**Importing Data with Cypher**

LOAD CSV WITH HEADERS FROM 'file:///UniqueStops.csv' AS row

MERGE (:Location {id: toInteger(row.AutoID), name: row.StopID, latitude: toFloat(row.Latitude), longitude: toFloat(row.Longitude)});

**CREATE INDEX FOR (n:Location) ON (n.id);**

CALL apoc.periodic.iterate(

  'LOAD CSV WITH HEADERS FROM "file:///TravelDistances.csv" AS row RETURN row',

  'MATCH (from:Location {id: toInteger(row.FromStopID)}), (to:Location {id: toInteger(row.ToStopID)})

   CREATE (from)-[:ROUTE {RouteID: row.RouteID, StopType: row.StopType, TravelTime: toFloat(row.TravelTimeSeconds), DistanceKM: toFloat(row.DistanceKM)}]->(to)',

  {batchSize: 1000, iterateList: true, parallel: true}

);

//

CALL apoc.periodic.iterate(

'LOAD CSV WITH HEADERS FROM "file:///TravelDistances.csv" AS row RETURN row',

'MERGE (from:Location {name: row.FromStopID})

MERGE (to:Location {name: row.ToStopID})

CREATE (from)-[:ROUTE {RouteID: row.RouteID, StopType: row.StopType, TravelTime: toFloat(row.TravelTimeSeconds), DistanceKM: toFloat(row.DistanceKM)}]->(to)',

{batchSize: 10000, parallel: true}

);

**Data interrogation and analysis**

Use Cypher to query the data:

Copy Code // Query all stops and total transit time for a route

MATCH (r:Route {route\_id: 'RouteID\_00143bda-0a6b-49ec-bb35-36593d303e77'})-[:HAS\_STOP]->(s:Stop)

RETURN r, s;

// Calculation of the total transport time for a route

MATCH (r:Route {route\_id: 'RouteID\_00143bda-0a6b-49ec-bb35-36593d303e77'})-[:HAS\_STOP]->(s1:Stop)-[t:TRAVEL\_TIME]->(s2:Stop)

RETURN r.route\_id, sum(t.time\_seconds) AS total\_travel\_time;

// Determine the point where a specific path number and starting point type is Station and use it as the starting point

MATCH (start:Location)-[r:ROUTE]->(end:Location)

WHERE r.StopType = 'Station'and r.RouteID='RouteID\_09a9a44d-e007-42de-9437-31b5fd392648'

return distinct end

/ Create graph projection

CALL gds.graph.project(

'myGraph',

'Location',

{

ROUTE: {

properties: 'DistanceKM'

}

}

);

/ Get a point with a specific path number and a starting point type of Station and use it as the starting point

MATCH (start:Location)-[r:ROUTE]->(station:Location)

WHERE r.StopType = 'Station' AND r.RouteID = 'RouteID\_09a9a44d-e007-42de-9437-31b5fd392648'

WITH DISTINCT station AS start

// Get all points associated with this path number

MATCH (:Location)-[r:ROUTE {RouteID: 'RouteID\_09a9a44d-e007-42de-9437-31b5fd392648'}]->(end:Location)

WITH DISTINCT start, collect(end) AS nodes

// Calculate the shortest path from the start point to all end points

CALL {

  WITH start, nodes

  UNWIND nodes AS end

  CALL gds.shortestPath.dijkstra.stream(

    'myGraph',

    { sourceNode: id(start), targetNode: id(end), relationshipWeightProperty: 'DistanceKM' }

  )

  YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs

  RETURN

    gds.util.asNode(sourceNode).id AS fromNode,

    gds.util.asNode(targetNode).id AS toNode,

    totalCost

}

WITH start, collect({fromNode: fromNode, toNode: toNode, cost: totalCost}) AS paths

CALL apoc.periodic.iterate(

  'MATCH (r:ROUTE) RETURN DISTINCT r.RouteID AS routeId',

  'WITH routeId

   CALL {

     WITH routeId

     CALL gds.graph.project(

       routeId,

       "Location",

       {

         ROUTE: {

           type: "ROUTE",

           properties: "TravelTime"

         }

       }

     )

     YIELD graphName

     RETURN graphName

   }

   RETURN graphName',

  {batchSize: 1}

);

CALL apoc.periodic.iterate(

  'MATCH (from:Location)-[r:ROUTE]->(to:Location) WHERE r.StopType = "Station" RETURN DISTINCT r.RouteID AS routeId, to',

  'MERGE (s:Start {routeId: routeId, id: id(to)})',

  {batchSize: 1}

);

LOAD CSV WITH HEADERS FROM 'file:///RouteSummary.csv' AS row

MERGE (r:Route {

    RouteID: row.RouteID,

    TravelTime: toFloat(row.TotalTravelTime),

    DistanceKM: toFloat(row.TotalDistanceKM),

    AverageSpeedKMH:toFloat(row.AverageSpeedKMH),

    PackageDensity: toFloat(row.TotalPackageDensity),

    AvgPackagesPerStop: toFloat(row.AvgPackagesPerStop),

    ServiceTimeRatio: toFloat(row.ServiceTimeRatio),

    TimeWindowCompliance: toFloat(row.TimeWindowCompliance),

    DeliverySuccessRate: toFloat(row.DeliverySuccessRate),

    RouteScore: (CASE row.RouteScore

                WHEN 'High' THEN 1

                WHEN 'Medium' THEN 2

                WHEN 'Low' THEN 3

                ELSE NULL

                END),

    InvalidSequenceScore: toFloat(row.InvalidSequenceScore)