Pandemic Information System Model

SYSTEMS AND METHODS FOR BIG AND UNSTRUCTURED DATA

PROF. MARCO BRAMBILLA

FIRST DELIVERY
NEO4J PROJECT

November 2021

Avci Oguzhan - 10557284
Gentile Nicole - 10594355
Rigamonti Davide - 10629791
Singh Raul - 10623232
Tagliaferri Mattia - 10572418

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Problem Specification

The scope of this project is to build the structure for an information system capable of managing pandemic information for a given country.
Even though there are no direct names in the given specific, we can assume without loss of generality, that the "pandemic" is referring to the *COVID-19* pandemic (the country that we will take into consideration is the *USA*).
The database receives data coming from tracking applications that use sensors in smartphones, wearable objects or other devices to understand whether two people had a contact; data includes date and time of the contact.
Moreover, some places like restaurants, theaters and hospitals, explicitly collect date and name of visiting people.
The idea is to use the system so that, if a person gets flagged as *infected*, we can understand who are all the people who had a contact with him/her and, in the meantime, build useful analytics with the gathered information.
Contacts can be of 3 different types:

- between familiars;
- between people that went to the same location considering a reasonable span of time between their accesses;
- given by a tracking application.

1.2 Hypoteses

- Infection

  People are considered either *infected* if they have a contagion date, *healed* if they have an healing date and a contagion date, or neither if they have none yet.

  We assumed that people can do tests without being infected (or after healing) and that they can decide not to get the vaccine; however, vaccinated people can still get infected.

  We also assumed that people can get covid at most once (realistic if we consider perfect antibodies); in this way it's easier to store and retrieve data in order to build statistics.

  Given the previous assumptions we consider reasonable to only store the date of the last negative test and that consider all vaccines as single-dose generic vaccines.

- Tracking

  Members of a family are assumed to be all the people who live together, relatives who see each other very often or roommates. Obviously, all members

of a family live in the same city and are considered ”always in contact” and families do not change over time.

For simplicity, contacts coming from tracking applications go from 28/10/2021 to 10/11/2021.

The tracking relationships assume that different devices can only see each other if they are of the same *device type*.

We assume that the tracking devices only function outside houses, therefore contacts between members of the same family (unless they meet outside) and contacts between people inside the same location at the same time won't be tracked.

When tracing back contacts from an *infected* person, only the ones that happened up to 14 days prior the positive test are taken into consideration, eventual contacts after the positive test up to the healing date are also considered.

When two people go to the same location, it's considered a valid contact only if the two accesses happen between 2 hours from one to the other.

- DB population

  In order to populate the database, we imposed that every person visits from 6 to 10 locations, in particular we have people who got covid after 18/10/2021 and went visiting places from 18/09/2021 to 17/10/2021 (since we assume that after a positive test they are put in quarantine) and all the others went to visit locations from 18/09/2021 to 17/11/2021.

  There may be some temporal/spatial inconsistencies inside the generated dataset, these will be overlooked since they are not crucial to the scope of this project.

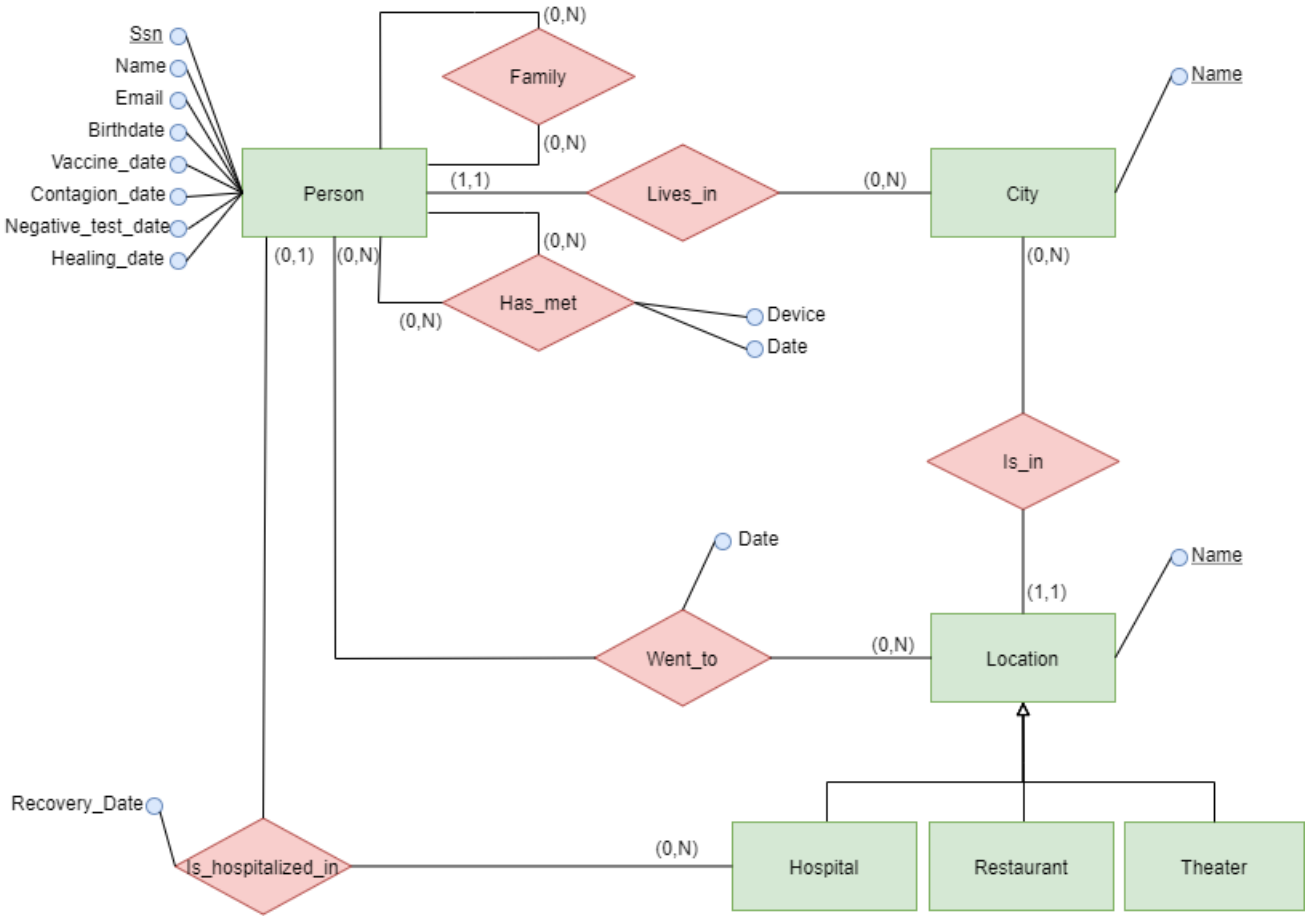
  We assumed that people can go to hospitals for two reasons: for an appointment/visit or if they are hospitalized because of *COVID-19*.

  In order to simplify data access we assumed that, for people who are hospitalized, date of hospitalization and contagion coincide and they leave the hospital on the healing date.

  We also assumed that all cities and location names are unique.

2 Database

2.1 ER Diagram



The diagram represents the first phase of the database design.

The diagram per se should be already auto-explicative. However, for better clarity, a more detailed description of each entity along with reasons of why such elements should be present in the database is provided:

- `Person`: it contains information that allows the system to keep track of the users and perform analytics on them.
- `City`: it contains information regarding cities that are present in the system. It can be useful to check where a person lives or a location resides in order to aggregate useful data.
- `Location`: it contains information regarding buildings that are present in the system and where people can be tracked explicitly. In the implementation, the various specializations of the *Location* "relation" were merged into the parent with the help of an attribute named `type`.

- `Hospital`: it is a specialization of the entity location. Given the current emergency, hospitals can offer hospitalization rooms for patients with *COVID-19* but can also be visited like other locations.
- `Restaurant`: it is a specialization of the entity location.
- `Theater`: it is a specialization of the entity location.

2.2 Dataset description

We represented a population of 800 people in the USA, considering 400 locations between 48 cities; data is recorded from February 2020.
We used three types of nodes: Person, Location and City. People are characterized by their name (composed by name and surname), birthdate, city, email, social security number, vaccine date, date of the last contagion, date of the last negative test and healing date.
Locations have a name and a type (restaurant, theater, hospital); type is used so that the dataset can be easily expanded with new types of locations.
Possible relationships are:

- `WENT_TO` to track people who visited a location at a certain time;
- `FAMILY` for family relationships;
- `IS_IN` between locations and cities;
- `LIVES_IN` to link people to the city where they live in;
- `HAS_MET` to indicate contacts between people using tracking app or devices;
- `IS_HOSPITALIZED_IN` to indicate people who are/were hospitalized for covid reasons.

The complete script used for the creation of the database, togheter with a database dump will be provided alongside this document.

2.3 Queries

Only the most significative queries are shown in this document.
All the implemented queries will be provided alongside this document.

N.B. Dates in input should be strings formated as 'YYYY-MM-DD', while datetimes as "YYYY-MM-DDThh:mm:ssZ".

2.3.1 Average age

```
WITH    apoc.date.convert(timestamp(), "ms", "d") AS now
MATCH   (pc:Person)
WHERE   pc.contagion_date IS NOT NULL AND
        NOT EXISTS((pc)-[:IS_HOSPITALIZED_IN]->(:Location {type:"Hospital"}))
WITH    apoc.date.parse(toString(pc.birthdate), "d", "yyyy-MM-dd") AS pc_birth,
        now
MATCH   (ph:Person)
WHERE   EXISTS((ph)-[:IS_HOSPITALIZED_IN]->(:Location {type:"Hospital"}))
WITH    apoc.date.parse(toString(ph.birthdate), "d", "yyyy-MM-dd") AS ph_birth,
        now, pc_birth

RETURN  avg((now-pc_birth)/365) AS avg_age_nothospitalized,
        avg((now-ph_birth)/365) AS avg_age_hospitalized;
```

Returns the average age for infected (but not hospitalized) and for infected and hospitalized people inside the database.
An example of result is:

avg_age_nothospitalized	avg_age_hospitalized
50.037848605577615	55.02631578947379

2.3.2 Cities

```
MATCH (cities:City)<-[:LIVES_IN]-(person:Person)
WHERE person.contagion_date >= date($date1) AND
person.contagion_date <= date($date2)
WITH cities, count(*) AS infected
ORDER BY infected DESC
RETURN cities.name AS city_name, infected;
```

Returns cities and number of infected people per city in a given span of time, results are sorted by the number of infected people. Takes the following parameters:

- `$date1` *(string)*: starting date;
- `$date2` *(string)*: ending date.

An example of result is:

	city_name	infected
1	Honolulu	8
2	Concord	8
3	Des Moines	7
4	Little Rock	6

2.3.3 Devices

```
MATCH   (p1:Person)-[met:HAS_MET]->(p2:Person)
WITH    apoc.date.parse(toString(p1.contagion_date), "ms", "yyyy-MM-dd")
            AS p1_contagion_date,
        apoc.date.parse(toString(p1.healing_date), "ms", "yyyy-MM-dd")
            AS p1_healing_date,
        apoc.date.parse(toString(p2.contagion_date), "ms", "yyyy-MM-dd")
            AS p2_contagion_date,
        met
WHERE   p1_contagion_date IS NOT NULL AND
        (p2_contagion_date IS NULL OR p2_contagion_date > p1_contagion_date)
WITH    apoc.date.add(p1_contagion_date, "ms", -14, "d") AS contagion_l,
        met, p1_healing_date
WHERE   met.date.EpochMillis >= contagion_l AND
        (p1_healing_date IS NULL OR met.date.EpochMillis < p1_healing_date)
WITH    count(DISTINCT met) AS infected_contacts,
        met.device AS device
ORDER BY infected_contacts DESC
RETURN  device, infected_contacts;
```

Returns the number of infected contacts registered for each type of device, results are sorted by the number of infected contacts. An example of result is:

device	infected_contacts
smartphone	46
other	42
wearable	23

2.3.4 Healed

```
MATCH   (p:Person)
WHERE   p.contagion_date IS NOT NULL AND p.healing_date IS NOT NULL AND
        p.healing_date >= date($date1) AND p.healing_date <= date($date2)
RETURN  count(p) AS n_healed;
```

Returns the number of healed people over a given span of time.
Takes the following parameters:

- `$date1` *(string)*: starting date;
- `$date2` *(string)*: ending date.

An example of result is:

n_healed
15

2.3.5 Hospitalized

```
MATCH (p:Person)-[r:IS_HOSPITALIZED_IN]->(h:Location)
WITH collect(p) as patients, h, r
WHERE r.date >= date($date1) and r.date <= date($date2)
RETURN h.name ah hospital, count(patients) as patients
ORDER BY count(patients) DESC;
```

Returns the number of patients hospitalized for covid reasons in each hospital in a given period of time, results are ordered by number of patients.
Takes the following parameters:

- `$date1` *(string)*: starting date;
- `$date2` *(string)*: ending date.

An example of result is:

	hospital	patients
1	Blue Hill Memorial Hospital	5
2	Baylor Scott and White Texas Spine and Joint Hospital	3
3	Stephens County Hospital	3
4	Citizens Medical Center	3

2.3.6 Infected

```
MATCH   (p:Person)
WHERE   p.contagion_date IS NOT NULL AND
        p.contagion_date >= date($date1) AND p.contagion_date <= date($date2)
RETURN  count(p) AS n_infected;
```

Returns the number of infected people over a given span of time.
Takes the following parameters:

- `$date1` *(string)*: starting date;
- `$date2` *(string)*: ending date.

An example of result is:

n_infected
157

2.3.7 Locations

```
MATCH (places:Location)<-[r:WENT_TO]-(person:Person)
WHERE datetime($date1) <= r.date AND r.date <= datetime($date2)
AND person.contagion_date is not null
AND person.contagion_date >= date($date1)
AND person.contagion_date <= date($date2)
WITH places, count(*) AS infected
ORDER BY infected DESC
RETURN places.name as location, places.type as type, infected;
```

Returns locations and the corresponding number of people who went there and possibly got infected during a certain span of time, results are sorted according to the number of visits.
Takes the following parameters:

- `$date1` *(string)*: starting date;
- `$date2` *(string)*: ending date.

An example of result is:

	location	type	infected
1	The China Doll	Restaurant	7
2	Cochran Memorial Hospital	Hospital	6
3	Century 16 Aurora	Theater	6
4	Carmike Patton Creek 15 + Imax	Theater	6

2.3.8 Notifications

```
MATCH   (p:Person {ssn:$ssn})
WHERE   p.contagion_date IS NOT NULL
CALL {
        WITH    p
        CALL    apoc.path.expandConfig(p, {
                        relationshipFilter: "FAMILY>",
                        minLevel: 1,
                        maxLevel: -1,
                        labelFilter: "+Person",
                        uniqueness:"NODE_GLOBAL",
                        bfs:true
        }) YIELD path AS path
        UNWIND tail(nodes(path)) AS x
        WITH DISTINCT x AS contact
        RETURN  contact, "family" AS tracing, null AS time

        UNION

        WITH    p
        MATCH   (p)-[met:HAS_MET]->(pc:Person)
        WITH    apoc.date.parse(toString(p.contagion_date), "ms", "yyyy-MM-dd")
                    AS contagion_date,
                apoc.date.parse(toString(p.healing_date), "ms", "yyyy-MM-dd")
                    AS healing_date,
                pc, met
        WITH    apoc.date.add(contagion_date, "ms", -14, "d") AS contagion_l,
                pc, met, healing_date
        WHERE   (met.date.EpochMillis >= contagion_l) AND
                (met.date.EpochMillis < healing_date OR healing_date IS NULL)
        RETURN  pc AS contact, met.device AS tracing, met.date AS time

        UNION

        WITH    p
        MATCH   (p)-[go1:WENT_TO]->(loc:Location)<-[go2:WENT_TO]-(pl:Person)
        WITH    apoc.date.parse(toString(p.contagion_date), "ms", "yyyy-MM-dd")
                    AS contagion_date,
                apoc.date.parse(toString(p.healing_date), "ms", "yyyy-MM-dd")
                    AS healing_date,
                pl, go1, go2, loc
        WITH    apoc.date.add(contagion_date, "ms", -14, "d") AS contagion_l,
                apoc.date.add(go1.date.EpochMillis, "ms", -2, "h") AS go1_l,
                apoc.date.add(go1.date.EpochMillis, "ms", 2, "h") AS go1_u,
                pl, go1, go2, loc, healing_date
        WHERE   go2.date.EpochMillis >= go1_l AND
                go2.date.EpochMillis <= go1_u AND
                go1.date.EpochMillis >= contagion_l AND
                (go1.date.EpochMillis < healing_date OR healing_date IS NULL)
        RETURN  pl AS contact, loc.name AS tracing, go1.date AS time
}
RETURN DISTINCT contact, collect({tracing:tracing, time:time}) AS info;
```

Returns a list of people that a specific infected person (given their *social security number*) may has been in contact with, considering family relationships, contacts from device tracking and inferred contacts from explicit data collection in specific locations. All the various contraints and assumptions are mentioned in the **hypoteses section**. For each contact we offer complete information on the person that came in contact with the infected and, when possible, place/device information and time of the contact. Takes the following parameters:

- `$ssn` *(string)*: social security number of the infected person.

An example of result is:

	contact	info

1	{
	"identity": 489,
	"labels": [
	"Person"
	],
	"properties": {
	"negative_test_date": "2020-08-17",
	"birthdate": "1994-03-23",
	"name": "Clare Pocock",
	"contagion_date": "2020-02-21",
	"vaccine_date": "2020-06-14",
	"healing_date": "2020-03-02",
	"email": "cpocock2o@blog.com",
	"ssn": "379-46-5993"
	}
	}
		[
		{
		"time": null,
		"tracing": "family"
		}
		]

⋮	⋮	⋮

21	{
	"identity": 788,
	"labels": [
	"Person"
	],
	"properties": {
	"negative_test_date": "2020-09-10",
	"birthdate": "1957-02-18",
	"name": "Rockwell Forrestall",
	"contagion_date": "2020-08-28",
	"vaccine_date": "2021-09-18",
	"healing_date": "2020-09-10",
	"email": "rforrestall5s@stanford.edu",
	"ssn": "721-07-2343"
	}
	}
		[
		{
		"time": null,
		"tracing": "family"
		}
		]

22	{
	"identity": 356,
	"labels": [
	"Person"
	],
	"properties": {
	"name": "Cecilio Kitchinghan",
	"birthdate": "1960-02-08",
	"email": "ckitchinghan5t@sbwire.com",
	"ssn": "665-75-3753"
	}
	}
		[
		{
		"time":
		"2021-10-31T14:04:20Z",
		"tracing": "other"
		}
		]

⋮	⋮	⋮

⋮	⋮	⋮

24	{
	"identity": 622,
	"labels": [
	"Person"
	],
	"properties": {
	"negative_test_date": "2021-08-27",
	"birthdate": "1959-02-19",
	"name": "Eileen Vennings",
	"contagion_date": "2021-08-13",
	"vaccine_date": "2021-03-31",
	"healing_date": "2021-08-27",
	"email": "evennings14@joomla.org",
	"ssn": "282-56-5584"
	}
	}
		[
		{
		"time":
		"2021-11-06T20:50:28Z",
		"tracing": "wearable"
		}
		]

⋮	⋮	⋮

29	{
	"identity": 115,
	"labels": [
	"Person"
	],
	"properties": {
	"name": "Cooper Glisenan",
	"contagion_date": "2021-10-28",
	"birthdate": "1943-11-27",
	"email": "cglisenan37@yandex.ru",
	"ssn": "797-60-6462"
	}
	}
		[
		{
		"time":
		"2021-10-10T20:17:38Z",
		"tracing":
		"Chi St Vincent
		Morrilton"
		}
		]

2.3.9 Vaccinated

```
MATCH (p:Person)
WHERE p.contagion_date >= p.vaccine_date
WITH count(*) AS vaccinated
MATCH (p1: Person)
WHERE p1.contagion_date IS NOT NULL
RETURN (toFloat(vaccinated)/count(*))*100 AS infected_vaccinated_percent
```

Returns the percentage of vaccinated infected people over the total number of infected (considering also the non-vaccinated). An example of result is:

infected_vaccinated_percent
34.835355285961874

2.4 Commands

Only the most significative commands are shown in this document.
All the implemented commands will be provided alongside this document.

2.4.1 Access

```
MATCH   (a:Person {ssn:$ssn}),
        (b:Location {name:$location})
CREATE  (a)-[r:WENT_TO {date:datetime($date)}]->(b);
```

Explicit registration of a visit to a location. Takes the following parameters:

- `$ssn` *(string)*: social security number of the person;
- `$date` *(string)*: date and time of the visit;
- `$location` *(string)*: name of the location.

2.4.2 Contact

```
MATCH   (a:Person {ssn:$ssn1}), (b:Person {ssn:$ssn2})
CREATE  (a)-[r:HAS_MET {date:datetime($date), device:$device}]->(b),
        (b)-[r:HAS_MET {date:datetime($date), device:$device}]->(a);
```

Registration of a contact via device. Takes the following parameters:

- `$ssn1` *(string)*: social security number of the first person;
- `$ssn2` *(string)*: social security number of the second person;
- `$date` *(string)*: date and time of the contact;
- `$device` *(string)*: type of device that detected a contact.

2.4.3 Negative test

```
MATCH (p {ssn: $ssn})
WITH p,
    CASE
        WHEN p.healing_date IS NULL AND p.contagion_date IS NOT NULL
                THEN date($test_date)
        ELSE p.healing_date
    END
    AS healing
SET p.negative_test_date = date($test_date), p.healing_date = healing;
```

Registration of a negative COVID test. It updates the date of the last negative test of the person with the given SSN; if the person was infected, the healing date is updated with the date of the test, otherwise it is not changed. Takes the following parameters:

- `$ssn` *(string)*: social security number;
- `$test_date` *(string)*: date of the negative test.

2.4.4 Positive test

```
MATCH (p {ssn: $ssn})
WITH p,
    CASE
        WHEN p.contagion_date IS NOT NULL THEN p.contagion_date
        ELSE date($test_date)
    END
    AS infection
SET p.contagion_date = infection;
```

Registration of a positive COVID test. The contagion date is set equal to the test date, considering that if a person got covid, they cannot get it again (in this case the contagion date is not updated). Takes the following parameters:

- `$ssn` *(string)*: social security number;
- `$test_date` *(string)*: date of the positive test.

3 Application

3.1 Description

The app is resize-responsive and is split in 3 tabs:

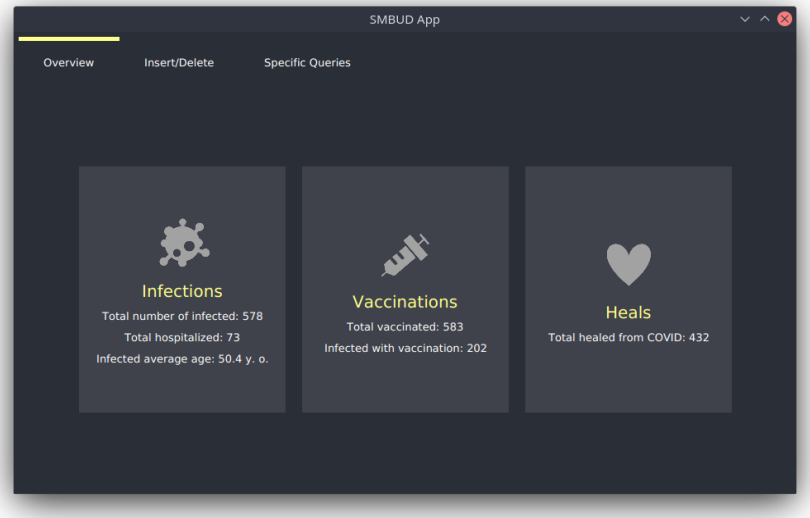
- The first, overview, contains a general view on the database, with general historical data, computed without date restrictions;
- The second tab contains different commands like adding a contact between two people or inserting a test result, every command will update the general overview, so changes will be visible instantly;
- The third tab contains date specific queries: by selecting the type of query from the dropdown menu and inserting start and end date, the app will output the required data, in form of table for some queries (like the cities with most infections).

Not all the written queries are implemented due to time restrictions, priority was given to commands and the queries necessary for the dashboard.

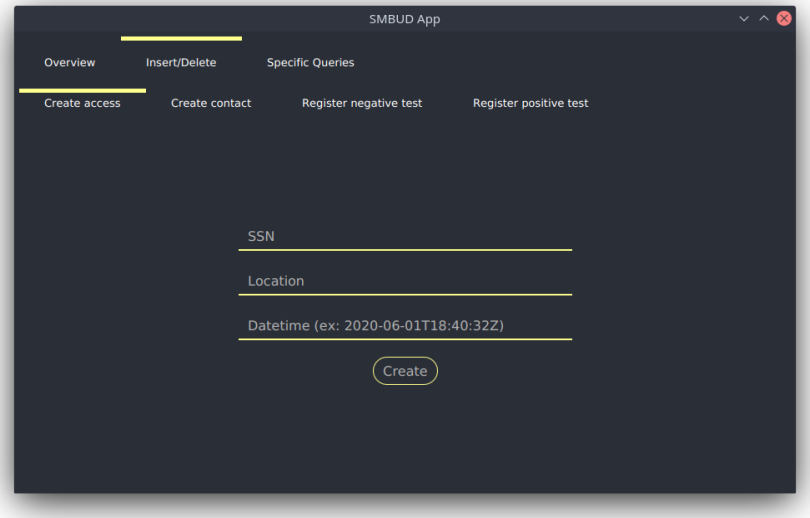
3.2 User Guide

The app is written in Java with the help of the JavaFX library, therefore it requires Java 11 or newer versions in order to run properly.
It will only boot if there's a Neo4j database opened on the same device using the default Bolt Protocol port: 7687.
The database must not be empty and its password specified at creation time should be "immuni".
The plugin APOC must be installed since most of the queries use its functionalities.

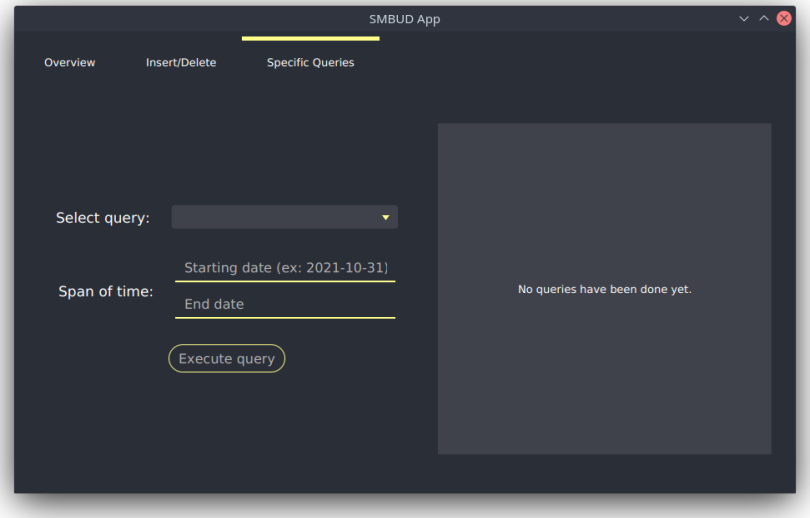
3.3 Screenshots



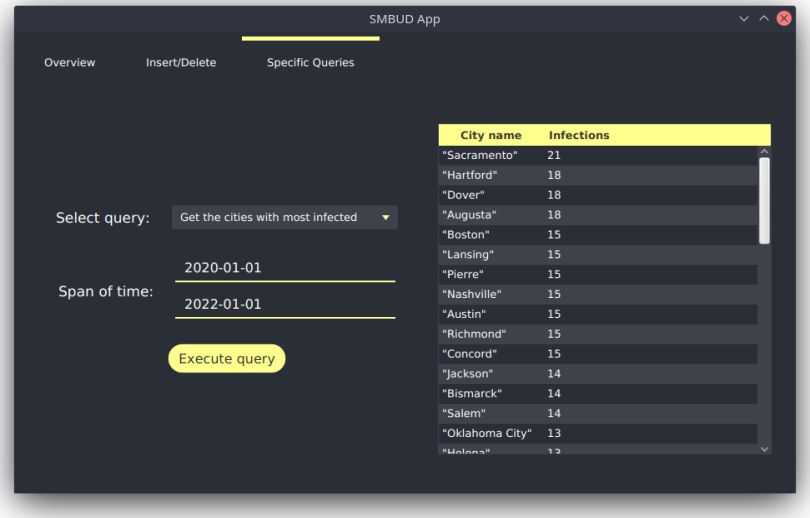
Main dashboard of the first tab.



Second tab containing various commands.



Third tab with the query selection system.



Result given by the execution of the query `QCities`. between the two visible dates

4 References and sources

In order to develop these project, the following tools were used:

- Neo4J and the Cypher language integrated with the APOC library to build the database and its queries;
- LaTeX to write the report;
- Github as a versioning and collaboration mean;
- Java, JavaFX, Scene builder and the Bolt Protocol to build the application;
- `https://www.Mockaroo.com` to generate realistic data for people;
- link to the resource used for the restaurant population;
- link to the resource used for the hospital population;
- link to the resource used for the theater population.