**MATCH**

MATCH (variable {propertyKey: propertyValue})

RETURN variable

MATCH (variable:Label {propertyKey: propertyValue, propertyKey2: propertyValue2})

RETURN variable

MATCH (m:Movie {released: 2003, tagline: 'Free your mind'})

RETURN m

MATCH (variable:Label {prop1: value, prop2: value})

RETURN variable.prop3

MATCH (p:Person {born: 1965})

RETURN p.name AS name, p.born AS `birth year`

MATCH (node1)-[:REL\_TYPEA \| :REL\_TYPEB]->(node2)

RETURN node1, node2

MATCH (p)-[rel:ACTED\_IN]->(m:Movie {title: 'The Matrix'})

RETURN p, rel, m

MATCH (p:Person)-->(m:Movie {title: 'The Matrix'})

RETURN p, m

MATCH (p:Person)-[rel]->(:Movie {title:'The Matrix'})

RETURN p.name, type(rel)

MATCH (p:Person)-[:REVIEWED {rating: 65}]->(:Movie {title: 'The Da Vinci Code'})

RETURN p.name

MATCH (p1:Person)-[:FOLLOWS]-(p2:Person {name:'Angela Scope'})

RETURN p1, p2

MATCH (p:Person)-[:FOLLOWS]->(:Person)-[:FOLLOWS]->(:Person {name:'Jessica Thompson'})

RETURN p

MATCH path = (:Person)-[:FOLLOWS]->(:Person)-[:FOLLOWS]->(:Person {name:'Jessica Thompson'})

RETURN path

MATCH (:Person {name: 'Diane Keaton'})-[movRel:ACTED\_IN]->

(:Movie {title:"Something's Gotta Give"})

RETURN movRel.roles

**FILTROS**

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE m.released = 2008 OR m.released = 2009

RETURN p, m

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE m.released >= 2003 AND m.released <= 2004

RETURN p.name, m.title, m.released

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE 2003 <= m.released <= 2004

RETURN p.name, m.title, m.released

MATCH (p:Person)-[:ACTED\_IN]->(:Movie {title: 'The Matrix'})

RETURN p.name

=

MATCH (p)-[:ACTED\_IN]->(m)

WHERE p:Person AND m:Movie AND m.title='The Matrix'

RETURN p.name

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE p.name='Jack Nicholson' AND exists(m.tagline)

RETURN m.title, m.tagline

MATCH (p:Person)-[:ACTED\_IN]->()

WHERE toLower(p.name) STARTS WITH 'michael'

RETURN p.name

¿

MATCH (p:Person)

WHERE p.name =~'Tom.\*'

RETURN p.name

?

MATCH (p:Person)-[:WROTE]->(m:Movie)

WHERE NOT exists( (p)-[:DIRECTED]->(m) )

RETURN p.name, m.title

MATCH (p:Person)

WHERE p.born IN [1965, 1970]

RETURN p.name as name, p.born as yearBorn

**AVANZADO**

MATCH (meg:Person)-[:ACTED\_IN]->(m:Movie)<-[:DIRECTED]-(d:Person),

(other:Person)-[:ACTED\_IN]->(m)

WHERE meg.name = 'Meg Ryan'

RETURN m.title as movie, d.name AS director , other.name AS `co-actors`

MATCH megPath = (meg:Person)-[:ACTED\_IN]->(m:Movie)<-[:DIRECTED]-(d:Person),

(other:Person)-[:ACTED\_IN]->(m)

WHERE meg.name = 'Meg Ryan'

RETURN megPath

MATCH (follower:Person)-[:FOLLOWS\* 1..3]->(p:Person)

WHERE follower.name = 'Paul Blythe'

RETURN p

MATCH p = shortestPath((m1:Movie)-[\*]-(m2:Movie))

WHERE m1.title = 'A Few Good Men' AND

m2.title = 'The Matrix'

RETURN p

MATCH (p:Person)

WHERE p.name STARTS WITH 'James'

OPTIONAL MATCH (p)-[r:REVIEWED]->(m:Movie)

RETURN p.name, type(r), m.title

type(r): “REVIEWED”; r: propiedades de la relación.

MATCH (a)-[:ACTED\_IN]->(m)<-[:DIRECTED]-(d)

RETURN a.name, d.name, count(\*)

All rows returned with the same values for a.name and d.name are counted and only returned once. The Cypher count() function is very useful when you want to count the number of occurrences of a particular query result. If you specify count(n), the graph engine calculates the number of occurrences of n. If you specify count(\*), the graph engine calculates the number of rows retrieved, including those with null values. When you use count(), the graph engine does an implicit group by based upon the aggregation.

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE p.name ='Tom Cruise'

RETURN collect(m.title) AS `movies for Tom Cruise`

MATCH (a:Person)-[:ACTED\_IN]->(m:Movie)

WITH a, count(a) AS numMovies, collect(m.title) AS movies

WHERE numMovies > 1 AND numMovies < 4

RETURN a.name, numMovies, movies

Recoge las relaciones entre actores y películas en las que han actuado, cuenta el número de relaciones como número de películas y éstas las reúne como películas.

En el WITH, lo que no sea variable ha de ser nombrado con un alias (AS).

MATCH (p:Person)

WITH p, size((p)-[:ACTED\_IN]->(:Movie)) AS movies

WHERE movies >= 5

OPTIONAL MATCH (p)-[:DIRECTED]->(m:Movie)

RETURN p.name, m.title

Busca el nombre de los actores que hayan actuado en 5 o más películas y opcionalmente el título de una película en la que hayan sido directores.

MATCH (p:Person)-[:DIRECTED | :ACTED\_IN]->(m:Movie)

WHERE p.name = 'Tom Hanks'

RETURN m.released, collect(DISTINCT m.title) AS movies

MATCH (p:Person)-[:DIRECTED | :ACTED\_IN]->(m:Movie)

WHERE p.name = 'Tom Hanks'

WITH DISTINCT m

RETURN m.released, m.title

DISTINCT o WITH DISTINCT elimina duplicados.

MATCH (m:Movie)

RETURN m.title as title, m.released as year ORDER BY m.released DESC LIMIT 10

MATCH (a:Person)-[:ACTED\_IN]->(m:Movie)

WITH m, count(m) AS numCast, collect(a.name) as cast

RETURN m.title, cast, numCast ORDER BY size(cast)

WITH [1, 2, 3] AS list

UNWIND list AS row

RETURN list, row

MATCH (actor:Person)-[:ACTED\_IN]->(:Movie)

WHERE exists(actor.born)

// calculate the age

with DISTINCT actor, date().year - actor.born as age

RETURN actor.name, age as `age today` ORDER BY actor.born DESC

**CREAR**

CREATE (optionalVariable optionalLabels {optionalProperties})

CREATE (m:Movie:Action {title: ' Batman Begins'})

RETURN m.title

CREATE

(:Person {name: 'Michael Caine', born: 1933}),

(:Person {name: 'Liam Neeson', born: 1952}),

(:Person {name: 'Katie Holmes', born: 1978}),

(:Person {name: 'Benjamin Melniker', born: 1913})

SET x:Label1:Label2 // adding two labels to node referenced by the variable x

MATCH (m:Movie)

WHERE m.title = 'Batman Begins'

SET m:Action

RETURN labels(m)

REMOVE x:Label // remove the label from the node referenced by the variable x

MATCH (m:Movie:Action)

WHERE m.title = 'Batman Begins'

REMOVE m:Action

RETURN labels(m)

SET x.propertyName1 = value1 , x.propertyName2 = value2

SET x += {propertyName1: value1, propertyName2: value2}

If you specify += when assigning to a property, the value at valueX is updated if the propertyNnameX exists for the node. If the propertyNameX does not exist for the node, then the property is added to the node.

MATCH (m:Movie)

WHERE m.title = 'Batman Begins'

SET m = {title: 'Batman Begins',

released: 2005,

lengthInMinutes: 140,

videoFormat: 'DVD',

grossMillions: 206.5}

RETURN m

MATCH (m:Movie)

WHERE m.title = 'Batman Begins'

SET m += { grossMillions: 300,

awards: 66}

RETURN m

REMOVE x.propertyName

SET x.propertyName = null

MATCH (m:Movie)

WHERE m.title = 'Batman Begins'

SET m.grossMillions = null

REMOVE m.videoFormat

RETURN m

CREATE (x)-[:REL\_TYPE]->(y)

MATCH (a:Person), (m:Movie)

WHERE a.name = 'Michael Caine' AND m.title = 'Batman Begins'

CREATE (a)-[:ACTED\_IN]->(m)

RETURN a, m

MATCH (a:Person), (m:Movie), (p:Person)

WHERE a.name = 'Liam Neeson' AND

m.title = 'Batman Begins' AND

p.name = 'Benjamin Melniker'

CREATE (a)-[:ACTED\_IN]->(m)<-[:PRODUCED]-(p)

RETURN a, m, p

SET r.propertyName1 = value1 , r.propertyName2 = value2

SET r += {propertyName1: value1, propertyName2: value2}

MATCH (a:Person), (m:Movie)

WHERE a.name = 'Christian Bale' AND m.title = 'Batman Begins'

CREATE (a)-[:ACTED\_IN {roles: ['Bruce Wayne', 'Batman']}]->(m)

RETURN a, m

MATCH (a:Person),(m:Movie)

WHERE a.name = 'Christian Bale' AND

m.title = 'Batman Begins' AND

NOT exists((a)-[:ACTED\_IN]->(m))

CREATE (a)-[rel:ACTED\_IN]->(m)

SET rel.roles = ['Bruce Wayne','Batman']

RETURN a, rel, m

MATCH (a:Person)-[rel:ACTED\_IN]->(m:Movie)

WHERE a.name = 'Christian Bale' AND m.title = 'Batman Begins'

REMOVE rel.roles

RETURN a, rel, m

MATCH (p:Person)

WHERE p.name = 'Liam Neeson'

DETACH DELETE p

When you specify DETACH DELETE for a node, the relationships to and from the node are deleted, then the node is deleted.

MERGE (variable:Label{nodeProperties})

RETURN variable

The MERGE clause is used to find elements in the graph. But if the element is not found, it is created. You use the MERGE clause 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.

Create a node and relationship to it uniquely in the context of another node.

MERGE (a:Actor {name: 'Michael Caine'})

SET a.born = 1933

RETURN a

MERGE (a:Person {name: 'Sir Michael Caine'})

ON CREATE SET a.born = 1934,

a.birthPlace = 'UK'

ON MATCH SET a.birthPlace = 'UK'

RETURN a

MERGE (variable:Label {nodeProperties})-[:REL\_TYPE]->(otherNode)

RETURN variable

MATCH (p:Person), (m:Movie)

WHERE m.title = 'Batman Begins' AND p.name ENDS WITH 'Caine'

MERGE (p)-[:ACTED\_IN]->(m)

RETURN p, m

MERGE (fromDate:Date {year: 2018})<-[:IN\_YEAR]-(toDate:Date {month: 'January'})

You should first MERGE your nodes and then your relationships. Only if you intentionally want to create a node within the context of another (like a month within a year) then a MERGE pattern with one bound and one unbound node makes sense.

**MORE**

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE p.name = $actorName

RETURN m.released, m.title ORDER BY m.released DESC

At runtime, if the parameter $actorName has a value, it will be used in the Cypher statement when it runs in the graph engine.

:param actorName => 'Tom Hanks'

Set the value of a single parameter.

:params {actorName: 'Tom Cruise', movieName: 'Top Gun'}

EXPLAIN MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE p.name = $actorName AND

m.released < $year

RETURN p.name, m.title, m.released

EXPLAIN provides estimates of the graph engine processing that will occur, but does not execute the Cypher statement.

PROFILE provides real profiling information for what has occurred in the graph engine during the query and executes the Cypher statement.

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

Example for ensuring that the title for a node of type Movie is unique. If we attempt to create a Movie with the title, The Matrix, the Cypher statement will fail because the graph already has a movie with that title.

CREATE CONSTRAINT ON (m:Movie) ASSERT exists(m.tagline)

Here is an example for adding the existence constraint to the tagline property of all Movie nodes in the graph.

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

We can create an existence constraint on a property as follows:

MATCH (p:Person), (m:Movie)

WHERE p.name = 'Jessica Thompson' AND

m.title = 'The Matrix'

MERGE (p)-[:REVIEWED {summary: 'Great movie!'}]->(m)

So after creating this constraint, if we attempt to create a :REVIEWED relationship without setting the rating property.

CALL db.constraints()

You can run the browser command :schema to view existing indexes and constraints defined for the graph.

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

Here we drop the existence constraint for the rating property for all REVIEWED relationships in the graph.

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

Here is an example to create this node key. This attempt to create the constraint failed because there are Person nodes in the graph that do not have the born property defined.

MATCH (p:Person)

WHERE NOT exists(p.born)

SET p.born = 0

We set these properties for all nodes in the graph that do not have born properties.

MATCH (m:Movie)

WHERE 1990 < m.released < 2000

SET m.videoFormat = 'DVD'

Testing the value of the released property of a Movie node using ranges.

CREATE INDEX ON :Movie(released)

Creating indexes.

MATCH (m:Movie)

WHERE m.released >= 2000

SET m.videoFormat = 'DVD';

MATCH (m:Movie)

WHERE m.released < 2000

SET m.videoFormat = 'VHS'

CREATE INDEX ON :Movie(released, videoFormat)

Composite index on these properties as follows.

CALL db.indexes()

DROP INDEX ON :Movie(released, videoFormat)

**IMPORT DATA**

LOAD CSV WITH HEADERS FROM url-value

AS row // row is a variable that is used to extract data

Here is the simplified syntax for using LOAD CSV:

LOAD CSV WITH HEADERS

FROM 'http://data.neo4j.com/intro-neo4j/movies\_to\_load.csv'

AS line

RETURN count(\*)

You can execute this Cypher statement to get a count of the data to be loaded from the movies\_to\_load.csv file so you have an idea of how much data will be loaded.

LOAD CSV WITH HEADERS

FROM 'https://data.neo4j.com/intro-neo4j/movies\_to\_load.csv'

AS line

RETURN \* LIMIT 1

You might even want to visually inspect the data before you load it to see if it is what you were expecting.

LOAD CSV WITH HEADERS

FROM 'http://data.neo4j.com/intro-neo4j/movies\_to\_load.csv'

AS line

RETURN line.id, line.title, toInteger(line.year), trim(line.summary)

You may want to format the data before it is loaded to confirm it matches what you want in your graph.

LOAD CSV WITH HEADERS

FROM 'https://data.neo4j.com/intro-neo4j/movies\_to\_load.csv'

AS line

CREATE (movie:Movie { movieId: line.id, title: line.title, released: toInteger(line.year) , tagline: trim(line.summary)})

The following query creates the Movie nodes using some of the data from movies\_to\_load.csv as properties.

LOAD CSV WITH HEADERS

FROM 'https://data.neo4j.com/intro-neo4j/persons\_to\_load.csv'

AS line

MERGE (actor:Person { personId: line.Id })

ON CREATE SET actor.name = line.name,

actor.born = toInteger(trim(line.birthyear))

In case you already have people in your database, you will want to avoid creating duplicates. That’s why instead of just creating them, we use MERGE to ensure unique entries after the import. We use the ON CREATE clause to set the values for name and born.

LOAD CSV WITH HEADERS

FROM 'https://data.neo4j.com/intro-neo4j/roles\_to\_load.csv'

AS line

MATCH (movie:Movie { movieId: line.movieId })

MATCH (person:Person { personId: line.personId })

CREATE (person)-[:ACTED\_IN { roles: [line.role]}]->(movie)

The query below matches the entries of line.personId and line.movieId to their respective Movie and Person nodes, and creates an ACTED\_IN relationship between the person and the movie. This model includes a relationship property of role, which is passed via line.role.

LOAD CSV WITH HEADERS

FROM 'https://data.neo4j.com/intro-neo4j/movie\_actor\_roles\_to\_load.csv'

AS line FIELDTERMINATOR ';'

MERGE (movie:Movie { title: line.title })

ON CREATE SET movie.released = toInteger(line.released),

movie.tagline = line.summary

MERGE (actor:Person { name: line.actor })

ON CREATE SET actor.born = toInteger(line.birthyear)

MERGE (actor)-[r:ACTED\_IN]->(movie)

ON CREATE SET r.roles = split(line.characters,',')

Here are the Cypher statements to load your file, which contains denormalized data.

**OTROS**

Asd

MATCH (n) WHERE NOT (EXISTS (n.name)) DETACH DELETE n