## **O que vamos aprender?**

Seja muito bem-vindo ao incrível universo do NoSQL ! Neste módulo, você vai ver conceitos do NoSQL, sua história, suas classes de banco de dados e, por fim, entraremos mais a fundo em um banco de uma dessas classes utilizando o MongoDB !

Dividimos a introdução em duas partes:

1. Inicialmente você verá por que o NoSQL surgiu.
2. Depois você verá quais são os tipos (classes) de bancos de dados NoSQL.

## **Introdução**

### **Um pouquinho de história**

Como tudo no universo da tecnologia, procuramos sempre criar algo para resolver um problema. No contexto dos bancos de dados, quando eles nasceram, há mais de 40 anos, havia um problema: os dados produzidos pelos computadores da época deveriam ficar armazenados e não poderiam ser perdidos quando as máquinas fossem desligadas, para que pudessem ser utilizados posteriormente. Porém, naquela época, a tecnologia de armazenamento estava engatinhando e discos eram muito caros. Por isso surgiu a necessidade de estruturar e normalizar os dados que seriam gravados nesses discos. Assim, os dados armazenados utilizariam menos espaço e, consequentemente, teriam um aproveitamento melhor de recursos, gerando economia ou a possibilidade de armazenar mais dados em um mesmo espaço. Nesse contexto, os bancos de dados relacionais foram criados, com todos os conceitos que você já viu no módulo anterior, como formas normais e estrutura de dados bem definidas.

O tempo passou, nossa tecnologia evoluiu e, com essa evolução, mais um problema surgiu: produzimos dados em uma quantidade muito grande, e muitas vezes de forma desestruturada. Esses dados são chamados de desestruturados por terem origem em diversas fontes, como sensores IOT ( *Internet of Things* , como uma geladeira conectada à internet, relógios inteligentes e carros autônomos), imagens e documentos não catalogados, etc. Estruturar, isto é, organizar essas fontes é possível, porém requer muito tempo e, com isso, as aplicações podem perder o *timing* de entrega das soluções. Esse problema precisava ser resolvido e, assim, nasceram os bancos de dados NoSQL!

O termo NoSQL foi originalmente criado em 1998 por Carlo Strozzi e, posteriormente, reintroduzido por Eric Evans em 2009, quando participou da organização de um evento para discutir bancos de dados open source e distribuídos . Por falar em distribuídos, esse é um conceito amplamente utilizado pelos bancos NoSQL: basicamente, esses bancos operam em computação distribuída, um conceito que aumenta muito sua escalabilidade e performance.

### **O que significa NoSQL?**

Não existe uma definição, digamos, "oficial" para o que realmente esse termo significa, mas uma das versões mais aceitas é: N ot O nly SQL , ou seja, "Não Somente SQL". Essa definição enfatiza que esses bancos podem utilizar linguagens semelhantes ao SQL para realizar consultas e demais operações, e não somente o SQL em si.

### **Particularidades do NoSQL**

Quando você estudou sobre bancos de dados relacionais, viu que esses bancos são baseados no conceito ACID (Atomicity, Consistency, Isolation, Durability) . Os bancos de dados NoSQL, em sua grande maioria, baseiam-se em um outro conceito: o BASE (Base Availability, Soft State and Eventually Consistent) .

Antes de detalhar cada ponto do conceito BASE , você precisa entender um termo importante que verá ao longo deste módulo: cluster . Um cluster, no contexto de banco de dados, se refere à capacidade de um conjunto de servidores ou instâncias se conectarem a um banco de dados. Uma instância é uma coleção de memória e processos que interagem com o banco de dados, que é o conjunto de arquivos físicos que realmente armazenam os dados.

O cluster oferece duas vantagens principais, especialmente em ambientes de banco de dados de alto volume:

* Tolerância a falhas ( *Fault Tolerance* ): como há mais de um servidor ou instância para os usuários se conectarem, o cluster oferece uma alternativa no caso de falha em um servidor. Quando se lida com dezenas de milhares de máquinas em um único centro de processamento de dados (CPD), também conhecido como data center, tais falhas são um problema presente;
* Balanceamento de carga ( *Load Balancing* ): o cluster geralmente é configurado para permitir que os usuários sejam automaticamente alocados ao servidor com o mínimo de uso para que, assim, se otimize o uso da estrutura disponível para o banco.

### **Detalhando o conceito BASE**

* Base Availability - BA
  + O banco de dados aparenta funcionar o tempo todo. Como existe o cluster, se um servidor falhar, o banco continuará funcionando por conta de outro servidor que suprirá essa falha;
* Soft State - S
  + Não precisa estar consistente o tempo todo. Ou seja: com um banco distribuído em várias máquinas e todas sendo usadas com igual frequência para escrita e consulta, é possível que, em dado momento, uma máquina receba uma escrita e não tenha tido tempo de "repassar" essa escrita para as demais máquinas do banco. Assim, se um usuário consultar a máquina que já foi atualizada e outro o fizer numa máquina menos atualizada, os resultados, que deveriam ser iguais, serão diferentes. Imagine a sua *timeline* do Facebook : nela são exibidos os posts de seus amigos, porém nem todos os posts são exibidos exatamente ao mesmo tempo. Nesse caso, o que acontece é que a informação foi enviada ao banco de dados, mas nem todos os servidores do cluster têm essa mesma informação ao mesmo tempo. Isso permite que o banco de dados possa gerenciar mais informações de escrita sem ter que se preocupar em replicá-las em uma mesma operação;
* Eventually Consistent - E
  + O sistema se torna consistente em algum momento. Como não temos a informação replicada "instantaneamente", esse ponto se encarrega de deixar o banco consistente "ao seu tempo". Isso porque, dependendo das configurações do cluster, essa replicação pode acontecer mais rapidamente ou não. Mas em algum momento as informações estarão consistentes e presentes em todos os servidores do cluster.

Uma outra característica marcante é a ausência de *schema* ou *schema* flexível. Isso quer dizer que não há necessidade de definição prévia do schema dos dados. Se por um lado isso torna muito mais dinâmico o processo de inclusão de novos atributos, por outro pode impactar a integridade desses dados. Não se preocupe: a seu tempo, todos esses conceitos ficarão bem mais claros.

## **NoSQL e suas classes**

Os bancos de dados NoSQL estão divididos em quatro principais tipos (que, no contexto de banco de dados, são chamados de classes ):

* Chave / Valor ( *Key / Value* )
* Família de Colunas ( *Column Family* )
* Documentos ( *Document* )
* Grafos ( *Graph* )

Cada uma dessas classes tem aplicações diferentes, e devemos sempre observar as características da classe para tirar o melhor proveito dela.

### **Chave / Valor - *Key / Value***

Essa primeira classe é considerada a mais simples. Os dados são armazenados num esquema de registros compostos por uma chave (identificador do registro) e um valor (todo o conteúdo pertencente àquela chave). Você consegue recuperar um registro do seu banco de dados através da chave. Consultas por algum conteúdo através do valor não são permitidas.

A maioria dos bancos de dados Chave / Valor utilizam-se do recurso de armazenamento *in-memory* (memória RAM) e, com isso, o acesso aos dados é extremamente rápido. Alguns cuidados, porém, devem ser tomados na questão da persistência desses dados, uma vez que eles estarão em uma área de memória volátil, não fazendo um transbordo para o disco (por default). Essa volatilidade se dá porque a memória RAM é totalmente apagada quando os computadores são reiniciados ou desligados, ou seja, é uma área temporária.

Sistemas que requerem algum tipo de cache utilizam bastante essa classe de bancos de dados.

Um exemplo de banco de dados da classe Chave / Valor é o Redis .

### **Família de Colunas - *Column Family***

Subindo um pouco mais a complexidade dos dados armazenados, essa segunda classe armazena os dados como um conjunto de três "chaves": linha, coluna e *timestamp* . As linhas e colunas concentram os dados, e as diferentes versões desses dados são identificadas pelo *timestamp* .

Destaque para o conceito de *masterless* , ou seja, não existe um único servidor no cluster que concentra a escrita; essas operações são atendidas pelo servidor que estiver mais "próximo" de onde a operação vier.

Sistemas em que dados analíticos em grande escala são o ponto-chave, o uso dessa classe é altamente recomendável.

Um exemplo de banco de dados da classe Família de Colunas é o Cassandra .

### **Documentos - *Document***

A classe mais flexível e com ampla aderência em vários casos de uso. Os dados são armazenados em estilo JSON , podendo ter vários níveis e subníveis, o que confere aos dados armazenados possibilidade de ter maior complexidade. A estrutura de um documento é muito parecida com o que armazenamos na classe Chave / Valor. Porém, com Documentos, não temos apenas uma chave e sim um conjunto de chaves e valores.

Por mais que *schemaless* (sem o uso de schema) seja um ponto presente na maioria dos bancos de dados NoSQL, na classe de Documentos temos esse conceito mais presente justamente pelo uso do JSON como padronização. Isso porque a inclusão, remoção ou alteração de tipos de dados são muito mais simples e fluídos utilizando JSON .

Sistemas que requerem uma gama de informações com diversos layouts e esquemas se encaixam muito bem nessa classe.

Um exemplo de banco de dados da classe Documentos é o CouchDB (além do próprio MongoDB que vamos ver neste bloco).

### **Grafos - *Graph***

A classe que consegue armazenar dados muito complexos. Os dados são compostos por nós (vértices do grafo), relacionamentos (arestas do grafo) e as propriedades ou atributos dos nós ou relacionamentos. Note que o relacionamento é o ponto central dessa classe. Nesses bancos de dados, o relacionamento é físico, sendo persistido como qualquer outro dado dentro do banco. Dessa forma, as consultas que requerem esses relacionamentos são extremamente performáticas.

Os Grafos estão muito mais presentes em seu dia a dia do que você possa imaginar. Empresas e aplicativos de transporte ou GPS, por exemplo, utilizam os algoritmos e bancos de dados de Grafos para diversas de suas operações, como encontrar o motorista mais perto de você, calcular o menor caminho de um ponto a outro e até mesmo fazer recomendações de produtos em sites de comércio eletrônico. Sistemas de recomendação e antifraude também têm encaixe perfeito para essa classe.

Um exemplo de banco de dados da classe Grafos é o Neo4j .

## **Recursos adicionais**

* [Paradigma-da-computacao-distribuida](https://imasters.com.br/arquitetura-da-informacao/paradigma-da-computacao-distribuida) .
* [NoSQL // Dicionário do Programador](https://youtu.be/1B64oqE8PLs)
* [Artigo que fala sobre as principais características do NoSQL](https://www.guru99.com/nosql-tutorial.html)
* [Pesquisa feita em 2019 sobre o uso de bancos de dados SQL vs NoSQL](https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/)
* [Artigo que explica quando usar e quando não usar NoSQL](https://medium.com/leroy-merlin-brasil-tech/devo-usar-nosql-e-mongodb-951693aa0d34)
* [Conceitos de Bancos de Dados – O que significa ACID](http://www.bosontreinamentos.com.br/bancos-de-dados/conceitos-de-bancos-de-dados-o-que-significa-acid/)
* [Site DB Engine com os bancos de dados mais usados](https://db-engines.com/en/ranking/)

## **Próximo**