Cypher Query Language

Dr James Xue

James.xue@northampton.ac.uk

CREATE UNIQUE

- The CREATE UNIQUE clause is a mix of MATCH and CREATE — it will match what it can, and create what is missing.
- The CREATE UNIQUE clause was removed in Cypher 3.2. Using the CREATE UNIQUE clause will cause the query to fall back to using Cypher 3.1. Use MERGE instead of CREATE UNIQUE;
- In the CREATE UNIQUE clause, patterns are used a lot.

Using CREATE UNIQUE

- Using MERGE the same level of uniqueness guaranteed by CREATE UNIQUE for nodes and relationships.

```
MERGE (p:Person {name: 'Joe'})
RETURN p

MATCH (a:Person {name: 'Joe'})
CREATE UNIQUE (a)-[r:LIKES]->(b:Person {name: 'Jill'})-[r1:EATS]->(f:Food {name: 'Margarita Pizza'})
RETURN a

MATCH (a:Person {name: 'Joe'})
CREATE UNIQUE (a)-[r:LIKES]->(b:Person {name: 'Jill'})-[r1:EATS]->(f:Food {name: 'Banana'})
RETURN a
```

- This will create two :Person nodes, a :LIKES relationship between them, and two :EATS relationships from one of the :Person nodes to two :Food nodes. No node or relationship is duplicated.

Using MERGE Instead

- The following set of queries — using MERGE — will achieve the same result:

```
MERGE (p:Person {name: 'Joe'})
RETURN p

MATCH (a:Person {name: 'Joe'})
MERGE (b:Person {name: 'Jill'})
MERGE (a)-[r:LIKES]->(b)
MERGE (b)-[r1:EATS]->(f:Food {name: 'Margarita Pizza'})
RETURN a

MATCH (a:Person {name: 'Joe'})
MERGE (b:Person {name: 'Jill'})
MERGE (a)-[r:LIKES]->(b)
MERGE (b)-[r1:EATS]->(f:Food {name: 'Banana'})
RETURN a
```

CREATE UNIQUE …

- The general guidance: use MERGE to prevent node duplication, but CREATE UNIQUE should be used to prevent relationship duplication.

```
MATCH (e:Episode {title: "foo"})
MERGE (p:Person {name: "Lynn Rose"})
ON CREATE SET p.firstname = "Lynn", p.surname = "Rose"
CREATE UNIQUE (e) <- [:INTERVIEWED_IN] - (p)
```

If the node doesn't exist, it is created using MERGE ON CREATE.
If the relationship doesn't exist, it is created using CREATE UNIQUE.
If both node and relationship exist, then nothing is changed.

CREATE UNIQUE …

- The general guidance: use MERGE to prevent node duplication, but CREATE UNIQUE should be used to prevent relationship duplication.

```
MATCH (e:Episode {title: "foo"})
MERGE (p:Person {name: "Lynn Rose"})
ON CREATE SET p.firstname = "Lynn", p.surname = "Rose"
CREATE UNIQUE (e) <- [:INTERVIEWED_IN] - (p)
```

If the node doesn't exist, it is created using MERGE ON CREATE.
If the relationship doesn't exist, it is created using CREATE UNIQUE.
If both node and relationship exist, then nothing is changed.

Try It Out

- Run the following commands (without existing ‘Lynn Rose’ node):

```
CREATE (e:Episode{title:"foo"})

MERGE (p:Person{name:"Lynn Rose"})
ON CREATE SET p.name="Lynn Rose", p.state="created"
ON MATCH SET p.state="matched"
CREATE UNIQUE (e)<-[:INTERVIEWED_IN]-(p)
```

Output:

name	Lynn Rose
state	created

Try It Out …

- The run the following commands (with existing ’Lynn Rose’ node):

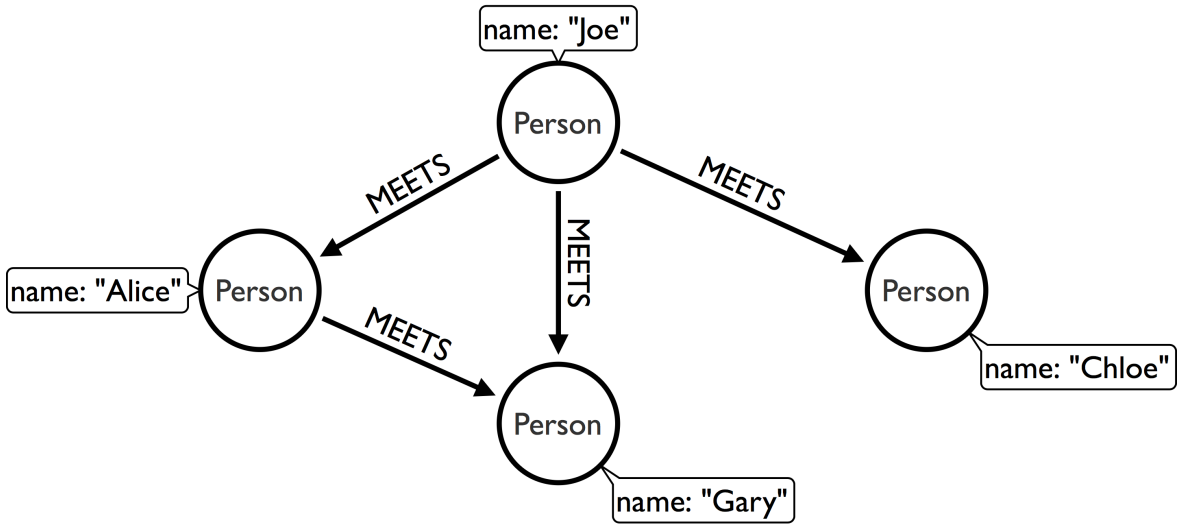
```
MATCH (n) detach delete n

CREATE (p:Person{name:"Lynn Rose"})

CREATE (e:Episode{title:"foo"})

MERGE (p:Person{name:"Lynn Rose"})
MERGE (e:Episode{title:"foo"})
ON CREATE SET p.name = "Lynn Rose",p.state = "created"
ON MATCH SET p.state="matched"
CREATE UNIQUE (e)<-[:INTERVIEWED_IN]-(p)
```

Output:

name	Lynn Rose
state	matched

Create Nodes If Missing

- If the pattern described needs a node, and it can't be matched, a new node will be created.




```
CREATE (:Person{name:'Joe'})
CREATE (:Person{name:'Alice'})
CREATE (:Person{name:'Chloe'})
CREATE (:Person{name:'Gary'})

MATCH (p1:Person{name:'Joe'}),(p2:Person{name:'Alice'})
CREATE (p1)-[:MEETS]->(p2)

MATCH (p1:Person{name:'Joe'}),(p2:Person{name:'Chloe'})
CREATE (p1)-[:MEETS]->(p2)

MATCH (p1:Person{name:'Joe'}),(p2:Person{name:'Gary'})
CREATE (p1)-[:MEETS]->(p2)

MATCH (p1:Person{name:'Alice'}),(p2:Person{name:'Gary'})
CREATE (p1)-[:MEETS]->(p2)

MATCH (p1:Person{name:'Joe'})
CREATE UNIQUE (p1)-[:LOVES]->(someone)
```

Note: before running the commands on the Right, it would be better to clear the database Using DETACH DELETE command.

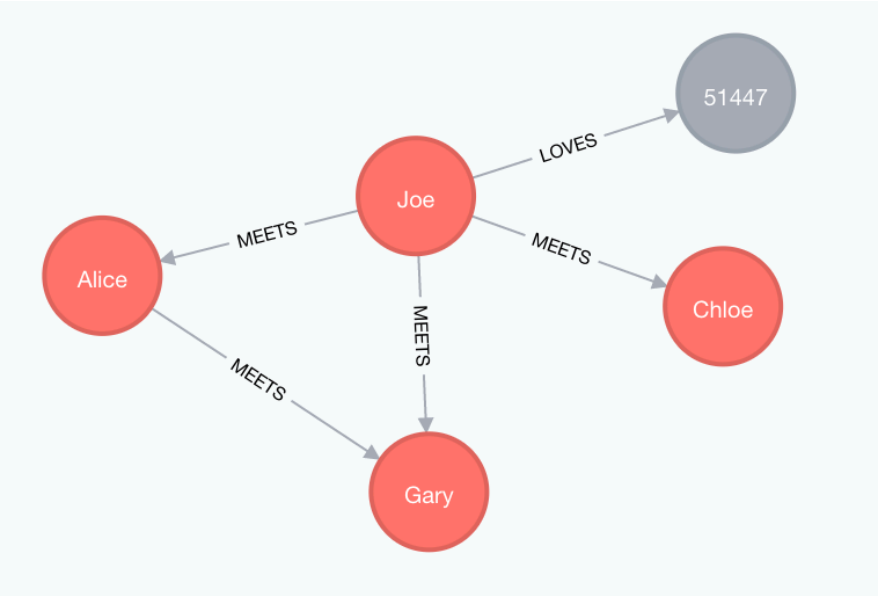
The node 'Joe' doesn't have any LOVES relationships, and so a node is created, and also a relationship to that node.

Try It Out

- Use the following command to verify the newly created node

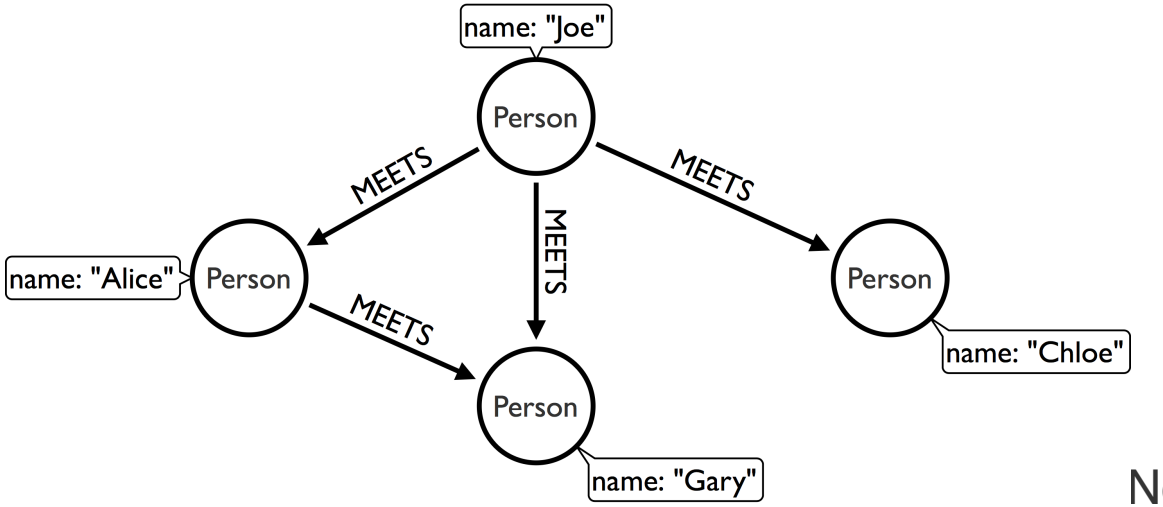
```
MATCH p=(n)-[:MEETS|KNOWS|LOVES]-()
RETURN p
```

The grey node is the ‘someone’ that ’Joe’ LOVES



Create Nodes With Values

- The pattern described can also contain values on the node. These are given using the following syntax: prop : <expression>.




```
MATCH (joe:Person{name:'Joe'})
CREATE UNIQUE (joe)-[:MEETS]->
(another:Person{name:'Scott'})
RETURN another
```

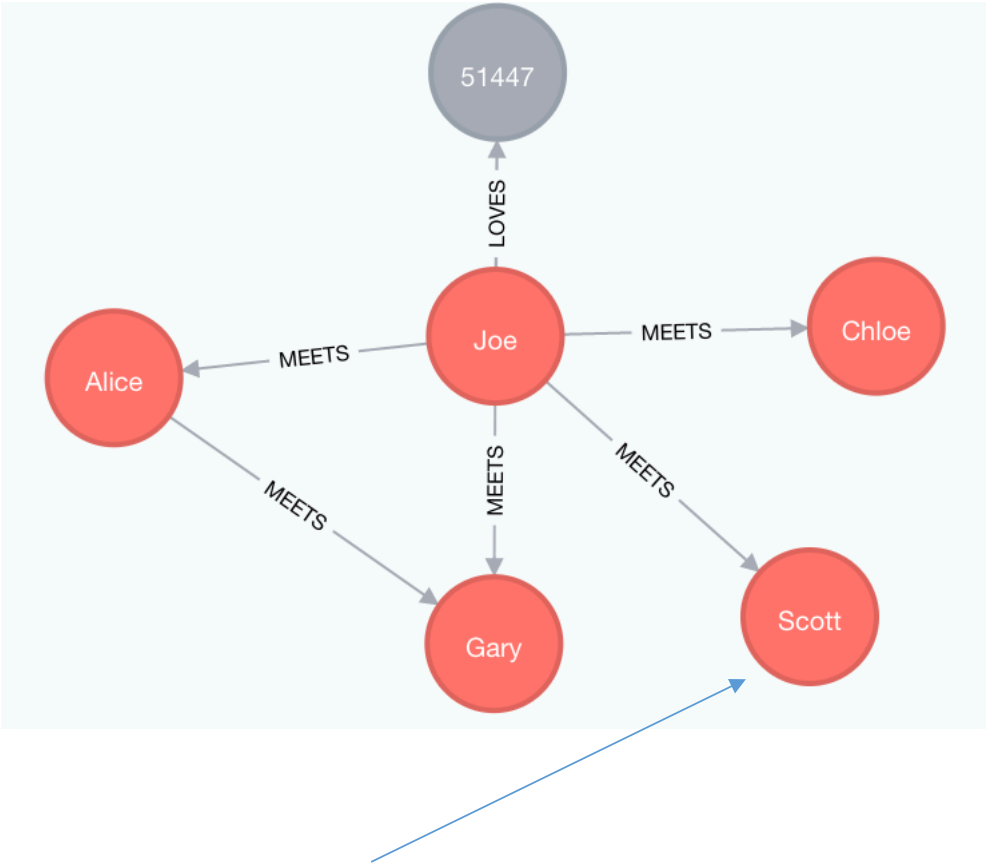
No node connected with the ‘Joe’ node has the name ‘Scott’, and so a new node is created to match the pattern.

Try It Out

- Run the following commands:

```
MATCH (joe:Person{name:'Joe'})
CREATE UNIQUE (joe)-[:MEETS]->
(another:Person{name:'Scott'})
RETURN another
```

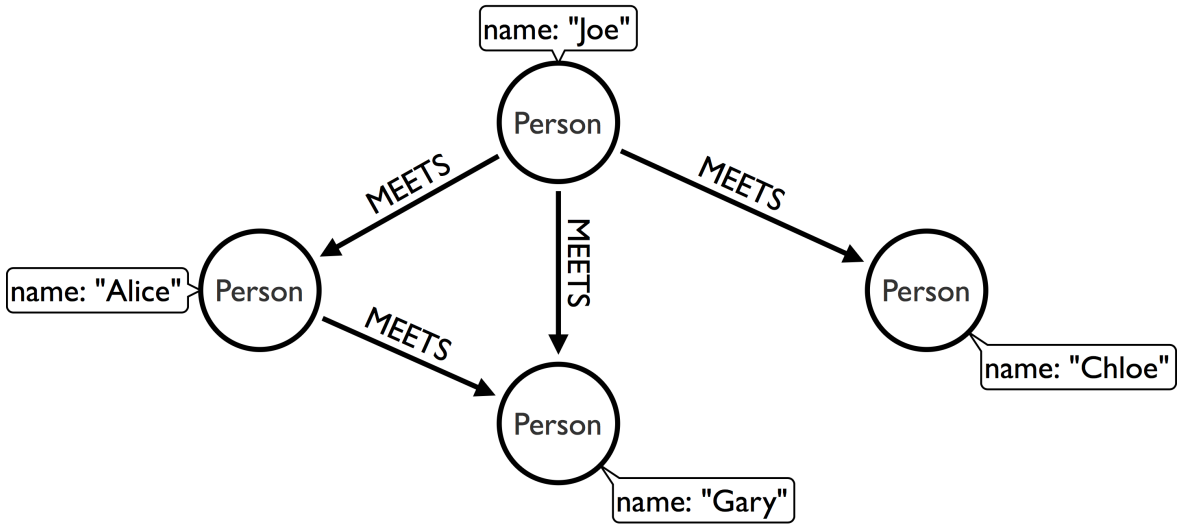
```
MATCH p=(n)-[:MEETS|KNOWS|LOVES]-()
RETURN p
```



The node ‘Scott’ is created

Create Labeled Node If Missing

- If the pattern described needs a labeled node and there is none with the given labels, Cypher will create a new one.




```
MATCH (scott:Person{name:'Scott'})
CREATE UNIQUE (scott)-[:KNOWS]->
(c:Professor{name:'Jennifer'})
```

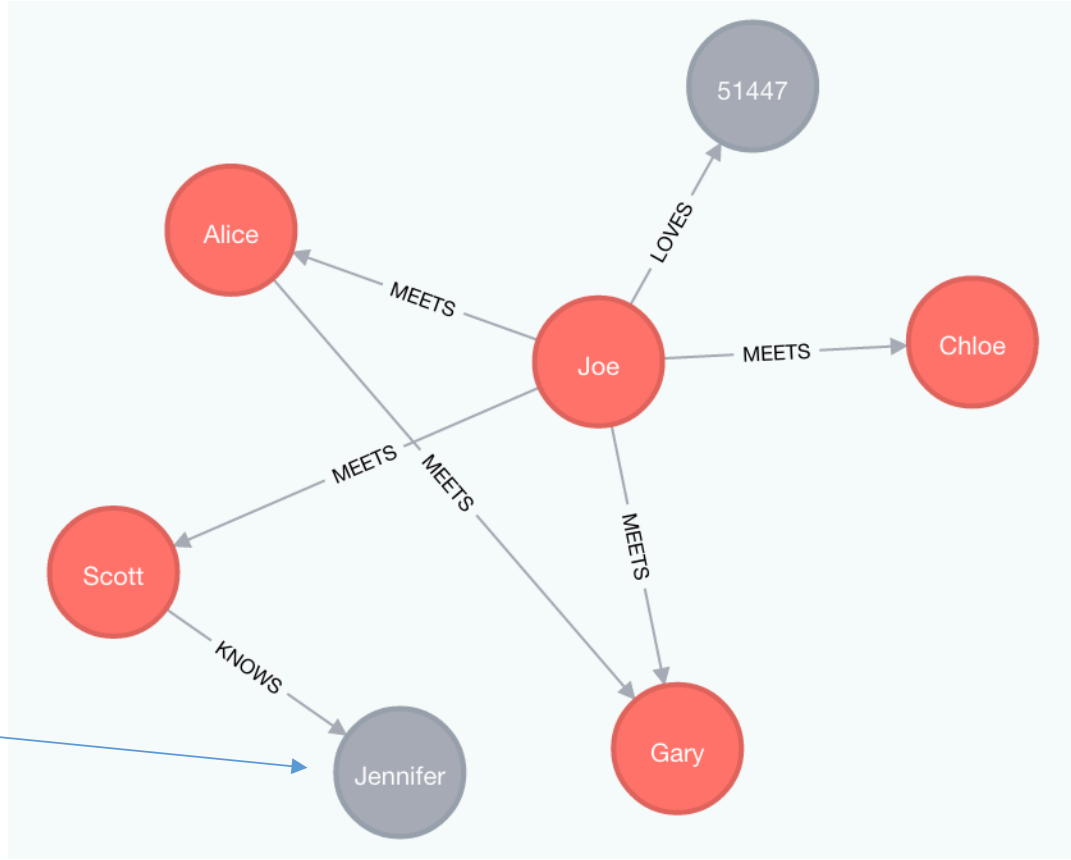
A new node ‘Jennifer’ with the label ‘Professor’ will be created and connected to the ‘Scott’ node.

Try it out

- Run the following commands to verify it.

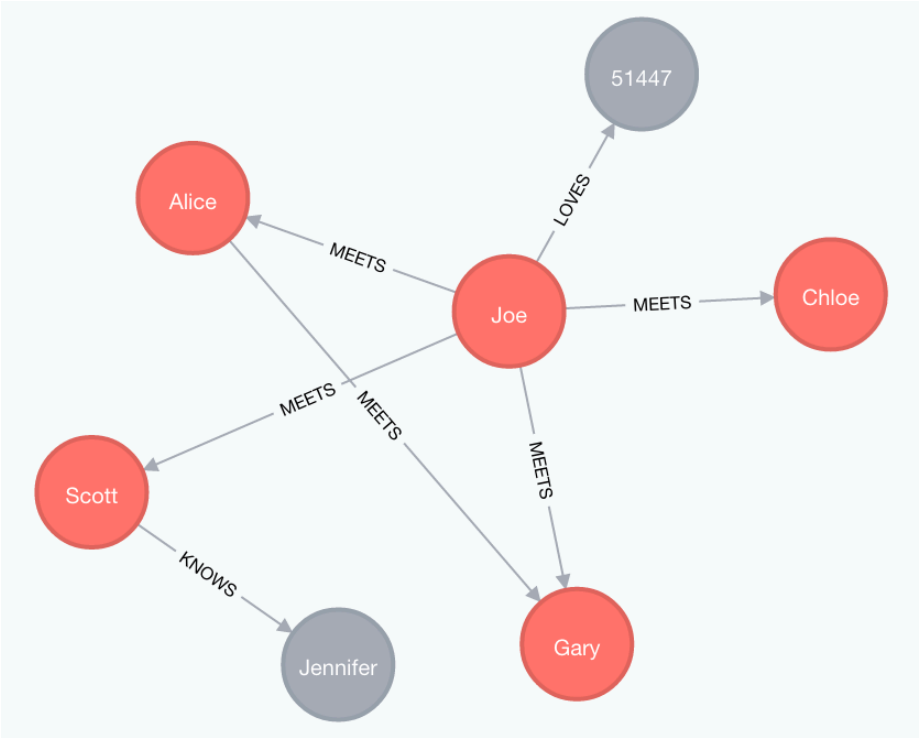
```
MATCH p=(n)-[:MEETS|KNOWS|LOVES]-()
RETURN p
```

A new node ‘Jennifer’ with the label ‘Professor’ will be created and connected to the ‘Scott’ node.



Create Relationship If It Is Missing

- The ‘Alice’ node is matched against the two nodes – ‘Gary’ and ‘Chloe’. One relationship already exists and can be matched, and the other relationship is created before it is returned.



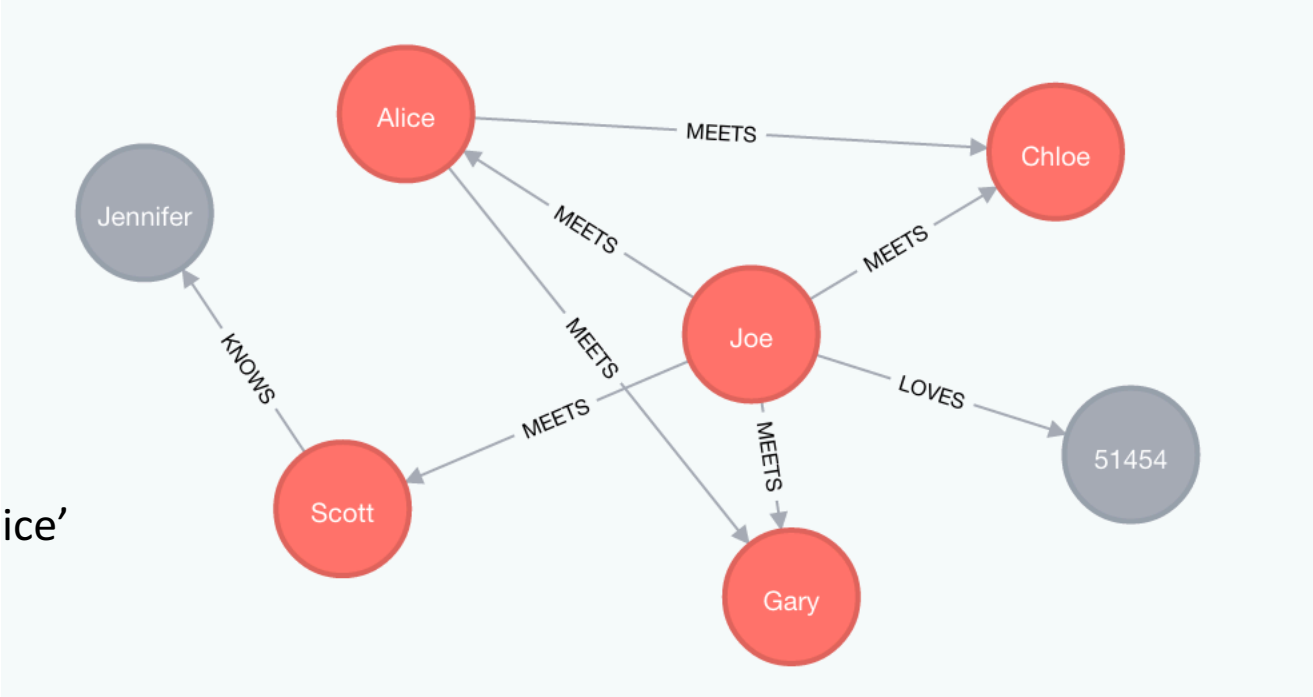

```
MATCH (alice:Person{name:'Alice'}), (others)
WHERE others.name IN ['Gary','Chloe']
CREATE UNIQUE (alice)-[r:MEETS]->(others)
RETURN r
```

Another MEETS relationship will be created between ‘Alice’ ’Chloe’ nodes.

Try It Out

- Run the following commands to verify the output:

```
MATCH p=(n)-[:MEETS|KNOWS|LOVES]-()
RETURN p
```



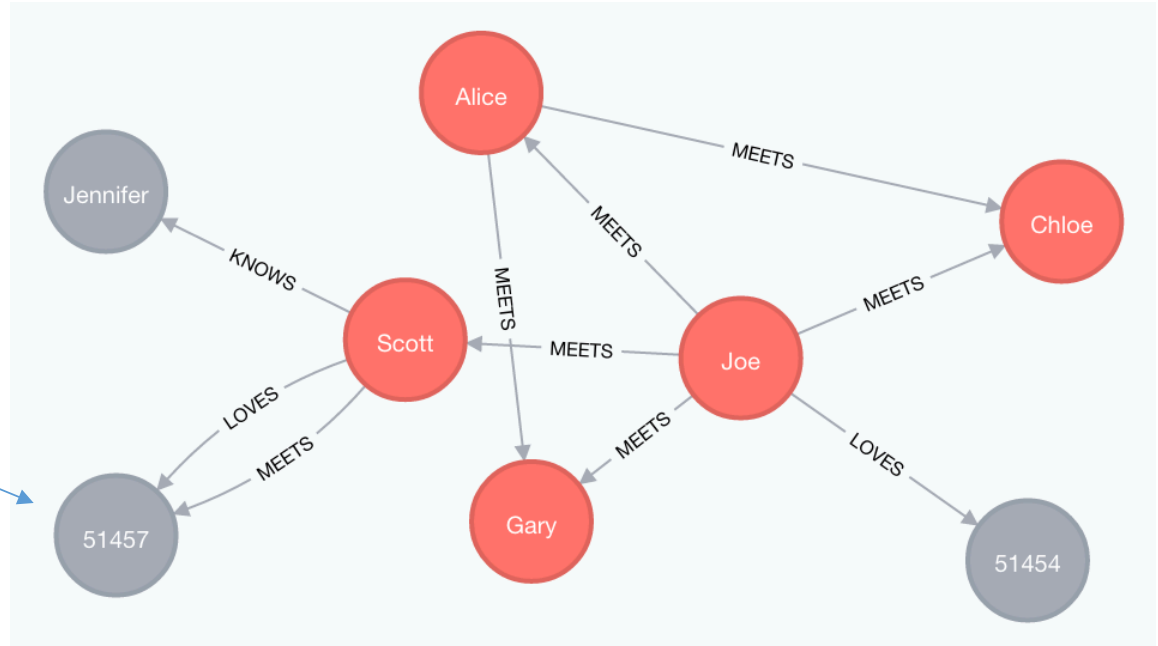
There is a MEETS relationship between ‘Alice’ and ‘Chloe’ nodes now.

Describe Complex Pattern

- The pattern described by CREATE UNIQUE can be separated by commas, just like in MATCH and CREATE.

```
MATCH (scott:Person{name:'Scott'})
CREATE UNIQUE (scott)-[:MEETS]->(another),(scott)-[:LOVES]->(another)
RETURN another
```

This is the ‘another’ node



Exercises

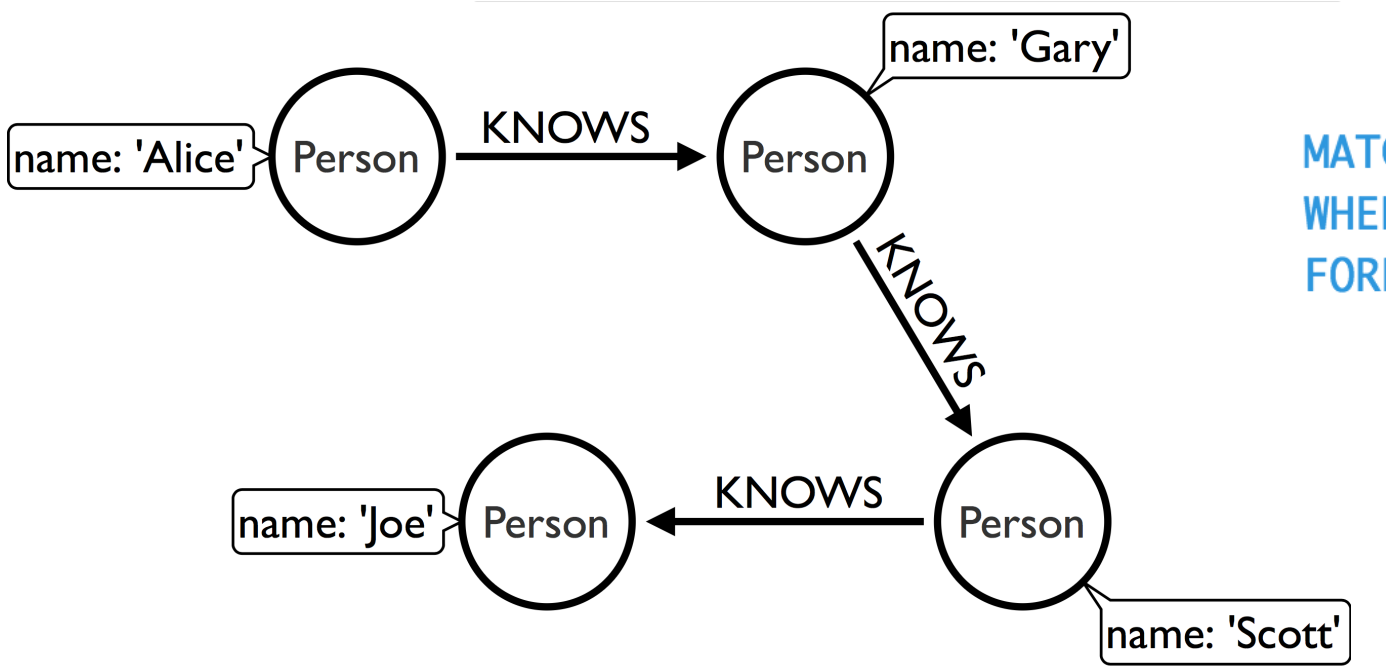
- Refer to the diagram on the previous slide, and write some commands to create a KNOWS relationship between the ‘Scott’ and ‘Gary’ nodes with an attribute:value ‘since:1996’.
- You need to use the CREATE UNIQUE command
- Refer to previous codes if you are not sure how to do it.

FOREACH

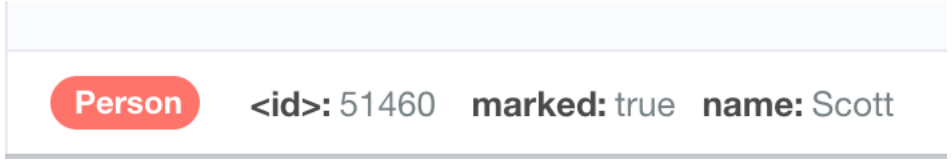
- The FOREACH clause is used to update data within a list, whether components of a path, or result of aggregation.
- Inside of the FOREACH parentheses, you can do any of the updating commands — CREATE, CREATE UNIQUE, MERGE, DELETE, and FOREACH.

Mark All Nodes Along A Path

- The following query will create a property called ‘marked’ for each node and SET the value to TRUE.




```
MATCH p=(begin)-[*]->(end)
WHERE begin.name='Alice' AND end.name='Joe'
FOREACH (n IN nodes(p) | SET n.marked = TRUE)
```



A new attribute called ‘marked’ is created and set to ‘TRUE’.

Exercises

- Create some graph nodes (without an attribute called ‘marked’) and relationships based on the model on the previous slide.
- Run the FOREACH command from the previous slice to set the value of ‘marked’ attribute to ‘TRUE’.
- Verify the output.