**Vision: SmartRoute – Graph-Driven Delivery Optimization**

**Neo4j Business Case Scenario: Optimizing Delivery Routes and Pickup Locations**

**Business Example:** AGM is looking to enhance its delivery system by adding more pickup locations and utilizing BART for transportation, avoiding traffic congestion and speeding up delivery times. This requires an understanding of the network of locations, routes, and their relationships.

**Solution with Neo4j:**

* **Modeling the Network:** Use Neo4j to model pickup locations, BART stations, and customer addresses as nodes. Relationships between these nodes represent possible routes, including considerations for distance, time, and traffic conditions.
* **Route Optimization:** Apply graph algorithms available in Neo4j, such as shortest path and Dijkstra's algorithm, to find the most efficient delivery routes.
* **Strategic Pickup Locations:** Analyze the graph to identify potential locations for new pickup points based on centrality measures, ensuring they are optimally placed for both customers and delivery logistics.

**Coding Approaches:**

**Modeling the Network:**

def find\_shortest\_path(self, start, end):

with self.driver.session() as session:

return session.read\_transaction(self.\_find\_and\_return\_shortest\_path, start, end)

@staticmethod

def \_find\_and\_return\_shortest\_path(tx, start, end):

query = (

"MATCH (start:Station {name: $start}), (end:Customer {name: $end}) "

"CALL gds.shortestPath.dijkstra.stream('myGraph', { "

"sourceNode: start, "

"targetNode: end, "

"relationshipWeightProperty: 'distance' "

"}) "

"YIELD path "

"RETURN path"

)

result = tx.run(query, start=start, end=end)

return [record["path"] for record in result]

# Example: Find shortest path

path = model.find\_shortest\_path("Downtown Berkeley", "Customer XYZ")

print(path)

**Route Optimization**

def find\_shortest\_path(self, start, end):

with self.driver.session() as session:

return session.read\_transaction(self.\_find\_and\_return\_shortest\_path, start, end)

@staticmethod

def \_find\_and\_return\_shortest\_path(tx, start, end):

query = (

"MATCH (start:Station {name: $start}), (end:Customer {name: $end}) "

"CALL gds.shortestPath.dijkstra.stream('myGraph', { "

"sourceNode: start, "

"targetNode: end, "

"relationshipWeightProperty: 'distance' "

"}) "

"YIELD path "

"RETURN path"

)

result = tx.run(query, start=start, end=end)

return [record["path"] for record in result]

# Example: Find shortest path

path = model.find\_shortest\_path("Downtown Berkeley", "Customer XYZ")

print(path)

**Strategic Pickup Locations Analysis**

def analyze\_centrality(self):

with self.driver.session() as session:

return session.read\_transaction(self.\_analyze\_and\_return\_centrality)

@staticmethod

def \_analyze\_and\_return\_centrality(tx):

query = (

"CALL gds.betweenness.stream('myGraph') "

"YIELD nodeId, score "

"RETURN gds.util.asNode(nodeId).name AS name, score "

"ORDER BY score DESC"

)

result = tx.run(query)

return [(record["name"], record["score"]) for record in result]

# Example: Analyze centrality to find strategic pickup locations

centrality\_scores = model.analyze\_centrality()

print(centrality\_scores)