

Data Warehousing - Project 2 Report

Quang Khanh Hua (22928469)

Nguyen Hai Tran (23035141)

University of Western Australia, Australia

Demonstration Clip: <https://youtu.be/bkBT2dHw81E>

Introduction

In the first project, business questions related to crime are answered with the crime datasets, using the data warehouse method, which includes concept hierarchy, fact/dimension tables, etc. Association rule mining was also used to find the correlation between different crime-related factors, giving us insights for future prediction.

In the second project, the graph database method will be used as another approach to answer some business questions related to crime. This will be done by using Cypher code in Neo4j graph database with APOC and Graph Data Science Library plugins.

Body

Comparing Graph Database capabilities with their counterparts

This graph database method has some benefits:

- The ease of observing the relationship between each node: the connection between each node can be easily seen by visualisation. For example, we can check different crime types happening in one location for a day.
- Flexibility design: New properties or nodes can be added, modified or removed in the database without making too complicated changes to the existing

database design.

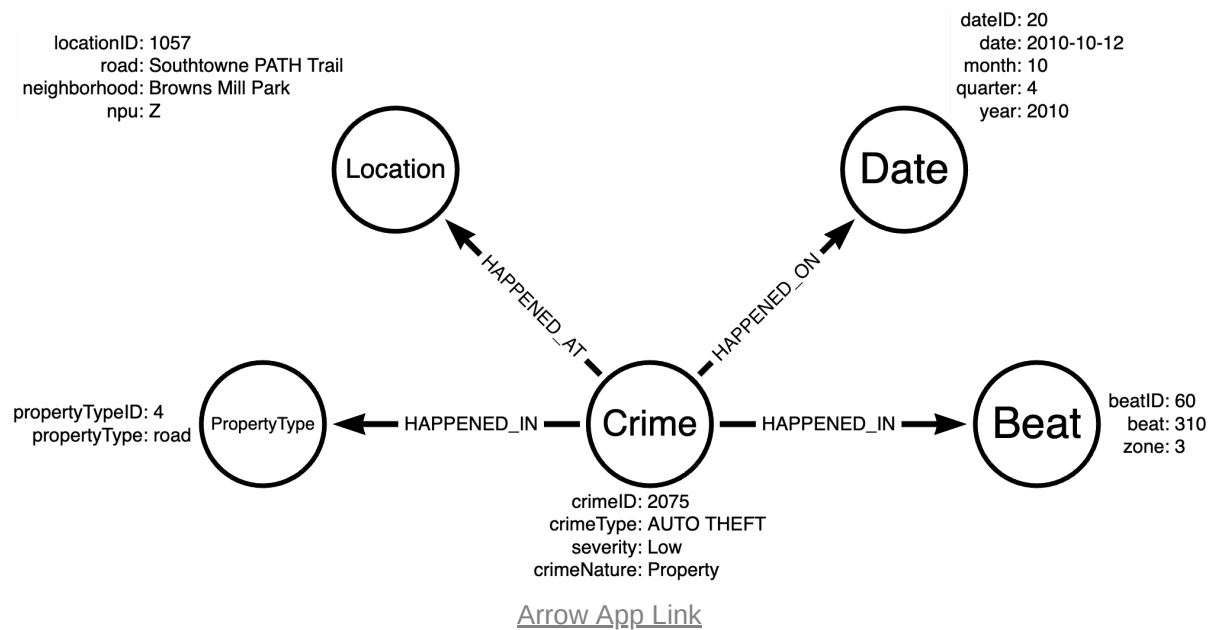
- Excellent performance in relationship queries: with highly connected relationship data, the graph databases query, Cypher, is optimised for efficient navigation of interconnected data.
- Higher data integrity: Since the graph database does not allow null values, it enforces a better, cleaner data model without data inconsistency, poor data quality, and errors caused by undefined value declaration. It also makes queries more efficient since they are not required to handle null values.
- Data insight and discovery: Performing Graph Data Science (GDS) is capable with a graph database, with some algorithms performance such as Breadth, Depth First or Shortest Path algorithms to show insights which help to answer some business questions.

However, the graph database method has some limitations when it is applied to this project:

- The speed of querying the database: Since this project has a large number of records, it takes a much longer time for the queries to give the results compared to relational databases. Therefore, we have to take a much smaller amount of records for this project
- The impact of the database size on the actual results: Due to improvements in speed, the size of the dataset needs to be sacrificed. The dataset will not be balanced due to the random selection using queries, with a much smaller number of records, resulting in the accuracy of the results and insights not be guaranteed 100%.
- The complexity of the query: as mentioned above, the graph databases are convenient for querying the relationship, seeing the graph visualisation and performing GDS. However, it is much more complicated in performing aggregate functions. For example, the MAX function cannot be performed in question 5 which has a high level of complexity.

Database Design

Here is the sample graph database design of this project:



Based on our current relational database, we made minor changes to the database as follows:

- Each crime record (row) in the table with “Crime” (a label) will be a node in the database.
- Every unique value of each table in location, date, property type and beat tables, will be stored as a unique node in the graph database. For example, the date “2010-01-01” will be a node and “2010-01-02” will be a different node.
- Every crime node is connected to a single node of the beat, date, location, and property nodes, while a node from the beat, date, location, and property nodes can be linked to multiple different crime nodes. For instance: there was only one crime with id number 3 on “Lenox Road” while on that street, there were many crimes recorded with different crime ids.

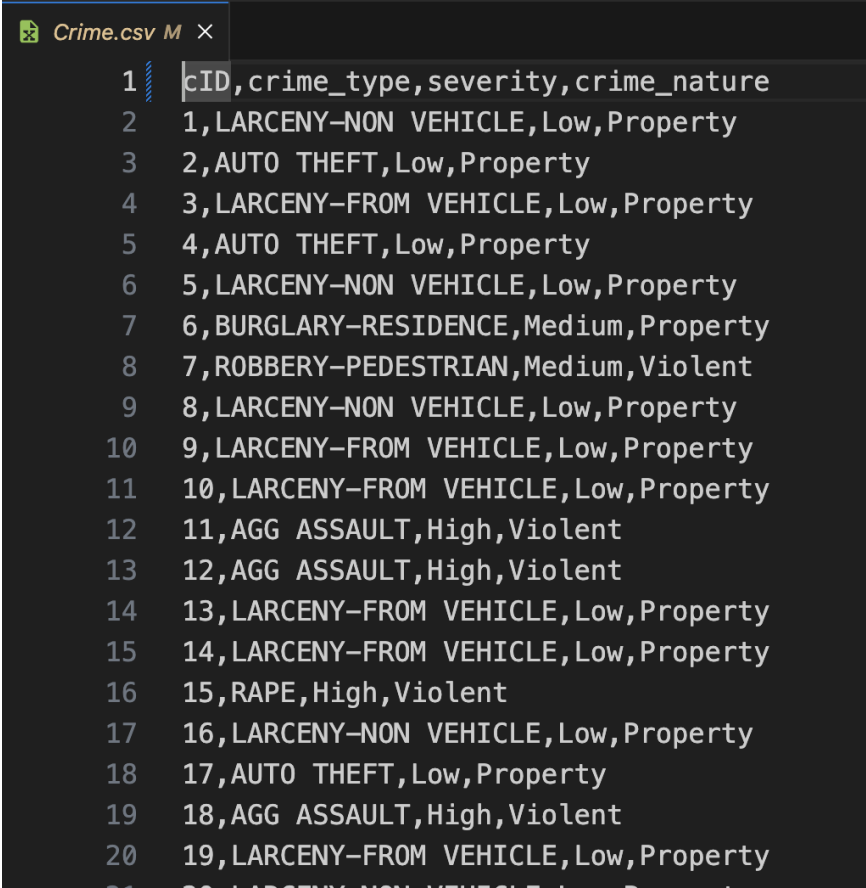
The reason this design came out because of the nature of the current dataset:

- In the crime relational database, it is noticeable that each entity has its own property. For example, each crime was recorded with its distinct properties such as crime type, severity (High, Medium, Low) and crime nature (violent, property). Therefore, it is reasonable to set each crime as a node with its own properties. This rule can be applied to property types, location (properties: road, neighbourhood, npu), date (properties: date, month, quarter, year), beat (properties: beat and zone)

- There are relationships between the crime entity with other entities. A recorded crime happened in a specific property type, happened at an exact location, happened on only one date and happened in a precise beat. Thus, the relationships will be well presented using the graph database.

CSV files for populating Graph Database

- To implement the graph database, we need 2 types of CSV files: data files and relationship files.
 - Data files: there are 5 files represent for 5 tables which are crime, property, location, date and beat. For crime table, each line in the table is a crime record, with its properties (road, neighbourhood and npu). With crime, property, location, date and beat table, an individual line is a unique value. All the tables will have ID column to make each row as a separated value. Here are the screenshots of the crime and property files:



```

1 cID,crime_type,severity,crime_nature
2 1,LARCENY-NON VEHICLE,Low,Property
3 2,AUTO THEFT,Low,Property
4 3,LARCENY-FROM VEHICLE,Low,Property
5 4,AUTO THEFT,Low,Property
6 5,LARCENY-NON VEHICLE,Low,Property
7 6,BURGLARY-RESIDENCE,Medium,Property
8 7,ROBBERY-PEDESTRIAN,Medium,Violent
9 8,LARCENY-NON VEHICLE,Low,Property
10 9,LARCENY-FROM VEHICLE,Low,Property
11 10,LARCENY-FROM VEHICLE,Low,Property
12 11,AGG ASSAULT,High,Violent
13 12,AGG ASSAULT,High,Violent
14 13,LARCENY-FROM VEHICLE,Low,Property
15 14,LARCENY-FROM VEHICLE,Low,Property
16 15,RAPE,High,Violent
17 16,LARCENY-NON VEHICLE,Low,Property
18 17,AUTO THEFT,Low,Property
19 18,AGG ASSAULT,High,Violent
20 19,LARCENY-FROM VEHICLE,Low,Property

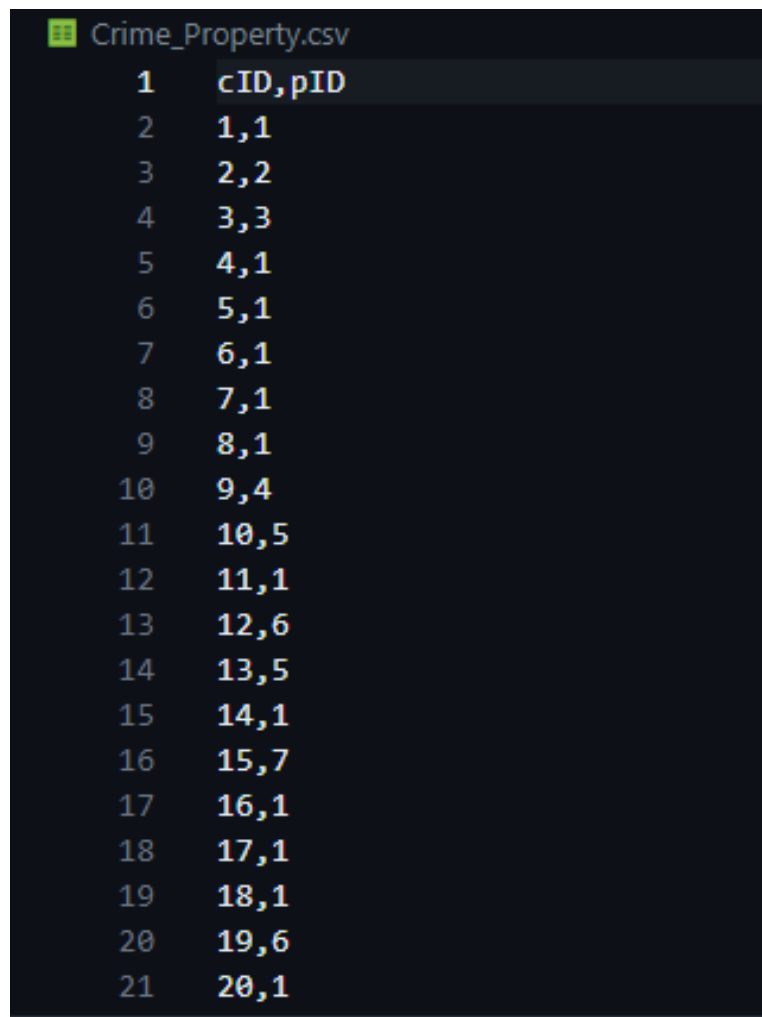
```

Crime.csv

Property.csv	
1	pID,type
2	1,house_number
3	2,office
4	3,shop
5	4,road
6	5,building
7	6,amenity
8	7,tourism
9	8,historic
10	9,leisure
11	10,man_made
12	11,landuse
13	12,highway
14	13,place
15	14,railway
16	15,craft
17	16,club
18	17,aeroway
19	18,healthcare
20	19,emergency
21	

Property.csv

- Relationship files: These files will be loaded into Neo4J using cypher queries to create relationships between crime nodes with property types, location, date and beats nodes. In each of the relationship files, there are only 2 columns: crime ID from crime nodes and another ID from other nodes. The number of lines in these files are the same as the number of line in Crime.csv file. Here is the file used to create the relationship of crimes HAPPENED_IN different types of property:



The image shows a code editor window with a dark background. At the top, the file name 'Crime_Property.csv' is displayed next to a green icon. Below the file name, the contents of the CSV file are shown as a list of 21 rows. The first row is a header with two columns: 'cID' and 'pID'. The subsequent 20 rows contain pairs of integers separated by a comma, representing the values for 'cID' and 'pID' respectively.

	cID	pID
1	1	1
2	2	2
3	3	3
4	4	1
5	5	1
6	6	1
7	7	1
8	8	1
9	9	4
10	10	5
11	11	1
12	12	6
13	13	5
14	14	1
15	15	7
16	16	1
17	17	1
18	18	1
19	19	6
20	20	1

Crime_Property.csv

- All the file above will be extracted using ETL process

ETL code explanation

```
# Read in dataset, process and then combine them into one dataframe
def process_data(file_path, dataset):
```

The function reads in the dataset from a file path and selects specific columns using "pd.read_csv()" method from pandas library. It then iterates over additional datasets, reads them into temporary dataframes, and combines them with the initial dataframe using "pd.concat()". Data cleaning is performed by dropping missing values, removing duplicates, and stripping leading/trailing white spaces. Filtering is applied to remove specific rows based on 'city' which is not 'Atlanta' values. Data transformation is done by adjusting 'crime' column values to remove the '-' character for ease of readability. The processed dataframe is then returned by the function.

```
# Categorize crime based on severity
def severity(crime):
```

This function categorises the input "crime" based on its severity into four categories: "Low", "Medium", "High", or "Unknown". If the input "crime" matches any of the values in the lists for each category, the function returns the corresponding severity category. If the input "crime" does not match any of the predefined values, it is categorised as "Unknown".

```
# Categorize crime based on severity
def crime_nature(crime):
```

This function categorise the input "crime" based on its nature: "Property" and "Violent". If the input "crime" does not match any of the predefined values, it is categorised as "Unknown".

```
def crime(dataframe):

def property(dataframe):

def date(dataframe):

def beat(dataframe):

def location(dataframe):
```

The provided functions aim to construct CSV files for importing nodes into a Neo4j graph database. The resulting CSV file represents different node types, with each row corresponding to a single node in the graph. The functions extract relevant columns from the processed dataframe and create new columns using aggregation functions from pandas library. The CSV files are formatted with a comma separator (sep=',') and a line terminator (lineterminator='\n'), adhering to the standard CSV format. Noticeably, "date" column in the date function is modified to match this date format: "yyyy-mm-dd". This modification is necessary to ensure compatibility with the datetime function, which is responsible for importing the date values into Neo4j.

```
def crime_property(dataframe, property_table):

def crime_beat(dataframe, beat_table):

def crime_date(dataframe, date_table):

def crime_location(dataframe, location_table):
```

The provided functions are responsible for constructing CSV files that facilitate the importation of relationships between nodes into a Neo4j graph database. Each function takes a dataframe as input, along with a corresponding table (e.g., `property_table`, `beat_table`, `date_table`, `location_table`) that contains relevant information for establishing the relationships.

The resulting CSV files consist of two columns: "crimeID," along with another column representing the related ID. This structure enables Neo4j's import mechanism to identify and establish relationships between two nodes based on the provided IDs.

```
dataframe = process_data(file_path, dataset)

crime_table = crime(dataframe)
property_table = property(dataframe)
date_table = date(dataframe)
beat_table = beat(dataframe)
location_table = location(dataframe)

crime_property_table = crime_property(dataframe, property_table)
crime_beat_table = crime_beat(dataframe, beat_table)
crime_date_table = crime_date(dataframe, date_table)
crime_location_table = crime_location(dataframe, location_table)
```

The code above processes a dataset and generates CSV files for importing nodes and relationships into a Neo4j graph database.

Graph Database import cypher

After all the required files are extracted, they are going to be load into Neo4J using Cypher queries. As mentioned in the limitation of the project, to improve processing speed of the queries, there are only 1000 lines selected randomly. Here are the queries to upload the crime and property files with creating relationship between them. The Cypher code for the remaining files and relationships will be applied in the same way.


```
//Loading Crime.csv
LOAD CSV WITH HEADERS FROM 'file:///Crime.csv' AS row
WITH row
ORDER BY rand()
LIMIT 1000
CREATE (c:Crime {
    cID: row.cID,
    crimeType: row.crime_type,
    severity: row.severity,
    crimeNature: row.crime_nature})

//Loading Property.csv
LOAD CSV WITH HEADERS FROM 'file:///Property.csv' AS row
CREATE (p:PropertyType {
    pID: row.pID,
    propertyType: row.type})

//Creating relationship between crime and property
LOAD CSV WITH HEADERS FROM 'file:///Crime_Property.csv' AS row
MATCH (c:Crime {cID: row.cID})
MATCH (p:PropertyType {pID: row.pID})
CREATE (c)-[r:HAPPENED_IN]->(p)
```

When the loading file and creating relationship are finished, then they will be used to answer some business questions related to crime topics.

Query results

```
1 //Q1
2 // How many crimes are recorded for a given crime type in a specified neighbourhood
  for a particular period?
3 MATCH (c:Crime)-[:HAPPENED_ON]→(d:Date),
4 | (c)-[:HAPPENED_AT]→(l:Location)
5 WHERE c.crimeType = 'AUTO THEFT'
6 | AND l.neighborhood = 'Downtown'
7 | AND d.year > 2000
8 | AND d.year < 2020
9 RETURN count(c) AS CrimeCount
```

CrimeCount	
Table	
1	3
Text	
Code	

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

```

2 // Find the neighbourhoods that share the same crime types, organise in decending
3 // order of the number of common crime types
4 // crime types, organise in decending order of the number of common crime types
5 MATCH (l1:Location)←[:HAPPENED_AT]-(c1:Crime), (l2:Location)←[:HAPPENED_AT]-(c2:Crime)
6 WHERE l1.neighborhood > l2.neighborhood
7 WITH l1.neighborhood AS Neighborhood1, l2.neighborhood AS Neighborhood2, COLLECT(DISTINCT c1.crimeType) AS
CommonCrimeTypes1, COLLECT(DISTINCT c2.crimeType) AS CommonCrimeTypes2
8 WHERE SIZE(CommonCrimeTypes1) = SIZE(CommonCrimeTypes2) AND SIZE([crimeType IN CommonCrimeTypes1 WHERE crimeType IN
CommonCrimeTypes2]) = SIZE(CommonCrimeTypes1) AND SIZE([crimeType IN CommonCrimeTypes1 WHERE crimeType IN
CommonCrimeTypes2]) = SIZE(CommonCrimeTypes2)
9 RETURN DISTINCT Neighborhood1, Neighborhood2, SIZE(CommonCrimeTypes1) AS CommonCrimeTypesCount
10 ORDER BY CommonCrimeTypesCount DESC

```

	Neighborhood1	Neighborhood2	CommonCrimeTypesCount
1	"Midtown"	"Mechanicsville"	8
2	"West End"	"Sylvan Hills"	7
3	"West End"	"Pittsburgh"	7
4	"Sylvan Hills"	"Pittsburgh"	7
5	"Vine City"	"Old Fourth Ward"	6
6	"Vine City"	"Perkerson"	6
7	"Grove Park"	"Grant Park"	6
8	"Perkerson"	"Old Fourth Ward"	6

Started streaming 270 records after 1 ms and completed after 247 ms.

```

1 // Q3
2 // Return the top 5 neighbourhoods for a specified crime for a specified duration.
3 MATCH (l:Location)←[:HAPPENED_AT]-(c:Crime)-[:HAPPENED_ON]→(d:Date)
4 WHERE d.year > 2010
5       AND d.year < 2020
6       AND c.crimeType = 'AUTO THEFT'
7 WITH l.neighborhood AS TopNeighborhoods, COUNT(l.neighborhood) as NeighborCount
8 RETURN TopNeighborhoods, NeighborCount
9 ORDER BY NeighborCount DESC
10 LIMIT 5

```

	TopNeighborhoods	NeighborCount
1	"Midtown"	6
2	"Castleberry Hill"	4
3	"Sweet Auburn"	3
4	"Grove Park"	3
5	"Downtown"	3

```

1 // Q4
2 // Find the types of crimes for each property type
3 MATCH (c:Crime)-[:HAPPENED_IN]→(p:PropertyType)
4 WITH p,propertyType AS PropertyType, COLLECT(c.crimeType) as CrimeTypes
5 RETURN DISTINCT PropertyType, CrimeTypes

```

PropertyType	CrimeTypes
"house_number"	["AGG ASSAULT", "ROBBERY-PEDESTRIAN", "LARCENY-FROM VEHICLE", "LARCENY-FROM VEHICLE", "LARCENY-NON VEHIC
"office"	["AUTO THEFT", "ROBBERY-PEDESTRIAN", "AUTO THEFT", "LARCENY-NON VEHICLE", "LARCENY-NON VEHICLE", "LARCENY-I
"shop"	["LARCENY-NON VEHICLE", "LARCENY-NON VEHICLE", "LARCENY-NON VEHICLE", "LARCENY-FROM VEHICLE", "LARCENY-NC
"road"	["BURGLARY-RESIDENCE", "AGG ASSAULT", "LARCENY-NON VEHICLE", "LARCENY-NON VEHICLE", "BURGLARY-NONRES", "L
"building"	["LARCENY-NON VEHICLE", "LARCENY-FROM VEHICLE", "BURGLARY-RESIDENCE", "ROBBERY-PEDESTRIAN", "LARCENY-FRC
"amenity"	["AUTO THEFT", "BURGLARY-RESIDENCE", "LARCENY-NON VEHICLE", "LARCENY-FROM VEHICLE", "LARCENY-NON VEHICLE"

Record fields have been truncated. Started streaming 13 records after 1 ms and completed after 4 ms.

```

1 // Q5
2 // Which month of a specified year has the highest crime rate? Return one record each
   for each beat
3 MATCH (c:Crime)-[:HAPPENED_ON]→(d:Date),
4 |   | (c)-[:HAPPENED_IN]→(b:Beat)
5 WHERE d.year = 2010
6 WITH b, d.month as month, COUNT(c) AS CrimeCount
7 ORDER BY b.beat, CrimeCount DESC
8 WITH b, COLLECT({month: month, CrimeCount: CrimeCount})[0] AS HighestMonth
9 RETURN b.beat AS beat, HighestMonth.month AS Month, HighestMonth.CrimeCount AS
   CrimeCount

```

PropertyType	CrimeTypes
"house_number"	["AGG ASSAULT", "ROBBERY-PEDESTRIAN", "LARCENY-FROM VEHICLE", "LARCENY-FROM VEHICLE", "LARCENY-NON VEHIC
"office"	["AUTO THEFT", "ROBBERY-PEDESTRIAN", "AUTO THEFT", "LARCENY-NON VEHICLE", "LARCENY-NON VEHICLE", "LARCENY-I
"shop"	["LARCENY-NON VEHICLE", "LARCENY-NON VEHICLE", "LARCENY-NON VEHICLE", "LARCENY-FROM VEHICLE", "LARCENY-NC
"road"	["BURGLARY-RESIDENCE", "AGG ASSAULT", "LARCENY-NON VEHICLE", "LARCENY-NON VEHICLE", "BURGLARY-NONRES", "L
"building"	["LARCENY-NON VEHICLE", "LARCENY-FROM VEHICLE", "BURGLARY-RESIDENCE", "ROBBERY-PEDESTRIAN", "LARCENY-FRC
"amenity"	["AUTO THEFT", "BURGLARY-RESIDENCE", "LARCENY-NON VEHICLE", "LARCENY-FROM VEHICLE", "LARCENY-NON VEHICLE"

Record fields have been truncated. Started streaming 13 records after 1 ms and completed after 4 ms.

```

1 // Q6
2 // What is the highest probability of crime severity(High, Medium, Low) for each zone?
3 MATCH (c:Crime)-[:HAPPENED_IN]→(b:Beat)
4 WITH b.zone AS zone, c.severity AS severity, count(*) AS frequency
5 ORDER BY zone, frequency DESC
6 RETURN zone, severity, frequency

```

	zone	severity	frequency
1	"1"	"Low"	27
2	"1"	"Medium"	13
3	"1"	"High"	2
4	"2"	"Low"	63
5	"2"	"Medium"	8
6	"2"	"High"	2
7			

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

```

1 // Q7 (Additional Question 2)
2 // Which crime nature (Property, Violent) has higher number of record for each
3 // property for each quater in a specific year?
4 MATCH (d1:Date {year:2012})←[:HAPPENED_ON]-(c1:Crime)-[:HAPPENED_IN]→
  (p1:PropertyType)
5 WHERE c1.crimeNature IN ["Violent","Property"]
6 RETURN d1.quarter AS quarter, p1.propertyType AS PropertyType , c1.crimeNature AS
  CrimeNature, COUNT(c1) AS NumberOfCrime
7 ORDER BY quarter, PropertyType

```

	quarter	PropertyType	CrimeNature	NumberOfCrime
1	1	"amenity"	"Property"	3
2	1	"building"	"Property"	3
3	1	"house_number"	"Property"	16
4	1	"house_number"	"Violent"	6
5	1	"leisure"	"Property"	1
6	1	"road"	"Property"	2
7				

Started streaming 28 records after 1 ms and completed after 4 ms.

Graph Data Science (GDS) Application

While discussing the data warehousing method, there were some science applications that could be used to find insights for answering business queries. One

of the applications of data warehousing that was implemented in the previous project is association rule mining, which showed hidden patterns, classification performance, and future prediction. In this project using the graph database method, there are some different algorithms that help to answer some demanding business inquiries, one of them is the maximum spanning tree.

The graph database is performed in Neo4j, there are some algorithm GDS packages that help finding insights. These algorithms have some traits which are suitable for this project graph database design: Directed, Heterogeneous and Weighted. Firstly, some algorithms (Depth First Search algorithm, Strongly Connected algorithm) in Neo4J are capable to work on a directed graph, for example, one crime node will have only 1 direction to each property, location, date, beat nodes. Secondly, the algorithms such as HashGNN node embedding algorithm are able to differentiate between nodes and relationships, this is called a heterogenous trait. Thirdly, the algorithms can take the benefits of nodes and relationship properties to convert them into weights to perform some useful search, some of the algorithms are shortest path algorithm or minimum spanning tree algorithm.

Maximum spanning tree

The maximum spanning tree algorithm is the spanning tree of a weighted graph, starts from a specified node to all its obtainable nodes and the set of relationships that connect the nodes all together with the largest possible weight. The result of the algorithm is the path with the maximum weight of each relationship.

In the crime graph database, the Maximum spanning tree algorithm is useful to find the number of crime incidents between 2 close locations. The specified node is a road and its reachable nodes are roads close to it. The weight in this case is the amount of common crime records between these 2 locations. Common crime records here means 2 roads need to have the same number of record of each crime type. If the weight of some pairs of roads are really high, it means that the road pairs are the hotspots of some criminal activities. This also gives more insights about the popular crime types happened in each pair to have some solution of reducing number of records in the future.

Degree Centrality

The Degree Centrality algorithm is an algorithm measuring the number of relationships that one node possessed. The relationships have 2 types: indegree

(from other nodes go to a specified node) or outdegree (from a specified node goes to other nodes).

In the current graph dataset, the Degree Centrality algorithm can be applied to evaluate the most popular property types with criminals. There are only crime attribute connected to property type attribute. Moreover, since there is only 1 type of relationship, from crime nodes to each property type node (outgoing), the number that reflects the popularity of a property with different crimes only, not be mixed with other attributes such as location, beat or date. Therefore, the Degree Centrality algorithm is suitable to answer correctly our business query.

Advantage of using Graph Data Science

The Degree Centrality and Maximum Spanning Tree algorithms show their advantages compared to some processes such as association rule mining, analytical processing of data warehousing method in transforming data into insights. With some data warehousing processes, the relationships between different locations or crime and property are not visible unless queries are applied some complex join operations. Additionally, finding results using data warehousing method requires complicated queries and many aggregations. While with graph database, there is existence of GDS library and it is easy to apply the algorithm directly to the project, makes the works become much less complexity, much faster and more efficient. The results of running the algorithm are the quantities or weights of the relationships, which are difficult to perform on data warehousing method because graph database are strong at showing the relationship between different attributes. Moreover, when more nodes and relationships are added in the future, the algorithms on graph database can easily handle the change and show the results right away compared to re-run the queries in data warehousing process.

Conclusion

Overall, this project shows how useful the graph database is. It can be seen through how the graph database is designed, implemented, some advantages and disadvantages of using it compared to relational data warehousing. It also showed how some business questions related to crimes are answered. The graph database science is also illustrated to see how some insights can be found using some algorithms such as Maximum spanning tree and Degree Centrality.

