Universidad Politécnica Salesiana

Nombre: Fernando Sanchez

Materia: Sistemas Expertos

Prueba 2

Fecha: 31/01/202

Tema:

Basados en casos.

- Diseñe y desarrolle un algoritmo Knn en Neo4j para:
  - Fila B – 1: Este es un conjunto de datos de empleados en una empresa y el resultado es estudiar sobre la deserción de los empleados, para ello se debe descargar los datos del siguiente link: http://smalldatabrains.com/wp-content/uploads/2018/03/data.csv [2].

Importar la API pandas para la manipulación y el análisis de datos para procesar los datos del documento excel "data.csv"

import pandas

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv("data.csv",sep=';')
lista = [list(row) for row in df.values]
### Imprimir el tamaño de la lista (datos obtenidos del archivo
print("Longitud de la lista: "+str(len(lista)))
```

```
Longitud de la lista: 14999
```

Importar la API py2neo

para el ingreso de los datos que se encuentran dentro de la lista

Conexión con Neo4j

Configure la URL de conexión con la base de datos de Ne04j:

graph = Graph("bolt://localhost:7687", aut="neo4j", password="Prueba2", secure=False)

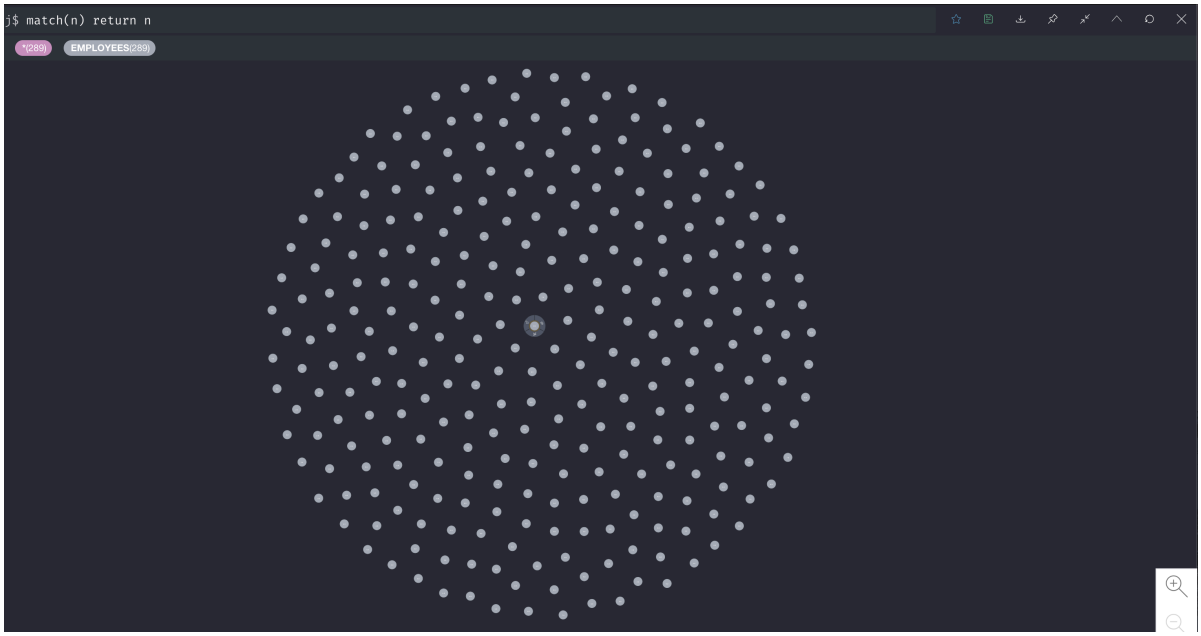
In [3]:
```
#IMPORTAR py2neo
from py2neo import Node, Relationship, Graph


# connect to authenticated graph database
graph = Graph("bolt://localhost:7687", aut="neo4j", password="P
```

In [4]:
```
print(lista)
```

```
0.99, 5.0, 262.0, 5.0, 0.0, 0.0, 1.0], [0.56, 0.71, 4.0, 296.0, 2.
0, 0.0, 0.0, 1.0], [0.44, 0.56, 2.0, 158.0, 3.0, 0.0, 0.0, 1.0], [
0.31, 0.56, 4.0, 238.0, 2.0, 0.0, 0.0, 1.0], [0.77, 0.93, 4.0, 231
.0, 5.0, 0.0, 0.0, 1.0], [0.44, 0.45, 2.0, 156.0, 3.0, 0.0, 0.0, 1
.0], [0.38, 0.46, 2.0, 145.0, 3.0, 0.0, 0.0, 1.0], [0.45, 0.48, 2.
0, 144.0, 3.0, 0.0, 0.0, 1.0], [0.38, 0.51, 2.0, 159.0, 3.0, 0.0,
0.0, 1.0], [0.36, 0.48, 2.0, 156.0, 3.0, 0.0, 0.0, 1.0], [0.75, 0.
9, 5.0, 256.0, 5.0, 0.0, 0.0, 1.0], [0.1, 0.93, 6.0, 298.0, 4.0, 0
.0, 0.0, 1.0], [0.1, 0.97, 6.0, 247.0, 4.0, 0.0, 0.0, 1.0], [0.45,
0.5, 2.0, 157.0, 3.0, 0.0, 0.0, 1.0], [0.42, 0.57, 2.0, 154.0, 3.0
, 1.0, 0.0, 1.0], [0.78, 1.0, 4.0, 253.0, 5.0, 0.0, 0.0, 1.0], [0.
45, 0.55, 2.0, 148.0, 3.0, 0.0, 0.0, 1.0], [0.84, 1.0, 4.0, 261.0,
5.0, 0.0, 0.0, 1.0], [0.11, 0.93, 6.0, 282.0, 4.0, 0.0, 0.0, 1.0],
[0.42, 0.56, 2.0, 133.0, 3.0, 0.0, 0.0, 1.0], [0.45, 0.46, 2.0, 12
8.0, 3.0, 0.0, 0.0, 1.0], [0.46, 0.57, 2.0, 139.0, 3.0, 0.0, 0.0,
1.0], [0.09, 0.79, 6.0, 293.0, 5.0, 0.0, 0.0, 1.0], [0.87, 0.83, 4
.0, 265.0, 6.0, 0.0, 0.0, 1.0], [0.1, 0.87, 6.0, 250.0, 4.0, 0.0,
0.0, 1.0], [0.91, 1.0, 5.0, 251.0, 6.0, 0.0, 0.0, 1.0], [0.76, 0.9
2, 4.0, 246.0, 5.0, 0.0, 0.0, 1.0], [0.74, 1.0, 5.0, 275.0, 5.0, 0
.0, 0.0, 1.0], [0.92, 0.93, 5.0, 240.0, 5.0, 0.0, 0.0, 1.0], [0.76
```

Ingreso de los datos de data.csv en Neo4j

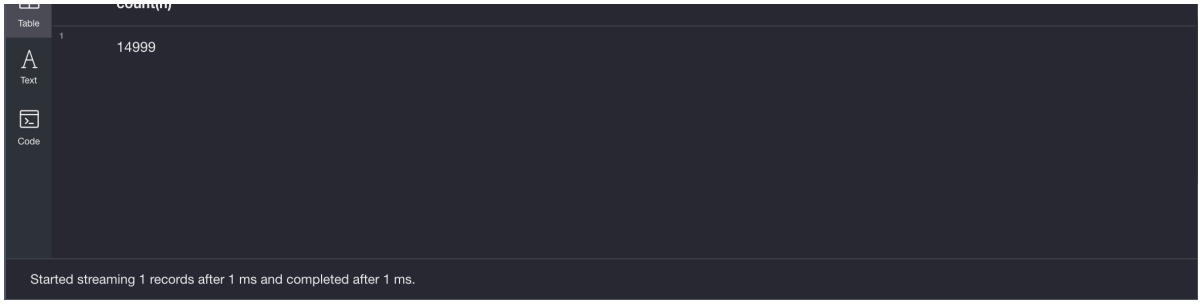
In [9]:
```
employee = str(lista)
employee = employee.replace("[[",'[')
employee = employee.replace("]]",']')
employee = employee.split('], [')
cont = 0

for contu in employee:
    cont = cont+1
    emplo = "EMPLOYEES" + str(cont)
    data = str(contu)
    data = data.replace("[",'')
    data = data.replace("]",'')
    data = data.replace("'",'')
    data = data.split(', ')
    #EMPLOYEES = PAIS(nombre='Ecuador').save()
    graph.run("CREATE (EMPLOYEES"+str(cont)+":EMPLOYEES {employ
              +"last_evaluation:"+data[1]+",number_project:"+da
              +"time_spend_company:"+data[4]+",Work_accident:"+
              +"left:"+data[7]+"})").data()
    ##employees = EMPLOYEES(employees_id = emplo, satisfaction_
      ##                    number_project = (data[2]), average
        ##                  Work_accident = (data[5]), promotio
```



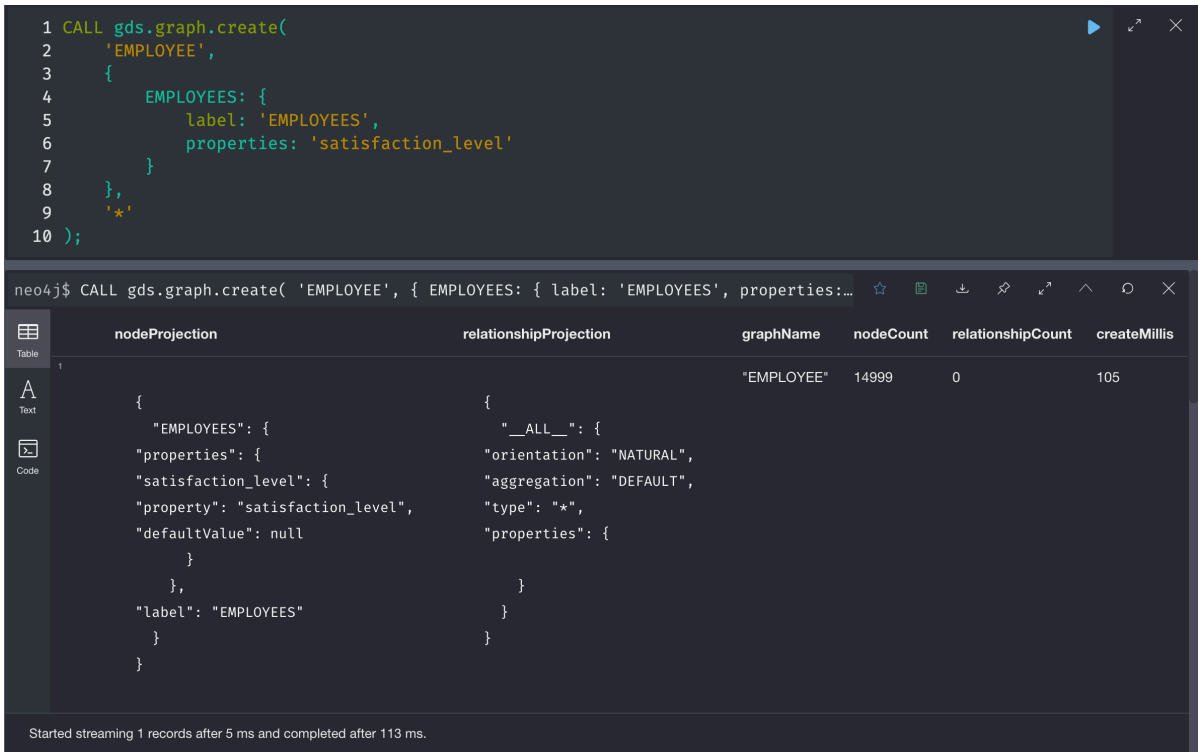

```
### Comprobar el número de datos ingresados con la siguiente consulta, donde es semilar a longitud de la lista que es de 14999
```

In [6]:
```
graph.run("MATCH(n) RETURN COUNT(n)").data()
```

Out[6]:
```
[{'COUNT(n)': 14999}]
```



Crear un gráfico para utilizar una proyección nativa y lo a lmacenará en el catálogo de gráficos con el nombre 'EMPLOYEE'.



Estimarán los requisitos de memoria para ejecutar el algori tmo, el algoritmo devuelve la puntuación de similitud para cada relación, lo cual nos devolverá 1000 fi las como resultados:

Resultado de las primeras filas de la primera consulta

```
CALL gds.beta.knn.stream('EMPLOYEE', {
    topK: 1,
    nodeWeightProperty: 'satisfaction_level',
    // The following parameters are set to produce a deterministic result
    randomSeed: 42,
    concurrency: 1,
    sampleRate: 1.0,
    deltaThreshold: 0.0
})
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).employees_id AS EMPLOYEE1, gds.util.asNode(node2).employees_id AS
EMPLOYEE2, similarity ORDER BY similarity DESCENDING, EMPLOYEE1, EMPLOYEE2
```

neo4j$ CALL gds.beta.knn.stream('EMPLOYEE', { topK: 1, nodeWeightProperty: 'satisfaction…

Table | Text | Code

	EMPLOYEE1	EMPLOYEE2	similarity
1	"EMPLOYEES1"	"EMPLOYEES965"	1.0
2	"EMPLOYEES10"	"EMPLOYEES11926"	1.0
3	"EMPLOYEES100"	"EMPLOYEES9515"	1.0
4	"EMPLOYEES1000"	"EMPLOYEES562"	1.0
5	"EMPLOYEES10000"	"EMPLOYEES8075"	1.0
6	"EMPLOYEES10001"	"EMPLOYEES11674"	1.0
7	…	…	…

Started streaming 14999 records in less than 1 ms and completed after 4 ms, displaying first 1000 rows.

Resultado de las ultimas filas de la primera consulta

```
CALL gds.beta.knn.stream('EMPLOYEE', {
    topK: 1,
    nodeWeightProperty: 'satisfaction_level',
    // The following parameters are set to produce a deterministic result
    randomSeed: 42,
    concurrency: 1,
    sampleRate: 1.0,
    deltaThreshold: 0.0
})
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).employees_id AS EMPLOYEE1, gds.util.asNode(node2).employees_id AS
EMPLOYEE2, similarity ORDER BY similarity DESCENDING, EMPLOYEE1, EMPLOYEE2
```

neo4j$ CALL gds.beta.knn.stream('EMPLOYEE', { topK: 1, nodeWeightProperty: 'satisfaction…

Table | Text | Code

	EMPLOYEE1	EMPLOYEE2	similarity
995	"EMPLOYEES10901"	"EMPLOYEES8096"	1.0
996	"EMPLOYEES10902"	"EMPLOYEES8456"	1.0
997	"EMPLOYEES10903"	"EMPLOYEES14937"	1.0
998	"EMPLOYEES10904"	"EMPLOYEES11130"	1.0
999	"EMPLOYEES10905"	"EMPLOYEES4700"	1.0
1000	"EMPLOYEES10906"	"EMPLOYEES10324"	1.0

Resultado de las primeras filas de la segunda consulta

```
CALL gds.beta.knn.stream('EMPLOYEE', {
    topK: 1,
    nodeWeightProperty: 'satisfaction_level',
    // The following parameters are set to produce a deterministic result
    randomSeed: 42,
    concurrency: 1,
    sampleRate: 1.0,
    deltaThreshold: 0.0
})
YIELD node1, node2, similarity WHERE similarity<1.0
RETURN gds.util.asNode(node1).employees_id AS EMPLOYEE1, gds.util.asNode(node2).employees_id AS
EMPLOYEE2, similarity  ORDER BY similarity DESCENDING, EMPLOYEE1, EMPLOYEE2
```

neo4j$ CALL gds.beta.knn.stream('EMPLOYEE', { topK: 1, nodeWeightProperty: 'satisfaction…

Table | Text | Code

	EMPLOYEE1	EMPLOYEE2	similarity
1	"EMPLOYEES11705"	"EMPLOYEES2390"	0.9900990099009903
2	"EMPLOYEES11903"	"EMPLOYEES7570"	0.9900990099009903
3	"EMPLOYEES14661"	"EMPLOYEES7532"	0.9900990099009903
4	"EMPLOYEES9481"	"EMPLOYEES3964"	0.9900990099009903
5	"EMPLOYEES10132"	"EMPLOYEES173"	0.9900990099009901
6	"EMPLOYEES10170"	"EMPLOYEES4432"	0.9900990099009901
7	…	…	…

Started streaming 198 records after 1 ms and completed after 296 ms.

Resultado de las ultimas filas de la segunda consulta

```
CALL gds.beta.knn.stream('EMPLOYEE', {
    topK: 1,
    nodeWeightProperty: 'satisfaction_level',
    // The following parameters are set to produce a deterministic result
    randomSeed: 42,
    concurrency: 1,
    sampleRate: 1.0,
    deltaThreshold: 0.0
})
YIELD node1, node2, similarity WHERE similarity<1.0
RETURN gds.util.asNode(node1).employees_id AS EMPLOYEE1, gds.util.asNode(node2).employees_id AS
EMPLOYEE2, similarity  ORDER BY similarity DESCENDING, EMPLOYEE1, EMPLOYEE2
```

neo4j$ CALL gds.beta.knn.stream('EMPLOYEE', { topK: 1, nodeWeightProperty: 'satisfaction…

Table | Text | Code

	EMPLOYEE1	EMPLOYEE2	similarity
193	"EMPLOYEES9841"	"EMPLOYEES1425"	0.9900990099009901
194	"EMPLOYEES11307"	"EMPLOYEES8507"	0.9803921568627451
195	"EMPLOYEES13260"	"EMPLOYEES5229"	0.9803921568627451
196	"EMPLOYEES3575"	"EMPLOYEES9499"	0.9803921568627451
197	"EMPLOYEES4035"	"EMPLOYEES3818"	0.9803921568627451
198	"EMPLOYEES6780"	"EMPLOYEES10180"	0.9803921568627451

Started streaming 198 records after 1 ms and completed after 296 ms.

- Ingresar cada uno de los datos en un nodo y obtener el grado de similitud se recomienda utilizar la distancia Euclidiana o Person, una vez obtenido la similitud ingresar datos de prueba para validar (Máximo 3 datos).

In [13]:
```
empl = "EMPLOYEESDP1"
graph.run("CREATE (EMPLOYEES"+str(cont)+":EMPLOYEES {employees_
```

Out[13]: []

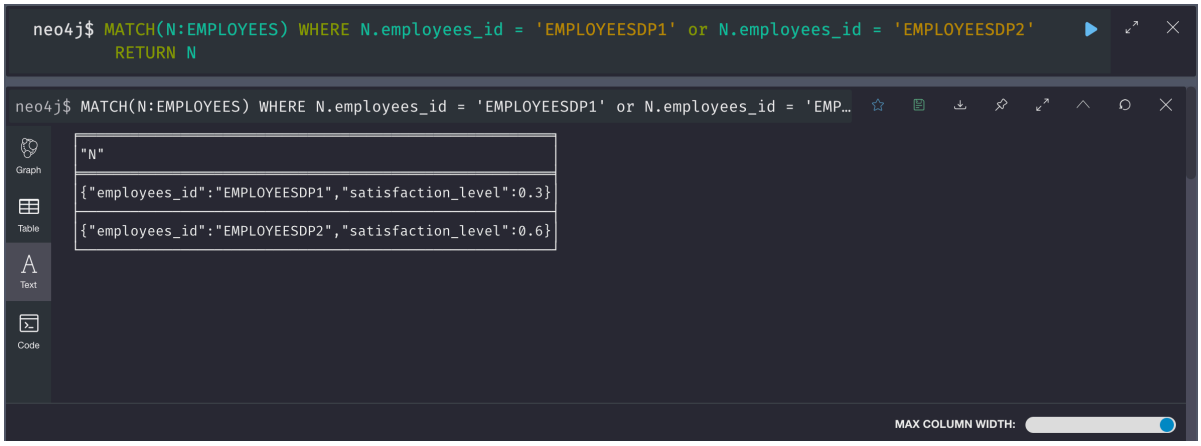
In [14]:
```
empl = "EMPLOYEESDP2"
graph.run("CREATE (EMPLOYEES"+str(cont)+":EMPLOYEES {employees_
```

Out[14]: []

In [16]:
```
graph.run("MATCH(N:EMPLOYEES) WHERE N.employees_id = 'EMPLOYEESDP1' OR
```

Out[16]:
```
[{'N': Node('EMPLOYEES', employees_id='EMPLOYEESDP1', satisfaction
_level=0.3)},
 {'N': Node('EMPLOYEES', employees_id='EMPLOYEESDP2', satisfaction
_level=0.6)}]
```

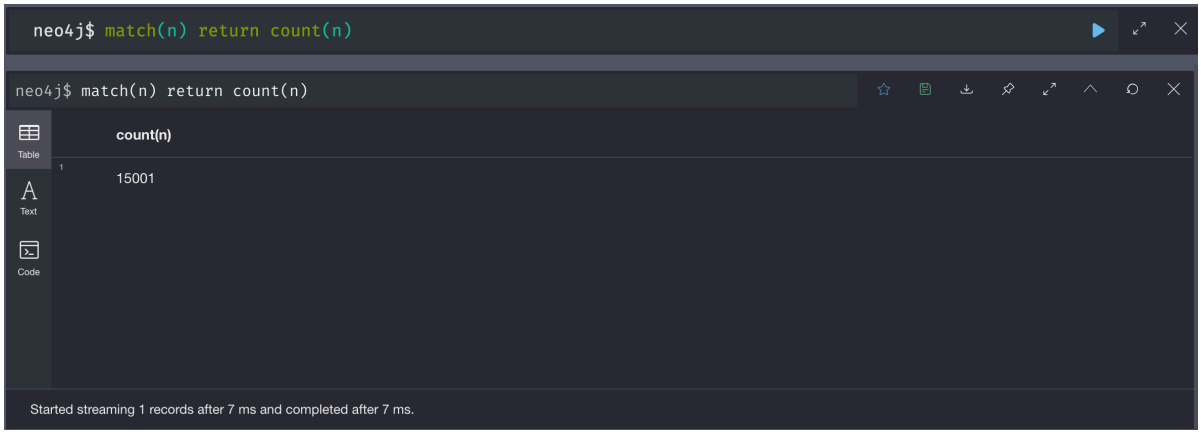
Comprobar el ingreso de los dos ultimos datos:



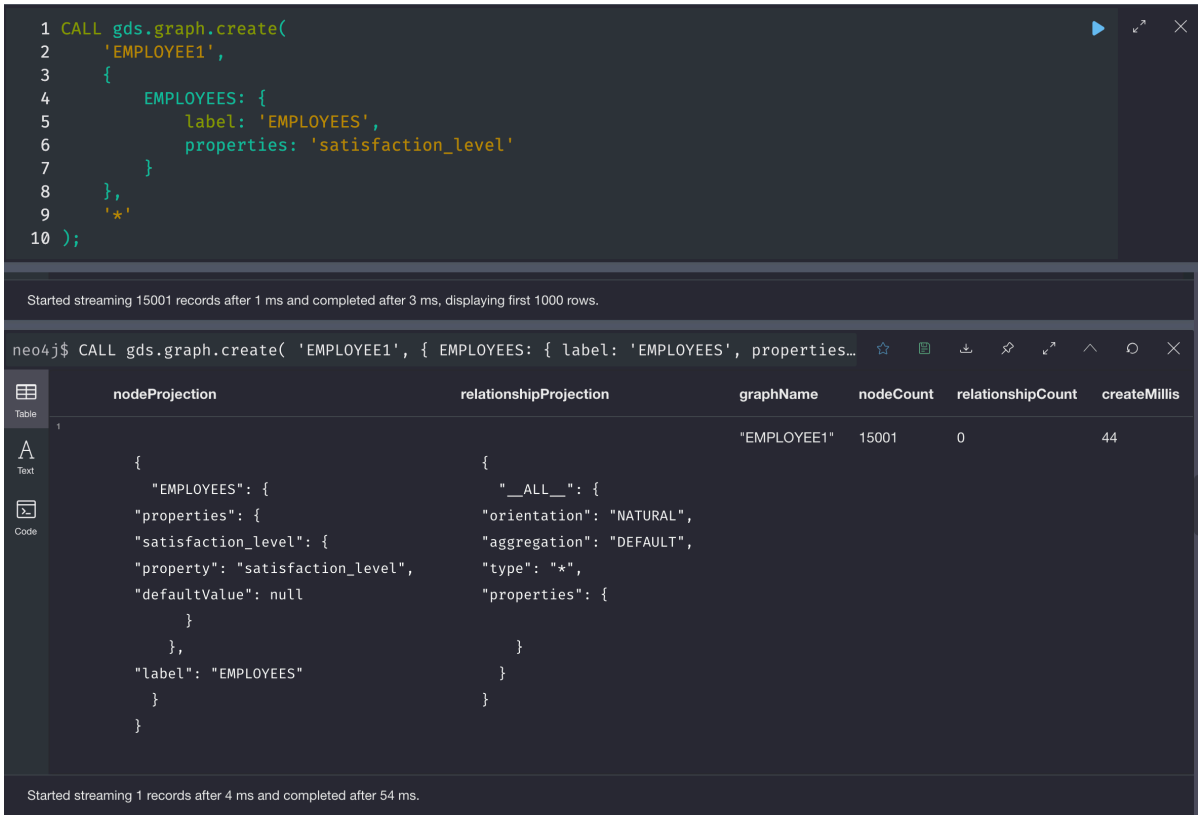
Comprobar el ingreso de los dos ultimos datos, con un count de los datos que se encuentras en la base de datos, más de los 14999 que están ingreesado como se mostro anteriormente:

In [17]:
```
graph.run("MATCH(n) RETURN COUNT(n)").data()
```

Out[17]:
```
[{'COUNT(n)': 15001}]
```



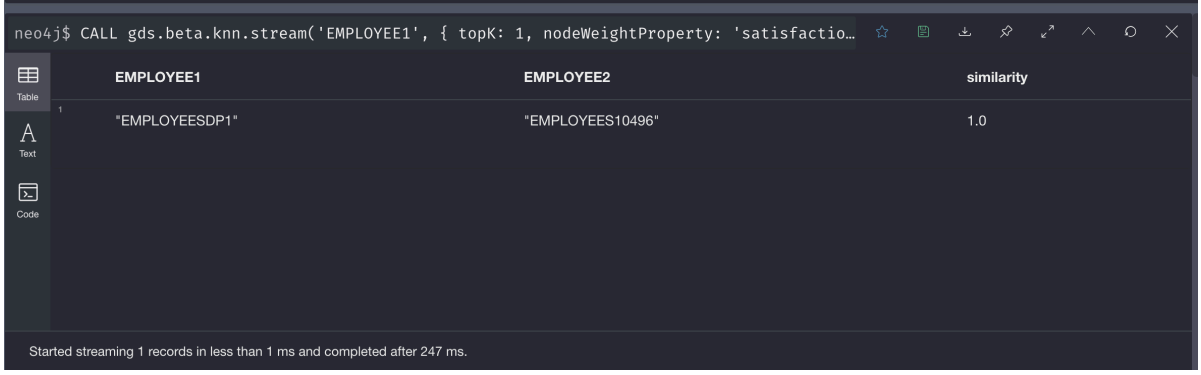
Crear un gráfico para utilizar una proyección nativa y lo a lmacenará en el catálogo de gráficos con el nombre 'EMPLOYEE1'.



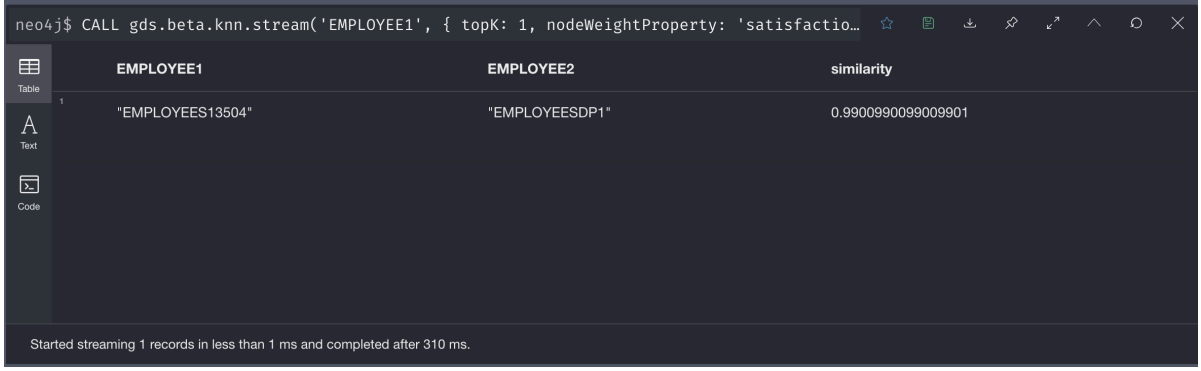
Estimarán los requisitos de memoria para ejecutar el algori tmo, el algoritmo devuelve la puntuación de similitud para cada relación, lo cual nos devolverá 1 fila como resultados, debido a que no escuentras Más similitudes:

Resultado de las primeras filas de la primera consulta con el satisfaction_level = "EMPLOYEESDP1"

```
CALL gds.beta.knn.stream('EMPLOYEE1', {
    topK: 1,
    nodeWeightProperty: 'satisfaction_level',
    // The following parameters are set to produce a deterministic result
    randomSeed: 42,
    concurrency: 1,
    sampleRate: 1.0,
    deltaThreshold: 0.0
})
YIELD node1, node2, similarity WHERE gds.util.asNode(node1).employees_id  = "EMPLOYEESDP1"
RETURN gds.util.asNode(node1).employees_id AS EMPLOYEE1, gds.util.asNode(node2).employees_id AS
EMPLOYEE2, similarity  ORDER BY similarity DESCENDING, EMPLOYEE1, EMPLOYEE2
```

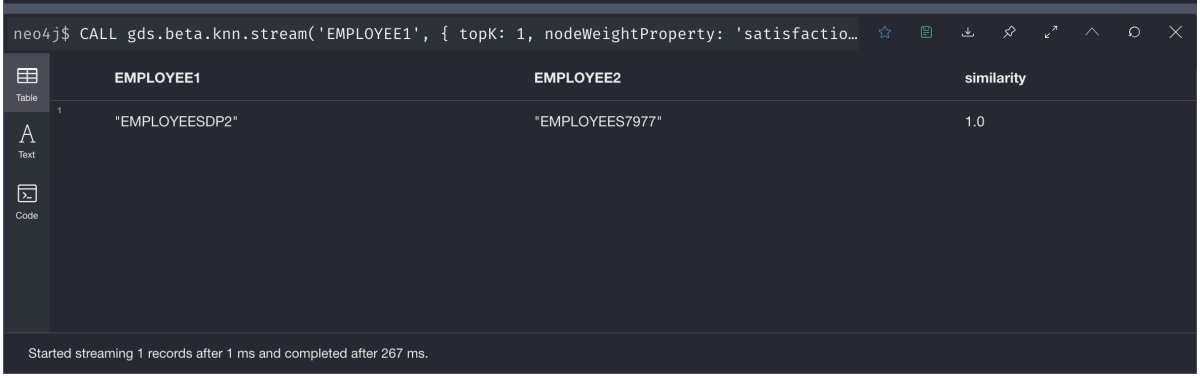



```
CALL gds.beta.knn.stream('EMPLOYEE1', {
    topK: 1,
    nodeWeightProperty: 'satisfaction_level',
    // The following parameters are set to produce a deterministic result
    randomSeed: 42,
    concurrency: 1,
    sampleRate: 1.0,
    deltaThreshold: 0.0
})
YIELD node1, node2, similarity WHERE gds.util.asNode(node2).employees_id  = "EMPLOYEESDP1"
RETURN gds.util.asNode(node1).employees_id AS EMPLOYEE1, gds.util.asNode(node2).employees_id AS
EMPLOYEE2, similarity  ORDER BY similarity DESCENDING, EMPLOYEE1, EMPLOYEE2
```

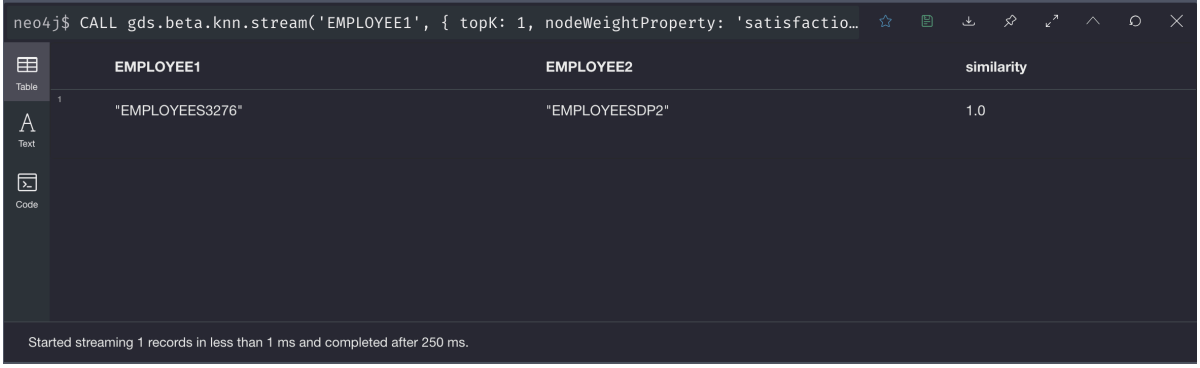


Resultado de las primeras filas de la primera consulta con el satisfaction_level = "EMPLOYEESDP2"

```
CALL gds.beta.knn.stream('EMPLOYEE1', {
    topK: 1,
    nodeWeightProperty: 'satisfaction_level',
    // The following parameters are set to produce a deterministic result
    randomSeed: 42,
    concurrency: 1,
    sampleRate: 1.0,
    deltaThreshold: 0.0
})
YIELD node1, node2, similarity WHERE gds.util.asNode(node1).employees_id  = "EMPLOYEESDP2"
RETURN gds.util.asNode(node1).employees_id AS EMPLOYEE1, gds.util.asNode(node2).employees_id AS
EMPLOYEE2, similarity  ORDER BY similarity DESCENDING, EMPLOYEE1, EMPLOYEE2
```




```
CALL gds.beta.knn.stream('EMPLOYEE1', {
    topK: 1,
    nodeWeightProperty: 'satisfaction_level',
    // The following parameters are set to produce a deterministic result
    randomSeed: 42,
    concurrency: 1,
    sampleRate: 1.0,
    deltaThreshold: 0.0
})
YIELD node1, node2, similarity WHERE gds.util.asNode(node2).employees_id  = "EMPLOYEESDP2"
RETURN gds.util.asNode(node1).employees_id AS EMPLOYEE1, gds.util.asNode(node2).employees_id AS
EMPLOYEE2, similarity  ORDER BY similarity DESCENDING, EMPLOYEE1, EMPLOYEE2
```



- Generar otro entorno en donde solo ingrese el 70% de l os datos y validar con el 30%.

In [27]:
```
print(round(len(lista)*0.7))
```

```
10499
```

Conexión con Neo4j

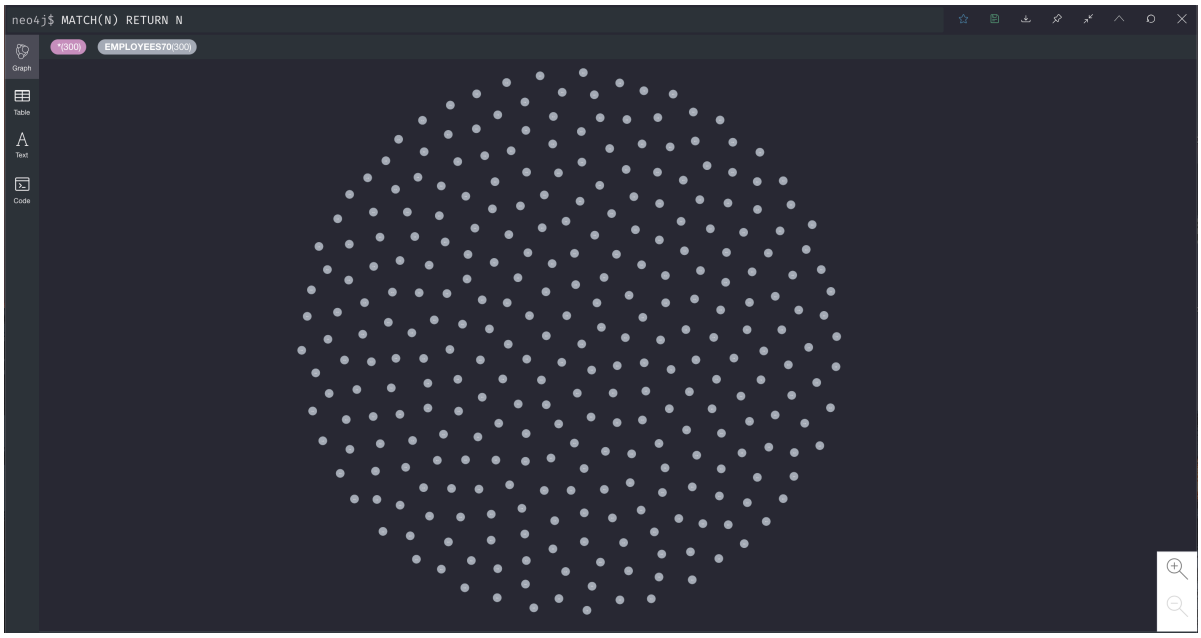
Configure la URL de conexión con la base de datos de Ne04j:

graph = Graph("bolt://localhost:7687", aut="neo4j", password="Prueba2.1", secure=False)

In [37]:
```
#IMPORTAR py2neo
from py2neo import Node, Relationship, Graph


# connect to authenticated graph database
graph = Graph("bolt://localhost:7687", aut="neo4j", password="P
```

Ingreso de los datos de data.csv en Neo4j

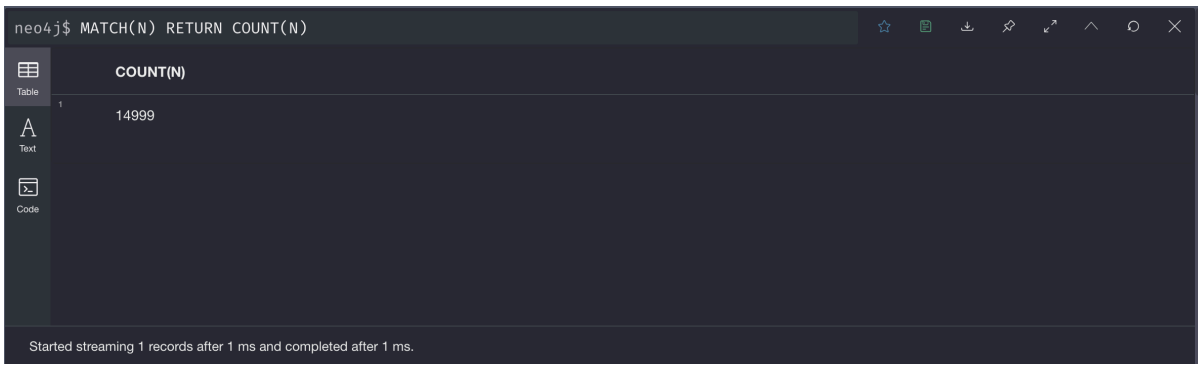
In [43]:
```
employee = str(lista)
employee = employee.replace("[[",'[')
employee = employee.replace("]]",']')
employee = employee.split('], [')
cont = 0

for contu in employee:
    cont = cont+1

    data = str(contu)
    data = data.replace("[",'')
    data = data.replace("]",'')
    data = data.replace("'",'')
    data = data.split(', ')
    if(round(len(lista)*0.7)>=cont):
        emplo = "EMPLOYEES70-" + str(cont)
        graph.run("CREATE (EMPLOYEES"+str(cont)+":EMPLOYEES70 {
              +"last_evaluation:"+data[1]+",number_project:"+da
              +"time_spend_company:"+data[4]+",Work_accident:"+
              +"left:"+data[7]+"})").data()

    else :

        emplo1 = "EMPLOYEES30-" + str(cont)
        graph.run("CREATE (EMPLOYEES"+str(cont)+":EMPLOYEES30 {
              +"last_evaluation:"+data[1]+",number_project:"+da
              +"time_spend_company:"+data[4]+",Work_accident:"+
              +"left:"+data[7]+"})").data()

```



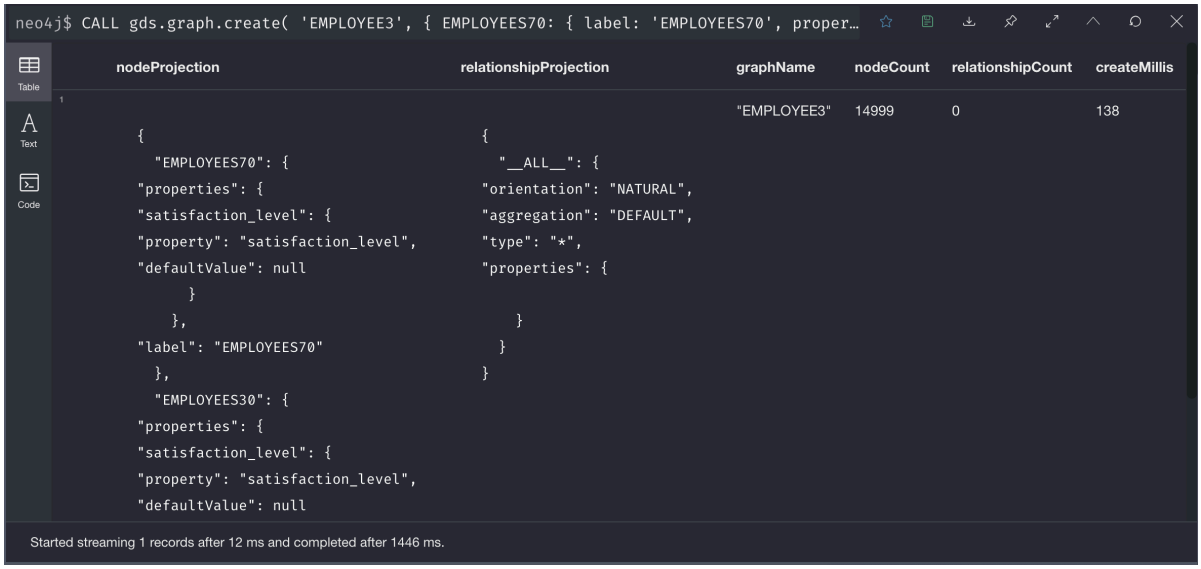
Comprobar el número de datos ingresados con la siguiente consulta, donde es semilar a longitud de la lista que es de 14999

In [44]:
```
1 graph.run("MATCH(n) RETURN COUNT(n)").data()
```

Out[44]:
```
[{'COUNT(n)': 14999}]
```

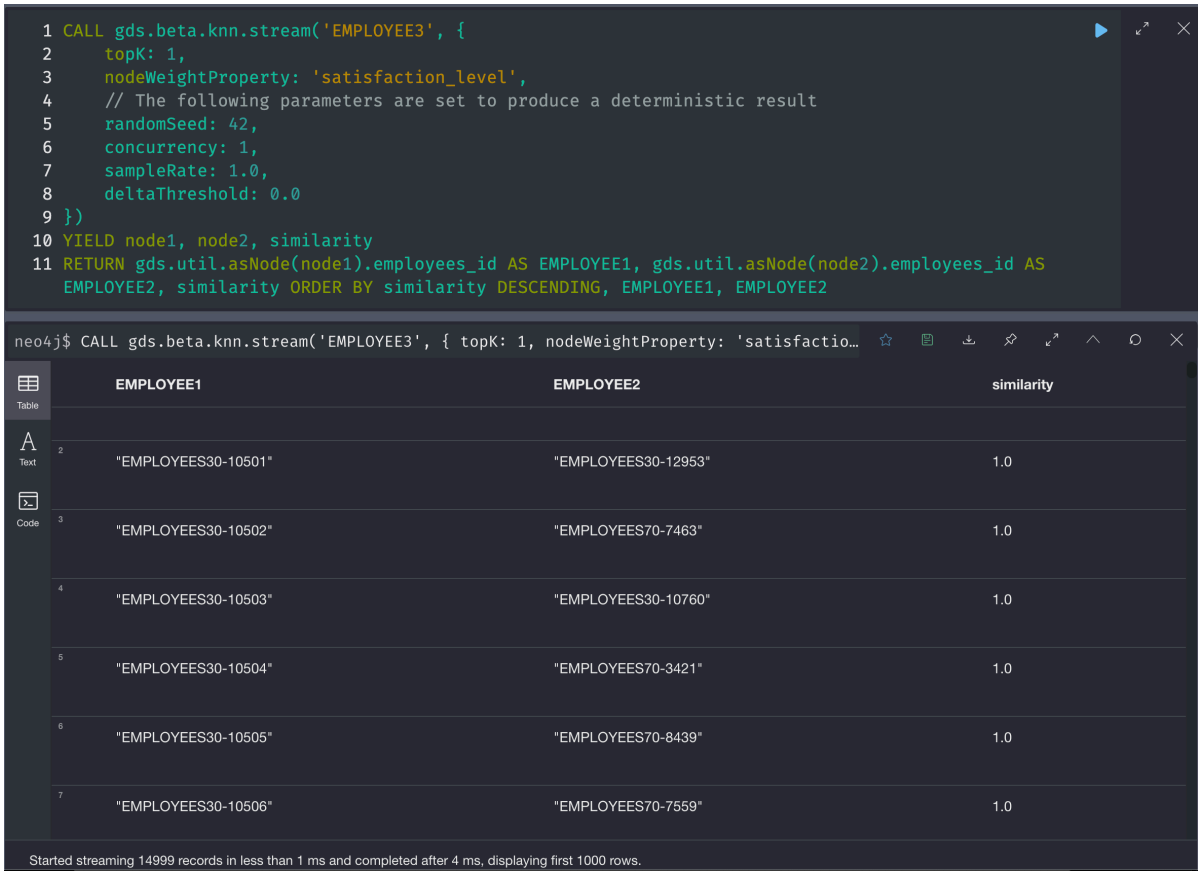



```
Crear un gráfico para utilizar una proyección nativa y lo a
lmacenará en el catálogo de gráficos con el
nombre 'EMPLOYEE3'.
```



Estimarán los requisitos de memoria para ejecutar el algori tmo, el algoritmo devuelve la puntuación de similitud para cada relación, lo cual nos devolverá 1000 fi las como resultados:

Resultado de las primeras filas de la primera consulta



Resultado de las ultimas filas de la primera consulta

```
CALL gds.beta.knn.stream('EMPLOYEE3', {
    topK: 1,
    nodeWeightProperty: 'satisfaction_level',
    // The following parameters are set to produce a deterministic result
    randomSeed: 42,
    concurrency: 1,
    sampleRate: 1.0,
    deltaThreshold: 0.0
})
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).employees_id AS EMPLOYEE1, gds.util.asNode(node2).employees_id AS
EMPLOYEE2, similarity ORDER BY similarity DESCENDING, EMPLOYEE1, EMPLOYEE2
```

neo4j$ CALL gds.beta.knn.stream('EMPLOYEE3', { topK: 1, nodeWeightProperty: 'satisfactio…

Table | Text | Code

	EMPLOYEE1	EMPLOYEE2	similarity
995	"EMPLOYEES30-11502"	"EMPLOYEES70-8123"	1.0
996	"EMPLOYEES30-11503"	"EMPLOYEES30-13069"	1.0
997	"EMPLOYEES30-11504"	"EMPLOYEES70-9415"	1.0
998	"EMPLOYEES30-11505"	"EMPLOYEES70-10064"	1.0
999	"EMPLOYEES30-11506"	"EMPLOYEES70-7481"	1.0
1000	"EMPLOYEES30-11507"	"EMPLOYEES30-13616"	1.0

Started streaming 14999 records in less than 1 ms and completed after 4 ms, displaying first 1000 rows.

Resultado de las primeras filas de la segunda consulta

```
CALL gds.beta.knn.stream('EMPLOYEE3', {
    topK: 1,
    nodeWeightProperty: 'satisfaction_level',
    // The following parameters are set to produce a deterministic result
    randomSeed: 42,
    concurrency: 1,
    sampleRate: 1.0,
    deltaThreshold: 0.0
})
YIELD node1, node2, similarity WHERE similarity<1
RETURN gds.util.asNode(node1).employees_id AS EMPLOYEE1, gds.util.asNode(node2).employees_id AS
EMPLOYEE2, similarity ORDER BY similarity DESCENDING, EMPLOYEE1, EMPLOYEE2
```

neo4j$ CALL gds.beta.knn.stream('EMPLOYEE3', { topK: 1, nodeWeightProperty: 'satisfactio…

Table | Text | Code

	EMPLOYEE1	EMPLOYEE2	similarity
3	"EMPLOYEES30-14661"	"EMPLOYEES70-7532"	0.9900990099009903
4	"EMPLOYEES70-9481"	"EMPLOYEES70-3964"	0.9900990099009903
5	"EMPLOYEES30-10733"	"EMPLOYEES30-13890"	0.9900990099009901
6	"EMPLOYEES30-10764"	"EMPLOYEES30-13020"	0.9900990099009901
7	"EMPLOYEES30-10909"	"EMPLOYEES70-3282"	0.9900990099009901
8	"EMPLOYEES30-10978"	"EMPLOYEES70-6087"	0.9900990099009901

Started streaming 198 records after 1 ms and completed after 956 ms.

Resultado de las ultimas filas de la segunda consulta

```
CALL gds.beta.knn.stream('EMPLOYEE3', {
    topK: 1,
    nodeWeightProperty: 'satisfaction_level',
    // The following parameters are set to produce a deterministic result
    randomSeed: 42,
    concurrency: 1,
    sampleRate: 1.0,
    deltaThreshold: 0.0
})
YIELD node1, node2, similarity WHERE similarity<1
RETURN gds.util.asNode(node1).employees_id AS EMPLOYEE1, gds.util.asNode(node2).employees_id AS
EMPLOYEE2, similarity ORDER BY similarity DESCENDING, EMPLOYEE1, EMPLOYEE2
```

neo4j$ CALL gds.beta.knn.stream('EMPLOYEE3', { topK: 1, nodeWeightProperty: 'satisfactio…

Table | Text | Code

	EMPLOYEE1	EMPLOYEE2	similarity
193	"EMPLOYEES70-9841"	"EMPLOYEES70-1425"	0.9900990099009901
194	"EMPLOYEES30-11307"	"EMPLOYEES70-8507"	0.9803921568627451
195	"EMPLOYEES30-13260"	"EMPLOYEES70-5229"	0.9803921568627451
196	"EMPLOYEES70-3575"	"EMPLOYEES70-9499"	0.9803921568627451
197	"EMPLOYEES70-4035"	"EMPLOYEES70-3818"	0.9803921568627451
198	"EMPLOYEES70-6780"	"EMPLOYEES70-10180"	0.9803921568627451

Started streaming 198 records after 1 ms and completed after 956 ms.