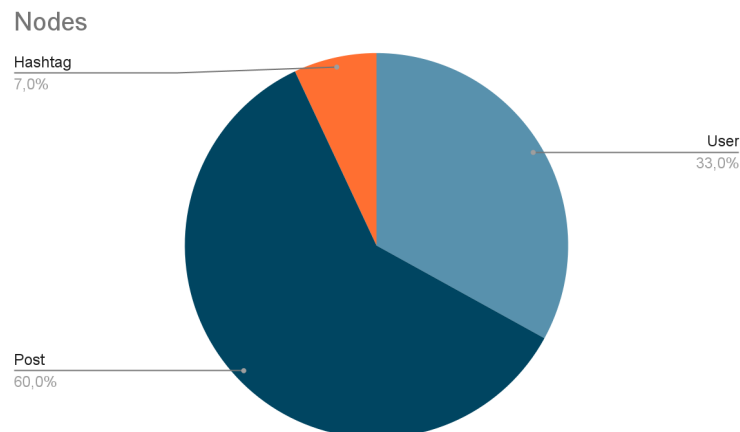


This document presents different query archetypes with visualization ideas.

Archetypes

1. Statistics (Quantities)

- Number of Nodes
- Number of Relations
- Can be visualized in a pie chart

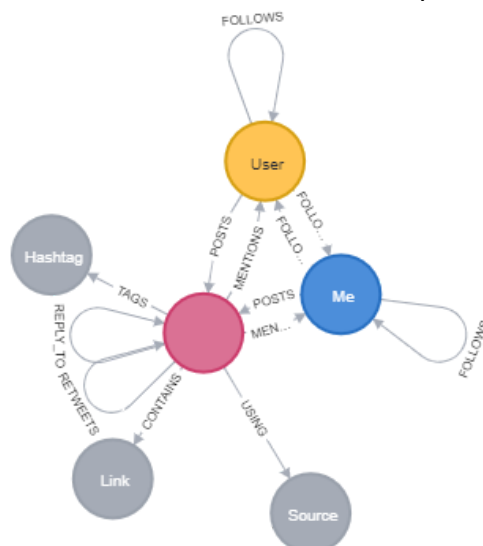


2. Node/Rel Information

- e.g. List all the attributes of a node/rel
- e.g. List all the outgoing rels / connected nodes
- e.g. Constraints (may be too technical)
- Can be visualized in a (nice) table or just a list, maybe a [expandable list](#)

3. Graph Visualisation

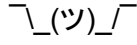
- Which nodes have what connection to others?
- Must be filterable to reduce complexity



c.

4. Shortest Path

- Find the shortest path between two nodes

- b. Visualized via a network/tree graph -> [Examples Dendrograms and Trees](#)
- 5. Centrality -> [Existing Neo4j Algorithms](#)
 - a. Find the most important nodes
 - i. PageRank -> Importance
 - ii. Betweenness Centrality -> Amount of Influence a of a node over the flow of information in a graph
 - b. Visualization
 - i. Graph through thickness of a node (maybe too large)
 - ii. Table (searchable, sortable → easier)
- 6. Communities
 - a. Louvain Algorithm
 - b. Visualization: 
 - i. Expandable Table → Not really nice, but does its job
 - ii. Graph → Not wished by stakeholders
- 7. [WorldMap](#) → Where is which part of the production line?
 - a. Really nice, but too application-oriented
- 8. List all paths from one node to another and their commonalities
 - a. Visualization: Linearize each of the paths

x -- y -- O -- z

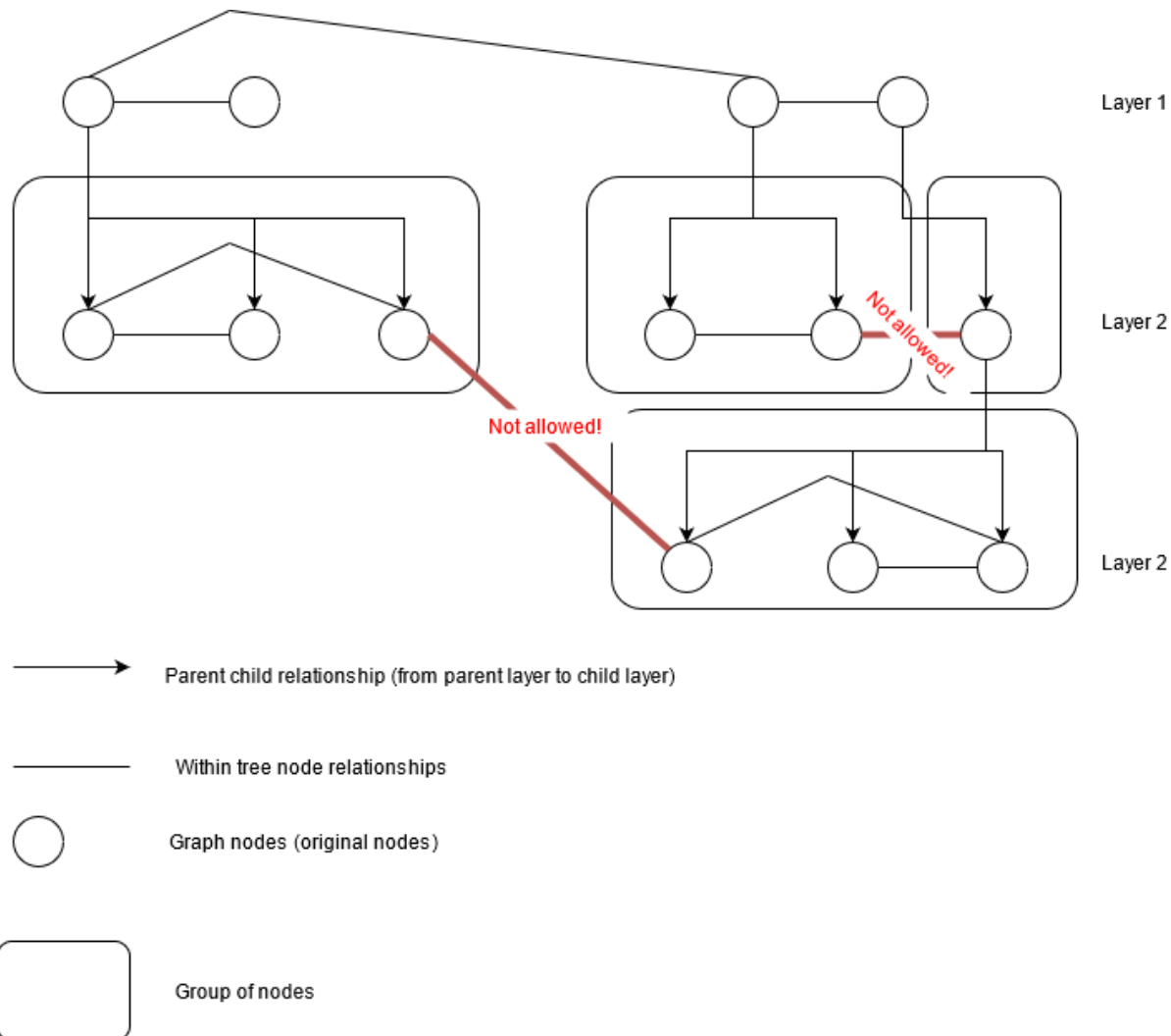
where O is a node that replaces a complete subgraph
 - b. Visualize common subgraphs / nodes

x -- y -- O1 -- C -- z

a -- O2 -- C -- d

where O1, O2 is a node that replaces a complete subgraph

where C is the subgraph both have in common (possibly not a single node, but a complete subgraph)
 - c. Maybe display the common node / subgraph (C) in a secondary view.
- 9. Hotness of a graph
 - a. Where is a graph hot with respect to a certain criterion
 - i. Last date of change
 - ii. Number of changes
 - iii. Number of connections
 - iv. etc.
 - b. Visualization: Subdivide the graph into subgraphs clustered by "hotness" and display them as bubbles in different colors.
- 10. Layer the nodes (tree-ify)
 - a. Description: Try to arrange the nodes in layers. Each layer consists of a group of nodes where each node in this group has the same "parent node". Each group of nodes can have internal relationships. Relationships to other groups of nodes (except for the parent node and the child nodes) are not permitted. A group of nodes can only have a single parent.
 - b. Use case: GIS data that represents actually hierarchical data (like countries, regions, cities, etc.) are "enhanced" with information. For example partner-towns or roads between cities and countries. Group this back to a hierarchical representation.



11. Greatest roundtrip

- Description: Pick the (closed or non-closed) path through a (filtered) graph that contains the greatest number of nodes (or measured by another criterion) where the sum of the distance between the nodes is not greater than a given threshold
- Use case: As a tourist I want to plan a tour through the city that I am visiting. I have interests, so some nodes are more interesting to me. I want to walk a certain distance at max. I want to end the tour where I started it (at the hotel). Plan me a tour pls.
- Visualization: Maybe just a linear list of nodes

12. Query by example

- Description: Get all the nodes of a certain type that are equal in properties to a node I selected (for example that they have both the same node linked to)
- Use case: I want to watch a movie of a certain genre (type of node). I give the system the information which movies I like. Movies are linked to other nodes that represent their properties. Get me a movie I like to watch.

13. Cluster around given node (not necessarily centrality of the graph)

- a. Try to cluster connected nodes of a specific node in a given number of sets, so that there are as few connections between the sets (not including the central node) as possible.
- b. Use case: I as a person in a network of projects and companies want to know how important I am for the projects and companies to work together.
- c. Other use case: I as the owner of a company have multiple projects that are for multiple customers. My company, as the central node, has too much work to do. I want to cluster the work items in such a way that the effects of killing one project does not (or not much) affect the other projects.

Frameworks

The following frameworks can be used to for visualization:

- [vis.js](#) (MIT or Apache)
 - Compatible with React over [3rd party project](#)
- [visx](#) (MIT)
 - Looks very promising
 - React ready
 - Also supports trees and networks
- <https://openbase.com/js/react-graph-vis> (MIT) ← Das ist das 3rd party project vom 1.
Stichpunkt - Nice coincidence 😄
 - Demo: <http://crubier.github.io/react-graph-vis/>