Graph & Modern Databases

COMP1835

1

1

Using Cypher in Neo4J

https://neo4j.com/style-guide/

2

2

Objectives

- Introduce more details on CREATE and MERGE commands
- Describe and create constraints
- Discuss indexes
- Introduce different ways to filter graph queries and traverse relationships
- Introduce Cypher functions
- Discuss how to analyse a query

3

3

Part 1

4

4

More on Creation of a Node

- We learned before that using Cypher you can create a node in Neo4j by using **CREATE** clause:

```
CREATE (a:Actor)
```

- Also you can create multiple nodes by separating the nodes specified with commas:

```
CREATE
(:Actor {name: 'Michael Caine', born: 1933}),
(:Actor {name: 'Liam Neeson', born: 1952}),
(:Actor {name: 'Katie Holmes', born: 1978}),
(:Actor {name: 'Benjamin Melniker', born: 1913})
```

- However, if you run this command twice it will create duplicate nodes!

5

5

Duplicates!

- If you use **CREATE** command to:
  - create a node and a node with the same property values exists, a duplicate node is created
  - create a relationship and the relationship exists, a duplicate relationship is created
  - create a label and the label already exists for the node, the node is not updated
  - create a property and the node's or relationship's property already exists, it is updated with the new value
- It is not a good practice to create duplicate nodes or relationships in a graph!

6

6

MERGE clause

- To avoid potential problem with a creation of the duplicates you can use **MERGE** command that
  - either will create new nodes and relationships
  - or will make structural changes to existing nodes and relationships
- **MERGE** command will help to:
  - Create a unique node based on label and key information for a property and if it exists, optionally update it
  - Create a unique relationship
- Example:

```
MERGE (a:Actor {name: 'Michael Caine', born: 1933})
RETURN a
```

  - If there is an existing node with specified label and node properties in the graph, no node is created, but if the node is not found in the graph, then the node is created

7

7

DELETE

- To delete a node or relationship, use the **DELETE** clause:

```
MATCH (n:Person { name: 'UNKNOWN' })
DELETE n
```

```
MATCH (a:Actor)-[rel:ACTED_IN]->(m:Movie)
WHERE a.name = 'Christian Bale' AND m.title = 'Batman Begins'
DELETE rel
RETURN a, m
```

  - However, if you attempt to delete a node in the graph that has relationships in or out of the node, the graph engine will return an error because deleting such a node will leave orphaned relationships in the graph
- To delete a node and its corresponding relationships use **DETACH DELETE:**

```
MATCH (a:Actor)
WHERE a.name = 'Liam Neeson'
DETACH DELETE a
```

It will remove the node and its relationships to any other nodes

8

8

Adding and Removing Labels

- You can add labels to a node after it has been created (or add a second label) by using the **SET** clause:

```
MATCH (m:Movie)
WHERE m.title = 'Batman Begins'
SET m:Action
RETURN labels(m)
```

`labels()` function returns the set of labels for the node

- To remove a label use **REMOVE** clause:

```
MATCH (m:Movie:Action)
WHERE m.title = 'Batman Begins'
REMOVE m:Action
RETURN labels(m)
```

9

9

Adding and Removing Properties

- You can add properties to the node, using the **SET** clause:

```
MATCH (m:Movie)
WHERE m.title = 'Batman Begins'
SET m.released = 2005, m.lengthInMinutes = 140
RETURN m
```

- There are two ways that you can remove a property from a node:
  - To use the **REMOVE** keyword
  - Or to set the property's value to **null**
  - Example:

```
MATCH (m:Movie)
WHERE m.title = 'Batman Begins'
SET m.released= null
REMOVE m.lengthInMinutes
RETURN m
```

10

10

Quiz

- **Question 1: You need to create several nodes in your graph. Which command will you use?**
  - **A.** `CREATE MULTI`
  - **B.** `CREATE`
  - **C.** `MATCH … CREATE`
  - **D.** `CREATE NODES`
- **Question 2: You are creating a relationship with `CREATE` command. If the relationship's property already exists, it will update it with the new value.**
  - **A.** `TRUE`
  - **B.** `FALSE`
- **Question 3: You are using `DETACH DELETE` command on the node. It will remove the node and its relationships to any other nodes.**
  - **A.** `TRUE`
  - **B.** `FALSE`

11

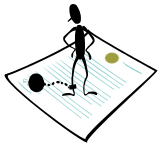
11

Part 2

12

12

Constraints



- In Neo4j, you can use Cypher to:
  - Add a uniqueness constraint that ensures that a value for a property is unique for all nodes of that type
  - Add an existence constraint that ensures that when a node or relationship is created or modified, it must have certain properties set
  - Add a node key that ensures that a set of values for properties of a node of a given type is unique
    - Constraints and node keys that enforce uniqueness are related to indexes
    - Existence constraints and node keys are only available in Enterprise Edition of Neo4j.

13

13

Creating Constraints

- Uniqueness constraint:

```
CREATE CONSTRAINT ON (m:Movie) ASSERT m.title IS UNIQUE
```

  - To ensure that the title for a node of type Movie is unique:
- Existence constraint:

```
CREATE CONSTRAINT ON ()-[rel:REVIEWED]-() ASSERT exists(rel.rating)
```

  - If we attempt to create a *:REVIEWED* relationship without setting the *rating* property we will get an error
- A node key:
  - is used to define the uniqueness constraint for multiple properties of a node of a certain type

```
CREATE CONSTRAINT ON (p:Person) ASSERT (p.name, p.born) IS NODE KEY
```

    - We will not be able to create a node with the same combination of properties *name* and *born*

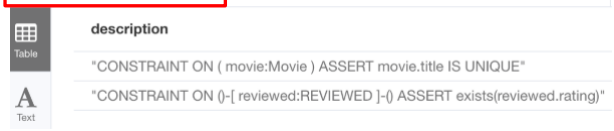
14

14

Retrieving and Deleting Constraints

- To see the set of constraints defined in the graph use:

```
$ CALL db.constraints
```



- To remove constraint, use **DROP CONSTRAINT** command:

```
DROP CONSTRAINT ON ()-[rel:REVIEWED]-()
ASSERT exists(rel.rating)
```

15

15

Indexes

- Neo4J database supports indexes:
  - a redundant copy of some of the data in the database for the purpose of making searches of related data more efficient.
  - However, the same as in RDBMS this comes at the cost of additional storage space and slower writes
  - Once an index has been created, it will be managed and kept up to date by the DBMS
- Neo4J enables the creation of indexes on one or more properties for all nodes that have a given label:
  - An index that is created on a single property for any given label is called a single-property index:

```
CREATE INDEX [index_name] FOR (n:LabelName)
ON (n.propertyName)
```

  - An index that is created on more than one property for any given label is called a composite index:

```
CREATE INDEX [index_name] FOR (n:LabelName)
ON (n.propertyName_1, … n.propertyName_n)
```

  - When the uniqueness and node key constraints are created, single-property and composite indexes are created respectively

16

16

Retrieving and Dropping Indexes

- If created, the single-property indexes will be used in queries with:
  - equality checks (=), range comparisons (>,>=,<, <=), list membership (IN), string comparisons (STARTS WITH, ENDS WITH, CONTAINS), existence checks (exists()), spatial distance searches (distance()), spatial bounding searches (point())
- To retrieve the indexes, you can use:

```
CALL db.indexes()
```

- Or run the browser command **`:schema`** to view existing indexes and constraints defined for the graph:

```
$ :schema
```

- To remove index, use **`DROP INDEX`** command:

```
DROP INDEX ON :Movie(released, videoFormat)
```

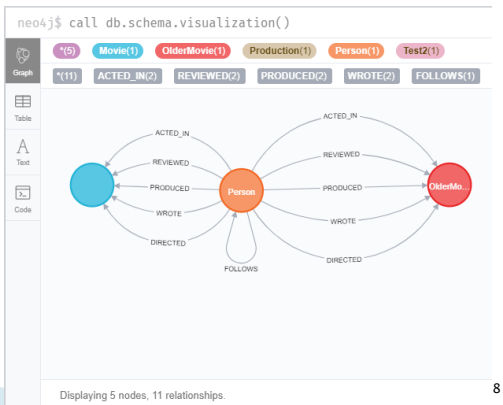
17

17

Examine the data model of the graph

- Before starting traversing the graph it is always useful to examine the data model of the existing graph.
- In order to display the schema of your database, you need to use procedure:

```
CALL
db.schema.visualization()
```



8

18

Filtering Queries with WHERE

- In order to get more flexibility in retrieving data for ranges, partial values, or other criteria use **WHERE** clause that follows a **MATCH** clause
  - In the **WHERE** clause, you can place conditions that are evaluated at runtime to filter the query.
  - The benefit of using a **WHERE** clause is that you can specify potentially complex conditions for the query.
  - **You must specify a variable for any node or relationship you are testing in the WHERE clause.**
- Example: instead of

```
MATCH (a:Person {name: 'Anita'} )
RETURN a
```

  - You can use

```
MATCH (a:Person)
WHERE a.name = 'Anita'
RETURN a
```

  - Using inequality:

```
MATCH (a:Person)
WHERE NOT a.name = 'Anita'
RETURN a
```

19

19

Using List of Values

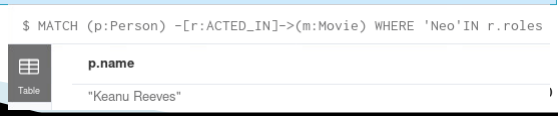
- If you have a set of values you want to use in your filter, use **IN** operator.
  - You can place either numeric or string values in the list
  - If you are testing with a property of a string type, then all the elements of the list should be strings
  - Example:

```
MATCH (p:Person)
WHERE p.born IN [1965, 1970]
RETURN p.name as name, p.born as yearBorn
```

  - You can also compare a value to an existing list in the graph
  - For example, since **ACTED_IN** relationship has a property, **roles** that contains the list of roles:

```
MATCH (p:Person)-[r:ACTED_IN]->(m:Movie)
WHERE  'Neo' IN r.roles AND m.title='The Matrix'
RETURN p.name
```

Result:



20

Logical operators

- You can combine conditions with logical operators similar to SQL:
- **AND** operator

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE m.released = 2019 AND p.name = 'John'
RETURN p.name, m.title, m.released
```

- **OR** operator

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE m.released = 2008 OR m.released = 2009
RETURN p, m
```

- Using ranges:

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE m.released >= 2003 AND m.released <= 2004
RETURN p.name, m.title, m.released
```

- The same as:

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE 2003 <= m.released <= 2004
RETURN p.name, m.title, m.released
```

21

21

Filtering on the Existence of a Property

- Often you need to know if a property exists on a node or relationship
  - Remember: in Neo4j, a property only exists (is stored) if it has a value (a null property will not be stored)
- Use the **`exists()`** method on that property
- Examples:
  - find all users who have a birthdate property:

```
MATCH (p:Person)
WHERE exists(p.birthdate)
RETURN p.name
```

  - find all ***`WORKS_FOR`*** relationships that have a ***`startYear`*** property:

```
MATCH (p:Person)-[rel:WORKS_FOR]->(c:Company)
WHERE exists(rel.startYear)
RETURN p, rel, c
```

22

22

Sorting and Limiting

- Use **ORDER BY** clause to sort the output results
  - However, you cannot use nodes or relationships in the **ORDER BY**, only properties of the nodes or relationships
  - Use **DESC[ENDING]** after the variable to sort in reverse order.

```
MATCH (a:Person)
RETURN a.name ORDER BY a.name
```

- To return a limited subset of the rows use **LIMIT.**

```
MATCH (a:Person)
RETURN a.name ORDER BY a.name LIMIT 10
```

- Use **SKIP** to trim the result set from the top.
  - For example, to return a subset of the result, starting from the fourth result:

```
MATCH (a:Person)
RETURN a.name
ORDER BY a.name
SKIP 3
```

23

Quiz

- **Question 1: A uniqueness constraint and an existence constraint in Cypher are equivalent to which constraints in SQL?**
  - **A.** Primary Key and Foreign Key
  - **B.** Unique Key and Primary Key
  - **C.** Unique Key and Not Null
  - **D.** Primary Key and Not Null
- **Question 2: In order to use** set of values in your filter you need to use the following operator:
  - **A.** `LIST`
  - **B.** `IN`
  - **C.** `LIST OF VALUES`
  - **D.** `IN LIST`
- **Question 3: In order to check if a property exists on a node or relationship which function you need to use?**
  - **A.** `exists()`
  - **B.** `present()`
  - **C.** `exist()`

24

Part 3

25

25

Functions

- Cypher supports rich set of build-in functions:
  - Predicate functions
  - Scalar functions
  - Aggregating functions
  - List functions
  - Mathematical functions – numeric, logarithmic and trigonometric
  - String functions
  - Temporal functions
  - Spatial functions
  - LOAD CSV functions
- As well user-defined functions (written in Java)
- More details here:
  - https://neo4j.com/docs/cypher-manual/current/functions/

26

26

Operators and Functions for Strings

- Cypher:
  - has a set of string-related keywords that you can use in your WHERE clauses to filter by string property values: **`STARTS WITH, ENDS WITH, CONTAINS`**
  - Example:

```
MATCH (p:Person)-[:ACTED_IN]->()
WHERE p.name STARTS WITH 'Michael'
RETURN p.name
```

  - supports number of string functions, for example: **`substring(),toLower(),toString(),toUpper(),trim(),`** etc
    - More details here: https://neo4j.com/docs/cypher-manual/current/functions/string/
    - Example:

```
MATCH (p:Person)-[:ACTED_IN]->()
WHERE toUpper(p.name) STARTS WITH 'MICHAEL'
RETURN p.name
```

    - Note: In the above example, if an index has been created for this property, it **will not be** used at runtime, since this property is converted to upper case.

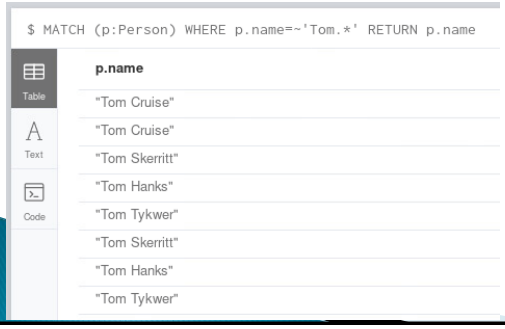
27

Regular Expressions

- To filter property values using regular expressions use the syntax **`=~`** to specify the regular expression
  - The property value must fully match the regular expression.
- Example:
  - retrieve all Person nodes with a name property that begins with 'Tom':

```
MATCH (p:Person)
WHERE p.name =~'Tom.*' RETURN p.name
```

- Result:



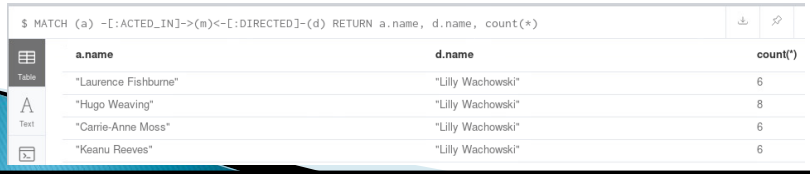
Note:
If you specify a regular expression, the index will never be used.

28

Aggregate Functions

- You can use aggregating functions in Cypher
  - Functions that take multiple values as arguments, and calculate and return an aggregated value from them:
  - avg(), count(), max(), min(), sum(), etc.
- In Cypher (unlike in SQL), you do not need specify a grouping key when perform aggregation in the query.
  - As soon as an aggregation function is used, all non-aggregated result columns become grouping keys.
  - The grouping is implicitly done, based upon the fields in the RETURN clause.

```
MATCH (a)-[:ACTED_IN]->(m)<-[:DIRECTED]-(d)
RETURN a.name, d.name, count(*)
```



29

29

Traversing relationships

- In a graph, we can traverse through nodes to obtain relationships further into the traversal
  - For example, in order to return all followers of the followers of Mary Tho by assigning a variable to the path and return the path:

```
MATCH  path = (:Person)-[:FOLLOWS]->(:Person)-[:FOLLOWS]
->(:Person {name:'Mary Tho'})
RETURN  path
```

- When querying the relationships in a graph, you can take advantage of the direction of the relationship to traverse the graph.
  - For example, get a result containing rows of actors and the movies they acted in, along with the director of each movie:

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person)
RETURN a.name, m.title, d.name
```

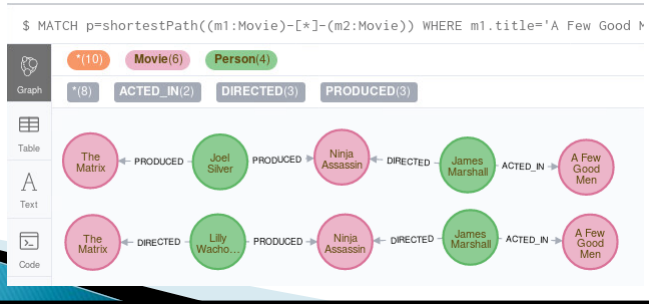
30

Finding The Shortest Path

- Very often using the shortest path between two nodes improves the performance of the query
  - To find the shortest path between two nodes use **`shortestPath()`** function
  - Example: find a shortest path between two movies:

```
MATCH p = shortestPath((m1:Movie)-[*]-(m2:Movie))
WHERE m1.title = 'A Few Good Men' AND m2.title = 'The Matrix'
RETURN  p
```

  - Result:



31

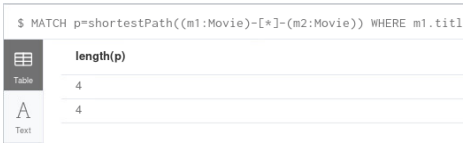
31

Path Functions

- **`length(path)`** – returns the number of relationships in the path

```
MATCH p = shortestPath((m1:Movie)-[*]-(m2:Movie))
WHERE m1.title = 'A Few Good Men' AND m2.title = 'The Matrix'
RETURN  length(p)
```

  - Result:



- **`nodes(path)`** – returns the nodes in the path as a list:

```
MATCH p = shortestPath((m1:Movie)-[*]-(m2:Movie))
WHERE m1.title = 'A Few Good Men' AND m2.title = 'The Matrix'
RETURN  nodes(p)
```

- **`relationships(path)`** – returns the relationships in the path as a list:

```
MATCH p = shortestPath((m1:Movie)-[*]-(m2:Movie))
WHERE m1.title = 'A Few Good Men' AND m2.title = 'The Matrix'
RETURN  relationships(p)
```

32

32

Analysing the Query

- There are two options to choose from when you want to analyse a query by looking at its execution plan:
- **EXPLAIN**
  - If you want to see the execution plan but not run the statement, prepend your Cypher statement with **EXPLAIN**
  - The statement will always return an empty result and make no changes to the database

```
EXPLAIN MATCH p = shortestPath((m1:Movie)-[*]-(m2:Movie))
WHERE m1.title = 'Apollo 13' AND m2.title = 'Cast Away'
RETURN  p
```

- **PROFILE**
  - If you want to run the statement and see which operators are doing most of the work, use **PROFILE**
  - This will run your statement and keep track of how many rows pass through each operator, and how much each operator needs to interact with the storage layer to retrieve the necessary data
  - Note: profiling your query uses more resources, so you should not profile unless you are actively working on a query!

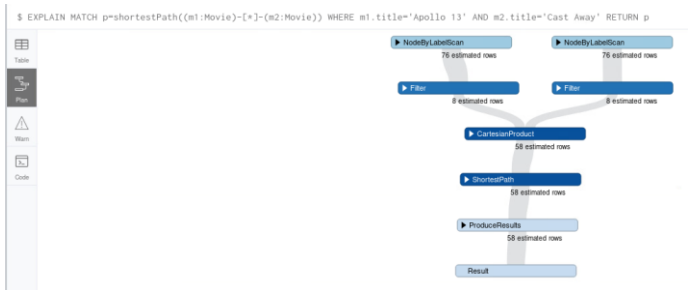
```
PROFILE MATCH p = shortestPath((m1:Movie)-[*]-(m2:Movie))
WHERE m1.title = 'Apollo 13' AND m2.title = 'Cast Away'
RETURN  p
```

33

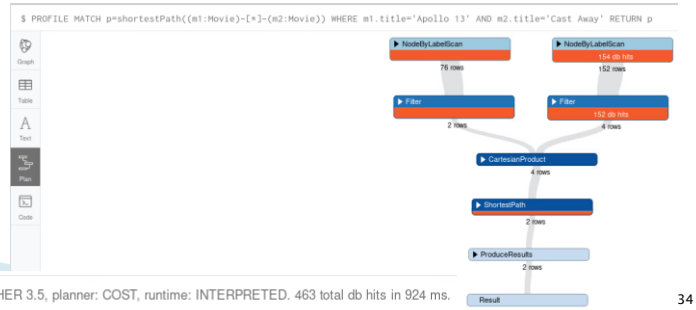
33

Results

- **EXPLAIN:**



- **PROFILE:**



Cypher version: CYPHER 3.5, planner: COST, runtime: INTERPRETED. 463 total db hits in 924 ms.

34

34

Quiz

- **Question 1. Which of the following is not a String function in Cypher?**
  - **A.** `toString()`
  - **B.** `toCypher()`
  - **C.** `trim()`
  - **D.** `toUpper()`
- **Question 2. In Cypher you need to specify a GROUP BY clause when perform aggregation in the query.**
  - **A.** True
  - **B.** False
- **Question 3. Which function do you need to use in order to find the shortest path between two nodes?**
  - **A.** `length(path)`
  - **B.** `shortestPath(path)`
  - **C.** `theshortestPath(path)`
  - **D.** `shortestWay(path)`

35

35

Essentials

- ✓ Introduced more details on CREATE and MERGE commands
- ✓ Described and create constraints
- ✓ Discussed indexes
- ✓ Introduced different ways to filter graph queries and traverse relationships
- ✓ Introduced Cypher functions
  - ✓ More details: https://neo4j.com/docs/cypher-refcard/current/
- ✓ Discussed how to analyse a query

36

36