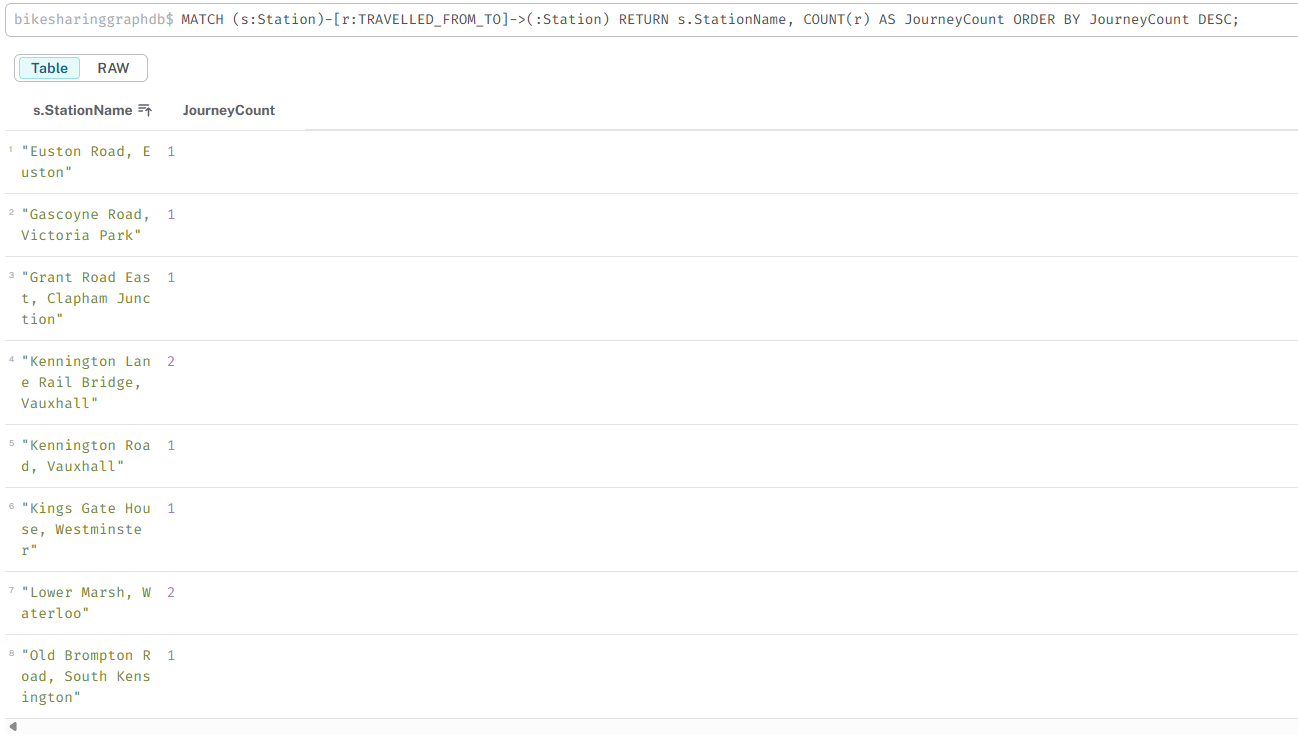
**SQL and CQL, Neo4j screen shots**

// Query 1: Total Journeys per Station (Starting Station)

MATCH (s:Station)-[r:TRAVELLED\_FROM\_TO]->(:Station)

RETURN s.StationName, COUNT(r) AS JourneyCount

ORDER BY JourneyCount DESC;

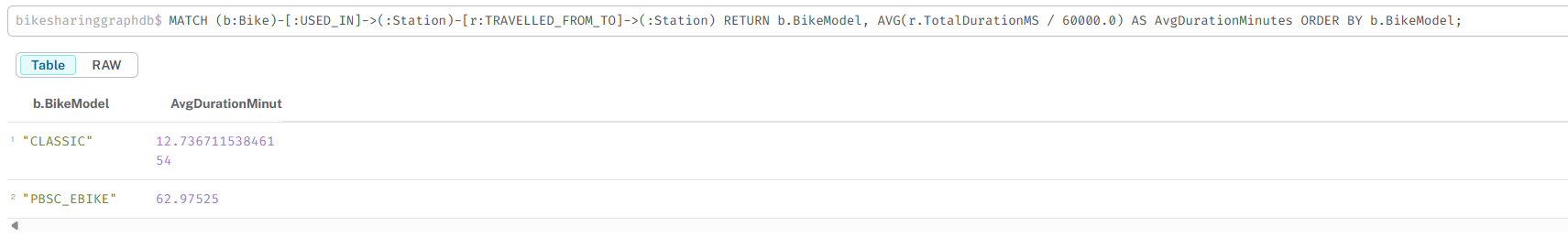


// Query 2: Average Journey Duration by Bike Model

MATCH (b:Bike)-[:USED\_IN]->(:Station)-[r:TRAVELLED\_FROM\_TO]->(:Station)

RETURN b.BikeModel, AVG(r.TotalDurationMS / 60000.0) AS AvgDurationMinutes

ORDER BY b.BikeModel;



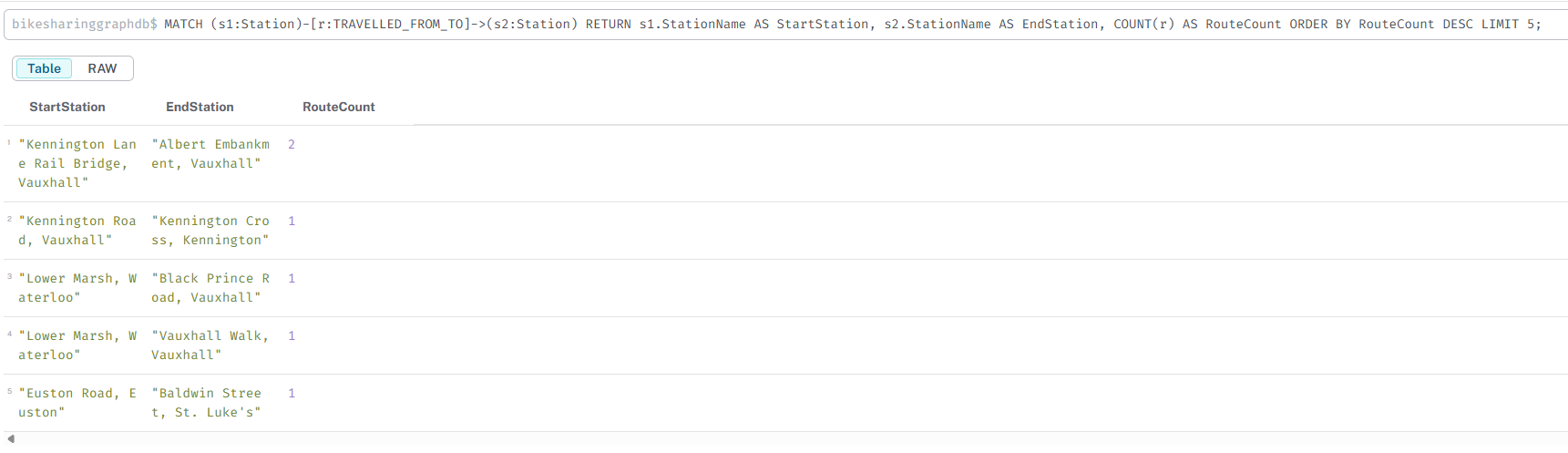
// Query 3: Most Popular Routes (Station Pairs)

MATCH (s1:Station)-[r:TRAVELLED\_FROM\_TO]->(s2:Station)

RETURN s1.StationName AS StartStation, s2.StationName AS EndStation, COUNT(r) AS RouteCount

ORDER BY RouteCount DESC

LIMIT 5;



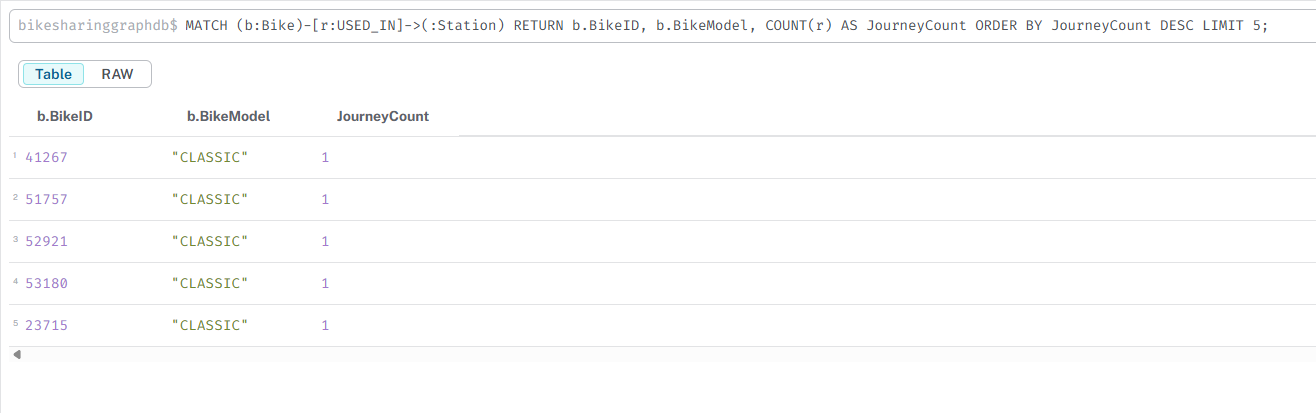
// Query 4: Bikes with Most Journeys

MATCH (b:Bike)-[r:USED\_IN]->(:Station)

RETURN b.BikeID, b.BikeModel, COUNT(r) AS JourneyCount

ORDER BY JourneyCount DESC

LIMIT 5;

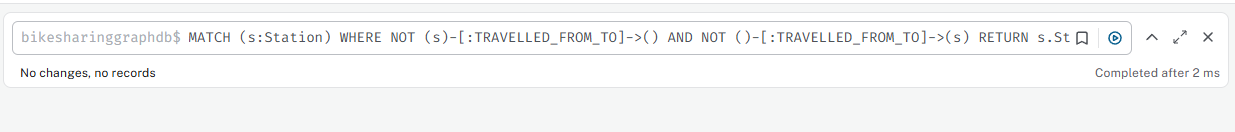


// Query 5: Stations with No Journeys

MATCH (s:Station)

WHERE NOT (s)-[:TRAVELLED\_FROM\_TO]->() AND NOT ()-[:TRAVELLED\_FROM\_TO]->(s)

RETURN s.StationName;



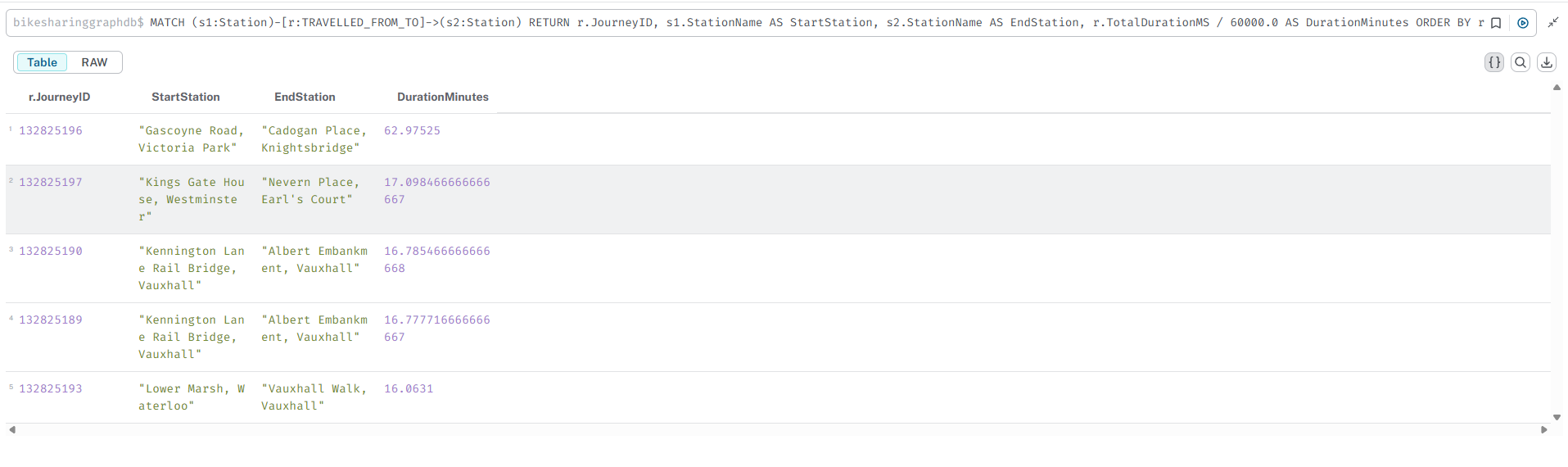
// Query 6: Longest Journeys by Duration

MATCH (s1:Station)-[r:TRAVELLED\_FROM\_TO]->(s2:Station)

RETURN r.JourneyID, s1.StationName AS StartStation, s2.StationName AS EndStation, r.TotalDurationMS / 60000.0 AS DurationMinutes

ORDER BY r.TotalDurationMS DESC

LIMIT 5;

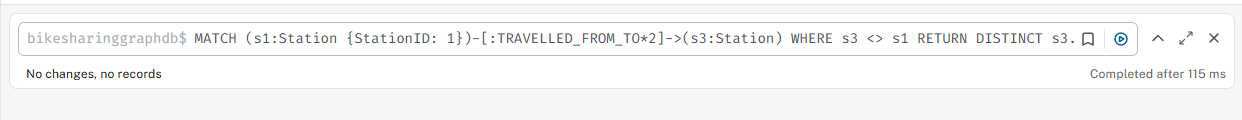


// Query 7: Station Connectivity (Stations Reachable in Two Hops)

MATCH (s1:Station {StationID: 1})-[:TRAVELLED\_FROM\_TO\*2]->(s3:Station)

WHERE s3 <> s1

RETURN DISTINCT s3.StationName AS ReachableStation;



-- Query 1: Total Journeys per Station (Starting Station)

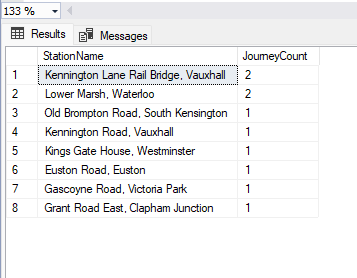
SELECT s.StationName, COUNT(j.JourneyID) AS JourneyCount

FROM Stations s

JOIN Journeys j ON s.StationID = j.StartStationID

GROUP BY s.StationName

ORDER BY JourneyCount DESC;



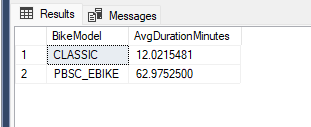
-- Query 2: Average Journey Duration by Bike Model

SELECT b.BikeModel, AVG(j.TotalDurationMS / 60000.0) AS AvgDurationMinutes

FROM Bikes b

JOIN Journeys j ON b.BikeID = j.BikeID

GROUP BY b.BikeModel;



-- Query 3: Most Popular Routes (Station Pairs)

SELECT

s1.StationName AS StartStation,

s2.StationName AS EndStation,

COUNT(j.JourneyID) AS RouteCount

FROM Journeys j

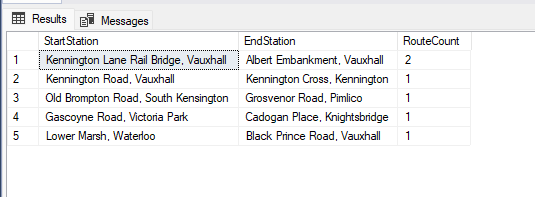
JOIN Stations s1 ON j.StartStationID = s1.StationID

JOIN Stations s2 ON j.EndStationID = s2.StationID

GROUP BY s1.StationName, s2.StationName

ORDER BY RouteCount DESC

OFFSET 0 ROWS FETCH NEXT 5 ROWS ONLY;



-- Query 4: Bikes with Most Journeys

SELECT b.BikeID, b.BikeModel, COUNT(j.JourneyID) AS JourneyCount

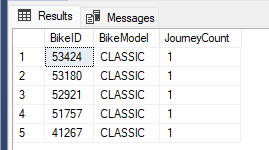
FROM Bikes b

JOIN Journeys j ON b.BikeID = j.BikeID

GROUP BY b.BikeID, b.BikeModel

ORDER BY JourneyCount DESC

OFFSET 0 ROWS FETCH NEXT 5 ROWS ONLY;



-- Query 5: Stations with No Journeys

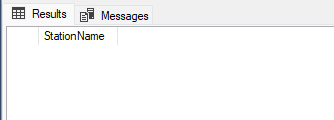
SELECT s.StationName

FROM Stations s

LEFT JOIN Journeys j1 ON s.StationID = j1.StartStationID

LEFT JOIN Journeys j2 ON s.StationID = j2.EndStationID

WHERE j1.JourneyID IS NULL AND j2.JourneyID IS NULL;



-- Query 6: Longest Journeys by Duration

SELECT

j.JourneyID,

s1.StationName AS StartStation,

s2.StationName AS EndStation,

j.TotalDurationMS / 60000.0 AS DurationMinutes

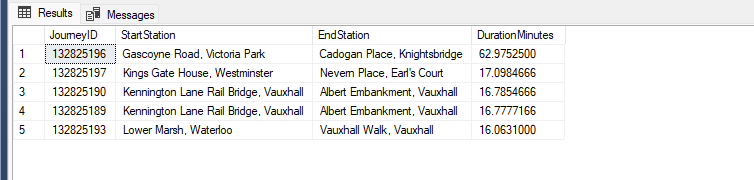
FROM Journeys j

JOIN Stations s1 ON j.StartStationID = s1.StationID

JOIN Stations s2 ON j.EndStationID = s2.StationID

ORDER BY j.TotalDurationMS DESC

OFFSET 0 ROWS FETCH NEXT 5 ROWS ONLY;



-- Query 7: Station Connectivity (Stations Reachable in Two Hops)

SELECT DISTINCT s3.StationName AS ReachableStation

FROM Journeys j1

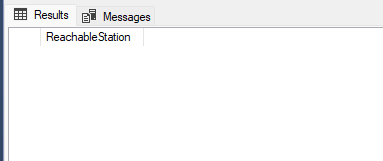
JOIN Journeys j2 ON j1.EndStationID = j2.StartStationID

JOIN Stations s1 ON j1.StartStationID = s1.StationID

JOIN Stations s3 ON j2.EndStationID = s3.StationID

WHERE s1.StationID = 1

AND s3.StationID <> s1.StationID;



**Compare Relational and Graph Databases**

**Instructions**:

1. **Result Consistency**:
   * Verify that SQL and CQL queries produce equivalent results (e.g., same counts, durations).
   * Note any discrepancies (e.g., due to import errors) and document them.
2. **Storage Differences**:
   * **Relational (SQL Server)**:
     + Data is stored in normalized tables (Stations, Bikes, Journeys) with foreign keys.
     + Relationships are implicit via StartStationID, EndStationID, and BikeID.
     + Requires joins to reconstruct relationships, which can be complex (e.g., Query 7 uses multiple JOINs).
     + Schema is rigid, requiring predefined tables and columns.
   * **Graph (Neo4j)**:
     + Data is stored as nodes (Station, Bike) and relationships (TRAVELLED\_FROM\_TO, USED\_IN).
     + Relationships are explicit, stored as first-class entities with properties.
     + Schema is flexible, allowing easy addition of new node/relationship types.
3. **Retrieval Differences**:
   * **Relational**:
     + Efficient for aggregations (Queries 1, 2, 4) due to indexing (e.g., IDX\_Journey\_StartDateTime).
     + Complex for relationship-based queries (Query 7 requires multiple JOINs, potentially slow for large datasets).
     + SQL syntax is verbose for path-based queries.
   * **Graph**:
     + Excels at relationship queries (Queries 3, 5, 7) via direct traversal (e.g., [:TRAVELLED\_FROM\_TO\*2]).
     + Cypher is concise and intuitive for connectivity (Query 7 is simpler than SQL).
     + May be slower for simple aggregations compared to optimized SQL.
4. **Performance**:
   * **Queries 1, 2, 4**: SQL may be faster due to indexing and optimized query execution plans.
   * **Queries 3, 5, 7**: Neo4j is typically faster for relationship-heavy queries, as it avoids joins and uses graph traversal.
   * **Query 6**: Comparable performance, as it’s a simple sort operation.
   * Example: Query 7 may take 50ms in SQL (due to JOINs) vs. 10ms in Neo4j (direct traversal).
5. **Ease of Use**:
   * SQL is familiar for tabular data but requires complex joins for relationships.
   * Cypher is intuitive for graph traversals, with Neo4j’s visual interface aiding analysis (e.g., Query 7’s graph visualization).