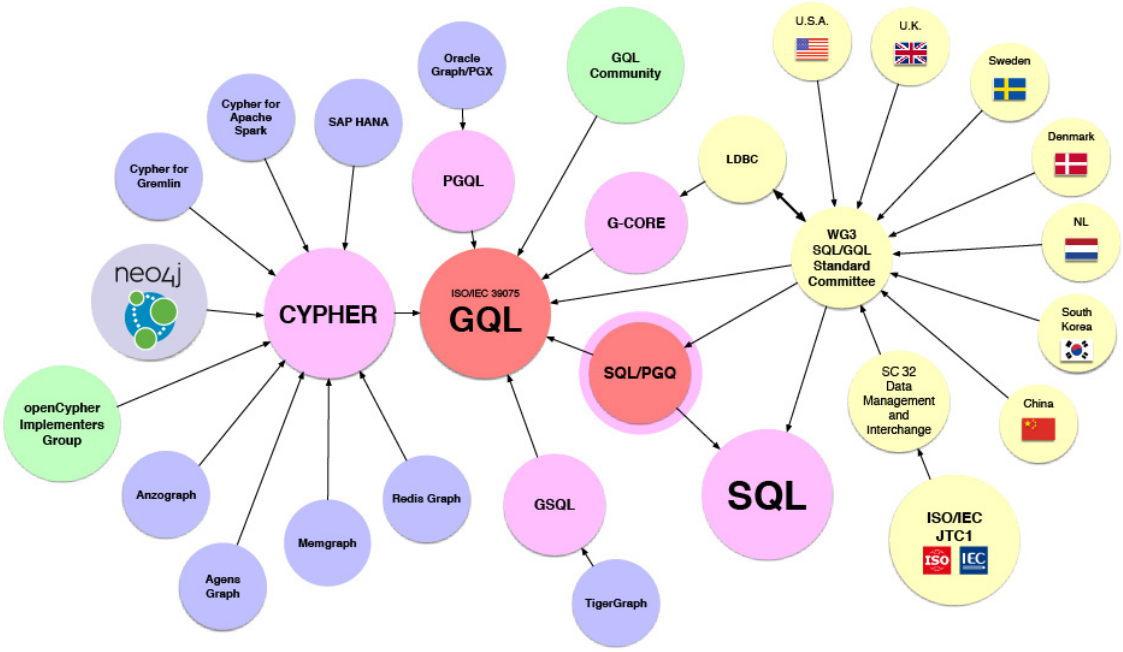
Graph Databases

Storing data without predefined models



David Hellinga

Semester 3 Lente 2022

Introduction

For Forged in the Lore I will need to store information with a flexible structure - after all, users get to define a lot of things themselves. Due to this it might be convenient to use a different storage method than the standard relational database. Graph databases are a possible option and in this document I will be researching whether they are a suitable solution for FitL

Research Questions

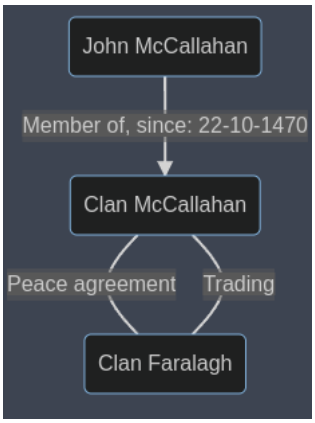
Is using a graph database for storing the Setting and Campaign information in FotL a good solution?

1. What kind of data will FotL store?
2. What are graph databases?
3. What are the strengths and weaknesses of graph databases?
4. Do the findings of 2 and 3 accommodate the requirements of 1?
5. What options are there for graph database frameworks?

Q1: What kind of data will FotL store?

Method: Problem Analysis

The purpose of FotL is to store all long term information for a story setting. Since there are a lot of different options that a writer might want to store, flexibility is required. Besides wanting to store lots of different entities - characters, places, objects, concepts, etc. - there can also be vast differences in the number of characteristics an entity has - one village might have a name and description while another might have a defined population, culture, whether it's fortified or not or anything else. Due to this it is important that users can define entities for themselves, both in name and in number and type of properties



The second data type that needs storing is the relation between those entities. This also needs to be highly customizable since there can be any number of different relationship definitions including extra data. As an example a character could be a member of a faction from a certain date. In addition, there can be multiple relations between two entities - two factions can have a peace agreement, be trading partners and have one of the two be a subject of the other.

This means our data storage has the following requirements:

1. Store many different entity types
2. Allow entities of the same type to have different properties
3. Store many different relations between the entities
4. Allow multiple relations between two entities
5. Allow relations to have extra properties

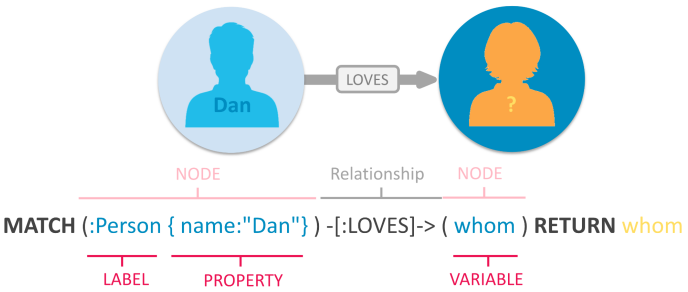
In addition to the data format related requirements we can also set a few meta requirements:

1. The primary goal is to store and visualise data.
    a. Once the data is present it is mostly read and not edited that often.
    b. It's important that users can easily understand the visualisation of the data.
2. It is acceptable to use multiple database systems if that is found to be appropriate

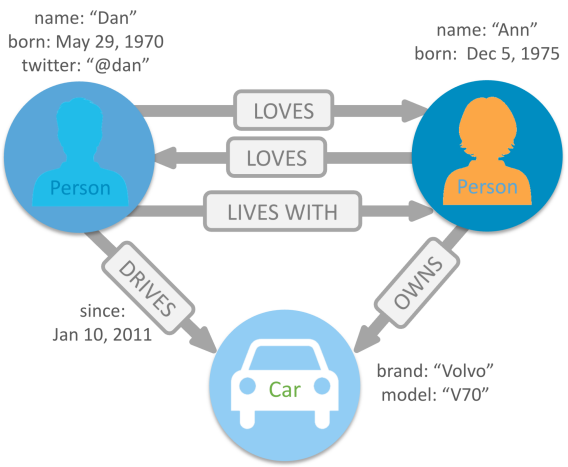
Q2: What are graph databases?

Method: Literature Study

"A graph database stores nodes and relationships instead of tables, or documents. Data is stored just like you might sketch ideas on a whiteboard. Your data is stored without restricting it to a predefined model, allowing a very flexible way of thinking about and using it." - Neo4j introduction to graph databases



The databases most of us are used to - relational databases - store data in a strictly defined format where predefined tables contain data that has to comply with that table's definition and relations are usually defined via their own strictly defined relational tables (*Relational Database*, n.d.). Flexibility is possible but usually comes in the form of abstraction - for example, by introducing a new table that contains definitions used by another table.



Graph databases handle data in a different way: data is stored in nodes, with each entity having its own node to which any number of properties can be added in the form of key-value pairs, while labels are used to specify the type of a node. Relations are stored as is - as their own directed and named objects that connect two nodes.

Relationships can also store their own properties and any number of relationships can exist between two nodes. (*What Is a Graph Database? - Developer Guides*, n.d.)

Graph databases also use their own query languages - which one depends on the platform - that are optimised for graphs and are based on the node-relationship-node structure of graph databases.

Q3: What are the strengths and weaknesses of graph databases?

Method: Literature Study

(Rund, 2017) | (*What Is a Graph Database? {Definition, Use Cases & Benefits}*, 2021) | (Schulz, 2020)

Advantages

1. Flexibility: Data can be easily changed and extended. Definitions can be altered dynamically and support the storage of formats that aren't specified ahead of time.
2. Representation of relationships: Relationships are represented in a format that is much closer to how most people visualise them, thus making it easier to understand them. This also means that the data is easier to display to users since a one-to-one display can be understood by most people.
3. Explicit relationships: Relationships can represent much more information without adding complexity to the database.
4. Relationship based search: The explicit relations allow for much faster traversal of multiple levels of relationship and makes relationship based searches significantly faster.
5. Query speed and scaling: The query speed of graph databases is only dependent on the number of concrete relationships the entities have, not on the total data volume in the database, because only data that is directly or closely related to the queried relationships is being read. Relational databases on the other hand depend both on the number of tables being joined and the amount of data in those tables because all of it has to be processed to resolve queries.

Disadvantages

1. Lack of standardised query language: Different graph database platforms use different languages for their queries, unlike relational databases which all support basic SQL in addition to any platform specific extensions. This makes switching platforms difficult.
2. No ORM/OGM: Graph databases often require manual querying since no or limited ORM's are available (For example Neo4J only has an OGM for Java).

3. Inappropriate for transactional systems: Graph databases are slow for processing large volumes of transactions and for queries that span the whole database.
4. Smaller user base and less support
5. Rapidly evolving technology: Makes comparing different platforms difficult because they change often
6. Possible missing operational features such as the rollback mechanism of transactions, data recovery options and issues with durability, very platform specific which issues are present.

Q4: Do the findings of Q2 and Q3 accommodate the requirements of Q1?

Method: Reflection of Literature Study on Problem Analysis

1. *Store many different entity types*

Labels allow us to have predefined generic entity types (eg. person, faction, item, etc.) while a property can be added to the entity to specify it's exact type. This allows for easy searching and grouping based on those labels while the property allows for more depth in typing and allows users to specify their own types. Thus this is possible

2. *Allow entities of the same type to have different properties*

Nodes can have any number of properties and still be identifiable based on their label, predefined properties and relationships. Thus this is possible

3. *Store many different relations between the entities*

Relationships, just like nodes, can have their own label in the form of a name and have properties to further specify the type of relationship.

4. *Allow multiple relations between two entities*

Graph databases allow for any number of relationships between two nodes.

5. *Allow relations to have extra properties*

Relations have their own properties.

1. The *primary goal is to store and visualise data.*
    a. *Once the data is present it is mostly read and not edited that often.*

This means that the downside of not being suitable for transactional systems is not an issue for this use case.

    b. *It's important that users can easily understand the visualisation of the data.*

The representation of relationships being close to how we would visualise them on a whiteboard makes them easy to understand for users. A javascript library such as vis.js will be necessary to actually display the data.

2. *It is acceptable to use multiple database systems if that is found to be appropriate*

Due to the transactional nature of something like an account system or chat functionality graph databases are not a good solution for those. Thus a second relational database will be required to store that data and some method will be necessary to link ownership of data in the graph database to the users from the relational database.

The lack of standardised language is not much of an issue due to there being no preexisting knowledge in the team. The varying nature of data means that an ORM would not be suitable anyways so that's not an issue for this system.

Q5: What options are there for graph database frameworks?

(*What Is a Graph Database? - Developer Guides*, n.d.)

Method: Multi-criteria decision making and Community Research

As with all technology choices there are many different platforms and vendors for graph databases. Many of the differences are not that relevant for a project such as this so I will mainly be looking at three things: The method of interacting with the DB - the **query language or OGM** -, popularity and compatibility with the rest of the project. **Popularity** is mainly important due to the ease of acquiring information/support and because it indicates how relevant knowledge of it is as a software engineer.

Graph database frameworks can be split into two categories: Pure graph databases and multi-model databases. (*Multi-Model Database*, n.d.) A pure graph database only supports a graph structure while a multi-model database supports other structures such as relational or document storage. This has the advantage of limiting the number of storage technologies used in the project's tech stack but also leads to a more monolithic architecture. The advantages of using a pure graph database with a separate relational database are familiarity - using a DB like PostgreSQL - and more likely built-in support in the backend framework. In this project that would take the form of Entity Framework Core handling the relational database which would also give access to tools like EF Core Identity.

There are a lot of different options for graph databases and with the very limited knowledge I have it's impossible to properly evaluate the options. Community ratings on pages like G2 (*Best Graph Databases in 2022*, 2020) and DB-Engines (*Graph DBMS*, n.d.) consistently put Neo4J at the top. This coincides with Neo4J having the largest market share,

satisfying the requirements and being of personal interest to me. Based on that Neo4J seems to be the best option.

Conclusion

Is using a graph database for storing the Setting and Campaign information in FotL a good solution?

Possibly. For storage and reading the data it is a good solution, but whether it will properly accommodate the type of writing we want to be able to do is not certain. This would require fully designing how this process should work and prototyping. Due to time constraints this will not be possible.

This leads us to the conclusion that Neo4J is a viable database platform to build FotL on but that - once the concrete requirements for how users should be able to store data are clear - an extra evaluation is necessary by prototyping the system.

References

Best Graph Databases in 2022. (2020, February 26). G2. Retrieved June 1, 2022, from https://www.g2.com/categories/graph-databases

graph DBMS. (n.d.). DB-Engines. Retrieved May 30, 2022, from https://db-engines.com/en/ranking/graph+dbms

Multi-model database. (n.d.). Wikipedia. Retrieved May 30, 2022, from https://en.wikipedia.org/wiki/Multi-model_database

Relational database. (n.d.). Wikipedia. Retrieved March 10, 2022, from https://en.wikipedia.org/wiki/Relational_database

Rund, B. (2017, March 10). *The Good, The Bad, and the Hype about Graph Databases for MDM | Transforming Data with Intelligence*. TDWI. Retrieved March 14, 2022, from https://tdwi.org/articles/2017/03/14/good-bad-and-hype-about-graph-databases-for-mdm.aspx

Schulz, Y. (2020, August 7). *A quick primer on graph databases*. IT World Canada. Retrieved March 10, 2022, from https://www.itworldcanada.com/blog/a-quick-primer-on-graph-databases/434215

What is a Graph Database? {Definition, Use Cases & Benefits}. (2021, April 22). phoenixNAP. Retrieved March 10, 2022, from https://phoenixnap.com/kb/graph-database

What is a Graph Database? - Developer Guides. (n.d.). Neo4j. Retrieved March 10, 2022, from https://neo4j.com/developer/graph-database/