CITS5504 DATA WAREHOUSING

PROJECT 2 – GRAPH DATABASE

Jiaze Li 23266049

Lei Chen 23623238

YouTube video link:
https://youtu.be/mGJMqLv3FwY

1 Design and Implementation

The Crime dataset is a complex and large dataset that contains a lot of information about criminal activities in a particular city over a period. To analyze and extract useful insights from this dataset, a property graph database can be used. In this report, we discuss the design of such a property graph database for the Crime dataset using the Arrows App to draw the nodes and relations.

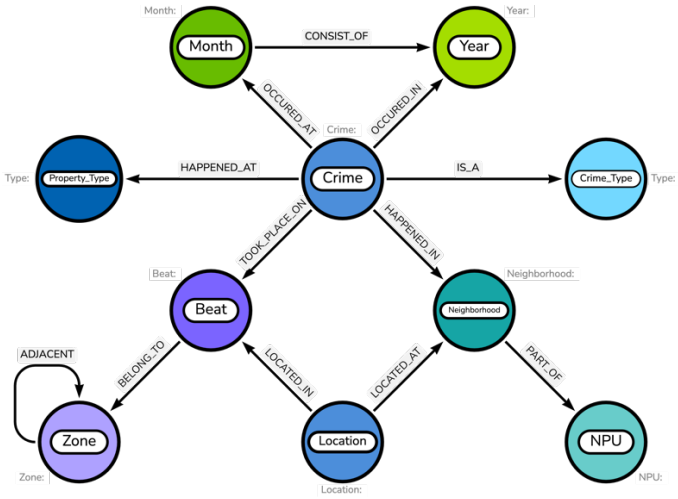


Figure 1. Graph Database Design

1.1 Nodes Design

In this crime dataset, the Neighborhood Planning Unit (NPU) represents the local community organization in the Atlanta area (Reference: [1]), while the Beat node signifies the territory assigned to a police officer for patrolling (Reference: [2]). By analyzing and connecting NPUs, neighborhoods, beats, and zones, the dataset provides a comprehensive representation of crime incidents and their associated information, offering valuable insights for future reports and decision-making.

Each crime incident, represented as a unique Crime node, is associated with a specific type of property, captured in the Property_Type node, which indicates the nature of the location where the crime took place, such as residential, commercial, etc.

Furthermore, the time of the crime occurrence is depicted through the Month and Year nodes, encapsulating the month and year of the crime respectively. The Crime_Type node provides a classification for the nature of the crime committed.

Finally, the Location node holds the specific area description of the crime scene, thereby providing a precise understanding of where the crime took place. Through the analysis of these nodes, the dataset presents a detailed mapping of crime incidents across various geographic and temporal parameters, aiding in effective crime pattern recognition and predictive policing.

1.2 Relations Design

To satisfy the basic requirements of queries, this project considers establishing eight relationships between eight nodes.

- **OCCURRED AT:** This relationship exists between Crime and Month nodes, indicating when the crime occurred.
- **OCCURRED IN:** It links the Crime and Year nodes, specifying the year the crime took place.
- **CONSIST OF**: The Month and Year nodes are linked by this relationship, showing that a particular Month consists of a specific Year.
- **LOCATED IN**: This relationship exists between the Location and Beat nodes, indicating the geographical location of the Beat.
- **BELONG TO**: This relationship connects a Beat node to a Zone node, indicating that the Beat belongs to the Zone.
- **LOCATED AT**: This relationship is between the Location and Neighborhood nodes, indicating the geographical location of the neighborhood.
- **PART OF**: This relationship connects a Location node to an NPU node, indicating that the neighborhood is part of the NPU.
- **HAPPENED IN**: It's a link between Crime and Neighborhood nodes, signifying that the crime happened in a certain Neighborhood.
- **HAPPENED AT**: This is a relationship between the Crime and PropertyType nodes, demonstrating that the crime happened at a certain type of property.
- **IS A**: This relationship connects a Crime node to a CrimeType node, indicating what type of crime it was.
- **TOOK PLACE ON**: This relationship connects a Crime node to a Beat node, indicating which beat the crime took place on.
- **ADJACENT**: This relationship connects a Zone node to another Zone node, indicating that the two Zones are adjacent to each other.

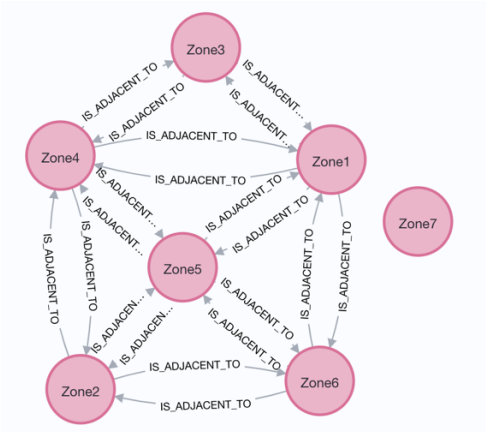


Figure 2. Zone nodes and their relationship

1.2.1 When to use a property rather than a node:

Choosing between a property and a node in a graph database is largely dictated by the complexity of the given data, the type of relationships it has with other data, and the query needs. In our case, the decision to separate 'Year' and 'Month' into individual nodes versus keeping them as properties within a 'Time' node hinges on the data analysis requirements.

In our crime dataset has a lot of time-based querying or time-based relationships (e.g., finding trends in crime rates over period), separating the 'year' and 'month' into individual nodes could be more advantageous. On the other hand, if these queries or relationships aren't as prevalent, keeping them as properties within a 'time' node would be simpler and more efficient.

1.2.2 When to add properties to a relation:

When it comes to adding properties to a relationship, it's generally about adding more context or detail to that relationship. Here are some scenarios when we need to add properties to a relationship:

- **Detailing the Relationship:** When we want to add more context or details about the relationship between two nodes, properties can be added to the relationship. For example, in the crime dataset, a property on the HAPPENED_IN relationship between Crime and Neighborhood nodes could indicate the time or date the crime happened, if this is not captured elsewhere.
- **Temporal Context:** When there's a temporal aspect to the relationship, properties can be useful. For example, if we want to capture when a crime first started occurring in a specific Beat, we could add a Start_Date property to the TOOK_PLACE_ON relationship between Crime and Beat.
- **Quantifying the Relationship:** When we want to represent the strength or weight of the relationship, adding a property can be beneficial. For instance, a Frequency property on the IS_A relationship between Crime and Crime_Type could indicate how often a certain type of crime occurs.
- **Status or State:** Properties can be added to relationships to denote the state of a relationship. For example, in the crime dataset, if we want to track the investigation status of a crime, we could add an Investigation_Status property to the IS_A relationship between Crime and Crime_Type.

1.2.3 When to add extra nodes and for what type of queries:

While adding extra nodes can provide additional detail and flexibility, it can also add complexity to our model. It's important to consider the cost-benefit tradeoff based on

the specific needs and the queries. Here are some scenarios when we need to add extra nodes for different queries:

- **Increase Query Efficiency**: Adding extra nodes can enhance performance and efficiency for certain types of queries. For instance, if the frequently query for crimes by a specific 'Day of the Week', instead of scanning through the entire 'Crime' node, it might be more efficient to have a separate 'Day' node connected to the 'Crime' node.
- **Additional Details/Complexity**: Extra nodes can also be added to capture more complex details in the dataset. For example, if we start collecting data about 'Police Officers' who oversee investigating the crimes, adding a 'Police Officer' node can help manage this new information and relate it to existing 'Crime' or 'Beat' nodes.
- **Expand Scope of Analysis**: When we want to expand the scope of the analysis, extra nodes can come in handy. We might decide to analyze 'Weapon' used in the crimes. By adding a 'Weapon' node and linking it to the 'Crime' node which can easily perform queries like "Find all crimes where a firearm was used".
- **Handling Hierarchical Data**: Extra nodes can be beneficial in creating hierarchical relationships. For example, if we want to further break down the 'Location' into 'Street', 'City', 'State', and 'Country', having separate nodes for these can maintain the hierarchy and allow more granular queries.
- **Enable New Types of Relationships**: Extra nodes can allow new types of relationships to be made. For example, if we introduce a 'Victim' node, we can establish relationships not just between the 'Victim' and the 'Crime', but also potentially with the 'Location', 'Beat', or 'Neighborhood' depending on the dataset.

1.3 Graph Database ETL using different tools

Method 1: Arrow tool import

a) Load the data into the graph database

We first save the file into Neo4j, then refresh the database to ensure that the data is read in.

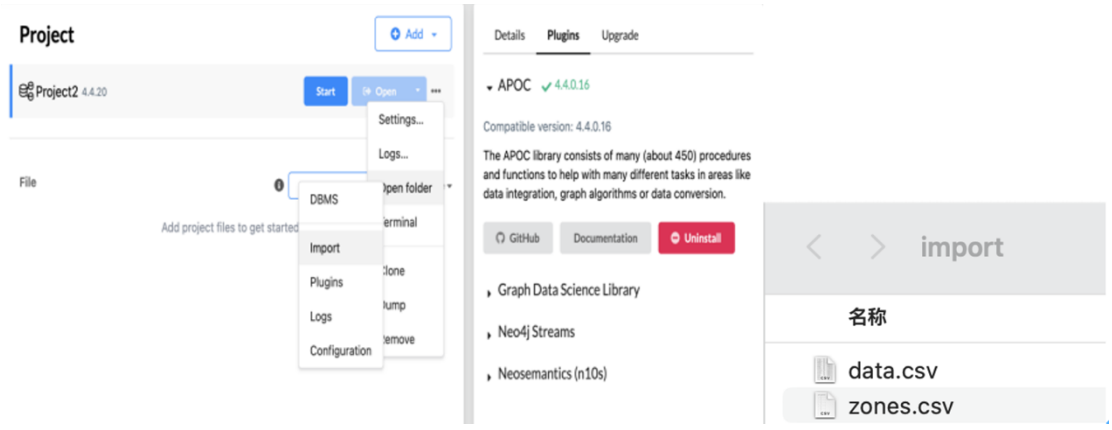


Figure 3. Neo4j ETL and import data

b) Importing data, creating nodes, and establishing relationships

- For the special relationship table (*data.csv and zones.csv*)

data.csv

CrimeType	Location	Beat	Neighborhood	NPU	PropertyType	Year	Month	Zone	Crime	ZoneKey
BURGLARY-RESIDENCE	242 ANDERSON AVE NW	108	Dixie Hills	J	house_number	2011	10	Zone1	1	1
AUTO THEFT	1270 CAROLINE ST NE	609	Edgewood	O	amenity	2012	10	Zone6	2	6
LARCENY-FROM VEHICLE	3180 ARDLEY RD SW	406	East Ardley Road	I	house_number	2016	10	Zone4	3	4
AGG ASSAULT	738 PRINCE PL NW	109	Center Hill	J	neighbourhood	2016	10	Zone1	4	1
AUTO THEFT	2909 CAMPBELLTON RD SW	409	Southwest	R	house_number	2010	10	Zone4	5	4

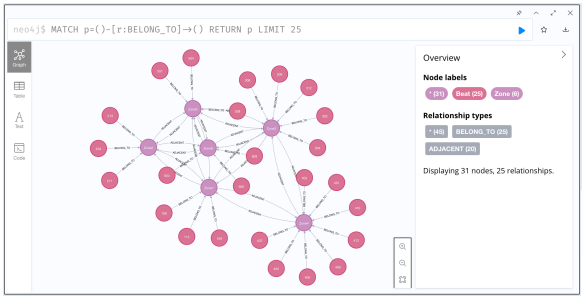
zones.csv

ZoneKey	AdjacentZoneKey
1	2
1	3
1	4
1	5
2	1

Figure 4. All data table and AdjacentZone table(zones.csv)

To achieve this design, we split the raw data into 10 NODE tables and 12 Relational tables and set up special tables for regions and adjacent regions to answer questions. The establishment of the relational table is like the establishment of primary and foreign keys and is used to form a correspondence between the different ids of two nodes. The following example shows two node tables, Adjacent Zone, and Zone, and the "ADJACENT" table that describes the relationship between them.

- For the other tables we use import data, create nodes, and establish relationships by Cyber codes.




```
LOAD CSV WITH HEADERS FROM 'file:///data.csv' AS row
MERGE (z:Zone {Police_Area: "Zone" + row.ZoneKey})
MERGE (b:Beat {Beat: row.Beat})

MERGE (b)-[:BELONG_TO]->(z)
```

Figure 5. Example of creating nodes and relationship by cyber codes

We import Zone and Beat data from the CSV file and create nodes and relationships for the data in the Neo4j database. By creating the Zone and Beat nodes and the relationships between them, we can better represent and query these entities and their associations in the database.

```
$ LOAD CSV WITH HEADERS FROM 'file:///data.csv' AS row MERGE (z:Zone {Police_Area: "Zon…
```

Added 2232 labels, created 2232 nodes, set 2232 properties, created 8096 relationships, completed after 3500 ms.

```
neo4j$ LOAD CSV WITH HEADERS FROM 'file:///zones.csv' AS row MERGE (z1:Zone {Police_Are…
```

Created 20 relationships, completed after 110 ms.

Figure 6. Creating nodes, importing data, and establishing relationships successfully

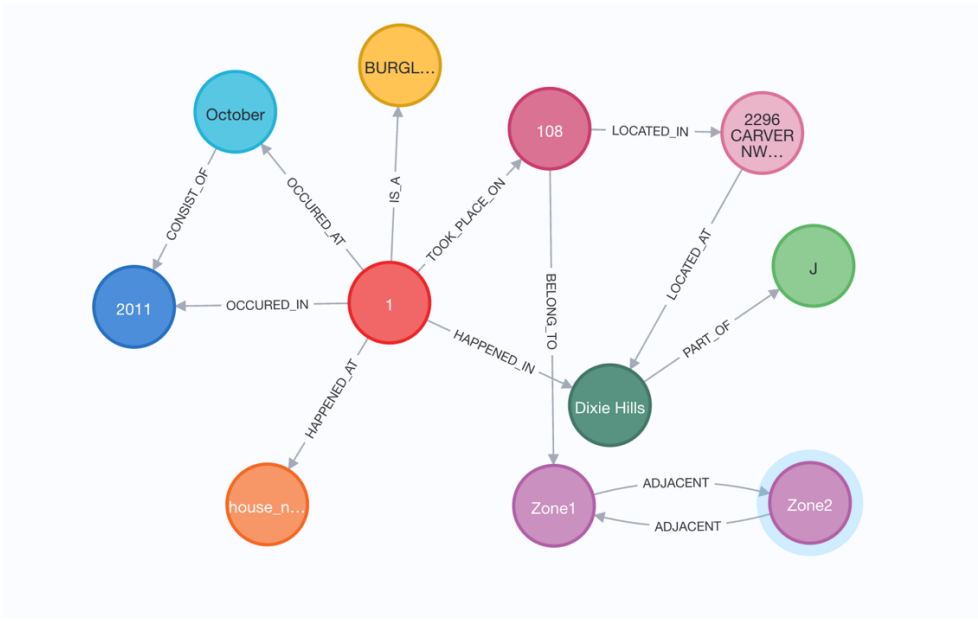


Figure 7. Graph Database Example

Method 2: Direct ETL import

a) Creating the target database

In DataGrip, we create a new database and its schema using the SQL console.

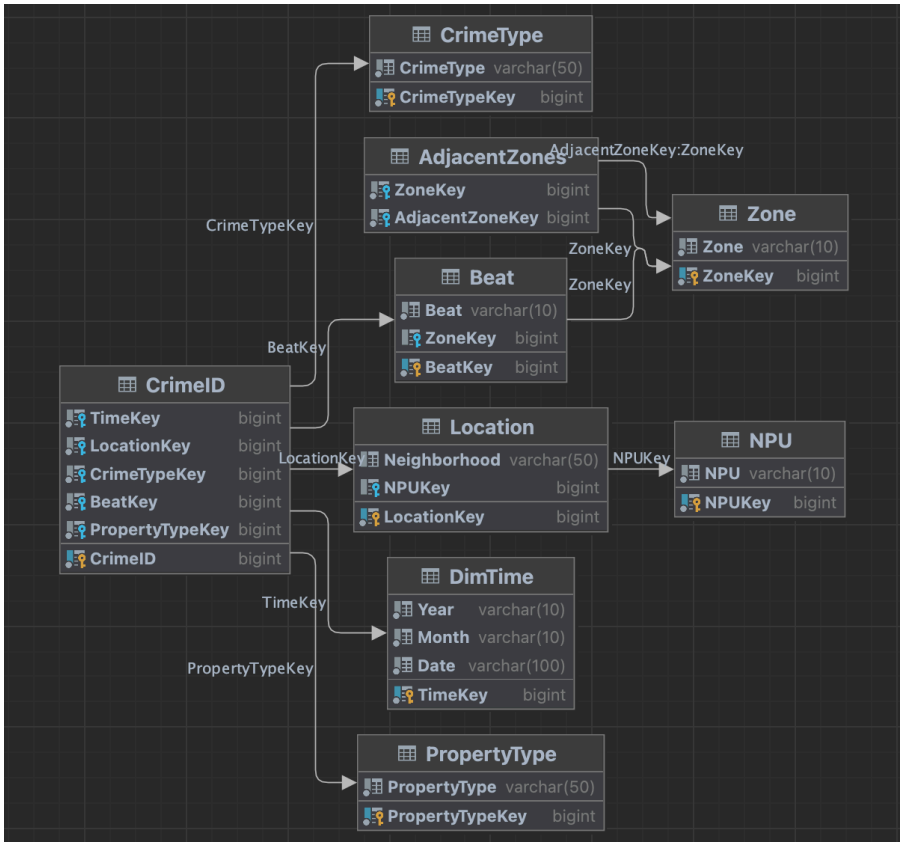


Figure 8. Show ER Diagram in DataGrip

b) Transform and clean the data to conform to the target database schema.

We generated the required data tables using Python scripts and imported them into DataGrip.

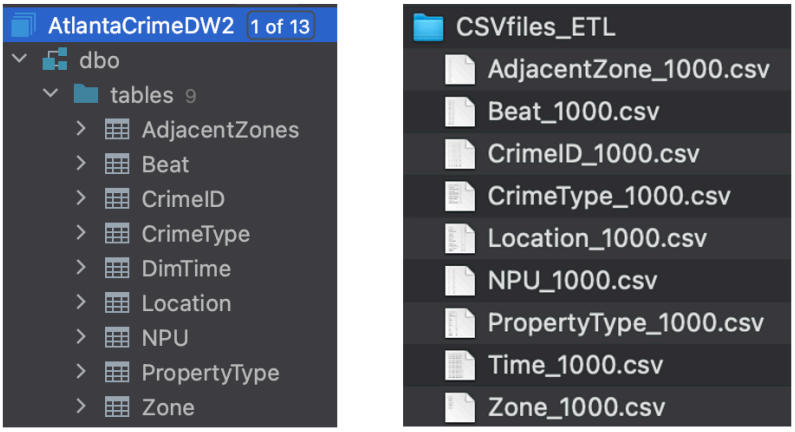


Figure 9. Creating tables and import data in DataGrip

c) Load the data into the target graph database using Neo4j ETL tool.

We established a connection between our database and the Neo4j ETL tool using a JDBC driver. This allowed us to import our data into the Neo4j graph database, facilitating seamless data transfer and ensuring that our data model and relationships were accurately represented in the graph database.

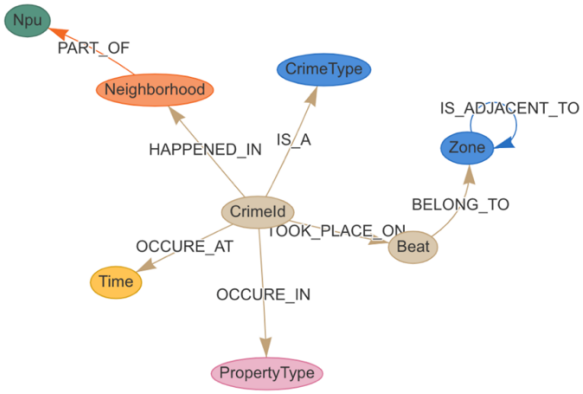


Figure 10. Check relationships and MAPPING in Neo4j

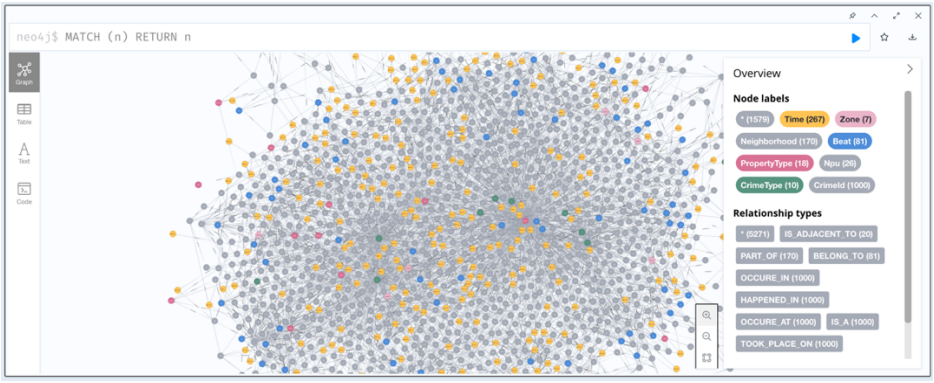


Figure 11. Insert successfully and explore the data in GraphDB

1.4 Conclusion

1.4.1 Graph Database Design using the Arrow tool in comparison with direct ETL import

While direct ETL import may be a more familiar way to traditional data warehousing, graph database design using the Arrow tool allows for a more intuitive, visual approach to designing the graph schema and adding nodes and relationships. Additionally, the Arrow tool offers the ability to quickly modify the graph schema and preview changes in real-time, making it a more agile approach to graph database design.

Arrow Tool with Cypher Queries:

Pros:

- **Streamlined Process**: We can import a single, comprehensive CSV file, and then use Cypher queries to create nodes and relationships directly in Neo4j. This eliminates the need for extensive pre-processing or multiple CSV files.
- **Greater Flexibility**: If we need to add a new node or relationship, we can do so simply by modifying the Cypher queries, rather than needing to reprocess the entire dataset. This makes the model much more adaptable to evolving requirements.
- **Visual Design**: The Arrow tool provides a visual way to design and validate the graph model before starting the data import.

Cons:

- **Limited to Design**: While the Arrow tool can help with designing the graph model, it does not directly assist in data transformation or loading.

Direct ETL Import:

Pros:

- **Automated and Repeatable**: Once established, the process can be automated and executed as required.

Cons:

- **Complexity**: Preparing the data requires a significant amount of work. For each node, a separate CSV file must be generated. This can involve creating unique keys for each table to establish relationships, which can increase the complexity of the table design and make the process time-consuming.
- **Less Flexibility**: If we need to add a new node to the graph, we'll need to reprocess the data, create a new CSV, and rebuild relationships in DataGrip. This means repeating the entire process, reducing reusability, and making the model less adaptable to changes in requirements.

- **Higher Technical Requirements**: This approach requires advanced skills in several different tools and languages (Python for data processing, SQL for database creation, and the ETL tool for data loading).

In summary, while direct ETL import provides a high degree of control, it can be complex and time-consuming, especially when the graph model needs to be updated or changed. On the other hand, the combination of the Arrow tool and Cypher queries offers a more streamlined, flexible approach, making it easier to adapt the model to changing requirements. This can save significant time and effort, especially in dynamic scenarios where the graph model needs to be regularly updated or expanded.

1.4.2 Design choices with pros and cons identified

The graphic database of this project considers the design of a wide variety of nodes and rich relationships between nodes, which naturally brings benefits:

Pros:

Extensibility:
The design of various nodes and relationships between them enables the database to represent more complex relationships and allows it to evolve as the requirements change. For example, if new data about 'Weapons' or 'Victims' become available, they can be added as new nodes. Similarly, if the 'Location' needs to be more detailed, new nodes like 'Street' or 'City' can be added.

Furthermore, consider the association between Neighborhood Planning Unit (NPU) and the neighborhood itself. A straightforward "PART_OF" relationship can effectively accommodate the querying functionality by designing Neighborhood as an attribute of the NPU. However, the merits of conceptualizing NPU as a node extend beyond this. By creating NPU as a node, it enables us to establish intricate relationships with the neighborhood. For instance, in the realm of administrative divisions, a neighborhood might straddle the border of two NPUs, resulting in a single neighborhood being part of two distinct NPUs.

Richness:
The graph database can represent rich relationships between nodes, such as the "BELONG_TO" relationship between Beat and Zone nodes, the "PART_OF" relationship between Neighborhood and NPU nodes, and the "HAPPENED_AT" relationship between Crime and PropertyType nodes. This enables a wide range of queries to be supported.

Flexibility:
Since each node has its own properties and relationships, the database can be easily modified or extended to include new nodes or relationships. This flexibility makes it

easier to adapt the database to new requirements or changes in the data model, like 'Year' and 'Month', or 'Neighborhood' and 'NPU', allows for flexible querying. Depending on the analysis, one can use either broad or granular nodes.

Detailed Analysis:

The graph's structure allows for granular analysis on different dimensions - temporal (with 'Month' and 'Year' nodes), geographic (with 'NPU', 'Neighborhood', 'Beat', and 'Zone' nodes), and by crime characteristics (with 'Crime_Type' and 'Property_Type' nodes). This can yield comprehensive insights.

Cons:

Redundancy:

The design of multiple nodes and relationships can lead to some redundancy in the data model. Some nodes could have been simplified as properties, especially for those with one-to-one mapping. This might lead to increased complexity in data preprocessing and updating.

NODE	COUNT
Crime	1000
Location	917
Neighborhood	163
Beat	81
NPU	26
PropertyType	18
CrimeType	10
Zone	7
TOTAL	2232

RELATIONSHIP	COUNT
HAPPENED_IN	1000
HAPPENED_AT	1000
OCCURED_AT	1000
OCCURED_IN	1000
TOOK_PLACE_ON	1000
IS_A	1000
LOCATED_IN	919
LOCATED_AT	917
PART_OF	170
BELONG_TO	81
ADJACENT	20
CONSIST_OF	9
TOTAL	8116

Figure 12. Number of all nodes and relationships in GraphDB

Query complexity:

The use of multiple nodes and relationships can result in complex and lengthy queries. For example, the "MATCH" statement in Cypher may require long paths, with parameters for each node and relationship clearly defined in the query code. This can increase the query workload and make it harder to understand and maintain the queries.

Performance:

The use of multiple nodes and relationships can potentially lead to performance issues, as queries may need to traverse more nodes and relationships to retrieve the required information. This can lead to longer query execution times and increased resource usage, especially in large-scale graph databases.

Resource-Intensive:

The numerous nodes and relationships in the graph may require substantial storage space and computational power. This could potentially lead to slower query performance and increased costs, particularly as the dataset grows.

2 Results: Cypher queries

2.1 Queries Outline

1. **Origin Queries:**

- How many crimes are recorded for a given crime type in a specified neighbourhood for a particular period?
- Find the neighbourhoods that share the same crime types, organise in decending order of the number of common crime types.
- Return the top 5 neighbourhoods for a specified crime for a specified duration.
- Find the types of crimes for each property type.
- Which month of a specified year has the highest crime rate?
- Find the zones that have are adjacent and sharing the same high crime months.

2. **Extra Queries**

- Query 1: The total number of crimes each NPU
- Query 2: Based on Query 1, find the NPU with the highest number of crimes and list the top five neighbourhoods with the highest number of crimes in that NPU
- Query 3: Find the average number of crimes per NPU, sorted by crime type
- Query 4: All crime types that occurred in 2012 and their corresponding crimes

In the following section, we will delve into the practical implementation of the graph database, demonstrating how each query corresponds to a particular graph. We'll explore the logic behind the "MATCH" statement in Cypher, which serves to identify relevant paths, nodes, relationships, and attributes. For each query proposed, we'll present its corresponding results and provide an analytical result.

2.2 The Result of Each Queries

1. Origin Queries:

- **How many crimes are recorded for a given crime type in a specified neighborhood for a particular period?**

Purpose:

To query the number of LARCENY-FROM VEHICLE crimes in the "Downtown" neighborhood between January 2009 to January 2010. (For January 2010 is not included thus actually January-December in 2009).

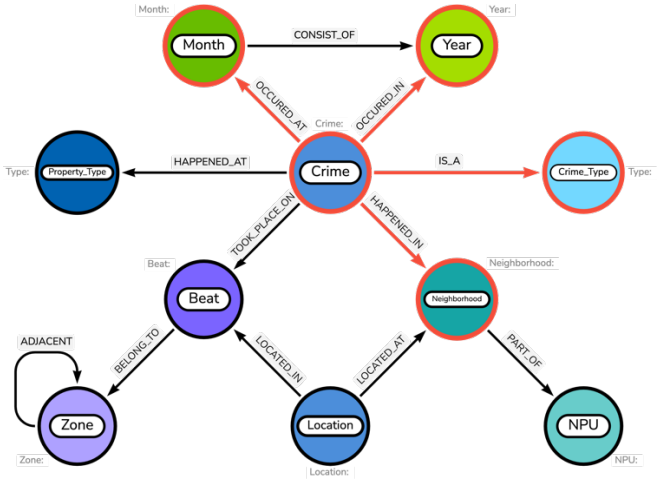


Figure 13. GraphDB for Query1

Query process:

Match the crime record with the type, location, and time of the crime.

The restricted type of offence is "LARCENY-FROM VEHICLE", the location is "Downtown", and the time is in between January 2009 to January 2010 (January 2010 is not included). The number of criminal records returned.

```
MATCH (ct:Crime_Type {Type: 'LARCENY-FROM VEHICLE'})<-[:IS_A]-(c:Crime)-[:OCCURED_IN]->(y:Year),
(c)-[:OCCURED_AT]->(m:Month)
MATCH (n:Neighborhood {Neighborhood: 'Downtown'})
MATCH (c)-[:HAPPENED_IN]->(n)
WHERE (y.Year = '2009' AND m.Month >= '1') OR (y.Year = '2010' AND m.Month < '1')
RETURN count(c) AS Crimes_Count
```

Crimes_Count
9

Started streaming 1 records in less than 1 ms and completed after 1 ms.

Figure 14. Result for Query1

Analysis of results:

Searching for specific types of crimes that occurred in downtown blocks in 2009, we selected vehicle theft as the crime type. The results of the inquiry indicate that during the specified period is starts in January 2009 and ends before January 2010 (It's January-December 2009), there were nine crimes of this type (vehicle theft) in the downtown block.

Discussion:

In our initial data model, we represented the 'Month' attribute as a string format, such as 'January', 'February', etc. We soon realized, however, that this format was not suitable for our needs when querying a specific time range down to the month. Specifically, we found that it's not straightforward to compare string-based months, as we would need to translate them into numeric values to make such comparisons.

To solve this issue and provide the flexibility we needed for our queries, we decided to change the 'Month' attribute to a numeric format, ranging from 1 (representing January) to 12 (representing December). This adjustment enabled us to easily query a specific time range down to the month, enhancing the precision of our analysis.

By transitioning the 'Month' attribute to numeric format, we made our data model more adaptable for the type of analysis we intended to conduct. It served as a reminder of the importance of carefully considering the data format and structure in the early stages of data modeling and making necessary changes promptly as the requirements evolved.

- **Find the neighborhoods that share the same crime types, organise in descending order of the number of common crime types.**
-

Purpose:

To find the number of crime types shared neighborhoods and arrange them in descending order.

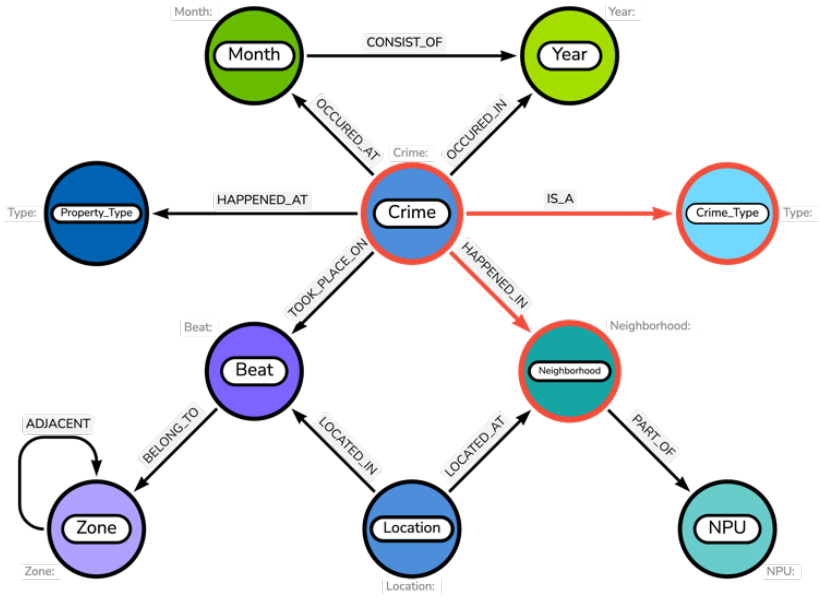
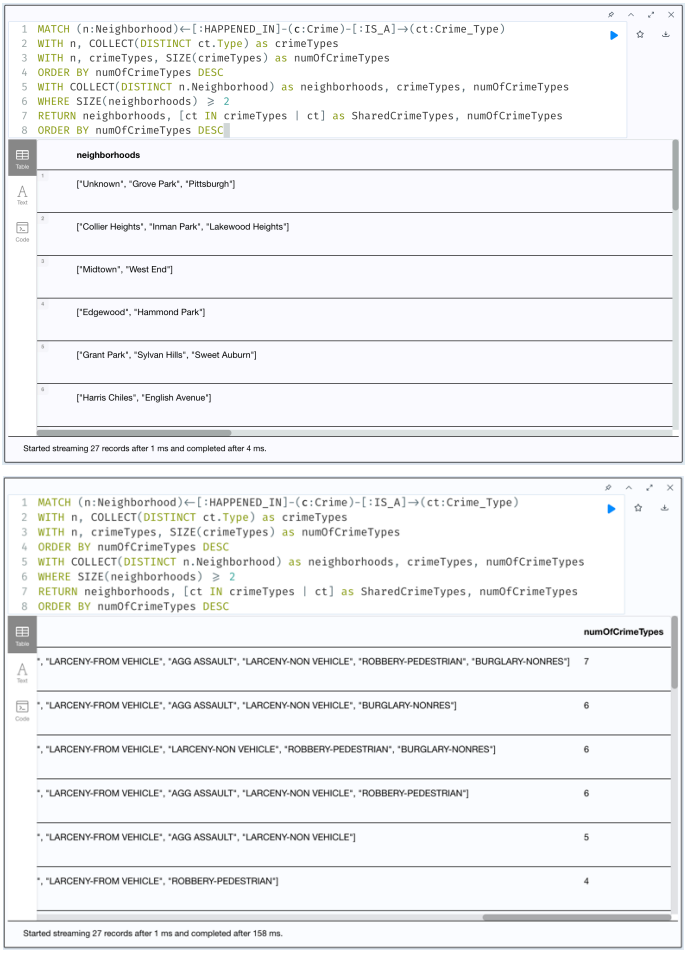


Figure 15. GraphDB for Query2

Query process:

First, find all the crimes that occurred in each neighborhood and collects the types of these crimes. Then, calculates the number of crime types that occurred in each neighborhood and sorts them in descending order by this number. Next, filters out neighborhoods where at least two types of crimes have occurred and collects these neighborhoods and their corresponding crime types and numbers. Finally, returns these neighborhoods, the types of crimes they share, and the number of crime types, and sorts them in descending order by the number of crime types.



```
MATCH (n:Neighborhood)←[:HAPPENED_IN]-(c:Crime)-[:IS_A]→(ct:Crime_Type)
WITH n, COLLECT(DISTINCT ct.Type) as crimeTypes
WITH n, crimeTypes, SIZE(crimeTypes) as numOfCrimeTypes
ORDER BY numOfCrimeTypes DESC
WITH COLLECT(DISTINCT n.Neighborhood) as neighborhoods, crimeTypes, numOfCrimeTypes
WHERE SIZE(neighborhoods) ⩾ 2
RETURN neighborhoods, [ct IN crimeTypes | ct] as SharedCrimeTypes, numOfCrimeTypes
ORDER BY numOfCrimeTypes DESC
```

	neighborhoods
1	["Unknown", "Grove Park", "Pittsburgh"]
2	["Collier Heights", "Inman Park", "Lakewood Heights"]
3	["Midtown", "West End"]
4	["Edgewood", "Hammond Park"]
5	["Grant Park", "Sylvan Hills", "Sweet Auburn"]
6	["Harris Chiles", "English Avenue"]

Started streaming 27 records after 1 ms and completed after 4 ms.

```
MATCH (n:Neighborhood)←[:HAPPENED_IN]-(c:Crime)-[:IS_A]→(ct:Crime_Type)
WITH n, COLLECT(DISTINCT ct.Type) as crimeTypes
WITH n, crimeTypes, SIZE(crimeTypes) as numOfCrimeTypes
ORDER BY numOfCrimeTypes DESC
WITH COLLECT(DISTINCT n.Neighborhood) as neighborhoods, crimeTypes, numOfCrimeTypes
WHERE SIZE(neighborhoods) ⩾ 2
RETURN neighborhoods, [ct IN crimeTypes | ct] as SharedCrimeTypes, numOfCrimeTypes
ORDER BY numOfCrimeTypes DESC
```

		numOfCrimeTypes
1	", "LARCENY-FROM VEHICLE", "AGG ASSAULT", "LARCENY-NON VEHICLE", "ROBBERY-PEDESTRIAN", "BURGLARY-NONRES"]	7
2	", "LARCENY-FROM VEHICLE", "AGG ASSAULT", "LARCENY-NON VEHICLE", "BURGLARY-NONRES"]	6
3	", "LARCENY-FROM VEHICLE", "LARCENY-NON VEHICLE", "ROBBERY-PEDESTRIAN", "BURGLARY-NONRES"]	6
4	", "LARCENY-FROM VEHICLE", "AGG ASSAULT", "LARCENY-NON VEHICLE", "ROBBERY-PEDESTRIAN"]	6
5	", "LARCENY-FROM VEHICLE", "AGG ASSAULT", "LARCENY-NON VEHICLE"]	5
6	", "LARCENY-FROM VEHICLE", "ROBBERY-PEDESTRIAN"]	4

Started streaming 27 records after 1 ms and completed after 158 ms.

Figure 16. Result for Query2

Analysis of results:

We identified neighbourhoods that shared the same type of crime, and from this result, we could observe that there might be some correlation between these neighbourhoods, such as socioeconomic conditions, policing conditions, etc. For example, if two neighbourhoods or more blocks share many of the same types of crime, then it may mean that they face similar policing problems and may require similar preventive measures and law enforcement strategies.

- **Return the top 5 neighborhoods for a specified crime for a specified duration.**

Purpose:

Check the top 5 neighborhoods where "LARCENY-FROM VEHICLE" crimes occurred in between January 2009 to January 2010. (For January 2010 is not included thus actually January-December in 2009), ranked in descending order by number of crimes.

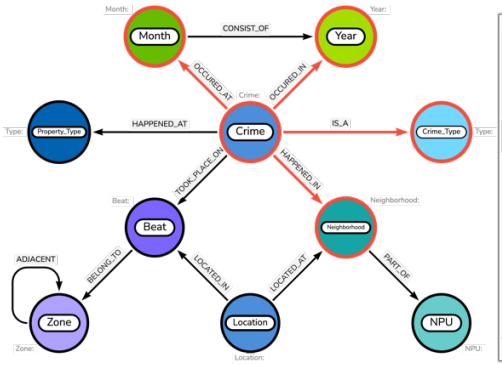
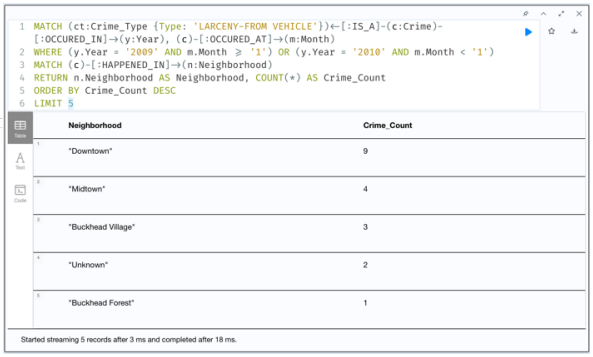


Figure 17. GraphDB for Query3



```
MATCH (ct:Crime_Type {Type: 'LARCENY-FROM VEHICLE'})<-[:IS_A]-(c:Crime)-
  [:OCCURED_IN]->(y:Year), (c)-[:OCCURED_AT]->(m:Month)
WHERE (y.Year = '2009' AND m.Month >= '1') OR (y.Year = '2010' AND m.Month < '1')
MATCH (c)-[:HAPPENED_IN]->(n:Neighborhood)
RETURN n.Neighborhood AS Neighborhood, COUNT(*) AS Crime_Count
ORDER BY Crime_Count DESC
LIMIT 5
```

Neighborhood	Crime_Count
"Downtown"	9
"Midtown"	4
"Buckhead Village"	3
"Unknown"	2
"Buckhead Forest"	1

Started streaming 5 records after 3 ms and completed after 18 ms.

Figure 18. GraphDB for Query3

Query process:

Match the crime record with the type, location, and year of the crime. The restricted crime type is "LARCENY-FROM VEHICLE" and the time period is between January 2009 to January 2010(January 2010 is not included. And we use the [(y.Year = '2009' AND m.Month >= '1') OR (y.Year = '2010' AND m.Month < '1')]. The number of crimes is calculated by neighborhood groups. Returns the name of the neighborhood and the number of crimes, in descending order by the number of crimes, with only the first 5 results returned.

Analysis of results:

The purpose of this query is to find the neighborhoods that had the highest number of vehicle theft crimes in between January 2009 to January 2010 (January 2010 is not included), to see which areas faced greater problems with such crimes that year. According to the results, the Downtown neighborhood had the highest number of vehicle thefts in 2009, with nine. Midtown was next with four crimes. This means that in 2009, vehicle theft in these neighborhoods was a relatively high problem.

- **Find the types of crimes for each property type.**

Purpose:

Look up all types of crime for each property type.

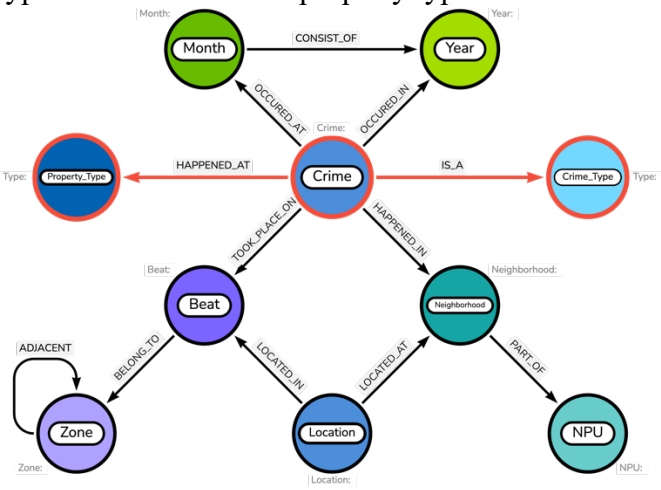


Figure 19. GraphDB for Query4

Query process:

Matching crimes to property types and crime types.

Group by property type and collect all crime types.

Returns a list of property types and corresponding crime types.

```
MATCH (p:Property_Type)-[:HAPPENED_AT]-(c:Crime)-[:IS_A]→(ct:Crime_Type)
RETURN p.Type AS Property_Type, collect(distinct ct.Type) AS Crime_Types
```

	Property_Type	Crime_Types
1	"house_number"	["BURGLARY-RESIDENCE", "AUTO THEFT", "LARCENY-FROM VEHICLE", "AGG ASSAULT", "LARCENY-NON VEHICLE", "ROBBE
2	"building"	["BURGLARY-RESIDENCE", "AUTO THEFT", "LARCENY-FROM VEHICLE", "AGG ASSAULT", "LARCENY-NON VEHICLE", "ROBBE
3	"neighbourhood"	["BURGLARY-RESIDENCE", "LARCENY-FROM VEHICLE", "AGG ASSAULT", "LARCENY-NON VEHICLE"]
4	"amenity"	["BURGLARY-RESIDENCE", "AUTO THEFT", "LARCENY-FROM VEHICLE", "AGG ASSAULT", "LARCENY-NON VEHICLE", "ROBBE
5	"city"	["BURGLARY-RESIDENCE", "AUTO THEFT", "LARCENY-FROM VEHICLE", "LARCENY-NON VEHICLE"]
6	"road"	["BURGLARY-RESIDENCE", "AUTO THEFT", "LARCENY-FROM VEHICLE", "AGG ASSAULT", "LARCENY-NON VEHICLE", "ROBBE

Started streaming 18 records after 1 ms and completed after 3 ms.

Figure 20. GraphDB for Query4

Analysis of results:

The results are clear in the graph, and we can see what types of crimes are occurring in different types of properties. This information is useful for analyzing crime patterns and developing preventive measures and enforcement strategies accordingly.

- **Which month of a specified year has the highest crime rate? Return one record each for each beat. (PG students also need to return one record for each zone.)**

1) Record for each beat：

Purpose:

Search for beat with the highest number of crimes per month in 2009.

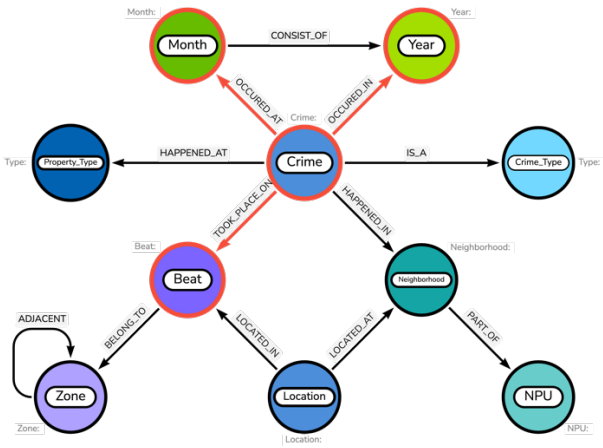


Figure 21. GraphDB for Query5.1

Query process:

First matches all crime events in 2009 by concatenating the crime, beat, property type, month, and year nodes. Then, calculates the total number of crimes for each beat and month. Finally, returns the beat and month with the highest number of crimes and the corresponding number of crimes, sorted from highest to lowest crime count. If there are multiple beat or months with the highest crime counts, it returns only one (the first police district and month with the highest crime count).

```
MATCH (b:Beat)-[:BELONG_TO]→(z:Zone),
      (c:Crime)-[:TOOK_PLACE_ON]→(b),
      (p:Property_Type)←[:HAPPENED_AT]-(c),
      (m:Month)←[:OCCURED_AT]-(c),
      (y:Year {Year: '2009'})←[:OCCURED_IN]-(c)
WITH b, m, COUNT(c) AS crimes
RETURN b.Beat AS Beat, m.Month AS Month, MAX(crimes) AS Highest_Crime_Rate
ORDER BY Highest_Crime_Rate DESC
LIMIT 1
```

Beat	Month	Highest_Crime_Rate
"511"	"10"	6

Started streaming 1 records after 1 ms and completed after 11 ms.

Figure 22. GraphDB for Query5.1

Analysis of results:

October - Beat (511)-6 crimes. According to the query results, we can see that District 511 had the highest number of crimes in October 2009, with a total of 6 crimes. This query helps us identify which beat has the highest crime rates during a given period.

2) Record for each zone：

Purpose:

Query the zone with the highest number of crimes per month in 2009.

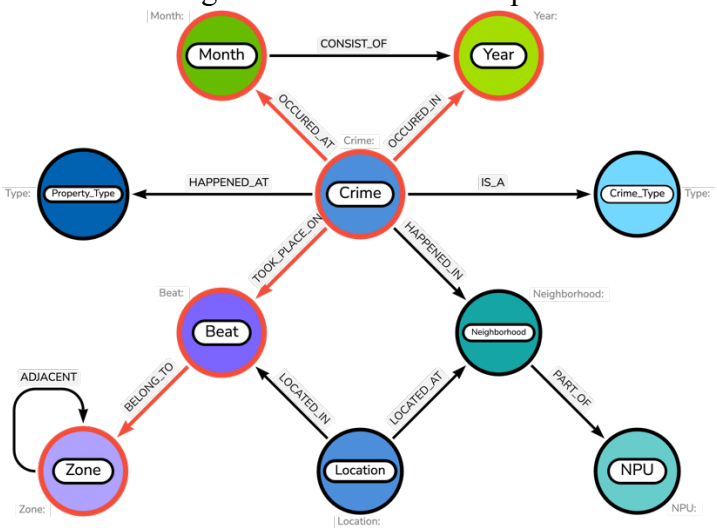


Figure 23. GraphDB for Query5.2

Query process:

First matches all crime incidents in 2009 by connecting the crime, zone, property type, month, and year nodes. Then, it calculates the total number of crimes for each zone and month. Finally, it returns the zone and month with the highest number of crimes, and the corresponding number of crimes, sorted from highest to lowest crime count. If there are multiple police districts or months with the highest crime counts, it returns only one (the first zone and month with the highest crime count).

```
MATCH (b:Beat)-[:BELONG_TO]→(z:Zone),
      (c:Crime)-[:TOOK_PLACE_ON]→(b),
      (p:Property_Type)←[:HAPPENED_AT]-(c),
      (m:Month)←[:OCCURED_AT]-(c),
      (y:Year {Year: '2009'})←[:OCCURED_IN]-(c)
WITH z, m, COUNT(c) AS crimes
RETURN z.Police_Area AS Zone, m.Month AS Month, MAX(crimes) AS Highest_Crime_Rate
ORDER BY Highest_Crime_Rate DESC
LIMIT 1
```

Zone	Month	Highest_Crime_Rate
"Zone5"	"10"	26

Started streaming 1 records after 1 ms and completed after 9 ms.

Figure 24. GraphDB for Query5.2

Analysis of results:

According to the results of the enquiry, we can see that in October 2009, Zone5 had the highest number of crimes, with a total of 26 crimes. This query helps us identify which zone has the highest crime rates during a given period.

- **Find the zones that have are adjacent and sharing the same high crime months.**

Purpose: To find areas that share the same high crime months as their neighbors.

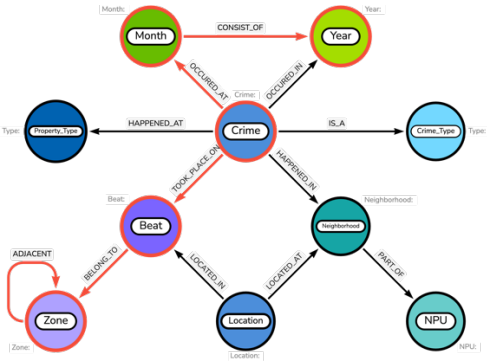
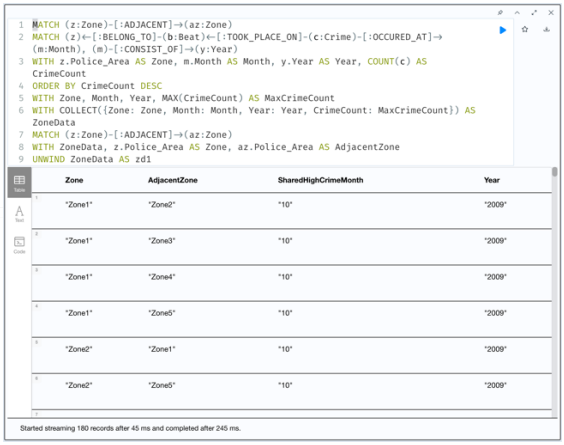


Figure 25. GraphDB for Query6



```
MATCH (z:Zone)-[:ADJACENT]→(az:Zone)
MATCH (z)←[:BELONG_TO]-(b:Beat)←[:TOOK_PLACE_ON]-(c:Crime)-[:OCCURED_AT]→
(m:Month), (m)-[:CONSIST_OF]→(y:Year)
WITH z.Police_Area AS Zone, m.Month AS Month, y.Year AS Year, COUNT(c) AS
CrimeCount
ORDER BY CrimeCount DESC
WITH Zone, Month, Year, MAX(CrimeCount) AS MaxCrimeCount
WITH COLLECT({Zone: Zone, Month: Month, Year: Year, CrimeCount: MaxCrimeCount}) AS
ZoneData
MATCH (z:Zone)-[:ADJACENT]→(az:Zone)
WITH ZoneData, z.Police_Area AS Zone, az.Police_Area AS AdjacentZone
UNWIND ZoneData AS zd1
```

	Zone	AdjacentZone	SharedHighCrimeMonth	Year
1	"Zone1"	"Zone2"	"10"	"2009"
2	"Zone1"	"Zone3"	"10"	"2009"
3	"Zone1"	"Zone4"	"10"	"2009"
4	"Zone1"	"Zone5"	"10"	"2009"
5	"Zone2"	"Zone1"	"10"	"2009"
6	"Zone2"	"Zone5"	"10"	"2009"

Started streaming 180 records after 45 ms and completed after 245 ms.

Figure 26. GraphDB for Query6

Query process:

Finds zone that share the highest crime rate in the same month and year. It first finds all zones and matches the number of crimes for each zone in each month and year. Then, the number of crimes in each zone, each month and year is sorted and only the maximum number of crimes in each zone, each month and year is retained. Next, this information is collected into the ZoneData list. Then, all neighboring zones are identified again, and the ZoneData list is expanded and compared to find the neighboring zones with the highest number of crimes in the same month and year. Finally, these zones, shared high crime months and years are returned and sorted by year, zone, adjacent zones and shared high crime months. (We added the year because we have many years of data in our dataset, and if we only showed the month, we wouldn't know for sure if the data for that month was correct)

Analysis of results:

According to the query results, we select the first few lines.

We can see that in 2009, there are several adjacent zones with the same crime peak months:

Zone1 and Zone2, Zone3, Zone4, Zone5 – October（10）

Zone2 and Zone1, Zone5 - October （10）

2. Extra Queries

- **Query 1: The total number of crimes each NPU**

Purpose of query:

Query the total number of crimes per Neighborhood Planning Unit (NPU).

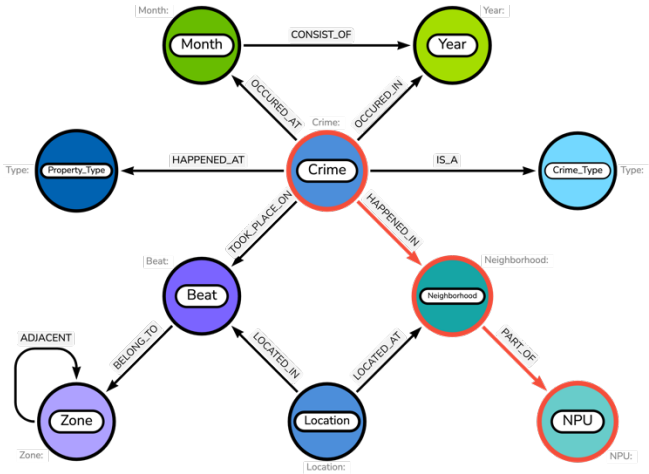
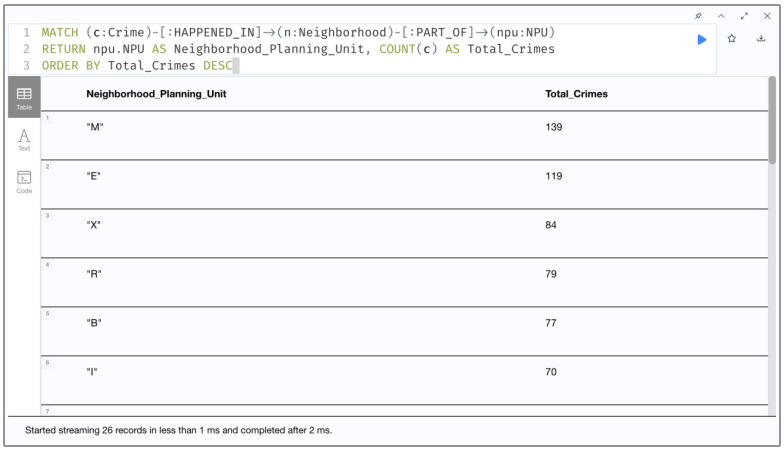


Figure 27. GraphDB for Extra Query1

Query process:

Match the relationship between criminal record, location, and NPU.
The number of crimes is calculated by NPU.
Returns the NPU and the corresponding number of crimes, in descending order by the number of crimes.



```
MATCH (c:Crime)-[:HAPPENED_IN]→(n:Neighborhood)-[:PART_OF]→(npu:NPU)
RETURN npu.NPU AS Neighborhood_Planning_Unit, COUNT(c) AS Total_Crimes
ORDER BY Total_Crimes DESC
```

	Neighborhood_Planning_Unit	Total_Crimes
1	"M"	139
2	"E"	119
3	"X"	84
4	"R"	79
5	"B"	77
6	"I"	70
7		

Started streaming 26 records in less than 1 ms and completed after 2 ms.

Figure 28. GraphDB for Extra Query1

Analysis of results:

We need to find out the total number of crimes per NPU (Neighborhood Planning Unit). Each NPU and its corresponding total number of crimes are shown. As can be seen from the table, NPU M had the highest number of crimes, which reached 139. This may mean that there is a high crime rate in the area and some measures may need to be taken to improve the security situation.
NPU Q had the lowest number of crimes at 1, indicating that the area is relatively well policed.

- **Query 2: Based on Query 1, find the NPU with the highest number of crimes and list the top five neighborhoods with the highest number of crimes in that NPU**

Purpose: Identify the NPU with the highest number of crimes and list the top 5 neighborhoods with the highest number of crimes.

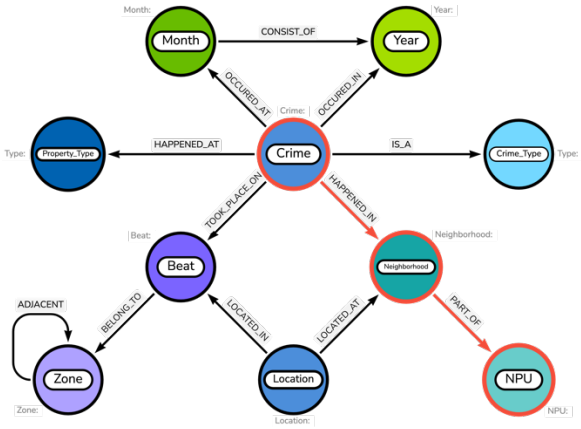


Figure 29. GraphDB for Extra Query2

Query process:

Match the relationship between criminal record, location, and NPU.
The number of crimes is calculated by NPU. The NPU with the highest number of crimes is taken out in descending order by the number of crimes.
Match the relationship between the location under the NPU and the criminal record. Group by location (neighborhood) to calculate the number of crimes.
Returns the NPU, the community's name, and the corresponding number of crimes, in descending order by the number of crimes, taking the first 5 results.

```
MATCH (npu:NPU {NPU: 'M'})<-[:PART_OF]-(n:Neighborhood)<-[:HAPPENED_IN]-(c:Crime)
WITH n, COUNT(c) AS totalCrimes
ORDER BY totalCrimes DESC
LIMIT 5
RETURN n.Neighborhood AS Neighborhood, totalCrimes AS TotalCrimes
```

	Neighborhood	TotalCrimes
1	"Downtown"	82
2	"Old Fourth Ward"	37
3	"Castleberry Hill"	10
4	"Sweet Auburn"	10

Started streaming 4 records in less than 1 ms and completed after 1 ms.

Figure 30. GraphDB for Extra Query2

Analysis of results:

The resulting table shows the NPU (M) with the highest number of crimes and its top five neighborhoods with the highest number of crimes. As you can see from the table, the Downtown neighborhood had the highest number of crimes, 82. This may mean that there is less security in this community, and something may need to be done to improve security. Other neighborhoods had relatively low numbers, such as "Old Fourth Ward" with 37 crimes, "Sweet Auburn" and "Castleberry Hill" with 10 crimes each.

- **Query 3: Find the average number of crimes per NPU, sorted by crime type**

Purpose: Look up the number of crimes under different crime types per NPU and the average number of crimes per neighborhood.

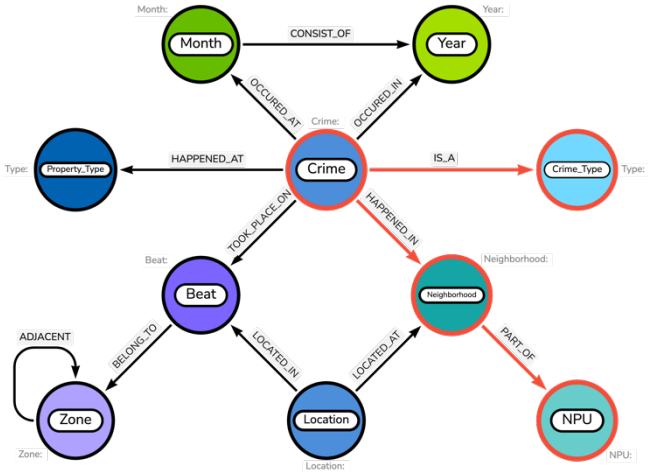
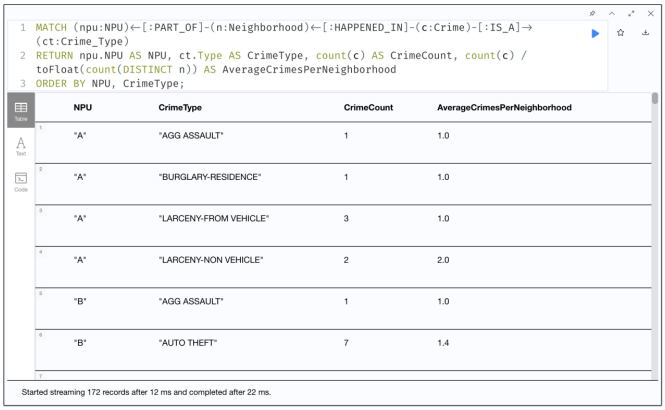


Figure 31. GraphDB for Extra Query3

Query process:

First finds the neighbors belonging to a specific NPU by the: PART_OF relation, then finds the crimes that occurred in those neighbors by the: HAPPENED_IN relation, and then finds the types of those crimes by the :IS_A relation. It then returns the number of crimes for each NPU and crime type and the average number of crimes in each neighborhood. The results are sorted by NPU and crime type.



```
MATCH (npu:NPU)←[:PART_OF]-(n:Neighborhood)←[:HAPPENED_IN]-(c:Crime)-[:IS_A]→
(ct:Crime_Type)
RETURN npu.NPU AS NPU, ct.Type AS CrimeType, count(c) AS CrimeCount, count(c) /
toFloat(count(DISTINCT n)) AS AverageCrimesPerNeighborhood
ORDER BY NPU, CrimeType;
```

	NPU	CrimeType	CrimeCount	AverageCrimesPerNeighborhood
1	"A"	"AGG ASSAULT"	1	1.0
2	"A"	"BURGLARY-RESIDENCE"	1	1.0
3	"A"	"LARCENY-FROM VEHICLE"	3	1.0
4	"A"	"LARCENY-NON VEHICLE"	2	2.0
5	"B"	"AGG ASSAULT"	1	1.0
6	"B"	"AUTO THEFT"	7	1.4
7				

Started streaming 172 records after 12 ms and completed after 22 ms.

Figure 32. GraphDB for Extra Query3

Analysis of results:

In NPU A, the average number of crimes for "AGG ASSAULT" and "BURGLARY RESIDENCE" was 1.0, indicating that these types of crimes occur relatively infrequently in different neighborhoods within the NPU. The average number of crimes of LARCENY-FROM VEHICLE" was 1.0, meaning that this type of crime also occurred less frequently in the communities of that NPU. The average number of crimes committed by "LARCENY-NON VEHICLE" was 2.0, indicating A relatively high frequency of this type of crime in the various communities of the NPU A.

- **Query 4: All crime types that occurred in 2012 and their corresponding crimes**

Purpose:

The purpose of this query is to find all crime types and their corresponding crimes that occurred in 2012.

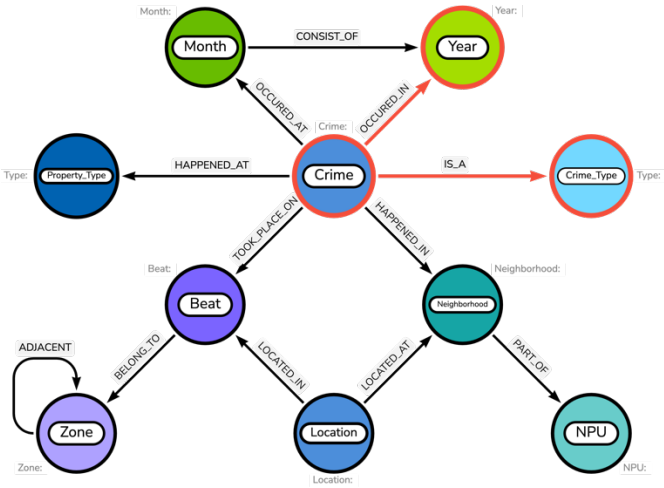


Figure 33. GraphDB for Extra Query4

Query process:

A node named "Crime", which is connected to the "CrimeType" node through the "IS_A" relationship. Similarly, the "Crime" node relates to the node named "Time" with the "OCCURRED_AT" relationship. The WHERE clause is used to filter query results. In this query, we are only interested in the case where the value of the "Year" attribute of the "Year" node is "2012".

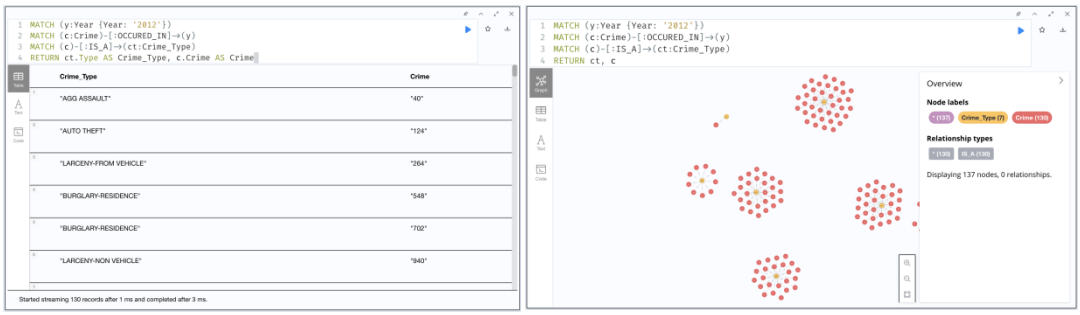


Figure 34. Result and GraphDB for Extra Query4

Analysis of results:

The query results in a table with two columns: Crime for all crimes committed in 2012 and the corresponding crime type.

The results can be used to analyze crime in 2012, for example, which types of crime were most common or what types of crime changed at different times of the year. Such analyses can help law enforcement better understand crime patterns to allocate resources and develop strategies more effectively.

3 Discussion

3.1 The capabilities of graph databases in comparison with their relational counterparts

Considering the distinct capabilities of graph databases and their relational counterparts, we've prepared a comparative analysis based on our Atlanta crime project context to highlight the unique strengths and potential challenges of each approach：

- **Modeling Complex Relationships**:
  Graph databases are efficient at simulating intricate relationships among entities. They can describe connections between nodes as edges, making data modelling flexible and expressive. In contrast, relational databases rely on tables with rows and columns, which can become unwieldy and difficult to manage when relationships are complex.

  This project involves numerous entities (e.g., Crime, Beat, NPU, Zone, PropertyType, etc.) and relationships between them. Analyzing crime patterns, such as identifying common crime types in certain neighborhoods or understanding crime trends over months and years, is much more straightforward in a graph database. In a relational database, this would require complex JOIN operations, which could be time-consuming and challenging to maintain.

- **Flexibility and Scalability**:
  In a graph database, it's relatively straightforward to introduce new types of relationships or nodes without disturbing existing structures. This is more challenging in a relational database, where schema changes can require substantial refactoring and can potentially disrupt existing data.

  With our crime graph database, it's relatively simple to introduce new nodes or relationships as the understanding of crime patterns evolves. For instance, we could add a new 'Weapon' node if we obtained data about weapons used in crimes. In a relational database, this could require substantial changes to the table structure, potentially impacting existing queries and analyses.

- **Use Cases:**
  Graph Databases are well-suited for use cases where data is highly interconnected and the relationship between the data points is equally or more important than the data points themselves. Relational Databases are traditionally used where the data's structure is known in advance, and it is more tabular than interconnected.

- **Query Language and Performance:**
  Graph databases like Neo4j use languages, Cypher, that make expressing and querying relationships straightforward. Also, the performance of graph databases stays relatively constant, even as the data grows, especially for querying connected data. Relational Databases always use SQL, the query language for relational databases, can become complex when dealing with multi-level relationships. The performance can decrease as data size and complexity of joins increase.

In conclusion, while both types of databases have their strengths and weaknesses, the choice between the two really comes down to the specifics of the problem that we are trying to solve and the nature of the data. The complexity and interconnectedness of the crime dataset makes it well-suited to a graph database. The graph model provides a more intuitive and flexible approach to understanding crime patterns, allowing us to easily add or modify nodes and relationships as the needs evolve. While a relational database could certainly store and analyze this data, it may not do so as efficiently or flexibly, especially when dealing with complex queries and real-time updates.

3.2 How Graph Data Science can be applied in at least one useful application of this graph database?

Graph Data Science is a powerful way to analyze and derive insights from complex systems and relationships that can be naturally modeled as graphs. Using the Atlanta crime dataset there are several ways Graph Data Science could be applied to extract valuable information:

- **Pattern Detection**
  By studying the relationships between different nodes (for instance, crimes, neighborhoods, and time), we can employ graph algorithms, such as community detection, to reveal underlying patterns in the data. These algorithms may identify clusters in which certain types of crimes frequently co-occur, or where certain neighborhoods repeatedly experience similar crime patterns. These patterns can assist law enforcement agencies in better understanding the criminal dynamics within their jurisdiction, thereby enabling them to devise more effective strategies to counteract crime.

  For example, in the context of problem 2, if two or more neighborhoods share a high number of the same types of crimes, it might suggest that they are confronted with similar law enforcement issues and might necessitate the implementation of similar preventative measures and law enforcement strategies.

- **Spatial analysis:**
  If geolocation data is included in the graph database, spatial analysis can be performed to understand the geographical distribution of crimes. This could be useful for urban planning, such as determining where to place new police stations, cameras, or street lighting to reduce crime.

  For example, in the context of additional query 1, we need to calculate the total number of crimes for each Neighborhood Planning Unit (NPU). The results might indicate that the NPU coded as 'M' has the highest crime rate. Therefore, it would be beneficial to implement new police stations, increase camera surveillance, or enhance street lighting in neighborhoods that fall under this NPU to mitigate criminal activity.

References:

[1] https://data.bloomberglp.com/company/sites/2/2016/09/paper_27.pdf
[2] https://en.wikipedia.org/wiki/Beat_(police)