Graph definition

All nodes need to be identified as part of a “universe”.

The first set of data models created in Neo4j, create the necessity to add more specific information to the node.

All nodes can include different properties which make almost impossible define a general routine to present information using the graphical interface, especially the text showed into the graph.

The creation of multiple nodes from different sources made almost impossible the option to reprocess a piece of the graph.

A simple example:

- One set of instructions created the country codes.
- One set of instructions created the WHO ICD 10 data structure.
- One set of instructions created the effect of a specific disease.
- One set of instructions created the Articles citations.
- One set of instructions created the Authors and the relationships with the countries where the data of the studies were generated.

If for some reason this data set needs to be merged with another dataset, this merge could create “noise” after the execution. To remove these new nodes and relationships, we need to have a reference to ***select*** *and* ***remove*** them and restore the previous state of the graph.

Identification Properties

These properties are not standardized or controlled by any organization, so it is possible to define your own node identification strategy. If you collaborate with other graphs, it is important to define the “***integration protocol***” to be able to integrate multiple “universes” in your graph.

Most implementations require the abstraction process to define some guide to be able to identify key elements, what to display, key elements to find the node, links to other nodes, etc.

Example:

- Id_Udsc Universe description. *Identify the graph database “universe”, to identify a group of nodes. This element could be substituted by a “**label**”, but increase the risk of “omission”. If we define that **all** nodes contain universe **identification** this will facilitate the process.*

- Id_Udsc:”ISO3_3166”. *Indicates that all these nodes came from one specific source, containing the ISO country codes.*
- *Id_Udsc:”WHO_ICD_10”. Indicates that all these nodes came from “World Health Organization” and correspond to the “International Statistical Classification of Diseases and Related Health Problems”.*

- Id_Nlab Node label. *This property contain the “**default**” property to use to present in the diagrams r visual objects.*
  - *Id_Nlab:”abbreviation”.*

UN_Specialized_Agencies:UN_Organs:Organization { org_code:"UN_007_002", name:"International Civil Aviation Organization ", **abbreviation:"ICAO"**, link_code:"UN_007", Id_Udsc:"UN_Org", **Id_Nlab:"abbreviation"**}
UN_Specialized_Agencies:UN_Organs:Organization { org_code:"UN_007_003", name:"International Fund for Agricultural Development ", **abbreviation:"IFAD"**, link_code:"UN_007", Id_Udsc:"UN_Org", **Id_Nlab:"abbreviation"**}
Continent_Region { **name:"Eastern Europe"**, ISOCode:"151", Id_Udsc:"ISO_3166", **Id_Nlab:"name"**}

Cypher references

Node creation

It is important to evaluate how the nodes will be updated in the graph, and have in mind that some *cypher commands* could create multiple nodes with ***similar information***. This could create duplicity in the graph.

All the commands containing the MERGE cypher syntax need to be reviewed, due the MERGE command executes the MATCH and CREATE to **whole** pattern. This indicates if for some reason a node with ***partial*** match exists, it will NOT be considered in the operation.

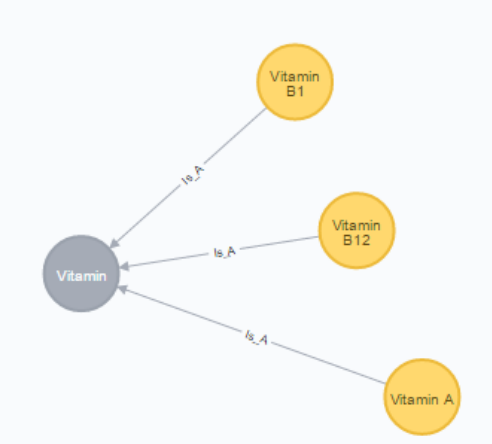
Labels considerations

Labels are a powerful tool to identify nodes into the graph. These labels need to be designed properly to avoid confusion in the node identification.

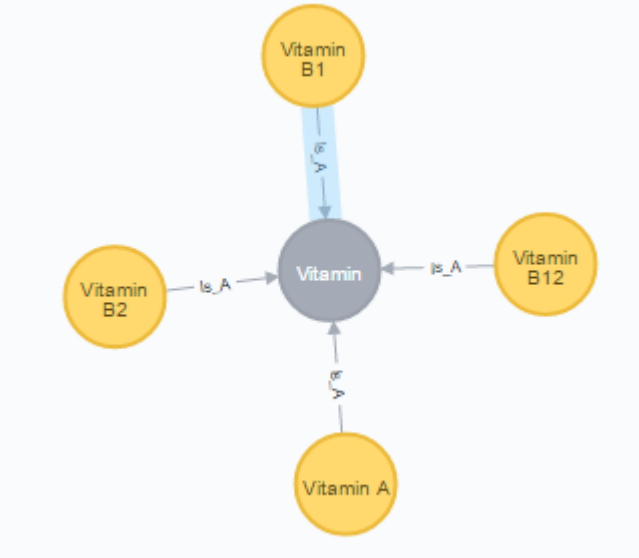
Example to create a confusion in the graph

In this example using the command MERGE and adding multiple LABELS to the same node, we will show how easy is to create a confusion to Graph viewer.

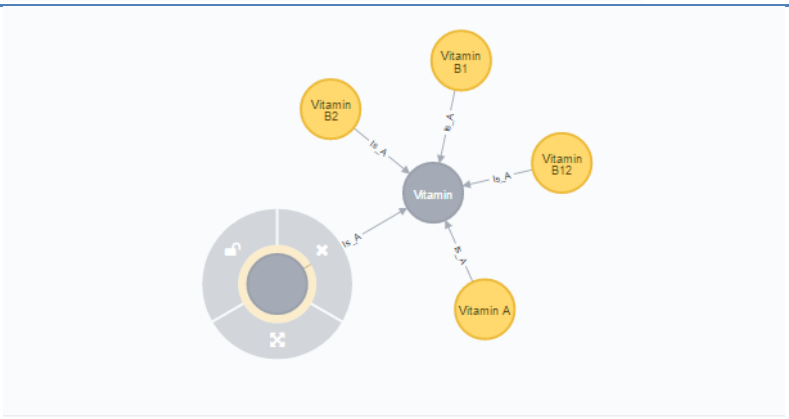
With only few nodes it is very easy to detect and rectify the Graph behavior, but when you have multiple definitions and thousands of nodes, this could be very complex, and the Graph analysis could lead to a misinterpretation of the information.

Cypher	Resulting Graph
Merge (toNutri005:Nutrient { name:"Vitamin"}) Merge (toVita002:Vitamin { common_name:"Vitamin B1", other_name:"Thiamine" }) Merge (toVita002)-[:Is_A]->(toNutri005)	

Merge (toNutri005:Nutrient { name:"Vitamin"})
Merge (toVita003**:Nutirent**:Vitamin { common_name:"Vitamin B2", other_name:"Riboflavin" })
Merge (toVita003)-[:Is_A]->(toNutri005)

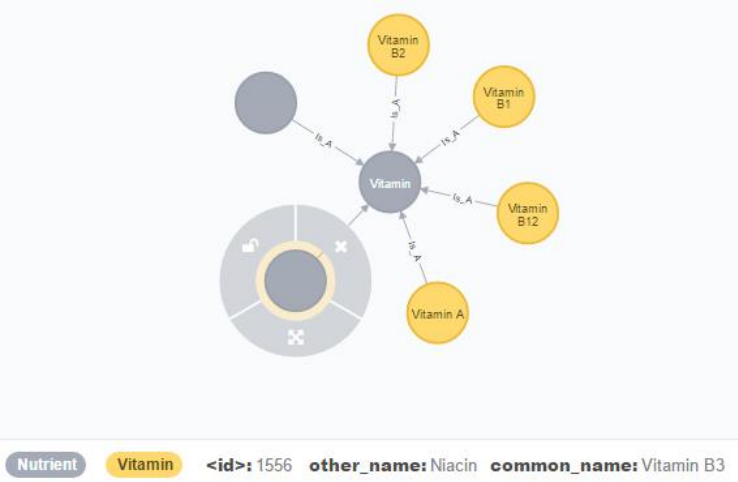


Merge (toNutri005:Nutrient { name:"Vitamin"})
Merge (toVita003**:Nutrient**:Vitamin { common_name:"Vitamin B2", other_name:"Riboflavin" })
Merge (toVita003)-[:Is_A]->(toNutri005)



In this example, the previous cypher created a Node for Vitamin B2, in this cypher, trying to correct the misspelled **Label**, it created a new label with the node included.

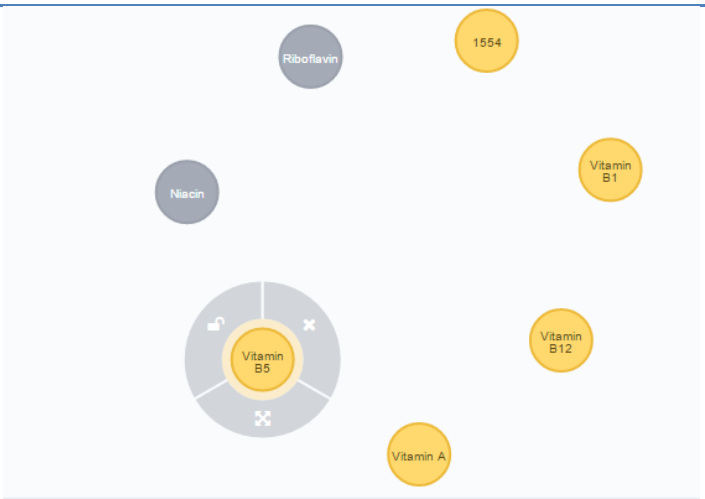
```
Merge (toNutri005:Nutrient { name:"Vitamin"} )
 Merge (toVita003:Vitamin:Nutrient { common_name:"Vitamin B3", other_name:"Niacin" } )
Merge (toVita003)-[:Is_A]->(toNutri005)
```



Nutrient Vitamin **<id>:** 1556 **other_name:** Niacin **common_name:** Vitamin B3

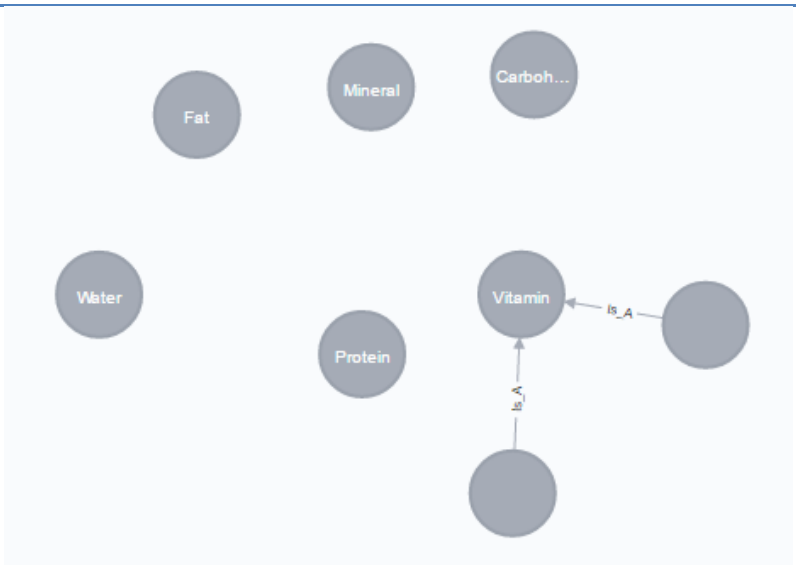
In this example the cypher create the node for Vitamin B3, but the name is not displayed correctly. This is related that the property to display is related to the **label Nutrient.**

```
Merge (toNutri005:Nutrient { name:"Vitamin"} )
 Merge (toVita005:Vitamin { common_name:"Vitamin B5", other_name:"Pantothenic acid" } )
Merge (toVita005)-[:Is_A]->(toNutri005)
```

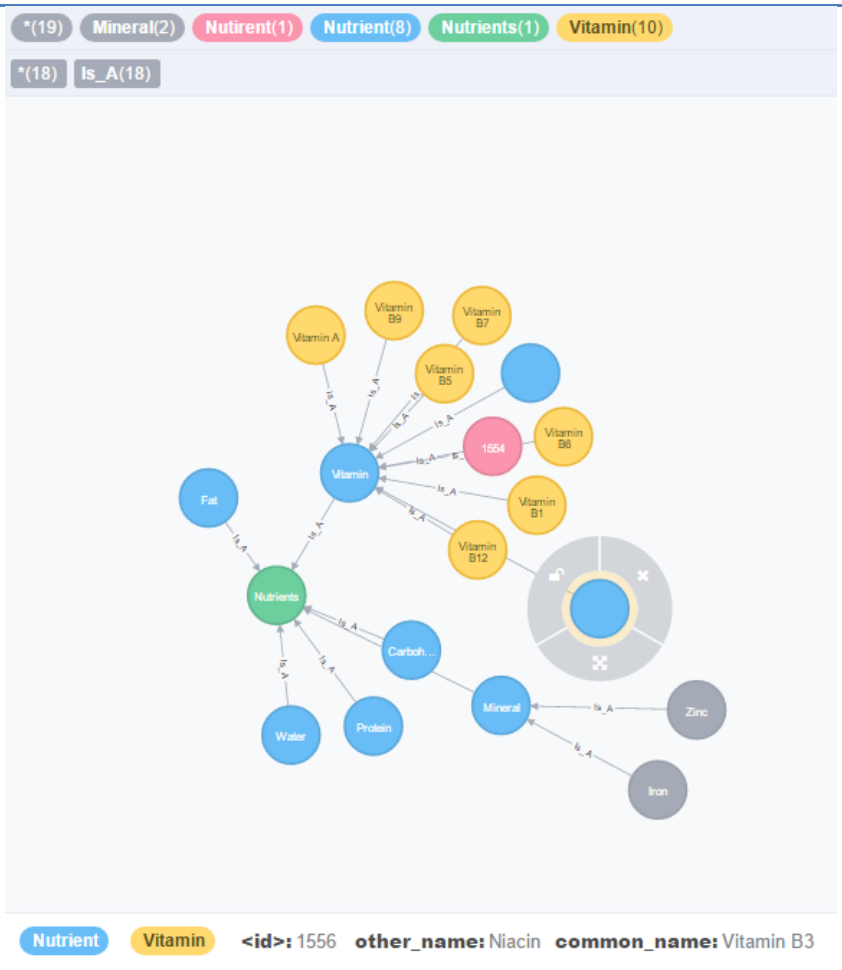


Vitamin **<id>:** 1557 **other_name:** Pantothenic acid **common_name:** Vitamin B5

In this case, The Vitamin B5 is created with only ONE label, and with ONE relationship to the **Vitamin *NODE*** with the **label Nutrient,** as we can see in the next graph.



It appears that we have two "*Nutrients" with no identification, which in reality are two Vitamins with the label Nutrient assigned.*

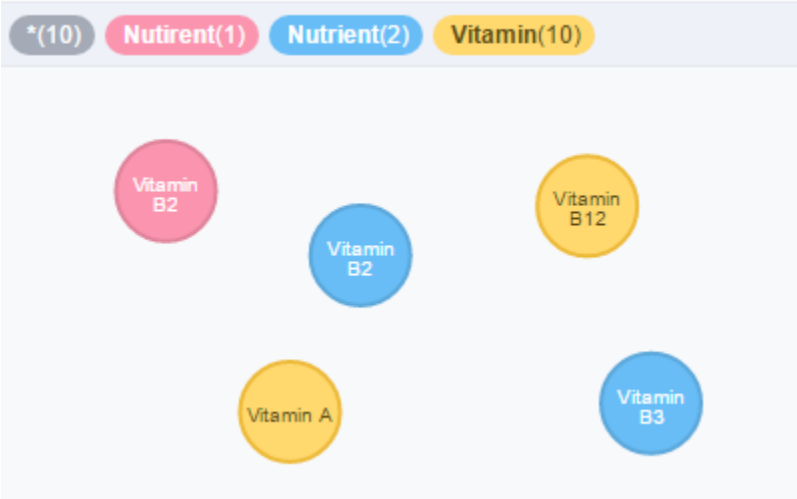


If we select the **label** "**Nutrient**", *blue circles,* we will see that 2 circles with no text are displayed. These

2 nodes are the nodes “vitamin” with the additional label “Nutrient” included.
In this case the graph could be correct, a vitamin is a nutrient.

But if we select the “**label” “Vitamin”** we will see more nodes than the number of existing vitamins, which means that we have duplicated nodes in the graph.

```
$ MATCH (n:Vitamin) RETURN n LIMIT 25
```



B2 has two nodes.
B2 has one node pointing to the wrong label.
B3 has one node and 2 labels, using the “Nutrient” label as a main label.

Fixing the Graph

“It is not possible to define that the graph is wrong, it is possible to define that it is not what we wanted.”

Fixing the double label assignment

We have:

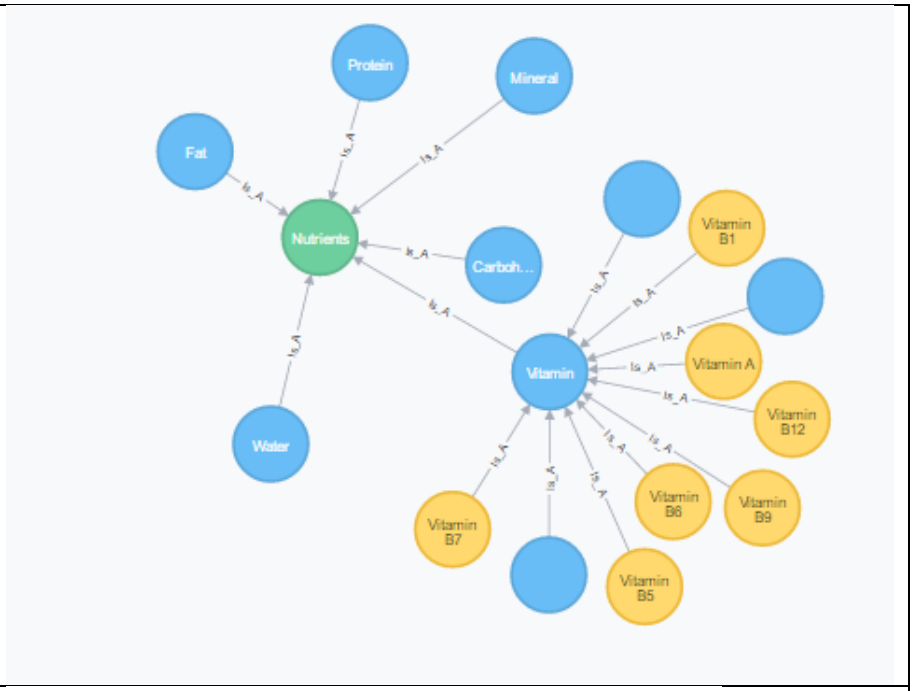
- One label was misspelled. (*Nutirent*)
- One additional label was added by design.
- The fix could be node by node.
- The fix could be selecting a group of nodes via “label”.

In this case we will:

- Fix the label “Nutirent” to “Nutrient” ONLY to the nodes with the label “Vitamin” included, assuming that was the desired action. (*With this procedure we are **duplicating** the node.*)

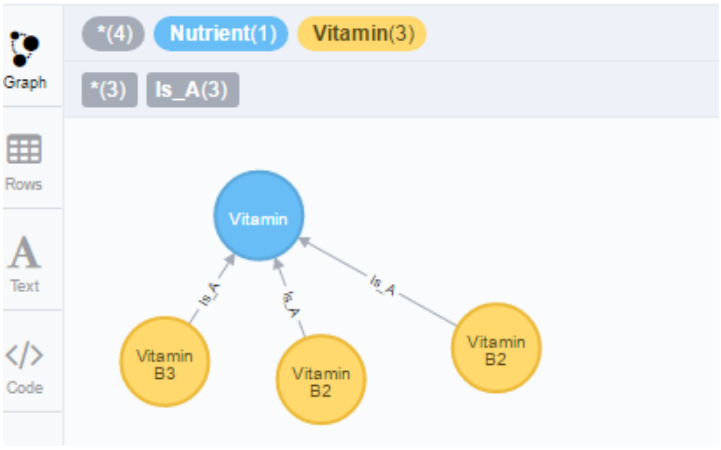
- This cypher, will **get** all the nodes with the label, **add** the label Nutrient, and **delete** the label Nutirent.

<table>
<tr>
<td>MATCH(n:Nutirent) SET n:Nutrient REMOVE n:Nutirent RETURN n

After this cypher we don’t know that we have duplicated “vitamins”, we will see that after we remove the “Nutrient” label</td>
<td> 
</td>
</tr>
<tr>
<td>MATCH (n:Nutrient:Vitamin) REMOVE n:Nutrient RETURN n

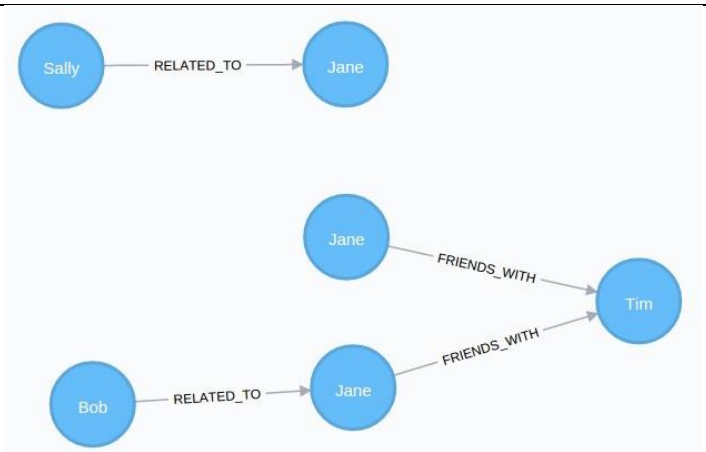
This cypher will select the nodes and remove the label.

Now we can see duplicated nodes, and if we doubled click on any of the vitamins we will discover that the link is correct.

Which one of the 2 B2 we will delete.</td>
<td>$ MATCH (n:Nutrient:Vitamin) REMOVE n:Nutrient RETURN n
 
</td>
</tr>
</table>

- Removing the duplicates. This process needs more than one step.

MATCH (n:Vitamin) WITH n.common_name as id_dup, collect(n) as nodes WHERE size(nodes) > 1 RETURN nodes	This cypher will find the nodes with duplicated properties, using one common label and one or more matching properties. This is to review the nodes to delete. **Note:** To be able to delete the node, it is necessary to delete the relationships of the node.
MATCH (n:Vitamin) WITH n.common_name as id_dup, collect(n) as nodes WHERE size(nodes) > 1 WITH head(nodes) as theNode, tail(nodes) as dupNodes LIMIT 1000 UNWIND dupNodes as delNodes OPTIONAL MATCH (delNodes)-[r]->(otherNodes) OPTIONAL MATCH (delNodes)<-[r2]-(otherNodes2) RETURN delNodes	This cypher will find the duplicated nodes and their relationships. This cypher could take long time. It could be narrowed if we know the relationships that we want to delete. Like **r**: **Is_A**
The full command: MATCH (n:Vitamin) WITH n.common_name as id_dup, collect(n) as nodes WHERE size(nodes) > 1 WITH head(nodes) as theNode, tail(nodes) as dupNodes LIMIT 1000 UNWIND dupNodes as delNodes OPTIONAL MATCH (delNodes)-[r]->(otherNodes) OPTIONAL MATCH (delNodes)<-[r2]-(otherNodes2) DELETE r, r2, delNodes	 The final graph for the label “Vitamin”.

Generic Example: MATCH (n:Person { name:"Jane" }) WITH collect(n) AS janes WITH head(janes) AS superJane, tail(janes) AS badJanes UNWIND badJanes AS badGirl OPTIONAL MATCH (badGirl)-[r:FRIENDS_WITH]->(other) OPTIONAL MATCH (badGirl)<-[r2:RELATED_TO]-(other2) DELETE r, r2, badGirl WITH superJane, collect(other) AS friends, collect(other2) AS related FOREACH (x IN friends | MERGE (superJane)-[:FRIENDS_WITH]->(x)) FOREACH (x IN related | MERGE (x)-[:RELATED_TO]->(superJane))	

Adding additional labels

It is possible that some “existing” nodes need to be updated with a new “label” and this could be done in this way:

- Setting a label to an existing node: MATCH (n: {id:*desired-id*}) SET n :*newLabel* RETURN n
- Setting a new label to al nodes with an specific label: MATCH (n:*existingLabel*) SET n:*newLabel* RETURN n