SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya School of Engineering
(formerly K J Somaiya College of Engineering)

Batch: D-2	Roll No.: 16010123325
Experiment No. 5	

TITLE: Create a graph database and perform basic analysis using Neo4j.

AIM: Create a property-graph database in Neo4j, populate it with nodes and relationships, and perform basic exploratory analysis using Cypher (counts, paths, and simple centrality-like measures), along with visual inspection in Neo4j Browser.

Expected OUTCOME of Experiment:
CO3: Perform the social data analytics

Books/ Journals/ Websites referred:

https://www.youtube.com/watch?v=8jNPelugC2s&ab_channel=LaitureAcademy

Pre Lab/ Prior Concepts:

Students should have a basic understanding of:

1. Graph theory basics: nodes/vertices, edges/relationships, degree, path
2. Property graph model: labels, relationship types, properties.
3. Cypher essentials: CREATE, MERGE, MATCH, WHERE, RETURN, ORDER BY, LIMIT, COUNT, WITH.
4. Constraints & indexes: why uniqueness matters; CREATE CONSTRAINT.
5. Neo4j tooling: Neo4j Desktop/Aura Free, Neo4j Browser result views (table vs graph).
6. CSV import with LOAD CSV

Instruction for Students:
Using Neo4j, design and create a small graph database of your choice (e.g., social network, movies, cities, or any domain). Insert at least 5–6 nodes and meaningful relationships, then perform basic analysis such as counting nodes/edges, finding friends-of-friends, degree of nodes, shortest path, and one additional query of your own design.

Procedure:

1) Setup

1. Install & open **Neo4j Desktop** (or use **Neo4j Aura Free**).
2. Create a new DBMS / project and start a database. Open **Neo4j Browser**.
3. In the Browser, confirm the connection (you should see a neo4j> prompt).

2) Create a clean schema

Run these in Neo4j Browser:

```
// Reset (ONLY if this is a fresh lab DB)
MATCH (n) DETACH DELETE n;

// Uniqueness constraint for Person name
CREATE CONSTRAINT person_name_unique IF NOT EXISTS
FOR (p:Person)
REQUIRE p.name IS UNIQUE;

// Optional index to speed up lookups
CREATE INDEX person_name_index IF NOT EXISTS
FOR (p:Person) ON (p.name);
```

3) Insert a mini social network

```
// Nodes
CREATE (:Person {name:'Aarav', city:'Mumbai', age:28});
CREATE (:Person {name:'Bhavna', city:'Pune', age:31});
CREATE (:Person {name:'Chetan', city:'Mumbai', age:26});
CREATE (:Person {name:'Deepti', city:'Delhi', age:29});
CREATE (:Person {name:'Eshan', city:'Bengaluru', age:33});

// Relationships (undirected concept, but stored as directed)
MATCH (a:Person {name:'Aarav'}), (b:Person {name:'Bhavna'}) CREATE (a)-[:KNOWS {since:2022}]->(b);
MATCH (a:Person {name:'Aarav'}), (c:Person {name:'Chetan'}) CREATE (a)-[:KNOWS {since:2023}]->(c);
MATCH (b:Person {name:'Bhavna'}), (c:Person {name:'Chetan'}) CREATE (b)-[:KNOWS {since:2021}]->(c);
MATCH (b:Person {name:'Bhavna'}), (d:Person {name:'Deepti'}) CREATE (b)-[:KNOWS {since:2020}]->(d);
MATCH (c:Person {name:'Chetan'}), (e:Person {name:'Eshan'}) CREATE (c)-[:KNOWS {since:2024}]->(e);
MATCH (d:Person {name:'Deepti'}), (e:Person {name:'Eshan'}) CREATE (d)-[:KNOWS {since:2022}]->(e);

// Optionally add reverse links to simulate undirected edges
```

```
MATCH (x)-[r:KNOWS]->(y) MERGE (y)-[:KNOWS {since:r.since}]->(x);
```

4) Quick sanity checks & exploration

```
// Counts
MATCH (n) RETURN COUNT(n) AS total_nodes;
MATCH ()-[r]->() RETURN COUNT(r) AS total_rels;

// What labels and relationship types exist?
CALL db.labels();
CALL db.relationshipTypes();

// Peek at the graph
MATCH (p:Person)-[r:KNOWS]->(q:Person)
RETURN p, r, q
LIMIT 50; // Switch to graph view in the Browser
```

5) Pattern queries (who knows whom?)

```
// People in Mumbai who know someone outside Mumbai
MATCH (p:Person {city:'Mumbai'})-[:KNOWS]->(q:Person)
WHERE p.city <> q.city
RETURN p.name AS from_mumbai, q.name AS knows_outside, q.city
ORDER BY from_mumbai;

// Friends-of-friends (length exactly 2) excluding direct friends/self
MATCH (p:Person {name:'Aarav'})-[:KNOWS]->()-[:KNOWS]->(fof:Person)
WHERE NOT (p)-[:KNOWS]->(fof) AND p <> fof
RETURN DISTINCT fof.name AS friend_of_friend;
```

6) Basic analysis

a) Degree (how connected is each person?)

```
// Total degree (since we stored both directions)
MATCH (p:Person)
RETURN p.name AS person, size( (p)--() ) AS degree
ORDER BY degree DESC;
```

b) Top “influencers” by degree (top 3)

```
MATCH (p:Person)
RETURN p.name AS person, size( (p)--() ) AS degree
ORDER BY degree DESC
LIMIT 3;
```

c) Mutual friends between two people

```
MATCH (a:Person {name:'Bhavna'})-[:KNOWS]->(m:Person)<-[:KNOWS]-(b:Person
{name:'Chetan'})
RETURN m.name AS mutual_friend;
```

d) Shortest path between two people

```
MATCH (src:Person {name:'Aarav'}), (dst:Person {name:'Eshan'})
CALL {
  WITH src, dst
  MATCH p = shortestPath( (src)-[:KNOWS*]-(dst) )
  RETURN p
}
RETURN p;
```

e) Triangles (3-cycles) through each node (simple clustering hint)

```
MATCH (a:Person)-[:KNOWS]->(b:Person),
    (b)-[:KNOWS]->(c:Person),
    (c)-[:KNOWS]->(a)
RETURN a.name AS person, COUNT(DISTINCT [a,b,c]) AS triangle_count
ORDER BY triangle_count DESC;
```

Optional (if Neo4j Graph Data Science (GDS) library is available in your setup): run degree/betweenness algorithms via GDS for more formal centralities.

7) Filtering, aggregation, and simple reporting

```
// Average age by city among people who know at least 2 others
MATCH (p:Person)
WITH p, size( (p)--() ) AS deg
WHERE deg >= 2
RETURN p.city AS city, ROUND( avg(p.age) ) AS avg_age, COUNT(*) AS people
ORDER BY people DESC;
```

8) (Optional) Import from CSV (if you prepared files in Neo4j's import folder)

```
// Example shape, not executed unless files exist at file:///...
LOAD CSV WITH HEADERS FROM 'file:///people.csv' AS row
MERGE (p:Person {name: row.name})
SET p.city = row.city, p.age = toInteger(row.age);

LOAD CSV WITH HEADERS FROM 'file:///knows.csv' AS row
MATCH (a:Person {name: row.src}), (b:Person {name: row.dst})
MERGE (a)-[:KNOWS {since: toInteger(row.since)}]->(b)
MERGE (b)-[:KNOWS {since: toInteger(row.since)}]->(a);
```

9) Visualize

- In Neo4j Browser, re-run:
- MATCH (p:Person)-[r:KNOWS]->(q:Person) RETURN p,r,q;
- Switch to **Graph** view; drag nodes, examine properties, hover edges to see since.

10) Wrap-up / Clean-up

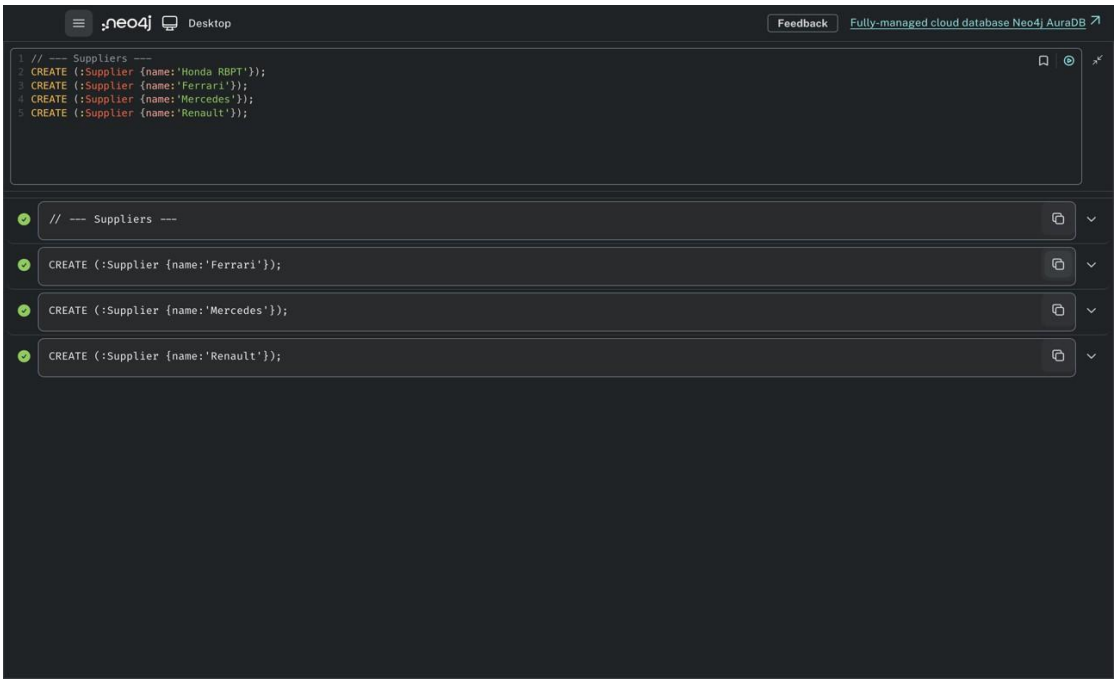
- Save your Cypher script (Browser "Saved scripts" or export).

sys stop the DB after taking screenshots for submission.

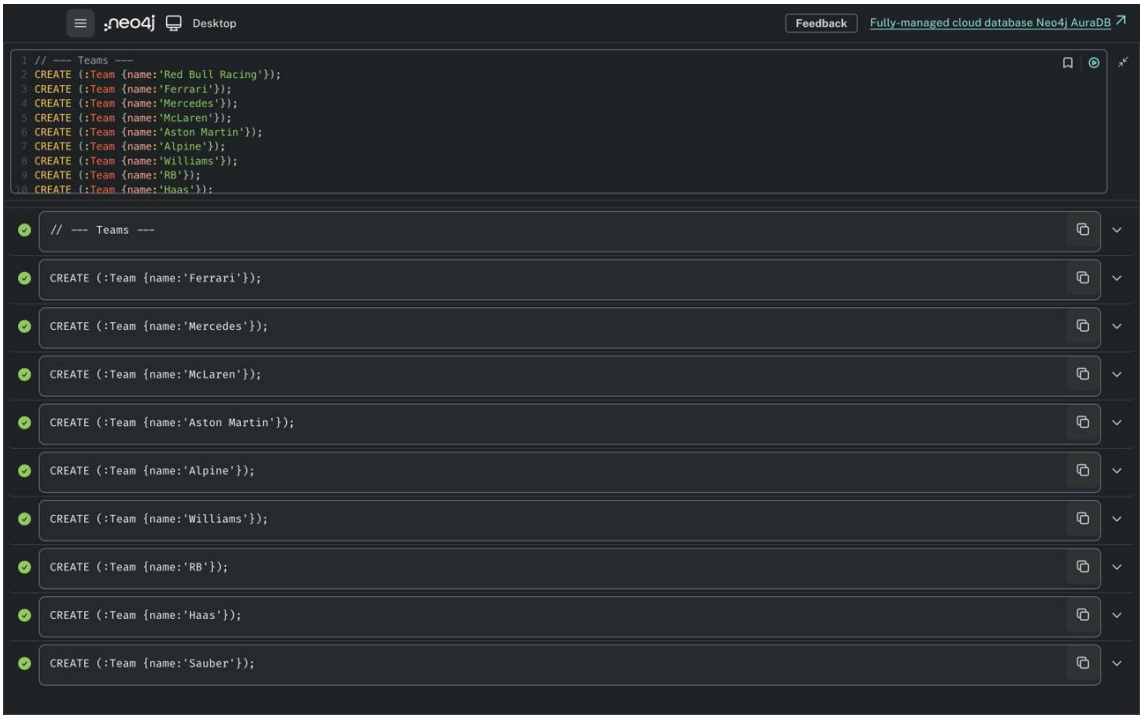
Implementation details:

Create Db

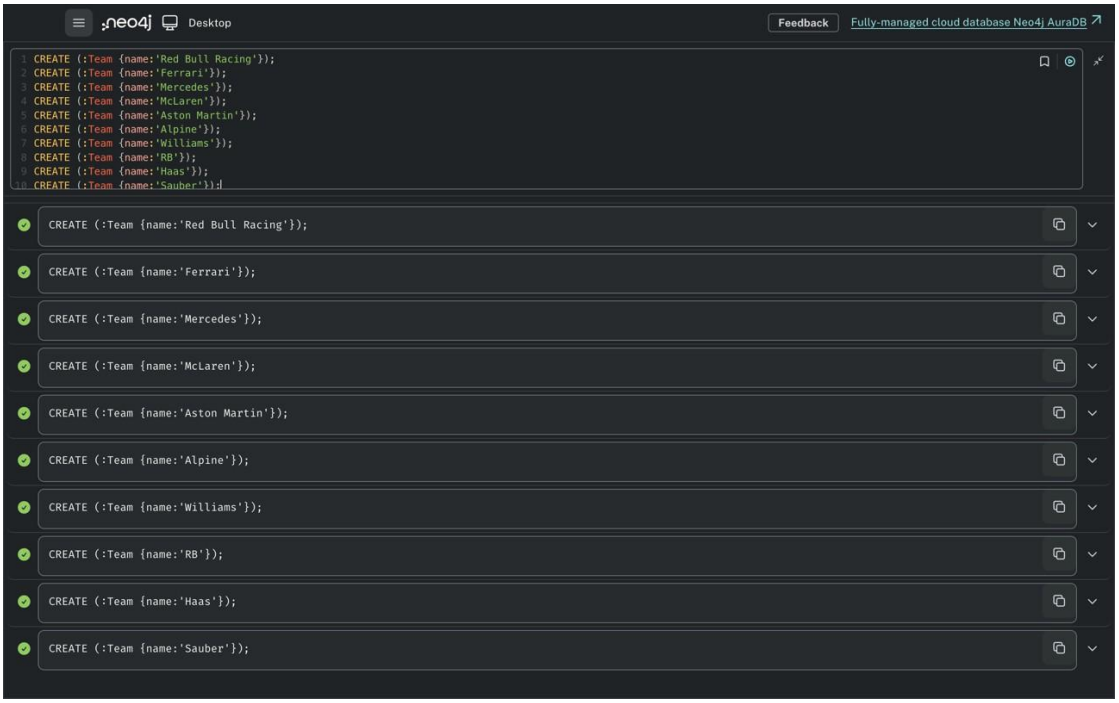
Node Creation-
Supplier



Team

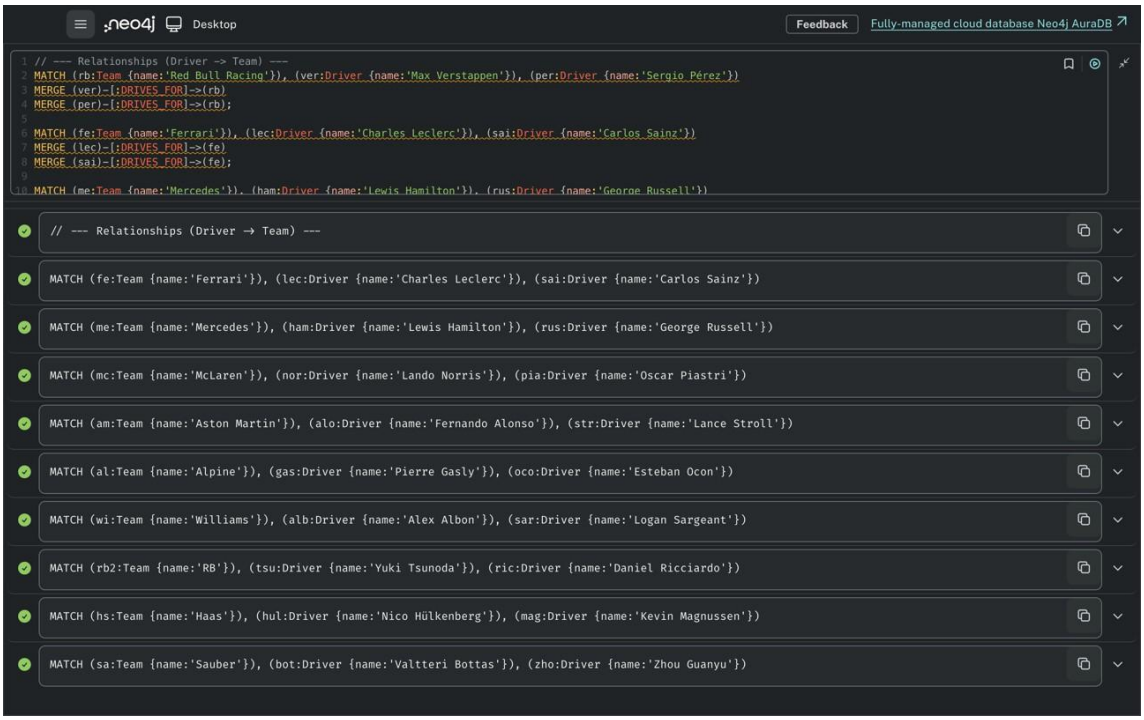

neo4j Desktop
Feedback
Fully-managed cloud database Neo4j AuraDB
// --- Teams ---
CREATE (:Team {name:'Red Bull Racing'});
CREATE (:Team {name:'Ferrari'});
CREATE (:Team {name:'Mercedes'});
CREATE (:Team {name:'McLaren'});
CREATE (:Team {name:'Aston Martin'});
CREATE (:Team {name:'Alpine'});
CREATE (:Team {name:'Williams'});
CREATE (:Team {name:'RB'});
CREATE (:Team {name:'Haas'});
CREATE (:Team {name:'Sauber'});

Driver

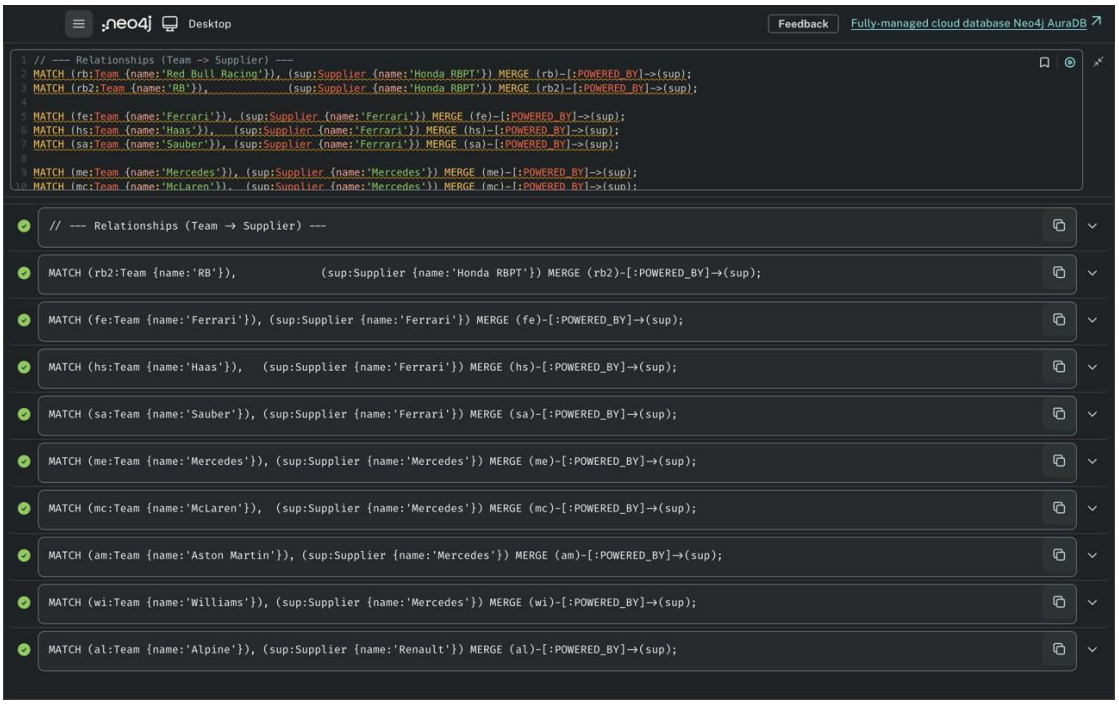

neo4j Desktop
Feedback
Fully-managed cloud database Neo4j AuraDB
CREATE (:Team {name:'Red Bull Racing'});
CREATE (:Team {name:'Ferrari'});
CREATE (:Team {name:'Mercedes'});
CREATE (:Team {name:'McLaren'});
CREATE (:Team {name:'Aston Martin'});
CREATE (:Team {name:'Alpine'});
CREATE (:Team {name:'Williams'});
CREATE (:Team {name:'RB'});
CREATE (:Team {name:'Haas'});
CREATE (:Team {name:'Sauber'});

Relationships

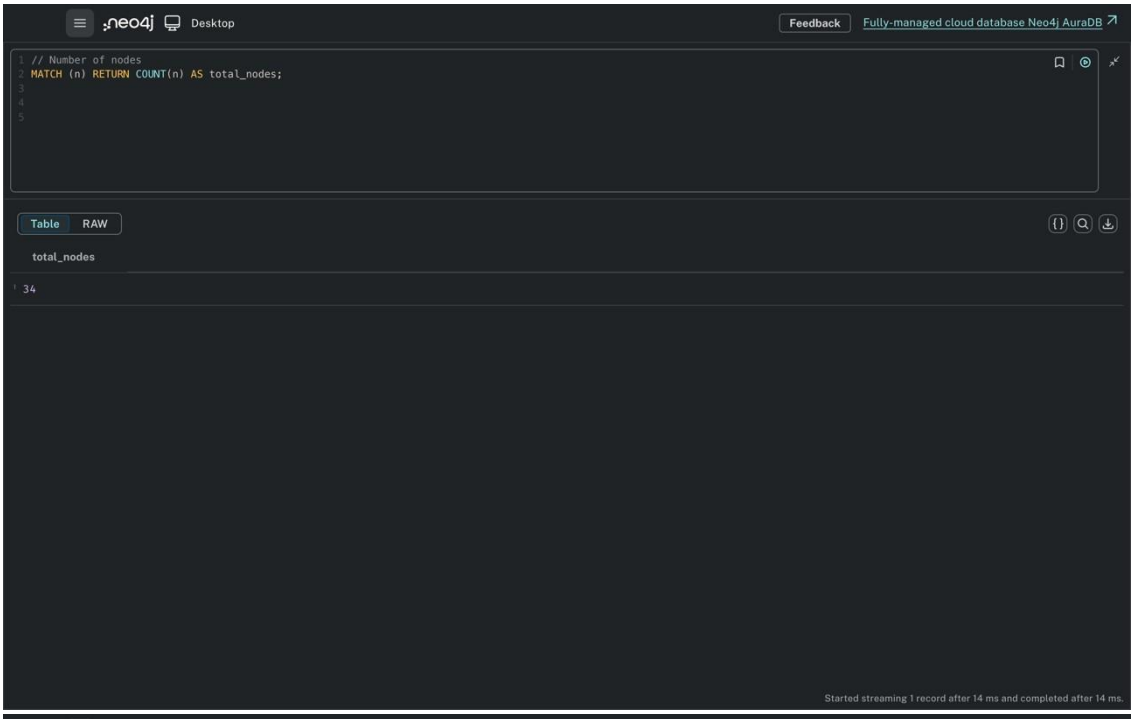
Driver → Team

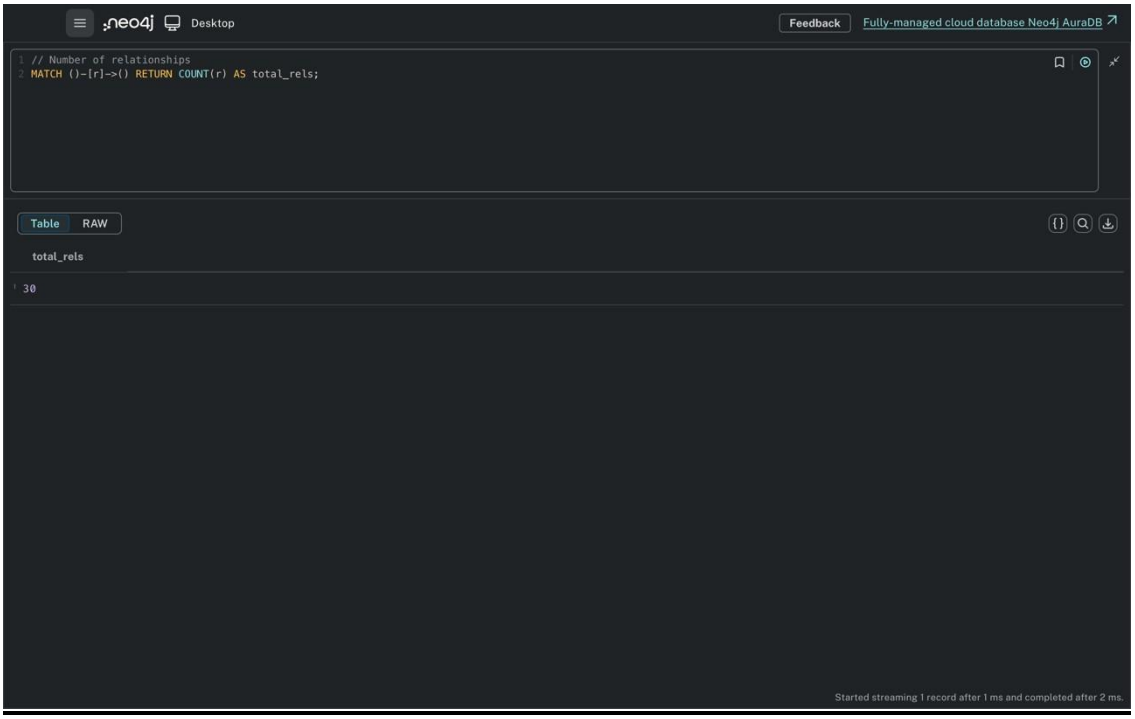


Team → Supplier

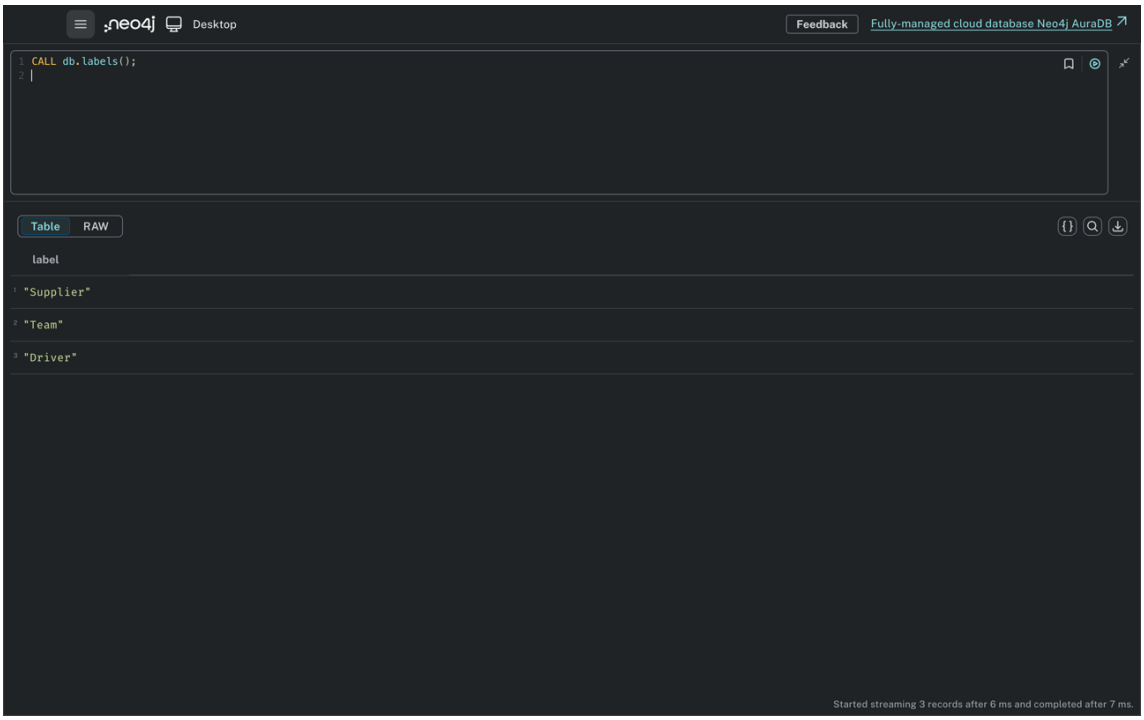


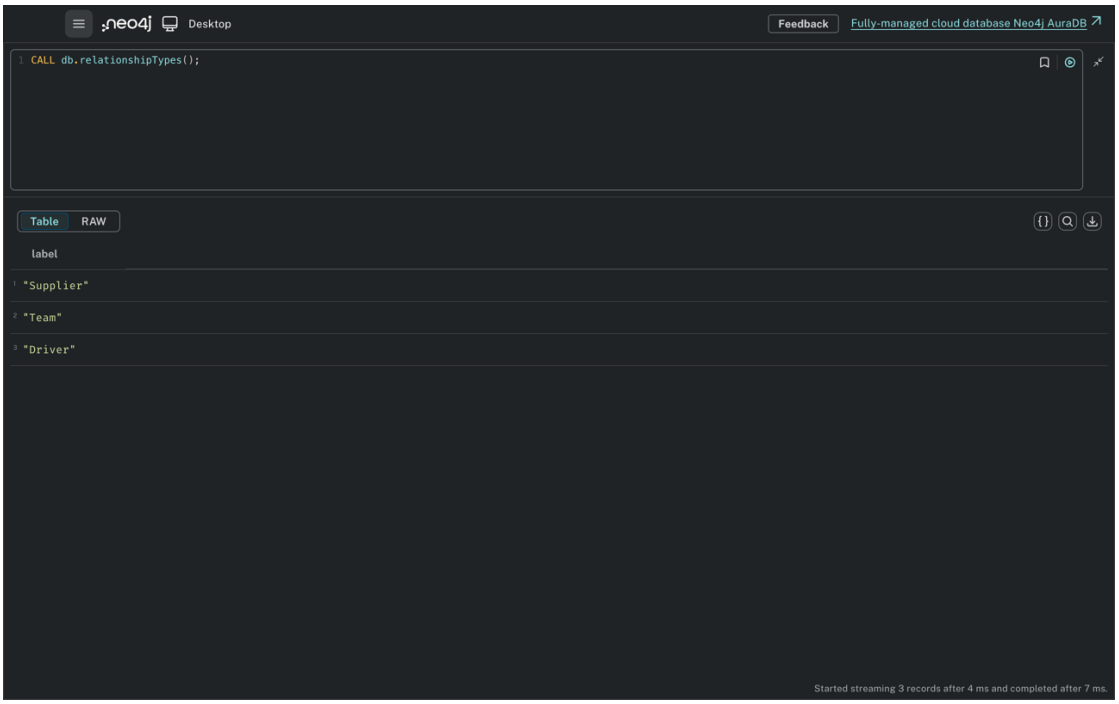
Counts



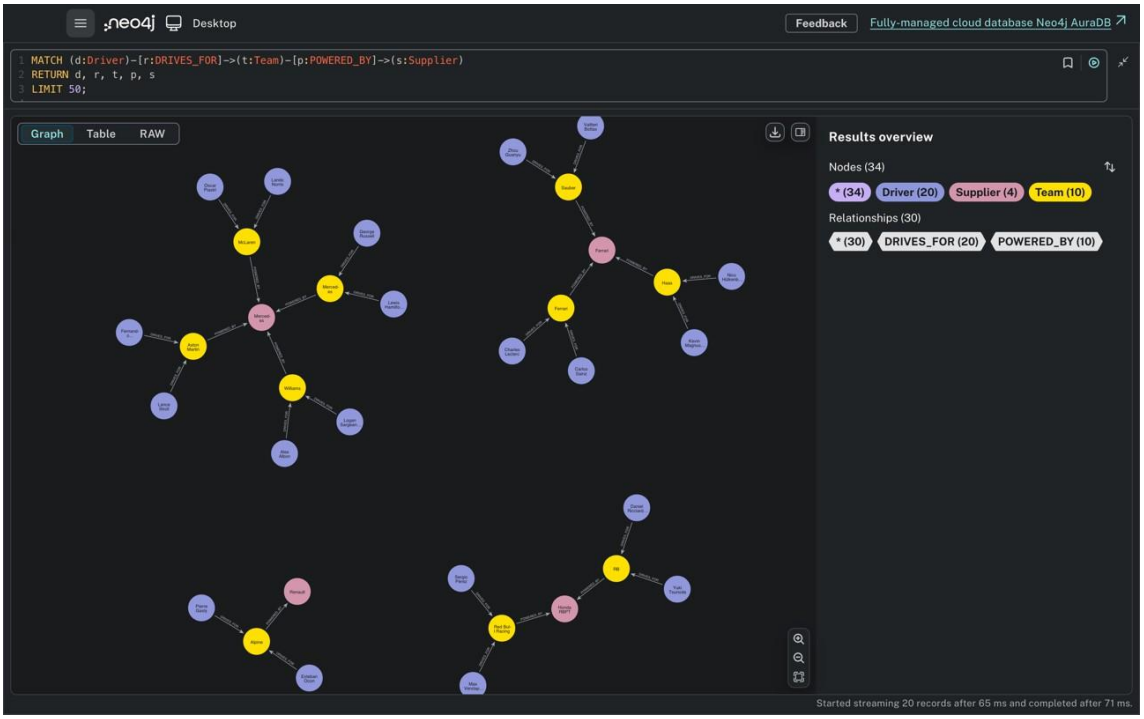


Labels


neo4j
Desktop
Feedback
Fully-managed cloud database Neo4j AuraDB
CALL db.labels();
Table
RAW
label
"Supplier"
"Team"
"Driver"
Started streaming 3 records after 6 ms and completed after 7 ms.

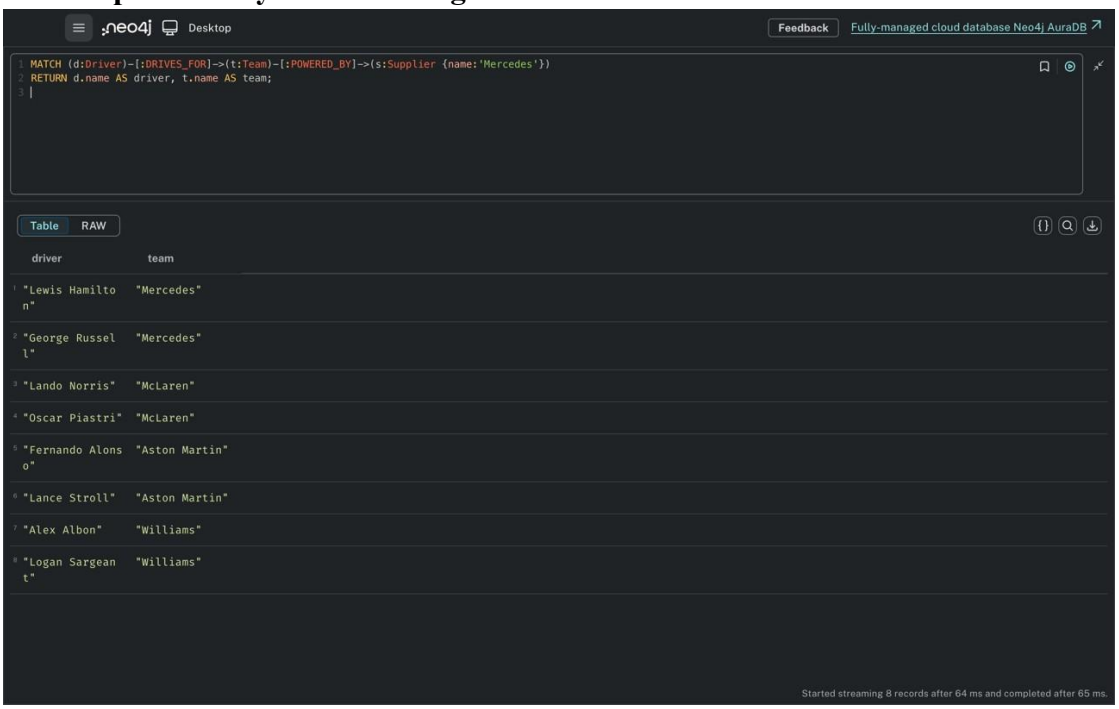

neo4j
Desktop
Feedback
Fully-managed cloud database Neo4j AuraDB
CALL db.relationshipTypes();
Table
RAW
label
"Supplier"
"Team"
"Driver"
Started streaming 3 records after 4 ms and completed after 7 ms.

Peek Graph

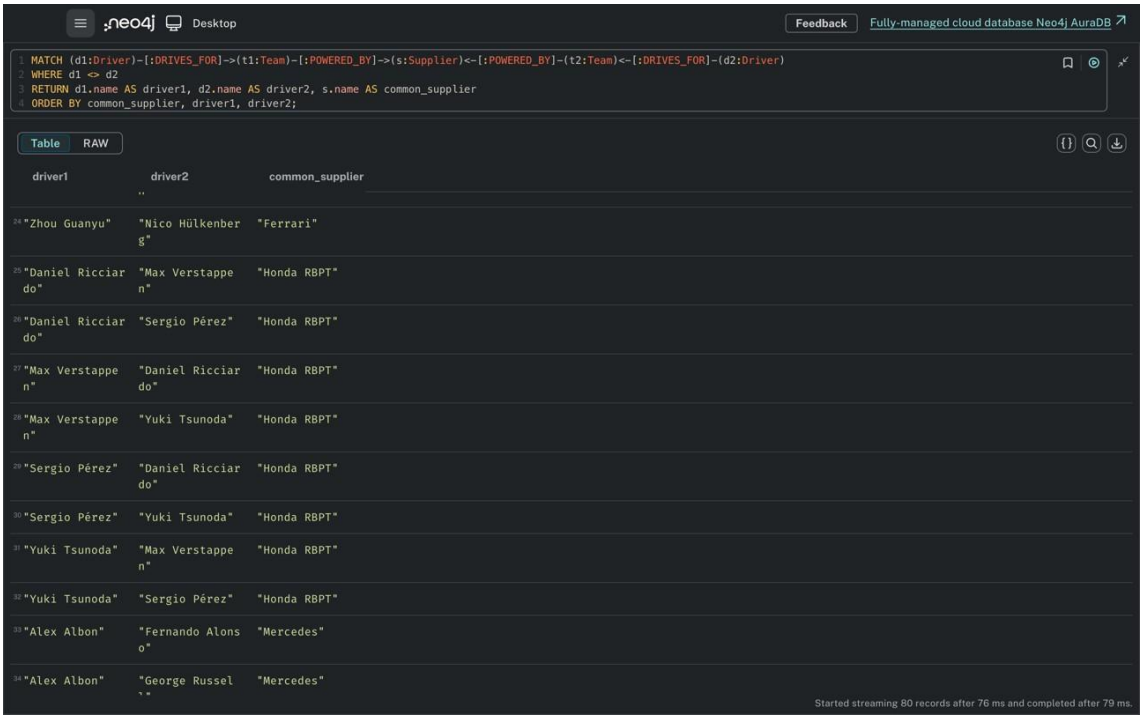


Pattern Queries

Drivers powered by Mercedes engines



Drivers with the same supplier (teammates across different teams)



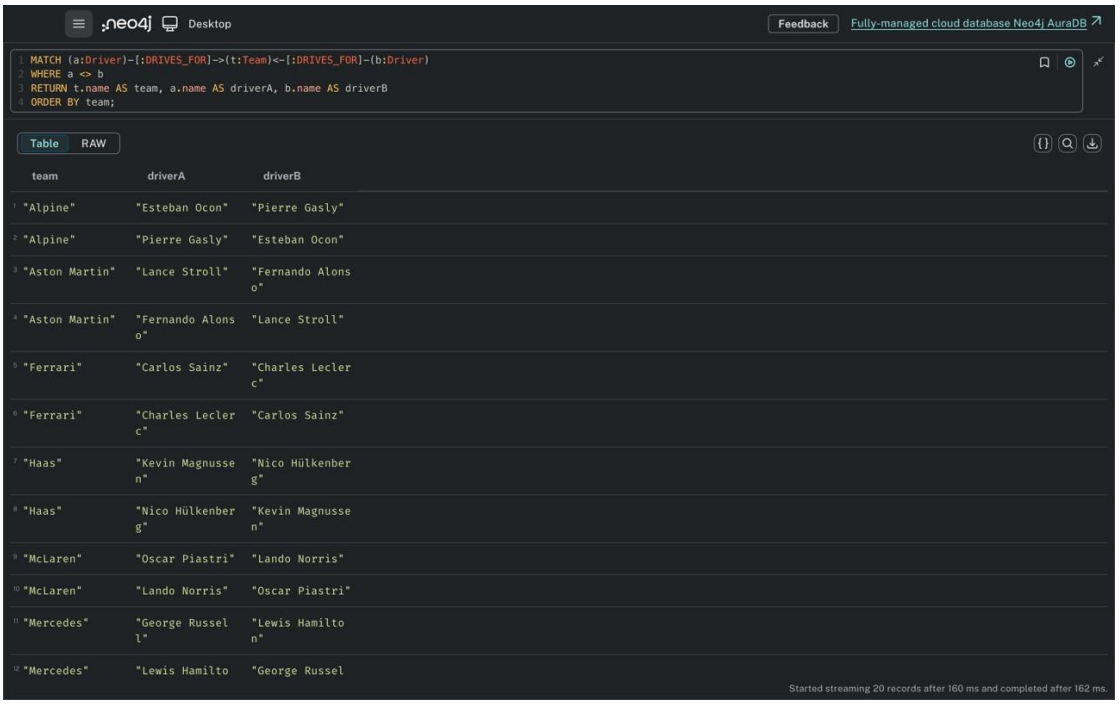
```
MATCH (d1:Driver)-[:DRIVES_FOR]->(t1:Team)-[:POWERED_BY]->(s:Supplier)<-[:POWERED_BY]-(t2:Team)<-[:DRIVES_FOR]-(d2:Driver)
WHERE d1 <> d2
RETURN d1.name AS driver1, d2.name AS driver2, s.name AS common_supplier
ORDER BY common_supplier, driver1, driver2;
```

Table RAW

	driver1	driver2	common_supplier
24	"Zhou Guanyu"	"Nico Hülkenberg"	"Ferrari"
25	"Daniel Ricciardo"	"Max Verstappen"	"Honda RBPT"
26	"Daniel Ricciardo"	"Sergio Pérez"	"Honda RBPT"
27	"Max Verstappen"	"Daniel Ricciardo"	"Honda RBPT"
28	"Max Verstappen"	"Yuki Tsunoda"	"Honda RBPT"
29	"Sergio Pérez"	"Daniel Ricciardo"	"Honda RBPT"
30	"Sergio Pérez"	"Yuki Tsunoda"	"Honda RBPT"
31	"Yuki Tsunoda"	"Max Verstappen"	"Honda RBPT"
32	"Yuki Tsunoda"	"Sergio Pérez"	"Honda RBPT"
33	"Alex Albon"	"Fernando Alonso"	"Mercedes"
34	"Alex Albon"	"George Russel"	"Mercedes"

Started streaming 80 records after 76 ms and completed after 79 ms.

Drivers in the same team (teammates)



```
MATCH (a:Driver)-[:DRIVES_FOR]->(t:Team)<-[:DRIVES_FOR]-(b:Driver)
WHERE a <> b
RETURN t.name AS team, a.name AS driverA, b.name AS driverB
ORDER BY team;
```

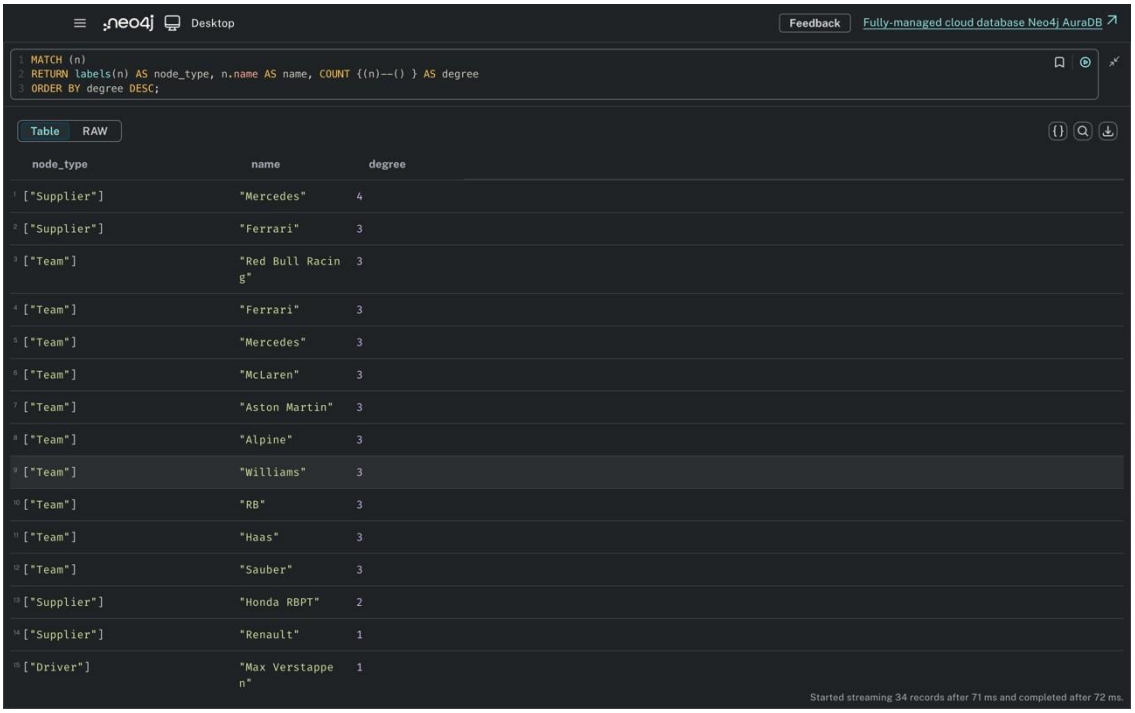
Table RAW

	team	driverA	driverB
1	"Alpine"	"Esteban Ocon"	"Pierre Gasly"
2	"Alpine"	"Pierre Gasly"	"Esteban Ocon"
3	"Aston Martin"	"Lance Stroll"	"Fernando Alonso"
4	"Aston Martin"	"Fernando Alonso"	"Lance Stroll"
5	"Ferrari"	"Carlos Sainz"	"Charles Leclerc"
6	"Ferrari"	"Charles Leclerc"	"Carlos Sainz"
7	"Haas"	"Kevin Magnussen"	"Nico Hülkenberg"
8	"Haas"	"Nico Hülkenberg"	"Kevin Magnussen"
9	"McLaren"	"Oscar Piastri"	"Lando Norris"
10	"McLaren"	"Lando Norris"	"Oscar Piastri"
11	"Mercedes"	"George Russell"	"Lewis Hamilton"
12	"Mercedes"	"Lewis Hamilto	"George Russel

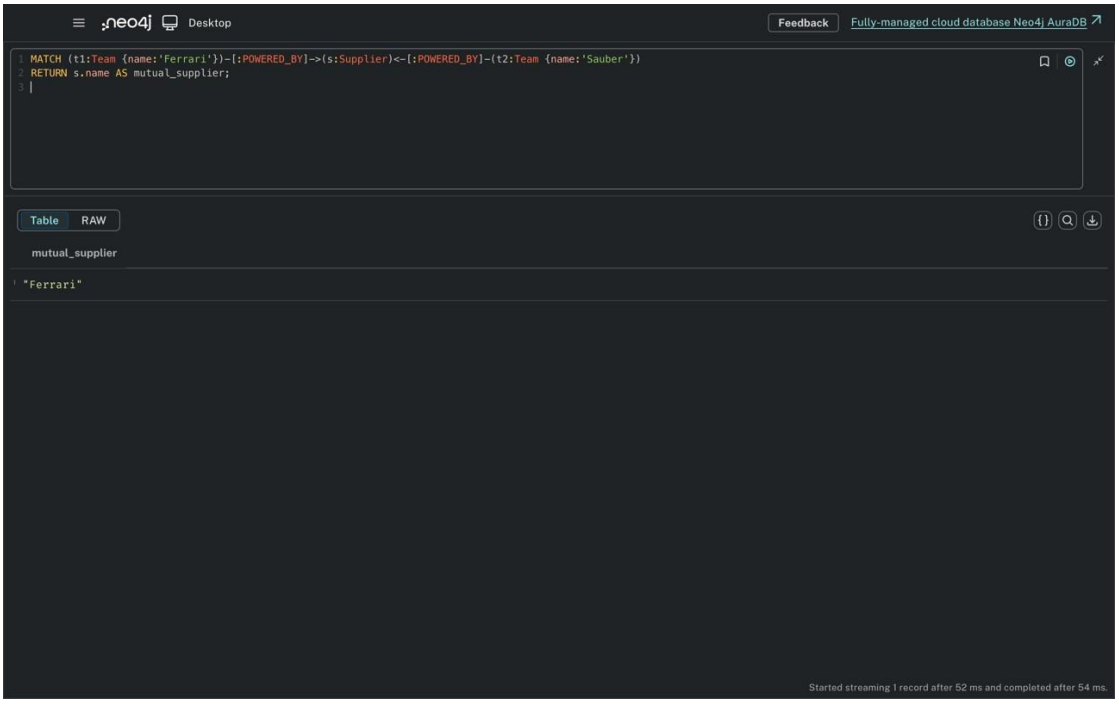
Started streaming 20 records after 160 ms and completed after 162 ms.

Basic Analysis

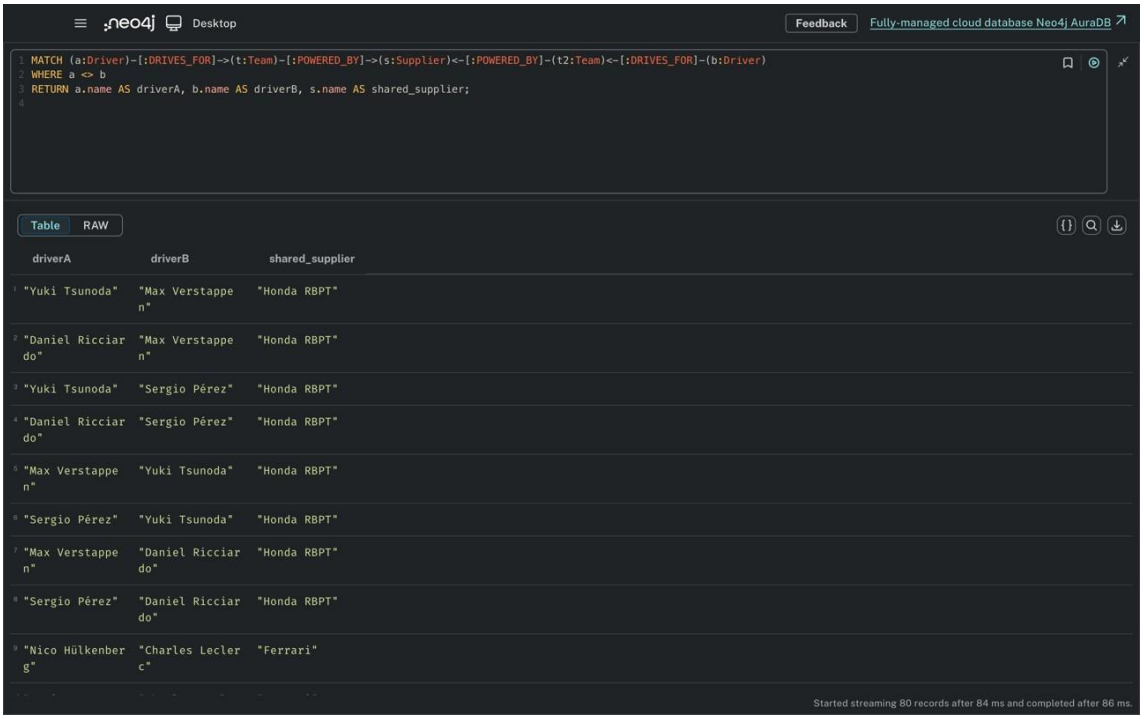
Degree



Mutual supplier between two teams

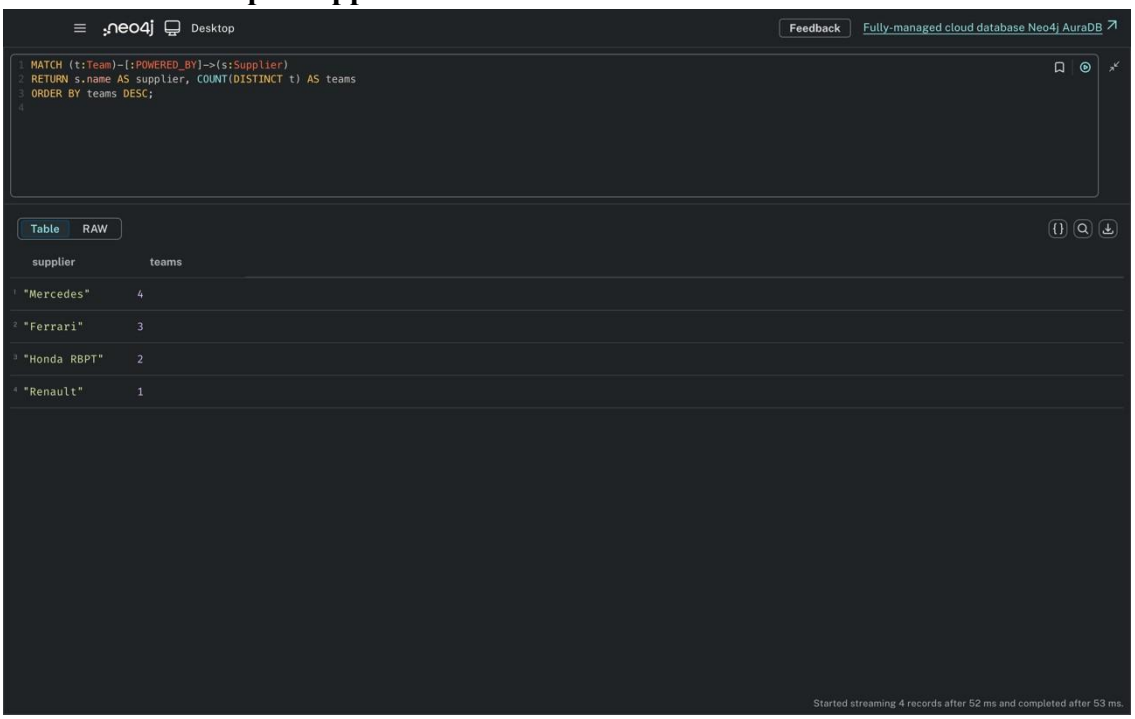


Triangles (3-cycles)

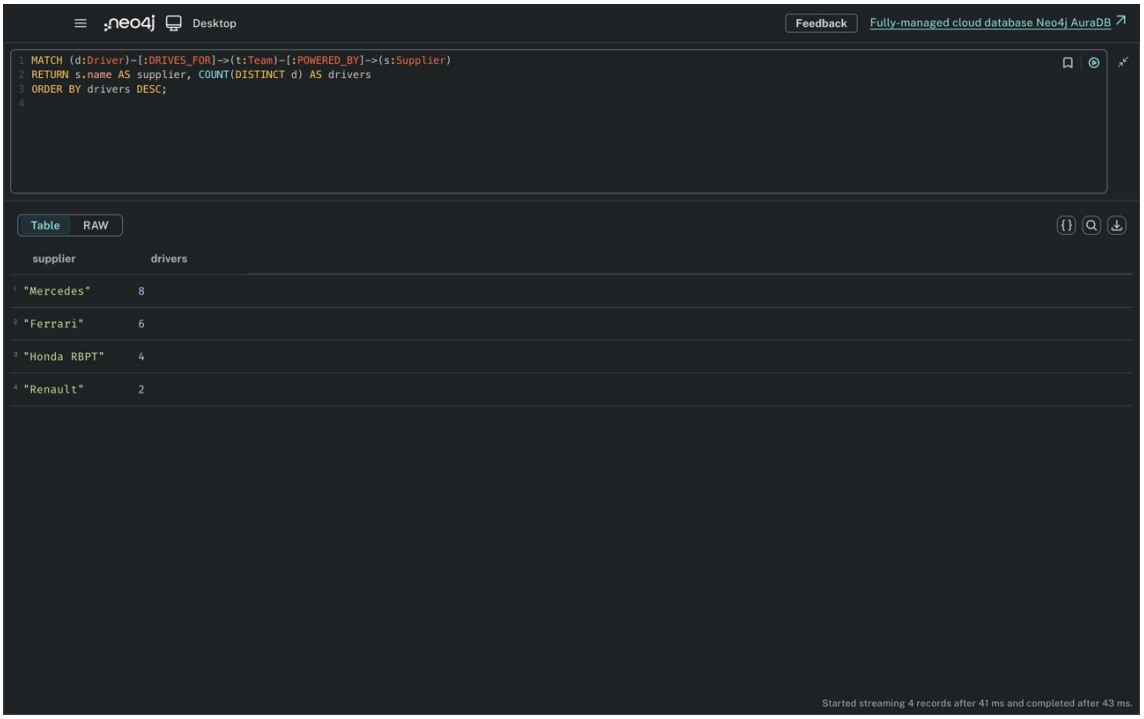


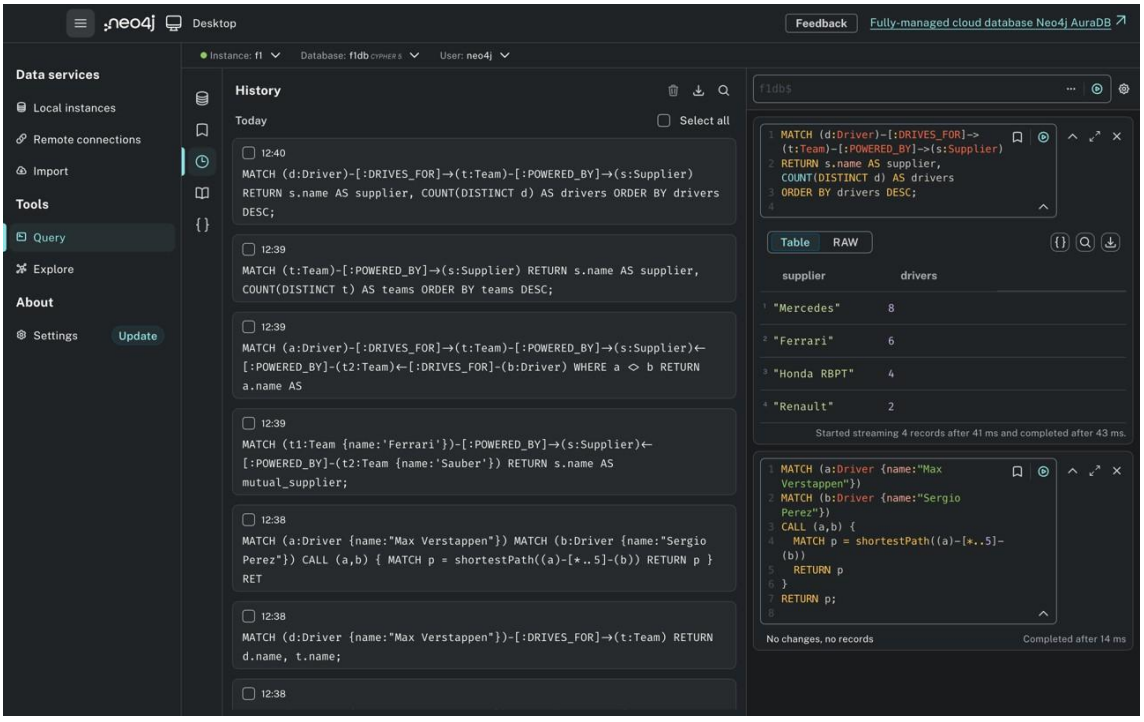
Filtering, aggregation, and reporting

Number of teams per supplier



Number of drivers per supplier





Date: _____ **Signature of faculty in-charge**

Post Lab Descriptive Questions:

1. How is data representation in a graph database different from a relational database? In what scenarios would a graph database be more efficient?

- In a **relational DB**, data is stored in **tables with rows and columns**, and relationships are modeled indirectly using foreign keys and joins.

- In a **graph DB**, data is stored as **nodes (entities) and relationships (edges)**, which directly represent real-world connections.

- **Graph DBs are more efficient** in scenarios with **highly connected data** (e.g., social networks, fraud detection, supply chains, recommendation systems) where traversing relationships is frequent and complex.

2. What does node degree signify? How did you compute it in your dataset?

- **Node degree** = number of relationships connected to a node.

- In the F1 dataset:

- Drivers have degree 1 (they link to their team).
- Teams have degree 3 (2 drivers + 1 supplier).
- Suppliers have higher degree (connected to several teams).

MATCH (n)

RETURN labels(n), coalesce(n.name,n.code), COUNT { (n)--() } AS degree;

3. What insights can shortest path queries provide in real-world scenarios?

- They show **how entities are connected with minimal steps**.

- In real-world:

  - Find the chain of acquaintances in a social network.
  - Discover fraud rings in finance.
  - In F1: path from one driver to another shows the supplier or team connections that link them.

4. What did the graph visualization in Neo4j Browser help you understand that a tabular result did not? Give at least one observation or insight you discovered in your dataset that would not have been obvious in a traditional database table.

- Visualization shows **clusters, hubs, and relationships** at a glance.

- Example from the F1 dataset: it was immediately clear that **Mercedes engines power multiple teams**, making the **supplier node a hub** something less obvious if you only looked at tables of team–supplier mappings.