POLITECNICO
MILANO 1863

PANDEMIC TRACKING SYSTEM FOR BIG DATA ANALYSIS WITH GRAPH APPROACH

Systems and Methods for Big and Unstructured Data

Nikita Rozov	**(10788621)**
Matteo Citterio	**(10620055)**
Sergei Nabatov	**(10788620)**
Federico Caspani	**(10622658)**
Oleksandr Schuchklyi	**(10788614)**

INDEX

GitHub Repository of the project https:/github.com/serynabatov/unstructured_data_topic_1

PROBLEM SPECIFICATION

The aim of the project is to create a system that manages all the useful information for tracking and monitoring the spread of the virus.

The system we developed is able to represent and store people's personal information, their current health situation and the way they interact with each other:

- they can live in the same place and be constantly in contact
- they can meet in a certain place at a certain time
- they can casually cross around (and the encounter is saved in coordinates as a place itself)

The database handles also the sources of information (that could be Tracing Apps, Manual Registration, etc.), useful for understanding the efficiency of a specific information channel.

The modeling phase passed through two different stages:

1. creation of an **E-R Model**
2. conversion of the **E-R Model** to a **Graph Model**

that will be specified later in a better way.

HYPOTHESIS AND ASSUMPTIONS

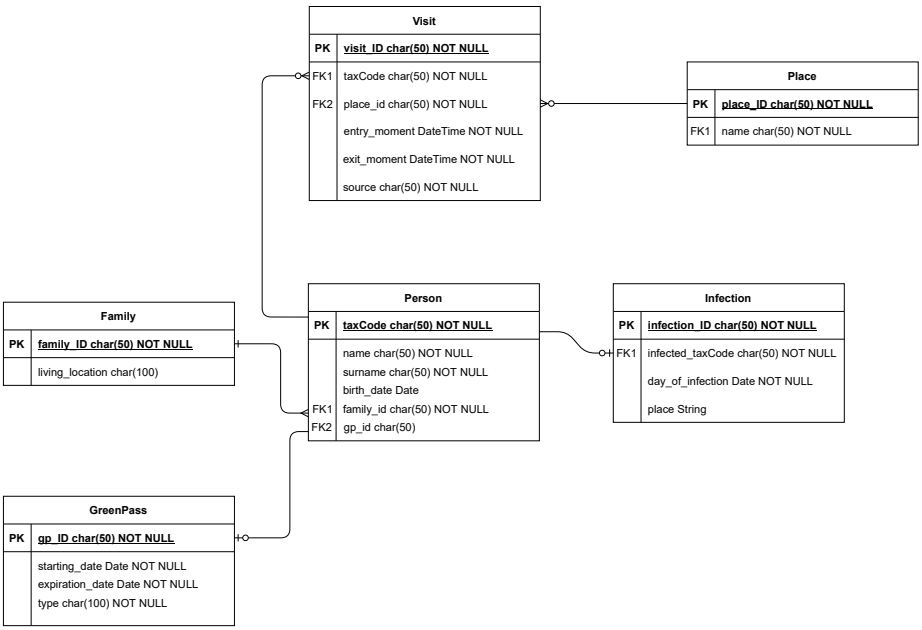
Several hypothesis have been made during the modeling part of the problem:

- *Quarantine* concept is not explicitly modeled.
  It's possible to infer it by searching if a person has been in contact with an infected.

- We made the assumption that whoever lives in the same place is a *family*.
  This hypothesis can lead to inconsistencies in surnames inside families but this is also a problem that could be easily solved by adding a "family" attribute (that has an unique and incremental value for each existing family) on each Person node or putting specific relationships to unify them, if it was a necessary and unavoidable step for the scope of pandemic tracking.

- Since It's impossible to know where an Infection took place, we assume that it happened in one of the places the Person visited in the last 14 days (duration time of the "contact" relationships), avoiding a direct arc between Infection and Place that is not real-case applicable.

E-R DIAGRAM



- **Person**: the person class of the model.
  It contains its personal data, a reference to its Green Pass, if present, and its “family”.

- **Family**: an aggregation table containing information about families, intended as per hypothesis.

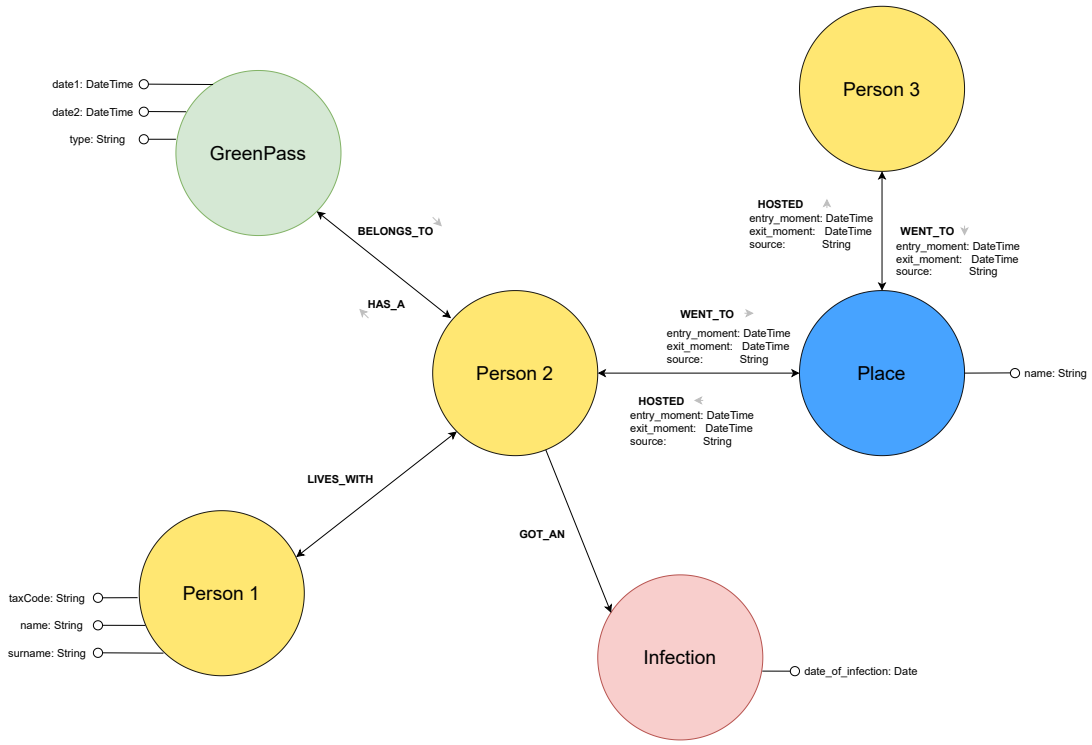
- **GreenPass**: contains the dates in between the pass is valid and the type that specifies if It comes from a *specific* v*accination* or a t*est*.

- **Infection**: contains the infected reference, the date of the infection and (eventually) the place of contagion.

- **Place**: represents all the places of contact between people.

- **Visit**: an association table for expressing a single visit.
  It contains the source of the data and the timestamp of the entrance and the exit one.

GRAPH DIAGRAM



The process of ER-to-Graph conversion introduced these substantial changes:

- The Visit table became bi-directional relationship (**WENT_TO**, **HOSTED**) between Person and Place nodes.
  These relationships contain all the attributes that were on the Visit table before.

- The Family table has been substituted with a set of **LIVES_WITH** relationships between every component and all the others.

- taxCode, from the Person table, is the only PK that persisted (?).

DATASET DESCRIPTION

The database has been populated through a Python Script, available at the following link:

https://github.com/serynabatov/unstructured_data_topic_1/blob/master/backend/populator.py

The script accesses to the Neo4j Sandbox team-shared database and calls a sequence of functions that create coherent data:

- ***compute_people()***
  creates a certain number of people with their personal data.

- ***compute_gp(taxCodes)***
  randomizes an amount of Green Passes and attaches them to someone.

- ***compute_families(taxCodes)***
  takes groups of people (with a cardinality between 2 to 5) and creates families.

- ***compute_infections(taxCodes)***
  randomizes the creation of infections and attaches them to someone and some places.

- ***compute_places_and_connections(taxCodes)***
  randomizes visits between people, considering actual places and coherent dates.

The script is dependent to a specific amount of generated data, that can be easily changed to allow bigger database populations.

The parameters used for generating randomized situations that compose the actual dataset are:

```
PEOPLE = 100
MIN_FAMILIES = 7
MAX_FAMILIES = 15
MIN_FAMILY_MEMBERS = 2
MAX_FAMILY_MEMBERS = 5
MIN_CONNECTIONS = 40
MAX_CONNECTIONS = 60
MIN_GPS = 30
MAX_GPS = 50
MIN_INFECTIONS = 10
MAX_INFECTIONS = 15
```

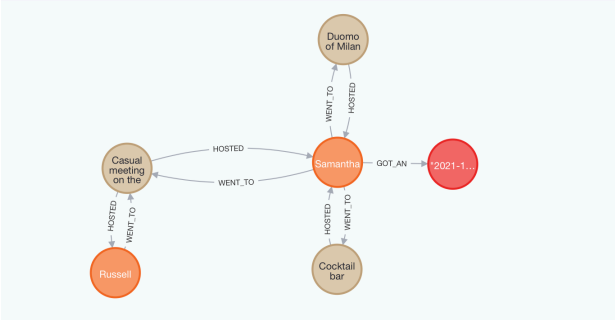
Below, we present some queries and commands that are inspired to the ones that can be useful in real applications context.

QUERIES

- *Show quarantined people that went to a place while being infected.*

```
MATCH (a:Place)-[c:HOSTED]→(p:Person)-[:GOT_AN]→(i:Infection)
WHERE c.entry_moment ⩾ i.date_of_infection
RETURN p,a
```

Result: in this case, Samantha met Russell and went to a bar and Duomo while she should've been in quarantine.



- *Display the top K places with most infected people, and the number of infected for each of these places.*

```
MATCH (p:Place)-[:HOSTED]→(:Person)-[r:GOT_AN]→(:Infection)
WITH p, COUNT(r) AS cnt ORDER BY cnt desc
RETURN collect(p.name) as names, cnt
LIMIT K
```

Explanation: K must be come as a user's choice and dynamically inserted.
The query counts, for each place, the infected customers and collects places' names in decreasing order depending on the count of before.
Result: the query forces the displayed elements to be, in this case, only the first four.

names
["Casual meeting on the street at datetime('2021-12-23T19:3')", "Casual meeting on the street at datetime('2022-10-28T9:23')", "Duomo of Milan", "Cocktail bar"]

- *Show a table containing each vaccination type and the number of people that have done it.*

```
MATCH (gp: GreenPass)
WITH  gp.type as Vaccination_type, COUNT(gp) as number
RETURN COLLECT(Vaccination_type) as type_of_vaccine, number  ORDER BY number DESC
```

Result: the query displays a no-rows-limit table regarding the “coverage” of each type of vaccine, including the (still valid) tests. Vaccination types with the same counts are merged into the same tuple through the *COLLECT* operator.

	type_of_vaccine	number
1	["Moderna Vaccination"]	6
2	["Astrazeneca Vaccination"]	5
3	["Covid-19 Test", "Johnson&Johnson Vaccination"]	4
4	["Sputnik V Vaccination", "Pfizer Vaccination"]	3

- *Show the daily infected/healthy ratio.*

```
MATCH (pInfected:Person)-[:GOT_AN]→(:Infection)
WITH COUNT(pInfected) as infected
MATCH (pHealthy:Person)
WHERE EXISTS ((pHealthy)-[:GOT_AN]→(:Infection)) = FALSE
WITH infected, COUNT(pHealthy) AS healthy
RETURN infected, healthy, (infected / toFloat(healthy)) AS dailyRatio
```

Explanation: the query counts the Person nodes with a linked Infection node and, sequentially, the number of Person nodes with no infections.
Then, it displays the ratio between these two values, in float format.

Result:

infected	healthy	dailyRatio
9	47	0.19148936170212766

- *Show the most visited day of the most visited place.*

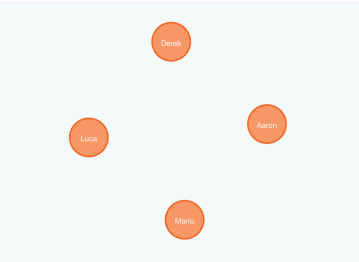
```
MATCH (:Person)-[r:WENT_TO]→(p:Place)
WITH COUNT(r) AS num, p
ORDER BY num DESC LIMIT 1

MATCH (a:Person)-[r1:WENT_TO]→(p)←[r2:WENT_TO]-(b:Person)
WHERE r1.exit_moment.epochSeconds > r2.entry_moment.epochSeconds AND r1.entry_moment < r2.exit_moment
WITH COUNT(a)+1 AS number, p.name AS place, r1, date(r1.entry_moment) AS date, a
RETURN date, number, place, collect(a) ORDER BY number DESC LIMIT 1
```

Explanation: this query searches for the Place with the biggest amount of customers, then searches the day the place is busiest (basing comparisons on timestamps in seconds).
(**N.B.:** the number of people in a day is taken as COUNT(a) + 1 because the beginning of the pattern matching is not outside of the “a” set!)

Result: the query returns specific **date**, **place**, **number** of people and a list (in JSON format) of the Person nodes that compose that **number** value.

	date	number	place
1	"2022-12-11"	4	"Stadium"



- *Show the daily stamp (infected/tested ratio).*

```
MATCH (pInfected:Person)-[:GOT_AN]→(i:Infection)
WHERE date.truncate('day', i.date_of_infection) = date()
WITH COUNT(pInfected) as infected
MATCH (pTested:Person)-[:HAS_A]→(gp:GreenPass)
WHERE gp.type = "Covid-19 Test" AND date.truncate('day', gp.date1) = date()
WITH infected, COUNT(pTested) AS tested
RETURN infected, tested, (infected / toFloat(tested)) AS dailyRatio
```

Explanation: same concept of the fourth query, but different meaning.
We decided to put it because of the highlight and deep analytical purpose it has during the pandemic period.

Result:

infected	tested	dailyRatio
1	2	0.5

COMMANDS

- *Set positive a list of already registered people*

```
MATCH (p1: Person {name:"N1",taxCode:"T1",Surname:"S1"}),
      (p2: Person {name:"N2",taxCode:"T2",Surname:"S2"}), …
CREATE
(p1)-[:GOT_AN]→
    (:Infection{date_of_infection:"2021-12-12T13:16:54.414000000Z"}),
 (p2)-[:GOT_AN]→
    (:Infection{date_of_infection:"2021-12-12T13:16:54.415000000Z"}),
…
```

Explanation: The command searches for new infected's Person nodes inside the dataset and creates an Infection node for each one of them.
Since the people's information can be inserted, for example, through an UI, the date_of_infection source is irrelevant: we can assume that the data comes immediately (or at least in the same day) and compile all of them with today's date, but we can also think about an user inserting it…

- *Attach a Green Pass with activation date D1, expiration date D2 and type T to a Person with taxCode TC and, eventually, delete its Infection.*

```
MATCH (a:Person)
WHERE a.taxCode= TC
CREATE (a)-[:HAS_A]→(gp:GreenPass {date1: D1, date2: D2, type: T})-[:BELONGS_TO]→(a)
WITH 1 as dummy
MATCH (a)-[r:GOT_AN]→(i:Infection)
WHERE a.taxCode= TC
DETACH DELETE i
```

Explanation: the command gets the requested Person node, creates the GreenPass node, links the two nodes with a double-sided relationship and tries to delete Infections that are reachable from that Person and not valid anymore.

- *Data Cleaning command: delete expired Green Passes and visits (after 14 days).*

```
MATCH (a)-[r:HAS_A|BELONGS_TO|WENT_TO|HOSTED]→(p)
WHERE (datetime().epochSeconds-datetime(r.exit_moment).epochSeconds ⩾ 86400*14)
    OR
      (datetime() > p.date2 OR datetime() > a.date2)
DELETE r
WITH 1 AS dummy
MATCH (gp:GreenPass) WHERE NOT EXISTS( (gp)←[:HAS_A]-(:Person) )
DELETE gp
```

Explanation: at first, the commands deletes all expired relationships (checks indiscriminately for Person <-> GreenPass and Person <-> Place ones). Depending on the type of matched relationship, analyzes it separately: in the first part of the WHERE clause, looks for expired “visit” connections (since they own the exit_moment attribute) and in the second part it searches and eliminates expired “Green Pass holding” arcs. Finally, it deletes GreenPass nodes ended up being disconnected by the arcs’ cleaning operation, because no longer valid.

APPLICATION

The main idea behind the application our team have developed is about giving a support in running the queries and commands that are also described in the **QUERIES** and **COMMANDS** chapters.

The task has been divided in two main parts:

1. ***BACKEND SECTION***

   This part of the work laid to the logic foundations of the application itself, finally producing a complete backend application that is able to query the Neo4J database and provide the answer on-demand, throughout Its API.
   A more detailed exposition about the backend stuff, and how to run and properly use it is available at this link:
   https://github.com/serynabatov/unstructured_data_topic_1/tree/master/backend

2. ***FRONTEND SECTION***

   The data provided by the Backend Application are then processed and displayed on the Graphic User Interface of our mobile application.
   You can find more details on how the GUI was designed and what are the functionalities that our application offers here:
   https://github.com/serynabatov/unstructured_data_topic_1/tree/master/mobile