NoSQL

The final words

What are the origins of NoSQL

- A Twitter hashtag ;)
- The Problem of computation and storage on multiple small machines

What are the two types of “NoSQL” database

According to Martin Fowler there are basically only 2 Types of NoSQL databases.

- Aggregation-Based ones (key-value, column based, document based, ….)
- Graph Databases

What was the main reason for the need of NoSQL

The Problem of data VS computation locations and hence multi-processor-clustering how do I get the data in a reasonable amount of time to the next cpu. That's not even that easy if you just got 1 machine with multiple cores.

Which, if you got 4000 CPUs and 4000 Harddisks distributed over several square meters, naturally will take a lot of time and energy (lots of time for a computer) -> Just to think about it 93769 meters per milliseconds is the approximate speed of electrical potential in a copper wire ... Which is f*cking fast …. But as soon we're adding the millions of fetch/read/write cycles needed to locate and coordinate data if you add this all up this calculation reaches enormous complexity…. I digress ...

And here is the dilemma! SQL was designed to save the data in a way that you would have to create multiple small databases to comply to "ACID" and this gets increasingly difficult as you scale and replicate.

NoSQL to the rescue?

NoSQL means nothing it’s just a stupid hashtag, because SQL is just a Language Definition and not a specific implementation. The problems their creators wanted to fix are not per se the ones of SQL more of the implementations. (personal opinion please correct me if I’m wrong)

So what they created and tried to achieve is more of a clustering concept and how to solve distributed data aggregations.

excluding graph databases they have a different purpose

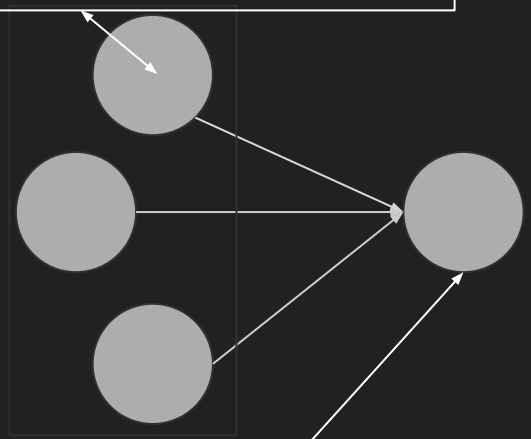
When to use what

- How does that data come in ?
- What do I need to access and how ?
  - Do I need to transform the data so the access is faster?
  - Do I need to transform the data so I use the minimal amount of space?
- Do I have a lot or reads or a lot of writes ?
- How important is consistency
  - Do I need foreign keys or references ?
  - Where do I want to handle the consistency (inside of the Application or the Database?)
  - How do I replicate ?
- Do I have a distributed “isomorphic/container” system or do i have role based clusters “this is my SQL-Server”

Let’s just explain the pro of Graph databases

The Question of “How and What” is key …. How to store and access information!

I am a SQL n:m relation entity i know what is related



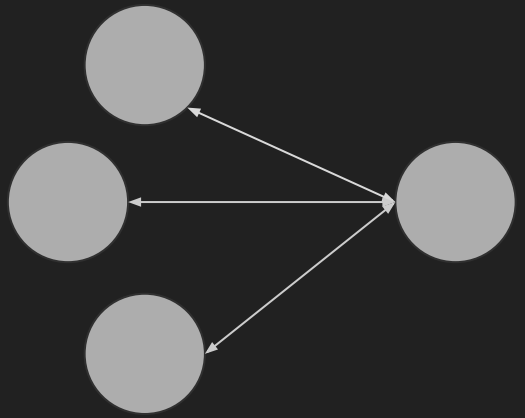
I am the data that gets related via n:m relation
I have no clue what/who I’m related to

SQL stores the information of the relation INSIDE of the Node. Which is neat if you wanna access the data that’s related to selected entities but it gets more complex as soon as you don’t wanna know the relation of data in the entity but relation of the entity to another entity stored nor inside the point of destination or origin.

Simple example -> i have 10 n:m relations to a table called processes so the 10 relation tables know which process_id is mapped to which arbitrary other table. In the other table we assume there is a 1:n relationship so you can always find out which process is mapped to the entity.

Let’s turn it around if we wanna know how to get the specific arbitrary table ID we would have to left join all relation tables and only then we would know which one is connected. Ofc there is the trick where we store an type to infer the correct table but this is once again more complex to formulate and compute

Let’s just explain the pro of Graph databases



A graph databases stores the relation as edges and does not abstract it away into an entity. Hence I always know what is connected to what and it’s rather fast and easy to find out what is related to what and this ofc can be done in deep nested structures.

Example: If you have a timeline and you wanna have the feeds of the people who like a picture on another tag this is where graph databases get interesting.

So long and thx for the fish