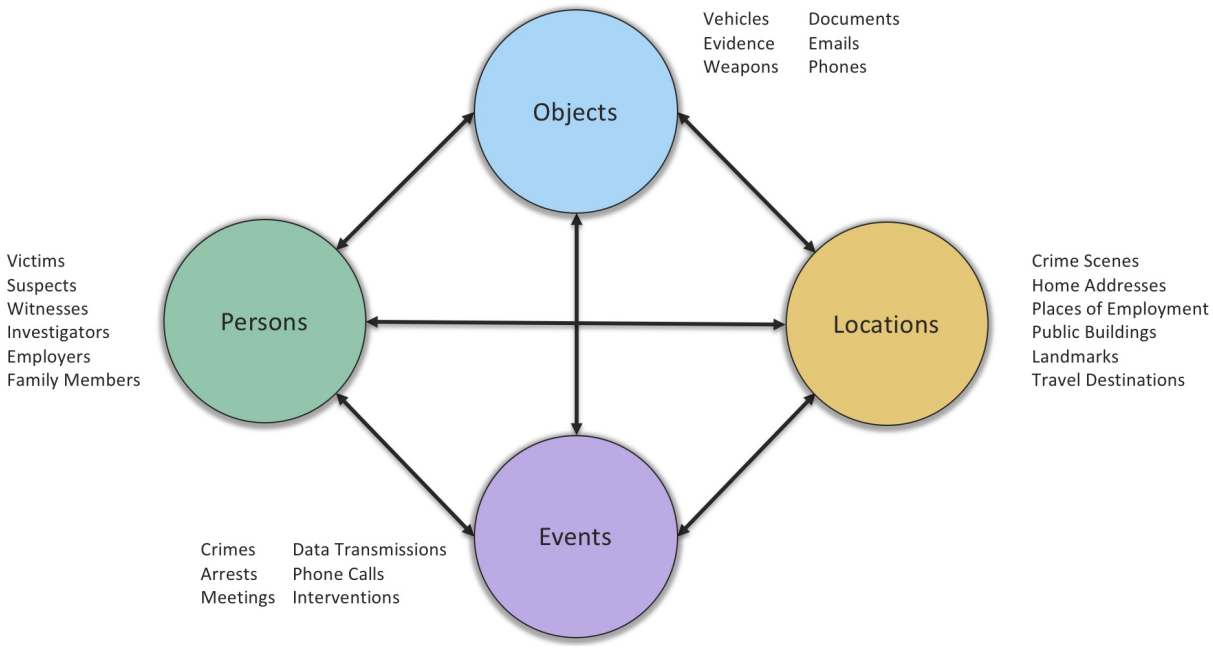
Neo4j : Crime Investingation

Gabriele Savoia

1. Descrizione e connessione con il Dataset

1.1 Descrizione

Per questo progetto è stata utilizzata la sandbox *crime investigation* di neo4j. Si tratta sostanzialmente di un esempio pratico di dati strutturati a grafo che seguono il modello POLE (People Object Location Event) caratterizzato dalle entità e relazioni riportate nella seguente figura :



I dati della sandobox si riferiscono ad un contesto poliziesco investigativo in cui sono riportate le informazioni inerenti ai crimini commessi (con i rispettivi soggetti coinvolti, organi di polizia iteressati, oggetti del crimine, luoghi dei reati ed anche le relazioni che intercorrono tra i soggetti in esame). Il dataset si basa sui dati pubblici dei crimini commessi a Manchester dall'agosto 2017 (disponibili nel sito), i quali, dopo essere stati ampliati e modificati, formano una struttura a grafo con i seguenti nodi e relazioni:

- **nodo Crime**, con le seguenti proprità:
  - **id**: univoco per crimine;
  - **type**: tipologia del crimine;
  - **date**: data in cui è stato registrato;
  - **last_outcome**: indica lo stato del crimine.
- **nodo Person**, con le seguenti proprietà:
  - **nhs_no**: identificativo persona;
  - **name** e **surname**
- **nodo Location**, con le seguenti proprietà:
  - **longitude** e **latitude**;
  - **address**: indirizzo,
  - **postcode**: contiene anche le informazioni dell'area
- **nodo Area**, con le seguenti proprietà:
  - **area_code**: codice dell'area;
- **nodo PostCode**, con le seguenti proprietà:
  - **code**: codice;
- **nodo Email**, con le seguenti proprietà:
  - **email_address**: email;
- **nodo Phone**, con le seguenti proprietà:
  - **phoneNo**: numero di telefono;
- **nodo PhoneCall**, con le seguenti proprietà:
  - **call_date**: istante di chiamata;
  - **call_time**: istante chiamata,
  - **call_type**: tipo di chiamata (messaggio o chiamata),
  - **call_duration**: durata della chiamata
- **nodo Object**, con le seguenti proprietà:
  - **description**: numero di telefono;
  - **id**: identificativo,
  - **type**: tipologia

- **nodo Vehicle**, con le seguenti proprietà:
  - **reg**: numero;
  - **year**: anno,
  - **model**: modello,
  - **make**: marca
- **nodo Officer**, con le seguenti proprietà:
  - **badge_no**: numero badge;
  - **rank**: livello,
  - **name** e **surname**

Per quanto riguarda le relazioni, alcune di quelle più interessanti sono le seguenti:

- relazione tra Persons:
  - **KNOWS**: relazione generica;
  - **FAMILY_REL**: relazione di parentela;
  - **KNOWS_LV**: relazione per indicare la convivenza;
  - **KNOWS_PHONE**: una persona che ha il numero di telefono di un'altra;
  - **KNWOS_SN**: c'è una interazione tramite social network.
- relazioni per modellare le chiamate tra Person:
  - con le relazioni **CALLED** e **CALLER** del nodo PhoneCall al nodo Person è possibile ricostruire le chiamate effettuate tra diversi soggetti.

L'insieme dei dati segue quindi il modello POLE ed è possibile visualizzarli di seguito:

1.2 Connessione

Una volta lanciata la sandbox neo4j è necessario collegarsi all'indirizzo specificato (in connection-detail).

In [1]:
```
from py2neo import Graph,Node,Relationship

graph = Graph("bolt://174.129.69.102:7687", auth=("neo4j", "beliefs-garage-lot"))
```

In [2]:
```
graph.run("MATCH (p:Person) RETURN count(DISTINCT p) AS total_people").data()
```

Out[2]:
```
[{'total_people': 369}]
```

1.3 Query particolari proposte dalla sandbox

Di seguito sono riportate alcune query proposte nello use case che utilizzano tecniche particolari (**non è stata considerata la parte di graph analytics**)

In [ ]:
```
# Query per effettuare ricerche basate sulla distanza tra punti definiti da
# coordinate latitudine e longitudine
query_1 = """
MATCH (l:Location {address: '1 Coronation Street', postcode: 'M5 3RW'})
WITH point(l) AS corrie
MATCH (x:Location)-[:HAS_POSTCODE]->(p:PostCode),
(x)<-[:OCCURRED_AT]-(c:Crime)
WITH x, p, c, distance(point(x), corrie) AS distance
WHERE distance < 500
RETURN x.address AS address, p.code AS postcode, count(c) AS crime_total, collect(distinct(c.type)) AS crime_type,
ORDER BY distance
LIMIT 10
"""


# L'obiettivo è quello di ricavare gli shortest path (entro un certo numero di hop) tra le persone coinvolte in
# crimini di droga non ancora risolti di cui l'ispettore avente badge_no: '26-52' sta indagando.
# La particolarità in questo caso è del doppio UNWIND che permette di trovare tutte le possibili coppie (prodotto
# cartesiano) in riferimento alle 'persons' che soddisfano la query (la WHERE con il < dopo le UNWIND serve
# a garantire che non ci siano duplicati tra le coppie (p1, p2) e che p1 != p2)
# es. la coppia (Annie, Bonnie) è presente 1 sola volta e non esiste un'altra coppia (Bonnie, Annie)
query_2 = """
MATCH (c:Crime {last_outcome: 'Under investigation', type: 'Drugs'})-[:INVESTIGATED_BY]->(:Officer {badge_no: '26-
(c)<-[:PARTY_TO]-(p:Person)
WITH COLLECT(p) AS persons
UNWIND persons AS p1
UNWIND persons AS p2
WITH * WHERE id(p1) < id(p2)
MATCH path = allshortestpaths((p1)-[:KNOWS|KNOWS_LW|KNOWS_SN|FAMILY_REL|KNOWS_PHONE*..3]-(p2))
RETURN path
"""


# Il concetto è simile alla query precedente, ma è esteso al concetto di persona vulnerabile.
# Si vuole infatti vedere se ci sono connessioni tra persone vulnerabili.
# In questo caso con la WHERE e il <> si garantisce solo che le coppie (p1, p2) siano tali che p1 != p2, ma
# permette la presenza di duplicati del tipo (Annie, Bonnie) e (Bonnie, Annie)
query_3 = """
MATCH (p:Person)-[:KNOWS]-(friend)-[:PARTY_TO]->(:Crime)
WHERE NOT (p:Person)-[:PARTY_TO]->(:Crime)
WITH p, count(distinct friend) AS dangerousFriends
ORDER BY dangerousFriends DESC
LIMIT 5
WITH COLLECT (p) AS people
UNWIND people AS p1
UNWIND people AS p2
WITH * WHERE id(p1) <> id (p2)
MATCH path = shortestpath((p1)-[:KNOWS*]-(p2))
RETURN path
"""

# Si tratta di un'ulteriore definizione di persona vulnerabile.
query_4 = """
MATCH (p:Person)-[:FAMILY_REL]-(relative)-[:KNOWS]-(famFriend)-[:PARTY_TO]->(:Crime),
(p)-[:CURRENT_ADDRESS]->(:Location)<-[:CURRENT_ADDRESS]-(relative)
WHERE NOT (p:Person)-[:PARTY_TO]->(:Crime) AND
 NOT (relative)-[:PARTY_TO]->(:Crime)
RETURN p.name AS name, p.surname AS surname, p.nhs_no AS id, count(DISTINCT famFriend) AS DangerousFamilyFriends
ORDER BY DangerousFamilyFriends DESC
LIMIT 5
"""
```

2. Modifica del grafo

2.1 Trasformazione da property a relazione per le tipologie di crimini

Nel modello attuale, per ciascun crimine è associata una *property* chiamata *type* per indicare la tipologia di crimine (di seguito sono riportate tutte le possibili tipologie attualmente presenti).

In [3]:
```
# Query 2.1.1
graph.run("MATCH (c:Crime) RETURN DISTINCT c.type AS type , count(*) AS crimes").data()
```

Out[3]:
```
[{'type': 'Public order', 'crimes': 4839},
 {'type': 'Robbery', 'crimes': 541},
 {'type': 'Theft from the person', 'crimes': 423},
 {'type': 'Vehicle crime', 'crimes': 2598},
 {'type': 'Burglary', 'crimes': 2807},
 {'type': 'Criminal damage and arson', 'crimes': 3587},
 {'type': 'Violence and sexual offences', 'crimes': 8765},
 {'type': 'Other theft', 'crimes': 2140},
 {'type': 'Shoplifting', 'crimes': 1427},
 {'type': 'Possession of weapons', 'crimes': 236},
 {'type': 'Bicycle theft', 'crimes': 414},
 {'type': 'Other crime', 'crimes': 651},
 {'type': 'Drugs', 'crimes': 333},
 {'type': None, 'crimes': 1}]
```

Si vuole adesso modificare questo modello al fine di gestire situazioni in cui un **crimine può essere associato a più tipi** es. "Drugs" e "Possesion of weapons" contemporaneamente. A questo punto è possibile procedere in due modi :

- definire *type* come una **lista**: riduce la complessità del grafo ma complica query del tipo "trovare i crimini con gli stessi tipi";
- modellare i tipi come **nodi**: aumenta la complessità del grafo ma permette di eseguire query come quella sopra citata in maniera più semplice ed efficace.

E' stata quindi scelta la modellazione tramite **nodi** (chiamati CrimeType) e di seguito è riportato il codice per la relativa creazione.

Prima di crearli, è stato scelto di impostare i **vincoli di unicità** sia dei nodi Crime che CrimeType che in automatico definiscono il relativo **indice sulle property** impostate come uniche.

In [4]:
```
# Vincoli di unicità per Crime e CrimeType (indice definito in automatico su 'crime.id' e 'crime_type.name')

query_1 = "CREATE CONSTRAINT ON (c:Crime) ASSERT c.id IS UNIQUE"
query_2 = "CREATE CONSTRAINT ON (c_type:CrimeType) ASSERT c_type.name IS UNIQUE"

graph.run(query_1)
graph.run(query_2)
```

Out[4]: (No data)

Per la creazione dei nodi, i passi seguiti sono i seguenti:

- definisco *types* come una lista (senza duplicati) di tutti i possibili tipi di crimine;
- effettuo l' UNWIND della lista così da avere una riga per ogni tipo di crimine e per ciascuno:
  - con la MATCH creo i bound node relativi ad un certo tipo di crimine;
  - tramite la MERGE creo (solo se non esiste) il nodo relativo al tipo di crimine;
  - la seconda MERGE permette di creare la relazione OF_TYPE tra i nodi crimine (di un certo tipo) e il relativo nodo del tipo
  - infine la property *type* è eliminata dai nodi Crime considerati

In [5]:
```
query = """
MATCH (c:Crime)
WITH COLLECT(DISTINCT(c.type)) AS types
UNWIND types AS type_
MATCH (bounded_c:Crime {type: type_})
MERGE (c_type:CrimeType {name: type_})
MERGE (bounded_c)-[:OF_TYPE]->(c_type)
REMOVE bounded_c.type
RETURN DISTINCT c_type.name
"""

graph.run(query).data()
```

Out[5]:
```
[{'c_type.name': 'Bicycle theft'},
 {'c_type.name': 'Burglary'},
 {'c_type.name': 'Criminal damage and arson'},
 {'c_type.name': 'Drugs'},
 {'c_type.name': 'Other crime'},
 {'c_type.name': 'Other theft'},
 {'c_type.name': 'Possession of weapons'},
 {'c_type.name': 'Public order'},
 {'c_type.name': 'Robbery'},
 {'c_type.name': 'Shoplifting'},
 {'c_type.name': 'Theft from the person'},
 {'c_type.name': 'Vehicle crime'},
 {'c_type.name': 'Violence and sexual offences'}]
```

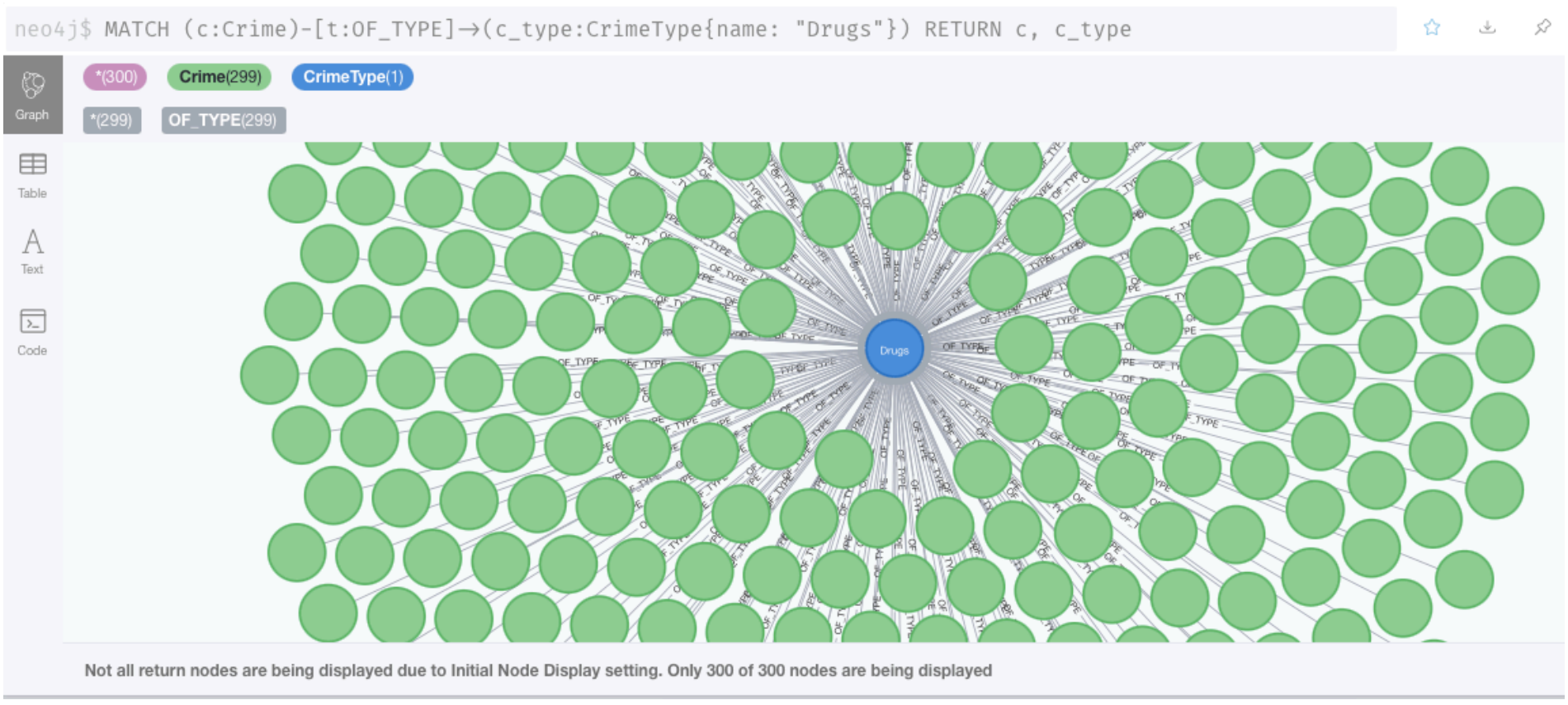
La seguente query ritorna per ciascuna tipologia il numero di crimini commessi. E' possibile notare che questo risultato è analogo a ciò che stato ottenuto con la query 2.2.1 (il nodo Crime che prima aveva type a None, adesso è modellato come un nodo Crime non associato a nessun nodo CrimeType).
Con questa struttura però, a differenza della precedente, è possibile gestire in maniera più efficiente (in relazione a certi tipi di query) le situazioni in cui ad un crimine sono associate più tipologie.

In [6]:
```
query = """
MATCH (c_type:CrimeType)
RETURN DISTINCT c_type.name AS type, size((c_type)<-[:OF_TYPE]-(:Crime)) AS crimes
"""

graph.run(query).data()
```

Out[6]:
```
[{'type': 'Bicycle theft', 'crimes': 414},
 {'type': 'Burglary', 'crimes': 2807},
 {'type': 'Criminal damage and arson', 'crimes': 3587},
 {'type': 'Drugs', 'crimes': 333},
 {'type': 'Other crime', 'crimes': 651},
 {'type': 'Other theft', 'crimes': 2140},
 {'type': 'Possession of weapons', 'crimes': 236},
 {'type': 'Public order', 'crimes': 4839},
 {'type': 'Robbery', 'crimes': 541},
 {'type': 'Shoplifting', 'crimes': 1427},
 {'type': 'Theft from the person', 'crimes': 423},
 {'type': 'Vehicle crime', 'crimes': 2598},
 {'type': 'Violence and sexual offences', 'crimes': 8765}]
```

Visualmente, in relazione ad un singolo CrimeType (es. *Drug*), si ottiene un sotto-grafo come quello nella figura qua riportata (sono visualizzati solamente 300 nodi ma in realtà ne sono presenti 333 per questa relazione).



2.2 Aumento dettaglio relazioni tra Person e Crime

Come suggerito anche nella sezione conclusiva della sandbox, si vuole adesso modificare la struttura del grafo aggiungendo maggiore specificità oltre alla relazione generica *PARTY_TO* tra *Person* e *Crime*.
In particolare si è interessati a capire in che modo (con quale "ruolo") un certo individuo è legato ad un crimine ed in particolare si vuole specificare relazioni del tipo: **WITNESS_TO**, **ACCUSED_OF**, **ACCOMPLICE_OF**, oltre che a **PARTY_TO.**

Come primo passo è stata analizzata più nel dettaglio l'attuale relazione *PARTY_TO* così da comprendere meglio la problematica da analizzare.

In [7]:
```
query_1 = """
MATCH (p:Person), (c:Crime)
RETURN count(DISTINCT p) AS total_people,
       count(DISTINCT c) AS total_crimes
"""
stats_1 = graph.run(query_1).data()

query_2 = """
MATCH (p:Person)-[r:PARTY_TO]->(c:Crime)
RETURN count(r) AS total_relations,
       count(DISTINCT p) AS distinct_people_related_to_crime,
       count(DISTINCT c) AS distinct_crime_with_person
"""
stats_2 = graph.run(query_2).data()

print('total_people : '+str(stats_1[0]['total_people']))
print('total_crimes : '+str(stats_1[0]['total_crimes']))
print('total_relations : '+str(stats_2[0]['total_relations'])+'\n')
print('distinct_people_related_to_crime : '+str(stats_2[0]['distinct_people_related_to_crime']))
print('distinct_crime_with_person : '+str(stats_2[0]['distinct_crime_with_person']))
print('property_of_relation : '+str(graph.run("MATCH ()-[r:PARTY_TO]->() RETURN keys(r)").data()[0]))
```

```
total_people : 369
total_crimes : 28762
total_relations : 55

distinct_people_related_to_crime : 29
distinct_crime_with_person : 55
property_of_relation : {'keys(r)': []}
```

Da questi valori è possibile vedere che :

- una persona partecipa a diversi crimini (*distinct_people_related_to_crime* < *total_relations*);
- ogni relazione si riferisce ad un crimine diverso (*distinct_crime_with_person* = *total_relations*);
- le relazioni non hanno property per dettagliare in che modo una persona partecipa ad un crimine.

Dopo l'analisi dell'attuale relazione tra *Person* e *Crime*, di seguito è riportata la descrizione delle relazioni che si intendono modellare:

- **PARTY_TO** : relazione *generica* per indicare il coinvolgimento di una certa persona in un crimine (sono lasciate quelle che esistono già nel grafo);
- **ACCUSED_OF** : relazione *specifica* per indicare una certa persona come accusata di un crimine. Per tutte le persone che hanno *attualmente* una relazione PARTY_TO con un crimine, associo la relazione *ACCUSED_OF* (li considero come accusati);
- **WITNESS_TO** : relazione *specifica* che indica una persona come testimone in un crimine. E' necessario introdurre anche la relazione *PARTY_TO*;
- **ACCOMPLICE_OF** : relazione *specifica* che definisce una persona come complice in un crimine. E' necessario introdurre anche la relazione *PARTY_TO*.

Per simulare queste relazioni nel grafo, sono stati effettuati i seguenti passaggi:

1. per **PARTY_TO**: sono lasciate le relazioni già esistenti nel grafo;
2. per **ACCUSED_OF**: relazioni aggiunte tra tutti i nodi aventi già la relazione *PARTY_TO*;
3. per **WITNESS_TO** e **ACCOMPLICE_OF**: è creato un apposito file csv contenente le colonne : *crime_id*, *WITNESS_TO*, *ACCOMPLICE_OF*. Ovvero per ogni riga è presente l'id di un crimine con i relativi id di persone considerate come testimoni e complici per quel crimine. In particolare sono stati considerati tutti i crimini con la relazione *PARTY_TO* e per la determinazione dei testimoni e dei complici la scelta è casuale su tutte quelle persone non ancora connesse ad un crimine.

Di seguito ci si è occupati dei **punti 1 e 2** (**PARTY_TO** e **ACCUSED_OF**).

In [8]:
```
query = """
MATCH (p:Person)-[r:PARTY_TO]->(c:Crime)
MERGE (p)-[r_accused:ACCUSED_OF]->(c)
RETURN count(r_accused) as number_rel_ACCUSED_OF
"""

graph.run(query).data()
```

Out[8]:
```
[{'number_rel_ACCUSED_OF': 55}]
```

Per il **punto 3**, di seguito è riportata la funzione per la generazione del file csv e poi successivamente questo sarà utilizzato per l'inserimento delle relazioni.

In [76]:

```
import csv
import random

def generate_dataset(graph):
    """
    Generazione csv per le nuove relazioni del grafo.
    COLONNE : crime_id, WITNESS_TO, ACCOMPLICE_OF

    """

    # TUTTI i Crime con la relazione PARTY_TO
    query_crimes_id = """
    MATCH (p:Person)-[:PARTY_TO]->(c:Crime)
    RETURN DISTINCT c.id AS id
    """
    crimes_id = [crime['id'] for crime in graph.run(query_crimes_id).data()]

    # TUTTE le persone non ancora in relazione con un crimine ('nhs_no' è l'identificativo univoco).
    query_people_id = """
    MATCH (p:Person)
    WHERE NOT (p)-[:PARTY_TO]->(:Crime)
    RETURN DISTINCT p.nhs_no AS id
    """
    people_id = [person['id'] for person in graph.run(query_people_id).data()]

    # Creazione csv con i relativi valori.
    with open('./data/dataset.csv', 'w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(["crime_id", "WITNESS_TO", "ACCOMPLICE_OF"])
        start = 0
        end = len(people_id)-1

        # per TUTTI i Crimes con relazione PARTY_TO definisco sia WITNESS che ACCOMPLICE
        for crime_id in crimes_id:

            # testimone e complice scelti casualmente tra le persone non ancora collegate a crimini
            witness_to = people_id[random.randint(start, end)]
            accomplice_of = people_id[random.randint(start, end)]

            # per evitare che il testimone e il complice siano la stessa persona nello stesso crimine
            while (witness_to == accomplice_of):
                accomplice_of = people_id[random.randint(start, end)]

            writer.writerow([crime_id, witness_to, accomplice_of])

generate_dataset(graph)
```

In [117…

```
! head ./data/dataset.csv
```

```
crime_id,WITNESS_TO,ACCOMPLICE_OF
00196bf89518161241d405622cfefa2cc3a02a1175b0133d9ca8d5d064e77981,595-90-8809,884-33-9676
875d234d8e6e4328aa54242aecbc5423ca3561cee28b6a420a33daee1109a261,213-32-1448,446-36-0249
de2d17e8d7782fbfb6f7ddd746954ae25dd3d5a7e6de4df429caa57f79474e48,689-65-9692,450-68-4090
de1c4d40bdb618664ce29c7edd6b329218b0c8ff01c18ab446755b38ab363d05,323-71-3763,256-31-7892
04b0d2f1b563a455d55b2f2c24ef86564f1146630fc7d392e657ecf5f47303da,463-00-8389,991-70-5333
211005c29462004762fe6e69cc516cd75c13e98f93b07101d16b1ad6cd65d5db,392-52-7176,852-52-0933
0bf340c716089d015fbbc05fdf0c907f56154b190b23da6f9706615ffabd6c20,659-67-2728,401-01-1355
2c3efee8d4682cff4081bebf3aecc9504c6a209b1f5dd11198f0c8c05904a05c,323-32-8478,535-67-6237
34f309ebafbd1c46b5c4832f63321a10330835a5ae323952419439b69633bb2c,493-92-1993,520-24-8922
```

Una volta creato, il file è caricato su Google Drive e di seguito è riportata la query per la creazione delle relazioni (per ogni relazione WITNESS_TO o ACCOMPLICE_OF è creata anche la relativa relazione PARTY_TO).
Con la prima MATCH definisco i 3 bound node *c*, *p_witness*, *p_accomplice* e poi con le MERGE successive creo le relazioni opportune. In questo modo la singola relazione è creata seguendo il concetto che se non esiste la crea e esiste non ne crea duplicati.

In [9]:
```
# google link : "https://drive.google.com/file/d/107No4P8O26oe6RUHdubi3mSucWenzMLO/view?usp=sharing"
# id : 107No4P8O26oe6RUHdubi3mSucWenzMLO
# link da usare : "https://docs.google.com/uc?id=107No4P8O26oe6RUHdubi3mSucWenzMLO&export=download"

# attuale : https://drive.google.com/file/d/107SZIP3I3Q-jv2FpMwzdL9LB4u9pIe_D/view?usp=sharing

query = """
LOAD CSV WITH HEADERS FROM 'https://docs.google.com/uc?id=107SZIP3I3Q-jv2FpMwzdL9LB4u9pIe_D&export=download' AS ro
MATCH (c:Crime {id: row.crime_id}),
      (p_witness:Person {nhs_no: row.WITNESS_TO}),
      (p_accomplice:Person {nhs_no: row.ACCOMPLICE_OF})
MERGE (p_witness)-[r_witness:WITNESS_TO]->(c)
MERGE (p_witness)-[r_witness_party:PARTY_TO]->(c)
MERGE (p_accomplice)-[r_accomplice:ACCOMPLICE_OF]->(c)
MERGE (p_accomplice)-[r_accomplice_party:PARTY_TO]->(c)
RETURN count(r_witness) AS number_r_witness,
       count(r_witness_party) AS number_r_witness_party,
       count(r_accomplice) AS number_r_accomplice,
       count(r_accomplice_party) AS number_r_accomplice_party
"""

graph.run(query).data()
```

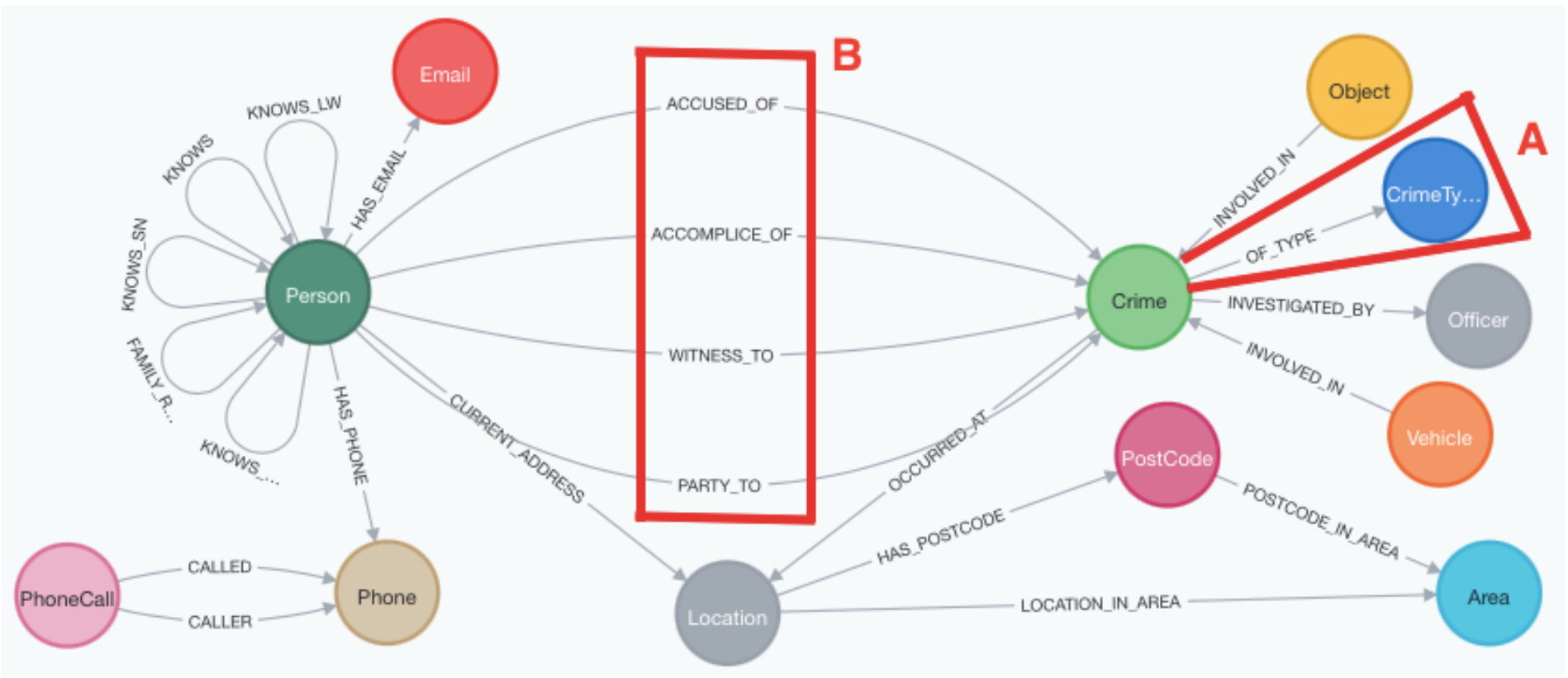
Out[9]:
```
[{'number_r_witness': 55,
  'number_r_witness_party': 55,
  'number_r_accomplice': 55,
  'number_r_accomplice_party': 55}]
```

In [10]:
```
query = """
MATCH (p:Person)-[r:PARTY_TO]->(c:Crime)
RETURN count(r) AS total_relations
"""

graph.run(query).data()
```

Out[10]:
```
[{'total_relations': 165}]
```

Sulla base delle modifiche al grafo effettuate, lo schema generale diventa quindi il seguente (con **A** indico le modifiche fatte in sezione 2.1 e con **B** nella 2.2).



3. Query

3.1. Per quali crimini risulta che l'accusato ha chiamato telefonicamente un proprio complice più di 1 volta

Con i dati a disposizione non è possibile ritornare nessun crimine con questa caratteristica. A fini dimostrativi quindi, è stato aggiunto in riferimento al crimine di id |*"00196bf89518161241d405622cfefa2cc3a02a1175b0133d9ca8d5d064e77981"* (con accusato avente *nhs:|"821-11-2735"*), un nuovo complice (*nhs_no:|"680-93-7668"*) il quale è stato chiamato e ha chiamato l'accusato del crimine considerato (sono state fatte query apposite per trovarlo, ed è stato visto che le chiamate tra i due sono in tutto 4: 2 in un senso e 2 nell'altro).
Di seguito quindi è riportato il codice per l'inserimento della nuova relazione *ACCOMPLICE_OF*.

In [11]:
```
# inserimento della nuova relazione : aggiungo un nuovo complice al crimine.
# questo complice è stato visto che ha contatti telefonici con l'accusato del crimine considerato

inser_query = """
MATCH (c:Crime {id: "00196bf89518161241d405622cfefa2cc3a02a1175b0133d9ca8d5d064e77981"}),
      (p:Person {nhs_no: "680-93-7668"})
MERGE (p)-[:ACCOMPLICE_OF]->(c)
RETURN size( (p)-[:ACCOMPLICE_OF]->(c) ) AS num_rel_p_c
"""
print(graph.run(inser_query).data())
```

```
[{'num_rel_p_c': 1}]
```

Sulla base della nuova relazione inserita, la query seguente ritorna il crimine nel quale l'accusato ha chiamato il complice più di 1 volta. Per modellare il caso in cui si vuole trovare i crimini in cui l'accusato e il chiamante hanno avuto un contatto telefonico (senza quindi considerare i vincoli su chi ha effettuato la chiamata e chi l'ha ricevuta), basta eliminare nella query la tipologia delle relazioni (al posto di [:CALLER] e [:CALLED] usare [] e []) e in questo caso la query ritornerebbe un numero di chiamate pari a 4, ovvero le chiamate totali effettuate tra i due soggetti.

In [13]:
```
query_1 = """
MATCH (chiamante)-[:ACCUSED_OF]->(c:Crime)<-[:ACCOMPLICE_OF]-(chiamato),
      (chiamante)-[:HAS_PHONE]->(telefono_chiamante:Phone),
      (telefono_chiamante:Phone)<-[:CALLER]-(chiamata:PhoneCall)-[:CALLED]->(telefono_chiamato:Phone),
      (chiamato)-[:HAS_PHONE]->(telefono_chiamato:Phone)
WITH c, chiamante, chiamato, count(*) AS n_chiamate
WHERE n_chiamate >= 2
RETURN c.id AS crimine,
       chiamante.nhs_no AS accusato_chiamante,
       chiamato.nhs_no AS complice_chiamato,
       n_chiamate
"""

display(graph.run(query_1).to_data_frame())
```

	crimine	accusato_chiamante	complice_chiamato	n_chiamate
0	00196bf89518161241d405622cfefa2cc3a02a1175b013...	821-11-2735	680-93-7668	2

3.2. Trovare il/i complice/i della persona con nhs_no='821-11-2735' per quei crimini da lui commessi dal 10 agosto al 20 agosto 2017

E' necessario effettuare uno split della data in modo da ricavare le componenti del giorno, del mese e dell'anno. In questo caso risulta che esistono 3 complici dell'accusato con nhs_no='821-11-2735' per i crimini da lui commessi dal 10/08/17 al 20/08/17.

In [15]:
```
query = """
MATCH (accusato:Person {nhs_no: '821-11-2735'})-[:ACCUSED_OF]->(c:Crime)<-[:ACCOMPLICE_OF]-(complice:Person)
WITH complice, [item in split(c.date, "/") | toInteger(item)] AS dateComp
WHERE date("2017-08-10") <= date({day: dateComp[0], month: dateComp[1], year: dateComp[2]}) <= date("2017-08-20")
RETURN DISTINCT complice.nhs_no AS complice
"""

display(graph.run(query).to_data_frame())
```

	complice
0	680-93-7668
1	884-33-9676
2	446-36-0249

3.3 Tipologia di crimine verificato più di frequente per ciascun postcode (riportare i primi 10 risultati)

In questo caso è stato necessario :

1. effettuare il conteggio di ciascun tipo di crimine per ogni *postcode* e quindi ottenere triple del tipo:
   postcode, *type_*, *num_type*
2. ordinare per *num_type*
3. per ciascun *postcode* calcolare la corrispondente lista dei tipi di crimini in esso verificati per poi selezionare solo il primo elemento (cioè il più frequente essendo la lista ordinata).

Questa query è stata fatta principalmente per fini dimostrativi in quanto il tutto potrebbe essere semplificato utilizzando le APOC. Lo stesso risultato infatti potrebbe essere ottenuto tramite l'utilizzo della *apoc.agg.maxItems()* che in sostanza prende in input delle coppie (item, value) e ritorna la lista di item che sono associati al value maggiore. Nel caso in esame, in riferimento a ciascun *postcode*, sono passati alla funzione i tipi di crimine verificati (item) con il relativo numero di occorrenze (value) ed in output è ritornato (per ogni diverso *postcode*) i/il corrispettivo tipo di crimine (.*items*) con l'occorrenza *maggiore* (.*value*).

In [23]:
```
# query senza APOC

query = """
MATCH (type:CrimeType)<-[OF_TYPE]-(c:Crime)-[:OCCURRED_AT]->(loc:Location)
WITH loc.postcode AS postcode, type.name AS type_, count(*) AS num_type
ORDER BY num_type DESC
RETURN postcode, collect(type_)[0] AS most_frequent_crime, max(num_type) AS frequence
LIMIT 10
"""

# query con APOC

query_apoc = """
MATCH (type:CrimeType)<-[OF_TYPE]-(c:Crime)-[:OCCURRED_AT]->(loc:Location)
WITH loc.postcode AS postcode, type.name AS type_, count(*) AS num_type
WITH postcode, apoc.agg.maxItems(type_, num_type) AS max_freq_type
RETURN postcode, max_freq_type.items AS most_frequent_crime, max_freq_type.value AS frequence
ORDER BY frequence DESC
LIMIT 10
"""

display(graph.run(query).to_data_frame())
display(graph.run(query_apoc).to_data_frame())
```

	postcode	most_frequent_crime	frequence
0	M1 1LU	Violence and sexual offences	47
1	M60 1TA	Shoplifting	32
2	M3 1DA	Violence and sexual offences	32
3	M40 9BY	Violence and sexual offences	30
4	M4 3AL	Shoplifting	28
5	M60 9AH	Violence and sexual offences	23
6	WN7 5SJ	Shoplifting	21
7	OL1 1QN	Shoplifting	21
8	M17 8JL	Shoplifting	17
9	OL16 1AW	Violence and sexual offences	17

	postcode	most_frequent_crime	frequence
0	M1 1LU	[Violence and sexual offences]	47
1	M60 1TA	[Shoplifting]	32
2	M3 1DA	[Violence and sexual offences]	32
3	M40 9BY	[Violence and sexual offences]	30
4	M4 3AL	[Shoplifting]	28
5	M60 9AH	[Violence and sexual offences]	23
6	WN7 5SJ	[Shoplifting]	21
7	OL1 1QN	[Shoplifting]	21
8	OL16 1AW	[Violence and sexual offences]	17
9	M17 8JL	[Shoplifting]	17

3.4 Dato l'id di un certo crimine, trovare le persone conoscenti con l'accusato (anche indiretti, ma al massimo di 2 hop) che partecipano in un qualche modo ad altri crimini (e NON a quello in esame) verificati in un raggio di 500 metri da quello considerato

E' stato scelto di utilizzare l'id |*"00196bf89518161241d405622cfefa2cc3a02a1175b0133d9ca8d5d064e77981"* per l'identificazione del crimine su cui la query va ad agire.
Per modellare la richiesta |*"...che partecipano in un qualche modo ad altri crimini..."* è utilizzata la relazione generica *PARTY_TO* presente per qualsiasi persona legata (testimone, accusato, complice) ad un certo crimine.

In [29]:
```
query = """
MATCH (accused:Person)-[:ACCUSED_OF]->(c:Crime {id: "00196bf89518161241d405622cfefa2cc3a02a1175b0133d9ca8d5d064e77
WITH accused, c, point(loc) AS coord_loc
MATCH (p:Person)-[:PARTY_TO]->(c_other:Crime)-[:OCCURRED_AT]->(loc_other:Location),
      (p)-[:KNOWS*..2]-(accused)
WITH accused, p, distance(coord_loc, point(loc_other)) AS distance
WHERE distance < 500
      AND NOT (p:Person)-[:PARTY_TO]->(c)
RETURN DISTINCT p.nhs_no AS conoscenti_accusato
"""
# il vincolo    NOT (p:Person)-[:PARTY_TO]->(c)    mi permette di selezionare quelle p che non hanno
# partecipato al crimine in esame. Non è quindi necessario controllare che    c_other <> c     e che   p <> accuse


graph.run(query).to_data_frame()
```

Out[29]:

	conoscenti_accusato
0	468-82-3915
1	678-06-9352
2	252-29-4929

3.5 Trovare il numero di persone 'vulnerabili' nel grafo

Estendendo il concetto di vulnerabilità dell'esempio della sandbox (slide 12/26), adesso una persona per poter essere considerata vulnerabile deve soddisfare *tutti* questi requisiti:

1. non aver mai partecipato nei ruoli di 'accusato' o 'complice' in un crimine;
2. essere testimone di un crimine *oppure* conoscere direttamente qualcuno che abbia partecipato ad un crimine come accusato o come complice.

In relazione a questa query sono riportati diversi approcci possibili :

- **metodo_1**: consente di trovare le persone vulnerabili sfruttando il predicato exists() che ritorna *true* se il pattern specificato esiste (*false* altrimenti) e le pipe per poter fare match su tipi diversi di relazioni utilizzando un solo pattern. Questo approccio risulta però poco flessibile in quanto permette sì di ritornare tutte le persone vulnerbili, ma non viene tenuto il riferimento alle relative persone pericolose (necessario per query più complesse);
- **metodo_2**: query con maggiore flessibilità in quanto riesco ad avere il riferimento, in relazione a ciascuna persona vulnerabile, della relativa lista di persone pericolose. In questo caso una certa persona *p* è considerata vulnerabile se la relativa *danger_list* è > 0. Infatti in riferimento al codice scritto, se la *danger_list* (i cui duplicati sono stati tolti tramite APOC) è > 0, significa che esiste almeno un *danger_1* o *danger_2* non *Null* (ricavati tramite OPTIONAL_MATCH), ovvero che il punto 2 della condizione di vulnerabilità è soddisfatto;
- **metodo_3**: è un'ulteriore modalità che sfrutta la stessa logica della *danger_list* di prima, ma per la sua creazione sono utilizzati i patter comprehension. La *danger_list* è infatti creata come un insieme senza duplicati a partire dall'unione delle due liste contenenti i valori di *danger_1* e *danger_2*. Per ciascuna di queste liste, gli elementi al proprio interno corrispondono ai nodi che fanno match con il pattern definito nella pattern comprehension.

In [30]:

```
##  metodo_1

query_1 = """
MATCH (p:Person)
WHERE NOT exists((p)-[:ACCUSED_OF|ACCOMPLICE_OF]->(:Crime))
      AND (
              exists((p)-[:WITNESS_TO]->(:Crime))
              OR
              exists((p)-[:KNOWS]-(:Person)-[:ACCUSED_OF|ACCOMPLICE_OF]->(:Crime))
          )
RETURN count(DISTINCT p) AS total_vulnerable_people
"""
print('metodo_1: ')
print(graph.run(query_1).data()[0])


##  metodo_2

query_2 = """
MATCH (p:Person)
WHERE NOT exists((p)-[:ACCUSED_OF|ACCOMPLICE_OF]->(:Crime))
OPTIONAL MATCH (p)-[:WITNESS_TO]->(:Crime)<-[:ACCUSED_OF|ACCOMPLICE_OF]-(danger_1:Person)
OPTIONAL MATCH (p)-[:KNOWS]-(danger_2:Person)-[:ACCUSED_OF|ACCOMPLICE_OF]->(:Crime)
WITH p, apoc.coll.toSet(COLLECT(DISTINCT danger_1.nhs_no)
                        +
                        COLLECT(DISTINCT danger_2.nhs_no)
                       ) AS danger_list
WHERE size(danger_list)>0
RETURN count(distinct(p)) AS total_vulnerable_people
"""
print('\nmetodo_2: ')
print(graph.run(query_2).data()[0])


##  metodo_3

query_3 = """
MATCH (p:Person)
WHERE NOT exists((p)-[:ACCUSED_OF|ACCOMPLICE_OF]->(:Crime))
WITH p, apoc.coll.toSet(
                        [ (p)-[:WITNESS_TO]->(:Crime)<-[:ACCUSED_OF|ACCOMPLICE_OF]-(danger_1:Person) | danger_1]
                        +
                        [ (p)-[:KNOWS]-(danger_2:Person)-[:ACCUSED_OF|ACCOMPLICE_OF]->(:Crime) | danger_2]
                       ) AS danger_list
WHERE size(danger_list)>0
RETURN count(distinct(p)) AS total_vulnerable_people
"""
print('\nmetodo_3: ')
print(graph.run(query_3).data()[0])
```

```
metodo_1:
{'total_vulnerable_people': 177}

metodo_2:
{'total_vulnerable_people': 177}

metodo_3:
{'total_vulnerable_people': 177}
```

3.6 Per ciascuna persona vulnerabile trovare la lista contenente le rispettive persone pericolose che si trovano nelle sue vicinanze. Riordinare i risultati sulla base del numero di persone pericolose ad esso vicino

Si tratta dell'estensione della query nella slide 15/26 della sandbox adattata alla nuova definizione di persona vulnerabile.
Sulla base della query 3.5 sono state aggiunte le clausole *WHERE* nelle relative pattern comprehension in modo tale da creare, per ciascun soggetto vulnerabile, una lista composta solamente dalle persone pericolose (senza duplicati) che attualmente si trovano nella stessa area del soggetto fragile.

In [31]:
```
query = """
MATCH (p:Person)-[:CURRENT_ADDRESS]->(:Location)-[:LOCATION_IN_AREA]->(area_p:Area)
WHERE NOT exists((p)-[:ACCUSED_OF|ACCOMPLICE_OF]->(:Crime))
WITH p, area_p, apoc.coll.toSet(
                                [ (p)-[:WITNESS_TO]->(:Crime)<-[:ACCUSED_OF|ACCOMPLICE_OF]-(danger_1:Person)
                                  WHERE (danger_1)-[:CURRENT_ADDRESS]->(:Location)-[:LOCATION_IN_AREA]->(area_p)
                                  | danger_1.nhs_no]
                                +
                                [ (p)-[:KNOWS]-(danger_2:Person)-[:ACCUSED_OF|ACCOMPLICE_OF]->(:Crime)
                                  WHERE (danger_2)-[:CURRENT_ADDRESS]->(:Location)-[:LOCATION_IN_AREA]->(area_p)
                                  | danger_2.nhs_no]
                               ) AS danger_list
WHERE size(danger_list)>0
RETURN p.nhs_no AS vulnerabile,
       area_p.areaCode AS area_code,
       danger_list AS lista_persone_pericolose,
       size(danger_list) AS num_persone_pericolose
ORDER BY num_persone_pericolose DESC
LIMIT 5
"""
graph.run(query).to_data_frame()
```

Out[31]:

	vulnerabile	area_code	lista_persone_pericolose	num_persone_pericolose
0	349-36-6161	SK3	[362-49-5861, 367-54-3328]	2
1	543-43-9738	WN3	[679-66-9286, 690-09-2036]	2
2	838-11-7607	M3	[821-11-2735]	1
3	709-43-8302	BL3	[577-47-4459]	1
4	640-26-0925	OL9	[589-69-0106]	1

In [ ]: