Anexo 4

A continuación se indican las querys de consulta de grafos de conocimientos propios de Neo4j. La implementación en Python se realizó mediante funciones que reciben parámetros pasados desde el FrontEnd como se especifica a continuación:

```
def nodo(self, tx,TIden):
tx.run("MERGE(:Type_of_identification{Type_of_identification:$TIden})",
TIden=TIden)
```

para crear los nodos en el grafo de conocimiento, se procedió a realizar la siguiente query en este caso para la información del usuario relacionada a la condición de desplazamiento, la cláusula MERGE se encarga de crear los nodos con la condición de que si este ya existe en el grafo de conocimiento no procederá a duplicar la información, dentro del nodo especificado con paréntesis se pone una etiqueta (`Condition_of_displacement`) para agrupar varios nodos a un solo tipo. Después con los signos de llaves se especifica las propiedades del nodo, en este caso se puede observar que el nodo “Condition_of_displacement” tiene como propiedad “desplazado”.

```
MERGE(:Condition_of_displacement{Condition_of_displacement:Desplazado})
```

En la siguiente cláusula MATCH hace una búsqueda de los campos especificados en el grafo de conocimiento, al usar una cláusula WHERE se especifica qué datos de la etiqueta requieren ser buscados, donde al final haciendo uso de la cláusula MERGE crea la relación entre los nodos buscados anteriormente. En la siguiente parte del código se buscan los nodos de tipo de identificación y la propiedad de tipo de identificación que fue creada por la ontología, donde hay una relación entre los dos nodos.

```
MATCH (n:Type_of_identification ), (s:n4sch__Property) where
n.Type_of_identification='cedula'  and
s.n4sch__name='Type_of_identification' MERGE (n)-[:is]->(s)
```

Para trabajar con la similaridad de nodos es necesario tener en cuenta a qué usuario se le aplicará la similitud, en el siguiente código se especificó el usuario, donde tenga relación con los nodos en este caso de creencias, para ello es necesario realizar una colección de IDs de creencias, donde se comparan dos conjuntos, en este caso comparar todas las creencias de los usuarios con las creencias del usuario en específico. Al final se hace un promedio de los resultados de aplicar al algoritmo de similaridad de Jaccard.

```
MATCH (n:Vaccinated{Vaccinated:Vacunado1}) -[:has]->(s:Belief)WITH n,
collect(id(s)) AS n1Creencias MATCH (v2:Vaccinated)-[:has] -> (Belief)
WHERE n <> v2 WITH n, n1Creencias, v2, collect(id(Belief)) AS
n2Creencias RETURN AVG(gds.alpha.similarity.jaccard(n1Creencias,
n2Creencias)) AS similarity")
```

La siguiente query retorna la id del nodo de similaridad relacionada a dicho usuario, estas id es necesaria para la actualización del mismo nodo de similaridad.

```
match (n:Vaccinated)-[:has]->(s:Similarity_B) with s,n,id(s) as id
where n.Vaccinated='Vacunado1' return id
```

Con la id identificada como se mencionó anteriormente se procedió a realizar la actualización del nodo de similaridad relacionada al usuario de la siguiente forma:

```
Match (n:Vaccinated),(s:Similarity_B)with n,s match (n)-[:has]->(s)
where n.Vaccinated='Vacunado1' set s.Similarity_B='id’
```