Features of NoSQL Databases

In this lesson, we will discuss the features of NoSQL databases.

We'll cover the following

In the introduction, you learned that the *NoSQL* databases are built to run on clusters in a distributed environment, powering Web 2.0 websites.

Now, let’s go over some features of *NoSQL* databases.

Pros of NoSQL databases#

Besides design, *NoSQL* databases are also developer friendly. What do I mean by that?

Gentle learning curve#

First, the learning curve is less than that of relational databases. When

working with relational databases, a big chunk of our time goes into learning how to design well-normalized tables, setting up relationships, trying to minimize joins, and so on.

Schemaless#

One needs to be pretty focused when designing the schema of a relational database to avoid running into any issues in the future.

Think of relational databases as a strict headmaster. Everything has to be in place, neat and tidy, and things need to be consistent. However, *NoSQL* databases are a bit chilled out and relaxed.

There are no strictly enforced schemas, so you can work with the data how you want. You can always change stuff and move things around. Entities have no relationships. Thus, things are flexible, and you can do stuff your way.

Wonderful, right?

Not always!! This flexibility is good and bad at the same time. Being so flexible, developer-friendly, having no joins, relationships, and so on make it good. However, we will talk about some downsides next.

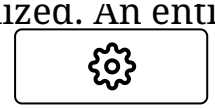
Cons Of NoSQL databases#

Inconsistency#

This introduces the risk of entities being inconsistent. Since an entity is spread throughout the database one has to update the new values of the entity at all places.

Failing to do so, makes the entity inconsistent. This is not a problem with relational databases since they keep the data normalized. An entity

relational databases since they keep the data normalized. An entity resides at one place only.



No support for ACID transactions#

Also, *NoSQL* distributed databases don't provide *ACID transactions*. A few that claim to do so, don't support them globally. They are limited to a certain entity hierarchy or a small region where they can lock down nodes to update them.

> **Note:** *Transactions in distributed systems come with terms and conditions applied.*

Conclusion#

My first experience with a *NoSQL* datastore was with the *Google Cloud Datastore*.

An upside I felt was that we don't have to be a pro in database design to write an application. Things were comparatively simpler because there was no stress of managing joins, relationships, n+1 query issues etc.

Just fetch the data using its key. You can also call it the ID of the entity. This is a *constant O(1)* operation, which makes the *NoSQL* database really fast.

I have designed a lot of *MySQL* DB schemas in the past with complex relationships, and I would say working with a *NoSQL* database is a lot easier than working with relationships.

It's alright if we need to make a few extra calls to the backend to fetch

data in separate calls. This doesn't make much of a difference. We can always cache the frequently accessed data to overcome that.

Popular NoSQL databases#

Some of the popular NoSQL databases used in the industry are *MongoDB*, *Redis*, *Neo4J*, and *Cassandra*.

So, by now, you have a pretty good idea of what NoSQL databases are. Let's have a look at some of the use cases that best fit them.

← Back

NoSQL Databases - Introduction

Next →

When to pick a NoSQL Database?

Mark as Completed

Report an Issue