1913-06-12","phoneNumber":"+351 382 991 6152","address":"
5 Pierstorff Lane","CF":"FABJOO13P06R920P","mail":"fjoost2y@jiathis.co
m","surname":"Joost","name":"Fabio"}
{"birthday":"1950-09-22","phoneNumber":"+86 738 780 1517","address":"4
66 Victoria Way","CF":"EWAJIR50E09B868M","mail":"ejiroutka3x@behance.n
et","surname":"Jiroutka","name":"Ewart"}
{"birthday":"1903-05-01","phoneNumber":"+86 571 886 2032","address":"9
5404 Eagle Crest Pass","CF":"DDEREG03A05H772D","mail":"dregardsoe38@go
ogle.es","surname":"Regardsoe","name":"Ddene"}
```

2. **Number of currently infected subjects**
   Number of people whose last test got a positive result and no further tests have been made to prove recovery.

```
MATCH (p:Person)
CALL {
    WITH p
    OPTIONAL MATCH (p)-[:TESTED]->(t:Test)
    WITH t
    ORDER BY t.date
    WITH last(collect(t)) AS most_recent
    WHERE most_recent.result="positive"
    RETURN most_recent
}
RETURN count(p)
```

Description

First the query retrieves all the people in the database, then it calls a procedure using the CALL clause. The CALL clause evaluates the subquery in order to return the most recent test of a person only if it's positive. In particular, the subquery, for each person, orders by date all the tests made. Then, from the collection of tests of each person, it extracts the most recent one and filters out those people for whom the test resulted to be negative. Therefore, we can use the result of the subquery to identify the current number of positive people.

Result on "Test_DB1"

In this case the people that satisfy the conditions should be P2,P4,P5,P6,P8,P9,P10 and P12, therefore the result should be 8, and so it is.

Result on the example database (section 5)

"count(p)"
9

3. **Contact Tracing queries**

Given the positive result for the test of a person, find out who came into contact with that person in the 14 days (our contact tracing window here). As stated in section 1, people can get into contact because:

1. they live together
2. they visited the same place at the same time
3. the contact tracing app registered a contact

Why this task was divided into three separate queries:
Considering that the application will have to deal with the notifications anyway (see notification buffering explained in the first part of the report), implementing all three parts of the query together while also trying to deal with order of relevance among relationships (LIVES_WITH more relevant than VISITED, as the contact will be more recent) and with different type of information they provide (neither the DETECTED_CONTACT nor the LIVES_WITH relationships consider the location, while VISITED does), would have been unnecessarily difficult. The application can easily do that while composing the notification of contact. The VALID_NOTIFICATION relationship will contain the information of the most relevant potential contagion extracted from these queries.

Contact tracing (1/3)

```
MATCH (p:Person)-[l:LIVES_WITH]-(p2:Person{CF:"P1"})
RETURN p,type(l) as type_of_contact
```

Description

The query matches a person, given its CF, with all the people he/she lives with, and returns them together with the type of relationship (which is needed to keep track of the reason of the potential contagion, the same goes for the two following queries). The query MATCH is bidirectional for the reason explained in the test database description.

Result on "Test_DB2"

We want to return the people who live with person P1, in the DB P1 lives with P2,P3 and P4. The query returns the correct result.

Result on the example database (section 5)

"p"	"type_of_contact"
{"birthday":"1901-04-02","phoneNumber":"+86 126 511 3017","address":"58 Iowa Place","CF":"PHIREX01S04C443H","mail":"prex2f@stumbleupon.com","surname":"Rex","name":"Philip"}	"LIVES_WITH"
{"birthday":"1913-06-12","phoneNumber":"+351 382 991 6152","address":"5 Pierstorff Lane","CF":"FABJOO13B06H874D","mail":"fjoost2y@jiathis.com","surname":"Joost","name":"Fabio"}	"LIVES_WITH"
{"birthday":"1945-04-10","phoneNumber":"+420 777 962 6946","address":"4754 Fairfield Center","CF":"JOREDM45C04S740A","mail":"jedmonston3f@disqus.com","surname":"Edmonston","name":"Jorgan"}	"LIVES_WITH"
{"birthday":"1912-08-19","phoneNumber":"+970 643 133 5902","address":"478 Eggendart Avenue","CF":"IGGSCH12B08P809A","mail":"ischubuserk@pbs.org","surname":"Schubuser","name":"Iggy"}	"LIVES_WITH"

Contact tracing (2/3)

```
MATCH
(p1:Person{CF:"P1"})-[u:USE_DEVICE]->(d1:Device)-[detected:DETECTED_CONTACT
]-(d2:Device)<-[u2:USE_DEVICE]-(p:Person)
WITH p,detected
ORDER BY detected.date DESC
WHERE duration.inDays(detected.date, date()).days <= 14
RETURN p, type(detected) as type_of_contact ,collect(detected.date)[..1] as
date_of_contact
```

Description:
The query matches a person given its CF with all the other people who came into contact with him/her, this type of contact is registered by the devices they use (they can also meet multiple times in a single day). All the contacts are then grouped by the person with whom the contact happened, and contacts are ordered by date in descending order . Contacts that took place more than 14 days previous are discarded (assume 14 days is our contact tracing window). The collect is used to retrieve, for each group (person) , the date of the last (= the most relevant) contact. The specific device that detected the contact is not relevant. Also in this case the query is bidirectional

Result on "Test_DB3":
Of the multiple contacts, also between different devices, the query correctly returns the date of the last contact for each person , regardless of the device that registered it

Result on the example database (section 5)

"p"	"type_of_contact"	"date_of_contact"
{"birthday":"1978-07-03","address":"40 West Center","phoneNumber":"+86 143 794 1627","CF":"OLYVOO78T07M825H","mail":"ovooght2d@earthlink.net","surname":"Vooght","name":"Olympie"}	"DETECTED_CONTACT"	["2021-11-11"]
{"birthday":"1976-02-03","phoneNumber":"+62 853 793 6131","address":"21925 Huxley Plaza","CF":"SHAOXB76H02A491P","mail":"soxbe3r@nationalgeographic.com","surname":"Oxbe","name":"Shanon"}	"DETECTED_CONTACT"	["2021-11-11"]
{"birthday":"1974-05-05","address":"185 West Point","phoneNumber":"+86 209 529 6816","CF":"DONATT74C05R672M","mail":"dattridgex@blogspot.com","surname":"Attridge","name":"Donnell"}	"DETECTED_CONTACT"	["2021-11-07"]
{"birthday":"1950-09-22","phoneNumber":"+86 738 780 1517","address":"466 Victoria Way","CF":"EWAJIR50M09M227L","mail":"ejiroutka3x@behance.net","surname":"Jiroutka","name":"Ewart"}	"DETECTED_CONTACT"	["2021-11-04"]

Contact tracing (3/3)

```
MATCH
(p:Person)-[v:VISITED]->(place:Place)<-[v2:VISITED]-(p2:Person{CF:"P1"})
WITH p,v,place
ORDER BY v.datetime_left DESC
WHERE
v.datetime_arrived<v2.datetime_left AND v2.datetime_arrived<v.datetime_left
AND
duration.inDays(datetime.truncate("day",v.datetime_left),  date()).days <=
14
RETURN p ,type(v) as typeOfContact,
collect(datetime.truncate("day",v.datetime_left))[..1] as dateOfContact
,collect(place)[..1] as placeOfContact
```

Description:

The query retrieves all the people who met a specific person, given his/her CF, in the last 14 days (assume 14 days is our contact tracing window). This time the contact takes place in a location where presence is registered, so we have access to the dates and times when the visitors arrived and left. Of course we also have the data regarding the place itself. First of all, the contacts are grouped by person, and ordered from the most to the least recent one. If the intervals in which the two people visited the same place do not overlap, there is no contact. Considering that people can meet multiple times in different places, the query returns only the date and place of the most recent contact.

The use of the datetime datatype for the VISITED relationship simplified a lot the condition for this query, relieving us from the need to manually implement checks over time intervals that span over multiple days and/or the transitions between consecutive days

Result on "Test_DB4"

A variety of cases were considered for each person in his/her contacts with person P1, the contact may have happened, may not have happened, may have happened over different dates and one of the two could have arrived in the particular location before the other person did. Some visits are more recent than 14 days and others are not. In the end, the expected result for the last contacts were the expected ones.

Result on the example database (section 5)

"p"	"typeOfContact"	"dateOfContact"	"placeOfContact"
{"birthday":"1967-02-04","phoneNumber":"+66 785 25 8 9622","address":"7 Caliangt Junction","CF":"LATM UM67D02H136D","mail":"lmumby44@mysql.com","surname":"Mumby","name":"Latia"}	"VISITED"	["2021-11-11T00:00:00+01:00"]	[{"placeID":19,"name":"Wordware","phoneNumber":"+3 89 638 547 1883","address":"3 Hazelcrest Crossing","mail":"asuattd@nhs.uk"}]
{"birthday":"1901-04-02","phoneNumber":"+86 126 51 1 3017","address":"58 Iowa Place","CF":"PHIREX01S0 4C443H","mail":"prex2f@stumbleupon.com","surname":"Rex","name":"Philip"}	"VISITED"	["2021-11-11T00:00:00+01:00"]	[{"placeID":19,"name":"Wordware","phoneNumber":"+3 89 638 547 1883","address":"3 Hazelcrest Crossing","mail":"asuattd@nhs.uk"}]

4. **People that have been infected more than once**
   People who contracted the virus at least twice, with a negative test in the middle to prove they had recovered

```
MATCH
(p:Person)-[:TESTED]->(t1:Test),
(p)-[:TESTED]->(t2:Test),
(p)-[:TESTED]->(t3:Test)
WHERE
t1.testID <> t2.testID AND
t1.result="positive" AND
t2.result="positive" AND
t3.result="negative" AND
t1.date < t3.date AND
t3.date< t2.date
RETURN DISTINCT p
```

Description:
First the query matches each person with the tests he/she was subjected to, then it looks for those people that tested negative after a previous positive test, and for whom a newer test proves them to have been infected again.

Result on "Test_DB1"
In this case the people that satisfy the conditions should be P5,P8,P10 and P12. The result is confirmed

Result on the example database (section 5)


```
╒═══════════════════════════════════════════════════════════════════════════════╕
│"p"                                                                            │
╞═══════════════════════════════════════════════════════════════════════════════╡
│{"birthday":"1963-01-24","phoneNumber":"+48 567 419 1087","address":"5│
│8631 Fremont Court","CF":"JAKSKA63L01C658E","mail":"jskaidg@usa.gov","│
│surname":"Skaid","name":"Jakie"}                                               │
└───────────────────────────────────────────────────────────────────────────────┘
```

5. **Ratio (infected people)/(people met) over the total population of the database**
   The VALID_NOTIFICATION relationship links the positive test of a person with the tests of people who had to be tested because of that positive result. We then see which of them later tested positive as well. The query aims to compute the (infected people)/(people met) ratio over the whole population of the database

```
MATCH (p:Person)
CALL{
  WITH p
  MATCH
(p)-[tt:TESTED]->(t:Test{result:"positive"})-[n1:VALID_NOTIFICATION]->(caus
ed_negatives:Test), (p1:Person)-[tt1:TESTED]->(caused_negatives)
  WHERE caused_negatives.result="negative"
  WITH count(caused_negatives) AS n_negatives
  RETURN n_negatives
 }
 WITH n_negatives,p
 CALL{
  WITH p
  MATCH
(p)-[tt:TESTED]->(t:Test{result:"positive"})-[n1:VALID_NOTIFICATION]->(caus
ed_positives:Test), (p1:Person)-[tt1:TESTED]->(caused_positives)
  WHERE caused_positives.result="positive"
  WITH count(caused_positives) AS n_positives
  RETURN n_positives
 }
 RETURN sum(n_positives)*1.0 / (sum(n_positives)+sum(n_negatives))
```

Description

The query retrieves, for each person with a positive test, all the tests that were generated for other people following the VALID_NOTIFICATION relationship. Thanks to the subqueries, we have access to all the tests of the people who came into contact with the specific positive subject. Among all these tests, using the two CALL clauses, we count the ones with positive results and those with negative results. We then sum the data we've just obtained to build the ratio on the whole population of the database.

Result on "Test_DB5"

In this case there are 20 VALID_NOTIFICATION relationships, which means that the test generated after the discovery of a new positives are 20 (in particular, 8 positives and 12 negatives). The result is confirmed, the ratio returned by the query is 0.4

Result on the example database (section 5)

"sum(n_positives)*1.0 / (sum(n_positives)+sum(n_negatives))"
0.2037037037037037

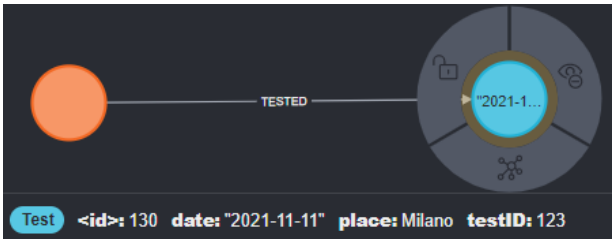
6.3 Commands

1. Booking of a test

```
MATCH (p:Person {CF: "RGHLNE77H66P688F"})
CREATE (p)-[t:TESTED]->(tt: Test  {testID: 123, date: date("2021-11-11"),
place:"Milano"})
```

Description

The command looks for the person who wants to book the test through the CF, and then it creates the test (with the desired data) and the relationship that connects the two of them

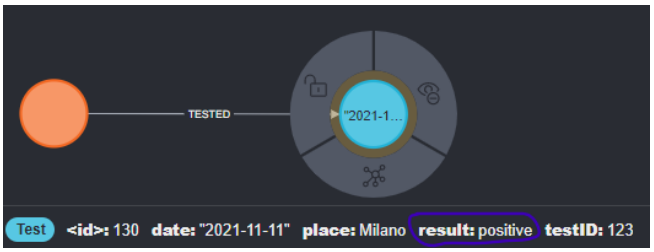


2. Update of a test that came out positive

```
MATCH (tt: Test {testID: 123})
SET tt.result = "positive"
```

Description

The command retrieves a specific test using its ID, and updates its result (sets it as “positive” in this case). After this happens (positive test update), a trigger should fire and start the contact tracing queries

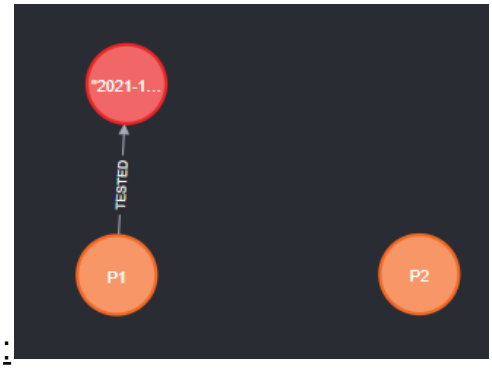


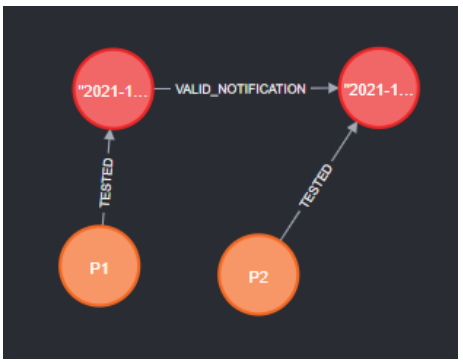
3. **Creation of a test due to a contact with a positive person**

```
MATCH (positive_test:Test {testID:1})
MATCH (p2:Person {CF:"P2"})
CREATE (positive_test)-
[:VALID_NOTIFICATION {type_of_contact:"VISITED", date_of_contact:
date("2021-11-08"), place_of_contact:45,
date_of_notification:positive_test.date}]
->(test_to_do: Test {testID:2, date: date("2021-11-16"),place:"Milano"})
CREATE (p2)-[:TESTED]->(test_to_do)
```

Description:

After a test is set to positive and the contact tracing process has ended, we find ourselves with the information about the contacts. Person P2 that met P1 (the one with the positive test) must be notified and will have to be tested. The command retrieves the positive test of P1, and creates the test to be done for person P2 by linking positive_test to test_to_do via the VALID_NOTIFICATION relationship, which holds the information about the potential contagion (in this case the two met in the Place with ID 45 on the 8/11/2021). Of course the test_to_do must also be linked with P2. In this example the new test is automatically booked in Milan on the 16/11/2021





7 Optional part: API server

For the optional part of the project, we decided to implement an API server to interact with our (remote) database as a web service.
After deciding to use python for this task, we selected the Flask framework for the external interface of our server and the neo4j module for python. Both the libraries are very well supported by the community.

The code, available in the "api" folder of our GitHub repository, has been decoupled in two submodules:

- **api.py** defines the resources, the endpoints and the methods that serve them. This module is mainly based on Flask and does not interact directly with the neo4j server.
- **database_interaction_library.py** is a backend module developed around the class "Graph", that models a proxy of the database aimed at masking the complexity of the interaction with the remote instance of neo4j. Each method invoked on the proxy creates a transaction that embeds the query or the command performed.

The endpoints offered by the server are:

- `/positives` returns the current count of positive people
- `/contact_tracing` given the id of a newly discovered positive, it returns the people to notify to perform contact tracing
- `/infected_after_vaccinated` returns the list of people that got the virus after being vaccinated
- `/infected_multiple_times` returns the list of people that got the virus multiple times
- `/diffusion_tracing` refer to query n.5
- `/register_test_result` let the user set the result of a given test on the database
- `/check_positive` given the id of a person, it returns true if that person is currently positive to the virus, and false otherwise.

Moreover, the server offers an `/helloworld` endpoint to test the interaction with neo4j, and a `/clear` method to clear the database for debugging.
Please note that the methods above represent just a subset of the functionalities we presented in the previous sections of this document. This implementation is just to be considered a starting point to develop a service that could represent the backend for any kind of user application.

We tested this module with the "requests" python package. The script "client.py", available alongside the code of the server, contains a few lines of code as a template.

For future developments we leverage the possibility to secure some of the endpoints with tokens, insert limitations to the rate of requests by the user and offer a high level interface for testing and documenting the api with tools like Swagger.

8 User guide

This final section contains some notes to get our system up and running.
All the code can be downloaded by our GitHub repository here.

8.1 Check dependencies

We tried to limit the dependencies of our code as much as possible. Given that the whole system has been developed in python, we just require an environment with the requirements listed in the "requirements.txt" file. All the packets listed there should be easily available with the "pip" packet manager.

8.2 Generating the database

The example database (section 5) can be directly replicated with the commands listed in the "DB.txt" file. To generate a new version, the script "generator.py" can be fine-tuned by adjusting the parameters.
To generate a new txt file, the user can redirect the standard output to a file of choice (python generator.py > output.txt).

8.3 Running the server

In order to get the API server up and running, the user just needs to run the "api.py" script in the previously prepared environment.
Once run, the command line interface will display the port the calls should refer to (typically 5000).
Please note that the delivered code is supposed to work with unauthenticated neo4j instances (this can be easily set in the neo4j.conf file, located in the installation folder of the neo4j server).
This is just to ease the delivery, we have already tested successfully authenticated connections.