Relational Database

This lesson introduces relational databases; it, goes over their advantages, uses, and the problems related to them.

WE'LL COVER THE FOLLOWING

Introduction

Today we are facing a rapid expansion of data-driven businesses, especially businesses that are web-based and have an enormous amount of data being transferred every second. In fact, almost 90% of all data on the web has been created in the last several years. Usually this data has to be stored in a *database*. However, since every business is different, there are different needs that need to be taken into account when looking at ways to store the data. There are two routes we can take:

- *Relational Databases*
- *NoSQL*

Further on, you will cover the strengths and weaknesses of both, but it will largely focus on *NoSQL* databases and why we need to consider them as a solution to our problems.

Relational Databases

The concept of relational database concept was first proposed in 1970 by Edgar F. Codd and has been ruling our world since the mid-1980s. Somewhere in the ’90s concepts like Object Databases threatened to take reign, but that never happened. There are many reasons why people love relational databases and why they have been so dominant for such a long time.

Why Use Relational Databases?

- Firstly, they are very close to the way we see the world. The entities in the application domain are modeled naturally using *records* and *tables*. Relationships between these tables are created based on the relationships that exist between the corresponding entities in the application domain.

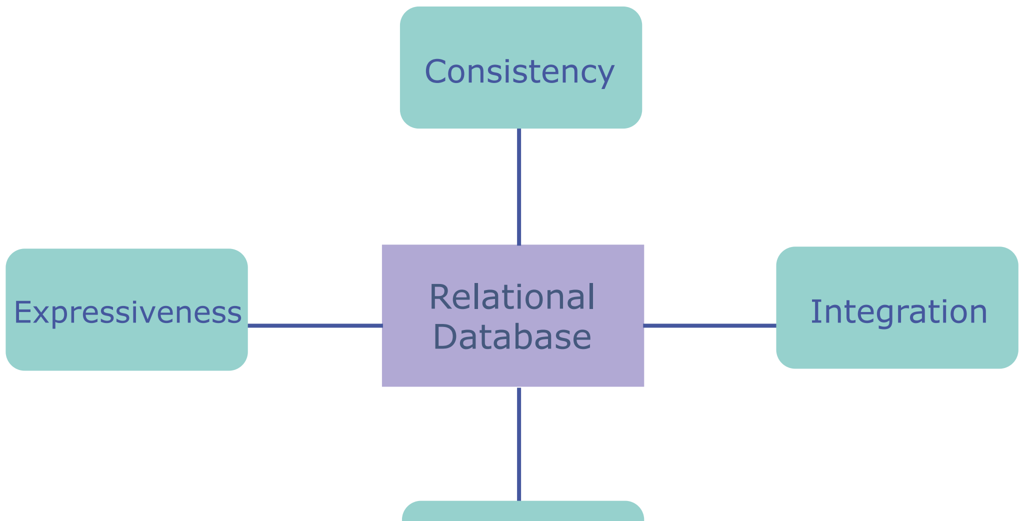
- *Relational databases* also have strong consistency, meaning that once a change is made to some of the records, it can be seen by every application in that domain. This made relational databases an ideal integration mechanism, with multiple applications sharing a common database and data.

- *Relational databases* have the *Transactions* feature. This gives us a way to assign changes to multiple tables and roll back previous changes, in case one instruction fails. This is a fairly complex, but powerful, feature used in data-intensive systems to provide stability.

- They have the *Joins* feature.

- *SQL* has pretty much become the standard language for a wide range of *relational databases*. And since it is very expressive and easy to learn, this made it more accessible to relational databases. Hence, SQL provides expressiveness to the developer.

So, to sum it up, *consistency*, *integration*, *expressiveness*, and *stability* are the reasons we can find relational databases within nearly every business solution today.



Problems of Relational Databases

Our world has changed in the last few years. As already mentioned, it is a world where a large amount of data is stored, transferred, and manipulated—most of which is *unstructured* data.

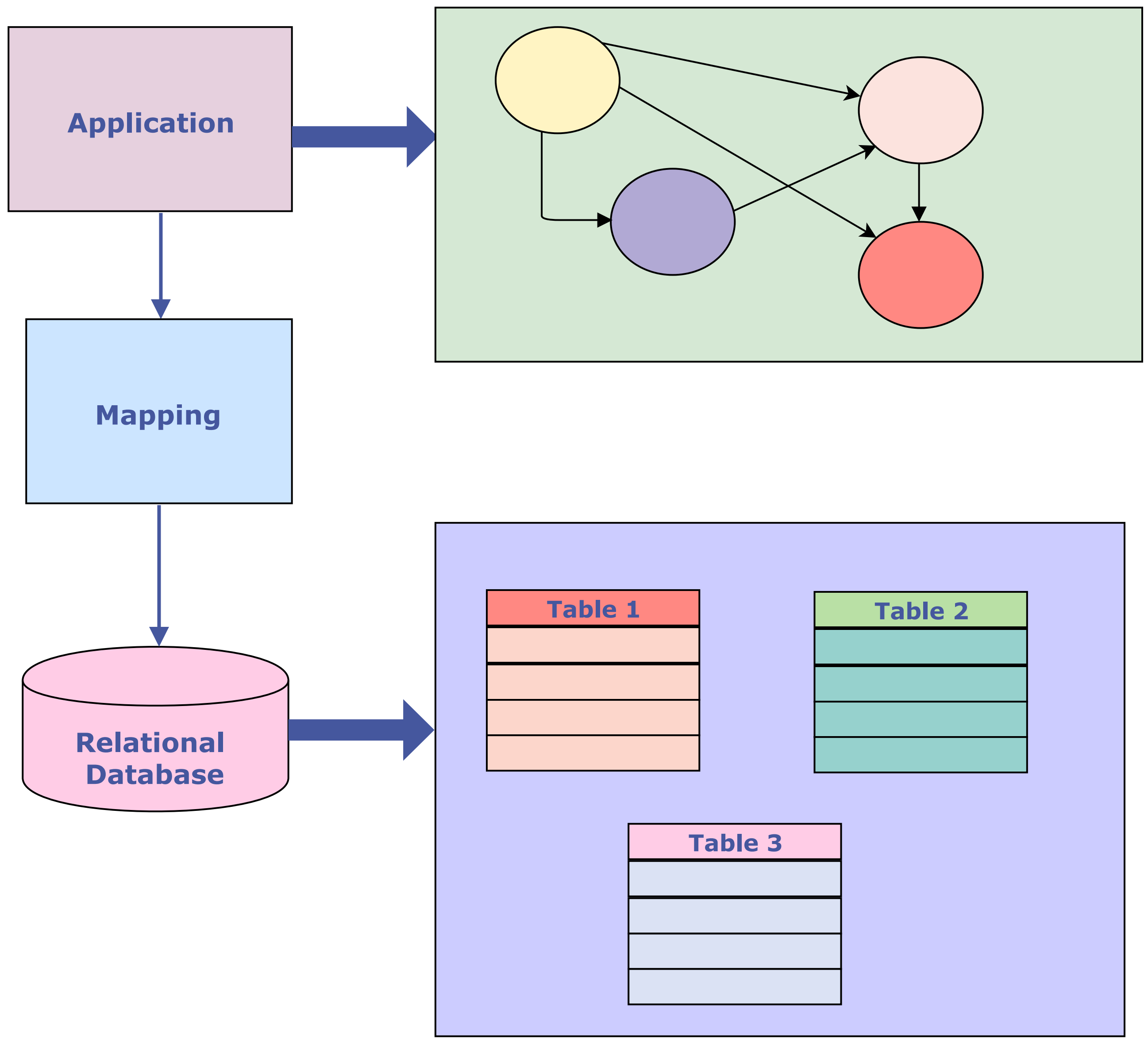
This doesn’t necessarily mean that the data lacks structure, but that structure (especially in tabular form) is hard to grasp and define. An example of an application using unstructured data would be *customer analytics*, where companies use collected data to analyze and improve customer relationship management.

We do not have as many standalone applications today, for which *relational databases* were made. However, these applications have changed with the introduction of *social media*, *internet of things*, *microservices*, etc. This advanced world, with distributed systems and a lot of data, has given space for the *NoSQL database* to blossom.

The biggest problem that relational databases have today is the impedance mismatch.

> **Impedance mismatch** is a term used, in general, to describe the difference between the *relational model* and the *in-memory* data structures.

For example, objects in application memory are often saved in the form of a graph, which is different from the tabular form that data is saved in, in the relational databases. This is why we always need to modify, cut and re-arrange our data before we save it to the relational database.



Why is this such a big problem?

Since these modifications are coupled with modifications in the application itself, it takes a long time to extend and maintain these databases and, today, applications and systems have a demand for 24/7 availability, which means that there is no longer time to turn these systems off for the weekend and update the database. Application users also demand rapid response times, so, the front-end can't afford to slow down while the datastore is updating itself.

Another issue with relational databases is that they are not that *cluster-oriented*. When you try to have replicas, the *writes* will have to be replicated to all the nodes; since all of them need to be consistent before a transaction is allowed to commit, this makes the *writes* an expensive operation. In relational databases, replicas affect performance significantly. Since you don't want to sacrifice consistency, you must wait for all the replicas to be consistent before you can read the data back.

you can read the data back.

The ability to manage data *dynamically*, to go to production quickly, and to develop software using agile methodologies all require more flexibility than relational databases can provide.

Now that you know all about the problems related to *relational databases*, we will learn about NoSQL databases and how they are used to overcome the problems in relational databases.