Advanced Data Management Project 2017/2018

Fighting Crime With Big Data

Contributors: Viktoriya Bodnar, Yasser Abou Haila

PART II

INSPIRED BY

https://datafloq.com/read/los-angeles-police-department-predicts-fights-crim/279

TECHNOLOGIES

First of all, since our data is dynamic (schema can vary a lot by adding new objects, new attributes, new relations and we need an efficient way to handle this) we cannot rely on traditional databases. Secondly, we have a lot of data (~4000 crimes, ~3000 criminals, ~3000 victims, ~3000 police departments, hundreds of cities, ~100 weapons and a number of relationships between these objects; and all these data correspond to just one month - December 2015 – and as you can imagine the yearly data can be really enormous). Thus, working with a huge amount of data needs a technology than can cope with it without any difficulty, but generally RDBMS have performance issues when working with large datasets. Thirdly, we give higher priority for analyzing the relationships between objects (criminals, victims, cities, weapons and police) than the objects itself. We would like to go deeper into interconnections among criminals and cities (for instance, find some patterns between city and the number of criminals in order to take some decisions to overcome this problem), crimes and weapons used (to see which weapon should be forbidden by law if it is still not), criminals and victims (if they are from the same city, if they are neighbors and so on),the PD functionality (if there are delays between when the crime was committed and when it was reported). In general our analysis is a mixture of operational and simple 'analytical-style' queries. We are not interested in very complex multilevel analysis, we just need a highly available database that can provide a good performance in adding

new objects, fast reads and gives the opportunity to visualize the results in order to have an intuition what is going on in our data. For more complex analysis of the data we can always rely on OLAP SQL. If we want perform different types of analysis at the same time(which is not our main goal), we can implement polyglot persistence in order to fulfill all the requirements. All of these lead us to conclude that the graph database is a reasonable option for above analysis. Why? Below, we state some very important reasons for that:

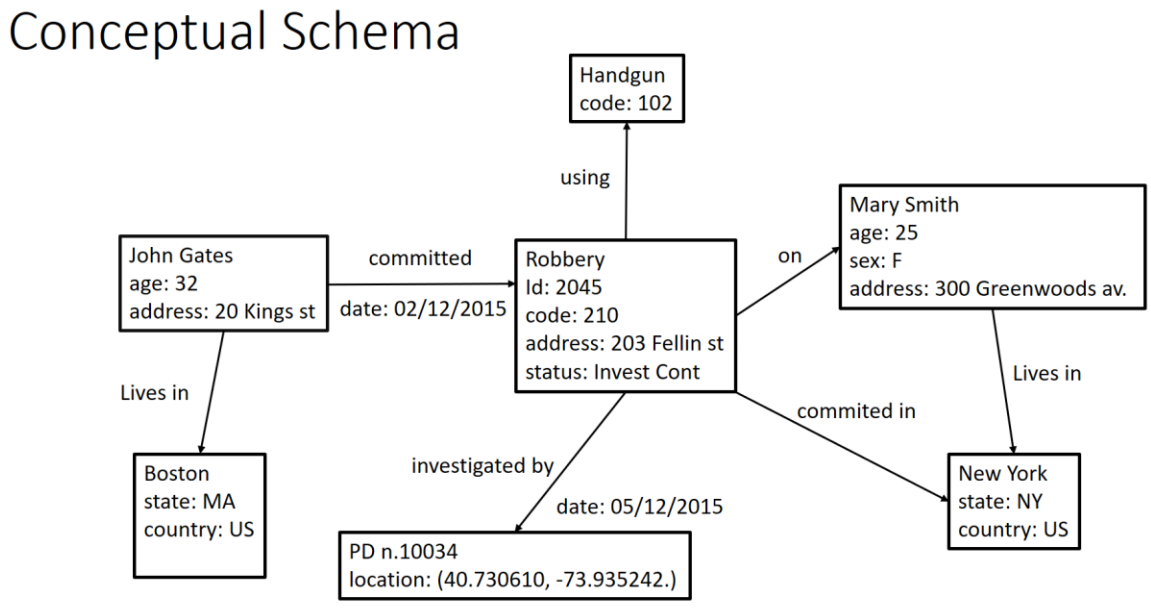
1. Analyzing the relationships in a graph is easier and it saves a lot of time in making sense of various data (can be also seen as a cost reduction for the company).
2. The way graph databases organize and store information helps to maintain the connectedness of multiple entities enabling computers to interpret related items in a context instead of just match words. Thus machines are able to store, manage and retrieve information based on meaning and logical relations.
3. This relationship-centered storage of information is particularly useful to analyze huge amounts of disparate data to identify patterns and obtain insights.

We could have implemented the graph structure with NoSQL solutions, but relationships between objects are not inherently supported because there are no joins. As a result, we use Neoj4, which is highly scalable open source technology whose popularity is growing day by day. In addition, the logical data model is extremely simple and can be seen as a generalization of the conceptual model (both are represented as graphs that facilitates the translation to 'tables'). Moreover, it has SQL like easy language CQL. However, it needs to be mentioned that it is quite difficult to implement partitioning in graph models because it is a hard NP-problem due to a very complicated 'between-clusters' traversal. As Neo4j does not provide a possibility to do that we will consider this only at the theoretical level. In case our data is so big that we need to do partitioning, we would ensure the approach of minimal point cut. It means that the group of nodes that are frequently queried together would be on the same cluster, and the number of relationships between clusters would be as small as possible. Since it is highly unusual to traverse an entire graph, most of the queries would operate inside a cluster and will still give a good performance. On the other hand, 'inter-clusters'

queries will require more effort and more time. Meanwhile, replication is generally possible by storing the entire graph on different machines and serving many requests against that copy quickly. Furthermore, the system supports ACID and comes with a web-based administration tool that includes full transaction support. The above information leads us to conclude that Neo4j relies on AC properties (Availability + Consistency). Another advantage is visual graph explorer, which is very useful to see links between the nodes. To sum up, even if Neo4j has its limitation in sharding and some side effects from deletion operation, it is being improved very often in order to overcome them and the benefits it brings have easily outweighed these drawbacks.

CONCEPTUAL DESIGN

In order to better represent our database we defined a conceptual schema. Given the fact, that we are developing graph database, the conceptual schema is also a graph. The graph can be defined as a set of nodes connected by arcs. Since, we have not only nodes that correspond to main entities and relationships, but also labels and properties, this type of graph is called a property graph and is defined as follows:



Rectangles represent entities and the arrows between them are relationships. Inside the boxes there are the attributes that describe the instance of particular entity. The relationships also have some properties that reside on the arcs they are related to. As we can see, the criminal John Gates who lives in Boston, committed a robbery on 02/12/2015. The crime was carried out using a handgun in New York. The Victim is 25 years old woman from New York. The investigation was held by one of the local police departments which registered the crime on 05/12/2015. The difference in dates can be seen as a delay of the system or can be explained by personal reasons why the victim did not report the act of crime immediately.

WORKLOAD

As we described previously our workload is a mixture of operational and basic analytical tasks. We are highly interested in relationships between the objects and as a results our workload mainly consists of read queries that rely on relationship traversal of the graph. Since, the database can be extended by adding new objects and of course new data, we will also perform some update and insert operations that are based on interconnectivity of our entities.

Below you can find the most frequent queries that we are interested in defined in natural language:

Q1. What are 5 cities where the most number of crimes occurred?

Q2. What are top 3 most used guns?

Q3. Retrieve number of crimes in a given city.

Q4. Compute the number of crimes and its description for all crime codes ordered by number of crimes.

Q5. What is the state where most criminals live?

Q6. Retrieve all victims under 18.

Q7. Number of female and male victims grouped by city.

Q8. Find the PD that investigated the biggest number of crimes.

Q9. What is the average age of criminals?

Q10. What is the delay between committed and reported crime (last crime registered in the system)

Q11. Find the total number of crimes around the country for the current month (December 2015)

Q12. Retrieve the address of the victim who was involved in the crime with a given crimeid.

Q13. Find all relationships of a given victim (show all entities and relationships)

Updates:

Q14. Add a new victim to a given crime.

Q15. Change the status of the investigation to “Arrest” given a crimeid.

Q16. Update the department location that was moved to another place.

Q17. Add the police department that investigates a given crime.

Extras:

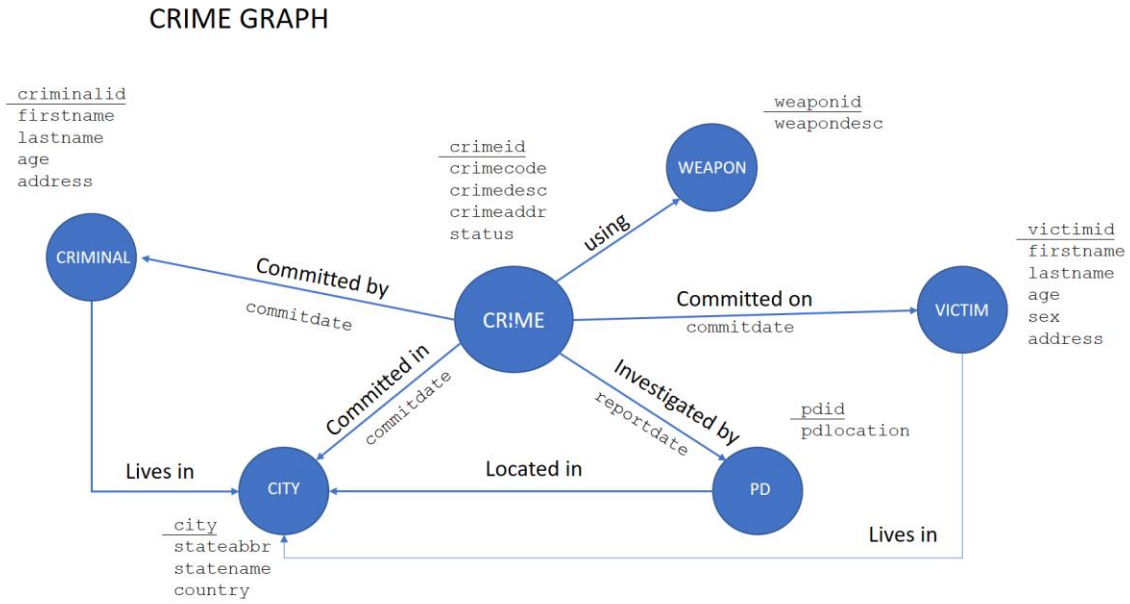
E1. Add a witness who saw a particular crime.

E2. Add a new attribute fingerprint for a given criminalid.

E3. Add a date when a particular PD finished the investigation.

(* possible deletion: Delete the department that was closed given its location. Deleting the police department leads to removing all its connections with crime, so it will be no more possible to see it as a participant in crime investigation which contradicts our previously defined requirement for keeping track all working departments)

LOGICAL DESIGN



Each node has its label (the name, e.g. CRIME, VICTIM…) and a set of properties (crimecode, crimedesc, firstname…). The properties that should be unique and act as a key are underlined (victimid,crimeid, pdid, city, criminalid). The relationships have also their own properties, which are in our case the commitdate or reportdate (to keep track of time aspect). In order to create a relationships the ids were inserted into CRIME node (victimid,pdid, city,criminalid). They are somehow represent the “foreign keys”.

IMPLEMENTATION DETAILS

Based on the fact that we are using Neo4j technology, the workload would be executed using CYPHER query language, which allows us to perform reads, updates and also deletion operations. Since this language is declarative, we only need to say what we want without specifying how. It is SQL like language which in still in development but very easy to use.

Q1. A simple use of match operator to find a pattern crime-committed in-city, ordering in descending way by number of crimes computed by count function and limit the output to top 5 results.

Q2. In this query most used gun means the gun with which the most number of crimes were committed. Here we are going to find pattern crime- using -weapon by matching the rest is the same approach as in the above query.

Q3. Match crimes and the city, where clause to select specific city and count function to find the number of crimes

Q4. Use of match operator to find crimes, count function to find number of crimes per crime code, results in descending order by the result of count function.

Q5. Match operator for criminal- lives in-city pattern, count function for finding total number of criminal and order by this number in descending way, the state with the highest number will be the first in the table, so enough to limit the output to 1.

Q6. Simple match victim and where selection on age.

Q7. Since our graph database can have an arbitrary number of properties for each entity, we can also have some missing values inside properties. To deal with empty and missing properties, we need to specify in selection that sex to return must be F or M. Order by crime city and sex for better visualization. The rest is quite trivial.

Q8. The same approach as in Q2.

Q9. Match criminal, avg function is going to be used to compute the average. The round also can be considered to have an integer value.

Q10. Giving the fact that in neo4j there is no type for dates, we will be using string representation. The actual delay (reporteddate – commitdate) in days and months can be calculated in case the year is the same. The last registered crime would be considered the reporteddate with the closest date to now. In order to find that, we just order dates in descending order and taka the first one by limit operation.

Q11. Here we are gonna match the pattern crime-committed in – city and crime-investigated by – pd, the selection would be on country, and in order to find

December 2015 we need to take a substring of the original date string and convert it to the integer for comparing with 12(December) and 2015 (year).

Q12. Usual match pattern on crime-committed on = victim with selection on crimeid.

Q13. To find all the relationships and nodes connected to a particular victim, we need to create a complete pattern following the logical schema.

Q14. In order to add a new victim to existing crime we just need to create a node with victim labels and its properties (they can be different from the usual). But that is not enough, we also need to create a relationship between the specific crime node and victim node created recently. At the end, the property for that relationship can be set (not necessarily).

Q15. Change the status with the use of set operator.

Q16. The location is changed using set clause.

Q17. To add new police department the procedure is the same as for Q14:creation of a new node, creation of a new relationship, setting the property.

E1. We need to create a new node with a new label Witness. Then the relationship seenby is created connecting the crime and created witness.

E2. Here we represent a fingerprint as a float number, however it can be actually stored differently. We just use the set operator and a new property name fingerprint.

E3. When the PD finishes its investigation, usually it also needs to change the status of the crime. In order to do that we have to set simultaneously a new property finishdate and the crime status. (*(Starting date can be considered as reporting date, with exceptions where we have more PD that work for the same crime)

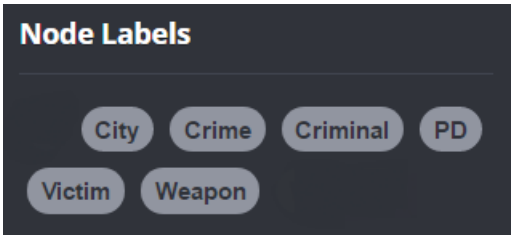
CLUSTERING

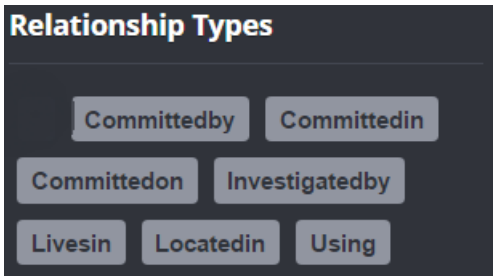
Due to the fact that clustering is only possible in Enterprise edition of Neo4j, we are not going to implement it. However, a possible protocol can be Master-Master replication because we require highly available database and fast replies without

intermediate point. Any data (graph or index) created on one replica is propagated to the others depending on the settings like number of replicas to actively push to during commit and frequency of pulling updates from the other replicas. Of course, in this scenario we can have some write conflicts as concurrent writes, but it is hardly possible that two DB admins from different departments will be inserting the same crime with different information. Usually, the crime is investigated by one police department that is responsible for data input into the system. In addition, the replication is asynchronous as we want to have a high throughput and availability. Because of the asynchronousness, our replica will become consistent only at some point, which means to have an eventual consistency. Finally, Neo4j clusters are fault tolerant that ensures the reliability of the system.

INSTANCE OF DATASET

In order to create our final dataset we had to clean, merge and transform our data. We did that following ETL approach. First of all, we extracted the data of interest by deleting irrelevant columns of original datasets in Excel. Then we performed some transformations in order to make the dataset correspond to neo4j format and get rid of noise (whitespaces trimming, date common format, upper/lower case problems, missing values inside important fields…). The cleaning part was done mainly in Trifacta. As a result we separated the dataset due to each entity where each row is an instance (For example: each row in crime represents the crime event and is described not only by its details but also by foreign keys like criminalid, victimid, pdid and city; each row in the criminals dataset represents one criminal and all its information…). The load phase was performed using cypher ‘load from csv’ command, which imports already prepared datasets into nodes. Indexes then were added in order to speed up queries, joins and relationship creation. The relationships were added using match (node)-[r]->(node) patterns. As a results, we obtained six node entities, seven relationships and a set of properties:





We created the above schema as follows (importing shared csv files from Google Drive directory)

PD

LOAD CSV WITH HEADERS from 'https://docs.google.com/spreadsheets/d/e/2PACX-1vQa5jpBaD0EebP6ISNvjdSNVplS1F25sV3GBZVF6xiU0tQqCcZ_cLnAokWtMhlo-riATU7fA14-

5bYx/pub?output=csv' as pds FIELDTERMINATOR ',' create(p:PD{pdid:toFloat(pds.PdId), pdcity:pds.City,pdlocation:pds.Location});

CITY

LOAD CSV WITH HEADERS from 'https://docs.google.com/spreadsheets/d/e/2PACX-1vSCy5KVPd6Yokhb9ULFenQUAk3euivdY_SkSutwR0k3IdHNI_RFCPElmZVd_My32mf7mKoqn8XARWdD/pub?output=csv' as cities

FIELDTERMINATOR ','

create(c:City{city:cities.city, stateabb:cities.state,statename:cities.stateName, country:cities.Country});

CRIME

LOAD CSV WITH HEADERS from 'https://docs.google.com/spreadsheets/d/e/2PACX-1vQHfy7jWkacX-qxqFq3GOA1NNHOhuqozx4i4KPkfdc3dvYydfcZpwio0JCTan1RCekKXPc0xppmM5li/pub?output=csv ' as crimes

FIELDTERMINATOR ','

create(c:Crime{crimeid:toFloat(crimes.DR_Number), datereport:crimes.Date_Reported,dateoccurred:crimes.Date_Occurred, crimecode:toInteger(crimes.Crime_Code),crimedesc:crimes.Crime_descr,weaponid:toInteger(crimes.Weapon_Used_Code),crimeaddr:crimes.Address,crimecity:crimes.City,victimid:toFloat(crimes.VictimId),criminalid:toFloat(crimes.CriminalId),pdid:toFloat(crimes.PdId),status:crimes.Status_Description});

WEAPON

LOAD CSV WITH HEADERS from 'https://docs.google.com/spreadsheets/d/e/2PACX-1vTAM36PJbVVts0ptOLJ9Tfm8ted0U014Mp7Cuw1scsDQlcwnPOvVdbmreb9Nr8lReTz6vE2eXV4HV75/pub?output=csv' as w

FIELDTERMINATOR ','

create(a:Weapon{weaponid:toInteger(w.Weapon_Used_Code), weapondesc:w.Weapon_Description});

VICTIM

LOAD CSV WITH HEADERS from 'https://docs.google.com/spreadsheets/d/e/2PACX-1vQ2AH9l4sW2V0oE3PC8Jsk0l3NifXWt_hZYHgkqwJBasE6jYYqG1iB25dQ_y3_yqUtZ8ctkieIRt05i/pub?output=csv' as c

FIELDTERMINATOR ','

create(v:Victim {victimid:toFloat(c.Victimid),age:toInteger(c.Victim_Age),sex:c.Victim_Sex, firstname:c.first_name,lastname:c.last_name,address:c.address,city:c.city});

CRIMINAL

LOAD CSV WITH HEADERS from 'https://docs.google.com/spreadsheets/d/e/2PACX-1vTVNBMRGoMONqaRnrj9ys3Jk3spmRnYVx8szRGwhWsnHUgBhVM3JpshDRdyobAfvTzNdIBwDdmzWOU3/pub?output=csv' as c

FIELDTERMINATOR ','

create(v:Criminal{criminalid:toFloat(c.CriminalId),lastname:c.LastName ,firstname:c.FirstName, age:toInteger(c.Age),address:c.Street,city:c.City});

INDEXES:

create index on :PD(pdid);

create index on :City(city);

create index on :Crime(crimeid);

create index on :Weapon(weaponid);

create index on :Victim(victimid);

create index on :Criminal(criminalid);

RELATIONSHIPS:

COMMITEDBY

match(a:Criminal) with a match (b:Crime)

where a.criminalid=b.criminalid create (b)-[r:Committedby {commitdate: b.dateoccurred}]->(a)

return r;

COMMITEDON

match(a:Crime) with a match (b:Victim)

where a.victimid=b.victimid create (a)-[r:Committedon {commitdate: a.dateoccurred}]->(b)

return r;

USING

```
match(a:Crime) with a match (b:Weapon)
where a.weaponid=b.weaponid create (a)-[r:Using]->(b)
return r;
```

LIVESIN

```
match(a:Criminal) with a match (b:City)
where a.city=b.city create (a)-[r:Livesin]->(b)
return r;

match(a:Victim) with a match (b:City)
where a.city=b.city create (a)-[r:Livesin]->(b)
return r;
```

COMMITEDIN

```
match(a:Crime) with a match (b:City)
 where a.crimecity=b.city create (a)-[r:Committedin{commitdate: a.dateoccurred}]->(b)
return r;
```

LOCATEDIN

```
match(a:PD) with a match (b:City)
where a.pdcity=b.city create (a)-[r:Locatedin]->(b)
return r;
```

INVESTIGATEDBY

```
match(a:Crime) with a match (b:PD)
where a.pdid=b.pdid create (a)-[r:Investigatedby {reporteddate:a.datereport}]->(b)
return r;
```

(*Cleanup

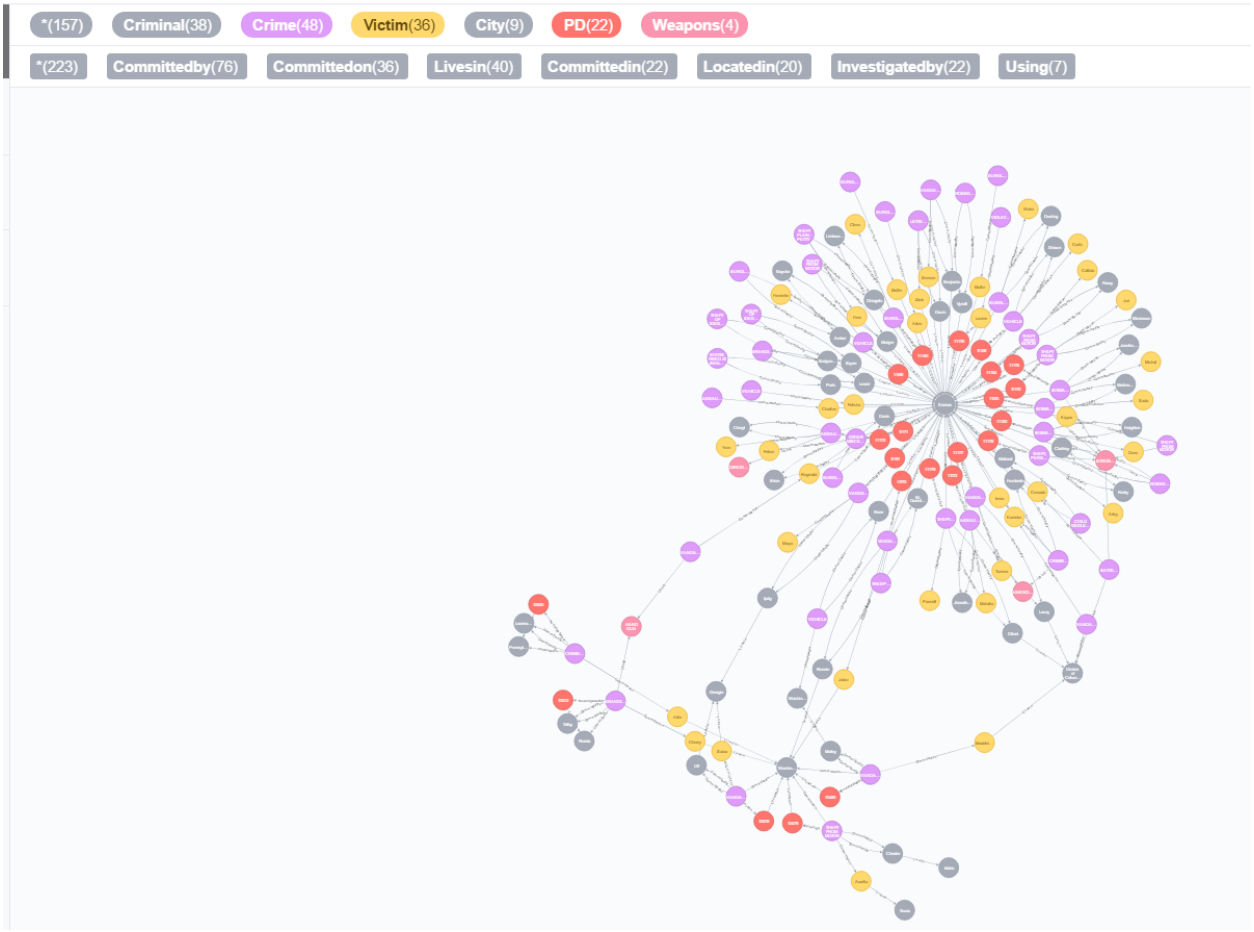
match (n:Crime) remove n.datereport, n.dateoccurred, n.criminalid, n.victimid, n.pdid, n.crimecity, n.weaponid;

match (n:Criminal) remove n.city;

match(n:Victim) remove n.city;

match (n:PD) remove n.pdcity;)

In the figure below you can see a portion of the created graph:



WORKLOAD IMPLEMENTATION

Here we reported the implementation of the queries defined above using CYPHER language and its operators:

#Q1

```
match(a:Crime)-[r:Committedin]->(b:City) return b,count(a) as numcrimes order by numcrimes desc limit 5;
```

#Q2

```
match(a:Crime)-[r:Using]->(w:Weapon) return w,count(a) as weaponused order by weaponused desc limit 3;
```

#Q3

```
match(a:Crime)-[r:Committedin]-(c:City) where c.city = 'Boston' return count(a) as crimecount;
```

#Q4

```
match(a:Crime) return a.crimecode,count(a) as crimecount,a.crimedesc order by crimecount desc;
```

#Q5

```
match(a:Criminal)-[r:Livesin]->(c:City) return c.stateabb,c.statename,count(a) as totcriminals order by totcriminals desc limit 1;
```

#Q6

```
match(v:Victim) where v.age<18 return v;
```

#Q7

```
match (v:Victim)-[r:Committedon]-(c:Crime)-[r1:Committedin]-(d:City) where v.sex='M' or v.sex ='F' return d.city,v.sex, count(v) as totvictims order by d.city,v.sex;
```

#Q8

```
match (p:PD)<-[r:Investigatedby]-(a:Crime) return p,count(a) as crimecount order by crimecount desc limit 1;
```

#Q9

```
match (c:Criminal) return round(avg(c.age));
```

#Q10

```
match ()-[q:Investigatedby]-(c:Crime)-[r:Committedby]-() return
q.reporteddate,r.commitdate order by q.reporteddate desc limit 1;
```

#Q11

```
match ()-[r:Investigatedby]-(c:Crime)-[r1:Committedin]->(b:City) where b.country =
'US' and toInteger(substring(r.reporteddate,0,2)) = 12 and
toInteger(substring(r.reporteddate,6,4)) = 2015 return count(c) as crimecount;
```

#Q12

```
match (c:Crime)-[r:Committedon]-(v:Victim) where c.crimeid = 150519716 return
v.address;
```

#Q13

```
match (q:City)<-[l:Livesin]-(v:Victim{victimid:3700 })<-[r:Committedon]-(a:Crime)-
[w]->(t)-[e]->(g:City) return q,l,v,r,a,w,t,e,g;
```

#Q14

```
create
(v:Victim{victimid:3900,sex:'F',firstname:'Maria',lastname:'Lopez',age:33,city:'Fresn
o'}) return v;
```

```
match (n:Crime{crimeid:150519710}) with n match(v:Victim{victimid:3900}) create
(n)-[r:Committedon]->(v) return r;
```

```
match ()-[r1:Committedon]-(n:Crime{crimeid:150519710})-[r:Committedon]-
>(v:Victim{victimid:3900})set r.commitdate=r1.commitdate return r;
```

#Q15

```
match(n:Crime{crimeid:150519718}) set n.status = 'Arrest' return n;
```

#Q16

match (n:PD) where n.pdid = 15009854964020 set n.pdlocation = '(41.881832, -87.623177)' return n;

#Q17

create (p:PD{pdid:9999999922,location:'(-23.3445,34.5554)'}) return p;

match (c:Crime{crimeid:150519693}) with c match (p:PD{pdid:9999999922}) create (c)-[r:Investigatedby]->(p) return r;

match ()<-[r1:Investigatedby]- (c:Crime{crimeid:150519693})-[r:Investigatedby]->(p:PD{pdid:9999999922}) set r.reporteddate = r1.reporteddate return r;

#E1

create (n:Witness{witnessid:1002,firstname:'Kevin',lastname:'Sailor',address:'203 Cross st'}) return n;

match (c:Crime{crimeid:150519693}) with c match (n:Witness{witnessid:1002}) create (c)-[r:Seenby]->(n) return c,r,n;

#E2

match (c:Criminal{criminalid:122}) set c.fingerprint =4546335634524 return c;

#E3

match (c:Crime)-[r:Investigatedby]->(p:PD{pdid:15010071864020}) set r.finishdate='01/01/2016',c.status='Arrest' return r,c,p;

CONCLUSIONS

As a conclusion, we can say that the graph database provided in the following document can be efficiently used for our target application. On one hand, it provides flexibility to schema representation and possibility to add new information easily. On the other hand, it also has a great advantage in data visualization and analysis due to the use of graph model. Finally, it presents high performance in terms of reply time for connected data and it is lightweight in memory usage.