

Figure 1: Graph from Arrow tool

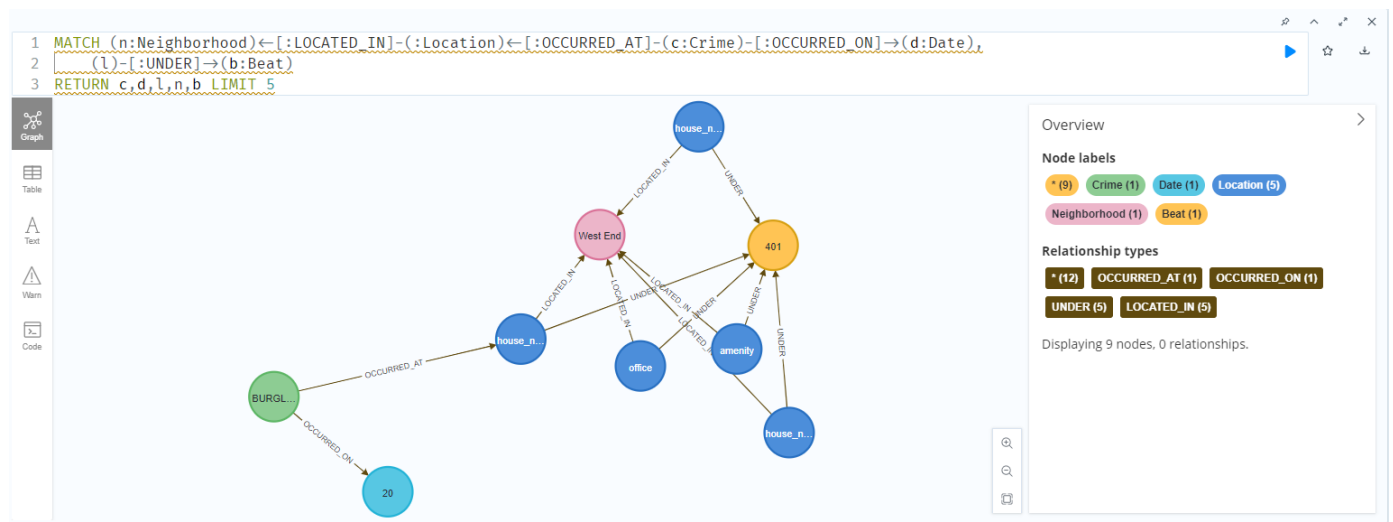


Figure 2: Graph from Neo4j

### Graph database creation:

// Load Crime Nodes

```
LOAD CSV WITH HEADERS FROM 'file:///crime_in.csv' AS row
CREATE (c:Crime {crimeID: toInteger(row.number),
               crimeType: row.crime,
               locationID:toInteger(row.location_id),
               dateID:toInteger(row.date_id),
               typeId:toInteger(row.type_id)
               })
```

```
1 // Load Crime Nodes
2 LOAD CSV WITH HEADERS FROM 'file:///crime_in.csv' AS row
3 CREATE (c:Crime {crimeID: toInteger(row.number),
4               crimeType: row.crime,
5               locationID:toInteger(row.location_id),
6               dateID:toInteger(row.date_id),
7               typeId:toInteger(row.type_id)
8               })
9
```

Added 1006 labels, created 1006 nodes, set 5030 properties, completed after 9 ms.

Table

Code

// Load Date Nodes

```
LOAD CSV WITH HEADERS FROM 'file:///date.csv' AS row
CREATE (d:Date {dateID: toInteger(row.date_id),
               day:toInteger(row.day),
               month:toInteger(row.month),
               year:toInteger(row.year)
               })
```

```
1 // Load Date Nodes
2 LOAD CSV WITH HEADERS FROM 'file:///date.csv' AS row
3 CREATE (d:Date {dateID: toInteger(row.date_id),
4               day:toInteger(row.day),
5               month:toInteger(row.month),
6               year:toInteger(row.year)
7               })
8
```

Added 16 labels, created 16 nodes, set 64 properties, completed after 3 ms.

Table

Code

```
// Load Location Nodes
LOAD CSV WITH HEADERS FROM 'file:///location.csv' AS row
CREATE (l:Location {locationID: toInteger(row.location_id),
    propertyID: toInteger(row.type_id),
    neighborhoodID:toInteger(row.neighborhood_id),
    property: row.type,
    beat: toInteger(row.beat)
    })
```

```
1 // Load Location Nodes
2 LOAD CSV WITH HEADERS FROM 'file:///location.csv' AS row
3 CREATE (l:Location {locationID: toInteger(row.location_id),
4     propertyID: toInteger(row.type_id),
5     neighborhoodID:toInteger(row.neighborhood_id),
6     property: row.type,
7     beat: toInteger(row.beat)
8     })
9
```

Added 871 labels, created 871 nodes, set 4355 properties, completed after 9 ms.

Table

Code

```
// Load Beat Nodes
LOAD CSV WITH HEADERS FROM 'file:///beat.csv' AS row
CREATE (b:Beat {beat: toInteger(row.beat),
    zone:toInteger(row.ZONE)
    })
```

```
1 // Load Beat Nodes
2 LOAD CSV WITH HEADERS FROM 'file:///beat.csv' AS row
3 CREATE (b:Beat {beat: toInteger(row.beat),
4     zone:toInteger(row.ZONE)
5     })
6
```

Added 68 labels, created 68 nodes, set 136 properties, completed after 3 ms.

Table

Code

// Load Neighborhood nodes

```
LOAD CSV WITH HEADERS FROM 'file:///neighborhood.csv' AS row
CREATE (n:Neighborhood {neighborhoodID: toInteger(row.neighborhood_id),
    neighborhood:row.neighborhood
})
```

```
1 // Load Neighborhood nodes
2 LOAD CSV WITH HEADERS FROM 'file:///neighborhood.csv' AS row
3 CREATE (n:Neighborhood {neighborhoodID: toInteger(row.neighborhood_id),
4     neighborhood:row.neighborhood
5 })
```

Added 104 labels, created 104 nodes, set 208 properties, completed after 4 ms.

Table

Code

// Create relationships between Crime and Date

```
MATCH (c:Crime), (d:Date)
WHERE c.dateID = d.dateID
CREATE (c)-[:OCCURRED_ON]->(d)
```

```
1 // Create relationships between Crime and Date
2 MATCH (c:Crime), (d:Date)
3 WHERE c.dateID = d.dateID
4 CREATE (c)-[:OCCURRED_ON]->(d)
```

Created 1006 relationships, completed after 8 ms.

Table

Warn

Code

// Create relationships between Crime and Location

```
MATCH (c:Crime), (l:Location)
WHERE c.locationID = l.locationID
CREATE (c)-[:OCCURRED_AT]->(l)
```

```
1 // Create relationships between Crime and Location
2 MATCH (c:Crime), (l:Location)
3 WHERE c.locationID = l.locationID
4 CREATE (c)-[:OCCURRED_AT]->(l)
5
```

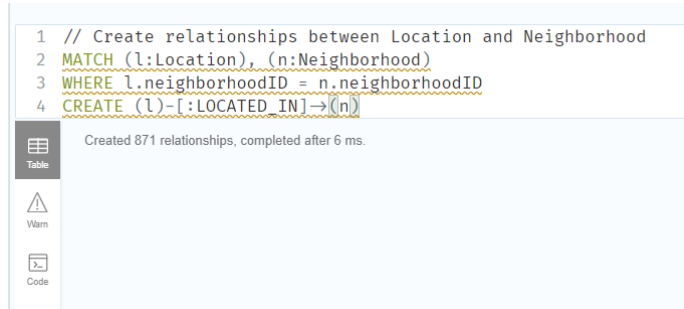
Created 1013 relationships, completed after 6 ms.

Table

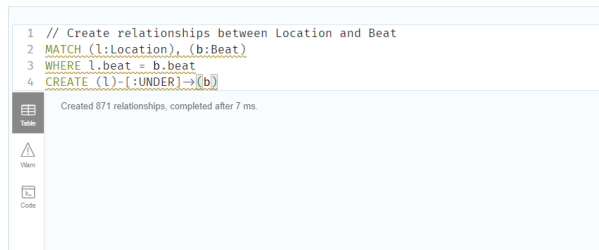
Warn

Code

```
// Create relationships between Location and Neighborhood
MATCH (l:Location), (n:Neighborhood)
WHERE l.neighborhoodID = n.neighborhoodID
CREATE (l)-[:LOCATED_IN]->(n)
```



```
// Create relationships between Location and Beat
MATCH (l:Location), (b:Beat)
WHERE l.beat = b.beat
CREATE (l)-[:UNDER]->(b)
```



### Queries:

// How many crimes are recorded for a given crime type in a specified neighbourhood for a particular period?

```
MATCH (c:Crime)-[:OCCURRED_AT]->(l:Location)-[:LOCATED_IN]->(n:Neighborhood),
      (c)-[:OCCURRED_ON]->(d:Date)
WHERE c.crimeType = 'LARCENY-NON VEHICLE'
      AND n.neighborhood='Downtown'
      AND d.day = 31
      AND d.month = 10
RETURN count(c) AS crimeCount
```

```
1 // How many crimes are recorded for a given crime type in a specified neighbourhood for a particular period?
2
3 MATCH (c:Crime)-[:OCCURRED_AT]->(l:Location)-[:LOCATED_IN]->(n:Neighborhood),
4     (c)-[:OCCURRED_ON]->(d:Date)
5 WHERE c.crimeType = 'LARCENY-NON VEHICLE'
6     AND n.neighborhood='Downtown'
7     AND d.day = 31
8     AND d.month = 10
9 RETURN count(c) AS crimeCount
```

	crimeCount
1	3

//Find the neighbourhoods that share the same crime types, organise in decending order of the number of common crime types.

```
MATCH (n1:Neighborhood)<-[:LOCATED_IN]-(:Location)<-[:OCCURRED_AT]-(c1:Crime)
WITH n1, COLLECT(DISTINCT c1.crimeType) AS crimeTypes
MATCH (n2:Neighborhood)<-[:LOCATED_IN]-(:Location)<-[:OCCURRED_AT]-(c2:Crime)
WHERE n2 <> n1
WITH n1, n2, COLLECT(DISTINCT c2.crimeType) AS commonCrimeTypes, crimeTypes
RETURN n1.neighborhood AS neighborhood1, n2.neighborhood AS neighborhood2, SIZE([x IN common
CrimeTypes WHERE x IN crimeTypes]) AS numCommonCrimeTypes
ORDER BY numCommonCrimeTypes DESC
```

1	//Find the neighbourhoods that share the same crime types, organise in decending order of the number of common crime types.		
2			
3	<b>MATCH</b> (n1:Neighborhood)<-[:LOCATED_IN]-(:Location)<-[:OCCURRED_AT]-(c1:Crime)		
4	<b>WITH</b> n1, <b>COLLECT</b> ( <b>DISTINCT</b> c1.crimeType) <b>AS</b> crimeTypes		
5	<b>MATCH</b> (n2:Neighborhood)<-[:LOCATED_IN]-(:Location)<-[:OCCURRED_AT]-(c2:Crime)		
6	<b>WHERE</b> n2 <> n1		
7	<b>WITH</b> n1, n2, <b>COLLECT</b> ( <b>DISTINCT</b> c2.crimeType) <b>AS</b> commonCrimeTypes, crimeTypes		
8	<b>RETURN</b> n1.neighborhood <b>AS</b> neighborhood1, n2.neighborhood <b>AS</b> neighborhood2, <b>SIZE</b> ([x <b>IN</b> commonCrimeTypes <b>WHERE</b> x <b>IN</b> crimeTypes]) <b>AS</b> numCommonCrimeTypes		
9	<b>ORDER BY</b> numCommonCrimeTypes <b>DESC</b>		

	neighborhood1	neighborhood2	numCommonCrimeTypes
1	"West End"	"Old Fourth Ward"	7
2	"West End"	"Inman Park"	7
3	"West End"	"Center Hill"	7
4	"West End"	"Edgewood"	7
5	"West End"	"Grove Park"	7
6	"West End"	"Pittsburgh"	7
7			

Started streaming 10712 records in less than 1 ms and completed in less than 1 ms, displaying first 1000 rows.

// Return the top 5 neighbourhoods for a specified crime for a specified duration.

```
MATCH (n:Neighborhood)-[:LOCATED_IN]-(:Location)-[:OCCURRED_AT]-(c:Crime)-
[:OCCURRED_ON]->(d:Date)
WHERE c.crimeType = 'AGG ASSAULT'
      AND d.day >=10
      AND d.day <=28
RETURN n.neighborhood AS neighborhood, COUNT(c) AS numCrimes
ORDER BY numCrimes DESC
LIMIT 5
```



The screenshot shows a database query interface. At the top, a SQL query is entered in a text area. Below the query, a table displays the results. The table has two columns: 'neighborhood' and 'numCrimes'. The results are ordered by 'numCrimes' in descending order, with 'Downtown' having the highest count of 5, followed by 'Pittsburgh' with 5, 'English Avenue' with 3, 'Midtown' with 3, and 'West End' with 3. The interface includes a 'Table' view icon on the left and a status bar at the bottom indicating that 5 records were streamed after 1 ms and completed after 4 ms.

	neighborhood	numCrimes
1	"Downtown"	5
2	"Pittsburgh"	5
3	"English Avenue"	3
4	"Midtown"	3
5	"West End"	3

Started streaming 5 records after 1 ms and completed after 4 ms.



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

**MATCH** (l:Location)←[:OCCURRED\_AT]-(c:Crime)

**RETURN** l.property **AS** propertyType, **COLLECT**(**DISTINCT** c.crimeType) **AS** crimeTypes

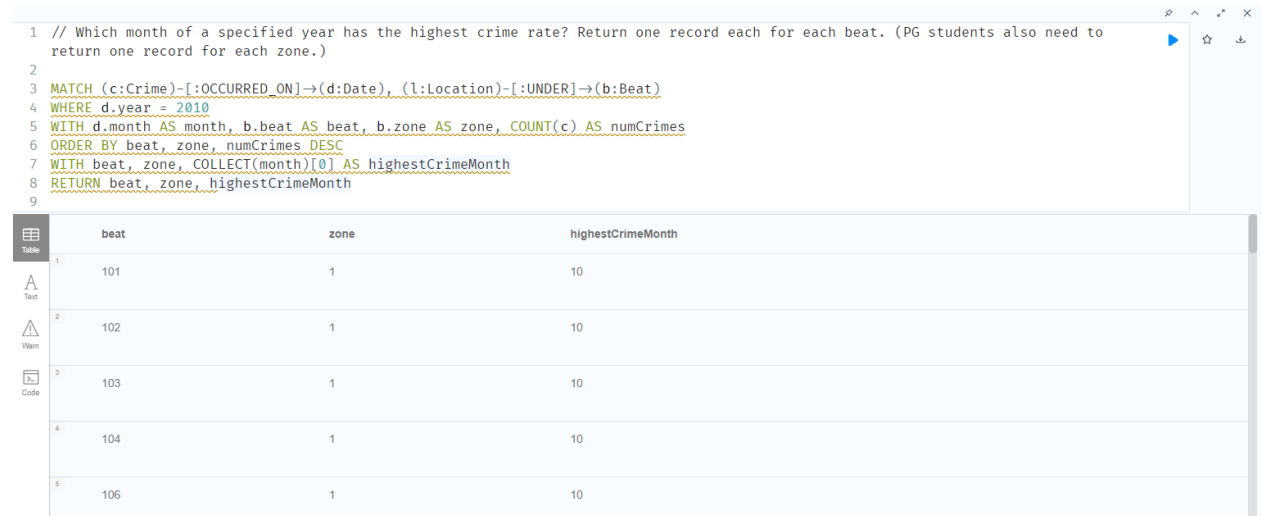


```
1 // Find the types of crimes for each property type.
2
3 MATCH (l:Location)←[:OCCURRED_AT]-(c:Crime)
4 RETURN l.property AS propertyType, COLLECT(DISTINCT c.crimeType) AS crimeTypes
5
```

	propertyType	crimeTypes
1	"office"	["AUTO THEFT", "LARCENY-FROM VEHICLE", "LARCENY-NON VEHICLE", "BURGLARY-NONRES"]
2	"shop"	["LARCENY-FROM VEHICLE", "LARCENY-NON VEHICLE", "AUTO THEFT", "ROBBERY-COMMERCIAL", "BURGLARY-RESIDENCE", "AGG ASSAULT"]
3	"house_number"	["LARCENY-NON VEHICLE", "ROBBERY-PEDESTRIAN", "AGG ASSAULT", "LARCENY-FROM VEHICLE", "AUTO THEFT", "BURGLARY-RESIDENCE", "BURGLARY-NONRES", "HOMICIDE", "ROBBERY"]
4	"building"	["LARCENY-FROM VEHICLE", "LARCENY-NON VEHICLE", "AUTO THEFT", "ROBBERY-PEDESTRIAN", "BURGLARY-RESIDENCE", "BURGLARY-NONRES"]
5	"amenity"	["AGG ASSAULT", "LARCENY-FROM VEHICLE", "AUTO THEFT", "LARCENY-NON VEHICLE", "BURGLARY-RESIDENCE", "ROBBERY-COMMERCIAL", "ROBBERY-PEDESTRIAN", "BURGLARY-NONRES"]
6	"tourism"	["RAPE", "LARCENY-FROM VEHICLE", "LARCENY-NON VEHICLE", "AGG ASSAULT", "ROBBERY-PEDESTRIAN"]

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

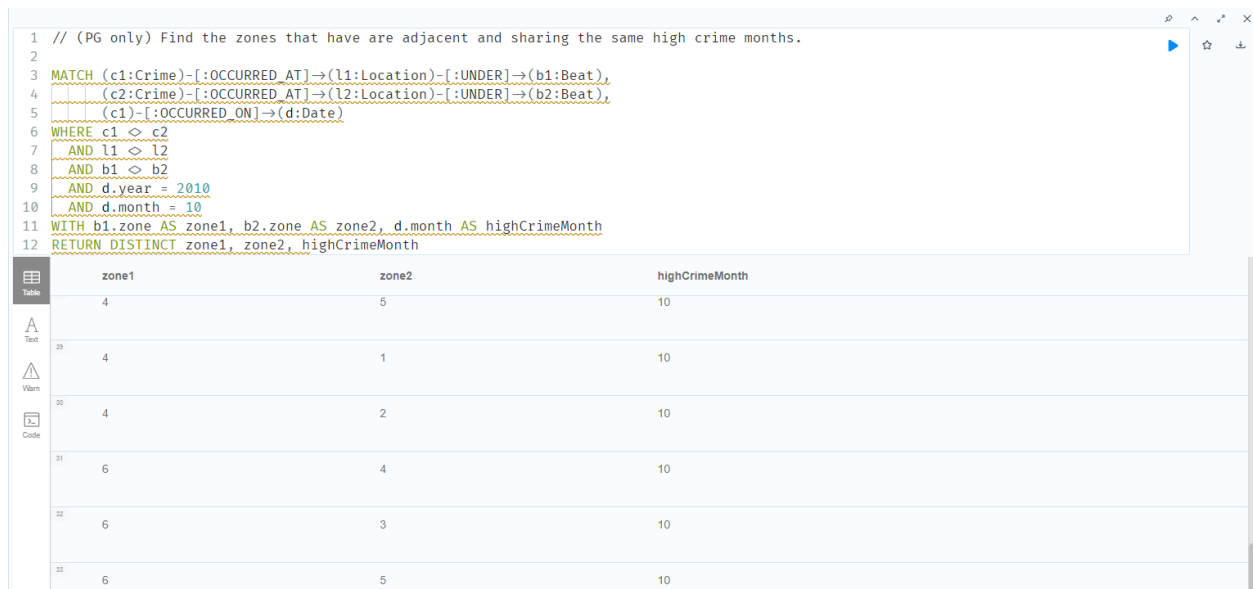
```
MATCH (c:Crime)-[:OCCURRED_ON]->(d:Date), (l:Location)-[:UNDER]->(b:Beat)
WHERE d.year = 2010
WITH d.month AS month, b.beat AS beat, b.zone AS zone, COUNT(c) AS numCrimes
ORDER BY beat, zone, numCrimes DESC
WITH beat, zone, COLLECT(month)[0] AS highestCrimeMonth
RETURN beat, zone, highestCrimeMonth
```



The screenshot shows a database query interface. The top part displays a SQL query with line numbers 1 through 9. The query is designed to find the month with the highest crime rate for each beat and zone in the year 2010. The bottom part shows the results of the query in a table with three columns: beat, zone, and highestCrimeMonth. The table contains five rows of data, all showing zone 1 and the month 10.

	beat	zone	highestCrimeMonth
1	101	1	10
2	102	1	10
3	103	1	10
4	104	1	10
5	106	1	10

```
// Find the zones that have are adjacent and sharing the same high crime months.  
MATCH (c1:Crime)-[:OCCURRED_AT]->(l1:Location)-[:UNDER]->(b1:Beat),  
      (c2:Crime)-[:OCCURRED_AT]->(l2:Location)-[:UNDER]->(b2:Beat),  
      (c1)-[:OCCURRED_ON]->(d:Date)  
WHERE c1 <> c2  
      AND l1 <> l2  
      AND b1 <> b2  
      AND d.year = 2010  
      AND d.month = 10  
WITH b1.zone AS zone1, b2.zone AS zone2, d.month AS highCrimeMonth  
RETURN DISTINCT zone1, zone2, highCrimeMonth
```



The screenshot shows a database query editor with a Cypher query and its results. The query is designed to find adjacent zones sharing the same high crime month in 2010. The results table displays three columns: zone1, zone2, and highCrimeMonth. The data shows six pairs of adjacent zones (4,5), (4,1), (4,2), (6,4), (6,3), and (6,5) all sharing the high crime month of 10.

zone1	zone2	highCrimeMonth
4	5	10
4	1	10
4	2	10
6	4	10
6	3	10
6	5	10

// Write cypher code to find which property type has the lowest count of any type of crime

```
MATCH (l:Location)<-[:OCCURRED_AT]-(c:Crime)
WITH l.property AS propertyType, COUNT(c) AS numCrimes
RETURN propertyType
ORDER BY numCrimes ASC
LIMIT 1
```

```
1 // Write cypher code to find which property type has the lowest count of any type of crime
2
3 MATCH (l:Location)<-[:OCCURRED_AT]-(c:Crime)
4 WITH l.property AS propertyType, COUNT(c) AS numCrimes
5 RETURN propertyType
6 ORDER BY numCrimes ASC
7 LIMIT 1
8
```

propertyType	
1	"place"

// Write cypher code to find which neighbourhood has the lowest crime count of all crime type

```
MATCH (n:Neighborhood)<-[:LOCATED_IN]-(l:Location)<-[:OCCURRED_AT]-(c:Crime)
WITH n, COUNT(c) AS numCrimes
RETURN n.neighborhood AS neighborhood
ORDER BY numCrimes ASC
LIMIT 1
```

```
1 // Write cypher code to find which neighbourhood has the lowest crime count of all crime type
2
3 MATCH (n:Neighborhood)<-[:LOCATED_IN]-(l:Location)<-[:OCCURRED_AT]-(c:Crime)
4 WITH n, COUNT(c) AS numCrimes
5 RETURN n.neighborhood AS neighborhood
6 ORDER BY numCrimes ASC
7 LIMIT 1
```

neighborhood	
1	"Capitol View Manor"

The Atlanta dataset's crime data lends itself well to analysis using the graph database.

1. Complex linkages between crimes, suspects, victims, and locations can be handled effectively using graph databases.
2. Investigating relationships and locating crime trends, hotspots, or repeated behaviors are made easier with the help of graph database traversal and pattern matching.
3. Scalability and good performance are features of graph databases. This is crucial for handling big quantities of criminal data and running intricate searches across various related entities.
4. Because graph databases are schema less, it is simple to adapt them to new attributes and growing criminal data models.
5. Context-rich investigations in graph databases offer more information on gang affiliations, social ties, and networks between suspects and victims.
6. Logical reasoning and semantic reasoning assist reveal hidden relationships and foresee probable links to criminal activity.
7. A chart database combines information from various sources and offers a consolidated view of crime data from various domains, including case reports, arrests, and court records.
8. Multiple agencies or departments can provide and analyze criminal data together thanks to the chart database's collaboration and data sharing features.
9. Chart databases can be used to determine resource allocations, criminal tendencies, and preemptive law enforcement tactics.
10. The chart database makes use of the chart structure to visualize crime networks and enable geographical analysis to comprehend crime trends in Atlanta.

YouTube Video link:

<https://youtu.be/peEenVdcj7g>