NoSQL

# Overview

NoSQL database systems are alternative models for data management compared to standard relational SQL databases. Due to their flexibility NoSQL databases can be very good at some tasks, but it is important to know the differences from SQL systems.

# Why

Relational Database Management Systems (RDBMS) which use SQL as standard can be queried very flexibly, with a wide range of standard queries which can be used and combined throughout databases. Since queries optimisation doesn't depend on schema design much, the schema can be normalised and managed easily. The with RDBMS is the queries can be quite expensive and do not scale particularly well in high traffic situations.

NoSQL databases such as MongoDB and AWS DynamoDB can be query very efficiently in a limited number of ways, any queries outside of these can be very expensive and slow. Therefore, NoSQL databases must be designed with the schema tailored to the most common requests on the database, allowing for very quick performance for use cases.

# Design

As mentioned previously, NoSQL design should be thoughtout before creation of the databases in order to allow for efficient queries. In general, as few NoSQL tables should be kept as possible and the database should be designed around three access paterns:

* Data Size - Knowing the amount of data being stored with help determine the most effective way to partition the data
* Data Shape - Instead of reshaping data when processed, a NoSQL database should organise its data to return the correct shape when queried
* Data Velocity - DynamoDB and others scale by increasing the number of phyical partitions available to process. By distributing the data across the partitions, the I/O can be best used to capacity.

Once access design is realised, data structure should be designed using the following principles:

* Keep related data together - Data which is related should be kept within close proximity and therefore there should be very little tables
* Use sort order - Related items can be grouped and queried effectively if their design causes them to sort together, i.e. use specific sortable primary keys
* Distribute queries - Queries should be designged to avoid hot spots and distribute traffic across as much of the database as possible
* Use global secondary indexes - Enable different queries than the main table can support while keeping them fast and inexpensive.