Nombre Roberto Pacho
Prueba SE 2

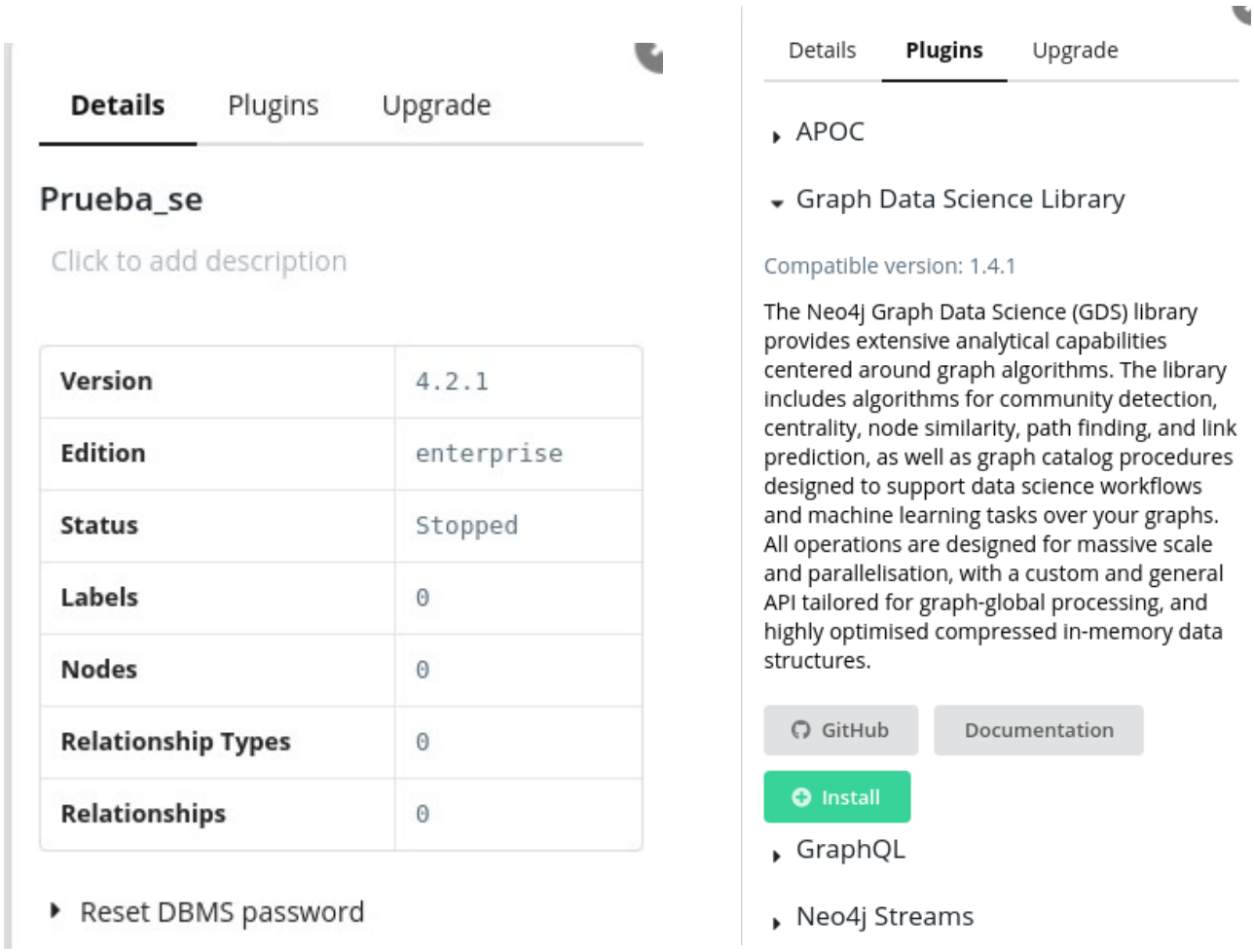
Objetivo

- Diseñe y desarrolle un algoritmo Knn en Neo4j para:

  Predicción de "género" mediante el valor de "Compra" y el tipo de "Ocupación", para descargar los datos del siguiente link: https://www.kaggle.com/alllexander/blackfriday

- 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).
- Generar otro entorno en donde solo ingrese el 70% de los datos y validar con el 30%.
- Agregar el grafico con los nodos conformados.
- El proceso de programación desarrollado deberá considerar los siguientes aspectos:
  Se deberá tener un archivo que tenga todos los procesos o código de búsqueda y datos de Neo4j (https://neo4j.com/docs/labs/apoc/current/export/cypher/).

- Cargamos los datos a Neo4j
  - Creamos la base de datos y descargamos el plugin necesario

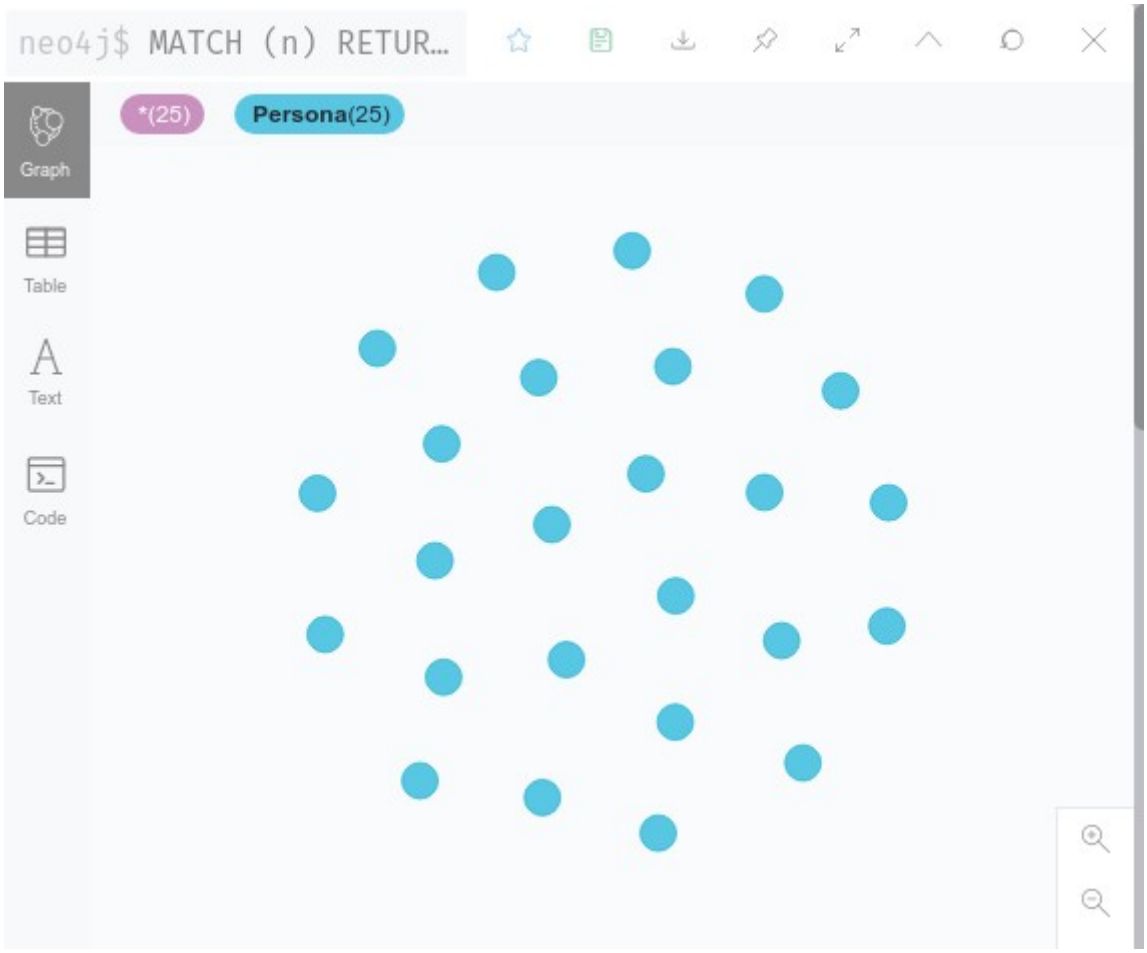


- Cargamos los datos en neo4j mediante el siguiente comando

LOAD csv WITH HEADERS FROM "http://localhost:11001/project-908e19e2-1ac4-4931-9bc1-d39e83c73a21/BlackFriday.csv" as datos create (datos.User_ID:Datos {Productos_ID: datos.Product_ID, genero: datos.Gender, ocupacion:datos.Occupation})

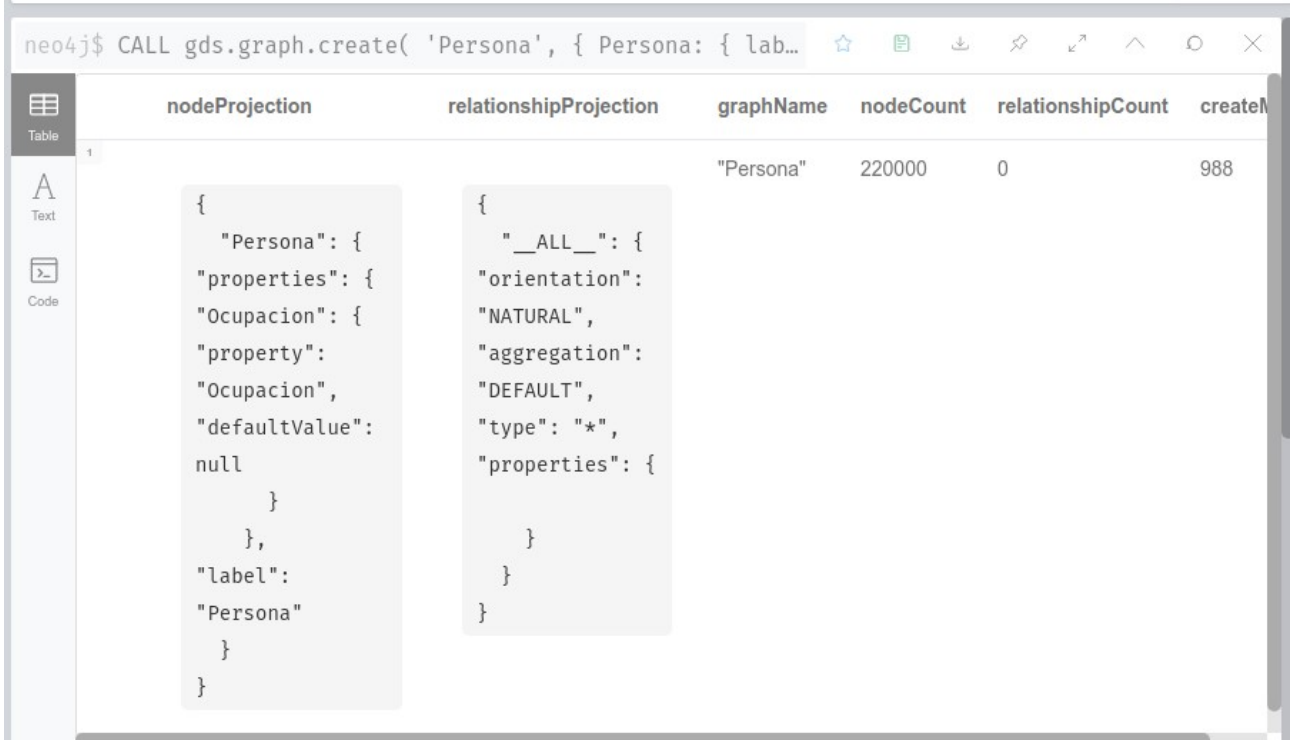
```
neo4j$ LOAD csv WITH HEADERS FROM "http://localhost:11001/project-
       908e19e2-1ac4-4931-9bc1-d39e83c73a21/BlackFriday.csv" as datos
       create (datos.User_ID:Datos {Productos_ID: datos.Product_ID,
       genero: datos.Gender, ocupacion:datos.Occupation})
```

- grafo



- crear el gráfico y lo almacenará en el catálogo de gráficos.

```
CALL gds.graph.create(
    'Persona',
    {
        Persona: {
            label: 'Persona',
            properties: 'Ocupacion'
        }
    },
    '*'
);
```



- se estimarán los requisitos de memoria para ejecutar el algoritmo:

```
CALL gds.beta.knn.write.estimate('Persona', {
  nodeWeightProperty: 'Ocupacion',
  writeRelationshipType: 'SIMILAR',
  writeProperty: 'score',
  topK: 1
})
YIELD nodeCount, bytesMin, bytesMax, requiredMemory
```

neo4j$ CALL gds.beta.knn.write.estimate('Persona', { node…

Table | Text

	nodeCount	bytesMin	bytesMax	requiredMemory
1	220000	16720824	58960824	"[16328 KiB ... 56 MiB]"

- En el stream modo de ejecución, el algoritmo devuelve la puntuación de similitud para cada relación. Esto nos permite inspeccionar los resultados directamente o procesarlos posteriormente en Cypher sin efectos secundarios. Por ejemplo, podemos ordenar los resultados para encontrar los nodos con el coeficiente de agrupamiento local más alto.

Lo siguiente ejecutará el algoritmo y transmitirá los resultados:

- comparación por producto

```
CALL gds.beta.knn.stream('Persona', {
    topK: 1,
    nodeWeightProperty: 'Ocupacion',
    // 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).Product_ID AS Producto1, gds.util.asNode(node2).Product_ID AS Producto2, similarity
ORDER BY similarity DESCENDING, Producto1, Producto2
```

neo4j$ CALL gds.beta.knn.stream('Persona', { topK: 1, nodeWeightProperty: 'Ocupacion', // The following parameters are set to…

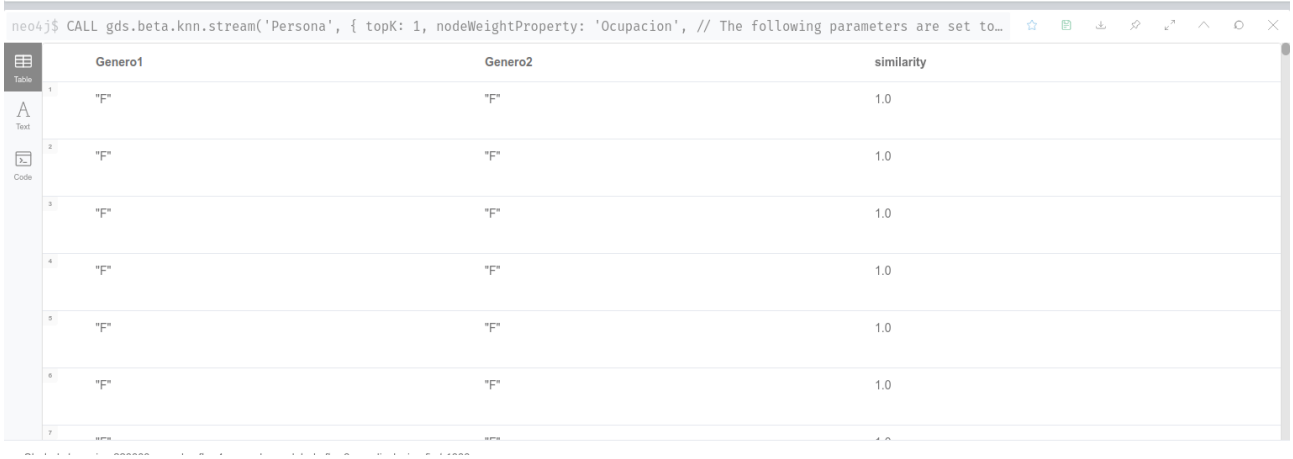
Table | Text | Code

	Producto1	Producto2	similarity
1	"P00000142"	"P00001042"	1.0
2	"P00000142"	"P00001042"	1.0
3	"P00000142"	"P00001742"	1.0
4	"P00000142"	"P00002542"	1.0
5	"P00000142"	"P00002942"	1.0
6	"P00000142"	"P00006142"	1.0
7	"P00000142"	"P00006142"	1.0

Started streaming 220000 records after 3 ms and completed after 8 ms, displaying first 1000 rows.

- Comparación por genero:

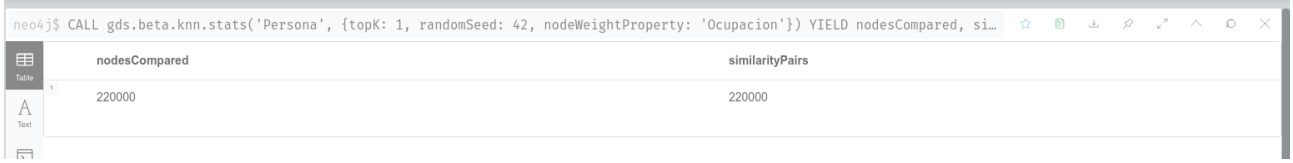
```
CALL gds.beta.knn.stream('Persona', {
    topK: 1,
    nodeWeightProperty: 'Ocupacion',
    // The following parameters are set to produce a deterministic result
    randomSeed: 42,
    concurrency: 1,
    sampleRate: 1.0,
    deltaThreshold: 0.0
})
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).genero AS Genero1, gds.util.asNode(node2).genero AS Genero2, similarity
ORDER BY similarity DESCENDING, Genero1, Genero2
```



	Genero1	Genero2	similarity
1	"F"	"F"	1.0
2	"F"	"F"	1.0
3	"F"	"F"	1.0
4	"F"	"F"	1.0
5	"F"	"F"	1.0
6	"F"	"F"	1.0

- Lo siguiente ejecutará el algoritmo y devolverá el resultado en forma de valores estadísticos y de medición:

```
CALL gds.beta.knn.stats('Persona', {topK: 1, randomSeed: 42, nodeWeightProperty: 'Ocupacion'})
YIELD nodesCompared, similarityPairs
```



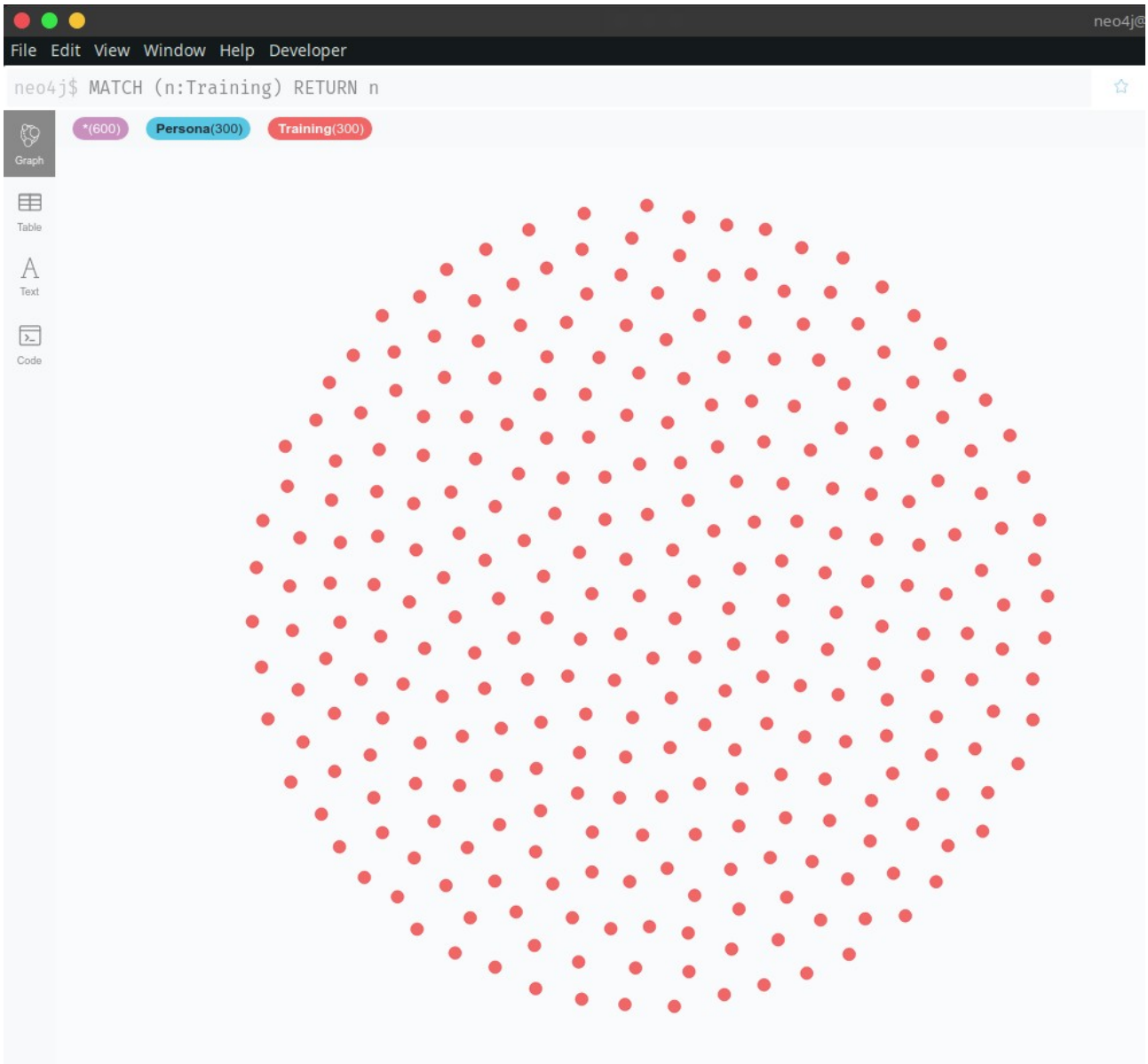
	nodesCompared	similarityPairs
1	220000	220000

- 5)Dividamos nuestro conjunto de datos en dos subconjuntos, donde el 70% de los nodos se marcarán como datos de entrenamiento y el 30% restante como datos de prueba. Hay un total de 537578 nodos en nuestro gráfico. TotalNodos del 70% es 376304,6. Marcaremos 376304,6 nodos como subconjunto de entrenamiento y el resto como prueba.

Sacamos el 70%
537578 x 70% = 376304,6
sacamos el 30%
537578 x 30% = 161273,4

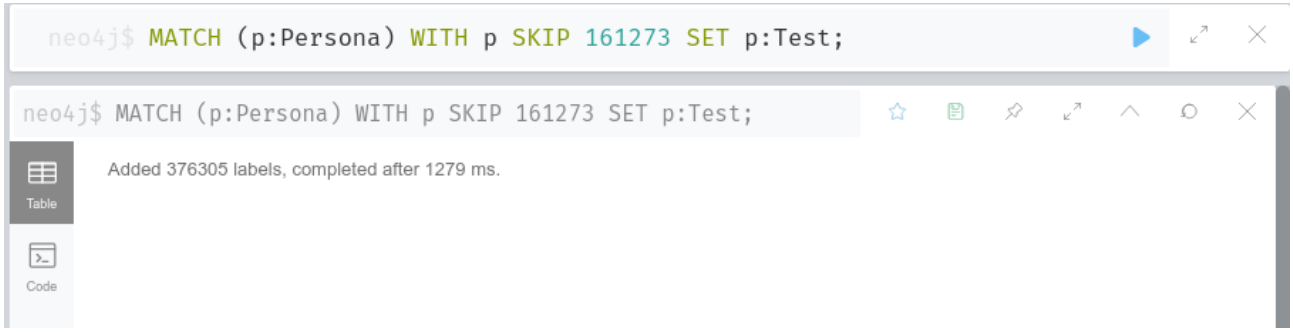
Nodos 70%

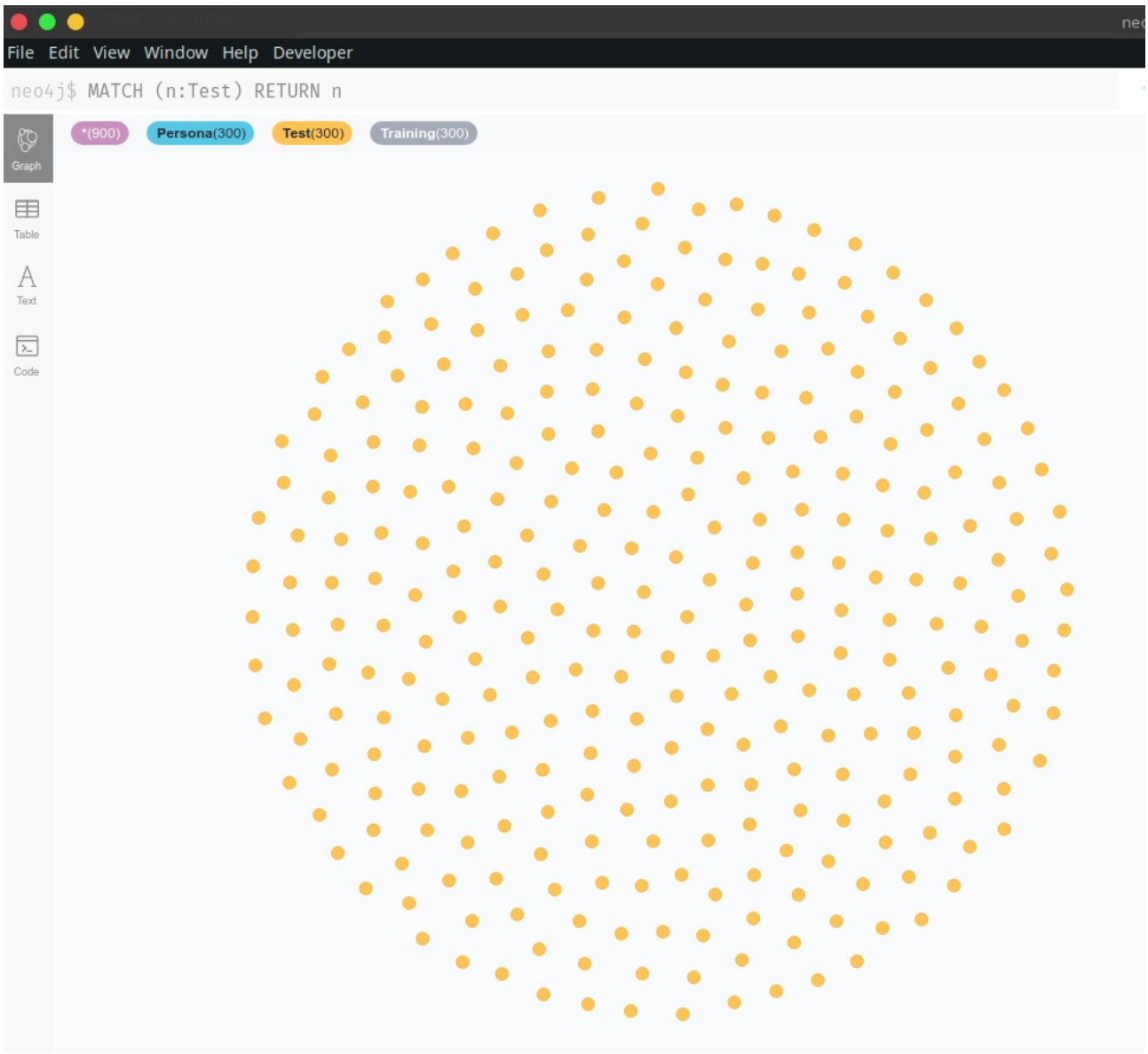
MATCH (p:Persona) WITH p LIMIT 376305 SET p:Training;



Nodos 30%

MATCH (p:Persona) WITH p SKIP 161273 SET p:Test;





- Transformar a vector de características, utilizando el valor de "Compra" y el tipo de "Ocupación".

MATCH (n:Persona)UNWIND n.features as feature WITH n,collect(CASE feature WHEN n.features[1] THEN toInteger(n.features[1]) WHEN n.features[8] THEN toInteger(n.features[8]) END) as feature_vector SET n.feature_vector = feature_vector