**HYBRID DATA CHALLENGE QUERIES**

**SQL Challenges:**

* **Dynamic JSON Construction:**

Task: Construct a dynamic JSON that not only includes direct connections but also nested JSON objects representing second-level connections.

* **Recursive Path Finding:**

Task: Use recursive common table expressions (CTEs) to find all possible routes from one city to another with a limit on the number of hops.

* **Aggregate Network Analysis:**

Task: Calculate the most central city based on the shortest total travel distance to all other cities.

**MongoDB Challenges:**

* **Multi-Level Document Embedding:**

Task: Embed documents representing direct and second-level connections within each city document.

* **Simulated Transactions:**

Task: Perform a transaction-like sequence in MongoDB where you update multiple documents based on route modifications.

* **Aggregation of Connection Degrees:**

Task: Calculate the degree of each city (number of direct connections) and order by those most connected.

**Neo4j Challenges:**

* **Complex Path Analysis:**

Task: Identify all paths that connect two cities with exactly three stops in between.

* **Centrality Calculation:**

Task: Compute centrality measures for nodes to determine the most influential city in the network based on its closeness to all other nodes.

* **Advanced Relationship Querying:**

Task: Find cities that form a triangular route with each leg not exceeding a specified maximum distance, illustrating a complex relationship pattern.

**UNIFIED QUERIES SET**

1. **Retrieve all direct connections from a specified city:**

Goal: Identify all cities directly connected to a given city, along with the distances.

1. **Calculate the total and average distance for routes from a specified city:**

Goal: Compute the sum and average of distances for all routes originating from a specified city.

1. **List cities with more than a specified number of direct connections:**

Goal: Find cities that serve as major hubs in the network, based on a threshold for the number of connections.

1. **Find the shortest route between two specified cities:**

Goal: Determine the shortest path from one city to another, focusing on minimal distance.

1. **Aggregate the total number of connections each city has:**

Goal: Calculate the total number of incoming and outgoing routes for each city.

1. **Identify cities that only have outgoing routes:**

Goal: Determine which cities do not have any incoming connections but have outgoing routes, highlighting potential data anomalies or unique cases.

1. **Rank cities by their connectivity to other cities within a certain distance:**

Goal: Order cities based on the number of connections they have to other cities within a specified distance range, identifying strategically placed cities for travel or transport.

**PERFORMANCE METRICS**

* **Execution Time Measurement:**

Use Python’s time module to measure how long each query takes to execute.

* **Resource Utilization Monitoring:**

Utilize the `*psutil*` library to monitor CPU and memory usage. Set up a background task that records these metrics at regular intervals (e.g., every second) during the query execution.

* **Visualization**: Plots the timing and the resources