Progetto realizzato da Elena Curti (matr. 185431)

# Crimes Database

## Introduzione

Lo scopo di questo progetto è di rispondere a dei research question attraverso le tecniche di graph analytics.
Verrà usato un database POLE (Person, Object, Location, Event) contenente informazioni sui dei reati (inventati) commessi a Greater Manchester, UK.
In particolare si vorrà rispondere principalmente ai seguenti questiti:

- Chi sono gli agenti di polizia migliori?
- Esistono dei gruppi di persone pericolose? Quali sono?
- Esiste un "collegamento" tra persone con precedenti per spaccio e altre persone pregiudicate?

Per rispondere a tali quesiti verranno usati tecniche e algorimi diversi, basandosi sui dati presenti nel database.

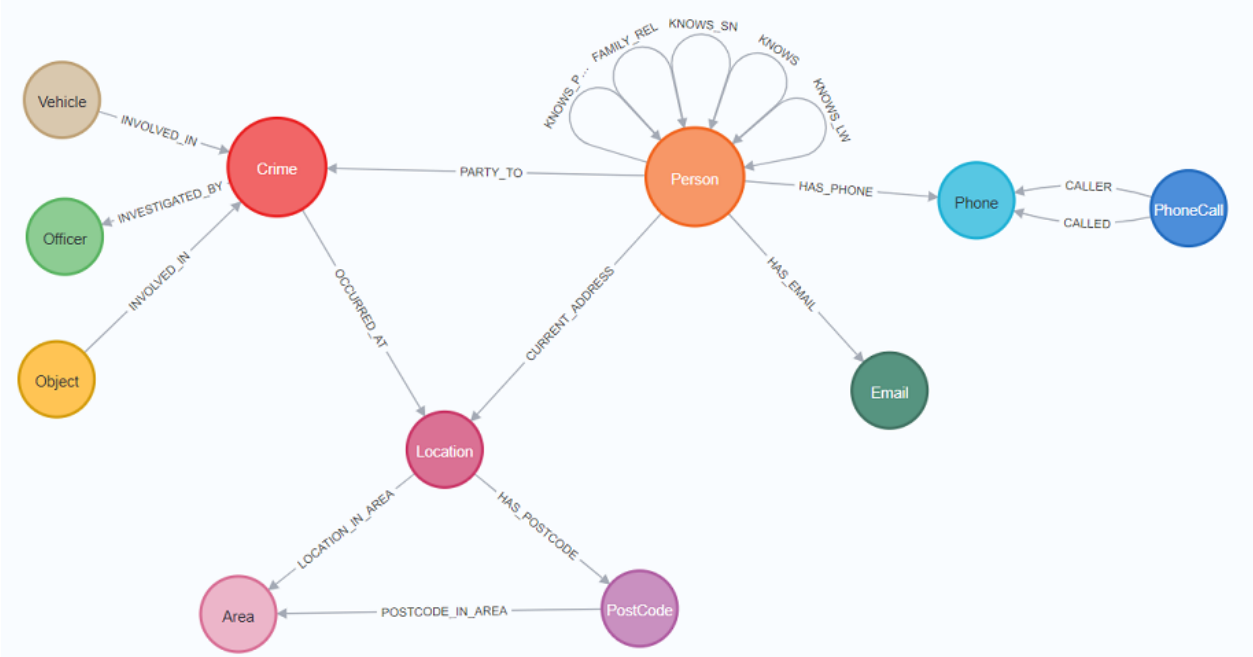
## Requisiti

E' necessaria l'installazione di Neo4j e del seguente pacchetto python:

In [1]:
```
# !pip install py2neo
```

## Descrizione del database

Il database usato è stato trovato al link: https://github.com/neo4j-graph-examples/pole.
Ha il seguente schema:



Sono quindi presenti:

- I crimini (nodo *Crime*). Le propietà memorizzate sono: "id", "date", "type", "last_outcome", "charge" e "note". Sono inoltre presenti:
  - Le prove raccolte (*Object*), con proprietà "id", "type", "description". Tutti gli oggetti hanno come come type "Evidence".
  - I veicoli coinvolti (*Vehicle*) con proprietà "make" (marca), "model", "year", "reg" (targa).
- Le persone (*Person*) che hanno compiuto reati o loro conoscenti. Proprietà: "nhs_no" (univoco), "name", "surname" e "age". Sono inoltre presenti dati aggiuntivi in altri nodi:
  - Numero di telefono (*Phone*), con propietà "phoneNo"
  - Le chiamate o i messaggi scambiati (*PhoneCall*) con proprietà: "call_date", "call_type", "call_duration", "call_time". call_type contiene "SMS" o "CALL".
  - L'email (*Email*) con proprietà "email_address"
- Le posizioni geografiche che rappresentano i luoghi dei crimini e le abitazioni delle persone. Sono memorizzati con tre nodi (dal luogo più preciso al più generale):
  - *Location* con proprietà: "address", "postcode", "longitude", "latitude"
  - *Postcode* con proprietà: "code"
  - *Area con* proprietà: "areaCode"

  Il postcode del Regno Unito è formato da due sezioni: l'area e il postcode vero e proprio. Ad esempio nel postcode "M1 1AA", "M1" indica il codice dell'area e "M1 1AA" l'intero codice postale. Il postcode è in genere limitato a una strada o a pochi isolati. L'area copre una città o un quartiere.
- I poliziotti (*Officer*) che hanno indagato un crimine. Propietà: "name", "surname", "badge_no", "rank".

Per memorizzare le conoscenze tra due persone sono presenti le seguenti relazioni:

- *FAMILY_REL*: le persone sono imparentate. E' presente la proprietà "rel_type" contenente "SIBLING" o "PARENT"
- *KNOWS_LW*: le due persone convivono (Lives With)
- *KNOWS_PHONE*: tra le due persone è stata effettuata almeno una chiamata o un messaggio.
- *KNOWS_SN*: le due persone si conoscono sui social network
- *KNOWS*: conoscenza generica, è presente se è presente almeno una tra le precedenti relazioni

A parte *FAMILY_REL*, nessun'altra relazione presente nel database contiene delle proprietà.

# Operazioni iniziali

Per il corretto funzionamento del progetto, occorre:

- Scaricare il seguente file https://github.com/neo4j-graph-examples/pole/blob/main/data/pole-50.dump
- Creare in Neo4j un DBMS a partire dal file scaricato. Scegliere come nome *neo4j* e come password *password1234*
- Scaricare nel DBMS i plugin "APOC" e "Graph Data Science Library"

- Far partire il DBMS

Eseguire poi le celle di codice, seguendo l'ordine proposto.

In [2]:
```
from py2neo import Graph
graph = Graph("bolt://localhost:7687",  auth=("neo4j", "password1234"))
print("Connessione al database eseguita correttamente!")
```

```
Connessione al database eseguita correttamente!
```

# 1. Chi sono gli agenti di polizia migliori?

Per trovare gli agenti di polizia migliori, uso la centrality analysis.

## 1.1. In base al numero dei crimini investigati (cypher query)

Cerco inizialmente gli agenti che hanno investigato su più crimini, usando una cypher query:

In [3]:
```
def print_n_results(cq, n=100):
    """ Funzione usata per stampare al massimo n righe dei risultati della query. Senza tale funzione, la stampa verrebbe automaticamente troncata a 3 righe. """
    ris = graph.run(cq)
    ris.sample_size = n
    display(ris)

cq = """
MATCH (o:Officer)<-[:INVESTIGATED_BY]-(c:Crime)
WITH o, count(c) as num_invest
RETURN o.badge_no as badge_number, o.name as nome, o.surname as cognome, o.rank as rango, num_invest
ORDER BY num_invest DESC
LIMIT 5 ;
"""
print_n_results(cq)
```

| badge_number | nome | cognome | rango | num_invest |
|---|---|---|---|---|
| 48-0216838 | Madelon | DeSousa | Sergeant | 50 |
| 04-5664884 | Cloe | Ings | Police Constable | 47 |
| 58-9758255 | Kania | Notti | Sergeant | 46 |
| 27-2227814 | Worthy | Nettles | Inspector | 45 |
| 79-9843712 | Olenolin | Klehyn | Sergeant | 44 |

Ai primi posti ci sono quindi Madelon DeSousa che ha indagato in 50 crimini, Cloe Ings (47 crimini), Kania Notti (46 crimini), Worthy Nettles (45 crimini) e Olenolin Klehyn (44 crimini).

## 1.2. In base al numero dei crimini (Page Rank e Betweenness Centrality)

Anche stavolta cerco gli stessi dati, usando però gli algoritmi di Page Rank e Betweenness Centrality forniti da gds.

In [4]:
```
# Creo la proiezione
if graph.run("call gds.graph.exists('officers')").data()[0]["exists"]:
  graph.run('call gds.graph.drop("officers")')

cq = """
CALL gds.graph.project(
  'officers',
  ['Officer', 'Crime'],
  {INVESTIGATED_BY: {orientation: 'UNDIRECTED'} }
) ;
"""

graph.run(cq)
```

Out[4]:

| nodeProjection | relationshipProjection | graphName | nodeCount | relationshipCount | projectMillis |
|---|---|---|---|---|---|
| {Crime: {label: 'Crime', properties: {}}, Officer: {label: 'Officer', properties: {}}} | {INVESTIGATED_BY: {orientation: 'UNDIRECTED', aggregation: 'DEFAULT', type: 'INVESTIGATED_BY', properties: {}}} | officers | 29762 | 57524 | 15 |

In [5]:
```
# Memory estimation
graph.run("""
CALL gds.pageRank.write.estimate('officers', { writeProperty: 'pageRank' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory
RETURN "Page Rank" as algoritmo, nodeCount, relationshipCount, bytesMin,
bytesMax, requiredMemory

UNION ALL

CALL gds.betweenness.write.estimate('officers', { writeProperty: 'betweenness'
})
YIELD  nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory
RETURN "Betweenness" as algoritmo, nodeCount, relationshipCount, bytesMin,
bytesMax, requiredMemory ;
""" )
```

Out[5]:

| algoritmo | nodeCount | relationshipCount | bytesMin | bytesMax | requiredMemory |
|---|---|---|---|---|---|
| Page Rank | 29762 | 57524 | 718808 | 718808 | 701 KiB |
| Betweenness | 29762 | 57524 | 8335024 | 8335024 | 8139 KiB |

In [6]:

```
# Risultati
print("Page Rank: ")
cq="""
CALL gds.pageRank.stream('officers')
YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS n, score AS pageRank
RETURN n.badge_no as badge_number, n.name as nome, n.surname as cognome, n.rank
as rango,pageRank
ORDER BY pageRank DESC
LIMIT 5 ;
"""
print_n_results(cq)


print("Betweenness Centrality: ")
cq="""
CALL gds.betweenness.stream('officers')
YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS n, score AS pageRank
RETURN n.badge_no as badge_number, n.name as nome, n.surname as cognome, n.rank
as rango,pageRank
ORDER BY pageRank DESC
LIMIT 5 ;
"""
print_n_results(cq)
```

```
Page Rank:
```

| badge_number | nome | cognome | rango | pageRank |
|---|---|---|---|---|
| 48-0216838 | Madelon | DeSousa | Sergeant | 22.602140755580333 |
| 04-5664884 | Cloe | Ings | Police Constable | 21.27718767680494 |
| 58-9758255 | Kania | Notti | Sergeant | 20.835536650546473 |
| 27-2227814 | Worthy | Nettles | Inspector | 20.39388562428801 |
| 79-9843712 | Olenolin | Klehyn | Sergeant | 19.952234598029545 |

```
Betweenness Centrality:
```

| badge_number | nome | cognome | rango | pageRank |
|---|---|---|---|---|
| 48-0216838 | Madelon | DeSousa | Sergeant | 1225.0 |
| 04-5664884 | Cloe | Ings | Police Constable | 1081.0 |
| 58-9758255 | Kania | Notti | Sergeant | 1035.0 |
| 27-2227814 | Worthy | Nettles | Inspector | 990.0 |
| 79-9843712 | Olenolin | Klehyn | Sergeant | 946.0 |

I risultati ottenuti sono ovviamente gli stessi trovati precedentemente.

## 1.3. In base al numero dei crimini indagati, con almeno 1 arresto (Degree centrality)

Per stabilire quali agenti siano i migliori, andrò stavolta a contare i soli crimini che hanno portato all'arresto di almeno una persona. Userò questa volta l'algoritmo Degree centrality.

In [7]:
```
# Creo la proiezione
if graph.run("call gds.graph.exists('officers-crime-con-persone')").data()[0]
["exists"]:
  graph.run('call gds.graph.drop("officers-crime-con-persone")')

cq="""
CALL gds.graph.project.cypher('officers-crime-con-persone',
  'MATCH (o:Officer|Crime)  RETURN DISTINCT id(o) as id',
  'MATCH (o:Officer)<--(c:Crime)<--(:Person) RETURN  id(o) AS source, id(c) as
target, "INVESTIGATED_BY" as type',
{validateRelationships:false}
);"""

graph.run(cq)
```

Out[7]:

| nodeQuery | relationshipQuery | graphName | nodeCount | relationshipCount | projectMillis |
|---|---|---|---|---|---|
| MATCH (o:Officer\|Crime) RETURN DISTINCT id(o) as id | MATCH (o:Officer)<--(c:Crime)<--(:Person) RETURN id(o) AS source, id(c) as target, "INVESTIGATED_BY" as type | officers-crime-con-persone | 29762 | 55 | 20 |

In [8]:
```
# Memory estimation
graph.run("""
CALL gds.degree.write.estimate('officers-crime-con-persone', { writeProperty:
'betweenness' })
YIELD  nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory
RETURN nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory ;
""" )
```

Out[8]:

| nodeCount | relationshipCount | bytesMin | bytesMax | requiredMemory |
|---|---|---|---|---|
| 29762 | 55 | 56 | 56 | 56 Bytes |

In [9]:

```
# Seleziono solo gli agenti che hanno arrestato una persona
cq="""
CALL gds.degree.stream('officers-crime-con-persone')
YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS n, score
RETURN n.badge_no as badge_number, n.name as nome, n.surname as cognome, n.rank
as rango,score
ORDER BY score DESC
LIMIT 5 ;
"""
print_n_results(cq)
```

| badge_number | nome | cognome | rango | score |
|---|---|---|---|---|
| 26-5234182 | Devy | Larive | Police Constable | 3.0 |
| 04-6245275 | Von | Death | Police Constable | 1.0 |
| 69-8126297 | Pauline | Petrasso | Police Constable | 1.0 |
| 99-0115424 | Kara-lynn | Ironside | Chief Inspector | 1.0 |
| 68-5266435 | Kris | Teaze | Sergeant | 1.0 |

Questa volta, gli agenti migliori sono: Devy Larive (con 3 investigazioni che hanno portato all'arresto di una o più persone), Von Death, Pauline Petrasso, Kara-lynn Ironside e Kris Teaze (tutte con 1 investigazione).

## 1.4. In base al numero di persone arrestate (HITS)

Non è detto che ad un crimine abbia partecipato una sola persona. Un agente che ha investigato un solo crimine potrebbe aver portato all'arresto di più persone. Per stabilire gli agenti migliori, quindi, cerco di capire quali agenti abbiano portato all'arresto di più persone. Userò stavolta l'algoritmo HITS.

In [10]:

```
# Creo la proiezione
if graph.run("call gds.graph.exists('officers-arresti')").data()[0]["exists"]:
  graph.run('call gds.graph.drop("officers-arresti")')

cq="""
CALL gds.graph.project.cypher('officers-arresti',
  'MATCH (o:Officer|Person)  RETURN DISTINCT id(o) as id',
  'MATCH (o:Officer)<--(c:Crime)<--(p:Person) RETURN distinct id(p) as source,
id(o) as target',
{validateRelationships:false}
```

```
);"""

graph.run(cq)
```

Out[10]:

| nodeQuery | relationshipQuery | graphName | nodeCount | relationshipCount | projectMillis |
|---|---|---|---|---|---|
| MATCH (o:Officer\|Person) RETURN DISTINCT id(o) as id | MATCH (o:Officer)<--(c:Crime)<--(p:Person) RETURN distinct id(p) as source, id(o) as target | officers-arresti | 1369 | 54 | 14 |

In [11]:

```
# In base al numero di persone arrestate
cq="""
CALL gds.alpha.hits.stream('officers-arresti',  {hitsIterations: 1})
YIELD nodeId, values
WITH gds.util.asNode(nodeId) AS n, values as score
RETURN n.badge_no as badge_number, n.name as nome, n.surname as cognome, n.rank
as rango,score
ORDER BY score DESC
LIMIT 5 ;
"""
print_n_results(cq)
```

| badge_number | nome | cognome | rango | score |
|---|---|---|---|---|
| 26-5234182 | Devy | Larive | Police Constable | {hub: 0.0, auth: 0.2672612419124244} |
| 54-9607307 | Gregorius | Shakesby | Police Constable | {hub: 0.0, auth: 0.1336306209562122} |
| 36-7678091 | Simmonds | Greensall | Police Constable | {hub: 0.0, auth: 0.1336306209562122} |
| 11-7546977 | Karlyn | Calladine | Inspector | {hub: 0.0, auth: 0.1336306209562122} |
| 38-6261333 | Chet | Vasic | Police Constable | {hub: 0.0, auth: 0.1336306209562122} |

Al primo posto troviamo di nuovo Devy Larive con uno score maggiore. In seconda posizione a pari merito ci sono invece Gregorius Shakesby, Simmonds Greensall, Karlyn Calladin e Chet Vasic.

## 1.5. Solo gli agenti di rango maggiore (Degree Centrality)

Questa volta cerco gli agenti di rango maggiore che hanno investigato su più crimini. Userò di nuovo il degree centrality.
Cerco inizialmente tutti i possibili ranghi:

In [12]:

```
graph.run("MATCH (o:Officer) RETURN COLLECT(DISTINCT o.rank) AS
tutti_i_ranghi")
```

Out[12]:

| tutti_i_ranghi |
|---|
| ['Chief Inspector', 'Inspector', 'Police Constable', 'Sergeant'] |

Dopo una breve ricerca su Internet (https://en.wikipedia.org/wiki/Police_ranks_of_the_United_Kingdom#Great_Britain) si può capire che il rango maggiore è Chief Inspector. Vado quindi a selezionare solo tali agenti:

In [13]:
```
if graph.run("call gds.graph.exists('officers-chief')").data()[0]["exists"]:
  graph.run('call gds.graph.drop("officers-chief")')

cq = """
CALL gds.graph.project.cypher('officers-chief',
  'MATCH (o:Officer|Crime) WHERE o.rank="Chief Inspector" OR o.rank IS NULL
RETURN DISTINCT id(o) as id',
  'MATCH (o:Officer)<--(c:Crime) RETURN  id(o) AS source, id(c) as target',
  {validateRelationships:false}
);
"""

graph.run(cq)
```

Out[13]:

| nodeQuery | relationshipQuery | graphName | nodeCount | relationshipCount | projectMillis |
|---|---|---|---|---|---|
| MATCH (o:Officer\|Crime) WHERE o.rank="Chief Inspector" OR o.rank IS NULL RETURN DISTINCT id(o) as id | MATCH (o:Officer)<--(c:Crime) RETURN id(o) AS source, id(c) as target | officers-chief | 28852 | 2500 | 47 |

In [14]:
```
# Memory estimation
graph.run("""
CALL gds.degree.write.estimate('officers-chief', { writeProperty: 'betweenness'
})
YIELD  nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory
RETURN nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory ;
""" )
```

Out[14]:

| nodeCount | relationshipCount | bytesMin | bytesMax | requiredMemory |
|---|---|---|---|---|
| 28852 | 2500 | 56 | 56 | 56 Bytes |

In [15]:
```
# Agenti di polizia con il rango maggiore (Chief Inspector) che hanno indagato
su più crimini
cq="""
CALL gds.degree.stream('officers-chief')
YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS n, score
RETURN n.badge_no as badge_number, n.name as nome, n.surname as cognome,n.rank
as rango, score
```

```
ORDER BY score DESC
LIMIT 5 ;
"""
print_n_results(cq)
```

| badge_number | nome | cognome | rango | score |
|---|---|---|---|---|
| 71-6459623 | Kort | Monelli | Chief Inspector | 40.0 |
| 14-3562764 | Urban | Stave | Chief Inspector | 39.0 |
| 49-5476687 | Roberto | Febry | Chief Inspector | 38.0 |
| 75-9485265 | Evey | Rahlof | Chief Inspector | 37.0 |
| 19-8319581 | Dottie | Syddie | Chief Inspector | 37.0 |

Possiamo quindi vedere che gli agenti di rango Chief Inspector che hanno investigato su più crimini sono: Kort Monelli, Urban Stave, Roberto Febry, Evey Rahlof e Dottie Syddie.
Si può notare anche che questi agenti non sono tra gli agenti stampati al punto 1.1 (dove non è stato tenuto conto del rango). Questo è probabilmente dovuto al fatto che i dati nel database sono casuali.

### Conclusioni

Per rispondere alla domanda iniziale, i migliori agenti sono coloro che sono stati stampati ai punti precedenti.

## 2. Esistono dei gruppi di persone pericolose? Quali sono?

Per trovare i gruppi di persone pericolose uso la community detection.

In [16]:
```
def print_in_columns(my_list):
    """ Funzione che stampa una lista su due colonne"""
    columns = 2  #if len(my_list)>10 else 3
    len_max = str(max([len(i) for i in my_list]) + 4)
    for i in range(0, len(my_list), columns):
        print("  ", end="")
        [eval('print(f"{i:'+len_max+'}", end="")') for i in
my_list[i:i+columns]]
        print("")

def print_id_gruppo(query_results):
  """ Funzione che stampa i gruppi di persone """
  for (communityId, gruppo) in query_results:
      print("---------- Gruppo "+str(communityId)+" ----------")
      # print(gruppo)
      print_in_columns(gruppo)
      print("")
```

## 2.1. Ricerca in base alle conoscenze generiche dei criminali (Algoritmo di Louvain)

In questa prima versione, cerco i gruppi di persone che hanno preso parte ad un crimine e che sono legati dalla relazione generica "KNOWS", senza quindi dare un peso al tipo di conoscenza. Per trovare i gruppi uso l'algoritmo di Louvain.

In [17]:
```
# Creo la proiezione
if graph.run("call gds.graph.exists('criminals-knows')").data()[0]["exists"]:
  graph.run('call gds.graph.drop("criminals-knows")')

cq = """
  CALL gds.graph.project.cypher('criminals-knows',
  'MATCH (p:Person)-->(:Crime) RETURN id(p) as id',
  'MATCH (criminal:Person)-[:KNOWS]-(conoscente:Person) RETURN DISTINCT
id(conoscente) as target, id(criminal) as source',
  {validateRelationships:false}
);
"""
graph.run(cq)
```

Out[17]:

| nodeQuery | relationshipQuery | graphName | nodeCount | relationshipCount | projectMillis |
|---|---|---|---|---|---|
| MATCH (p:Person)--> (:Crime) RETURN id(p) as id | MATCH (criminal:Person)-[:KNOWS]-(conoscente:Person) RETURN DISTINCT id(conoscente) as target, id(criminal) as source | criminals-knows | 29 | 30 | 12 |

In [18]:
```
graph.run("""CALL gds.louvain.write.estimate('criminals-knows', {
writeProperty: 'community' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory
RETURN nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory ;
""")
```

Out[18]:

| nodeCount | relationshipCount | bytesMin | bytesMax | requiredMemory |
|---|---|---|---|---|
| 29 | 30 | 7153 | 566272 | [7153 Bytes ... 553 KiB] |

In [19]:
```
cq = """
CALL gds.louvain.stream('criminals-knows')
  YIELD nodeId, communityId
WITH gds.util.asNode(nodeId) AS node, communityId
WITH communityId, apoc.coll.sort(collect(node.name + " " + node.surname + " ("
+ node.nhs_no+")")) AS gruppo
WHERE size(gruppo)>1
```

```
RETURN *
"""

print_id_gruppo(graph.run(cq))
```

```
---------- Gruppo 27 ----------
  Diana Murray (900-41-3309)          Jessica Kelly (311-75-6483)
  Kathleen Peters (250-75-5238)       Phillip Williamson (337-28-4424)
  Raymond Walker (879-22-8665)

---------- Gruppo 17 ----------
  Alan Ward (881-20-2396)        Brian Morales (335-71-7747)
  Jack Powell (249-54-6589)

---------- Gruppo 24 ----------
  Billy Moore (846-48-9238)         Lillian Martinez (397-28-4474)

---------- Gruppo 25 ----------
  Andrea Montgomery (351-83-4608)    Donald Robinson (873-28-7561)
```

L'algoritmo ci restituisce quattro gruppi. Analizzo il primo e cerco i conoscenti di Diana:

In [20]:
```
print_n_results("""
MATCH (diana:Person {nhs_no:"900-41-3309"})-[:KNOWS]-
(conoscenti_di_diana:Person)
RETURN conoscenti_di_diana.name, conoscenti_di_diana.surname,
conoscenti_di_diana.nhs_no, EXISTS ( (conoscenti_di_diana)-->(:Crime) ) AS
ha_precedenti
""")
```

| conoscenti_di_diana.name | conoscenti_di_diana.surname | conoscenti_di_diana.nhs_no | ha_precedenti |
|---|---|---|---|
| Kathleen | Peters | 250-75-5238 | true |
| Jessica | Kelly | 311-75-6483 | true |
| Kathy | Wheeler | 218-31-0921 | false |
| Melissa | Warren | 520-24-8922 | false |

Come si può vedere Kathleen e Jessica compaiono nel gruppo, mentre Kathy e Melissa no. Questo perché Kathleen e Jessica hanno dei precedenti penali mentre Kathy e Melissa no.
Cerchiamo di capire ora come mai compaiono anche Raymond e Phillip nel gruppo:

In [21]:
```
print_n_results("""
MATCH (raymond:Person {nhs_no:"879-22-8665"})-[:KNOWS]-
(conoscenti_di_raymond:Person)
RETURN conoscenti_di_raymond.name, conoscenti_di_raymond.surname,
conoscenti_di_raymond.nhs_no, EXISTS ( (conoscenti_di_raymond)-->(:Crime) ) AS
ha_precedenti
""")
```

| conoscenti_di_raymond.name | conoscenti_di_raymond.surname | conoscenti_di_raymond.nhs_no | ha_precedenti |
|---|---|---|---|
| Kathleen | Peters | 250-75-5238 | true |
| Phillip | Williamson | 337-28-4424 | true |

Si può notare quindi che Diana e Raymond hanno tra le conoscenze in comune Kathleen, che quindi "unisce" i due gruppi di conoscenti.

## 2.2. Ricerca in base al tipo di conoscenze (Modularity Optimization)

Per trovare i gruppi darò stavolta un peso diverso alle diverse tipologie di conoscenze. Ho deciso di dare più importanza ai conviventi (KNOWS_LW); in secondo luogo ai parenti (FAMILY_REL), poi alle persone che si sono almeno chiamate o scambiate messaggi (KNOWS_PHONE) ed infine a coloro che si conoscono sui social network (KNOWS_SN). Userò questa volta la modularity optimization.

In [22]:
```
## Creo il project
if graph.run("call gds.graph.exists('criminals-knows-pesate')").data()[0]
["exists"]:
  graph.run('call gds.graph.drop("criminals-knows-pesate")')

cq = """
  CALL gds.graph.project.cypher('criminals-knows-pesate',
   'MATCH (p:Person)-->(:Crime) RETURN id(p) as id',
   'MATCH (criminal:Person)-[]-(conoscente:Person)
    RETURN id(conoscente) as target,
    CASE
        WHEN EXISTS ( (conoscente)-[:KNOWS_LW]-(criminal) ) THEN 10
        WHEN EXISTS ( (conoscente)-[:FAMILY_REL]-(criminal) ) THEN 8
        WHEN EXISTS ( (conoscente)-[:KNOWS_PHONE]-(criminal) ) THEN 4
        WHEN EXISTS ( (conoscente)-[:KNOWS_SN]-(criminal) ) THEN 3
    END AS peso, id(criminal) as source',
  {validateRelationships:false}
);
"""
graph.run(cq)
```

Out[22]:

| nodeQuery | relationshipQuery | graphName | nodeCount | relationshipCount | projectMillis |
|---|---|---|---|---|---|
| MATCH (p:Person)--> (:Crime) RETURN id(p) as id | MATCH (criminal:Person)-[]-(conoscente:Person) RETURN id(conoscente) as target, CASE WHEN EXISTS ( (conoscente)-[:KNOWS_LW]-(criminal) ) THEN 10 WHEN EXISTS ( (conoscente)-[:FAMILY_REL]-(criminal) ) THEN 8 WHEN EXISTS ( (conoscente)-[:KNOWS_PHONE]-(criminal) ) THEN 4 WHEN EXISTS ( (conoscente)-[:KNOWS_SN]-(criminal) ) THEN 3 END AS peso, id(criminal) as source | criminals-knows-pesate | 29 | 60 | 18 |

In [23]:

```
cq = """
CALL gds.beta.modularityOptimization.stream('criminals-knows-pesate',
{relationshipWeightProperty: 'peso' })
YIELD nodeId, communityId
WITH gds.util.asNode(nodeId) AS node, communityId
WITH communityId, apoc.coll.sort(collect(node.name + " " + node.surname + " ("
+ node.nhs_no+")")) AS gruppo
WHERE size(gruppo)>1
RETURN *
"""
print_id_gruppo(graph.run(cq))
```

```
---------- Gruppo 14 ----------
  Diana Murray (900-41-3309)       Kathleen Peters (250-75-5238)
  Raymond Walker (879-22-8665)

---------- Gruppo 17 ----------
  Alan Ward (881-20-2396)        Brian Morales (335-71-7747)
  Jack Powell (249-54-6589)

---------- Gruppo 24 ----------
  Billy Moore (846-48-9238)         Lillian Martinez (397-28-4474)

---------- Gruppo 25 ----------
  Andrea Montgomery (351-83-4608)    Donald Robinson (873-28-7561)

---------- Gruppo 28 ----------
  Jessica Kelly (311-75-6483)         Phillip Williamson (337-28-4424)
```

L'unica differenza dal risultato precedente è che il gruppo 14 e 28 prima era unito. Questo perché probabilmente ci sono conoscenze più significative tra gli individui di un gruppo rispetto ai componendi dell'altro grupo.

## 2.3. Aggiungo gli abitanti della stessa città (Weakly Connected Components)

Come si può vedere esistono persone senza un legame di conoscenza che però abitano vicine.

In [24]:
```
display(graph.run("""
MATCH (p1:Person)-->(:Location)-->(:Area)<--(:Location)<--(p2:Person)
WHERE NOT EXISTS ( (p1)-[]-(p2))
RETURN COUNT(*) AS num_persone_che_non_si_conoscono_ma_che_abitano_vicine """))
```

| num_persone_che_non_si_conoscono_ma_che_abitano_vicine |
|---|
| 1800 |

Aggiungo quindi questa possibile conoscenza, tra le conoscenze già presenti, con un peso basso. Userò stavolta l'algoritmo Weakly Connected Components.

In [25]:
```
# Creo il project
if graph.run("call gds.graph.exists('criminals-knows-pesate-con-
vicini')").data()[0]["exists"]:
  graph.run('call gds.graph.drop("criminals-knows-pesate-con-vicini")')

cq = """
  CALL gds.graph.project.cypher('criminals-knows-pesate-con-vicini',
   'MATCH (p:Person)-->(:Crime) RETURN id(p) as id',
   'MATCH (criminal:Person),(conoscente:Person)
    WHERE criminal<>conoscente
    RETURN id(conoscente) as target,
    CASE
        WHEN EXISTS ( (conoscente)-[:KNOWS_LW]-(criminal) ) THEN 10
        WHEN EXISTS ( (conoscente)-[:FAMILY_REL]-(criminal) ) THEN 8
        WHEN EXISTS ( (conoscente)-[:KNOWS_PHONE]-(criminal) ) THEN 4
        WHEN EXISTS ( (conoscente)-[:KNOWS_SN]-(criminal) ) THEN 3
        WHEN EXISTS ( (criminal)-->(:Location)-->(:Area)<--(:Location)<--
(conoscente) ) THEN 1
    END AS peso, id(criminal) as source',
  {validateRelationships:false}
);
"""
graph.run(cq)
```

Out[25]:

| nodeQuery | relationshipQuery | graphName | nodeCount | relationshipCount | projectMillis |
|---|---|---|---|---|---|
| MATCH (p:Person)-->(:Crime) RETURN id(p) as id | MATCH (criminal:Person), (conoscente:Person) WHERE criminal<>conoscente RETURN id(conoscente) as target, CASE WHEN EXISTS ( (conoscente)-[:KNOWS_LW]-(criminal) ) THEN 10 WHEN EXISTS ( (conoscente)-[:FAMILY_REL]-(criminal) ) THEN 8 WHEN EXISTS ( (conoscente)-[:KNOWS_PHONE]-(criminal) ) THEN 4 WHEN EXISTS ( (conoscente)-[:KNOWS_SN]-(criminal) ) THEN 3 WHEN EXISTS ( (criminal)-->(:Location)-->(:Area)<--(:Location)<--(conoscente) ) THEN 1 END AS peso, id(criminal) as source | criminals-knows-pesate-con-vicini | 29 | 812 | 1601 |

In [26]:

```
graph.run("""
CALL gds.wcc.write.estimate('criminals-knows-pesate-con-vicini', {
writeProperty: 'component' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory
RETURN nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory
""")
```

Out[26]:

| nodeCount | relationshipCount | bytesMin | bytesMax | requiredMemory |
|---|---|---|---|---|
| 29 | 812 | 328 | 328 | 328 Bytes |

In [27]:

```
cq = """
CALL gds.wcc.stream('criminals-knows-pesate-con-vicini', {
relationshipWeightProperty: 'peso' })
YIELD nodeId, componentId
WITH gds.util.asNode(nodeId) AS node, componentId
WITH componentId, apoc.coll.sort(collect(node.name + " " + node.surname + " ("
+ node.nhs_no+")")) AS gruppo
WHERE size(gruppo)>1
RETURN *
"""
print_id_gruppo(graph.run(cq))
```

```
---------- Gruppo 0 ----------
  Alan Ward (881-20-2396)             Amanda Robertson (455-19-0708)
  Amy Bailey (276-19-9235)            Andrea Montgomery (351-83-4608)
  Annie George (575-05-6519)          Billy Moore (846-48-9238)
  Brian Morales (335-71-7747)         Carlos Black (859-81-0332)
  Craig Marshall (890-58-5813)        David Mills (589-69-0106)
  Diana Murray (900-41-3309)          Donald Robinson (873-28-7561)
  Ernest Clark (205-52-5697)          Fred Williamson (468-82-3915)
  Gary Vasquez (679-81-5309)          Jack Powell (249-54-6589)
  Jessica Kelly (311-75-6483)         Joan Flores (247-72-6304)
  Kathleen Peters (250-75-5238)       Kenneth Carroll (252-29-4929)
  Lillian Martinez (397-28-4474)      Maria Hughes (678-06-9352)
  Michelle Patterson (600-92-0643)    Norma Payne (699-85-1673)
  Phillip Williamson (337-28-4424)    Raymond Walker (879-22-8665)
  Rebecca Long (785-79-1645)          Stephanie Hughes (821-11-2735)
  Victor Harper (220-62-1837)
```

L'algoritmo restituisce un unico gruppo di 29 persone, contenente quindi un potenziale gruppo di persone pericolose.

## 2.4. Aggiunta dei conoscenti dei criminali (Strongly Connected Components)

Proverò questa volta a formare i gruppi includendo anche i conoscenti dei criminali, che potrebbero non aver commesso reati, ma che potrebbero comunque essere considerarti pericolosi. Non darò pesi alle conoscenze. Userò stavolta l'algoritmo Strongly Connected Components.

In [28]:
```
# Creo il project
if graph.run("call gds.graph.exists('criminals-conoscenti')").data()[0]
["exists"]:
  graph.run('call gds.graph.drop("criminals-conoscenti")')

cq = """
  CALL gds.graph.project.cypher('criminals-conoscenti',
  ' MATCH (p:Person)
    WHERE
        EXISTS ( (p)-->(:Crime) )                          // Persone che
hanno preso parte a un crimine
        OR EXISTS ( (p)-[]-(:Person)-[:PARTY_TO]->(:Crime) )  // Conoscenti dei
criminali
    RETURN id(p) as id',

  'MATCH (criminal)-[:KNOWS]-(conoscente:Person) RETURN DISTINCT id(conoscente)
as target, id(criminal) as source',
  {validateRelationships:false}
);
"""
```

```
graph.run(cq)
print("Proiezione crata correttamente!")
```

```
Proiezione crata correttamente!
```

In [29]:

```
cq = """
CALL gds.alpha.scc.stream('criminals-conoscenti')
YIELD nodeId, componentId
WITH gds.util.asNode(nodeId) AS node, componentId
WITH componentId, apoc.coll.sort(collect(node.name + " " + node.surname + " ("
+ node.nhs_no+")")) AS gruppo
WHERE size(gruppo)>1
RETURN *
"""
print_id_gruppo(graph.run(cq))
```

```
---------- Gruppo 0 ----------
  Alan Ward (881-20-2396)                Amanda Robertson (455-19-0708)
  Amy Bailey (276-19-9235)               Amy Murphy (367-54-3328)
  Andrea George (800-46-2184)            Andrea Montgomery (351-83-4608)
  Andrea Moreno (240-77-5251)            Ann Fox (576-99-9244)
  Anna Chapman (878-32-2595)             Anne Freeman (804-54-6976)
  Annie Duncan (863-96-9468)             Annie George (575-05-6519)
  Arthur Willis (271-78-8919)            Ashley Robertson (554-93-4466)
  Billy Moore (846-48-9238)              Bobby Russell (680-93-7668)
  Bonnie Gilbert (622-53-3302)           Brandon Martin (853-69-5350)
  Brenda Edwards (778-24-6852)           Brian Morales (335-71-7747)
  Carl Fuller (358-70-5810)              Carl Lawrence (271-53-9609)
  Carlos Black (859-81-0332)             Carlos Matthews (896-25-8370)
  Catherine White (270-75-5897)          Charles Alexander (820-74-9970)
  Christopher Patterson (256-31-7892)    Craig Marshall (890-58-5813)
  David Mills (589-69-0106)              Denise Brown (335-36-7489)
  Dennis Bradley (817-89-6264)           Dennis Mcdonald (442-54-3651)
  Diana Murray (900-41-3309)             Diane Bradley (338-31-6051)
  Donald Robinson (873-28-7561)          Ernest Clark (205-52-5697)
  Ernest Thompson (918-23-4923)          Evelyn Wood (484-38-1830)
  Fred Williamson (468-82-3915)          Gary Vasquez (679-81-5309)
  Harry Lopez (915-75-5600)              Henry Coleman (706-24-9599)
  Henry Jacobs (348-63-8190)             Howard Day (806-49-7942)
  Jack Powell (249-54-6589)              James Hudson (899-42-6637)
  Janet Cunningham (358-19-2542)         Jennifer Murray (653-64-4372)
  Jennifer Rogers (534-62-8298)          Jessica Kelly (311-75-6483)
  Joan Flores (247-72-6304)              Jose Green (383-26-3713)
  Justin Payne (438-49-1893)             Kathleen Peters (250-75-5238)
  Kathryn Allen (883-98-9193)            Kathy Wheeler (218-31-0921)
  Kelly Peterson (434-76-7869)           Kelly Robertson (585-01-6112)
  Kenneth Carroll (252-29-4929)          Lillian Martinez (397-28-4474)
  Linda Baker (675-57-0293)              Linda Boyd (696-28-4220)
  Lois Larson (691-51-9612)              Louis Richards (362-49-5861)
  Maria Hughes (678-06-9352)             Mary Murray (258-98-0727)
  Mary Young (791-62-3536)               Matthew Howell (302-08-4754)
  Matthew Phillips (569-73-6017)         Melissa Gibson (537-30-3032)
  Melissa Mills (383-09-0127)            Melissa Warren (520-24-8922)
  Michael Martin (318-22-2828)           Michelle Patterson (600-92-0643)
  Nicholas Mason (506-14-4016)           Norma Payne (699-85-1673)
  Pamela Gibson (838-11-7607)            Patricia Butler (895-45-1258)
  Paul Arnold (633-71-2217)              Philip Gardner (775-75-2532)
  Philip Scott (660-62-5107)             Phillip Williamson (337-28-4424)
  Rachel Hunter (568-22-7686)            Raymond Walker (879-22-8665)
  Raymond Williamson (452-95-4283)       Rebecca Lee (450-68-4090)
  Rebecca Long (785-79-1645)             Roger Brooks (288-59-8593)
  Rose Crawford (673-28-6979)            Rose Parker (545-33-7212)
  Ryan Smith (867-78-8919)               Sandra Ruiz (640-26-0925)
  Sean Myers (917-80-4656)               Stephanie Hughes (821-11-2735)
  Theresa Powell (260-95-0836)           Victor Harper (220-62-1837)
  Virginia Allen (910-33-1979)           Virginia Gibson (314-45-9119)
  Walter James (329-25-8167)             Wanda Weaver (543-43-9738)
  William Dixon (763-27-9872)
```

L'algoritmo restituisce un unico gruppo contenente le persone stampate precedentemente ed i loro conoscenti, indicando quindi la presenza di una relazione tra essi.
Questo fornisce una risposta ancora più precisa alla seconda research question.

# 3. Esiste un collegamento tra persone con precedenti per spaccio e altre persone pregiudicate?

Cercherò stavolta i possibili collegamenti tra le persone con precedenti per spaccio e altre persone pregiudicate, al fine di cercare altri possibili spacciatori. Per trovare tali collegamenti userò gli algoritmi di Shortest Path.

Prendo intanto due persone: una coinvolta in un crimine di tipo Drugs e l'altra pregiudicata ma che non abbia mai compiuto un crimine Drugs.

In [30]:
```
cq = """
MATCH (p1:Person)-->(c1:Crime {type:"Drugs"}), (p2:Person)-->(c2:Crime)
WHERE
    NOT EXISTS ( (p2)-->(:Crime {type:"Drugs"} ))
    AND NOT EXISTS ( (p1)-[]-(p2))
RETURN p1.nhs_no AS spacciatore_nhs_no, p1.name AS spacciatore_name, p2.nhs_no
AS pregiudicato_nhs_no, p2.name AS pregiudicato_name
LIMIT 1
"""
display(graph.run(cq))

data = graph.run(cq).data()[0]
spacciatore_nhs_no = data["spacciatore_nhs_no"]
pregiudicato_nhs_no = data["pregiudicato_nhs_no"]
```

| spacciatore_nhs_no | spacciatore_name | pregiudicato_nhs_no | pregiudicato_name |
|---|---|---|---|
| 879-22-8665 | Raymond | 821-11-2735 | Stephanie |

Creo ora una proiezione con pesi diversi, in base alla relazione che unisce due nodi. In questo modo riesco a trovare il collegamento più importante che unisce due persone. Tutte le relazioni in ordine di importanza sono:

- CURRENT_ADDRESS, OCCURRED_AT, INVOLVED_IN, PARTY_TO, KNOWS_LW. In questo modo do più importanza alle persone conviventi, ai crimini, ai luoghi dei crimini e alle abitazioni delle persone.
- HAS_POSTCODE, FAMILY_REL. In questo modo do un'importanza secondaria (ma comunque alta) ai parenti e alle persone che vivono nella stessa strada.
- POSTCODE_IN_AREA, LOCATION_IN_AREA. In questo modo do un'importanza leggermente minore alle persone che vivono nella stessa città.
- HAS_PHONE, KNOWS_PHONE, CALLER, CALLED. Do quindi meno importanza alle persone che si conoscono solo per uno scambio di chiamate o messaggi.
- KNOWS_SN. Il legame meno importante è quello tra i social network.

Si fa notare che a differenza dei casi al punto 2, in questo caso più l'importanza di una relazione cresce, più il peso associato diminuisce. Questo perché stavolta vengono usati algoritmi di Shortest

Path che cercano il percorso con peso minore.

In [31]:

```
# Creo la proiezione
if graph.run("call gds.graph.exists('drugs-groups')").data()[0]["exists"]:
  graph.run('call gds.graph.drop("drugs-groups")')

cq = """
  CALL gds.graph.project.cypher(
    'drugs-groups',
    'MATCH (p) RETURN id(p) as id',
    'MATCH (a)-[r]-(b)
     RETURN id(a) as source, id(b) as target,
      CASE

        WHEN EXISTS ( (a)-[:CURRENT_ADDRESS]-(b) ) THEN 1
        WHEN EXISTS ( (a)-[:OCCURRED_AT]-(b) ) THEN 1
        WHEN EXISTS ( (a)-[:INVOLVED_IN]-(b) ) THEN 1
        WHEN EXISTS ( (a)-[:PARTY_TO]-(b) ) THEN 1
        WHEN EXISTS ( (a)-[:KNOWS_LW]-(b) ) THEN 1

        WHEN EXISTS ( (a)-[:FAMILY_REL]-(b) ) THEN 2
        WHEN EXISTS ( (a)-[:HAS_POSTCODE]-(b) ) THEN 2

        WHEN EXISTS ( (a)-[:POSTCODE_IN_AREA]-(b) ) THEN 3
        WHEN EXISTS ( (a)-[:LOCATION_IN_AREA]-(b) ) THEN 3

        WHEN EXISTS ( (a)-[:HAS_PHONE]-(b) ) THEN 4
        WHEN EXISTS ( (a)-[:KNOWS_PHONE]-(b) ) THEN 4
        WHEN EXISTS ( (a)-[:CALLER]-(b) ) THEN 4
        WHEN EXISTS ( (a)-[:CALLED]-(b) ) THEN 4

        WHEN EXISTS ( (a)-[:KNOWS_SN]-(b) ) THEN 5

        ELSE 10
      END AS peso, type(r) AS type',
    {validateRelationships:false}
);
"""
graph.run(cq)
```

Out[31]:

| nodeQuery | relationshipQuery | graphName | nodeCount | relationshipCount | projectMillis |
|---|---|---|---|---|---|
| MATCH (p) RETURN id(p) as id | MATCH (a)-[r]-(b) RETURN id(a) as source, id(b) as target, CASE WHEN EXISTS ( (a)-[:CURRENT_ADDRESS]-(b) ) THEN 1 WHEN EXISTS ( (a)-[:OCCURRED_AT]-(b) ) THEN 1 WHEN EXISTS ( (a)-[:INVOLVED_IN]-(b) ) THEN 1 WHEN EXISTS ( (a)-[:PARTY_TO]-(b) ) THEN 1 WHEN EXISTS ( (a)-[:KNOWS_LW]-(b) ) THEN 1 WHEN EXISTS ( (a)-[:FAMILY_REL]-(b) ) THEN 2 WHEN EXISTS ( (a)-[:HAS_POSTCODE]-(b) ) THEN 2 WHEN EXISTS ( (a)-[:POSTCODE_IN_AREA]-(b) ) THEN 3 WHEN EXISTS ( (a)-[:LOCATION_IN_AREA]-(b) ) THEN 3 WHEN EXISTS ( (a)-[:HAS_PHONE]-(b) ) THEN 4 WHEN EXISTS ( (a)-[:KNOWS_PHONE]-(b) ) THEN 4 WHEN EXISTS ( (a)-[:CALLER]-(b) ) THEN 4 WHEN EXISTS ( (a)-[:CALLED]-(b) ) THEN 4 WHEN EXISTS ( (a)-[:KNOWS_SN]-(b) ) THEN 5 ELSE 10 END AS peso, type(r) AS type | drugs-groups | 61521 | 211680 | 2757 |

In [32]:

```
# Memory estimation
print_n_results("""
MATCH (source:Person {nhs_no: '"""+spacciatore_nhs_no+"""'}), (target:Person
{nhs_no: '"""+pregiudicato_nhs_no+"""'})
CALL gds.shortestPath.dijkstra.write.estimate('drugs-groups', {sourceNode:
source, targetNode: target, writeRelationshipType: 'PATH'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory
RETURN "Dijkstra senza pesi" as algoritmo, nodeCount, relationshipCount,
bytesMin, bytesMax, requiredMemory

UNION ALL

MATCH (source:Person {nhs_no: '"""+spacciatore_nhs_no+"""'})
CALL gds.allShortestPaths.delta.write.estimate('drugs-groups', {sourceNode:
source, relationshipWeightProperty: 'peso', writeRelationshipType: 'PATH' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory
RETURN "All Shortest Paths con pesi" as algoritmo, nodeCount,
relationshipCount, bytesMin, bytesMax, requiredMemory

""" )
```

| algoritmo | nodeCount | relationshipCount | bytesMin | bytesMax | requiredMemory |
|---|---|---|---|---|---|
| Dijkstra senza pesi | 61521 | 211680 | 1984768 | 1984768 | 1938 KiB |
| All Shortest Paths con pesi | 61521 | 211680 | 1599856 | 4646760 | [1562 KiB ... 4537 KiB] |

In [33]:
```
def print_results_shortest_path(cq):
    """ Funzione che stampa i risultati dello shortest path in modo piu' leggibile """
    lista_data = graph.run(cq).data()
    for data in lista_data:
        print("----------------------")
        print("Costo totale:", data["totalCost"])
        print("Costo ad ogni nodo:", data["costs"])
        print("Shortest Path:")

        ordine_nodi = data["ordine_nodi"]
        from pprint import pprint
        for nodo in ordine_nodi:
            text = ""
            tipo_nodo = str(nodo.labels)[1:]
            diz = dict(nodo)
            if tipo_nodo == "Person":
                text = diz["name"]+ " " + diz["surname"] + " (" +diz["nhs_no"]+")"
            elif tipo_nodo == "Crime":
                text = "Reato del " + diz["date"] + ". Tipo: "+diz["type"]+". Esito: "+ diz["last_outcome"]
            elif tipo_nodo == "Location" or tipo_nodo == "Area":
                text = tipo_nodo + " = " +  str(diz)[1:-1].replace("'", "", -1)
            else:
                text = nodo
            print("\t", text)
```

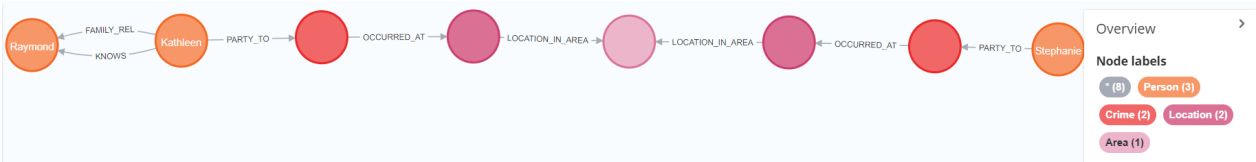
## 3.1. Trovo il collegamento usando i pesi (All Shortest Path)

Cercherò innanziutto un possibile collegamento tra le persone in base ai pesi dati precedentemente. Userò l'algoritmo All Shortest Path.

In [34]:
```
cq = """
MATCH (source:Person {nhs_no: '"""+spacciatore_nhs_no+"""'}), (target:Person {nhs_no: '"""+pregiudicato_nhs_no+"""'})
CALL gds.allShortestPaths.delta.stream( 'drugs-groups', {sourceNode: source,
```

```
relationshipWeightProperty: 'peso'})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
WITH targetNode, totalCost, [nodeId IN nodeIds | gds.util.asNode(nodeId)] AS
ordine_nodi, costs
WHERE targetNode = id(target)
RETURN totalCost, ordine_nodi, costs;
"""
# print(cq)
print_results_shortest_path(cq)
```

```
--------------------
Costo totale: 12.0
Costo ad ogni nodo: [0.0, 2.0, 3.0, 4.0, 7.0, 10.0, 11.0, 12.0]
Shortest Path:
         Raymond Walker (879-22-8665)
         Kathleen Peters (250-75-5238)
         Reato del 31/08/2017. Tipo: Vehicle crime. Esito: Investigation complete; no susp
ect identified
         Location = postcode: BL7 9DW, address: 103 Threadfold Way, latitude: 53.614334, l
ongitude: -2.426834
         Area = areaCode: BL7
         Location = postcode: BL7 9YT, address: 60 Deakins Mill Way, latitude: 53.62624, l
ongitude: -2.442588
         Reato del 16/08/2017. Tipo: Violence and sexual offences. Esito: Unable to prosec
ute suspect
         Stephanie Hughes (821-11-2735)
```




## 3.2. Trovo il collegamento senza usare i pesi (Dijkstra)

Proverò stavolta a cercare un collegamento tra Raymond e Stephanie senza usare i pesi, considerando quindi come distanza il numero di nodi. Userò l'algoritmo di Dijkstra.

In [35]:

```
cq = """
MATCH (source:Person {nhs_no: '"""+spacciatore_nhs_no+"""'}), (target:Person
{nhs_no: '"""+pregiudicato_nhs_no+"""'})
CALL gds.shortestPath.dijkstra.stream( 'drugs-groups', {sourceNode: source,
targetNode: target})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
WITH totalCost, [nodeId IN nodeIds | gds.util.asNode(nodeId)] AS ordine_nodi,
costs
RETURN totalCost, ordine_nodi, costs;
"""
print_results_shortest_path(cq)
```

```
--------------------
Costo totale: 7.0
Costo ad ogni nodo: [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0]
Shortest Path:
        Raymond Walker (879-22-8665)
        Phillip Williamson (337-28-4424)
        Brian Morales (335-71-7747)
        Brenda Edwards (778-24-6852)
        Ashley Bennett (508-10-3584)
        Diane Bradley (338-31-6051)
        Pamela Gibson (838-11-7607)
        Stephanie Hughes (821-11-2735)
```


Non sembra esserci quindi un collegamento "importante" tra Raymond e Stephanie:

- Il primo risultato mostra che un familiare di Raymond ha compiuto un reato nella stessa città in cui Stephanie ha compiuto un reato.
- Il secondo invece mostra un collegamento "distante" tante persone.

## 3.3. Collegamento tra due persone generiche (Dijkstra)

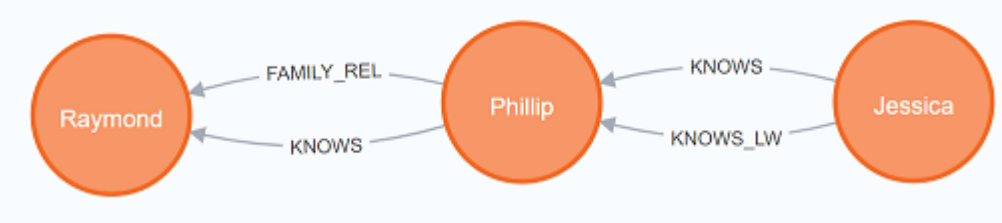
Cercherò questa volta un collegamento tra una generica persona con precedenti per droga e una generica persona con precedenti. Andrò poi a selezionare le persone che hanno il collegamento più corto:

In [36]:
```
cq = """
MATCH (source:Person)-->(:Crime {type:"Drugs"}), (target:Person)
WHERE
    EXISTS ( (target)-->(:Crime) )
    AND NOT EXISTS ( (target)-->(:Crime {type:"Drugs"}))
    AND NOT EXISTS ( (source)-[]-(target))
WITH source, COLLECT(DISTINCT id(target)) AS lista_id_target

CALL gds.allShortestPaths.dijkstra.stream('drugs-groups', {
    sourceNode: source,
    relationshipWeightProperty: 'peso'
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
WHERE targetNode IN lista_id_target
WITH totalCost, [nodeId IN nodeIds | gds.util.asNode(nodeId)] AS ordine_nodi,
costs
RETURN totalCost, ordine_nodi, costs
ORDER BY totalCost
LIMIT 1
```

```
"""
print_results_shortest_path(cq)
```

```
--------------------
Costo totale: 3.0
Costo ad ogni nodo: [0.0, 2.0, 3.0]
Shortest Path:
         Raymond Walker (879-22-8665)
         Phillip Williamson (337-28-4424)
         Jessica Kelly (311-75-6483)
```




Questa volta il collegamento è molto più importante: Raymond (condannato per droga) ha un parente che abita con Jessica (pregiudicata).