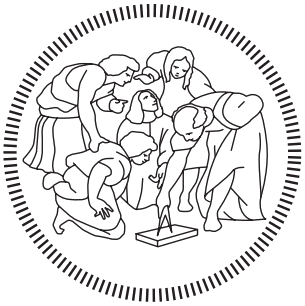
AY 2021-2022



POLITECNICO
MILANO 1863

SMBUD Project

Covid Tracing-Oriented Neo4j Database

Pablo Giaccaglia - 10626428
Santi Pier Pistone - 10867402
Salvatore Cassata - 10560790
Stefano Vighini - 10622788
Zhitao He - 10763530

Master of Science in
Computer Science & Engineering

December 24, 2021

Contents

1 Introduction

Considering the scenario in which there's the need to build a system for managing the **COVID-19 pandemic** in a specific country, our project focuses on the data perspective level. This is why we designed and implemented a **Neo4j** [1] data structure to face the need of contact tracing functionality, to monitor the viral diffusion.

2 Specification & Hypothesis

The usage of this application involves the following actors :

- **normal citizens**, who get in touch with each other
- **workers**, who need to trace part of the contacts

and the following devices :

- a **server** storing a database (which could evolve in a distributed system, with every country having its own database)
- **devices** such as smartphones or wearable, which can use Bluetooth and store a simple script, and have an internet connection (or can communicate to a device with it).

The purpose of this project is to track the spread of **COVID-19**. The first design choice concerning the database records it's indeed about tracking. There are mainly 2 ways to do it:

- **automatically** (with smartphones or wearables): we suppose that, depending on the country, we might have different applications or technologies that keep track of the fact that people occasionally are in the same place at the same time, so the system is designed to also store contact tracing app data. This type of contact tracking is represented by the **MET** relationship.
- **manually** (explicit data collection): data coming from specific locations, such as restaurants, theaters or hospitals, that explicitly record people's presence, asking them personal information. This type of contact tracking is represented by the **WENT TO** relationship

We could have used the **MET** relationship in both cases, but this division leads to a less redundant database. For example, if 100 people go in the same place on a certain date, 100^2 relationships between people should be stored if there wasn't a **WENT TO** relationship. In this way instead, only 100 relationships are needed, resulting in a way lighter database.

We also chose to have **Vaccines** and **Tests** as entities and to store the information about date, number, vaccination city and result in the corresponding relationships. This choice doesn't really make the difference, as we could also have kept this information in the entities. Anyways, for future use there could be the need to store additional information in vaccines and tests, so we made this choice to avoid redundancy.

The relationships' names **RECEIVED** and **TOOK** have to be interpreted along with Vaccine and Test, as this DB is only for pandemic purposes, so these have an unique meaning.
We also numbered the vaccines taken in the RECEIVED relationship to have quicker queries, as it could be an useful information to extract, in addition to being a much more efficient design choice than determining it through queries.

3 Data Model (ER Diagram)

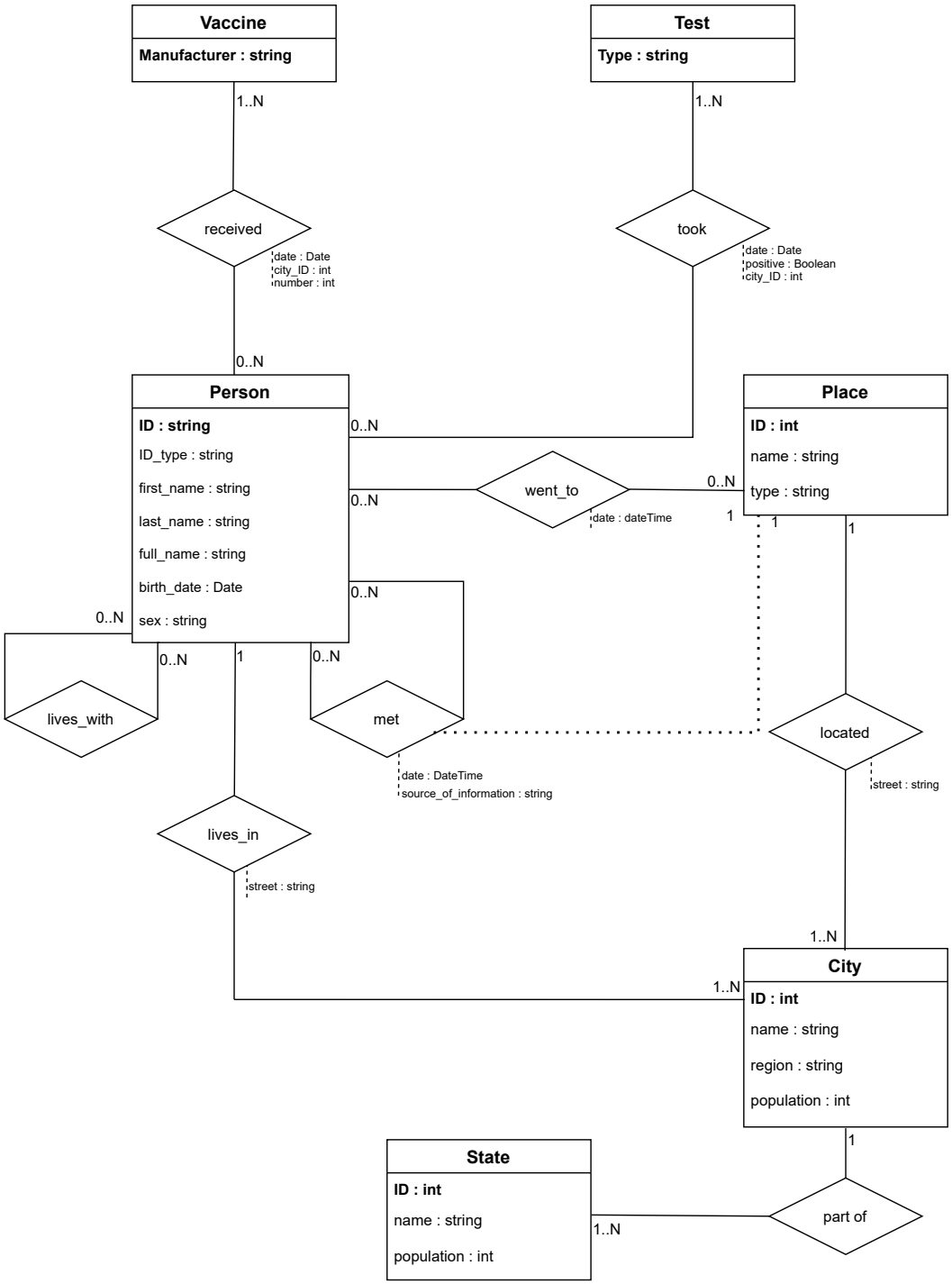


Figure 1: ER Diagram

The model's entities are:

- **People**: each person has a corresponding node in the database with its own features (attributes) which are :
  - Personal Recognition Document ID
  - Personal Recognition Document Type
  - First Name
  - Last Name
  - Full Name
  - Birth Date
  - Sex
- **Covid vaccines**: each vaccine manufacturer has a corresponding node in the database with its own features (attributes), which are :
  - Manufacturer Name
- **Covid tests**: each test typology has a corresponding node in the database with its own features (attributes), which are :
  - Test Type {Molecular test/Antigen test/Antibody test}
- **Places**: each place has a corresponding node in the database with its own features (attributes), which are :
  - Place ID
  - Name
  - Type {restaurant/cafe/theater/cinema/hospital}
- **Cities**: each city has a corresponding node in the database with its own features (attributes), which are :
  - City ID
  - Name
  - Region
  - Population amount
- **States**: each state has a corresponding node in the database with its own features (attributes), which are :
  - State ID
  - Name
  - Population amount

The data retrieval process to populate the database with these entities and further clarifications is explained in section 4. Along with these nodes, the Neo4j database is composed by several relationships, illustrated in the following section and in section 3, where there is a brief in-depth explanation for each one.

3.1 Relationships

Several types of relationships are stored in the system. The following sections illustrate their visual representation taken from the **Neo4j Bloom** [2] graph exploration application followed by a brief explanation.
It is important to point out that **"undirected"** relationships are intended to be **symmetrical** (e.g "MET RELATIONSHIP" : for every P1 that met P2, also P2 met P1), even though Neo4j doesn't explicitly allows to have them, but allows to have the notion of undirected edges at query time[3].

3.1.1 RECEIVED

This directed relationship connects a **Person** node to a **Covid Vaccine**. Multiple RECEIVED relationships can exists for the same person, indicating the fact that he/she has received multiple Covid vaccine doses of the same manufacturer. In our sample dataset each **Person** node is associated with at most 2 *"RECEIVED"* relationships.

The connection stores the following attributes:

- Date of the vaccination
  (**ranging from 2021-01-05 to 2021-11-7**)
- The city of the vaccination
- The number of vaccination dose.

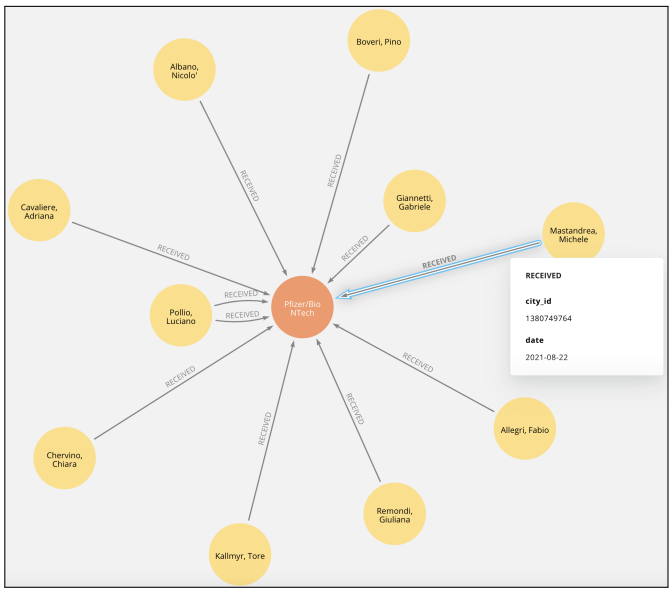


Figure 2: *"RECEIVED"* Relationship Visualization

3.1.2 TOOK

This directed relationship connects a **Person** node to a **Covid Test** node. Multiple TOOK relationship can exists for the same person, indicating the fact that he/she has done multiple tests. In our sample dataset each **Person** node is associated with at most 4 *"TOOK"* relationships.

The connection stores the following attributes:

- Date of the test
  (**ranging from 2020-03-10 to 2021-11-07**)
- The city where the test has been taken
- The outcome of the test {**Positive/Negative**}

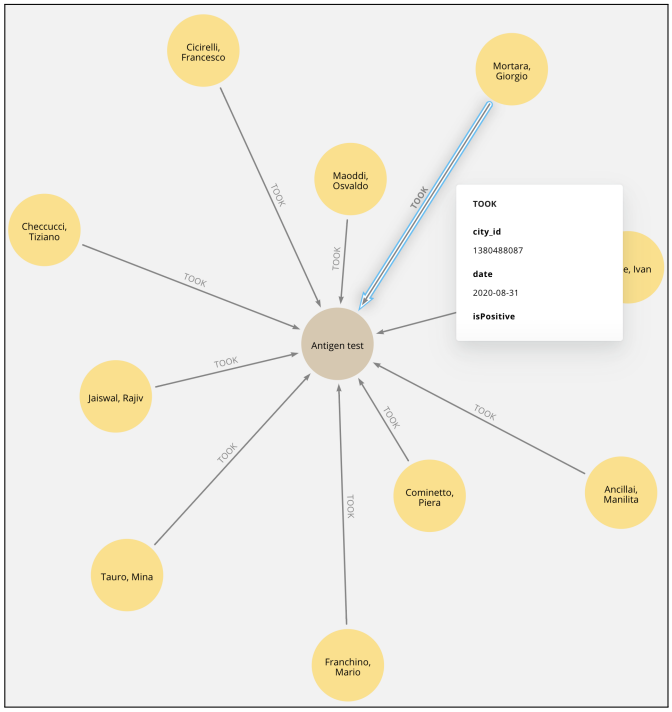


Figure 3: *"TOOK"* Relationship Visualization

3.1.3 WENT TO

This directed relationship connects a **Person** node to a **Place** node. If the database contains such information it means that there has been an explicit manual registration from specific locations, such as restaurants, theaters or hospitals, that explicitly record people's presence, asking them for their names. In our sample dataset, each **Person** node is associated with exactly 50 *"WENT TO"* relationships.

The connection stores the following attributes:

- Date of the registration (**ranging from 2020-3-10 to 2021-11-07**)

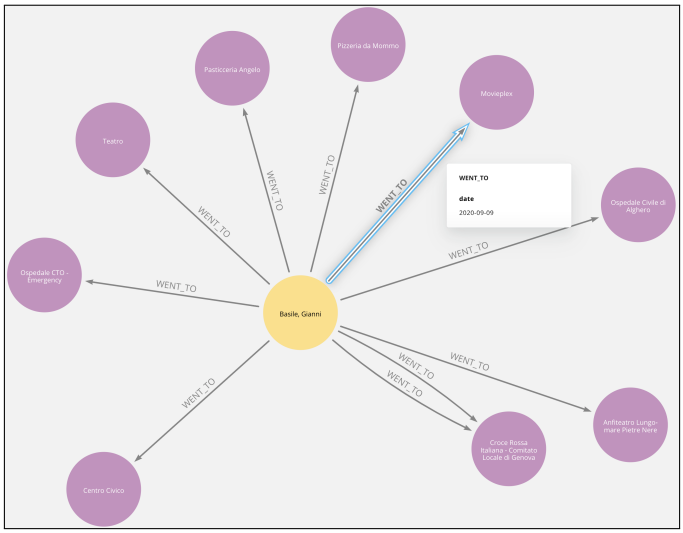


Figure 4: *"WENT TO"* Relationship Visualization

3.1.4 LIVES WITH

This undirected relationship connects two **Person** nodes. Every pair of people who belong to the same family or live in the same house are considered **always** in touch. The assumption is that we don't want the tracing system to be recording every second contact of people living together, so they are in contact by default. In our sample dataset each **Person** node is associated with at most 6 *"LIVES WITH"* relationships. The connection doesn't store attributes.

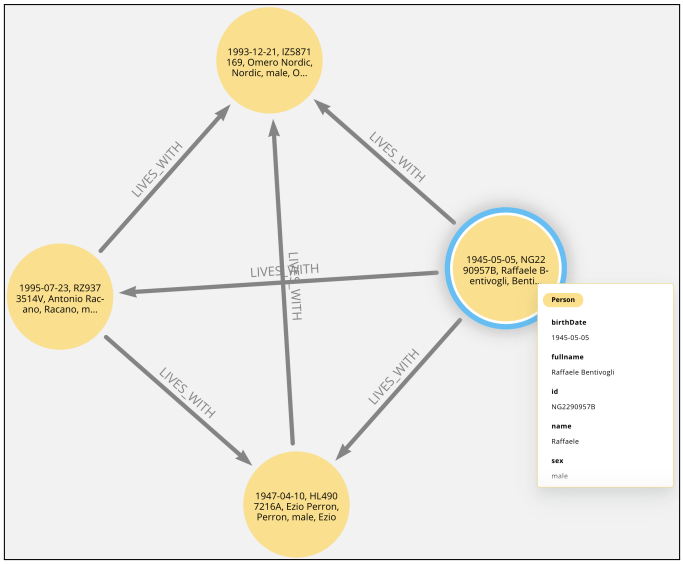


Figure 5: *"LIVES WITH"* Relationship Visualization

3.1.5 LIVES IN

This directed relationship connects a **Person** node to a **City** node. We assume that each **Person** node is associated with exactly one
"LIVES IN" relationship.

The connection stores the following attributes:

- Address of the residence

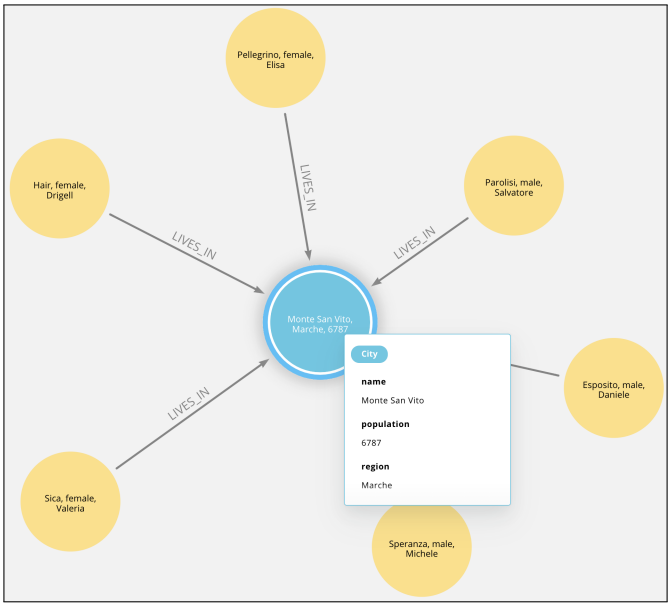


Figure 6: *"LIVES IN"* Relationship Visualization

3.1.6 MET

This undirected relationship connects two **Person** nodes. This information comes from contact tracing app or technologies that keep track of the fact that people occasionally are in the same place at the same time. We assume that the technologies providing this information don't keep track of the location of the contact. In our sample dataset each **Person** node is associated with a number of *"MET* relationships ranging from 10 to 50 (inclusive).

The connection stores the following attributes:

- Date of the contact (**ranging from 2020-03-10 to 2021-11-07**)
- Source of tracking info {e.g **Smartphone/Smartwatch/Tracking Wearable**}

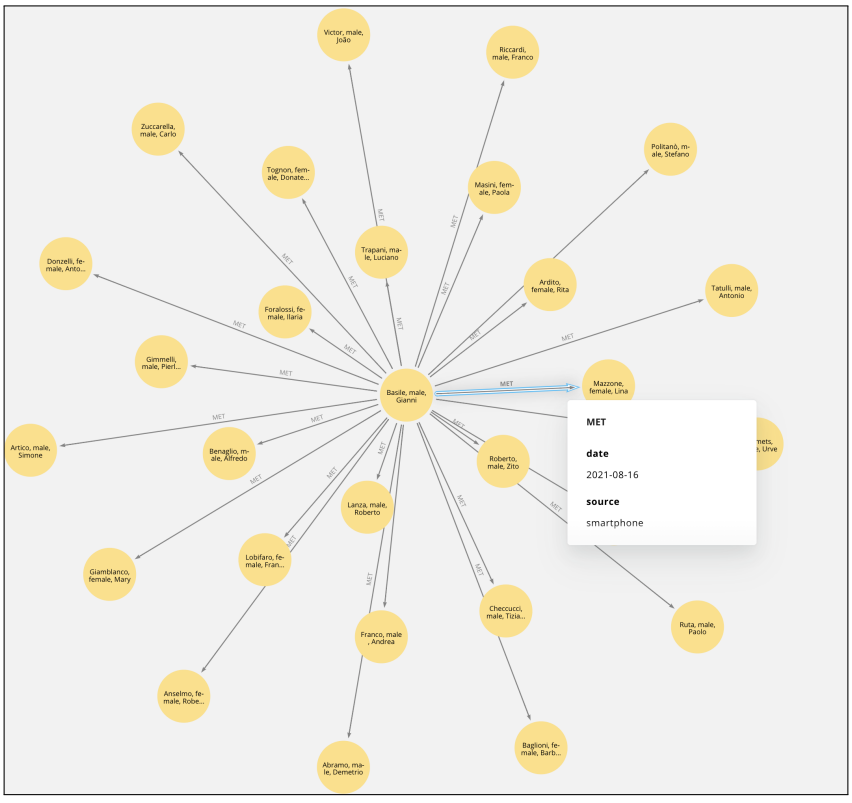


Figure 7: *"MET"* Relationship Visualization

3.1.7 LOCATED

This directed relationship connects a **Place** node to a **City** node. We assume that each **Place** node is associated with exactly one *"LOCATED"* relationship.

The connection stores the following attributes:

- Address of the location

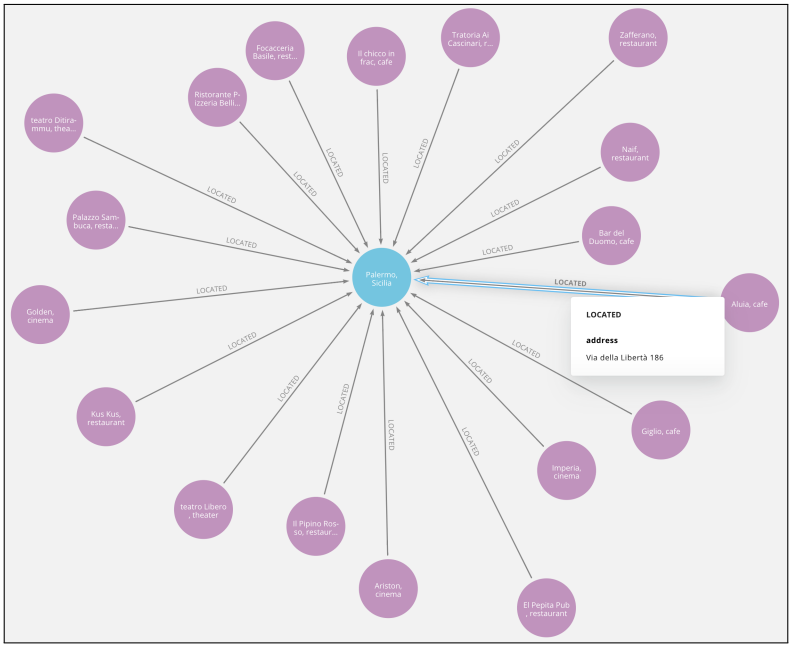


Figure 8: *"LOCATED"* Relationship Visualization

3.1.8 PART OF

This directed relationship connects a **City** node to a **State** node. We assume that each **City** node is associated with exactly one *"PART OF"* relationship. The connection doesn't store attributes.

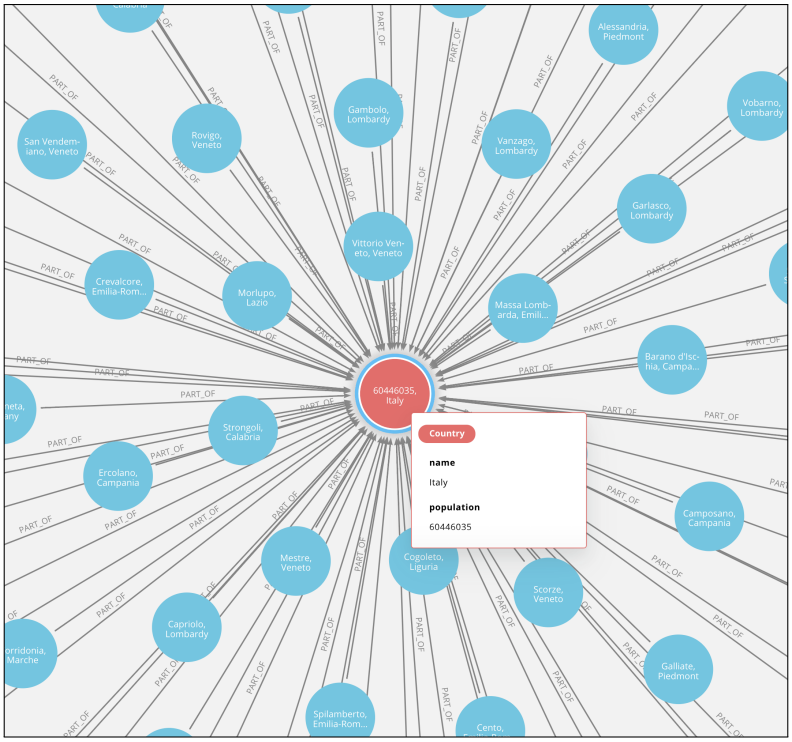


Figure 9: *"PART OF"* Relationship Visualization

4 Database Creation

The database creation process has relied on various **CSV** datasets freely available online and on **Python**[4] programming language, in order to extract the information suitable to build an appropriate dataset fulfilling our needs. Since information comes from different providers, such as public companies (e.g **DatiOpen.it**), the need to have a data processing pipeline was high. Thus through several Python functions and libraries we sequentially applied different modification to data, including:

- Null fields rows removal
- Duplicate rows removal
- Standardization of fields to have only first letter uppercase
- Document delimiter conversion to match CSV standard (e.g from "**;**" to "**,**" delimiter)
- Generation of random documents IDs, with respect to the real document number format (e.g : 2 letters and 7 digits for Italian passport number)
- Address of places with **OpenStreetMap API**[5] or pick a random one if not found with given parameters
- Generation of random birth dates and assignment of random residence address, Covid tests, vaccine doses, visited places and contact tracking records to people.

In the following sections, along with a brief description of particular problems we had to face, for each entity of the database we show portions of initial and final structure of the corresponding CSV files.

4.1 People

The original data comes from the **2019 Facebook breach**[6]. We selected a single text file coming from the Italian section of the dump, firstly reduced the amount of information (original file size is around **800 MB**) to 5000 entries, then we parsed it, dealing with a non standard data record format, removed unused information (such as relationship status) and added random realistic information (such as birth date).

```
393301033276:100005382931120:Andreaselene:Simonini:female:Vado Ligure:Vado Ligure::Simonini store:::
393301035513:100011637224869:Enrico:Filippi:male::Sulmona:In a relationship::::
393301042451:100003253206061:Giovanni:Salice:male::::Ristorante Pizzeria Bella Napoli:::
393301045191:100014730188555:Gino:Giuliana:male:Priolo Gargallo:Campobello di Licata::pensionato:::
```

Figure 10: Original structure

name	surname	fullname	sex	birth date	id	id_type
Alice	Cappelli	Alice Cappelli	female	1967-12-31	VY63643PJ	Italian electronic identity card
Giuseppe	Gullotto	Giuseppe Gullotto	male	1938-12-07	PQ3049354	Italian passport
Ruggero	de Conti	Ruggero de Conti	male	1960-11-12	UM1658390	Italian passport

Figure 11: Final structure

4.2 Countries

The original data come from a **Kaggle user**[7], from whose data we removed the median age lifespan field.

Country	Population	Med. Age
China	1440297825	38
India	1382345085	28
United States	331341050	38
Indonesia	274021604	30
Pakistan	221612785	23

Figure 12: Original structure

Country	Population
China	1440297825
India	1382345085
United States	331341050
Indonesia	274021604
Pakistan	221612785

Figure 13: Final structure

4.3 Cities

The original data comes from the **simplemaps**[8] world cities csv dataset, from which we removed unused fields (such as **iso2** country code) and filtered out non Italian cities.

city	city_ascii	lat	lng	country	iso2	iso3	admin_name	capital	population	id
Tokyo	Tokyo	35.6897	139.6922	Japan	JP	JPN	Tōkyō	primary	37977000	1392685764
Jakarta	Jakarta	-6.2146	106.8451	Indonesia	ID	IDN	Jakarta	primary	34540000	1360771077
Delhi	Delhi	28.6600	77.2300	India	IN	IND	Delhi	admin	29617000	1356872604
Mumbai	Mumbai	18.9667	72.8333	India	IN	IND	Mahārāshtra	admin	23355000	1356226629
Manila	Manila	14.6000	120.9833	Philippines	PH	PHL	Manila	primary	23088000	1608618140

Figure 14: Original structure

city	city_ascii	iso3	admin_name	population	id
Milan	Milan	ITA	Lombardy	1366180	1380724377
Naples	Naples	ITA	Campania	966144	1380646673
Turin	Turin	ITA	Piedmont	870952	1380244627
Palermo	Palermo	ITA	Sicilia	668405	1380428434
Genoa	Genoa	ITA	Liguria	580097	1380716540
Bologna	Bologna	ITA	Emilia-Romagna	389261	1380202039

Figure 15: Final structure

4.4 Places

Since we decided to focus on an Italian oriented dataset, original CSVs were retrieved from the Italian Open Data website **DatiOpen.it**[9]. Although we were able to find datasets about restaurants, cafes, theaters, cinemas and hospitals, the separator in all these files is a semicolon, a lot of data is duplicated or missing and an **OpenStreetMap ID** is used instead of an explicit address. Even though this ID is used in our database, to have meaningful entries we needed to retrieve, when possible, through the **OpenStreetMap API**, or pick randomly, a huge amount of addresses, since some of these files have more than **30.000** records.

city	admin_name	name	ID
AGLIE'	Piemonte	Agriturismo Cascina La Desiderata	2909468474
AGLIE'	Piemonte	La Pulperia - Osteria y Bar de Tapas	2909469529
AGLIE'	Piemonte	Trattoria Scudo di Francia	2909469554
ALA DI STURA	Piemonte	Maronero	1760944832
ALA DI STURA	Piemonte	Ristorante Pizzeria Alafishing Tre Ponti	2942035682
ALICE SUPERIORE	Piemonte	Agriturismo Garavot	2931926843

Figure 16: Original structure

city	admin_name	name	ID	address
Aglie'	Piemonte	Agriturismo Cascina La Desiderata	2909468474	Strada per Feletto 2
Aglie'	Piemonte	La Pulperia - Osteria y Bar de Tapas	2909469529	Corso Camillo Benso Conte di Cavour 16
Aglie'	Piemonte	Trattoria Scudo di Francia	2909469554	Via Principe Amedeo 30
Ala Di Stura	Piemonte	Maronero	1760944832	Piazza Centrale 12
Ala Di Stura	Piemonte	Ristorante Pizzeria Alafishing Tre Ponti	2942035682	Ivrea 79
Alice Superiore	Piemonte	Agriturismo Garavot	2931926843	Malalbergo 159

Figure 17: Final structure

4.5 Tests & Vaccines

The Vaccines CSV containing information related to world countries comes from **ourworldindata.org** [10] and didn't need manipulations. The covid test type CSV was written from scratch with reference to a **FDA article** [11] regarding Covid tests.

location	vaccine
Austria	Johnson&Johnson
Austria	Moderna
Austria	Oxford/AstraZeneca
Austria	Pfizer/BioNTech
Belgium	Johnson&Johnson
Belgium	Moderna
Belgium	Oxford/AstraZeneca
Belgium	Pfizer/BioNTech

type
Molecular test
Antigen test
Antibody test

Figure 18: Final structures

4.6 Relationships

In this subsection some of the relationships are briefly described from a creation point of view, highlighting the random data generation procedures handled to create pseudo realistic database content:

- **RECEIVED**: 25% of people have a single dose of Vaccine, whose administration ranges from **2021-10-7** to **2021-11-7**. Moreover 25% of people have a double dose of Vaccine, whose first dose ranges from **2021-01-05** to **2021-06-05** and the second dose's date is **after 28 days**. In both cases assignment of vaccination dates has been done starting from the eldest.
- **TOOK**: each person has a randomly picked number of samples assigned, whose date ranges from **2021-01-05** to **2021-11-7** and whose outcome is determined randomly.
- **WENT TO**: each person is associated with **50** places in the database through this relationship. Places are picked randomly and retrieved through their unique ID, moreover the registration date ranges from **2020-3-10** to **2021-11-07**.
- **LIVES WITH**: this relationship is created forming randomly groups of cohabitants sizing from 1 to 6 people.
- **LIVES IN**: this relationship's creation is strictly related to the previous one, since the same randomly picked address is assigned to each cohabitant.
- **MET**: this is the most time consuming relationship creation among the ones involved in database creation. A random amount of **MET** relationships, ranging form 10 to 50, is assigned to each person. For each creation various steps are performed. Firstly a random place ID is picked. Then a random person ID is picked, with appropriate checks to avoid reflexive relationships. Lastly a query is performed to verify that the two people aren't living together, if so the relationship is created, with a date attribute ranging from **2020-03-10** to **2021-11-07**.

5 Queries & Commands

5.1 Queries

The following subsection presents some useful queries and their possible textual result, with the aim of showing database's utility

- Find the Country with the highest number of positive Covid samples taken

```
MATCH (:Test)-[took:TOOK]-(p:Person)-[:LIVES_IN]-(c:City)-[:PART_OF]-(s:Country)
WHERE took.isPositive = true
WITH COUNT(p) AS positive_samples, s.name AS country
ORDER BY positive_samples DESC
LIMIT 1
RETURN country, positive_samples
```

country	positive_samples
Italy	10763

- Find the place where the highest number of people, tested positive to a Covid sample performed on a certain date, went in the previous n (e.g. 300) days.

```
MATCH (:Test)-[took:TOOK]-(p:Person)-[w:WENT_TO]-(place:Place)-
[l:LOCATED]-(c:City)-[pa:PART_OF]-(co:Country)
WHERE took.isPositive = true
AND took.date = date('2021-1-1')
AND w.date >= took.date - duration({days: 300})
AND w.date <= took.date
WITH COUNT(p) AS people, place.name AS name, place.type AS type,
l.address + ", " + c.name + ", " + c.region + ", " + co.name AS address
ORDER BY people DESC
LIMIT 1
RETURN name, type, address, people;
```

name	type	address	people
ABCinema d'Essai	cinema	Vicolo Baratono 124, Ivrea, Piedmont, Italy	63784

- List of people who tested positive to a Covid sample and the count of people they met over the last n (e.g.: 300) days

```
MATCH (t:Test)-[took:TOOK]-(p1:Person)-[m:MET]->(p2:Person)
WHERE took.isPositive = true
AND m.date >= took.date - duration({days: 300})
AND m.date <= took.date
WITH COUNT(p2) AS people_met, p1.name AS name
ORDER BY people_met DESC
RETURN name, people_met;
```

name	people_met
Carlo Solinas	1494
Franco Lai	1194
Giannino Molinaro	1186
Mario Facchini	1178
Nico Borracci	1177

- Find the city with the highest number of first dose vaccinated people up to a given date

```
MATCH (:Vaccine)-[r:RECEIVED]-(p:Person)-[:LIVES_IN]-(c:City)-[:PART_OF]-(co:Country)
WHERE r.date <= date('2022-01-01')
AND r.number = 1
WITH COUNT(p) AS people,
c.name + ", " + c.region + ", " + co.name AS city
ORDER BY people DESC
LIMIT 1
RETURN city, people;
```

city	people
Milano, Lombardia, Italy	1076041

- Find the tracking source that registered the highest number of contacts

```
MATCH ()-[m:MET]->()
WITH m.source AS source, COUNT(m.source) AS num_sources
ORDER BY num_sources DESC
LIMIT 1
RETURN source, num_sources;
```

source	num_contacts
smartphone	27634

- Find the age range (**10 years**) who has been infected the most and the number of infections (in relation to the number of people composing that category)

```
MATCH (:Test)-[took:TOOK]-(p:Person)
WITH "["+(duration.between(p.birthDate,datetime()).years/10)*10+","
+(duration.between(p.birthDate,datetime()).years/10+1)*10+")" AS age_range,
COUNT(*) AS total_people,
SUM(CASE WHEN took.isPositive = true THEN 1 ELSE 0 END) AS positive_people,
SUM(CASE WHEN took.isPositive = true THEN 1 ELSE 0 END)*1.0/COUNT(*) AS rate
ORDER BY rate DESC
LIMIT 1
RETURN age_range,total_people,positive_people, round(rate, 2, 'HALF_UP');
```

age_range	total_people	positive_people	rate
[10,20)	8543	4676	0.55

- Find the vaccine with less positive samples registered until a certain date (in relation to the number of days past the vaccination)

```
MATCH (v:Vaccine)-[r:RECEIVED]-(p:Person)-[took:TOOK]-(test:Test)
WHERE r.number = 1
WITH v.manufacturer AS manufacturer,
SUM(CASE WHEN took.isPositive = true AND r.date <= took.date THEN 1 ELSE 0 END)
AS positive_people,
SUM(duration.inDays(r.date, date('2021-11-7')).days) AS total_days
ORDER BY total_days/positive_people ASC
RETURN manufacturer, total_days/positive_people;
```

manufacturer	total_days/positive_people
Johnson&Johnson	835
Pfizer/BioNTech	1262
Oxford/AstraZeneca	1321
Moderna	1372

- List the places visited by a given person over the last n (e.g.: 300) days starting from the positive Covid sample taken in a given date

```
MATCH (:Test)-[took:TOOK]-(p:Person)-[w:WENT_TO]-(place:Place)-
[l:LOCATED]-(city:City)-[pa:PART_OF]-(c:Country)
WHERE p.fullname = 'Nanda Donati'
AND took.isPositive = TRUE
AND took.date = date('2020-07-29')
AND w.date >= took.date - duration({days: 20})
AND w.date <= took.date
WITH place.name AS name,
l.address + ", " + city.name + ", " + city.region + ", " + c.name AS address,
w.date AS date, place.type AS type
ORDER BY date
RETURN name, type, address, date;
```

name	type	address	date
Movie Planet	cinema	Via della Libertà 84, Bellinzago Novarese, Piedmont, Italy	"2020-12-14"
Ospedale dei villeggianti	hospital	Ciclabile lungomare di Rimini 111, Rimini, Emilia-Romagna, Italy	"2020-12-22"
Antico Caffè Centrale	cafe	Via della Repubblica 38, Barga, Tuscany, Italy	"2020-12-27"

5.2 Commands

The following subsection presents some useful commands, with the aim of showing database's utility.

- **CSV LOAD Commands**. Column names are shown in paragraph 4.
  - Person nodes CSV Load

```
LOAD CSV WITH HEADERS FROM $path
AS line FIELDTERMINATOR ','
WITH line
MERGE (:Person
{name: line.name,
surname: line.surname,
fullname: line.fullname,
sex: line.sex,
birthDate: date(line.birth_date),
id: line.id,
idType: line.id_type});
```

  - Country nodes CSV Load

```
LOAD CSV WITH HEADERS FROM $path
AS line FIELDTERMINATOR ','
WITH line
MERGE (:Country
{name: line.Country, population: toInteger(line.Population)});
```

  - City nodes CSV Load

```
LOAD CSV WITH HEADERS FROM $path
AS line FIELDTERMINATOR ','
WITH line
MERGE (:City
{name: line.city_ascii,
id: toInteger(line.id),
population: toInteger(line.population),
region: line.admin_name}});
```

- Place nodes CSV Load (type = Restaurant/Cafe/Theater/Cinema/Hospital/...

```
LOAD CSV WITH HEADERS FROM $path
AS line FIELDTERMINATOR ','
WITH line
MERGE (:Place
{name: line.name,
id: toInteger(line.ID),
type: $type});
```

- Vaccine nodes CSV Load

```
LOAD CSV WITH HEADERS FROM $path
AS line FIELDTERMINATOR ','
WITH line
MERGE (:Vaccine {manufacturer: line.vaccine});
```

- Test nodes CSV Load

```
LOAD CSV WITH HEADERS FROM $path
AS line FIELDTERMINATOR ','
WITH line
MERGE (:Test {type : line.type});
```

- Update person's residence (with relationships adjustments)

```
MATCH (person:Person) - [old_street :LIVES_IN] - (old_city : City)
WHERE person.id = $person_id
DELETE old_street

WITH person
MATCH (person) - [old_lives_with :LIVES_WITH] - (old_cohabitant : Person)
DELETE old_lives_with

WITH person
MATCH(new_city:City)
WHERE new_city.id = $new_city_id

MATCH (new_cohabitant : Person) - [new_relationship :LIVES_IN] - (new_city)
WHERE new_relationship.address = $new_address
MERGE (new_cohabitant)-[:LIVES_WITH {address : $new_address}]->(person)
MERGE (person)-[:LIVES_IN {address :$new_address }]->(new_city)

RETURN person;
```

- Create new **TOOK** relationship

```
MATCH (person:Person)
MATCH (test:Test)
WHERE person.id = $id
AND test.type = $type
MERGE (person)-[took:TOOK {date:$date, city_id:$city, isPositive:$isPositive}]-(test)
RETURN person, test, took;
```

- Create a new family (**ids**: string that contains id of new family members, delimited by a comma)

```
MATCH (person:Person)-[lives_with:LIVES_WITH]-(:Person)
MATCH(person)-[lives_in:LIVES_IN]-(:City)
WHERE person.id IN split($ids,,  )
DELETE lives_with, lives_in

MATCH (p1:Person)
MATCH (p2:Person)
MATCH (c:City)
WHERE p1.id IN split($ids,",")
AND p2.id IN split($ids,",")
AND p1.id <> p2.id
AND c.id = $city
MERGE (p1)-[:LIVES_WITH]-(p2)
MERGE (p1)-[:LIVES_IN {address : $address}]-(c)
```

- Delete a Person node

```
MATCH (n: Person{fullname: $value})
DETACH DELETE n;
```

A Delivery content

Apart from this report, the delivery folder also contains the following files :

- **Python scripts** to create CSVs and database. The Python project is mainly composed by 2 files
  - **utils.py**: it contains various utility functions to generate random data&manipulate CSV files.
  - **main.py**: it contains a class to handle database connection and nodes&relationships creation through various functions.
- **Folders** containing the various versions of CSVs files, to keep track of the subsequent modifications (these are located inside the Python project folder, inside the **datasets** folder)
- **Database dump** file.
- **Text file** with queries&commands.
- **"Import" folder** containing files to place inside Neo4j "Import" folder to execute **LOAD CSV** commands

B Notes on database creation script

The creation script, which can be executed invoking the `populateDatabase` method of class `CovidGraphHandler` located inside file **main.py**, takes approximately **6 hours to complete**.

What it creates inside the Neo4j database are:

- **12014** nodes :
  - 5000 **Person** nodes
  - 4883 **Place** nodes
  - 2123 **City** nodes
  - 4 **Vaccine** nodes
  - 3 **Test** nodes
  - 1 **Country** node
- **296682** directed (593364 undirected) relationships :
  - **2123** directed (4246 undirected) **PART OF** relationships
  - **1139** directed (2278 undirected) **LOCATED** relationships
  - **3752** directed (7504 undirected) **RECEIVED** relationships
  - **6537** directed (13074 undirected) **TOOK** relationships
  - **8441** directed (16882 undirected) **LIVES WITH** relationships
  - **5000** directed (10000 undirected) **LIVES IN** relationships
  - **119651** directed (239302 undirected) **MET** relationships
  - **150040** directed (300080 undirected) **WENT TO** relationships

C Meta graph

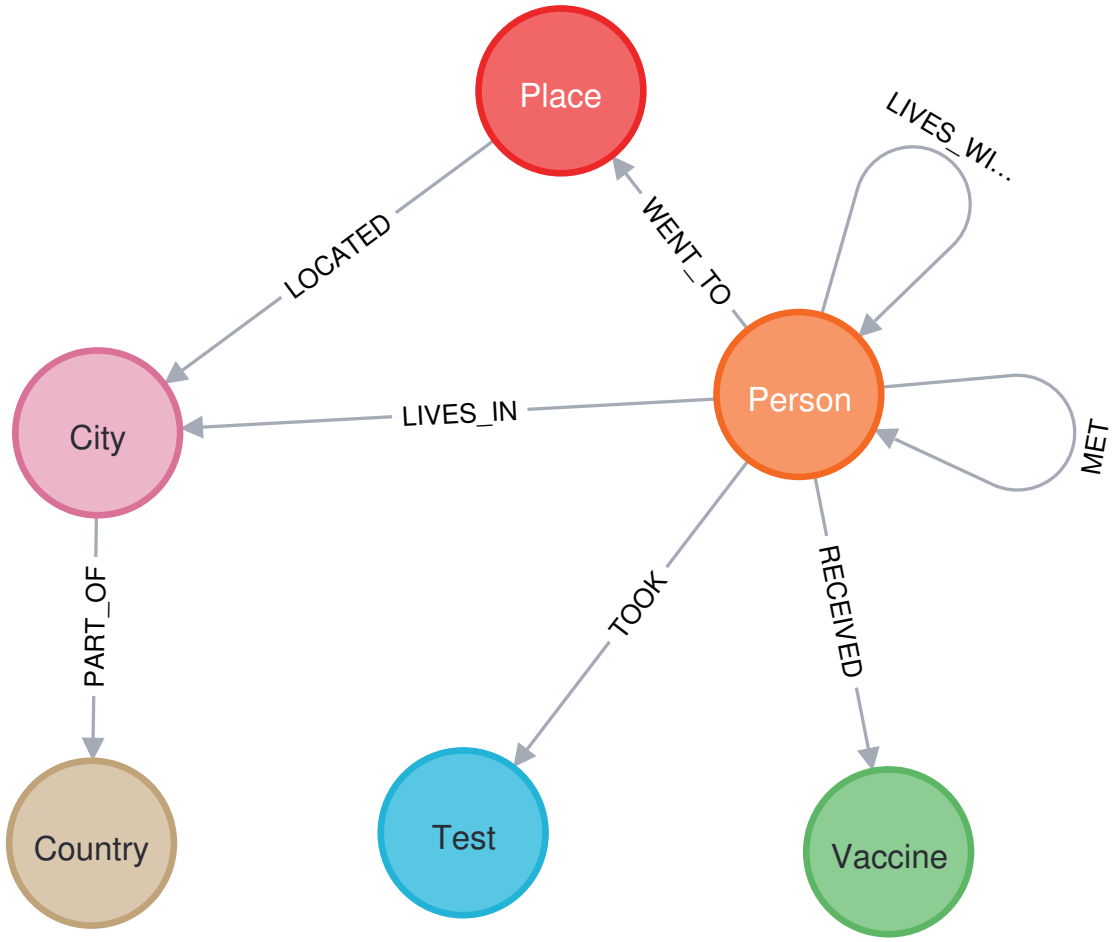


Figure 19: Meta graph visualization

References

[1] Neo4j, "The world's leading graph database," *https://neo4j.com*.

[2] Neo4j, "Neo4j bloom," *https://neo4j.com/product/bloom/*.

[3] Neo4j, "Neo4j undirected graph," *https://community.neo4j.com/t/unable-to-create-undirected-graph/18405*.

[4] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[5] OpenStreetMap *https://www.openstreetmap.org*.

[6] TheGuardian, "Facebook data leak," *https://www.theguardian.com/technology/2021/apr/03/500-million-facebook-users-website-hackers*, 2021.

[7] T. N. Prabhu, "Population by Country," *https://www.kaggle.com/tanuprabhu/population-by-country-2020*, 2020.

[8] simplemaps, "World cities database," *https://simplemaps.com/data/world-cities*, 2021.

[9] DatiOpen, "Dati open, il portale italiano dell'open data," *http://www.datiopen.it*.

[10] ourworldindata, "Coronavirus (covid-19) vaccinations," *https://ourworldindata.org/covid-vaccinations*, 2021.

[11] FDA, "Coronavirus disease 2019 testing basics," *https://www.fda.gov/consumers/consumer-updates/coronavirus-disease-2019-testing-basics*, 2021.