

DATS 6102: Group Assignment

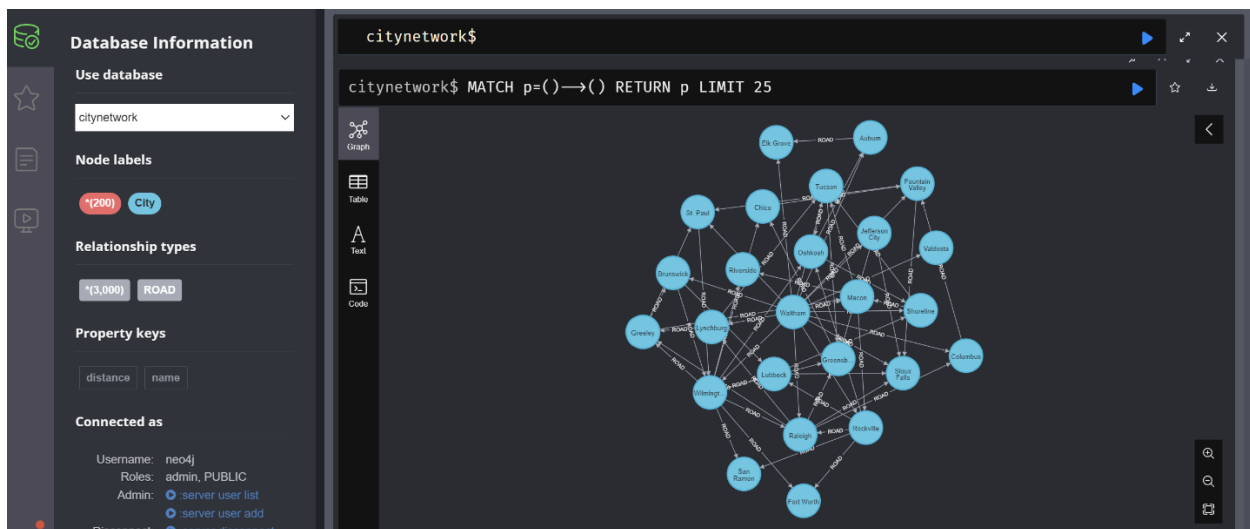
Group 3: Fyrooz Anika Khan, Fred Liu, Fardin Hafiz, Suraj Ravindra Kapare

Professor: Hazim Shatnawi

Session: Fall 2024

Results Documentation

The dataset 'road_network.csv' when uploaded in Neo4j Desktop looks like the following:



Dataset in Neo4j Desktop

The following section contains a brief description of the Cypher queries:

Explanation of Query 1

The task was to list all roads (connections) from or to a specific city ("Atlanta") and show the destination cities and distances. The provided data accomplishes this by:

The screenshot shows a Neo4j Cypher query in a dark-themed editor. The query is as follows:

```

citynetwork$
1 MATCH (c:City)-[r:ROAD]→(d:City)
2 WHERE c.name='Atlanta'
3 RETURN c.name AS From_City, d.name AS To_City, r.distance AS Distance
4 UNION
5 MATCH (d:City)-[r:ROAD]→(c:City)
6 WHERE c.name='Atlanta'
7 RETURN d.name AS From_City, c.name AS To_City, r.distance AS Distance

```

Below the query editor, the results are displayed in a table view. The table has three columns: From_City, To_City, and Distance. It contains five rows of data, all originating from Atlanta.

	From_City	To_City	Distance
1	"Atlanta"	"Eden Prairie"	188
2	"Atlanta"	"Hagerstown"	125
3	"Atlanta"	"Hoboken"	251
4	"Atlanta"	"North Charleston"	99
5	"Atlanta"	"Mission Viejo"	219

Query Output 1

- **Listing Roads from Atlanta:** The data shows all the cities directly connected to "Atlanta" through a road, indicating the direction as originating from "Atlanta" (From_City).
- **Showing Destinations and Distances:** For each connection originating from "Atlanta", the data lists the destination city (To_City) and the corresponding distance of the road (Distance).

In summary, this information can be used to understand the connectivity of "Atlanta" in the road network, showing how it is linked to other cities and the distance for each connection.

Explanation of Query 2

The task was to retrieve connections where the distance is greater than a specific value (e.g., 100 km), showing both cities and the distance. The provided data accomplishes this by:

<pre> 1 MATCH (c1:City)-[r:ROAD]→(c2:City) 2 WHERE r.distance > 100 3 RETURN c1.name AS City1, c2.name AS City2, r.distance AS Distance </pre>			
	City1	City2	Distance
1	"Murfreesboro"	"Mobile"	106
2	"Murfreesboro"	"Gilbert"	381
3	"Murfreesboro"	"Hagerstown"	467
4	"Murfreesboro"	"San Bernardino"	134
5	"Murfreesboro"	"Billings"	302
6	"Murfreesboro"	"San Rafael"	323

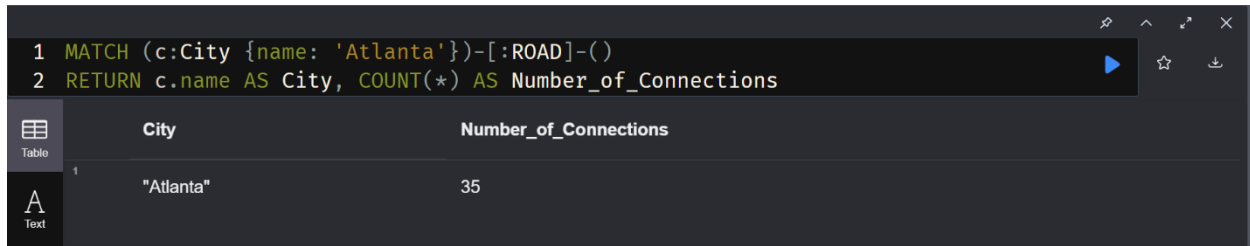
Query Output 2

- **Listing Connections with Distances Greater than 100 km:** The data shows road connections where the distance between the cities is greater than 100 km.
- **Showing Both Cities and the Distance:** For each qualifying connection, the data lists both the origin city (City1) and the destination city (City2) along with the corresponding distance of the road (Distance).

In summary, this information can be used to understand which city pairs have relatively longer road connections in the road network, focusing on distances that exceed the specified threshold.

Explanation of Query 3

The task was to count how many roads (connections) are connected to a specific city (e.g., "Atlanta"). The provided data accomplishes this by:



The screenshot shows a Cypher query editor with two lines of code:

```
1 MATCH (c:City {name: 'Atlanta'})-[:ROAD]-()
2 RETURN c.name AS City, COUNT(*) AS Number_of_Connections
```

Below the query, the output is displayed in a table format. The table has two columns: 'City' and 'Number_of_Connections'. There is one row of data showing 'Atlanta' with 35 connections.

	City	Number_of_Connections
1	"Atlanta"	35

Query Output 3

- **Identifying the Specific City:** The data focuses on "Atlanta," which is the city in question.
- **Counting Total Connections:** The data indicates that "Atlanta" has 35 connections. This count includes all roads that either start from or lead to "Atlanta," encompassing both incoming and outgoing connections.

In summary, this information is useful to understand the connectivity level of "Atlanta" within the road network, indicating how well-connected the city is in terms of the number of roads linking it to other cities.

Explanation of Query 4

The task was to find cities connected to a city (New York) with a distance below a specific value. This was accomplished by:

citynetwork\$

```

1 MATCH (c1:City)-[r:ROAD]→(c2:City)
2 WHERE c1.name = 'New York' AND r.distance < 200
3 RETURN c1.name AS From_City, c2.name AS Connected,
   r.distance AS Distance
4 UNION
5 MATCH (c2:City)-[r:ROAD]→(c1:City)
6 WHERE c1.name = 'New York' AND r.distance < 200
7 RETURN c2.name AS From_City, c1.name AS Connected,
   r.distance AS Distance
8

```

	From_City	Connected	Distance
1	"New York"	"Lynnwood"	138
2	"New York"	"Tacoma"	117
3	"New York"	"Carrollton"	63
4	"New York"	"Gilbert"	174
5	"New York"	"Lakeville"	87

Query Output 4

- **Match Cities by reference city and distance:** Matching all of the cities that are connected to New York and below a specified distance of 200 km.
- **Presenting the findings:** The output demonstrates New York as the “From_City” since this is the base city that is being analyzed for closeness. The “connected” shows the city connected to New York and under 200 km of distance. The “distance” column shows the distance between the cities.

In summary, this data was effective in showing the cities that are 200 km away from New York City, providing valuable insight on what cities are in a desired radius.

Explanation of Query 5

The task was to list all roads leading to a specific city by:

citynetwork\$

```

1 MATCH (c1:City)-[r:ROAD]→(c2:City)
2 WHERE c2.name = 'Detroit'
3 RETURN c1.name AS From_City, c2.name AS To_City,
  r.distance AS Distance
4
5

```

Rerun

	From_City	To_City	Distance
1	"North Miami"	"Detroit"	360
2	"Richardson"	"Detroit"	254
3	"Sioux Falls"	"Detroit"	51
4	"Victorville"	"Detroit"	335
5	"Arcadia"	"Detroit"	296

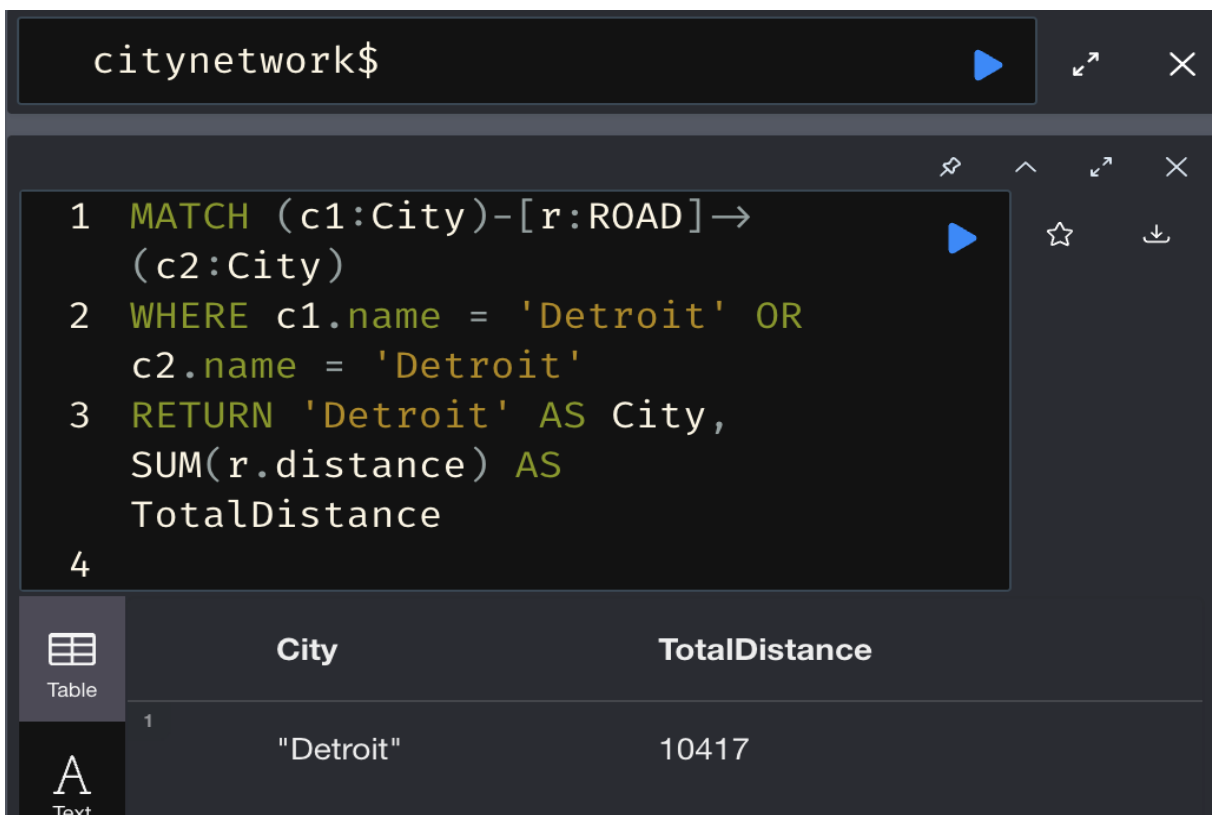
Query Output 5

- **Match cities that are connected to Detroit:** Matches all of the cities that are leading to Detroit, since these indicate all of the roads that are connected.
- **Presenting the findings:** The output has a similar format to the previous query as it shows all of the roads that are connected to Detroit as the “From_City”. The “To_City” is Detroit since the query asks for the roads leading to Detroit. The “Distance” demonstrates the distance of the road.

In summary, this data was effective in showing the roads that are connected to Detroit, providing valuable insight into what roads lead into the city of Detroit and what the lengths of the roads are.

Explanation of Query 6

The task was to calculate the total distance of all roads connected to a city (Detroit) by:



The screenshot shows a Cypher query interface with a prompt 'citynetwork\$' and a query editor containing the following code:

```
1 MATCH (c1:City)-[r:ROAD]→  
  (c2:City)  
2 WHERE c1.name = 'Detroit' OR  
  c2.name = 'Detroit'  
3 RETURN 'Detroit' AS City,  
  SUM(r.distance) AS  
  TotalDistance  
4
```

Below the query editor, there is a table view showing the results of the query. The table has two columns: 'City' and 'TotalDistance'. The first row shows 'Detroit' with a total distance of 10417.

	City	TotalDistance
1	"Detroit"	10417

Query Output 6

- **Match roads that go to and from Detroit:** Matches all of the cities that are connected to Detroit, since these indicate all of the roads that are connected. The sum of all of these roads is then taken with the sum function.
- **Presenting the findings:** The output shows the total distance of the roads that connect to the city of Detroit.

In summary, this data was effective in showing the sum of all the roads that are connected to Detroit, providing valuable insight into the length of roads that lead into the city of Detroit.

Explanation of Query 7

The task was to find the shortest road between two cities (Buffalo and Arizona) by:

citynetwork\$

1 MATCH (c1:City)-[r:ROAD]→

(c2:City)

2 WHERE c1.name = 'Phoenix' AND

c2.name = 'Buffalo'

3 RETURN c1.name AS From_City,

c2.name AS To_City, r.distance

AS Distance

4 ORDER BY r.distance ASC

5 Limit 1

☆

↓

Table

	From_City	To_City	Distance
1	"Phoenix"	"Buffalo"	230

Query Output 7

- **Match roads that go between the desired cities of Buffalo and Arizona:** Matches all of the roads that connect the cities of Buffalo and Arizona. Then the roads are placed into ascending order, so the shortest road is first. The limit is then set to 1 so only the shortest road is shown.
- **Presenting the findings:** The output shows the shortest road between Buffalo and Arizona. The output has a similar format to the previous queries as it shows the two cities and the distance between them.

In summary, this query was useful in correctly finding the shortest road that connects two cities (Buffalo and Arizona).

Explanation of Query 8

The task was to find all cities directly connected to both "Denver" and "Seattle" in the road network. The provided data accomplishes this by:



	ConnectedCity
1	"Waltham"
2	"Escondido"
3	"Citrus Heights"
4	"Hoffman Estates"
5	"Fort Worth"
6	"Chicago"

Query Output 8

- **Identifying Connections for Denver:** The query identifies all cities directly connected to "Denver" via roads. This includes all outgoing connections from "Denver" to other cities.
- **Identifying Connections for Seattle:** Similarly, it identifies all cities directly connected to "Seattle," focusing on outgoing connections from "Seattle" to other cities.
- **Finding Common Connections:** The query determines the intersection of the two sets of connected cities (from "Denver" and "Seattle"). Only cities that appear in both sets are returned, ensuring they are directly linked to both "Denver" and "Seattle."

In summary, this information highlights cities that serve as mutual connection points between "Denver" and "Seattle," providing insight into shared routes or hubs in the road network.

Explanation of Query 9

The task was to identify all cities in the road network that have more than three direct connections (incoming or outgoing) and display the count of these connections. The provided data accomplishes this by:

```

1 MATCH (city:City)-[r:DISTANCE]→()
2 WITH city, COUNT(r) AS connections
3 WHERE connections > 3
4 RETURN city.name AS City, connections;
5

```

	City	connections
1	"Waltham"	16
2	"Hanford"	11
3	"Escondido"	19
4	"Stockton"	19
5	"Raleigh"	16
6	"Hoboken"	12

Query Output 9

- **Counting Total Connections:** For each city in the network, the query counts the total number of connections (roads) associated with that city. This includes both outgoing and incoming connections to ensure a comprehensive measure of connectivity.
- **Filtering Cities with High Connectivity:** It filters out cities with fewer than or equal to three connections. Only cities with more than three connections are retained in the results.
- **Returning the Results:** The query returns the names of these highly connected cities along with the exact count of their direct connections.

In summary, this information is useful for identifying central cities in the network with significant connectivity, potentially serving as major hubs or key transit points.

Explanation of Query 10

The task was to calculate the total distance of all road connections in the network. The provided data accomplishes this by:

```

1 MATCH ()-[r:DISTANCE]->()
2 RETURN SUM(r.miles) AS TotalNetworkDistance;
3

```

TotalNetworkDistance
821847

Query Output 10

- **Summing Up Distances:** The query accesses all road connections (relationships) in the network. It retrieves the distance for each road and computes the sum of all distances.
- **Providing the Network-Wide Total:** The total sum represents the cumulative distance of all direct connections in the road network.

In summary, this information helps in understanding the overall size and extent of the road network. It can be used to assess the total travel capacity or infrastructure scope represented by the dataset.

Extra Credit Tasks:

Explanation of Query E: 1.a

The task was to find cities that have exactly two direct connections (either incoming or outgoing). The provided data accomplishes this by:

```

1 MATCH (p:Node)-[r]->()
2 WITH p.name AS Node, count(r) AS RelationshipCount
3 WHERE RelationshipCount = 2
4 RETURN Node AS city_with_two_connections;

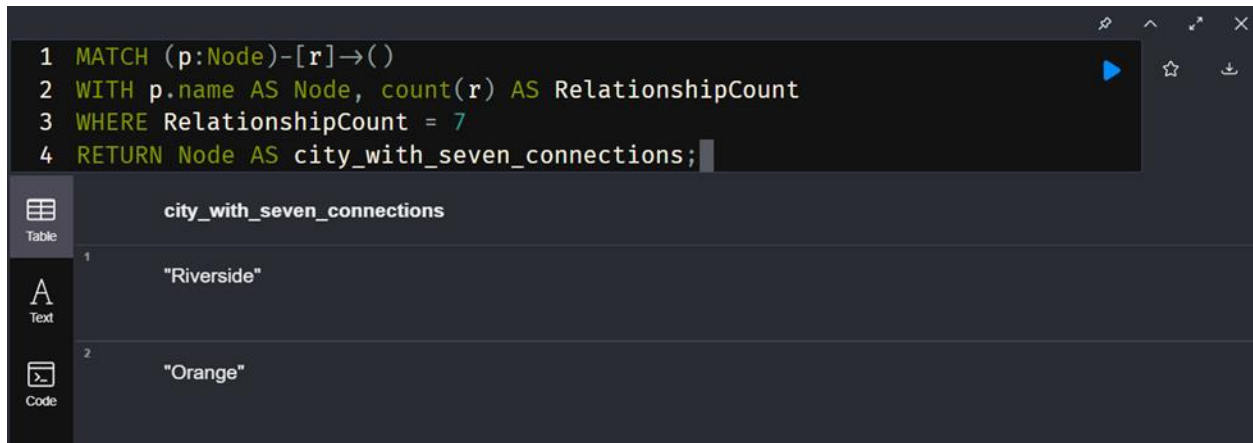
```

(no changes, no records)

Query Output E: 1.a

- **Counting Connections:** The code counts the number of connections (RelationshipCount) for each city. A connection is represented by an outgoing relationship [r] originating from a node (p:Node).
- **Filtering Cities with Two Connections:** The code then filters the cities to keep only those with exactly two direct connections by adding a WHERE clause (RelationshipCount = 2).
- **Returning City Names:** Finally, the names of the cities that meet the criteria are returned as city_with_two_connections.

In summary, this information can be used to understand which cities in the road network have limited connections of exactly two direct links to other cities. This can help in identifying cities that may be relatively less integrated within the network compared to others. The output “(no changes, no records)” shows that no city has only 2 connections. We can modify the code to see output that contains city names, for example, with 7 connections. The output screenshot is provided below:



```

1 MATCH (p:Node)-[r]→()
2 WITH p.name AS Node, count(r) AS RelationshipCount
3 WHERE RelationshipCount = 7
4 RETURN Node AS city_with_seven_connections;

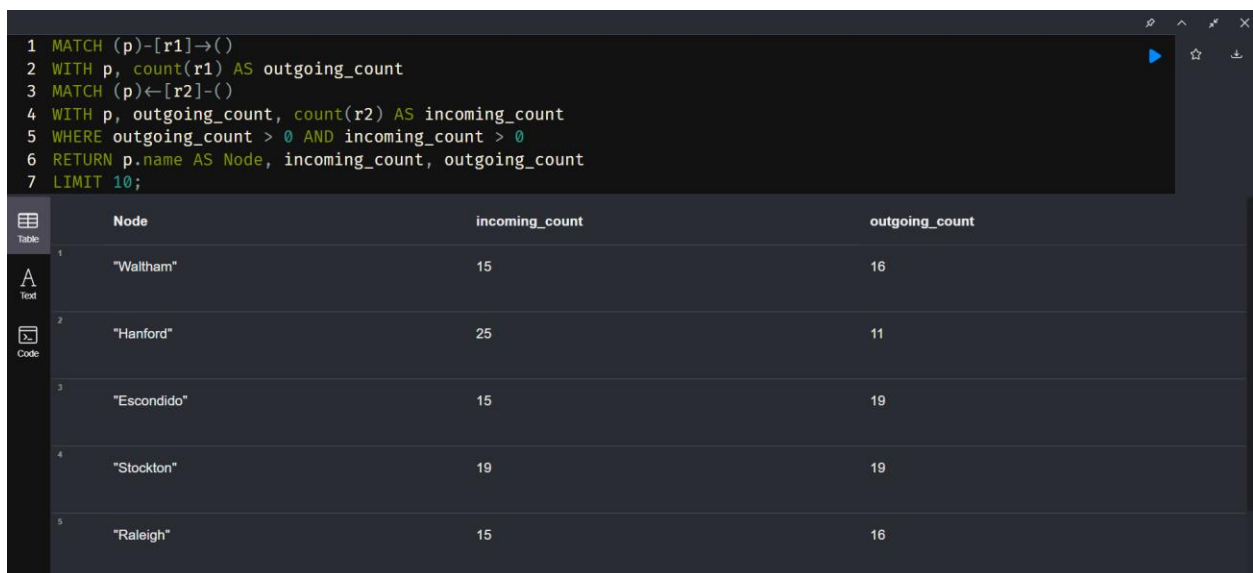
```

	city_with_seven_connections
1	"Riverside"
2	"Orange"

Query Output E: 1.b

Explanation of Query E: 2

The task was to identify cities that have both incoming and outgoing connections and list the total count of each type. The provided code accomplishes this by:



```

1 MATCH (p)-[r1]→()
2 WITH p, count(r1) AS outgoing_count
3 MATCH (p)←[r2]-()
4 WITH p, outgoing_count, count(r2) AS incoming_count
5 WHERE outgoing_count > 0 AND incoming_count > 0
6 RETURN p.name AS Node, incoming_count, outgoing_count
7 LIMIT 10;

```

	Node	incoming_count	outgoing_count
1	"Waltham"	15	16
2	"Hanford"	25	11
3	"Escondido"	15	19
4	"Stockton"	19	19
5	"Raleigh"	15	16

Query Output E: 2

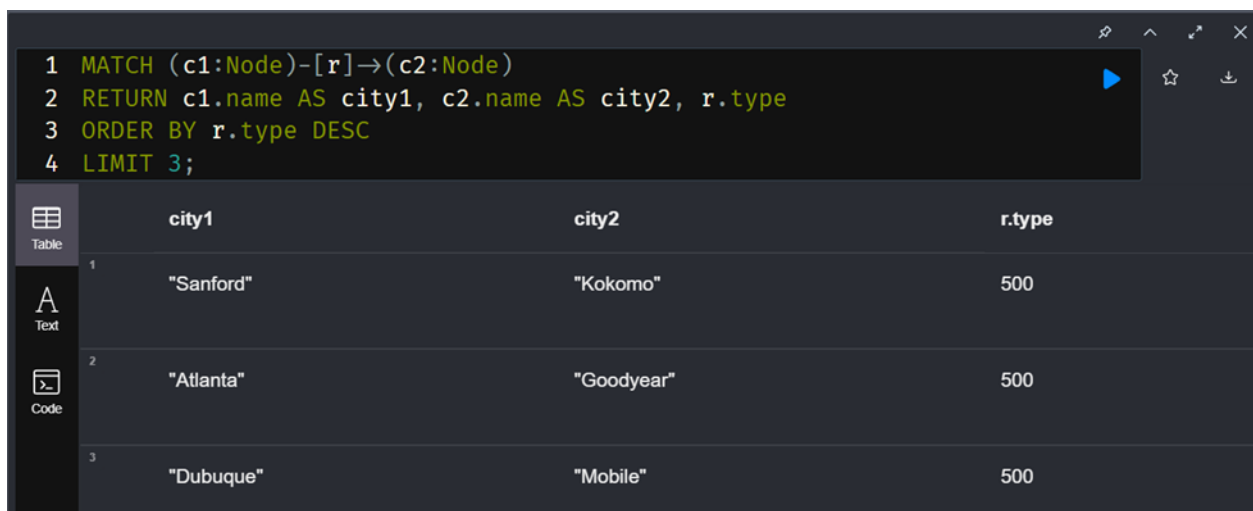
The task was to identify cities that have both incoming and outgoing connections and list the total count of each type. The provided code accomplishes this by:

- **Counting Outgoing Connections:** The code counts all outgoing relationships (r1) from each city. This count is labeled as `outgoing_count` and represents how many roads lead out of the given city.
- **Counting Incoming Connections:** Lines 3 and 4 count all incoming relationships (r2) to each city. This count is labeled as `incoming_count` and represents how many roads are incoming to the given city.
- **Filtering Cities with Both Types of Connections:** The WHERE clause filters the results to only include cities that have at least one outgoing and one incoming connection (`outgoing_count > 0 AND incoming_count > 0`).
- **Returning Results:** Finally, the query returns the city names (Node), the count of incoming connections (`incoming_count`), and the count of outgoing connections (`outgoing_count`).

In summary, this information is useful for identifying cities that have incoming and outgoing connections within the road network. It helps in understanding which cities are central to the network. The output shows how well-connected cities like "Waltham" and "Hanford" are by listing them along with the corresponding numbers of their incoming and outgoing links.

Explanation of Query E: 3

The task was to identify the top three pairs of cities that have the longest direct distance between them. The provided code accomplishes this by:



```
1 MATCH (c1:Node)-[r]→(c2:Node)
2 RETURN c1.name AS city1, c2.name AS city2, r.type
3 ORDER BY r.type DESC
4 LIMIT 3;
```

	city1	city2	r.type
1	"Sanford"	"Kokomo"	500
2	"Atlanta"	"Goodyear"	500
3	"Dubuque"	"Mobile"	500

Query Output E: 3

- **Matching All Pairs of Cities:** The query starts by matching all nodes (c1:Node and c2:Node) that are connected by a relationship ([r]).
- **Selecting City Names and Distances:** Line 2 returns the names of the two cities involved in each connection along with the distance (r.type).
- **Sorting by Distance:** Line 3 sorts the results in descending order based on the distance.
- **Limiting the Results:** Line 4 returns only the top three pairs of cities with the longest direct distances.

In summary, this information can be used to identify key routes within the road network that have the greatest distance between cities, helping to highlight potentially significant or challenging routes.

Analysis and Observations

Neo4j provided fast responses for complex queries like finding connections, counting roads, or filtering distance. The traversal features of Neo4j allows for easy querying of nodes for various types of relationships. Cypher queries used by Neo4j are also very easy to understand.

- **Performance:** When we need to find connections or paths between cities, Neo4j could quickly traverse the network of cities and roads, which would be slower in a traditional database. A large amount of data was handled, and the output was provided very quickly. The data was also presented in a very understandable fashion. For example, task 12 demonstrated the graph model's strength in distinguishing between incoming and outgoing connections.
- **Ease of Querying:** The syntax of cypher queries is very intuitive. This allowed each of the tasks to be performed effectively and efficiently since it was very easy to implement the ideas, we had into Neo4j. Neo4j made it simple to query graph data for tasks like identifying certain paths and counting relationships. The data's graph structure made it possible to use Cypher to create straightforward searches that highlighted patterns and relationships. The 'COUNT' function made it relatively easy to complete tasks that required counting connections, including locating cities with exactly two connections (Task 11). Such aggregations are naturally supported by the graph structure.
- **Differences and Challenges:** It was time-consuming to prepare the dataset rather than performing the queries. While Cypher is user-friendly, it still required some time to know how to upload the big dataset in batches. In Task 13, determining the top three pairs of cities with the greatest direct distances required sorting relationships by a property

('r.type'). This query highlighted Neo4j's ability to handle property-based sorting effectively. However, reliable results required that the attribute be correctly typed (numeric rather than string). Some tasks, like finding cities connected to both "Denver" and "Seattle," required more advanced query techniques like collecting and filtering data sets. While this was a great learning experience, it was challenging to get everything right on the first try.

- **Insights Gained:** Tasks involving relationships (like roads between cities) are naturally suited for Neo4j. The database efficiently handled queries that involve finding and counting these relationships.
- **Graph-Related Observations:** Neo4j is particularly good at finding the shortest paths between nodes (cities), which is a common need in road networks. It's easier to spot patterns and explain results when we can see how nodes and relationships connect.

In short, Neo4j proved to be a powerful tool for analyzing road networks. Its ability to handle relationships and connections naturally, combined with the simplicity of Cypher, made complex tasks manageable. Working with Neo4j for this assignment offered valuable insights into how graph databases handle network analysis tasks. Overall, the experience was intuitive and rewarding, but it also came with a few challenges worth discussing.