Project 2 : Graph Database

Franco Meng, 2023, May

Abstract :

The graph database project is a continuation of the data warehousing project, the same data set was used to build the graph database.
The design, ELT process, queries have all been updated from the previous relational database, in order to suit the purposes of property graph database and answer different business queries.

In comparison with traditional SQL databases, graph database has its unique capabilities:

1. Relationships: comparing with traditional SQL databases, where any relationships are presented through fact/dimension tables, and single/multiple joins, the graph database has much better and intuitive way to represent relationships. Especially with many-to-many relationships, there is no extra bridging table needed, any relationship can be easily added/deleted. It is also able to naturally model the real-world scenarios, such as social network, supply chain, fraud detection etc where involves complex relationships, much easier to represent to the wider range of audiences who has limited knowledge of traditional databases and domain knowledges.

2. Performance: Graph database has superior performance on queries, it only query the part of the graph paths where satisfy the query, as a result, graph database can perform complex and deep 'joins' and traversals through nodes and relationships, this makes graph database suitable for scenarios where relationships are critical and performance is paramount.

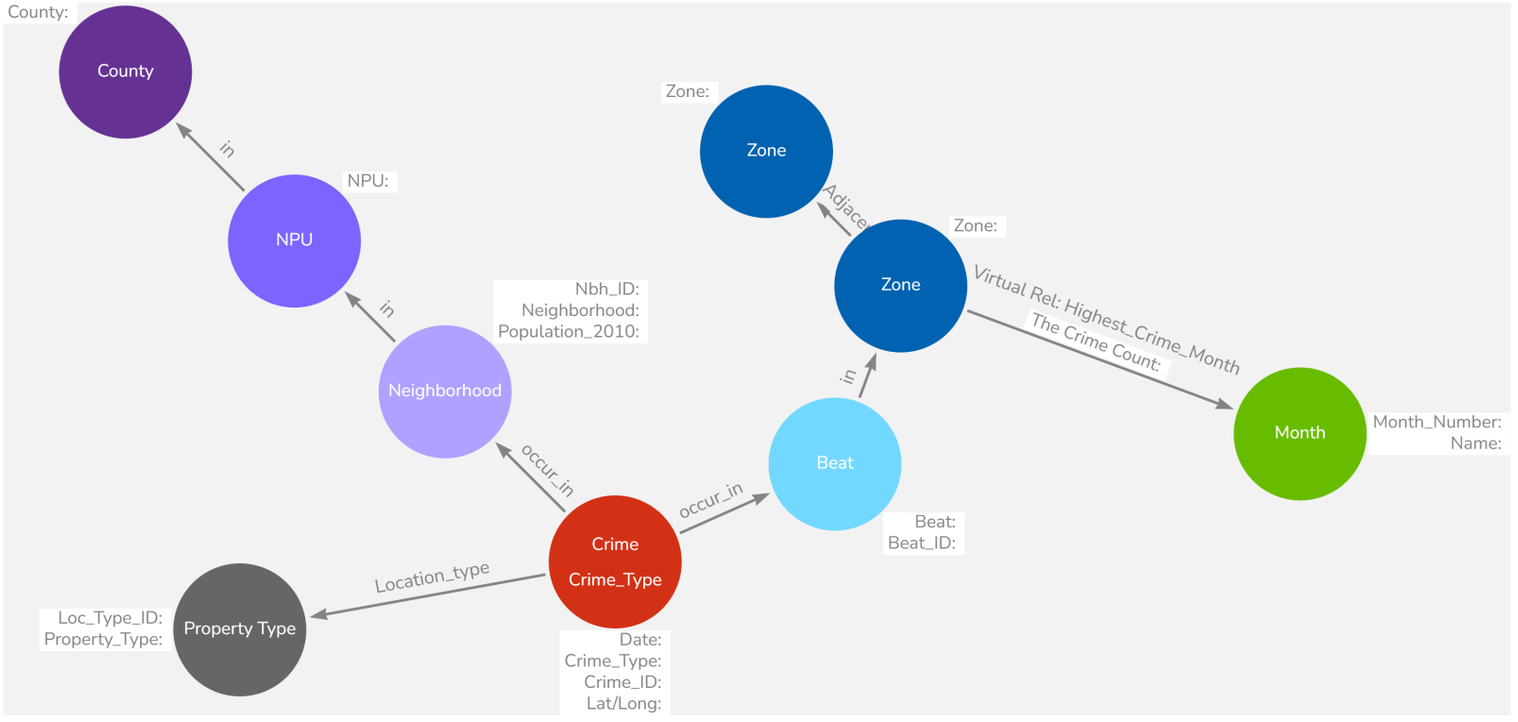
3. Schema free or flexible: Graph database structure can be easily modified and updated without specifically changing the schema, this is super beneficial in current fasting evolving modern world. Graph database do not need follow the ACID principle

4. Algorithms: Graph database provide powerful packages and algorithms for analytics and data explorations. In Neo4j Graph data science library, the algorithms including centrality, community detection, similarity, path finding, etc can be easily used to solve complex queries and reveal deep knowledge of the data. For example, what's the best path for police force to patrol through a city? Or revisit the previous crime site based on weighted importance of the crime cases? How much similarity different suburbs have in terms of population, crime rate? These can be extremely difficult or impossible to achieve in traditional database. The COLLECT, UNWIND, PATTERN COMPREHENSIONS are also extreme powerful syntax to work with lists.

5. Graph database do not need NULL value has a placeholder like in tabular form, if no information is provided, then no property, relationship will be needed.

• Design:

A property graph database was design by using the Arrows App tool. To transform the relational database into nodes, relationships, and properties.
Arrow tool was used to model the graph database, with updates from previous SQL databases. Neo4j have provided a simple easy to use ETL tool to connect and create the graph database based on the relationship database schema. However with its limitations, where similar schema will be created and each dimension table will be transform into one type of nodes. I have reprocessed the data through Python, in order to better suit the updated design of the graph database. Arrow tools allow any node, label, relationship, property creation, and can transform the design into a new 'schema' easily for graph database. Please see more details below



The red crime nodes, has two labels. **CRIME & CRIME_TYPE**
The properties are : **DATE, CRIME_TYPE, CRIME_ID, LAT/LONG**

*There was no date nodes created, as the **DATE** property of crime nodes, holds **datetime data type**, for easy and fast querying.

The green month nodes, has one label. **MONTH**
The properties are: **MONTH_NUMBER, NAME.**

*Only virtual relationships will be created for the Month Nodes, in order to clearly visualize query results in graph.

The purple nodes are, **NEIGHBORHOOD, NPU, COUNTY**, with one label each, and different properties as displayed.
The relationships are **IN**, to represent the hierarchy.

Similarly the blue nodes are **BEAT, ZONE.**
The zone has relationship **ADJACENT** within, to represent which zones are adjacent to which zones, geographically.

The grey nodes, has one label, Property Type. It represent what type of the location the crime has happened, e.g.: House, shopping mall, park etc.

Comparing with previous data warehouse, I have remodified the Python to suit the new design, one of the biggest advantages is graph database support date/ datetime data type, like Python, this facilitates the efficiency of any type queries regarding the date, in stead of creating a massive date table. It is certainly possible to create a **time tree** with different nodes representing date/month etc, but with crime database, only date only relates to one type of the node: Crimes. No date information is associated to neighborhood, zone etc.

Some extra descriptive measurement like population_2010 of each suburb have also been enriched. To pair with the dataset as this dataset only includes crime from 2010 after stratified sampling process

• ETL Process:

The Python scripts were updated from previous scripts, in order to implement the new graph databased design and to answer different business queries.

Due the low processing speed of Neo4J graph database. Nearly 3000 data entry have been extracted for the graph database.

The 29593 rows of all the 2010 crime data have been firstly extracted. **Stratified sampling** was implemented to further reduce the data to 2958 rows.
The stratified sampling was to ensure the reduced sample to be used, will still reflect the original full dataset.

Below left shows the raw percentages of each month's total numbers of crimes, divided by total crime numbers in 2010, based on raw dataset of 29593 rows.
Right shows the percentage result after sampling and reducing the dataset, based on 2958 rows, 10% of the raw dataset. The percentage results are nearly identical .

```
round(graph_df["crime"].groupby(graph_df.date.dt.month).agg('count')/len(graph_df)*100,2)
```

```
date
1      9.12
2      6.50
3      8.64
4      9.84
5     10.12
6     10.15
7     10.68
8      4.52
10    10.67
11    10.22
12     9.53
Name: crime, dtype: float64
```

```
round(graph_final['crime'].groupby(graph_final.date.dt.month).agg('count')/len(graph_final)*100,2)
```

```
date
1      9.13
2      6.49
3      8.65
4      9.84
5     10.11
6     10.14
7     10.68
8      4.53
10    10.68
11    10.21
12     9.53
Name: crime, dtype: float64
```

In order to better reflect the crime situation, a dataset of population information for each neighborhood in 2010, was retrieved from link .

The data then has been enriched in Python notebook, with this population info, as demonstrated on the right.

Population = 0 , means there was no population recorded in the table.

```
In [199]: graph_Nbh_dim_1 = pd.merge(left=graph_Nbh_dim, right=pop_df, how='left', left_on='neighborhood', right_on='Neighborhood')


graph_Nbh_dim_1['Population (2010)'] =  graph_Nbh_dim_1['Population (2010)'].fillna(0)
graph_Nbh_dim_2 = graph_Nbh_dim_1.drop(['Neighborhood'], axis=1)

graph_Nbh_dim_2 = graph_Nbh_dim_2.rename(columns={'Population (2010)': 'Population'})
graph_Nbh_dim_2
```

Out[199]:

	Nbh_ID	neighborhood	npu	county	Population
0	1	The Villages at Carver	Y	Fulton County	1039.0
1	2	Virginia Highland	F	Fulton County	0.0
2	3	Campbellton Road	R	Fulton County	4709.0
3	4	Adamsville	H	Fulton County	2403.0
4	5	Ben Hill	P	Fulton County	1725.0
...	...	...	...	...	...
195	196	Arlington Estates	P	Fulton County	776.0

Please refer Python Scripts to see other updates of the code.

Below are the sample of all the exported CSV files after the data extraction and transformation in Python, combine with Cypher codes, to demonstrate how the graph database was created.

Cypher Code
Creating **BEAT & ZONE** nodes and properties, and relationships **IN**

Favorite: 01_Load_Beat

```
//01_Load_Beat
LOAD CSV WITH HEADERS FROM 'file:///graph_beat_dim.csv' AS row
MERGE (b:Beat{Beat:toInteger(row.beat),Beat_ID:toInteger(row.beat_ID)})
MERGE (z:Zone {Zone:toInteger(row.zone)})
WITH b,z
MERGE (b) - [:in] → (z)
```

CSV Sample

```
beat_ID,beat,zone
1,305,3
2,601,6
3,410,4
4,407,4
5,414,4
6,210,2
7,408,4
8,211,2
9,206,2
10,403,4
```

Cypher Code
Creating **Neighborhood, Npu, County** nodes and properties, and relationships **IN.**

Favorite: 02_Load_Neighborhood

```
//02_Load_Neighborhood
LOAD CSV WITH HEADERS FROM 'file:///graph_Nbh_dim.csv' AS row
MERGE (nb:Neighborhood {Neighborhood:row.neighborhood,Nbh_ID:toInteger(row.Nbh_ID),Population_2010:toInteger(row.Population)})
MERGE (np:Npu {Npu:row.npu})
MERGE (cty:County {County:row.county})
WITH nb,np,cty
MERGE (nb) - [:in] → (np)
MERGE (np) - [:in] → (cty)
```

CSV Sample

```
Nbh_ID,neighborhood,npu,county,Population
1,The Villages at Carver,Y,Fulton County,1039.0
2,Virginia Highland,F,Fulton County,0.0
3,Campbellton Road,R,Fulton County,4709.0
4,Adamsville,H,Fulton County,2403.0
5,Ben Hill,P,Fulton County,1725.0
6,Pine Hills,B,Fulton County,8033.0
7,Venetian Hills,S,Fulton County,3790.0
8,Fairburn Mays,H,Fulton County,3144.0
9,Lindbergh/Morosgo,B,Fulton County,0.0
10,Buckhead Village,B,Fulton County,1343.0
11,Princeton Lakes,P,Fulton County,2429.0
12,Cascade Avenue/Road,S,Fulton County,2416.0
13,Lakewood Heights,Y,Fulton County,2177.0
14,Midtown,E,Fulton County,16569.0
15,Downtown,M,Fulton County,13411.0
```

Cypher Code
Creating **Property_Type** nodes and properties

```
Favorite: 03_Load_Property_Type*
1 //03_Load_Property_Type
2 LOAD CSV WITH HEADERS FROM 'file:///graph_Loc_type_dim.csv' AS row
3 MERGE (pt:Pro_Type {Property_Type:row.type,Loc_Type_ID:toInteger(row.Loc_type_ID) })
```

CSV Sample

```
Loc_type_ID,type
1,house_number
2,building
3,shop
4,amenity
5,tourism
6,suburb
7,road
8,neighbourhood
9,office
```

Cypher Code
Creating **BEAT** nodes and properties, Creating all the relationships with BEAT, NEIGHBORHOOD, PROPERTY_TPYE based on the FK

```
Favorite: 04_Load_Crime_and_Relationships*
1  //04_Load_Crime_and_Relationships
2  LOAD CSV WITH HEADERS FROM 'file:///graph_final.csv' AS row
3  MATCH (b:Beat {Beat_ID: toInteger(row.beat_ID)}),
4  (n:Neighborhood { Nbh_ID: toInteger(row.Nbh_ID)}),
5  (l:Pro_Type { Loc_Type_ID: toInteger(row.Loc_type_ID)})
6  MERGE (c:Crime {Crime_ID:toInteger(row.number), Crime_Type:row.crime, Date:date(row.date),
7  location:point({ latitude: toFloat(row.lat), longitude: toFloat(row.long)})})
8  WITH c,b,n,l,row
9  MERGE (c) - [:occured_in] → (b)
10 MERGE (c) - [:location_type] → (l)
11 MERGE (c) - [:occured_in] → (n)
12
```

CSV Sample

```
number,crime,lat,long,date,Nbh_ID,Loc_type_ID,beat_ID
100111262.0,Larceny-From Vehicle,33.71691,-84.39276,2010-01-11,1,1,1
100132212.0,Larceny-From Vehicle,33.77386,-84.3631,2010-01-13,2,1,2
100132067.0,Burglary-Residence,33.70103,-84.45746,2010-01-13,3,1,3
100230665.0,Burglary-Nonres,33.75811,-84.50389,2010-01-23,4,1,4
100071177.0,Auto Theft,33.67826,-84.51892,2010-01-07,5,1,5
100301808.0,Agg Assault,33.8278,-84.3563,2010-01-30,6,1,6
100110279.0,Auto Theft,33.7125,-84.44815,2010-01-11,7,1,7
100030089.0,Robbery-Pedestrian,33.74107,-84.50997,2010-01-03,8,1,4
100111164.0,Burglary-Residence,33.81973,-84.37137,2010-01-11,9,2,8
100160482.0,Robbery-Pedestrian,33.84064,-84.37653,2010-01-16,10,1,9
100241293.0,Larceny-From Vehicle,33.65674,-84.5076,2010-01-24,11,3,5
100231979.0,Burglary-Residence,33.72611,-84.44437,2010-01-23,12,1,10
100170208.0,Burglary-Residence,33.70199,-84.3827,2010-01-17,13,1,11
```

Cypher Code
Using APOC, creating **CRIME TYPE labels dynamically** for all the crime nodes, based on different crime types.

```
Favorite: 05_Convert Crime Type to Labels*
1 //05_Convert Crime Type to Labels
2 MATCH (n:Crime)
3 WITH DISTINCT n.Crime_Type AS crimetype, collect(n) AS crimes
4 CALL apoc.create.addLabels(crimes, [apoc.text.upperCamelCase(crimetype)]) YIELD node
5 RETURN *
```

· Queries, Cypher Codes & Results

1.How many crimes are recorded for a given crime types in a specific neighborhood for a particular period ?

```
Favorite: 06_Query 1*
1  //06_Query 1
2  MATCH (c:LarcenyFromVehicle) ⟶ (nb:Neighborhood {Neighborhood:"Downtown"})
3  WHERE c.Date.month ⩾ 10 and c.Date.year = 2010
4  WITH c, nb
5  ORDER BY c.Date
6  RETURN  nb.Neighborhood as Neighborhood, c.Date.year as Year, collect(distinct(c.Date.month)) as Months, count(c) as Total_Crimes
```

Result below shows : Crime Type : **Larceny From Vehicle**. Neighborhood : **Downtown**, Time Period. **2010**, **Oct, Nov, Dec**

Advantages:

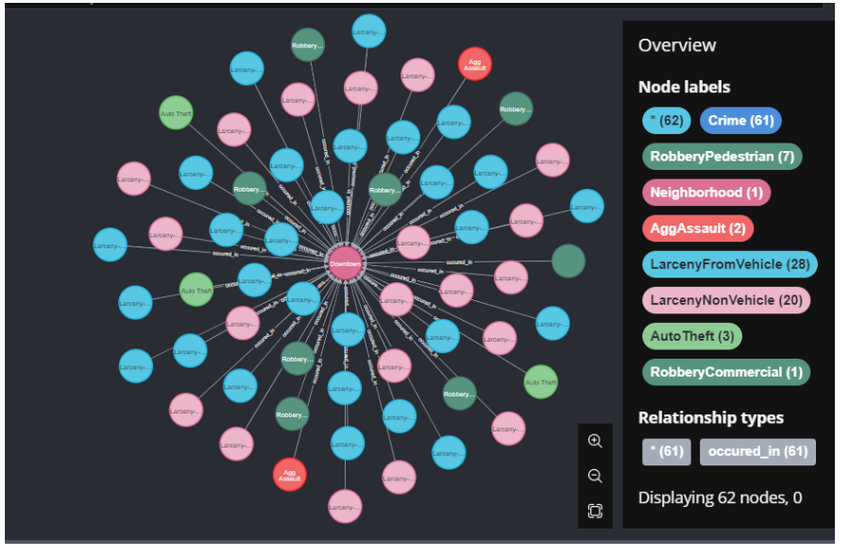
The date property of the crime nodes, which holds datetime data type, can be easily accessed on year, month, date level. To filter crime nodes based on time period being queried.

The Second label of crime types on Crime nodes, can also be easily used for queries, to directly access the certain crime type.

Table / Text / Code

	Neighborhood	Year	Months	Total_Crimes
1	"Downtown"	2010	[10, 11, 12]	28

*Below demonstrated how different crime types can be represented in different colors for clear visualization, thanks to the second crime type labels



2.Find the neighborhoods that shares the same crime types, organize in descending order of the number of common crime types.

```
//07_Query 2_APOC
MATCH (c1:Crime)⟶(n1:Neighborhood)
WITH *
ORDER BY ID(c1)
WITH n1,apoc.coll.sort( collect(distinct c1.Crime_Type)) AS c1Crimes
MATCH (c2:Crime)⟶(n2:Neighborhood)
WHERE ID(n1) < ID(n2)
WITH *
ORDER BY ID(c2)
WITH n1, c1Crimes, n2, apoc.coll.sort(collect(distinct c2.Crime_Type)) AS c2Crimes
WHERE c1Crimes = c2Crimes
with collect(distinct n1.Neighborhood) as result1, collect(distinct n2.Neighborhood) as result2, c1Crimes
With result1 + result2 as finalresult, c1Crimes
RETURN SIZE(c1Crimes) as Count, apoc.coll.sort(c1Crimes) as Common_Crime_Types, apoc.coll.sort(apoc.coll.toSet(finalresult)) as
Neighborhoods_Share_Common_Crime_Types
ORDER BY Count DESC, Common_Crime_Types[0]
```

In order to compare distinct crime types of each neighborhood, and return all the suburb that shares exactly same distinct crime types.

The query conducted a pairwise comparison of all the possible pairs by using node IDs, then collect all the distinct crime types and sort in order.

APOC.COLL.SORT procedure was used multiple times to ensure meaningful comparison. As lists are ordered.

APOC.COLL.TOSET procedure was also used to simplify the code, avoiding using excessive collect and unwind.

Results are shown in the next page.

```
neo4j$ //07_Query 2_APOC MATCH (c1:Crime)-->(n1:Neighborhood) WITH * ORDER BY ID(c1) WITH n1,apoc.coll.sort( collect(distinct c
```

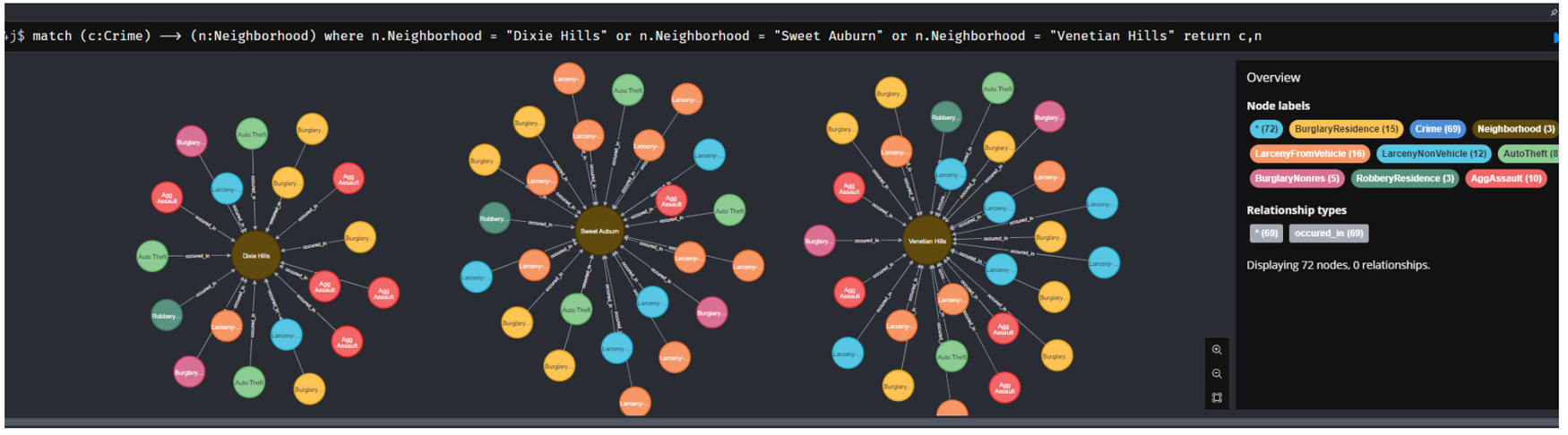
```
MATCH (c1:Crime)-->(n1:Neighborhood) WITH * ORDER BY ID(c1) WITH n1,apoc.coll.sort( collect(distinct c1.Cri…
```

Count	Common_Crime_Types	Neighborhoods_Share_Common_Crime_Types
9	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Rape", "Robbery-Commercial", "Robbery-Pedestrian"]	["Downtown", "Midtown"]
9	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Robbery-Commercial", "Robbery-Pedestrian", "Robbery-Residence"]	["Old Fourth Ward", "Perkerson"]
8	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Robbery-Pedestrian", "Robbery-Residence"]	["Grove Park", "Lindridge/Martin Manor", "Pittsburgh", "South Atlanta"]
8	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Robbery-Commercial", "Robbery-Pedestrian"]	["Grant Park", "Sylvan Hills"]
7	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Robbery-Pedestrian"]	["Custer/McDonough/Guice", "East Lake", "Edgewood", "Greenbriar", "Hammond Park", "Harland Terrace", "Historic Westin Heights/Bankhead", "West End", "Westview"]
7	["Agg Assault", "Auto Theft", "Burglary-Residence", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Robbery-Pedestrian", "Robbery-Residence"]	["Ashview Heights", "Campbellton Road", "Vine City"]
7	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Robbery-Residence"]	["Dixie Hills", "Sweet Auburn", "Venetian Hills"]
6	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Larceny-From Vehicle", "Larceny-Non Vehicle"]	["Adams Park", "Cabbagetown", "Cascade Avenue/Road", "Hunter Hills", "Kirkwood", "North Buckhead", "Peoplestown", "Pine Hills", "Southwest", "Summerhill"]

Results above were firstly ordered descending by number of common crime types, then sorted Alphabetically within all the lists, also sorted between lists in Common_Crime_Types.

Below graph has visualized one of the result above: The suburbs ["Dixie Hills", "Sweet Auburn", "Venetian Hills"], share the same 7 types of crimes
["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Robbery-Residence"]

We can again easily tell by the color of different crime types, and further validated the accuracy of the results.



3. Return the top 5 neighborhoods for a specified crime for a specified duration.

```
1 //08_Query 3
2 MATCH (c:RobberyCommercial|RobberyPedestrian|RobberyResidence) ⟶ (nb:Neighborhood)
3 WHERE c.Date ⩾ date("2010-02-23") and c.Date ⩽ date("2010-09-01")
4 RETURN  nb.Neighborhood as Neighborhood, Count(c) as
  Total_of_Robbery_Crimes_Between_0223_to_0901_In_2010
5 ORDER BY Total_of_Robbery_Crimes_Between_0223_to_0901_In_2010 DESC
6 LIMIT 5
```

Table | Text | Code

	Neighborhood	Total_of_Robbery_Crimes_Between_0223_to_0901_In_2010
1	"Downtown"	11
2	"Pittsburgh"	7
3	"Midtown"	6
4	"Old Fourth Ward"	5
5	"Sylvan Hills"	3

Started streaming 5 records after 1 ms and completed after 18 ms.

The left result displayed the top 5 neighborhood for all the **robbery** crime, during **2010-02-23 to 2010-09-01.**

The query is fairly straight forward. Again, the datetime data type can be easily queried any period of the time.

The crime type labels can also be easily used, Making the code more easily to read too.

4. Find the types of crimes for each property type.

```
//09_Query_4_APOC
MATCH (p:Pro_Type)
WITH p
ORDER BY p.Property_Type
return p.Property_Type as Property_Type, apoc.coll.sort(apoc.coll.toSet([(c:Crime) ⟶ (p) |  c.Crime_Type])) as Crime_Types
```

Table | Text | Code

	Property_Type	Crime_Types
1	"amenity"	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Robbery-Commercial", "Robbery-Pedestrian"]
2	"building"	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Rape", "Robbery-Commercial", "Robbery-Pedestrian", "Robbery-Residence"]
3	"city"	["Agg Assault", "Auto Theft", "Burglary-Residence", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Robbery-Pedestrian"]
4	"club"	["Larceny-From Vehicle"]
5	"craft"	["Larceny-From Vehicle", "Robbery-Pedestrian"]
6	"hamlet"	["Larceny-Non Vehicle"]

The above result displayed all the distinct crime type list for each property type.

The query is fairly straight forward. Pattern comprehension was used to collect crime types for each property type, Apoc sort and toSet procedures were also used.

The Cypher collect syntax can also be easily used for such type of queries.

5. Which month of a specified year has the highest crime rate ? Return one record each for each beat, and each zone too.

```
1  //10_Query 5_Beat
2  MATCH (c:Crime) - [h:occured_in] → (b:Beat)
3  WITH b.Beat as beat, c.Date.month as Month,  count(*) as result
4  ORDER BY beat, result DESC
5  WITH beat, COLLECT (Month) as monthlist, COLLECT (result) as resultlist, COLLECT(result)[0] as topcrime
6  WITH beat, monthlist, [x in resultlist WHERE x = topcrime |x] as newlist
7  WITH beat, monthlist , newlist, size(newlist) as topn
8  RETURN beat, monthlist[0..topn] as highest_crime_occurrence_month_in_2010, newlist[0] as highest_crime_count
```

Table | Text | Code

	beat	highest_crime_occurrence_month_in_2010	highest_crime_count
1	101	[11]	6
2	102	[11]	10
3	103	[12, 10]	5
4	104	[10]	9
5	105	[7, 5]	6
6	106	[4]	7
7	107	[2, 1]	6
8	108	[12, 7, 6, 11]	5
9	109	[6, 11, 7]	4

The left result displayed all the beats, with highest crime month in 2010, followed by the count of how many crimes .

The query is a bit more complex. It is easy to return the top 1, however the situation would occur when 2, or more months for certain beat, would have exactly same crime count as highest.

For example.

The beat 103, both October and December has 5 times of crime happened.

The beat 108, a total of 4 months shares the same number of the highest crime numbers.

The query is capable of accurately return such situation into a list.

```
1 //10_Query_5_Zone
2 MATCH (c:Crime) ⟶ (b:Beat) ⟶ (z:Zone)
3 WITH z.Zone as zone, c.Date.month as Month,  count(*) as result
4 ORDER BY zone, result DESC
5 WITH zone, COLLECT (Month) as monthlist, COLLECT (result) as resultlist, COLLECT(result)[0] as topcrime
6 WITH zone, monthlist, [x in resultlist WHERE x = topcrime |x] as newlist
7 WITH zone, monthlist , newlist, size(newlist) as topn
8 Return zone, monthlist[0..topn] as highest_crime_occurrence_month_in_2010, newlist[0] as highest_crime_count
```

Table | Text | Code

	zone	highest_crime_occurrence_month_in_2010	highest_crime_count
1	1	[11]	52
2	2	[11, 6]	44
3	3	[7]	62
4	4	[12]	60
5	5	[7]	59
6	6	[10]	62

The left result displayed all the zones, with highest crime month in 2010, followed by the count of how many crimes .

The query is capable of accurately return such situation into a list.

6. Find the zones that are adjacent , and sharing the same high crime months.

Before answering this question, the adjacent zone information were manually collected from the map, then used Cypher to load the relationship into the database.

Month nodes have also been created. In order to create virtual relationships to visualized the results.

*Favorite: 11_Create_Adjecent_Rel**

```
//11_Create_Adjecent_Rel
LOAD CSV WITH HEADERS FROM 'file:///adjacent_zone.csv' AS row
MATCH (z:Zone {Zone: toInteger(row.zone)}),
(z1:Zone {Zone: toInteger(row.adjecent_zone)})
CREATE (z)-[:ADJECENT]->(z1)
```

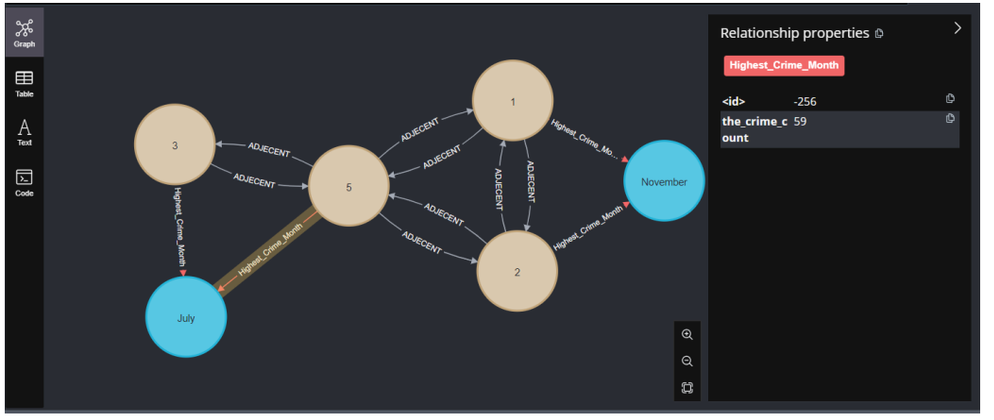
*Favorite: 12_Create_Month_Nodes**

```
//12_Create_Month_Nodes
LOAD CSV WITH HEADERS FROM 'file:///month.csv' AS row
CREATE (m:Month {
    Month_Number: toInteger(row.Month_Number),
    Name: row.Month_Name})
```

Query syntax as below

```
//13_Query_6
MATCH (c:Crime) ⟶ (b:Beat) ⟶ (z:Zone)
WITH z.Zone as zone, c.Date.month as Month,  count(*) as result
ORDER BY zone, result DESC
WITH zone, COLLECT (Month) AS monthlist, COLLECT (result) AS resultlist, COLLECT(result)[0] AS topcrime
WITH zone, monthlist, [x IN resultlist WHERE x = topcrime |x] AS newlist
WITH zone, monthlist , newlist, size(newlist) AS topn
WITH zone, monthlist[0..topn] AS highest_crime_occurrence_month, newlist[0] AS highest_crime_count
UNWIND highest_crime_occurrence_month AS month
WITH  month,zone, highest_crime_count
MATCH (z:Zone{Zone:zone})
MATCH (m:Month{Month_Number:month})
CALL apoc.create.vRelationship(z, 'Highest_Crime_Month', {the_crime_count:highest_crime_count}, m) YIELD rel
WITH month, collect(zone) AS zones,collect (rel) AS rels
MATCH (z:Zone) -[a:ADJECENT] - (z1:Zone)
MATCH (m:Month{Month_Number:month})
WHERE z.Zone IN zones AND z1.Zone IN zones
RETURN startNode(a), endNode(a), m, rels
```

Result Graph as below



The above result graph shows.

Zone 3, and Zone 5 are adjacent, and July is their highest crime rate month.

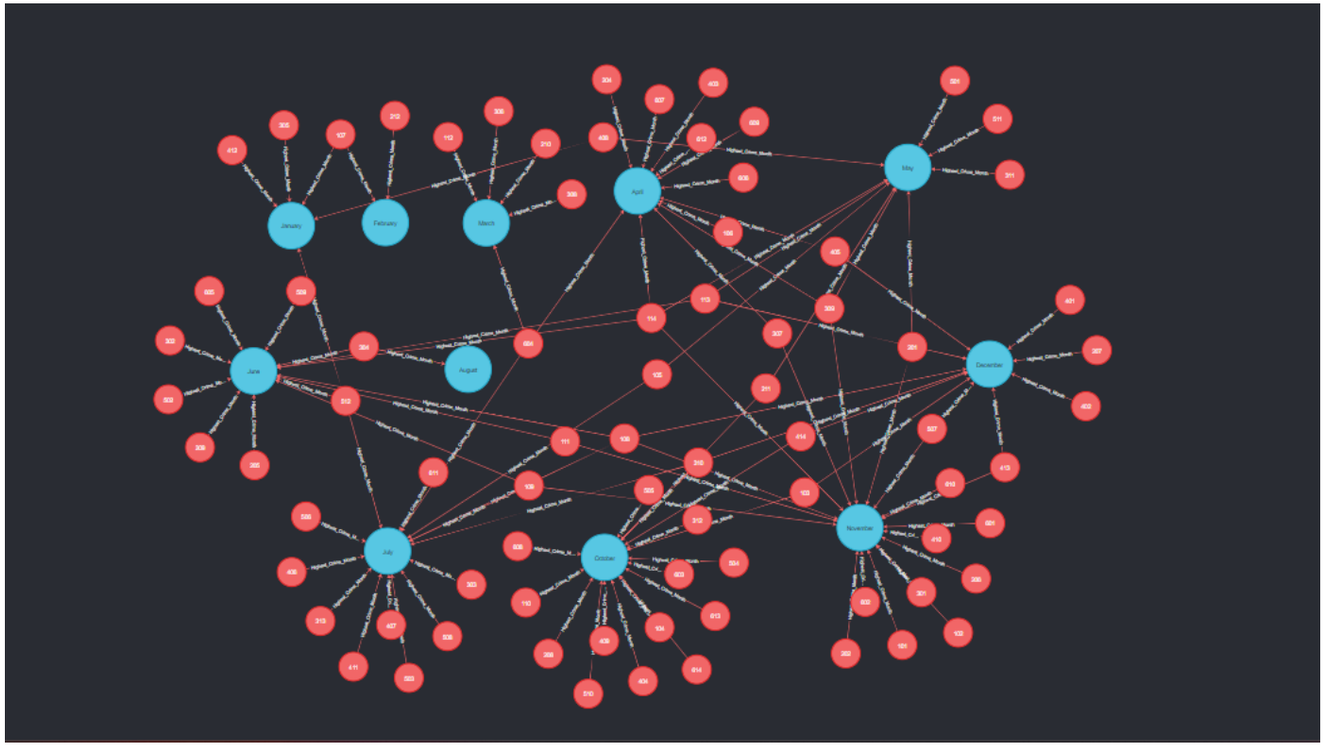
Zone 1, and Zone 2 are adjacent, and November is their highest crime rate month.

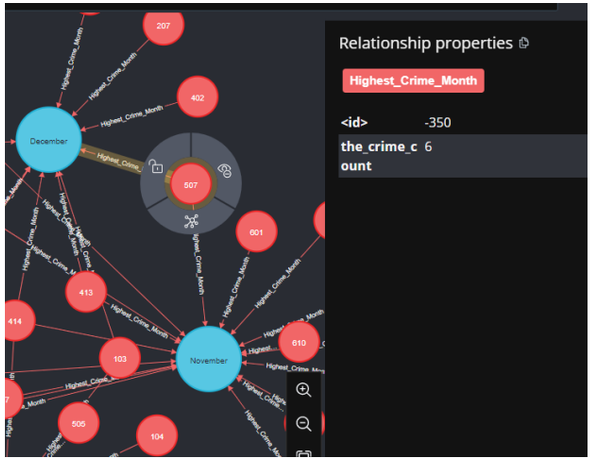
The **Red Virtual Relationship** , has a property of **the_crime_count,** which stores the actually numbers of crimes .

eg : The highlighted virtual relationship above, shows there were 59 times of crime happened , in Zone 5 in July 2010.

7. In police terminology, a beat is the territory that a police officer is assigned to patrol. Therefore it will be very useful to find out which beats share the same highest crime months. So the police force focus can be better organized in advance.

```
//14_Query_7
MATCH (c:Crime) - [h:occured_in] → (b:Beat)
WITH b.Beat as beat, c.Date.month as Month,  count(*) as result
ORDER BY beat, result DESC
WITH beat, COLLECT (Month) as monthlist, COLLECT (result) as resultlist, COLLECT(result)[0] as topcrime
WITH beat, monthlist, [x in resultlist WHERE x = topcrime |x] as newlist
WITH beat, monthlist , newlist, size(newlist) as topn
WITH beat, monthlist[0..topn] as highest_crime_occurrence_month_in_2010, newlist[0] as highest_crime_count
UNWIND highest_crime_occurrence_month_in_2010 AS month
WITH  month,beat, highest_crime_count
MATCH (b:Beat{Beat:beat})
MATCH (m:Month{Month_Number:month})
CALL apoc.create.vRelationship(b, 'Highest_Crime_Month', {the_crime_count:highest_crime_count}, m) YIELD rel
RETURN b,m,rel
```





The result of above query is demonstrated through graph with virtual relationships on the left.

It is very easy and clear to see that months like October, July, would need more patrol police in more various beats, comparing with February, August.

If we are interested in certain beat, eg 507.
We can find the connected month nodes, November & December.
These two months are the highest crime months for beat 507. Police patrol should focus more during these two month.

The property of the virtual relationship shows above, further tells the highest crime count of these two months for beat 507 are both 6.

8. In relational SQL database design, it is difficult to add descriptive data into the cube design. For example, the population of each suburb, the average income of each NPU etc. But for property graph database, it is easy to add/update such information on any nodes.

As the population of each suburb in 2010 has been enriched through Python, and added as node property. This query is not just simply comparing the absolute number of the crime occurrence, but more specifically look into the crime 'rate'.

For example, the 100 crimes that have happend in neighborhood A, doesn't mean A is safer than neighborhood B which has 50 crimes happened. The population of neighborhood A maybe 10 times greater than neighborhood B.

```
1 //15_Query_8
2 MATCH (n:Crime) - [:occured_in] → (nbh:Neighborhood)
3 WHERE nbh.Population_2010 <> 0
4 WITH nbh, count(*) as crimes
5 RETURN nbh.Neighborhood as Neighborhood, crimes as Count_of_Crimes_in_2010, nbh.Population_2010 as Population_in_2010,
  round((crimes*1.0/nbh.Population_2010)*100,4)+ "%" as Percentage
6 ORDER BY Percentage DESC
```

	Neighborhood	Count_of_Crimes_in_2010	Population_in_2010	Percentage
1	"Marietta Street Artery"	32	745	"4.2953%"
2	"Lenox"	57	1663	"3.4275%"
3	"Berkeley Park"	41	1400	"2.9286%"
4	"The Villages at Carver"	28	1039	"2.6949%"
5	"Castleberry Hill"	29	1285	"2.2568%"
6	"Ashview Heights"	26	1292	"2.0124%"
7	"West End"	79	4270	"1.8501%"
8	"Downtown"	225	13411	"1.6777%"
9	"Lakewood Heights"	35	2177	"1.6077%"
10	"Pittsburgh"	58	3658	"1.5856%"
11	"South Atlanta"	27	1738	"1.5535%"

The results returned a percentage for each neighborhood, calculated by number of crime occurrences in that neighborhood, divided by its population recorded in 2010. Then sorted by the percentage DESC.

The "Downtown" has clearly higher crime numbers (225) than "Marietta Street Artery" (32), but if we consider the population. Marietta Street Artery may actually be a less safer suburb.

*The query has excluded the suburbs with no population info.

9. When a certain crime case is unsolved. for example. Homicide _ Crime_ID: 103550044091 . In order to revisit the previous homicide cases to find links. Please list all the other Homicide cases that has happened in the same year of 2010, order by the geodesic distance based on latitude/longitude.

```
1 MATCH(a:Homicide{Crime_ID:103550044091})
2 MATCH (b:Beat)<--(c:Homicide) -->(n:Neighborhood)
3 WHERE c.Crime_ID <> a.Crime_ID
4 WITH  c, point.distance(a.location,c.location) as Distance, n, b
5 RETURN c.Crime_ID as Crime_ID, Distance, c.Date as Date, n.Neighborhood as Neighborhood,
  b.Beat as Beat
6 ORDER BY Distance
```

Table | Text | Code

	Crime_ID	Distance	Date	Neighborhood	Beat
1	101950274044	3383.123811362108	"2010-07-14"	"Lakewood Heights"	307
2	101201748028	6434.9059632589415	"2010-04-30"	"Oakland City"	403
3	103051706082	10661.98312280321	"2010-11-01"	"English Avenue"	506
4	103262151086	12347.906156279894	"2010-11-22"	"Virginia Highland"	601

Started streaming 4 records after 1 ms and completed after 2 ms.

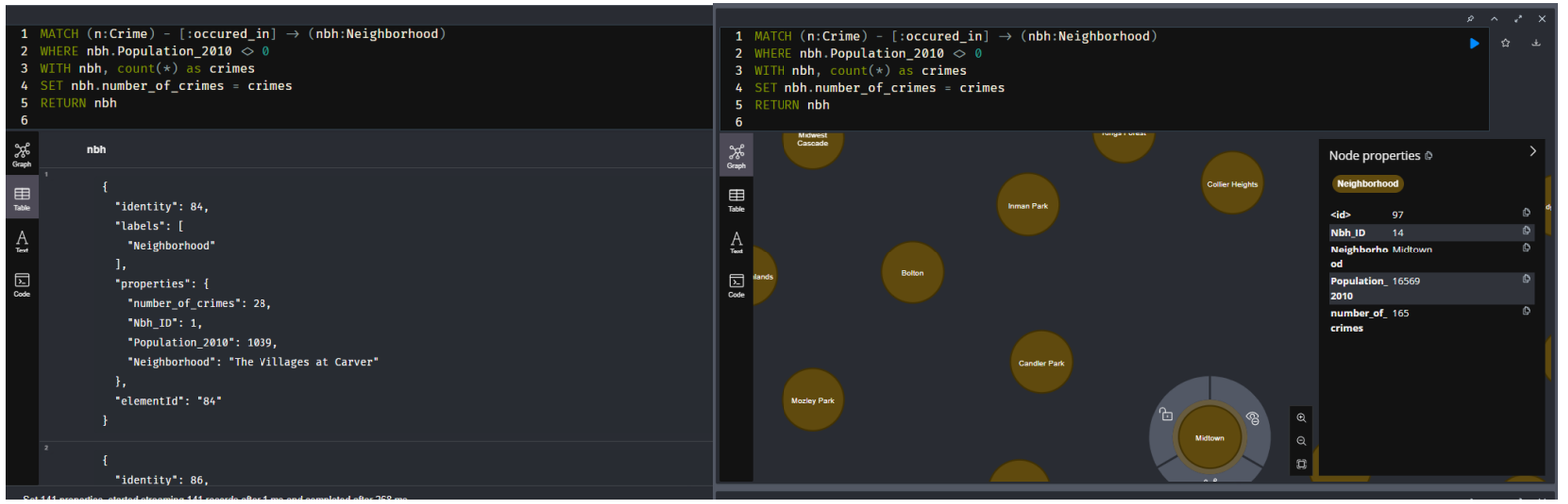
The result on the left shows all the homicide cases that happened in 2010, ordered by the distance from the base crime.

The result shows the closest crime that has happened was around 3383m away, on 2010-07-14, in Neighborhood "Lakewood Heights", Beat 307.

Graph Data Science Part 1

Neo4j Graph Data Science Library has provided many powerful algorithms including path finding, community detection, similarity etc.
Below is a demonstration on how to use Filtered K-Nearest Neighbors algorithm for detect similarities between suburbs.

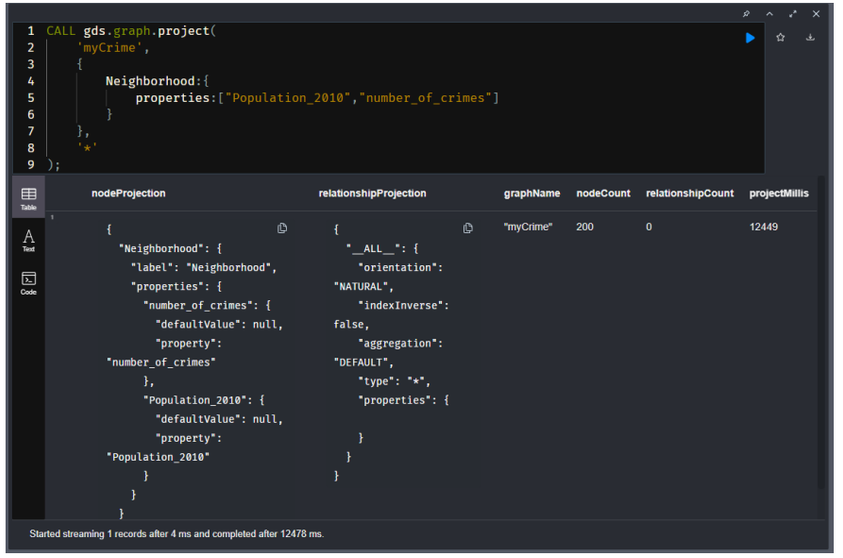
Based on two properties of the nodes, "number of crimes", and "population". If the similarity score is high, it would be reasonable to provide similar police patrol in these alike suburbs.



Step 1:

A new property: number of crimes. has been counted and added on the Neighborhood nodes

*Neighborhood without population info will be excluded, hence filtered k-nearest neighbor algorithm was used.



Step 2:

Before running GDS algorithms, we need firstly project our graph for the Graph Data Science library

```
MATCH(n:Neighborhood) WHERE n.Population_2010 > 0
with Collect (n) AS filtered_nbh
CALL gds.alpha.knn.filtered.stream ("myCrime",{
    topK: 1,
    nodeProperties: ['Population_2010','number_of_crimes'],
    sourceNodeFilter: filtered_nbh,randomSeed: 1337,concurrency: 1,sampleRate : 1.0,
    deltaThreshold: 0.0
})
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).Neighborhood as Nbh1, gds.util.asNode(node2).Neighborhood as Nbh2,similarity
ORDER BY similarity DESCENDING, Nbh1, Nbh2
```

Table | Text | Code

	Nbh1	Nbh2	similarity
1	"Leila Valley"	"Rosedale Heights"	0.625
2	"Rosedale Heights"	"Leila Valley"	0.625
3	"Rockdale"	"Whittier Mill Village"	0.5714285714285714
4	"Whittier Mill Village"	"Rockdale"	0.5714285714285714
5	"Cabbagetown"	"West Lake"	0.55
6	"West Lake"	"Cabbagetown"	0.55

Step 3:

Two properties of Neighborhood Nodes were used for feeding into the algorithms .["Population_2010", "number of_crimes"]

Left is the script to run the algorithms.

The results are ordered by Similarity scores.

Below are the details of these top similar neighborhood. It further proved the effectiveness of the similarity algorithms.

Both pair has the same number of crimes, with very close population recorded in 2010.

n.Neighborhood	n.Population_2010	n.number_of_crimes
"Leila Valley"	847	9
"Rosedale Heights"	844	9
"Whittier Mill Village"	617	3
"Rockdale"	611	3

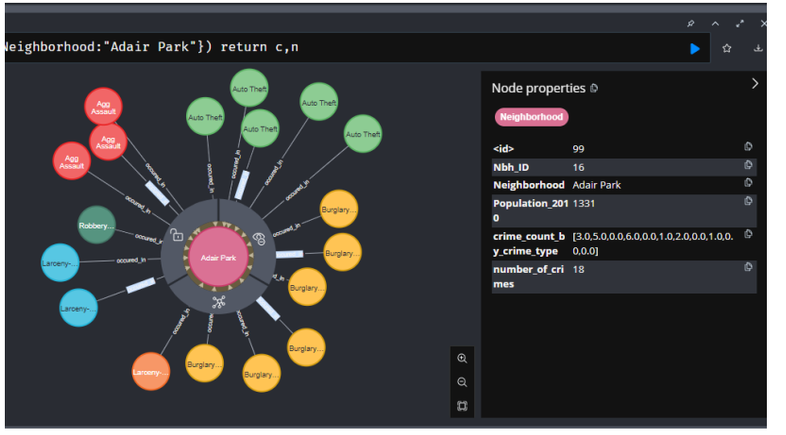
Graph Data Science Part 2

In Part 1 the properties "Population _ 2010" and "Number of Crimes were used to find similar Neighborhood.
In this Part 2: a list of crime numbers for all the crime types was created, set as a new property "crime_count_by_crime_types", for each neighborhood.
In order to better compare similarities.

For the meaningful comparison, the list results are following the same crime types orders as below. On the right is to validate the correct results.

```
1  //18_Data_Science_2_Query_Result
2  MATCH (c:Crime)
3  WITH apoc.coll.sort(collect(distinct (c.Crime_Type))) as crime_types
4  UNWIND crime_types as ccc
5  WITH ccc
6  MATCH (n:Neighborhood)
7  WITH apoc.coll.sort(collect(distinct (n.Neighborhood))) as Neighborhoods , ccc
8  UNWIND Neighborhoods as nnn
9  WITH ccc,nnn
10 OPTIONAL MATCH (c1:Crime {Crime_Type:ccc}) ⟶ (n:Neighborhood{Neighborhood:nnn})
11 WITH toFloat(count(c1)) as result, ccc, nnn
12 RETURN collect (result) as results, nnn , collect(ccc) as all_crime_types
```

	results	nnn	all_crime_types
1	[3.0, 5.0, 0.0, 6.0, 0.0, 1.0, 2.0, 0.0, 1.0, 0.0, 0.0]	"Adair Park"	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Homicide", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Rape", "Robbery-Commercial", "Robbery-Pedestrian", "Robb
2	[1.0, 4.0, 1.0, 5.0, 0.0, 6.0, 4.0, 0.0, 0.0, 0.0, 0.0]	"Adams Park"	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Homicide", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Rape", "Robbery-Commercial", "Robbery-Pedestrian", "Robb
3	[0.0, 3.0, 3.0, 3.0, 0.0, 0.0, 7.0, 0.0, 0.0, 1.0, 0.0]	"Adamsville"	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Homicide", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Rape", "Robbery-Commercial", "Robbery-Pedestrian", "Robb
4	[3.0, 1.0, 0.0, 6.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	"Almond Park"	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Homicide", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Rape", "Robbery-Commercial", "Robbery-Pedestrian", "Robb
5	[0.0, 0.0, 0.0, 3.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	"Amal Heights"	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Homicide", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Rape", "Robbery-Commercial", "Robbery-Pedestrian", "Robb
6	[0.0, 4.0, 1.0, 3.0, 0.0, 8.0, 2.0, 0.0, 0.0, 1.0, 0.0]	"Ansley Park"	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Homicide", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Rape", "Robbery-Commercial", "Robbery-Pedestrian", "Robb
7	[0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]	"Arden/Habersham"	["Agg Assault", "Auto Theft", "Burglary-Nonres", "Burglary-Residence", "Homicide", "Larceny-From Vehicle", "Larceny-Non Vehicle", "Rape", "Robbery-Commercial", "Robbery-Pedestrian", "Robb



As you can see , I've cast the count of each crime types into **Float** number, the reason behind is due to Data Science Library Similarity Metrics.

Only Jaccard Similarity and Overlap Coefficient are provided for list of interger comparison. However they used intersection, union, and minimum set to compare, which is not the most suitable as the list of numbers for each crimes are all ordered in the same way. So the metrics provided for compare floating-point number lists are a lot more meaningful in this case.

1.1.2. List of integers

When a property is a list of integers, similarity can be measured with either the Jaccard similarity or the Overlap coefficient.

Jaccard similarity

$$J(p_s, p_t) = \frac{|p_s \cap p_t|}{|p_s \cup p_t|}$$

Figure 2. size of intersection divided by size of union

Overlap coefficient

$$O(p_s, p_t) = \frac{|p_s \cap p_t|}{min(|p_s|, |p_t|)}$$

Figure 3. size of intersection divided by size of minimum set

1.1.3. List of floating-point numbers

When a property is a list of floating-point numbers, there are three alternatives for computing similarity between two nodes.

The default metric used is that of Cosine similarity.

Cosine similarity

$$cosine(p_s, p_t) = \frac{\sum_i p_s(i) \cdot p_t(i)}{\sqrt{\sum_i p_s(i)^2} \cdot \sqrt{\sum_i p_t(i)^2}}$$

Figure 4. dot product of the vectors divided by the product of their lengths

Notice that the above formula gives a score in the range of `[-1, 1]` . The score is normalized into the range `[0, 1]` by doing `score = (score + 1) / 2`.

The other two metrics include the Pearson correlation score and Normalized Euclidean similarity.

Pearson correlation score

$$pearson(p_s, p_t) = \frac{\sum_i (p_s(i) - \overline{p_s}) \cdot (p_t(i) - \overline{p_t})}{\sqrt{\sum_i (p_s(i) - \overline{p_s})^2} \cdot \sqrt{\sum_i (p_t(i) - \overline{p_t})^2}}$$

Figure 5. covariance divided by the product of the standard deviations

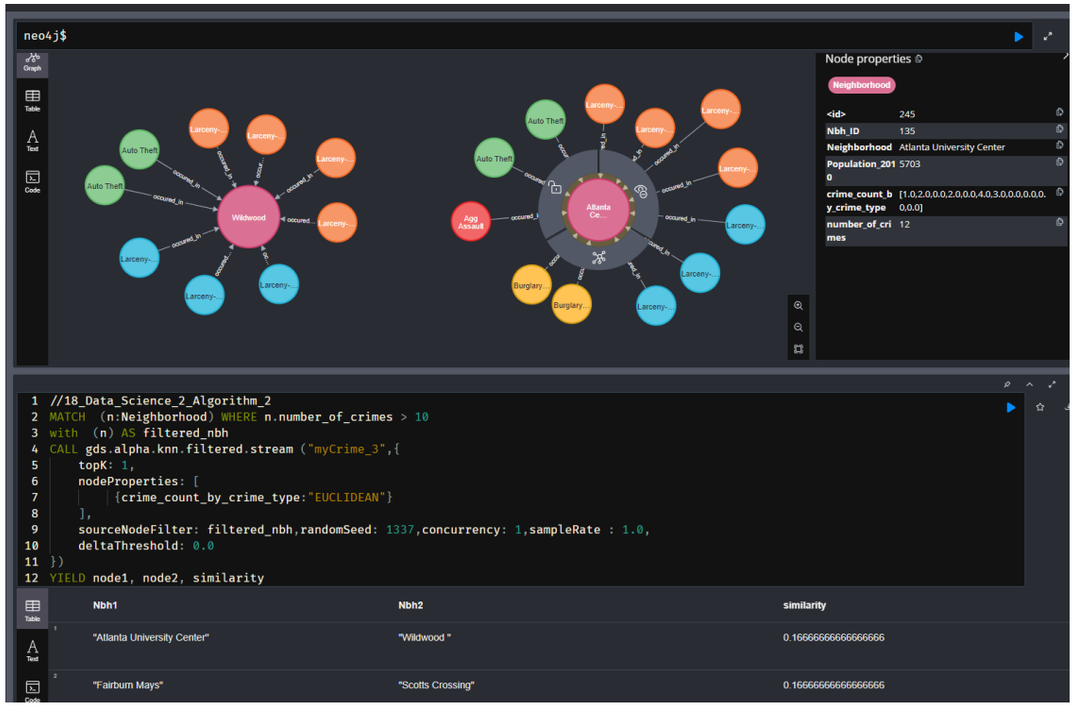
As above, the formula gives a score in the range `[-1, 1]`, which is normalized into the range `[0, 1]` similarly.

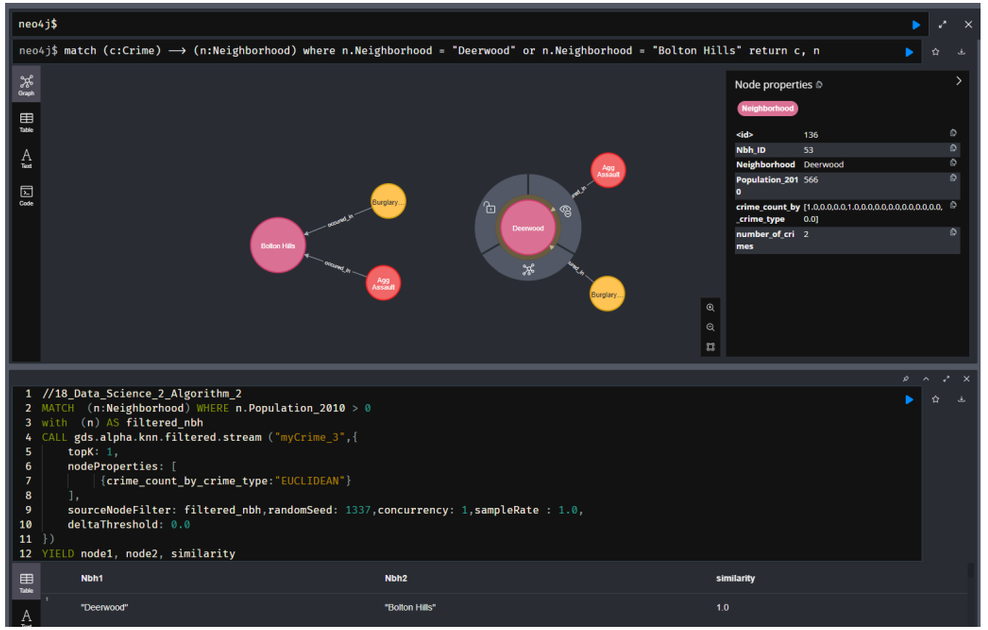
Euclidean similarity

$$ED(p_s, p_t) = \sqrt{\sum_i (p_s(i) - p_t(i))^2}$$

Figure 6. the root of the sum of the square difference between each pair of elements

The result from this formula is a non-negative value, but is not necessarily bounded into the `[0, 1]` range. To bound the number into this range and obtain a similarity score, we return `score = 1 / (1 + distance)` , i.e., we perform the same normalization as in the case of scalar values.





Ref : https://neo4j.com/docs/graph-data-science/current/algorithms/knn/

For the above, **Euclidean similarity** were used. On the left the source node filter filtered out all the total crime count less than 10 suburbs, right filtered out suburbs with no population info.

I believe this results will be beneficial to detect neighborhood with similar crime patterns , for police force planning.
And there are countless results can be returned if different source/end node filters, or different metrics like Pearson Correlation Score were used, multiple properties of the nodes can also be feed in the algorithms to compare at the same time.

Summary :

Overall, my design of the graph database is simple without being unnecessarily complicated, it can answer all the queries, with meaningful findings returned by using Cypher, APOC and data science library.

Road information can further be introduced, combine with GPS coordinates from A to B as an estimated cost in A* shortest path algorithm. The data science library is able to generate shortest path for police force in the area, without repetition on the same road, at the cost of unnecessary labour, time etc.

More relationships can be created between neighborhoods and beats, as no total order needs to be followed, If certain neighborhood are covered by multiple beats, extra relationships can be easily added to all the beats.

If I really need to pick cons in the graph database, is the speed to handle large scale of the data, It would be interesting to load the whole dataset of 250k rows of data. Also with nodes display limitation, some relationship may not be shown, even it existed, that can be confusing as well.

Also due to the flexibility to add / delete any node/relationship. It is more prone to human errors. All the properties stored in nodes and relationship will also be deleted.

Overall, I believe graph database may not be an alternative, but at least a great addition to the traditional relationship database. Part of historical dataset can be easily exported for graph database to model differently, in order to utilize the powerful algorithms and superior relationships handling capabilities to deep dive , and explore all the meaningful analytic results. It definitely an awesome way to present the data.