# Leveraging Graph Data Science with Neo4j

## Advanced Insights and Applications

Ali Balaj

FZ Business Informatics

December 2, 2024

# Table of Contents

# Cypher Code Guidance: Querying Data

Listing 1: Query Examples

```
// Create an empty node:
CREATE ()
// Create an node with label:
CREAT (:Student)

// Create a node with multiple labels:
CREATE (:Sudent:Lecturer)

// Filter nodes with specific label:
MATCH (node:Student) retrun node

// Create a node with property
CREATE (node:Student{field:"Mechatronics"}) RETURN node

// Create a node with multiple labels and multiple
    properties
create (node:Teacher:Student{name "Jens Schuhmacher",
    birth_year: 1987}) return node
```

# Cypher Code Guidance: Querying Data

Listing 2: Query Examples

```
// Filter with AND including property
match (node)
where node.name= "Benjamin" AND node.birth_year= 1988
return node

// Or Operator
match (node)
where node.name= "Benjamin" OR  node.birth_year= 1950
return node

// IN Operator
match (node)
where node.birth_year in [1988,1950]
return node

// ID
match (node:Student)
where ID(node)=3
return node
```

# Cypher Code Guidance: Querying Data

Listing 3: Query Examples

```
// Update and add properties
match (node:Student)
where id(node)=4
set node.name="Amna Shahbaz", node.birth_year= 1993, node.
    country= "Pakistan"
return node


// Adding another label to a labeled node
match (node:Student)
where ID(node)= 4
set node:Resercher
return node
```

# Cypher Code Guidance: Querying Data

Listing 4: Query Examples

```
// Remove Label
match (node:Student)
where ID(node)=4
remove node:Researcher
return node


// Remove Property
match (node:Student)
where ID(node)=4
remove node.name, node:Researcher
return node
```

# Cypher Code Guidance: Querying Data

Listing 5: Query Examples

```
// Remove node with has no relationship!
match (node:Researcher)
where ID(node)=4
delete node

//Remove all nodes with Student label
match (node:Student)
delete node
```

# Cypher Code Guidance: Querying Data

Listing 6: Query Examples

```
create (node1:Student{name:"Ali Balaj"}),(node2:Lecturer{
    name:"Ralph Hoch"})
create (node1)-[:Is_tought_by]->(node2)
return node1,node2

//creating nodes and relationship
create (node1:Student{name:"Ali Balaj"}),(node2:Student{name
    :"Amna Shahbaz"})
create (node1)-[:Is_teaching{type:"par-time"}]->(node2)
return node1, node2

// Creating the realtionship between 2 nodes:
match (node:Student),(node2:Student)
where ID(node)=2 and ID(node2)=3
create (node)<-[:likes{type:"very much"}]-(node2)
return node,node2
```

# Cypher Code Guidance: Querying Data

Listing 7: Query Examples

```
// using merge to avoid duplicate relationship
match (node1:Boy{name:"Christoph"}),(node2:Girl{name:"Anna
    "}),(node3:Girl{name:"Maria"})
create (node2)-[:Loves{type:"very_much"}]->(node1)<-[:likes{
    type:"a bit"}]-(node3)
return node1,node2,node3

//Any direction any relationship
match(node:Girl)-[]-(nodes)
where ID(node)=2
return node, nodes

// Or in property:
match (node)<-[:Loves | likes]-(n)
where node.name="Christoph"
return node,n
```

# Cypher Code Guidance: Querying Data

Listing 8: Query Examples

```
// Retrun relationship and connect result node option
match (node)<-[rel:Loves]-(n)
where node.name="Christoph"
return node,n,rel

// update the property of the relationship
CREATE (node:Student {name: "Ali"})-[:is_learning {progress:
    "good"}]->(n1:Subject {name: "Cypher"})
CREATE (node)-[:is_learning {progress: "excellent"}]->(n2:
    Subject {name: "Python"})
RETURN node, n1, n2;

MATCH (node:Student {name: "Ali"})-[rel:is_learning]->(n:
    Subject {name: "Cypher"})
SET rel.progress = "excellent", rel.teaches = "good"
RETURN node, rel, n;
```

# Cypher Code Guidance: Querying Data

Listing 9: Query Examples

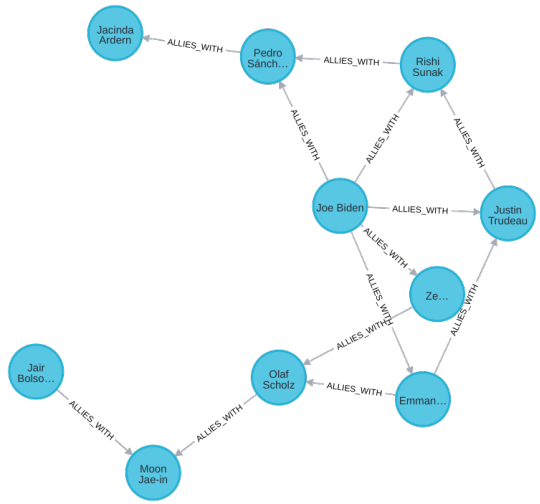
```
//remove the property of the relationship
MATCH (node:Student {name: "Ali"})-[rel:is_learning]->(n:
    Subject {name: "Cypher"})
remove rel.teaches
RETURN node, rel, n

// delete a relationship
match (node)-[rel:is_learning]->(s)
where node.name="Ali" and s.name="Python"
delete rel
return node,s
```

# Creating Data with Cypher

```
CREATE (biden:President {name: 'Joe Biden', country: 'United States'}),
       (macron:President {name: 'Emmanuel Macron', country: 'France'}),
       (zelenskyy:President {name: 'Volodymyr Zelenskyy', country: 'Ukraine'}),
       (trudeau:President {name: 'Justin Trudeau', country: 'Canada'}),
       (sunak:President {name: 'Rishi Sunak', country: 'United Kingdom'}),
       (sanchez:President {name: 'Pedro Sanchez', country: 'Spain'}),
       (scholz:President {name: 'Olaf Scholz', country: 'Germany'}),
       (ardern:President {name: 'Jacinda Ardern', country: 'New Zealand'}),
       (bolsonaro:President {name: 'Jair Bolsonaro', country: 'Brazil'}),
       (moon:President {name: 'Moon Jae-in', country: 'South Korea'}),

       (biden)-[:ALLIES_WITH]->(macron),
       (biden)-[:ALLIES_WITH]->(zelenskyy),
       (biden)-[:ALLIES_WITH]->(trudeau),
       (biden)-[:ALLIES_WITH]->(sunak),
       (biden)-[:ALLIES_WITH]->(sanchez),
       (macron)-[:ALLIES_WITH]->(scholz),
       (macron)-[:ALLIES_WITH]->(trudeau),
       (trudeau)-[:ALLIES_WITH]->(sunak),
       (sunak)-[:ALLIES_WITH]->(sanchez),
       (zelenskyy)-[:ALLIES_WITH]->(scholz),
       (scholz)-[:ALLIES_WITH]->(moon),
       (sanchez)-[:ALLIES_WITH]->(ardern),
       (bolsonaro)-[:ALLIES_WITH]->(moon);
```



# Cypher Code Guidance: Data Preparation

Listing 10: Data Loading and Relationship Creation

```
// 1. Remove all nodes and relationships
MATCH (node)
DETACH DELETE node;

// 2. Load data from a CSV file
LOAD CSV WITH HEADERS FROM 'file:///worldcities.csv' AS row
CREATE (:City {name: row.city, lat: toFloat(row.lat), lng:
    toFloat(row.lng),
                country: row.country, admin: row.admin_name,
                population: toInteger(row.population)});

// 3. Create relationships between cities and countries
MATCH (c:City)
MERGE (co:Country {name: c.country})
MERGE (c)-[:LOCATED_IN]->(co);
```

# Cypher Code Guidance: Querying Data

1. Query the cities in Germany (Table): Finds all cities located in Germany and returns their names and populations.
2. Query the cities in Austria (Graph): Finds the country "Austria" and retrieves all cities connected to it, returning both the country and city information.

Listing 11: Query Examples

```
// 4. Query cities in Germany (Table format)
MATCH (c:City)-[:LOCATED_IN]->(co:Country {name: 'Germany'})
RETURN c.name, c.population;

// 5. Query cities in Austria (Graph format)
MATCH (a:Country {name: 'Austria'})<-[:LOCATED_IN]-(c:City)
RETURN a, c;
```

# Questions?

**Thank you!**

Any questions?