

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=LectureAcademy

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

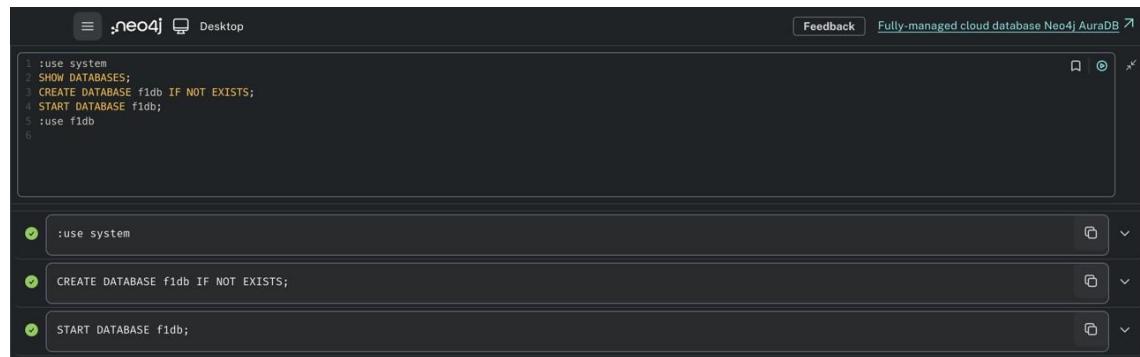
- In Neo4j Browser, re-run:
- MATCH (p:Person)-[r:KNOWS]->(q:Person) RETURN p,r,q;
- Switch to **Graph** view; drag nodes, examine properties, hover edges to see since.

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



```

1 :use system
2 SHOW DATABASES;
3 CREATE DATABASE f1db IF NOT EXISTS;
4 START DATABASE f1db;
5 :use f1db
6

```

The screenshot shows the Neo4j Desktop interface. In the top right corner, it says "Fully-managed cloud database Neo4j AuraDB". The main area displays the following Cypher code:

```

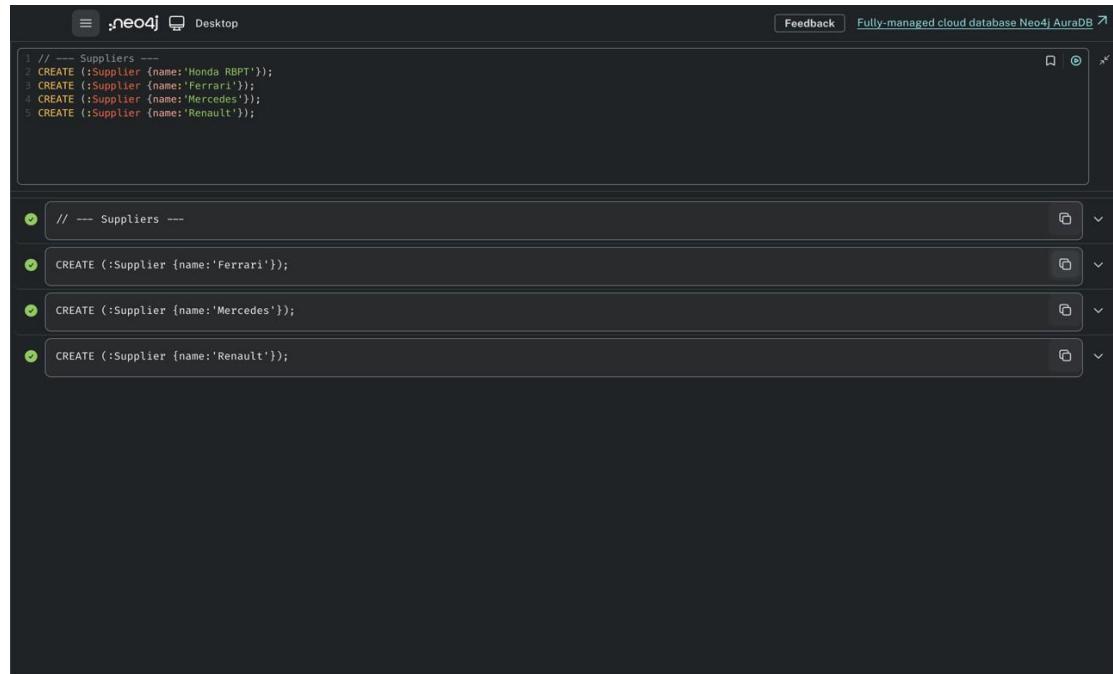
1 :use system
2 SHOW DATABASES;
3 CREATE DATABASE f1db IF NOT EXISTS;
4 START DATABASE f1db;
5 :use f1db
6

```

Below the code, three individual steps are highlighted with green checkmarks:

- :use system
- CREATE DATABASE f1db IF NOT EXISTS;
- START DATABASE f1db;

Node Creation- Supplier



```

1 // --- Suppliers ---
2 CREATE (:Supplier {name:'Honda RBPT'});
3 CREATE (:Supplier {name:'Ferrari'});
4 CREATE (:Supplier {name:'Mercedes'});
5 CREATE (:Supplier {name:'Renault'});

```

The screenshot shows the Neo4j Desktop interface. In the top right corner, it says "Fully-managed cloud database Neo4j AuraDB". The main area displays the following Cypher code:

```

1 // --- Suppliers ---
2 CREATE (:Supplier {name:'Honda RBPT'});
3 CREATE (:Supplier {name:'Ferrari'});
4 CREATE (:Supplier {name:'Mercedes'});
5 CREATE (:Supplier {name:'Renault'});

```

Below the code, four individual steps are highlighted with green checkmarks:

- // --- Suppliers ---
- CREATE (:Supplier {name:'Ferrari'});
- CREATE (:Supplier {name:'Mercedes'});
- CREATE (:Supplier {name:'Renault'});



Team

The screenshot shows the Neo4j browser interface with a query editor at the top containing the following Cypher code:

```
1 // --- Teams ---
2 CREATE (:Team {name:'Red Bull Racing'});
3 CREATE (:Team {name:'Ferrari'});
4 CREATE (:Team {name:'Mercedes'});
5 CREATE (:Team {name:'McLaren'});
6 CREATE (:Team {name:'Aston Martin'});
7 CREATE (:Team {name:'Alpine'});
8 CREATE (:Team {name:'Williams'});
9 CREATE (:Team {name:'RB'});
10 CREATE (:Team {name:'Haas'});
```

Below the code, ten items are listed, each with a green checkmark indicating success:

- 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:'Saufer'})

Driver

The screenshot shows the Neo4j browser interface with a query editor at the top containing the same Cypher code as the previous screenshot:

```
1 // --- Teams ---
2 CREATE (:Team {name:'Red Bull Racing'});
3 CREATE (:Team {name:'Ferrari'});
4 CREATE (:Team {name:'Mercedes'});
5 CREATE (:Team {name:'McLaren'});
6 CREATE (:Team {name:'Aston Martin'});
7 CREATE (:Team {name:'Alpine'});
8 CREATE (:Team {name:'Williams'});
9 CREATE (:Team {name:'RB'});
10 CREATE (:Team {name:'Haas'});
```

Below the code, ten items are listed, each with a green checkmark indicating success:

- 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:'Saufer'})



Relationships Driver → Team

The screenshot shows the Neo4j Desktop interface with a query editor and a results pane. The query editor contains the following Cypher code:

```
1 // --- Relationships (Driver -> Team) ---
2 MATCH (rb:Team {name:'Red Bull Racing'}), (ver:Driver {name:'Max Verstappen'}), (per:Driver {name:'Sergio Pérez'})
3 MERGE (ver)-[:DRIVES_FOR]->(rb)
4 MERGE (per)-[:DRIVES_FOR]->(rb);
5
6 MATCH (fe:Team {name:'Ferrari'}), (lec:Driver {name:'Charles Leclerc'}), (sai:Driver {name:'Carlos Sainz'})
7 MERGE (lec)-[:DRIVES_FOR]->(fe)
8 MERGE (sai)-[:DRIVES_FOR]->(fe);
9
10 MATCH (me:Team {name:'Mercedes'}), (ham:Driver {name:'Lewis Hamilton'}), (rus:Driver {name:'George Russell'})
11
12 // --- Relationships (Driver → Team) ---
13
14 MATCH (fe:Team {name:'Ferrari'}), (lec:Driver {name:'Charles Leclerc'}), (sai:Driver {name:'Carlos Sainz'})
15
16 MATCH (me:Team {name:'Mercedes'}), (ham:Driver {name:'Lewis Hamilton'}), (rus:Driver {name:'George Russell'})
17
18 MATCH (mc:Team {name:'McLaren'}), (nor:Driver {name:'Lando Norris'}), (pia:Driver {name:'Oscar Piastri'})
19
20 MATCH (am:Team {name:'Aston Martin'}), (alo:Driver {name:'Fernando Alonso'}), (str:Driver {name:'Lance Stroll'})
21
22 MATCH (al:Team {name:'Alpine'}), (gas:Driver {name:'Pierre Gasly'}), (oco:Driver {name:'Esteban Ocon'})
23
24 MATCH (wi:Team {name:'Williams'}), (alb:Driver {name:'Alex Albon'}), (sar:Driver {name:'Logan Sargeant'})
25
26 MATCH (rb2:Team {name:'RB'}), (tsu:Driver {name:'Yuki Tsunoda'}), (ric:Driver {name:'Daniel Ricciardo'})
27
28 MATCH (hs:Team {name:'Haas'}), (hul:Driver {name:'Nico Hülkenberg'}), (mag:Driver {name:'Kevin Magnussen'})
29
30 MATCH (sa:Team {name:'Sauber'}), (bot:Driver {name:'Valtteri Bottas'}), (zho:Driver {name:'Zhou Guanyu'})
```

The results pane shows 30 successful relationship creation operations, each with a green checkmark and a copy icon.

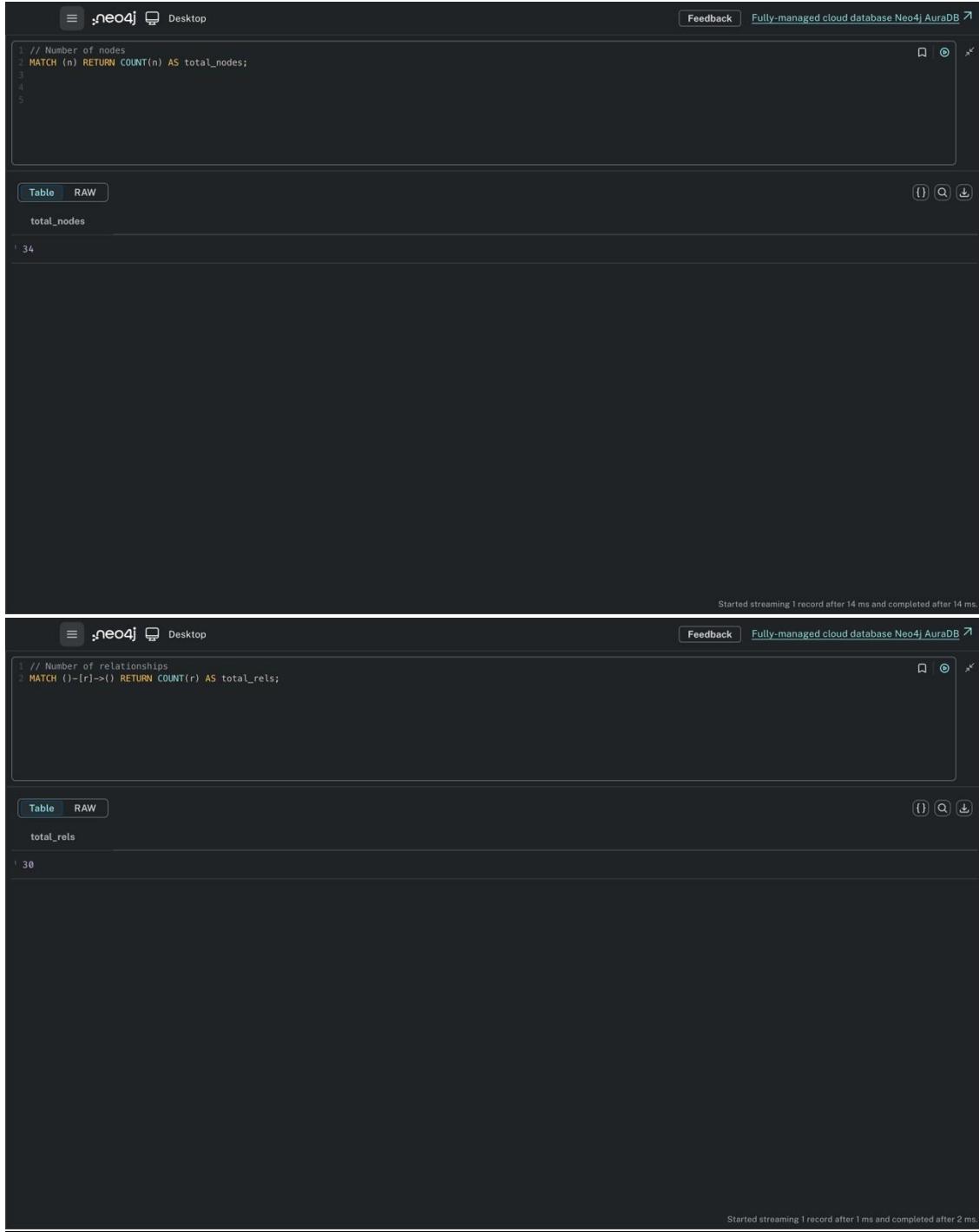
Team → Supplier

The screenshot shows the Neo4j Desktop interface with a query editor and a results pane. The query editor contains the following Cypher code:

```
1 // --- Relationships (Team -> Supplier) ---
2 MATCH (rb:Team {name:'Red Bull Racing'}), (sup:Supplier {name:'Honda RBPT'}) MERGE (rb)-[:POWERED_BY]->(sup);
3 MATCH (rb2:Team {name:'RB'}), (sup:Supplier {name:'Honda RBPT'}) MERGE (rb2)-[:POWERED_BY]->(sup);
4
5 MATCH (fe:Team {name:'Ferrari'}), (sup:Supplier {name:'Ferrari'}) MERGE (fe)-[:POWERED_BY]->(sup);
6 MATCH (hs:Team {name:'Haas'}), (sup:Supplier {name:'Ferrari'}) MERGE (hs)-[:POWERED_BY]->(sup);
7 MATCH (sa:Team {name:'Sauber'}), (sup:Supplier {name:'Ferrari'}) MERGE (sa)-[:POWERED_BY]->(sup);
8
9 MATCH (me:Team {name:'Mercedes'}), (sup:Supplier {name:'Mercedes'}) MERGE (me)-[:POWERED_BY]->(sup);
10 MATCH (mc:Team {name:'McLaren'}), (sup:Supplier {name:'Mercedes'}) MERGE (mc)-[:POWERED_BY]->(sup);
11
12 // --- Relationships (Team → Supplier) ---
13
14 MATCH (rb2:Team {name:'RB'}), (sup:Supplier {name:'Honda RBPT'}) MERGE (rb2)-[:POWERED_BY]->(sup);
15
16 MATCH (fe:Team {name:'Ferrari'}), (sup:Supplier {name:'Ferrari'}) MERGE (fe)-[:POWERED_BY]->(sup);
17
18 MATCH (hs:Team {name:'Haas'}), (sup:Supplier {name:'Ferrari'}) MERGE (hs)-[:POWERED_BY]->(sup);
19
20 MATCH (sa:Team {name:'Sauber'}), (sup:Supplier {name:'Ferrari'}) MERGE (sa)-[:POWERED_BY]->(sup);
21
22 MATCH (me:Team {name:'Mercedes'}), (sup:Supplier {name:'Mercedes'}) MERGE (me)-[:POWERED_BY]->(sup);
23
24 MATCH (mc:Team {name:'McLaren'}), (sup:Supplier {name:'Mercedes'}) MERGE (mc)-[:POWERED_BY]->(sup);
25
26 MATCH (am:Team {name:'Aston Martin'}), (sup:Supplier {name:'Mercedes'}) MERGE (am)-[:POWERED_BY]->(sup);
27
28 MATCH (wi:Team {name:'Williams'}), (sup:Supplier {name:'Mercedes'}) MERGE (wi)-[:POWERED_BY]->(sup);
29
30 MATCH (al:Team {name:'Alpine'}), (sup:Supplier {name:'Renault'}) MERGE (al)-[:POWERED_BY]->(sup);
```

The results pane shows 30 successful relationship creation operations, each with a green checkmark and a copy icon.

Counts



```
// Number of nodes
MATCH (n) RETURN COUNT(n) AS total_nodes;
```

total_nodes

34

Started streaming 1 record after 14 ms and completed after 14 ms.

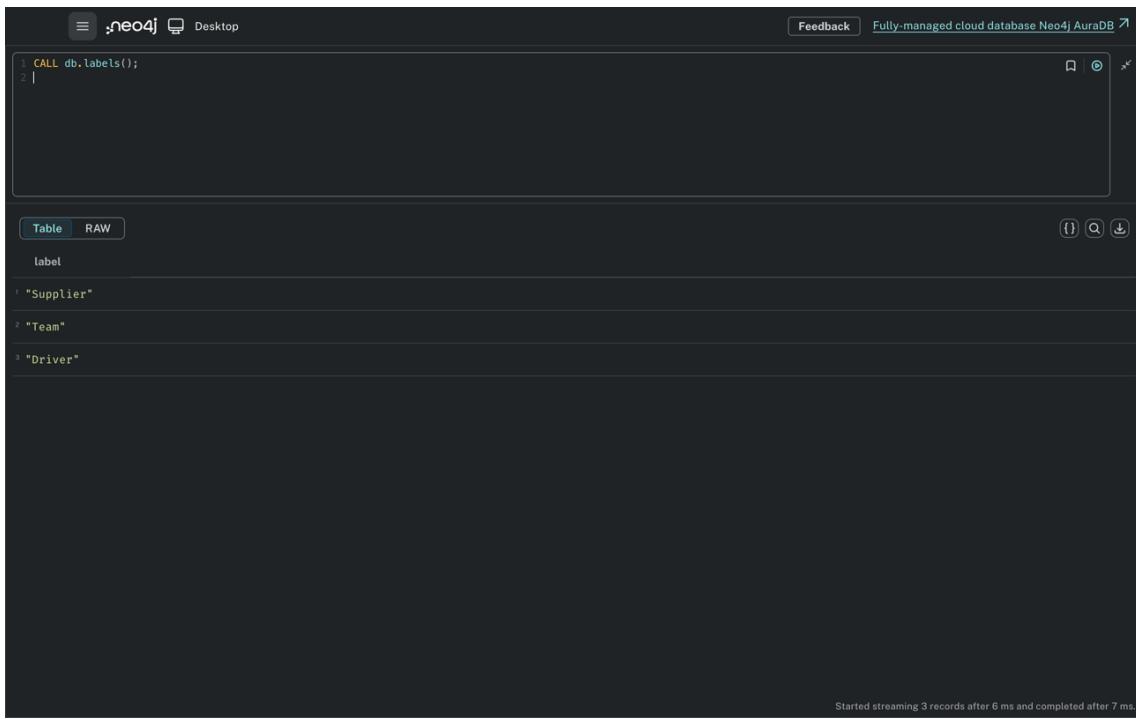
```
// Number of relationships
MATCH ()-[r]->() RETURN COUNT(r) AS total_rels;
```

total_rels

30

Started streaming 1 record after 1 ms and completed after 2 ms.

Labels



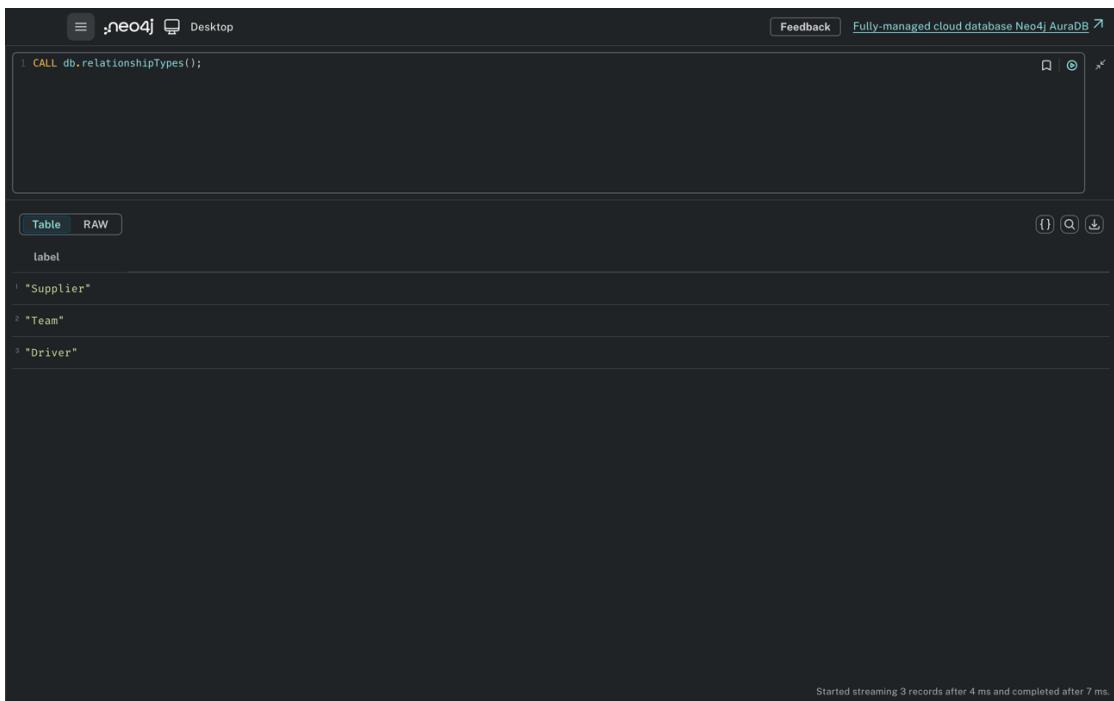
The screenshot shows the Neo4j Desktop interface with a dark theme. In the top-left corner, there's a small icon of a computer monitor with the text ':neo4j'. To its right is a 'Desktop' button. On the far right of the header bar are 'Feedback', 'Fully-managed cloud database Neo4j AuraDB', and a user profile icon. Below the header is a code editor window containing the following Cypher query:

```
1 CALL db.labels();  
2 |
```

Below the code editor is a table with two tabs: 'Table' (which is selected) and 'RAW'. The table has one column labeled 'label'. The data returned by the query is:

label
1 "Supplier"
2 "Team"
3 "Driver"

In the bottom right corner of the table area, there are three small icons: a copy symbol, a search symbol, and a download symbol. At the very bottom of the interface, a status message reads: 'Started streaming 3 records after 6 ms and completed after 7 ms.'



This screenshot is identical to the one above it, showing the Neo4j Desktop interface with a dark theme. It features the ':neo4j' icon, a 'Desktop' button, and a header with 'Feedback', 'Fully-managed cloud database Neo4j AuraDB', and a user profile icon. The code editor window contains the same Cypher query:

```
1 CALL db.relationshipTypes();  
2 |
```

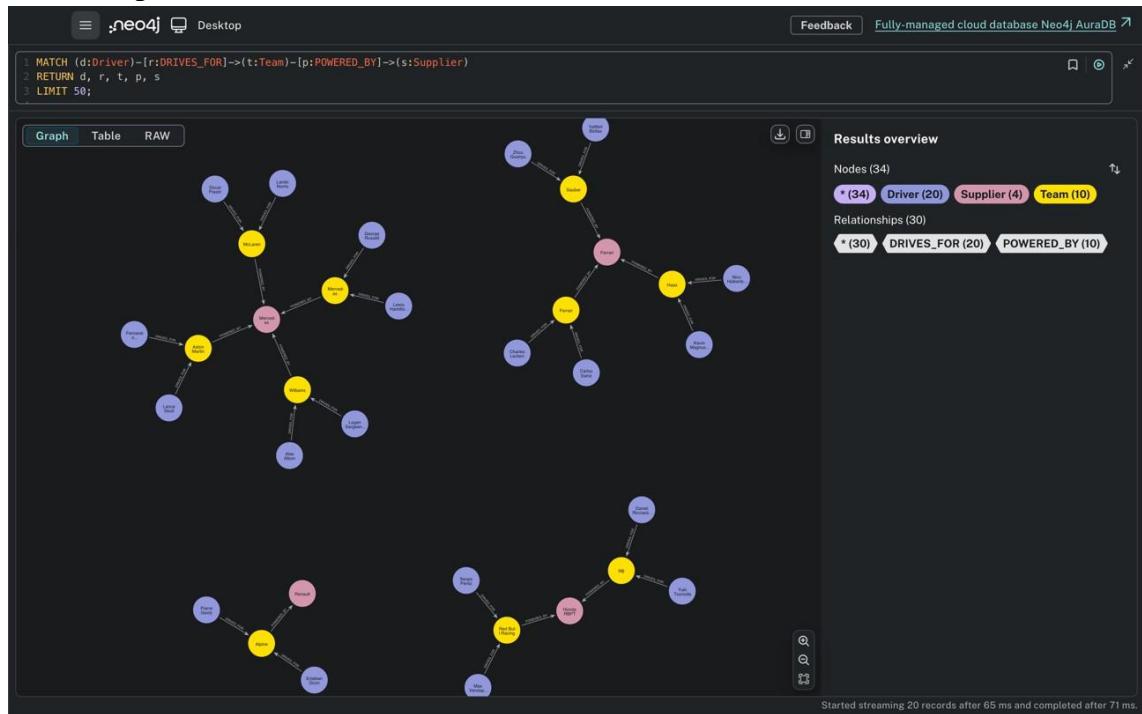
The table below shows the relationship types:

label
1 "Supplier"
2 "Team"
3 "Driver"

The status message at the bottom right indicates: 'Started streaming 3 records after 4 ms and completed after 7 ms.'

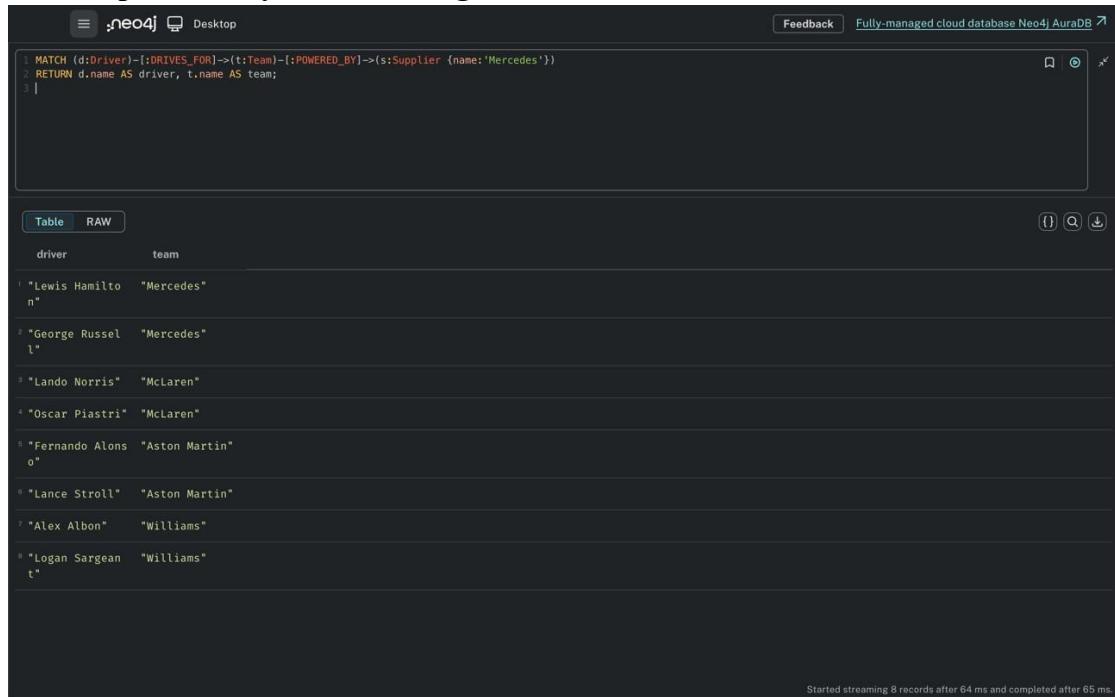


Peek Graph



Pattern Queries

Drivers powered by Mercedes engines



Drivers with the same supplier (teammates across different teams)

neo4j Desktop Feedback Fully-managed cloud database Neo4j AuraDB

```

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

Table RAW

driver1	driver2	common_supplier
"Zhou Guanyu"	"Nico Hülkenberg"	"Ferrari"
"Daniel Ricciardo"	"Max Verstappen"	"Honda RBPT"
"Daniel Ricciardo"	"Sergio Pérez"	"Honda RBPT"
"Max Verstappen"	"Daniel Ricciardo"	"Honda RBPT"
"Max Verstappen"	"Yuki Tsunoda"	"Honda RBPT"
"Sergio Pérez"	"Daniel Ricciardo"	"Honda RBPT"
"Sergio Pérez"	"Yuki Tsunoda"	"Honda RBPT"
"Yuki Tsunoda"	"Max Verstappen"	"Honda RBPT"
"Yuki Tsunoda"	"Sergio Pérez"	"Honda RBPT"
"Alex Albon"	"Fernando Alonso"	"Mercedes"
"Alex Albon"	"George Russell"	"Mercedes"

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

Drivers in the same team (teammates)

neo4j Desktop Feedback Fully-managed cloud database Neo4j AuraDB

```

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

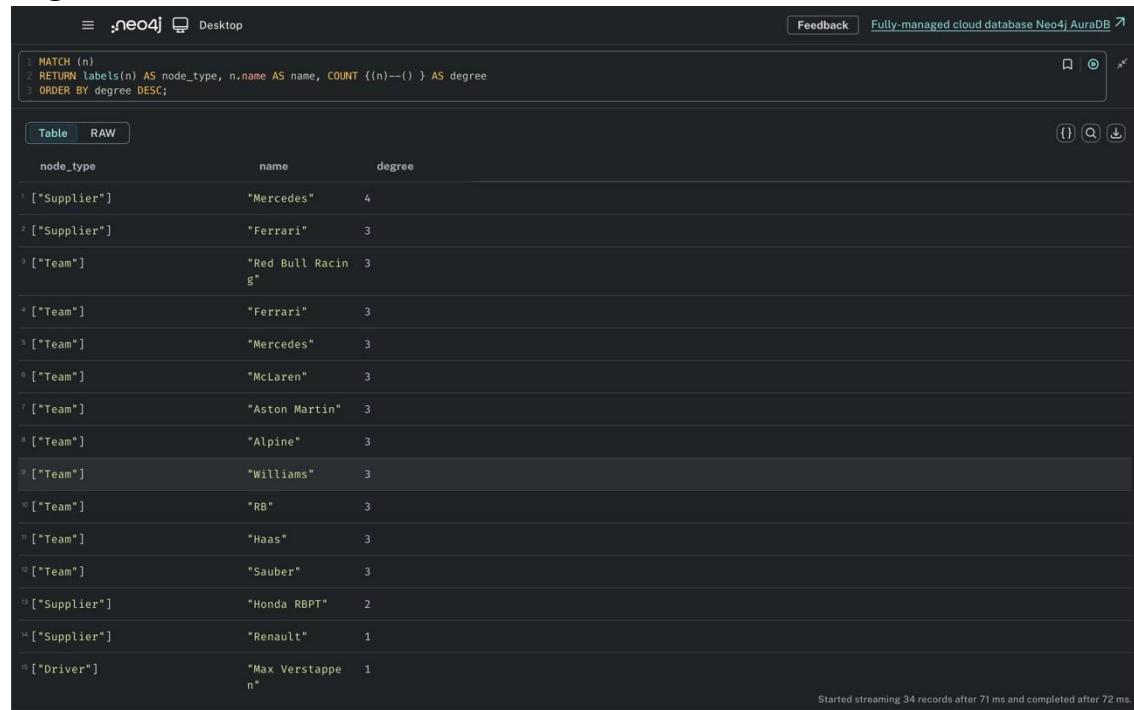
Table RAW

team	driverA	driverB
"Alpine"	"Esteban Ocon"	"Pierre Gasly"
"Alpine"	"Pierre Gasly"	"Esteban Ocon"
"Aston Martin"	"Lance Stroll"	"Fernando Alonso"
"Aston Martin"	"Fernando Alonso"	"Lance Stroll"
"Ferrari"	"Carlos Sainz"	"Charles Leclerc"
"Ferrari"	"Charles Leclerc"	"Carlos Sainz"
"Haas"	"Kevin Magnussen"	"Nico Hülkenberg"
"Haas"	"Nico Hülkenberg"	"Kevin Magnussen"
"McLaren"	"Oscar Piastri"	"Lando Norris"
"McLaren"	"Lando Norris"	"Oscar Piastri"
"Mercedes"	"George Russell"	"Lewis Hamilton"
"Mercedes"	"Lewis Hamilton"	"George Russell"

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

Basic Analysis

Degree



```

1 MATCH (n)
2 RETURN labels(n) AS node_type, n.name AS name, COUNT {{n}--()} AS degree
3 ORDER BY degree DESC;

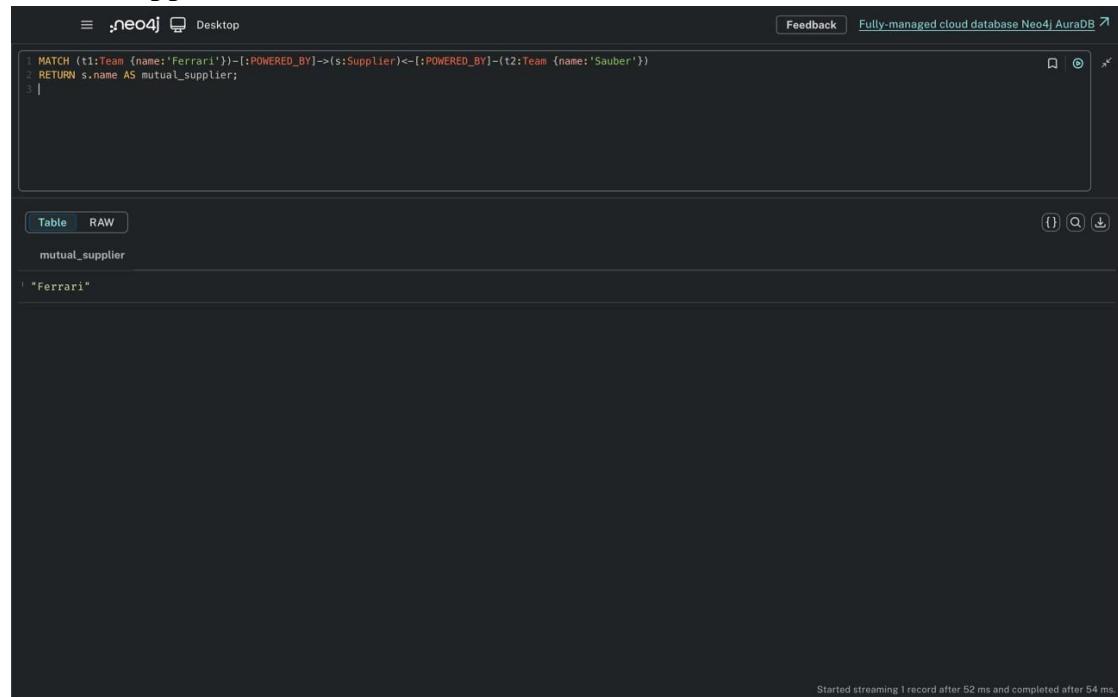
```

Table RAW

node_type	name	degree
["Supplier"]	"Mercedes"	4
["Supplier"]	"Ferrari"	3
["Team"]	"Red Bull Racing"	3
["Team"]	"Ferrari"	3
["Team"]	"Mercedes"	3
["Team"]	"McLaren"	3
["Team"]	"Aston Martin"	3
["Team"]	"Alpine"	3
["Team"]	"Williams"	3
["Team"]	"RB"	3
["Team"]	"Haas"	3
["Team"]	"Sauber"	3
["Supplier"]	"Honda RBPT"	2
["Supplier"]	"Renault"	1
["Driver"]	"Max Verstappen"	1

Started streaming 34 records after 71 ms and completed after 72 ms.

Mutual supplier between two teams



```

1 MATCH (t1:Team {name:'Ferrari'})-[:POWERED_BY]->(s:Supplier)<-[{:POWERED_BY}]- (t2:Team {name:'Sauber'})
2 RETURN s.name AS mutual_supplier;
3

```

Table RAW

mutual_supplier
"Ferrari"

Started streaming 1 record after 52 ms and completed after 54 ms.

Triangles (3-cycles)

≡ :neo4j 🖥 Desktop Feedback Fully-managed cloud database Neo4j AuraDB ↗

```

1 MATCH (a:Driver)-[:DRIVES_FOR]-->(t:Team)-[:POWERED_BY]->(s:Supplier)<--[:POWERED_BY]-(t2:Team)<--[:DRIVES_FOR]-(b:Driver)
2 WHERE a <> b
3 RETURN a.name AS driverA, b.name AS driverB, s.name AS shared_supplier;
4

```

Table RAW

driverA	driverB	shared_supplier
1 "Yuki Tsunoda"	"Max Verstappen"	"Honda RBPT"
2 "Daniel Ricciardo"	"Max Verstappen"	"Honda RBPT"
3 "Yuki Tsunoda"	"Sergio Pérez"	"Honda RBPT"
4 "Daniel Ricciardo"	"Sergio Pérez"	"Honda RBPT"
5 "Max Verstappen"	"Yuki Tsunoda"	"Honda RBPT"
6 "Sergio Pérez"	"Yuki Tsunoda"	"Honda RBPT"
7 "Max Verstappen"	"Daniel Ricciardo"	"Honda RBPT"
8 "Sergio Pérez"	"Daniel Ricciardo"	"Honda RBPT"
9 "Nico Hülkenberg"	"Charles Leclerc"	"Ferrari"

Started streaming 80 records after 84 ms and completed after 86 ms.

Filtering, aggregation, and reporting

Number of teams per supplier

≡ :neo4j 🖥 Desktop Feedback Fully-managed cloud database Neo4j AuraDB ↗

```

1 MATCH (t:Team)-[:POWERED_BY]->(s:Supplier)
2 RETURN s.name AS supplier, COUNT(DISTINCT t) AS teams
3 ORDER BY teams DESC;
4

```

Table RAW

supplier	teams
1 "Mercedes"	4
2 "Ferrari"	3
3 "Honda RBPT"	2
4 "Renault"	1

Started streaming 4 records after 52 ms and completed after 53 ms.



Number of drivers per supplier

neo4j Desktop

```
1 MATCH (d:Driver)-[:DRIVES_FOR]-(t:Team)-[:POWERED_BY]-(s:Supplier)
2 RETURN s.name AS supplier, COUNT(DISTINCT d) AS drivers
3 ORDER BY drivers DESC;
4
```

Feedback Fully-managed cloud database Neo4j AuraDB

Table RAW

supplier	drivers
1 "Mercedes"	8
2 "Ferrari"	6
3 "Honda RBPT"	4
4 "Renault"	2

{ } { } { } { }

Started streaming 4 records after 41 ms and completed after 43 ms.

neo4j Desktop

History

Today

12:40

```
MATCH (d:Driver)-[:DRIVES_FOR]-(t:Team)-[:POWERED_BY]-(s:Supplier)
RETURN s.name AS supplier, COUNT(DISTINCT d) AS drivers ORDER BY drivers DESC;
```

12:39

```
MATCH (t:Team)-[:POWERED_BY]-(s:Supplier) RETURN s.name AS supplier,
COUNT(DISTINCT t) AS teams ORDER BY teams DESC;
```

12:39

```
MATCH (a:Driver)-[:DRIVES_FOR]-(t:Team)-[:POWERED_BY]-(s:Supplier)<-
[:POWERED_BY]-(t2:Team)-[:DRIVES_FOR]-(b:Driver) WHERE a <-> b RETURN
a.name AS
```

12:39

```
MATCH (t1:Team {name:'Ferrari'})-[:POWERED_BY]-(s:Supplier)<-
[:POWERED_BY]-(t2:Team {name:'Saufer'}) RETURN s.name AS
mutual_supplier;
```

12:38

```
MATCH (a:Driver {name:"Max Verstappen"}) MATCH (b:Driver {name:"Sergio Perez"})
CALL (a,b) { MATCH p = shortestPath((a)-[*..5]-(b)) RETURN p }
RET
```

12:38

```
MATCH (d:Driver {name:"Max Verstappen"})-[:DRIVES_FOR]-(t:Team) RETURN
d.name, t.name;
```

12:38

```
MATCH (d:Driver {name:"Max Verstappen"})-[:DRIVES_FOR]-(t:Team) RETURN
d.name, t.name;
```

History

flfdb\$

```
1 MATCH (d:Driver)-[:DRIVES_FOR]->(
:t:Team)-[:POWERED_BY]->(s:Supplier)
2 RETURN s.name AS supplier,
COUNT(DISTINCT d) AS drivers
3 ORDER BY drivers DESC;
4
```

Table RAW

supplier	drivers
1 "Mercedes"	8
2 "Ferrari"	6
3 "Honda RBPT"	4
4 "Renault"	2

{ } { } { } { }

Started streaming 4 records after 41 ms and completed after 43 ms.

```
1 MATCH (a:Driver {name:"Max Verstappen"})
2 MATCH (b:Driver {name:"Sergio Perez"})
3 CALL (a,b) {
4   MATCH p = shortestPath((a)-[*..5]-
(b))
5   RETURN p
6 }
7 RETURN p;
8
```

No changes, no records

Completed after 14 ms

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).
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.