Curso NoSQL Neo4J

UNIVERSIDAD TECNOLOGICA NACIONAL

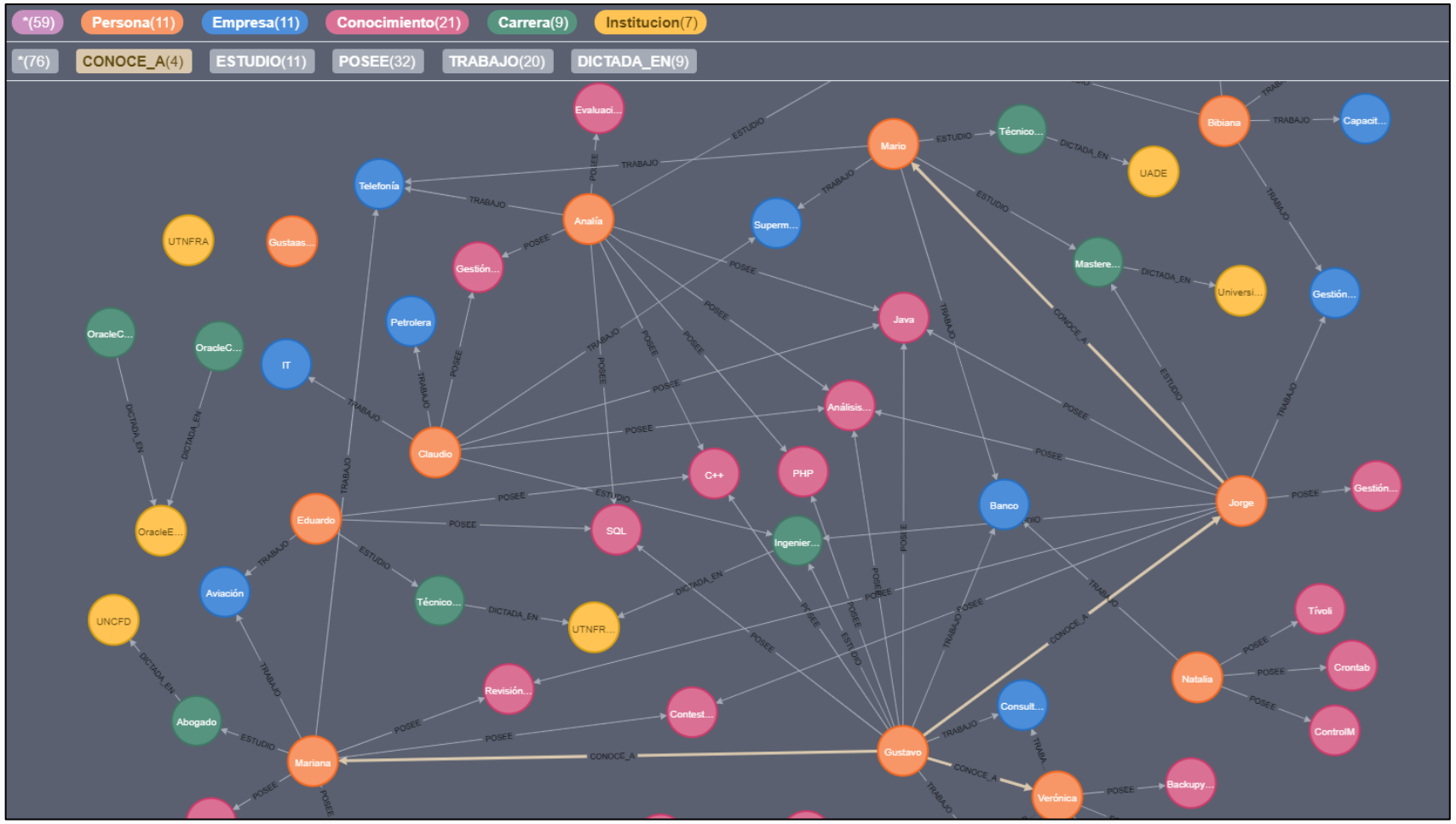
Modificación de Nodos y Relaciones / Merge / Consultas Avanzadas

DBlandIT

Índice

Agregar etiquetas a un Nodo

Agregar etiquetas a un Nodo



DBlandIT

Agregar etiquetas a un Nodo

Sintaxis

```
MATCH (nodo:etiqueta {propiedades})
WHERE condiciones
SET nodo:Etiqueta
```

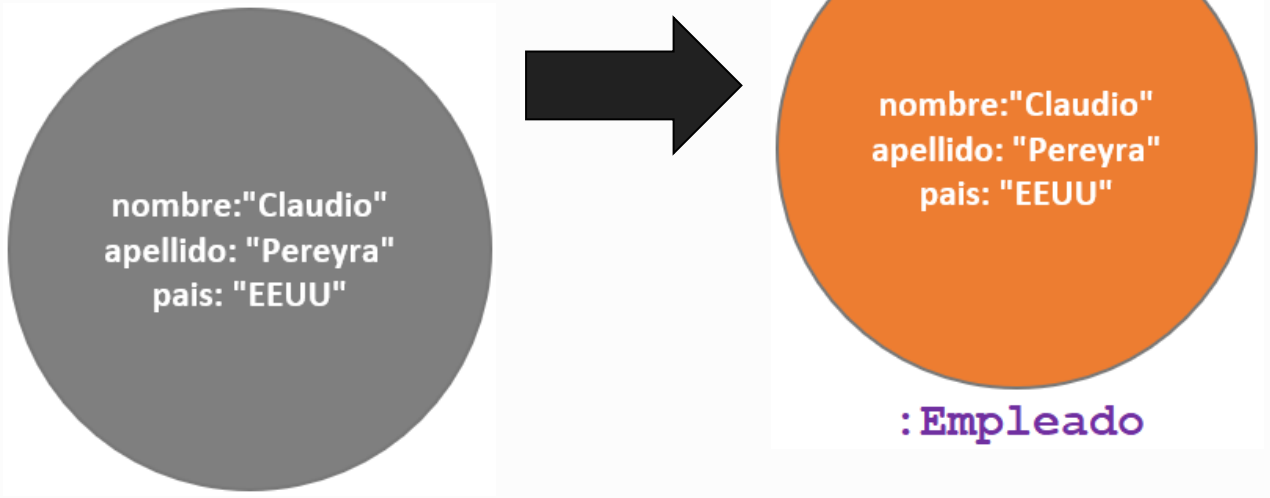
Permite agregar una etiqueta a un nodo determinado.

Ejemplo 1

Agregar a Claudio Pereyra la etiqueta **"Empleado".**

```
MATCH (a:Persona {nombre:"Claudio", apellido: "Pereyra"})
SET a:Empleado RETURN a
```


```
{
  "nombre": "Claudio",
  "email": "cpereyra30@yahoo.com.ar",
  "fechanac": "18/05/1993",
  "apellido": "Pereyra",
  "pais": "EstadosUnidos"
}
```

Agregar etiquetas a un Nodo

Puede darse el caso de que uno o más nodos del grafo no tengan etiquetas (esto es totalmente posible).

Si quisiéramos ponerle etiquetas a dichos nodos y el grafo fuera muy grande, sería una tarea compleja identificarlos a todos de forma manual.

Se puede encontrar a todos los nodos sin etiquetas definidas en un grafo de la siguiente forma:

Sintaxis

```
MATCH (n)
WHERE size(labels(n)) = 0
RETURN n
```

Agregar Propiedades a un Nodo

Sintaxis

```
MATCH (nodo:etiqueta {propiedades})
WHERE condiciones
SET nodo.propiedad=valor
```

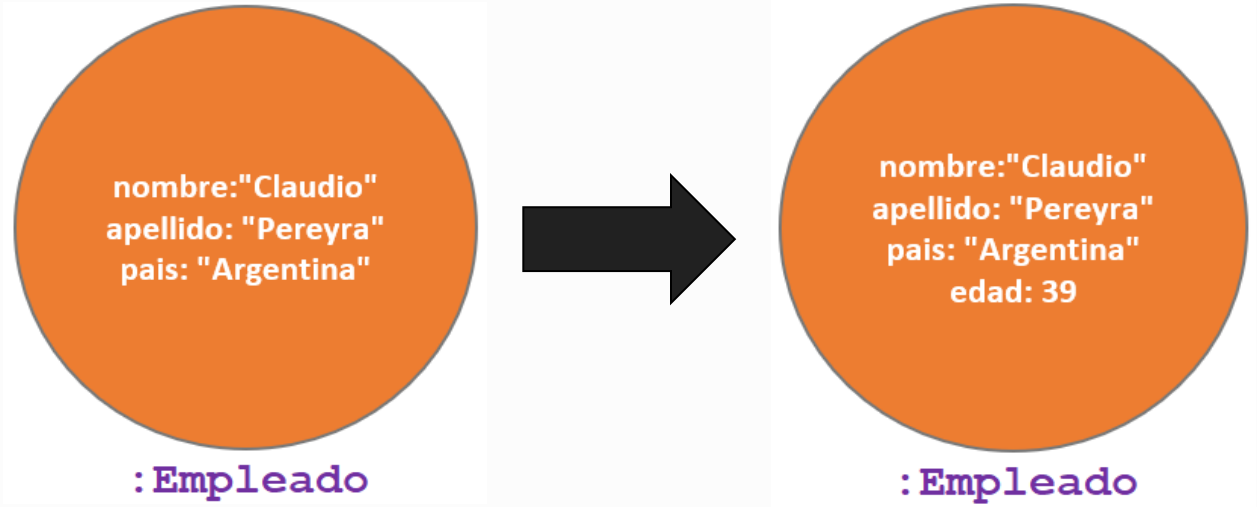
Permite agregar una nueva propiedad a un nodo o modificar el valor de una propiedad existente en un nodo.

Ejemplo 1

Agregar la edad de Claudio Pereyra con valor **39.**

```
MATCH (a:Persona {nombre:"Claudio", apellido: "Pereyra"})
SET a.edad=39
RETURN a
```


```
{
  "edad": 39,
  "nombre": "Claudio",
  "email": "cpereyra30@yahoo.com.ar",
  "fechanac": "18/05/1993",
  "apellido": "Pereyra",
  "pais": "Argentina"
}
```

Modificar Propiedades de un Nodo

Sintaxis

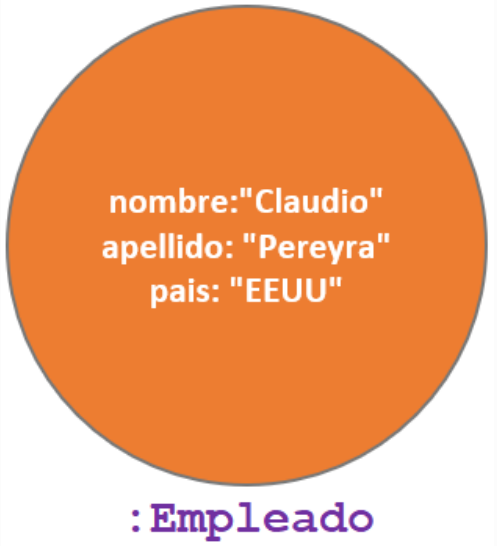
```
MATCH (nodo:etiqueta {propiedades})
WHERE condiciones
SET nodo.propiedad=valor
```

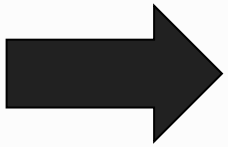
Permite agregar una nueva propiedad a un nodo o modificar el valor de una propiedad existente en un nodo.

Ejemplo 1

Cambiar el valor del país de Claudio Pereyra a **Argentina.**

```
MATCH (a:Persona {nombre:"Claudio", apellido: "Pereyra"})
SET a.pais="Argentina"
RETURN a
```



Eliminar la etiqueta de un Nodo

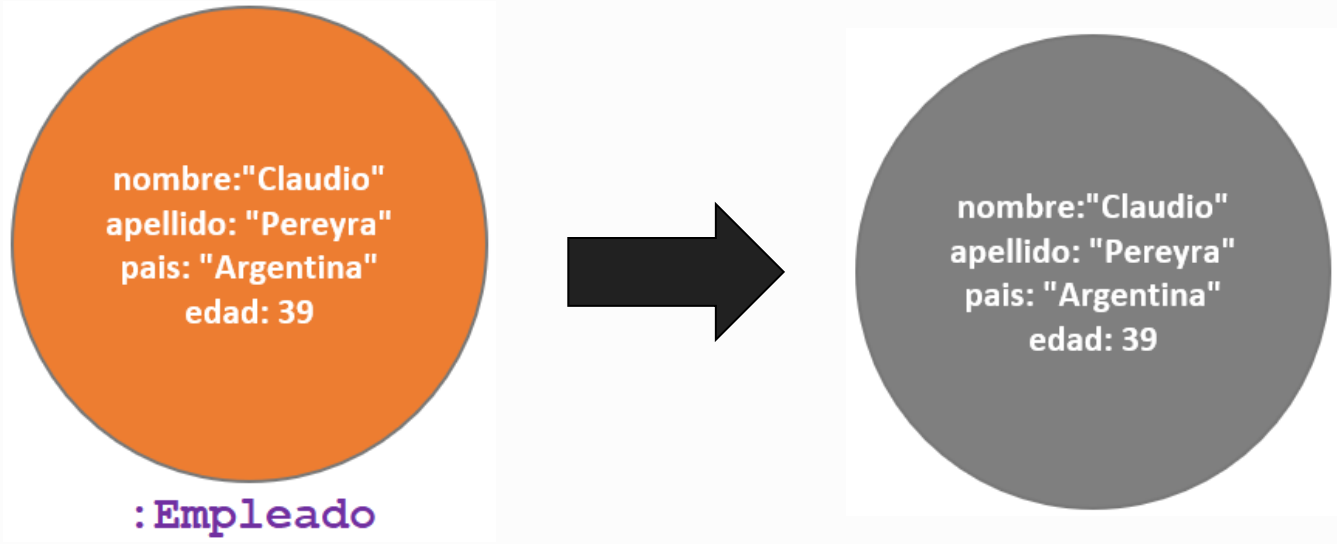
Permite eliminar una etiqueta de un nodo determinado.

Sintaxis

```
MATCH (nodo:etiqueta {propiedades})
WHERE condiciones
REMOVE nodo:Etiqueta
```

Ejemplo 1: Eliminar la etiqueta **"Empleado"** a Claudio Pereyra**.**

```
MATCH (a:Persona {nombre:"Claudio", apellido: "Pereyra"})
REMOVE a:Empleado
```



Eliminar múltiples etiquetas de un Nodo

Permite eliminar una etiqueta a un nodo determinado.

Sintaxis

```
MATCH (nodo:etiqueta {propiedades})
WHERE condiciones
REMOVE nodo:Etiqueta
```

Ejemplo 1

Eliminar a Claudio Pereyra **varias etiquetas.**

```
MATCH (a:Persona {nombre:"Claudio", apellido: "Pereyra"})
REMOVE a:Empleado:Padre:Docente
```

Agregar Propiedades a una Relación

Sintaxis

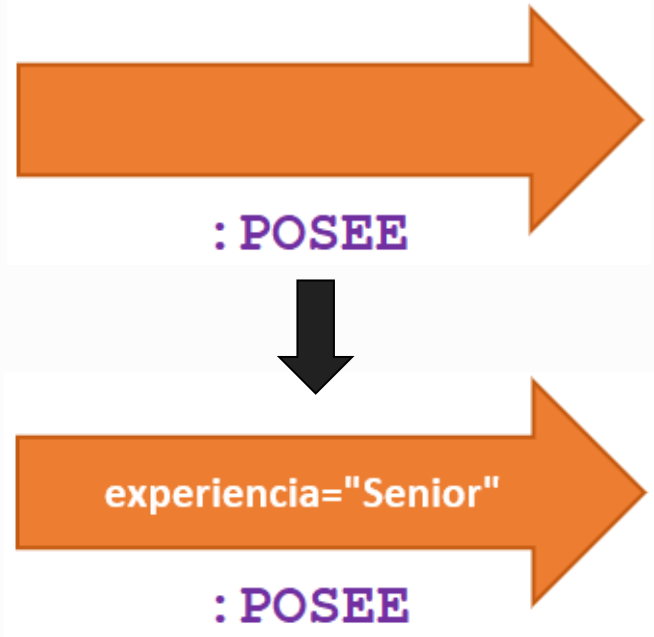
```
MATCH (nodo:etiqueta {propiedades})-[rel:tipo {propiedades}]- (nodo)
WHERE condiciones
SET rel.propiedad=valor
```

Permite agregar una nueva propiedad a una relación ó modificar el valor de una propiedad existente en una relación.

Ejemplo

Agregar Propiedad **experiencia** con valor **Senior** a la Relación **POSEE** para **Claudio Pereyra** con el conocimiento **Análisis Funcional .**

```
MATCH (p:Persona {nombre:"Claudio", apellido: "Pereyra"})
      -[r:POSEE]->
      (c:Conocimiento {nombre:"Análisis Funcional"})
SET r.experiencia="Senior"
```



Modificar Propiedad de una Relación

Sintaxis

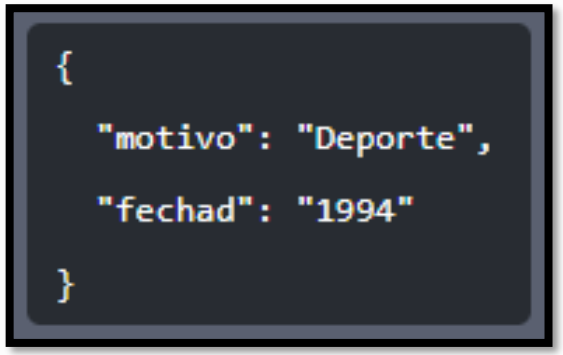
```
MATCH (nodo:etiqueta {propiedades})-[rel:tipo {propiedades}]- (nodo)
WHERE condiciones
SET rel.propiedad=valor
```

Permite agregar una nueva propiedad a una relación ó modificar el valor de una propiedad existente en una relación.

Ejemplo

Modificar en la Relación **CONOCE_A** , su propiedad "**motivo",** el cual posee valor es "**Facultad"** para la persona con id 3380 que CONOCE_A **Mario López.**

```
MATCH (p:Persona)-[r:CONOCE_A]-> (o:Persona {nombre:"Mario", apellido:"López"}
WHERE id(p)=3380
SET r.motivo="Deporte"
```



Eliminar Propiedades de un Nodo

Sintaxis

```
MATCH (nodo:etiqueta {propiedades})
WHERE condiciones
REMOVE nodo.propiedad
```

Permite eliminar una propiedad de un nodo.

Ejemplo 1

Eliminar la propiedad edad de Claudio Pereyra.

```
MATCH (a:Persona {nombre:"Claudio", apellido: "Pereyra"})
REMOVE a.edad
RETURN a
```

Eliminar Propiedades de una Relación

Sintaxis

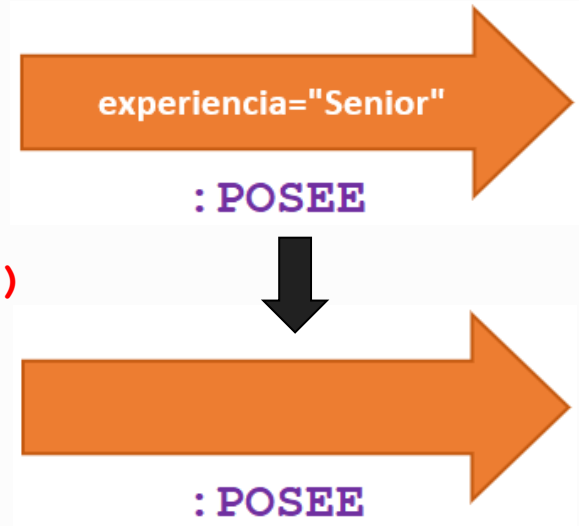
```
MATCH (nodo:etiqueta {propiedades})-[rel:tipo {propiedades}]- (nodo)
WHERE condiciones
REMOVE rel.propiedad=valor
```

Permite eliminar una propiedad de una relación

Ejemplo

Eliminar Propiedad **experiencia** con valor **Senior** a la Relación **POSEE** para **Claudio Pereyra** con el conocimiento **Análisis Funcional .**

```
MATCH (p:Persona {nombre:"Claudio", apellido: "Pereyra"})
  -[r:POSEE]-> (c:Conocimiento {nombre:"Análisis Funcional"})
  REMOVE r.experiencia
```

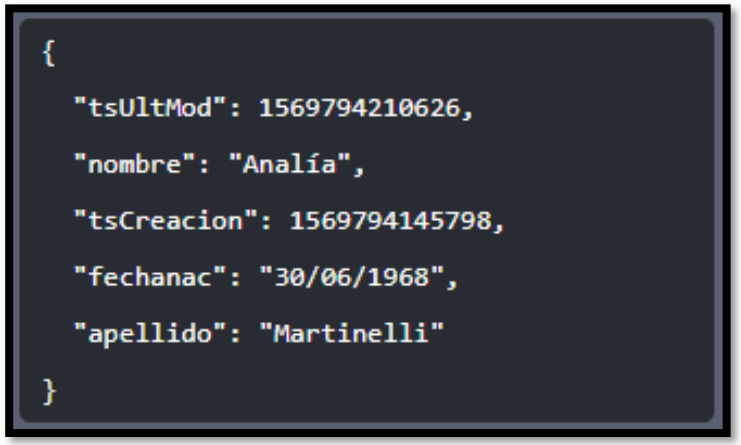


Crear o Modificar un Nodo

COMANDO MERGE(nodo)

Sintaxis

```
MERGE ...
ON CREATE SET ...
ON MATCH SET ...
```



Ejemplo

Crear un nodo para Analía Martinelli sólo si no existe, con fecha de nacimiento 30/06/1968 y la propiedad tsCreacion. Si existe, setear esta fecha 30/06/1968 y la propiedad tsUltMod.

```
MERGE (n:Persona {nombre: "Analía", apellido: "Martinelli"})
ON CREATE SET n.fechanac="30/06/1968",n.tsCreacion=timestamp()
ON MATCH SET n.fechanac="30/06/1968",n.tsUltMod=timestamp()
RETURN n
```

Crear o Modificar una Relación

COMANDO MERGE(relación)

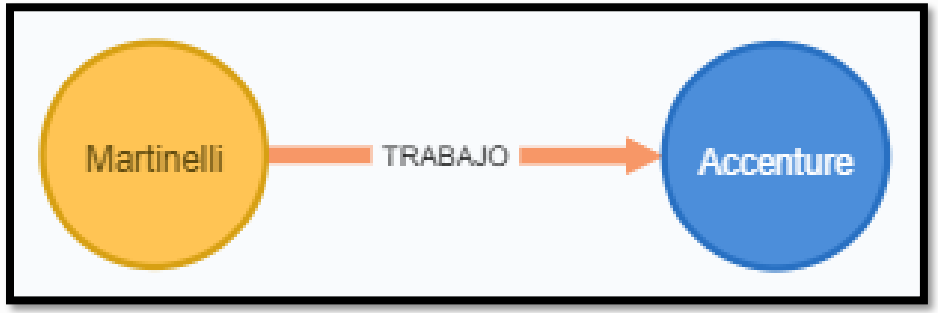
Sintaxis

```
MERGE ...
ON CREATE SET ...
ON MATCH SET ...
```

Ejemplo

Crear la relación TRABAJO si no existe para la Persona Martinelli con la empresa Accenture.

```
MATCH (p:Persona {apellido: "Martinelli"}),(e:Empresa {nombre:"Accenture"})
MERGE (p)-[r:TRABAJO]->(e)
RETURN p, r, e
```



Crear o Modificar una Relación

COMANDO MERGE(relación)

Ejemplo

Crear o modificar en caso de que exista una relación de TRABAJO para la persona con Apellido Martinelli y la empresa Accenture, actualizando la fechad y fechah de la relación. Actualizando las propiedades tsCreacion ó tsUltMod según corresponda.

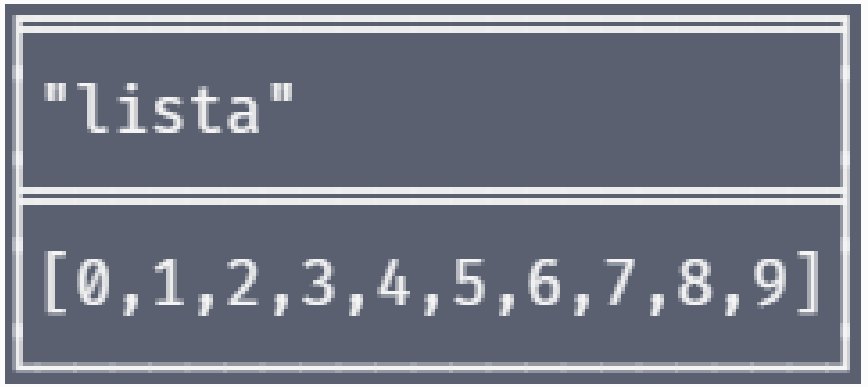
```
MATCH (p:Persona {apellido: "Martinelli"}),(e:Empresa {nombre:"Accenture"})
MERGE (p)-[r:TRABAJO]->(e)
ON CREATE SET r.fechad="20130501",r.fechah="20170630", r.tsCreacion=timestamp()
ON MATCH SET r.fechad="20130501",r.fechah="20170630", r.tsUltMod=timestamp()
RETURN p, r, e
```

Listas o Colecciones

Listas

Una lista se crea usando corchetes y dentro de los mismos separando los elementos por comas.

```
RETURN [0,1,2,3,4,5,6,7,8,9] AS lista
```



Listas

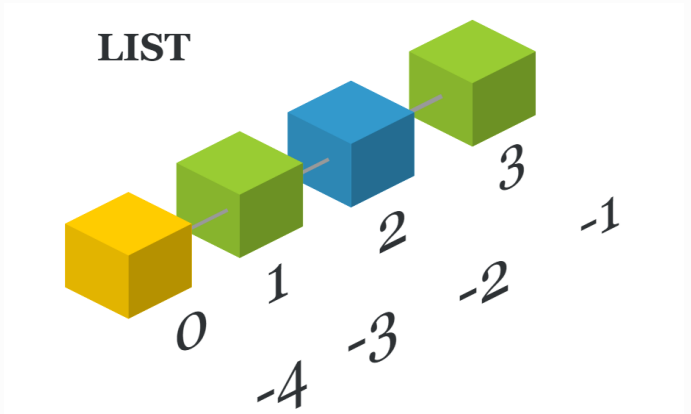
Función range: devuelve una lista que comprende todos los valores enteros dentro de un rango limitado por un valor inicial (start) y un valor final (end), donde la diferencia o salto (step) entre dos valores cualesquiera consecutivos es constante, por ejemplo una progresión aritmética. Para crear rangos con valores enteros decrecientes, usar un valor negativo de salto. Se devuelve un rango vacío si el valor del salto es negativo y start - end son positivos, o viceversa, por ejemplo range (0, 5, -1).

Sintaxis

```
range(start, end)[step]
```

```
RETURN range(0,10)
```

"range(0,10)"
[0,1,2,3,4,5,6,7,8,9,10]



Retornar un elemento determinado o un rango de elementos:

```
RETURN range(0,10)[4]
```

"range(0,10)"
[0,1,2,3,4,5,6,7,8,9,10]

"range(0,10)[4]"
4

```
RETURN range (0,10)[-2]
```

"range(0,10)"
[0,1,2,3,4,5,6,7,8,9,10]

"range (0,10)[-2]"
9

Listas

Función range

```
RETURN range(0,10)[-5..]

RETURN range (0,10)[..3]

RETURN range (0,10,2)
```

"range(0,10)[-5..]"
[6,7,8,9,10]

"range (0,10)[..3]"
[0,1,2]

"range (0,10,2)"
[0,2,4,6,8,10]

Retorna una lista de valores en un rango, con un determinado salto, ante omisión el valor de salto es 1.

"range(0,10)"
[0,1,2,3,4,5,6,7,8,9,10]

Listas de Comprensión

Son construcciones que dispone Cypher para crear listas en función a otras listas.
Ejemplo de una lista de 0 a 10, tomo los múltiplos de tres y los elevo al cuadrado generando una nueva lista.

```
RETURN [x IN range(0,10) WHERE x % 3 = 0 | x^2] AS resultado
```

"resultado"
[0,3,6,9]

"resultado"
[0.0,9.0,36.0,81.0]

DBlandIT

Listas

Sintaxis

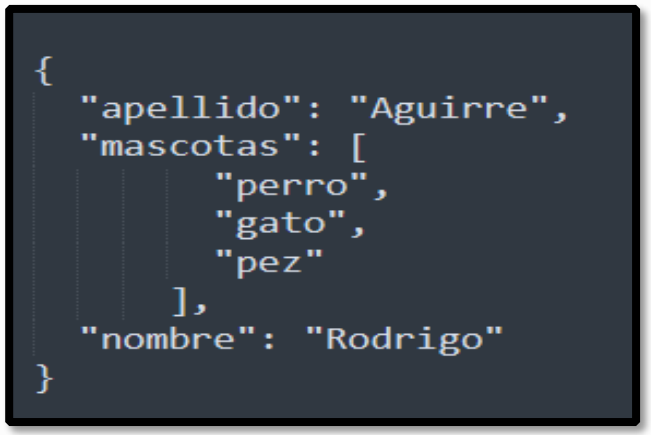
```
MATCH (nodo:etiqueta {propiedades})
WHERE condiciones
SET nodo.propiedad=[valor1, valor2, valorN]
```

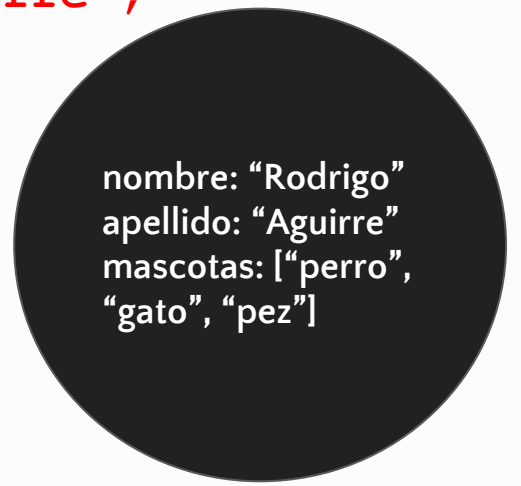
Permite agregar una nueva propiedad de tipo lista a un nodo o modificar el valor de una propiedad existente en un nodo.

Ejemplo 1

Crear un nodo Persona de nombre Rodrigo, apellido Aguirre y que tenga de mascota un perro, un gato y un pez

```
CREATE(a:Persona {nombre:"Rodrigo", apellido: "Aguirre",
mascotas:["perro", "gato", "pez"]})
RETURN a
```





Listas

Sintaxis

```
MATCH (nodo:etiqueta {propiedades})
WHERE condiciones
SET nodo.propiedad=[valor1, valor2, valorN]
```

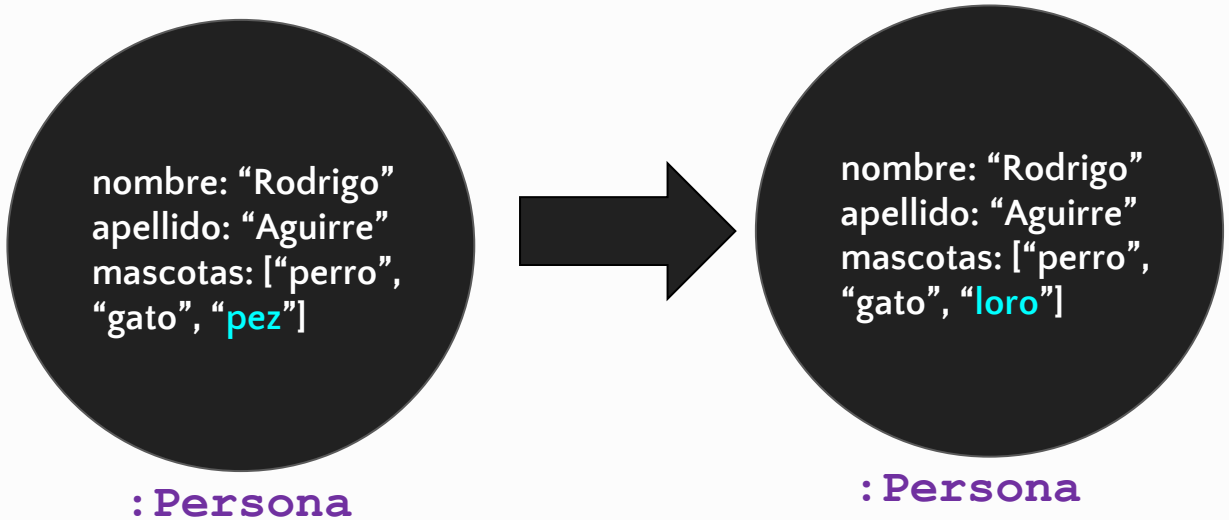
Permite agregar una nueva propiedad de tipo lista a un nodo o modificar el valor de una propiedad existente en un nodo.

Ejemplo 1

Cambiar las mascotas de Rodrigo Aguirre para que tenga un perro, un gato y un loro

```
MATCH(a:Persona {nombre:"Rodrigo", apellido: "Aguirre"})
SET a.mascotas=["perro", "gato", "loro"]
RETURN a
```


```
{
  "apellido": "Aguirre",
  "mascotas": [
        "perro",
        "gato",
        "loro"
    ],
  "nombre": "Rodrigo"
}
```

Listas

Patrones de Comprensión

Son construcciones sintácticas que dispone Cypher para crear listas en función del matcheo de patrones.

Ejemplo

Devuelve la lista de rubros de las empresas en las que haya trabajado una persona de apellido Corrado.

```
MATCH (p:Persona { apellido: 'Corrado' })
RETURN [(p)-[:TRABAJO]->(e:Empresa) | e.rubro] AS rubrosTrabajados
```

"rubrosTrabajados"
["Banco","Telefonía","Consultoría"]

DBlandIT

Funciones de Listas

Función keys()

keys()

- Retorna una lista con todos las propiedades de un nodo o relación dados.

```
keys(nodo) | keys(relacion)
```

Ejemplos

```
MATCH (a:Empresa {nombre:"IBM"})
RETURN keys(a)
```

keys(a)
["id", "tamano", "nombre", "rubro", "ubicacion"]

```
MATCH ()-[r]->(b:Empresa {nombre:"IBM"})
RETURN keys(r), type(r)
```

keys(r)	type(r)
["fechah", "fechad", "puesto", "sueldo"]	"TRABAJO"

DBlandIT

Funciones de Listas

Función nodes()

- Retorna una lista con todos los nodos de camino definido.
  nodes(camino)

nodes()

```
MATCH p =(a)-->(b)-->(c)
WHERE a.apellido = 'Corrado'
AND c.apellido = 'López'
RETURN nodes(p)
```

```
MATCH p =(a)-->(b)-->(c)
WHERE a.apellido = 'Corrado'
AND c.apellido = 'López'
RETURN p
```

"nodes(p)"
[{"fechanac":"01/08/1966","apellido":"Corrado","nombre":"Gustavo","email":"gustavo.corrado@gmail.com","pais":"Argentina"},{"fechanac":"27/09/1980","apellido":"Lupis","nombre":"Jorge","email":"jlup@gmail.com","pais":"Argentina"},{"fechanac":"11/02/1970","apellido":"López","nombre":"Mario","email":"mario.lopez@gmail.com","pais":"Argentina"}]

p
(:Persona {fechanac: "01/08/1966",apellido: "Corrado",nombre: "Gustavo",email: "gustavo.corrado@gmail.com",pais: "Argentina"})-[:CONOCE_A {motivo: "Estudio",fechad: "1994"}]->(:Persona {fechanac: "27/09/1980",apellido: "Lupis",nombre: "Jorge",email: "jlup@gmail.com",pais: "Argentina"})-[:CONOCE_A {motivo: "Estudio",fechad: "1994"}]->(:Persona {fechanac: "11/02/1970",apellido: "López",nombre: "Mario",email: "mario.lopez@gmail.com",pais: "Argentina"})

Funciones de Listas

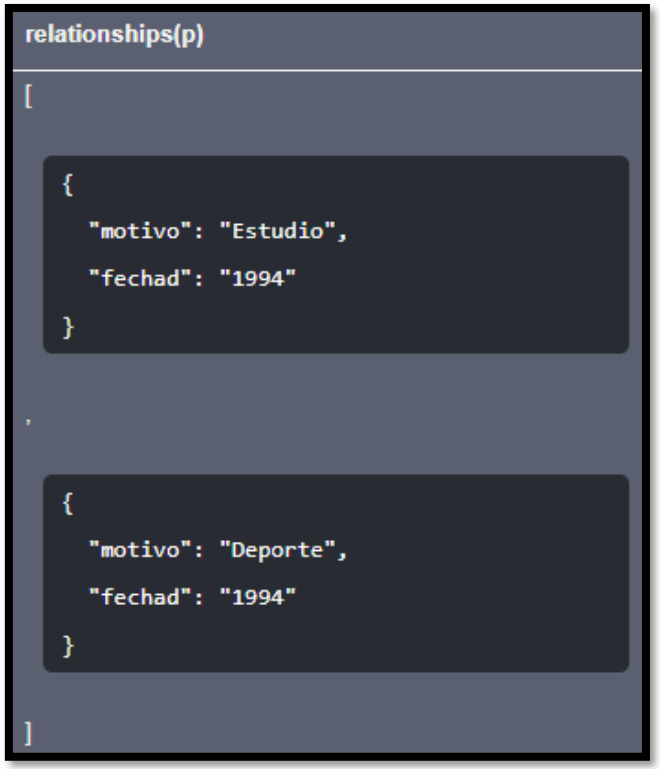
Función relationships()

`relationships()`

- Retorna una lista con todas las relaciones de camino definido.

`relationships(camino)`

```
MATCH p =(a)-->(b)-->(c)
WHERE a.apellido = 'Corrado'
AND c.apellido = 'López'
RETURN relationships(p)
```



Funciones de Listas

Función labels()

labels()

- Retorna una lista con todas las etiquetas de un nodo dado.

```
labels(nodo)
```

```
MATCH (a:Persona {apellido:"Martinelli"}) SET a:Empleado:Estudiante:Docente

MATCH (a:Persona {apellido:"Martinelli"}) RETURN labels(a)
```

labels(a)
["Persona", "Empleado", "Estudiante", "Docente"]

Funciones de Listas

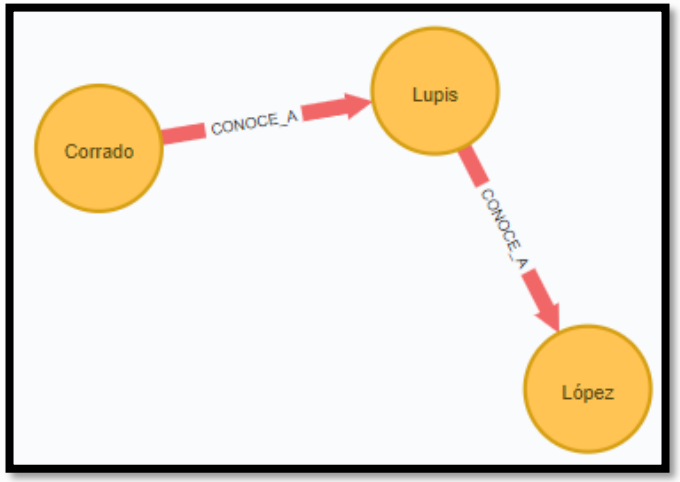
Función reduce()

`reduce()`

- Retorna el resultado de aplicar una expresión sobre cada elemento de la lista en conjunto.

```
reduce(accumulator = initial, variable IN list | expression)
```

```
MATCH p =(a)-->(b)-->(c)
WHERE a.apellido = 'Corrado'
AND c.apellido = 'López'
RETURN reduce(cantidad = 0, n IN nodes(p) | cantidad + 1) AS cantPersonas
```



cantPersonas
3

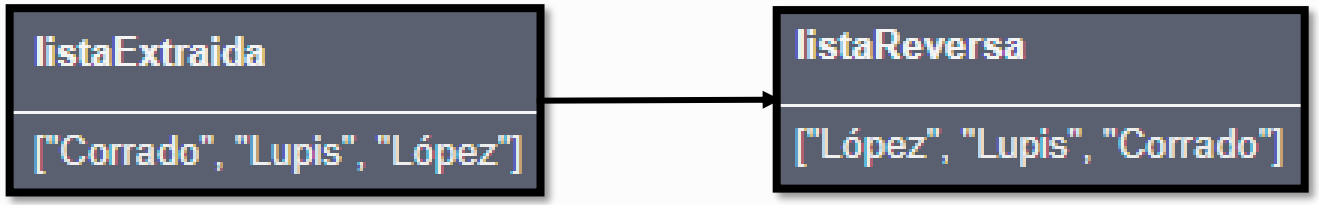
Funciones de Listas

Función reverse() Retorna una lista con los elementos en orden inverso a la lista original

`reverse()`

Ejemplo:

```
MATCH p =(a)-->(b)-->(c)
WHERE a.apellido = 'Corrado' AND c.apellido='López'
WITH [n IN nodes(p)| n.apellido] AS listaExtraida
RETURN reverse(listaExtraida) AS listaReversa
```



Funciones de Listas

Función tail() Retorna una lista con todos sus elementos, exceptuando al primero

```
tail()
```

Ejemplo:

```
MATCH p =(a)-->(b)-->(c)
WHERE a.apellido = 'Corrado' AND c.apellido='López'
WITH [n IN nodes(p)| n.apellido] AS listaExtraida
RETURN tail(listaExtraida) AS listaTail
```

Funciones Predicado

Funciones Predicado

Retornan Verdadero o Falso para los argumentos dados

`all(), any(), exists(), none(), single()`

```
all()       - Todos los ítems de la lista cumplen con el predicado.

any()       - Algún Ítem de la lista cumple con el predicado.

exists()    - Evalúa la existencia de un atributo o propiedad.

none()      - Ningún ítem de la lista cumple con el predicado.

single()    - Sólo un ítem de la lista cumple con el predicado.
```

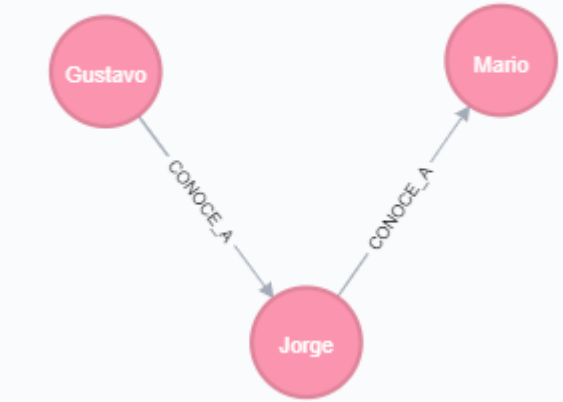
Funciones Predicado

Función all()

Testea que el predicado sea Verdadero para todos los elementos de una lista.

Argumentos

lista: Una expresión que retorna una lista.
variable: Esta variable será utilizada dentro del predicado.
predicado: Este predicado es testeado contra todos los ítems en la l




```
MATCH lista =(a:Persona)-[*1..2]->(b:Persona)
WHERE a.nombre = 'Gustavo' AND b.nombre = 'Mario'
AND ALL (varNodo IN nodes(lista) WHERE varNodo.pais = "Argentina")
RETURN lista
```

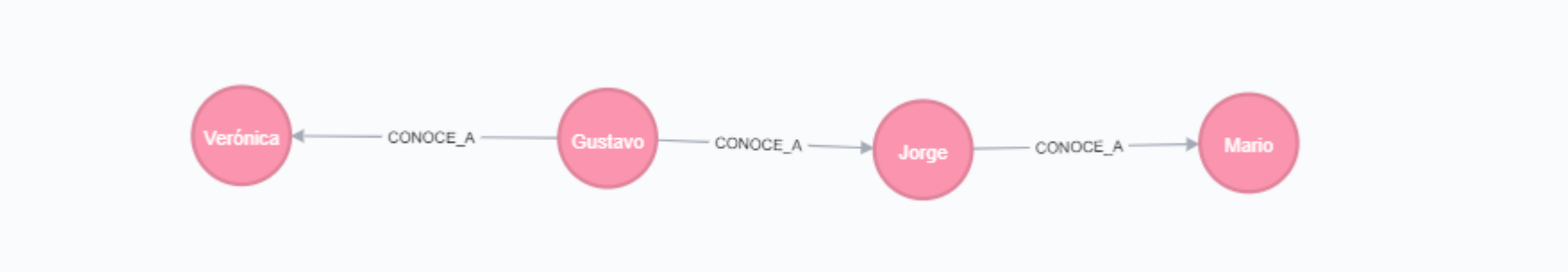
DBlandIT

Funciones Predicado

Ejemplo 2 - Función any()

Testea que el predicado sea Verdadero para al menos un elemento de una lista.

```
MATCH lista =(a:Persona)-[*1..2]->(b:Persona)
WHERE a.nombre = 'Gustavo' AND b.nombre IN ['Mario','Mariana','Verónica']
AND ANY (varNodo IN nodes(lista) WHERE varNodo.pais = 'Argentina')
RETURN lista
```

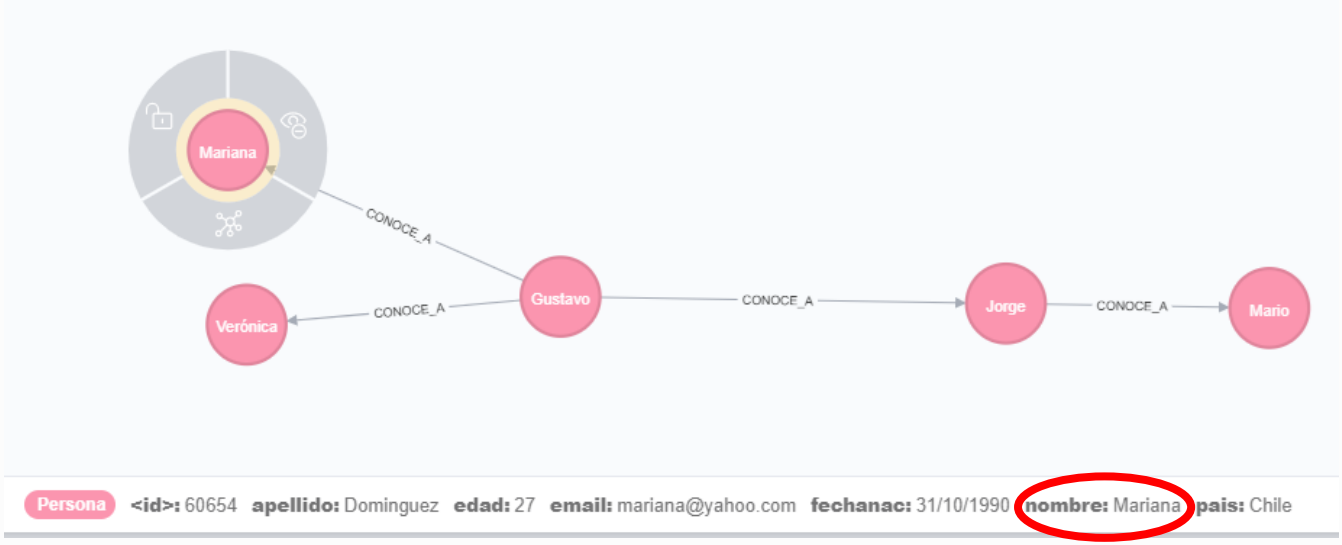


Funciones Predicado

Ejemplo 2 - Función any()

Testea que el predicado sea Verdadero para alguno de los elementos de una lista.

```
MATCH lista =(a:Persona)-[*1..2]->(b:Persona)
WHERE a.nombre = 'Gustavo' AND b.nombre IN ['Mario','Mariana','Verónica']
AND ANY (varNodo IN nodes(lista) WHERE varNodo.pais = 'Argentina')
RETURN lista
```



Funciones Predicado

Ejemplo 3 - Función exists()

Modificamos la persona con apellido Dominguez agregandole un campo edad con valor 27

```
MATCH (n:Persona {apellido:"Dominguez"}) SET n.edad=27 RETURN n
```

Buscamos todos los nodos Persona en los que exista el atributo edad.

```
MATCH (n:Persona)
WHERE (n.edad) IS NOT NULL
RETURN n.nombre, n.apellido, n.edad
```

n.nombre	n.apellido	n.edad
"Mariana"	"Dominguez"	27
"Claudio"	"Pereyra"	39
"Lis"	null	25

Funciones Predicado

Ejemplo 4 -Operador NOT y Función exists()

Buscamos todos los nodos Persona en los que No exista el atributo edad.

```
MATCH (n:Persona)
WHERE (n.edad) IS NULL
RETURN n.nombre, n.apellido
```

n.nombre	n.apellido
"Alejandro"	"Ramirez"
"Analía"	"Martinelli"
"Gustavo"	"Corrado"
"Analía"	"Díaz"
"Mario"	"López"
"Natalia"	"Ferreira"
"Eduardo"	"García"
"Bibiana"	"González"
"Jorge"	"Lupis"
"Verónica"	"Mendez"

Mapas

Mapas

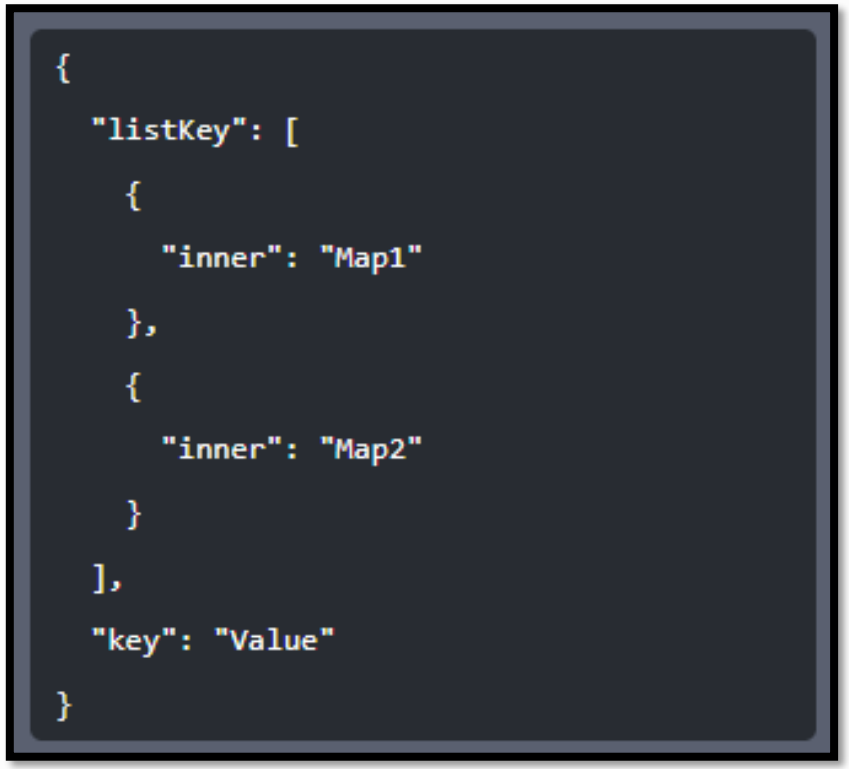
Mapas de Proyección

Cypher soporta el concepto de Mapas de Proyección, permitiendo construir mapas de proyección para valores de nodos, relaciones u otros mapas.

Ejemplo

```
RETURN
{ key: 'Value',
  listKey: [
              { inner: 'Map1' },
              { inner: 'Map2' }
            ]
}
```

```
{"listKey":[{"inner":"Map1"},{"inner":"Map2"}],"key":"Value"}
```



Mapas

Mapas de Proyección

Ejemplo:

```
MATCH (p:Persona)-[:POSEE]->(c:Conocimiento)
WITH p, count(c) AS cantConocimientos
RETURN p {.apellido, cantConocimientos}
```

El comando devuelve un JSON con un mapa de valores con el apellido y la cantidad de conocimientos de cada persona.

"p"
{"apellido":"Corrado","cantConocimientos":5}
{"apellido":"Díaz","cantConocimientos":7}
{"apellido":"Dominguez","cantConocimientos":4}
{"apellido":"Pereyra","cantConocimientos":3}
{"apellido":"Ferreira","cantConocimientos":3}
{"apellido":"García","cantConocimientos":2}
{"apellido":"Lupis","cantConocimientos":6}
{"apellido":"Mendez","cantConocimientos":3}

Mapas

Mapas de Proyección

Ejemplo

```
MATCH (p:Persona)-[:POSEE]->(c:Conocimiento)
WITH p, count(c) AS cantConocimientos
RETURN p {.*, cantConocimientos}
```

El comando devuelve un JSON con un mapa de valores todos las propiedades y la cantidad de conocimientos de cada persona.

p


```
{
  "fechanac": "01/08/1966",
  "apellido": "Corrado",
  "cantConocimientos": 5,
  "nombre": "Gustavo",
  "email": "gustavo.corrado@gmail.com",
  "pais": "Argentina"
}
```

```
{
  "fechanac": "10/12/1978",
  "apellido": "Díaz",
  "cantConocimientos": 7,
  "nombre": "Analía",
  "email": "adiaz@hotmail.com",
  "pais": "Argentina"
```


Otras Cláusulas

OPTIONAL MATCH

CLAUSULA OPTIONAL MATCH

Esta cláusula es similar a un Left Join de SQL. En el caso que una persona no matchee con otros nodos o no posea relaciones, el comando retornará un valor NULO.

Ejemplo 1:

```
MATCH (a:Persona { apellido: 'Ramirez' })
OPTIONAL MATCH (a)-->(x)
RETURN a.apellido, a.nombre, x
```

a.apellido	a.nombre	x
"Ramirez"	"Alejandro"	null

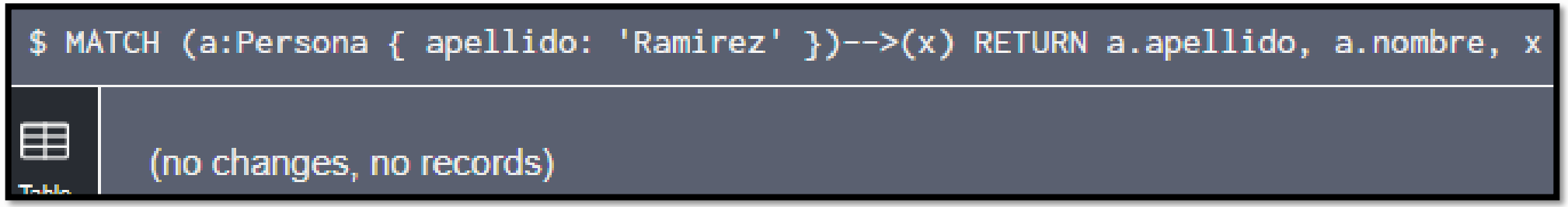
OPTIONAL MATCH

CLAUSULA OPTIONAL MATCH (Cont.)

Como se observa en este contraejemplo, al no existir otros nodos o relaciones que macheen con la persona de apellido Ramírez, el comando no devuelve registros.

Contra ejemplo 1

```
MATCH (a:Persona { apellido: 'Ramirez' })-->(x)
RETURN a.apellido, a.nombre, x
```



OPTIONAL MATCH

CLAUSULA OPTIONAL MATCH (Cont.)

Esta cláusula es similar a un Left Join de SQL.
En este caso que un conocimiento NUNCA poseerá algo, pero el comando igualmente lo devuelve con su valor en NULO.

Ejemplo 2

```
MATCH (a:Conocimiento {nombre:"Java"})
OPTIONAL MATCH (a)-[r:POSEE]->()
RETURN a.nombre, r
```

a.nombre	r
"Java"	null

WITH

CLAUSULA WITH

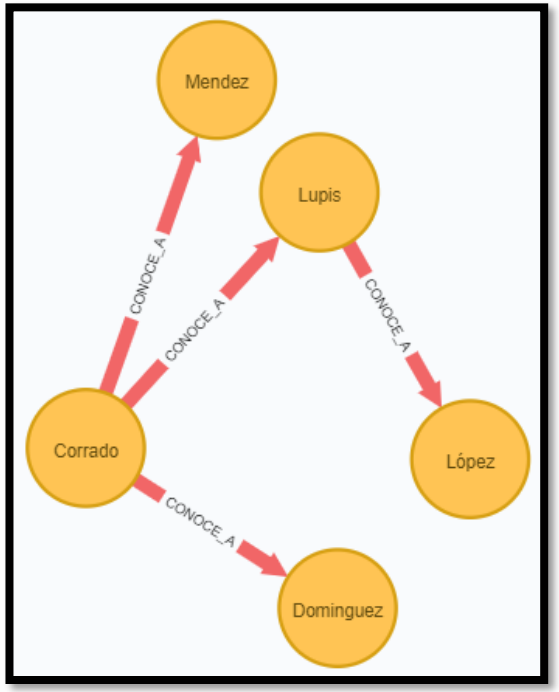
Este ejemplo del WITH es similar al GROUP BY y HAVING SQL.

Ejemplo 1

Obtener nombre y apellido de las personas que posean más de 5 relaciones salientes.

```
MATCH (p:Persona)-->()
WITH p, count(*) AS relaciones
WHERE relaciones > 5
RETURN p.nombre,p.apellido,relaciones
```

p.nombre	p.apellido	relaciones
"Gustavo"	"Corrado"	12
"Analía"	"Díaz"	9
"Mariana"	"Dominguez"	7
"Claudio"	"Pereyra"	7
"Jorge"	"Lupis"	10



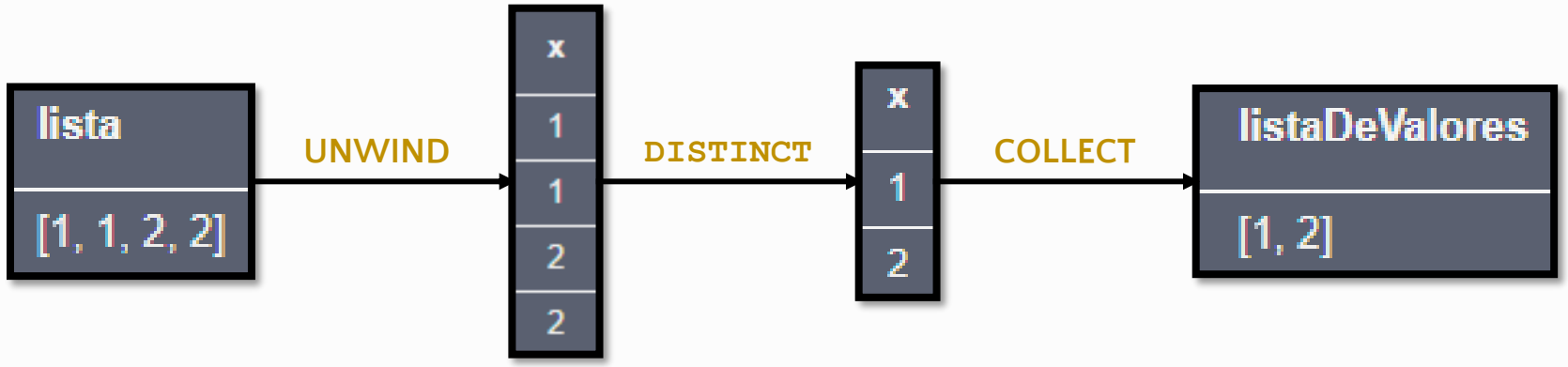
WITH y UNWIND

CLAUSULAS WITH y UNWIND / Función COLLECT

- La cláusula **UNWIND** aplana un array en múltiples filas.
- La función **COLLECT** toma una serie de valores y arma una Lista con los mismos.

Ejemplo 2

```
WITH [1, 1, 2, 2] AS lista
UNWIND lista AS x  // aplana la lista de valores en cuatro filas
WITH DISTINCT x    // Toma valores únicos por cada duplicado
RETURN collect(x) AS listaDeValores // Devuelve una nueva lista de valores
```



WITH y UNWIND

CLAUSULAS WITH y UNWIND / Función COLLECT

- La cláusula **UNWIND** aplana un array en múltiples filas.
- La función **COLLECT** toma una serie de valores y arma una Lista con los mismos.

Ejemplo 3
Lista de las personas que tienen más de 10 relaciones salientes.

```
MATCH (a:Persona)-->()
WITH a.apellido AS apellidoPersona
WITH apellidoPersona, count(*) AS cantRelaciones
WHERE cantRelaciones >= 10
RETURN collect(apellidoPersona) AS personasConRelaciones
```

personasConRelaciones
["Díaz", "Corrado", "Lupis"]

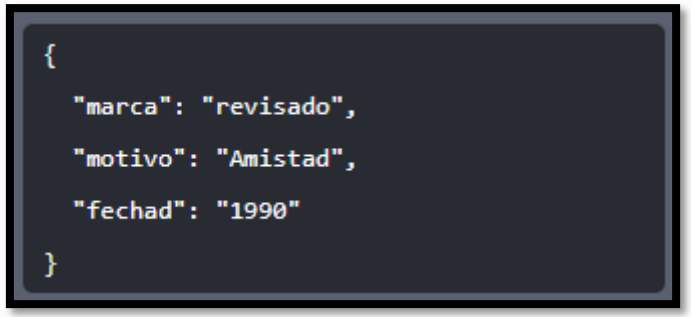
FOREACH

CLAUSULA FOREACH

La cláusula **FOREACH** es usada para modificar datos de una lista llamada **grafo1**, ya sean elementos de la misma o el resultado de funciones agregadas

Ejemplo 1

```
MATCH grafo1 =(p1:Persona)-[CONOCE_A]->(p2:Persona)
WHERE p1.apellido='Corrado' AND p2.apellido STARTS WITH 'D'
FOREACH (elem IN nodes(grafo1)| SET elem.marca = "actualizado")
```



FOREACH

CLAUSULA FOREACH

Ejemplo 2

```
MATCH lista =(p1:Persona)-[CONOCE_A]->(p2:Persona)
WHERE p1.apellido='Corrado' AND p2.apellido STARTS WITH 'D'
FOREACH (elem IN relationships(lista)| SET elem.marca = 'revisado')
```

```
"lista"
```

```
[{"marca":"actualizado","nombre":"Gustavo","email":"gustavo.corrado@gmail.com","fechanac":"01/08/1966","
apellido":"Corrado","pais":"Argentina"},{"marca":"revisado","motivo":"Amistad","fechad":"1990"},{"marca"
:"actualizado","fechanac":"31/10/1990","apellido":"Dominguez","edad":27,"nombre":"Mariana","email":"mari
ana@yahoo.com","pais":"Chile"}]
```

DBlandIT

Consultas Complejas

Consultas Complejas

Ejemplo 1

Lista de personas que trabajan o trabajaron en empresas en las que trabajó una persona con apellido "Corrado", pero que no son sus contactos, para así poder sugerirle a él nuevos contactos.

```
MATCH (a:Persona)-[:TRABAJO]->()<-[:TRABAJO]-(b:Persona {apellido:"Corrado"})
WHERE id(a) <> id(b)
AND NOT (a)-[:CONOCE_A]-(b)
RETURN DISTINCT a.nombre AS nombreSugerido, a.apellido AS apellidoSugerido,
b.nombre, b.apellido
```

nombreSugerido	apellidoSugerido	b.nombre	b.apellido
"Mario"	"López"	"Gustavo"	"Corrado"
"Natalia"	"Ferreira"	"Gustavo"	"Corrado"
"Analía"	"Martinelli"	"Gustavo"	"Corrado"

Consultas Complejas

Ejemplo 2

Ranking de los primeros 2 conocimientos que poseen más personas egresadas de la carrera "Ing en Sistemas de Información“

```
MATCH (a:Persona)-[:POSEE]->(c:Conocimiento)
MATCH (a)-[e:ESTUDIO]->(d:Carrera {nombre: "IngenSistemasdeInformación"})
WHERE e.estado = "Completo"
WITH d.nombre as carrera, c.nombre as conocim, count(DISTINCT a) AS cantidad
RETURN carrera, conocim, cantidad
```

carrera	conocim	cantidad
"IngenSistemasdeInformación"	"Java"	2
"IngenSistemasdeInformación"	"PHP"	1
"IngenSistemasdeInformación"	"C++"	1
"IngenSistemasdeInformación"	"SQL"	1
"IngenSistemasdeInformación"	"AnálisisFuncional"	2
"IngenSistemasdeInformación"	"Revisiónyelaboracióndecontratos"	1

Consultas Complejas

Ejemplo 3
Lista de conocimientos que poseen las personas que estudiaron en cada carrera.

```
MATCH (a:Persona)-[:POSEE]->(c:Conocimiento), (a)-[:ESTUDIO]->(d:Carrera)
RETURN d.nombre, c.nombre, count(distinct a) AS cantidad
ORDER by d.nombre, cantidad DESC
```

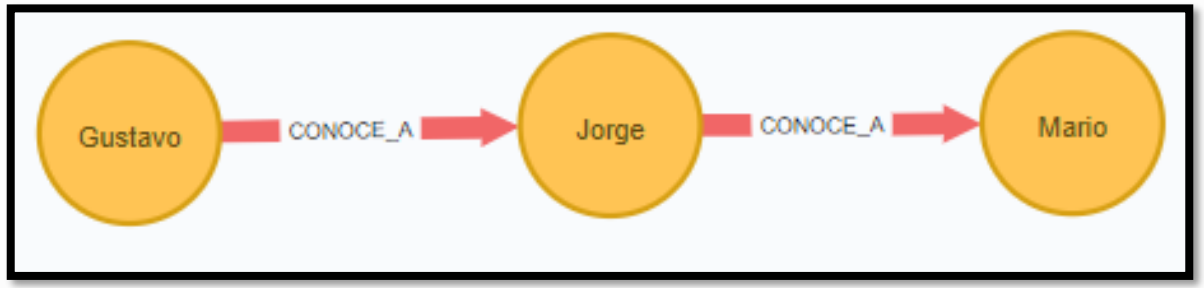
d.nombre	c.nombre	cantidad
"Derecho"	"Contestacióndedemandas"	1
"Derecho"	"Creacióndesociedades"	1
"Derecho"	"Revisiónyelaboracióndecontratos"	1
"Derecho"	"Gestión"	1
"IngenSistemasdeInformación"	"Java"	3
"IngenSistemasdeInformación"	"AnálisisFuncional"	3
"IngenSistemasdeInformación"	"PHP"	1
"IngenSistemasdeInformación"	"C++"	1
"IngenSistemasdeInformación"	"SQL"	1
"IngenSistemasdeInformación"	"Gestióndeproyectos"	1
"IngenSistemasdeInformación"	"Revisiónyelaboracióndecontratos"	1

Consultas Complejas

Ejemplo 4

Obtener el camino más corto en la relación CONOCE_A entre Gustavo y Mario, para cualquier dirección y longitud.

```
MATCH (a {nombre:"Gustavo"}), (b {nombre:"Mario"}),
camino=shortestPath((a)-[:CONOCE_A*]->(b))
RETURN camino
```



Consultas Complejas

Ejemplo 5

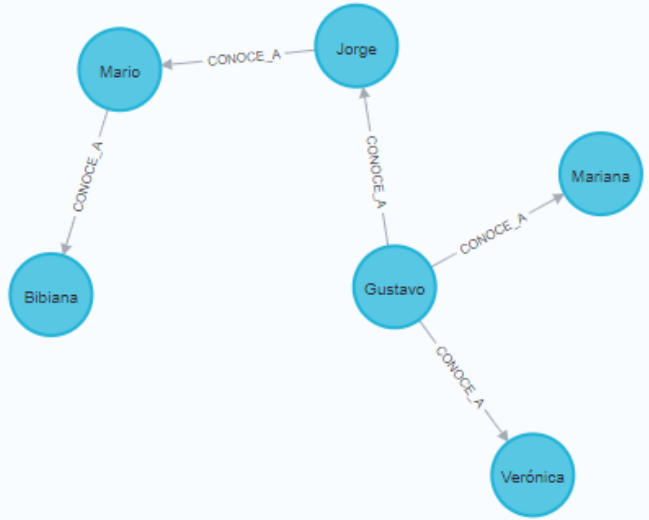
Lista de conocimientos que poseen una persona y que también la poseen otras personas que no son sus contactos, para sugerirle nuevos contactos.

```
MATCH (a:Persona)-[:POSEE]->(d)<-[:POSEE]-(b:Persona)
WHERE id(a) <> id(b)
AND NOT (a)-[:CONOCE_A]-(b)
RETURN DISTINCT a.nombre, a.apellido,d.nombre, b.nombre, b.apellido
```

a.nombre	a.apellido	d.nombre	b.nombre	b.apellido
"Claudio"	"Pereyra"	"Java"	"Gustavo"	"Corrado"
"Analía"	"Díaz"	"Java"	"Gustavo"	"Corrado"
"Analía"	"Díaz"	"PHP"	"Gustavo"	"Corrado"
"Eduardo"	"García"	"C++"	"Gustavo"	"Corrado"
"Analía"	"Díaz"	"C++"	"Gustavo"	"Corrado"
"Analía"	"Díaz"	"SQL"	"Gustavo"	"Corrado"
"Eduardo"	"García"	"SQL"	"Gustavo"	"Corrado"
"Claudio"	"Pereyra"	"AnálisisFuncional"	"Gustavo"	"Corrado"

Consultas Complejas

Ejemplo 6
Ranking de las 3 personas más populares: las que son conocidas por más personas en la red **(suponiendo que el conocimiento NO es mutuo)**




```
MATCH (a:Persona)-[:CONOCE_A]->(b:Persona)
WITH a as persona, count(b) AS cantidad
RETURN persona, cantidad
ORDER BY cantidad DESC LIMIT 3;
```

"persona"	"cantidad"
{"fechanac":"01/08/1966","apellido":"Corrado","nombre":"Gustavo","email":"gustavo.corrado@gmail.com","pais":"Argentina"}	3
{"fechanac":"11/02/1970","apellido":"López","nombre":"Mario","email":"mario.lopez@gmail.com","pais":"Argentina"}	1
{"fechanac":"27/09/1980","apellido":"Lupis","nombre":"Jorge","email":"jlup@gmail.com","pais":"Argentina"}	1

DBlandIT

info@dblandit.com
+54 11 3889-4009
www.dblandit.com/