



**Persistência e Pesquisa de Dados**

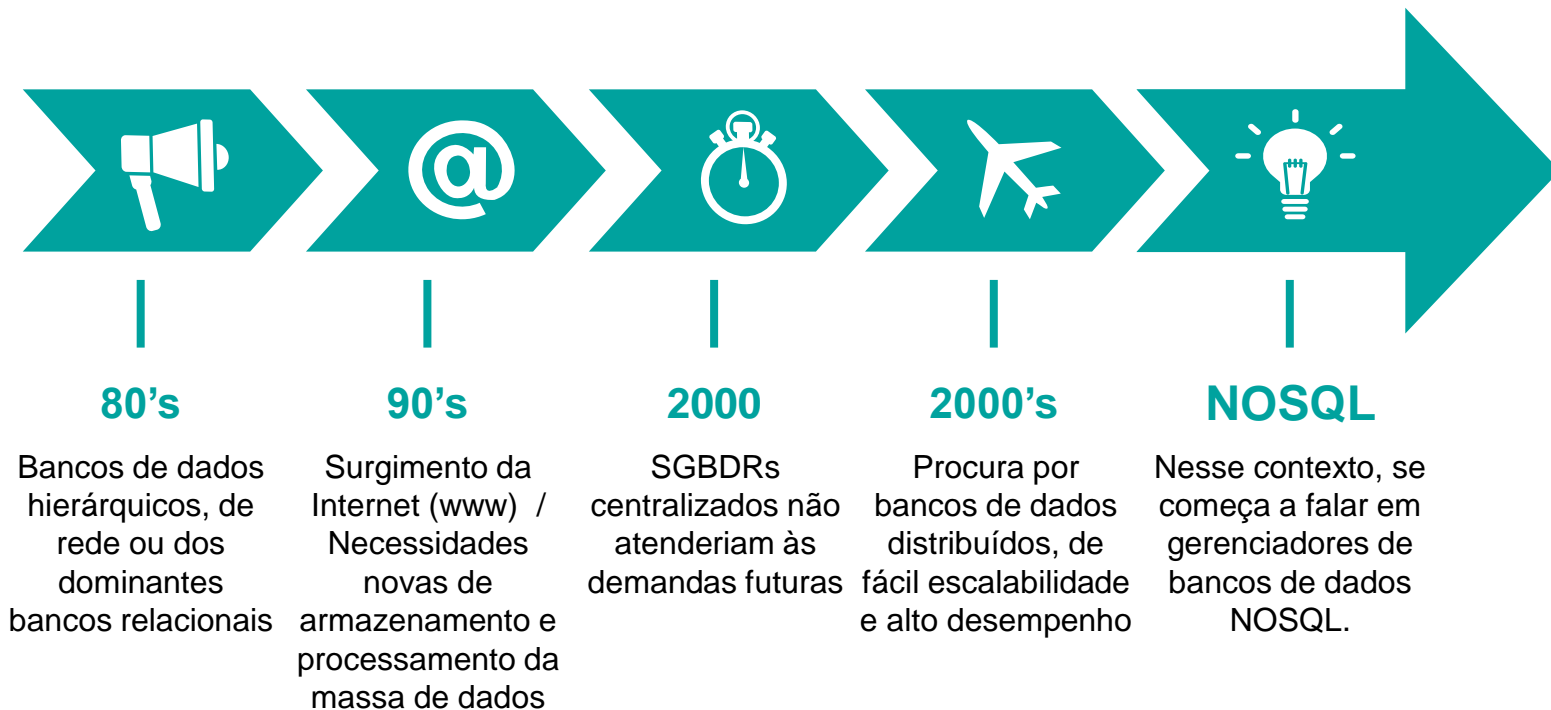
**Capítulo 3. Bancos de Dados NoSQL**

**Prof. Gustavo Aguilar**



## **Aula 3.1. O Movimento NoSQL**

- ☐ Pré-Movimento NoSQL.
- ☐ Surgimento do termo NoSQL.
- ☐ O Movimento NoSQL.



- O termo **NoSQL** surgiu em 1998;
- Quando **Carlo Strozzi** batizou seu novo produto de NoSQL:
  - Um banco de dados relacional;
  - Não fornecia nenhum tipo de linguagem SQL para consulta;
  - Era pronunciado como **noseequeel** (No + SQL);
  - De código aberto e que executava e interagia com o UNIX;
  - Gravação e recuperação dos dados:
    - Operadores" que executavam uma única função.
    - Fluxo de dados era fornecido pelo próprio mecanismo de redirecionamento de entrada / saída do UNIX;
    - Cada operador processava os dados e os transmitia ao próximo operador por meio da função pipe.

- Os dados do *NoSQL* eram armazenados em arquivos *ASCII UNIX*:
  - Manipulados por utilitários *UNIX* comuns: *ls*, *mv*, *cp* e editores como o *'vi'*.
- Manipulação dos dados sem a linguagem SQL → “*non SQL*”
  - No sentido de realmente não ser um banco de dados SQL
- Formato dos arquivos → baseado em relação (tabela, linhas e colunas) → **família de SGBD relacional.**



- Strozzi afirmava categoricamente que seu produto “*nada tinha a ver com o recém-nascido **Movimento NOSQL***”:
  - “...é um banco de dados relacional para todos os efeitos e apenas intencionalmente não usa SQL como uma linguagem de consulta...”
  - “...o recém-chegado é principalmente um conceito ...., que se afasta do modelo relacional...”
  - “....deveria ter sido chamado de ‘**NoREL**’, ...., já que **não ser baseado em SQL é apenas uma consequência óbvia de não ser relacional, e não o contrário**”.

- No início de 2009 ➔ termo NoSQL voltou à tona:
  - Johan Oskarsson, desenvolvedor na *Last.fm*.
  - Organizou um evento para discutir "bancos de dados distribuídos, não relacionais e de código aberto".
  - Termo tentava rotular o surgimento de um número crescente de repositórios de dados distribuídos, não relacionais:
    - Clones de código aberto do *Google Bigtable/MapReduce* e do *DynamoDB* da *Amazon*.
  - Desde então ➔ termo *NoSQL* tem sido atribuído aos **SGBDs não relacionais** (e não somente mais aos “não SQL”):
    - Mecanismo de armazenamento de dados que não é baseado no formato tabular (linhas x colunas).



- Termo *NoSQL* passou a ser escrito como **NOSQL**;
- SGBDs “Não Apenas SQL” → **Not Only SQL**;



- Enfatizar que suportavam outras linguagens de consulta, além da SQL.

- Em 29 de julho de 2011:
  - Fundador da Couchbase e inventor do CouchDB, Damien Katz
- +
- Richard Hipp, inventor do SQLite
- ‘UNQL QUERY LANGUAGE UNVEILED BY COUCHBASE AND SQLITE’
- Linguagem de consulta padrão NOSQL ➔ **UnQL**
  - Unstructured Query Language;
  - Pronúncia “Uncle”;
  - Tentar se popularizar da mesma forma como o SQL do movimento relacional se popularizou;
  - Site do projeto: <http://unql.sqlite.org/index.html/wiki?name=UnQL>.

- ☑ Pré-Movimento NoSQL disparado pelas novas necessidades e inflexibilidades dos SGBDRs.
- ☑ Surgimento do Termo NoSQL inicialmente como um SGBDR sem suporte à linguagem SQL.
- ☑ O Movimento NoSQL concretizando e popularizando o termo NoSQL para designar os SGBDs não relacionais.

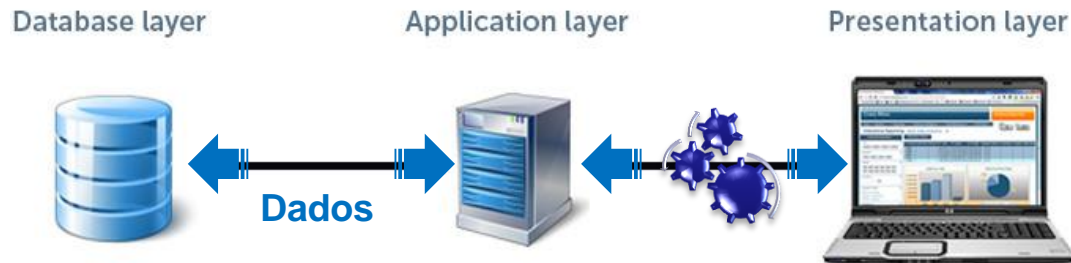
- ❑ Características dos Bancos de Dados NoSQL.



## **Aula 3.2. Características dos Bancos de Dados NoSQL**

- ☐ Características dos Bancos de Dados NoSQL.
- ☐ Teorema de CAP.
- ☐ ACID x BASE.

- Manipulação de **grandes massas** de dados;
- Camada de banco simplificada:
  - Função primária → **armazenar dados**;
  - Complexidade, lógica e consistência de dados → camada de aplicação.
  - **Construção** mais **rápida** do schema físico;

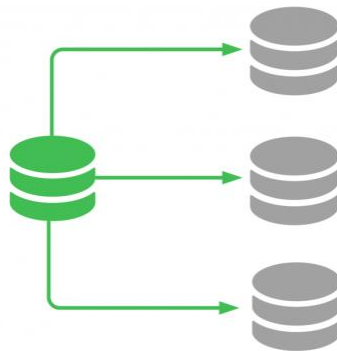


- **Custo:** normalmente bem mais reduzido que o custo dos SGBDRs;
- **Ausência de Esquema (Schema Less) ou Esquema Dinâmico:**
  - Definição dinâmica, sob demanda da estrutura de armazenamento de dados;
  - Estruturas de dados não precisam ser pré criadas → definidas em tempo de inserção dos dados. ~~CREATE TABLE + INSERT.~~
- **API de Acesso Simples:** foco em como recuperar os dados de forma eficiente e não em como armazená-los em estruturas complexas;



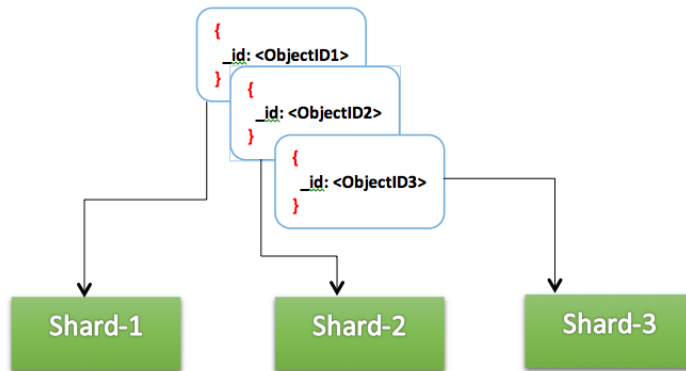
- **Suporte Nativo a Replicação:**

- Alta disponibilidade dos dados.



- **Suporte Nativo a Sharding:**

- Particionamento horizontal
- Processamento paralelo
  - Distribuição da carga
  - Melhor aproveitamento do hardware



- Escalabilidade Horizontal



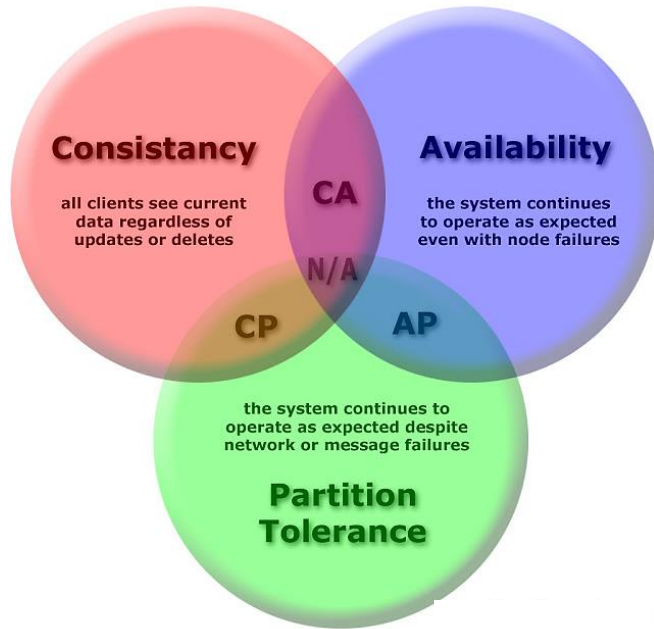
- Escalabilidade Vertical



- Controle de Concorrência Multiversão (Multiversion Concurrency Control - MVCC):
  - Controle mais flexível para as transações;
  - Operações de leitura ocorrendo em paralelo com as operações de escrita.

- Criado por Eric Brewer em 2000;
- **Três principais requisitos sistêmicos** em um ambiente distribuído:
  - Consistency, Availability, Partition tolerance;
  - **Consistência:** todos veem os mesmos dados;
  - **Disponibilidade:** o sistema está disponível quando solicitado, independentemente de ter ocorrido uma falha em alguns dos servidores;
  - **Tolerância à Partição:** perda de conexão entre os nós ou perda de dados em um determinado nó não causam impacto.
- Sistema distribuído pode garantir **apenas dois desses requisitos**.

# Teorema de CAP



- Consequências significantes para os SGBDs não relacionais distribuídos:
  - Com foco na disponibilidade e tolerância à partição;
  - Consistência em segundo plano;
  - Propriedades ACID não aplicáveis em sua totalidade → confiabilidade dos dados ?

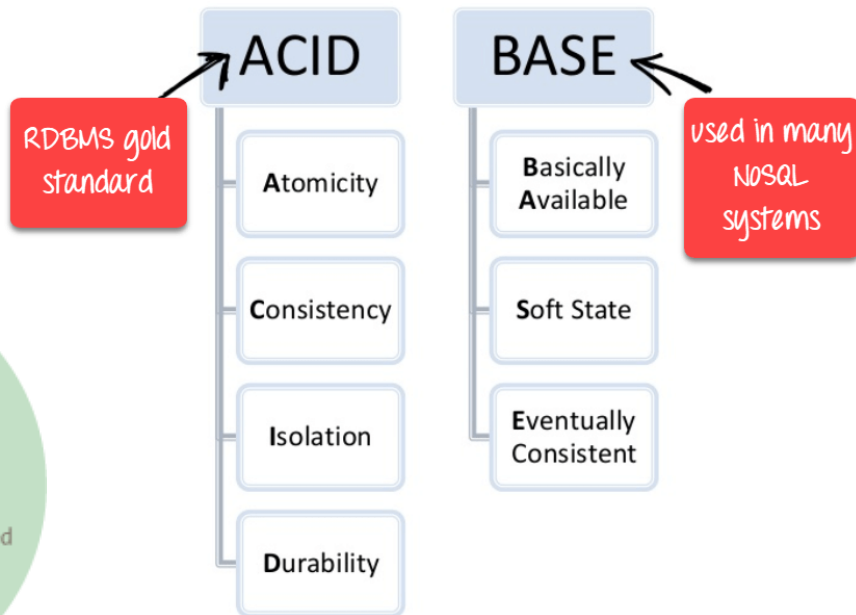
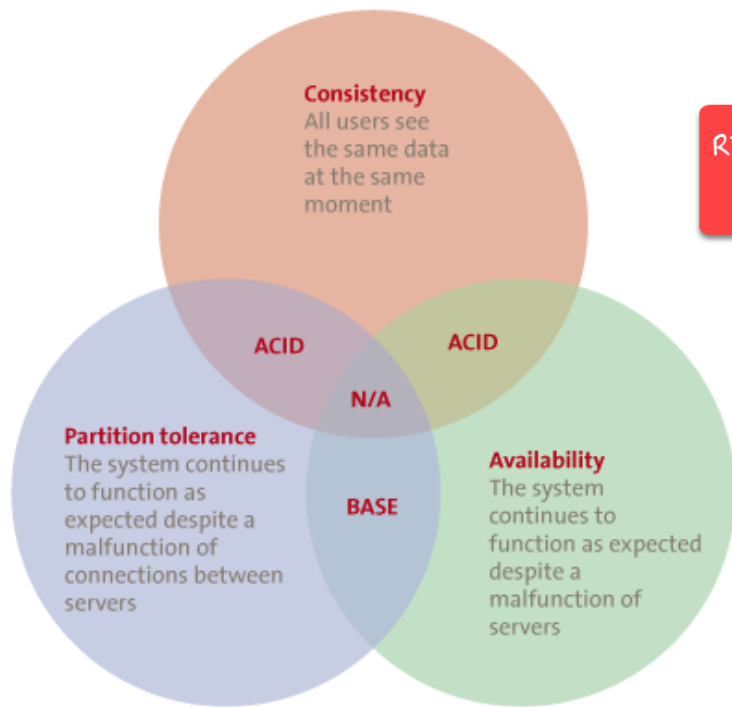


- Soft state: o estado do sistema pode mudar ao longo do tempo, mesmo durante momentos sem entrada de dados → necessário para a consistência eventual.

- Proposto um modelo alternativo de consistência → **BASE**

- Basically Available: haverá uma resposta a qualquer solicitação, mesmo que seja de falha;
- Eventual consistency: dados são propagados para todos os nós, assincronamente.
  - Dados continuam sendo gravados sem verificação da consistência de todas as transações.

# ACID x BASE



*“BASE: Uma Alternativa Ácida”* - Dan Pritchett.

- ☑ Características dos Bancos de Dados NoSQL com grande foco em manipulação de grandes massas de dados, camada de banco simplificada e flexível para armazenamento de dados.
- ☑ Teorema de CAP trazendo à tona a possibilidade de convivência de somente 2 dos 3 requisitos sistêmicos em um ambiente distribuído.
- ☑ ACID x BASE, onde os sistemas relacionais atendem normalmente às propriedades ACID e os não relacionais às propriedades BASE, criadas para atender a demanda emergente da era da IOT.

- ❑ Famílias de Bancos de Dados NoSQL.

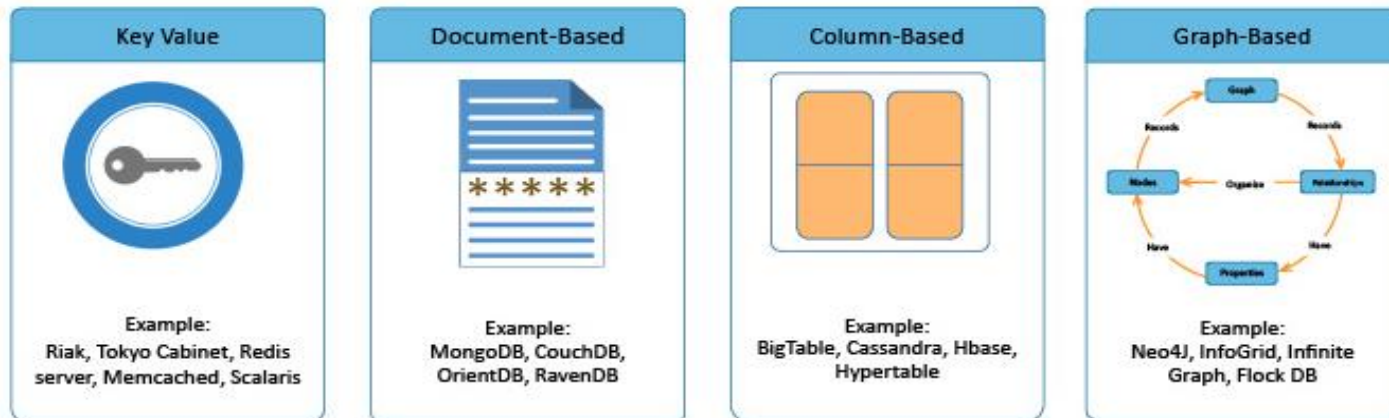




## **Aula 3.3. Famílias de Bancos de Dados NoSQL**

- ☐ Famílias de Bancos de Dados NoSQL.
- ☐ Banco de Dados Chave-Valor (Key-Value).
- ☐ Banco de Dados Baseado em Documento (Document Database).
- ☐ Banco de Dados Colunar (Column-Based).
- ☐ Banco de Dados Baseado em Grafo.
- ☐ Banco de Dados Multimodelo.

- Classificam os SGBDs NOSQL de acordo com o modelo de dados utilizado nas estruturas de armazenamento de dados:
  - Chave-valor, Documento, Colunas, Grafo



- <http://nosql-database.org>

- Modelo de armazenamento de dados mais simples;
- Usa o conceito de array associativo → mapa ou dicionário:

- **Identificador alfanumérico (chave)**

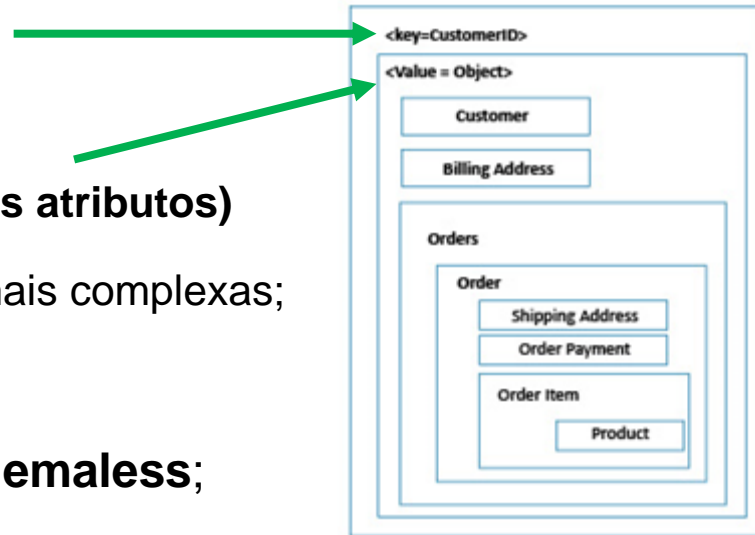
+

- **Valores vinculados à essa chave (os atributos)**

- **Chaves** → strings simples ou listas mais complexas;

- **Valores** → armazenado como *BLOB*.

- Não tem um schema definido → **schemaless**;

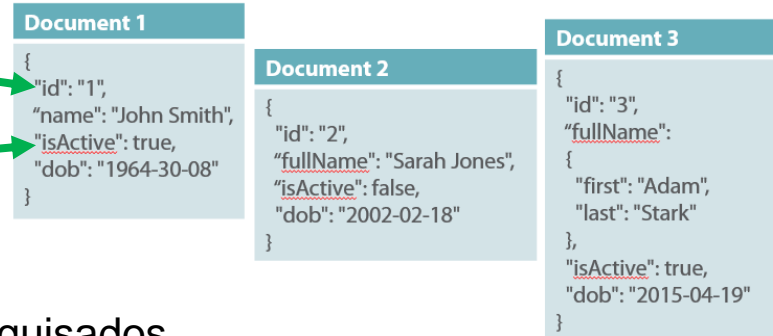


# Banco de Dados Chave-Valor (Key-Value)

- Extremamente performáticos e escaláveis;
- Muito usados para:
  - Armazenamento de informações de sessão, carrinho de compras;
  - Perfis de usuários e preferências, etc.
- Não são recomendados para:
  - Schemas físicos com relacionamento entre os dados;
  - Transações com múltiplas operações;
  - Consultas por dados e atributos:
    - Buscas de dados só podem ser realizadas com base nas chaves e limitadas a correspondências exatas.



- Document Database;
- Dados armazenados em documentos: JSON, BSON, XML, etc.;
- Documentos armazenados em *collection* / *hierarquia de diretórios* / *tag* / *metadados não visíveis*;
- Identificador único do documento;
- Conjunto de campos e valores
  - *Field-values*;
  - Podem ser facilmente indexados e pesquisados.
- Quantidade de campos pode diferir de documento para documento;



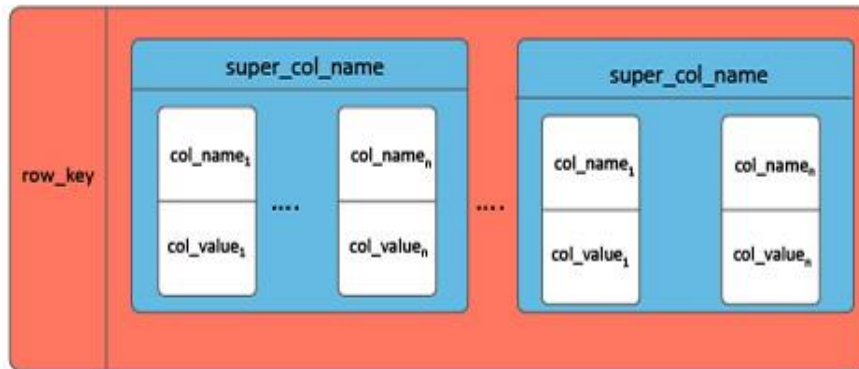
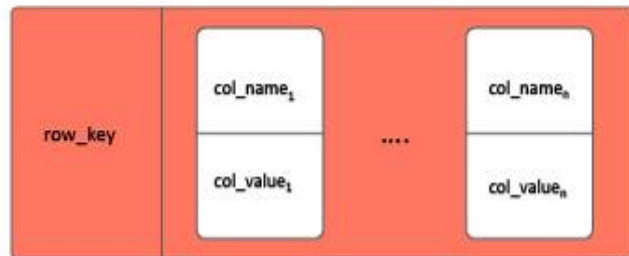
- Modelagem → dados frequentemente consultados são armazenados em conjunto, no mesmo documento;
- Muito indicados para:
  - Sistemas que precisam de flexibilidade e dinamismo no schema físico para armazenamento de dados diferentes;
  - Grande quantidade de operações de leitura e escrita;
  - Dados no formato JSON.
- Não são recomendados para cenários com transações complexas, queries muito longas ou com muitos relacionamentos.



- Armazenam dados em famílias de colunas como linhas;
  - Famílias de Colunas → dados relacionados que são acessados juntos.
- Cada linha contém várias colunas associadas à uma chave;
- “Super coluna” → conjunto de colunas;
- Fisicamente, todos os valores de uma coluna são gravados juntos:
  - Mesmo registro;
  - De forma ordenada:
    - Coluna 1 + valores da coluna 1
    - Coluna 2 + valores da coluna 2
    - ...



# Banco de Dados Colunar (Column-Based)

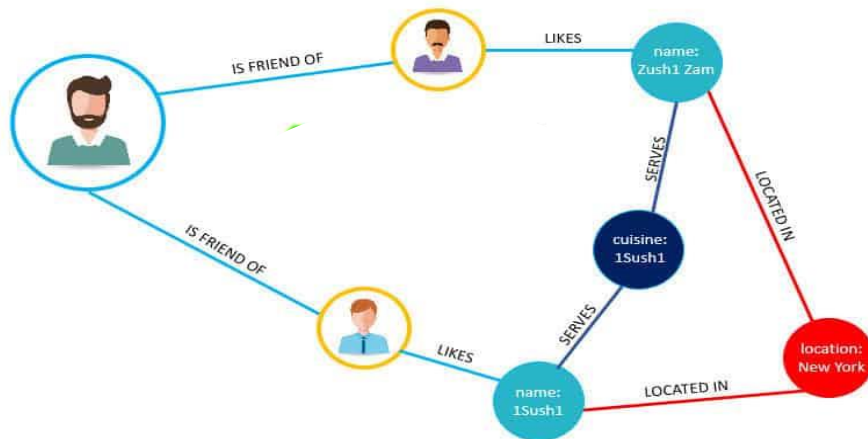


- As linhas não precisam ter as mesmas colunas;
- Uma coluna pode ser adicionada a qualquer linha, a qualquer momento, sem ser adicionada às outras linhas.

- Indicado para:
  - Leitura e gravação de grandes volumes de dados com eficiência
    - Nativamente autoindexado;
    - Usa menos espaço em disco do que um banco de dados relacional contendo os mesmos dados.
  - Operações MIN, MAX, SUM, COUNT, e AVG;
  - Ambientes que requerem escalabilidade rápida;
  - Recomendado para sistemas distribuídos.



- Armazena os dados e suas relações com outros dados na forma de **nós e arestas**;
- Cada relação pode ter um conjunto de propriedades;
- Arestas possuem uma direção;



- Indicado para quando se está mais **preocupado com as relações entre os dados** do que com os dados em si;
- Muito recomendado para:
  - Aplicativos de redes sociais;
  - Aplicações de recomendações;
  - Sistemas de navegação e mapas;
  - Gerenciamento de processos e de redes;
  - Sistemas de transportes.



- Pertencem à **mais de uma família**:
  - Implementam **mais de um modelo de armazenamento**.



Documento  
Colunar  
Grafo



Documento  
Chave-valor  
Grafo



Documento  
Chave-valor  
Grafo

- ☑ Existem quatro famílias de bancos de dados NoSQL, cada uma com um propósito e aplicabilidade bem definidos.

- ☐ Vantagens e desvantagens dos Bancos de Dados NoSQL.

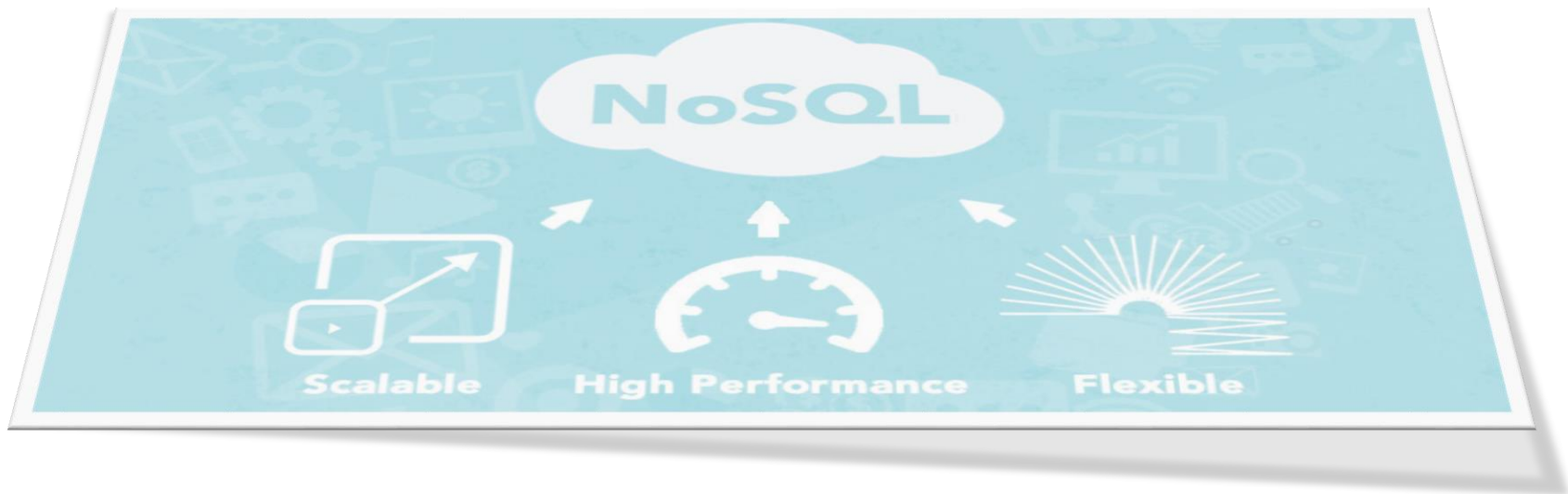


## **Aula 3.4. Vantagens e desvantagens dos Bancos de Dados NoSQL**



- ❑ Prós e contras dos Bancos de Dados NoSQL.

- Apesar de todas as inovações e características atrativas:
  - Prós e contras frente os bancos de dados relacionais (ou outro tipo);
  - Vantagens e desvantagens específicas de cada SGBD NoSQL.



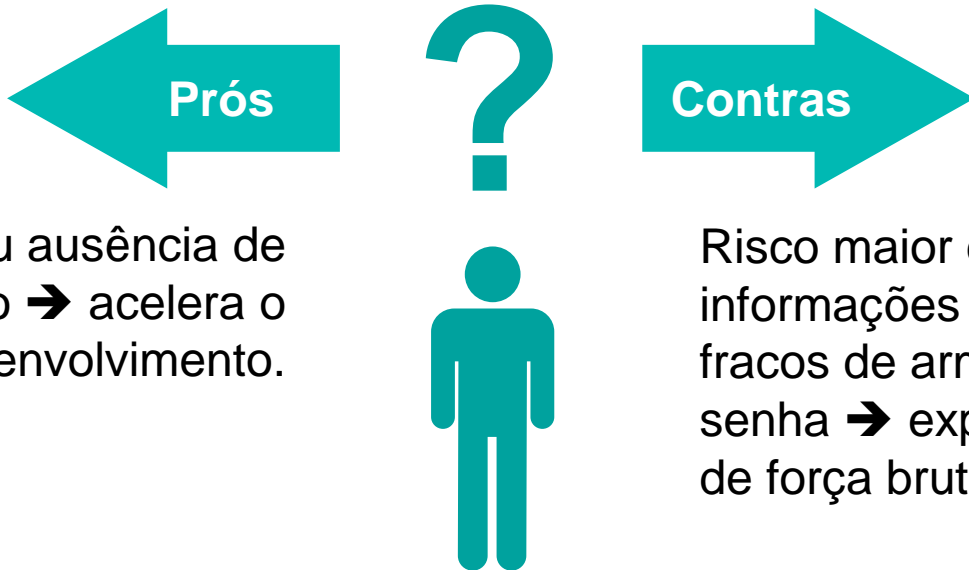


Alta performance e escalabilidade, sendo muito apropriados para grandes volumes de dados.

Modelo de dados simples, com uma linguagem de consulta enxuta → redução da curva de aprendizado.

Não implementam sempre as propriedades ACID → requer código extra → diminui performance.

Grande maioria dos SGBDs NOSQL garantem apenas uma consistência eventual dos dados → leituras “sujas”.

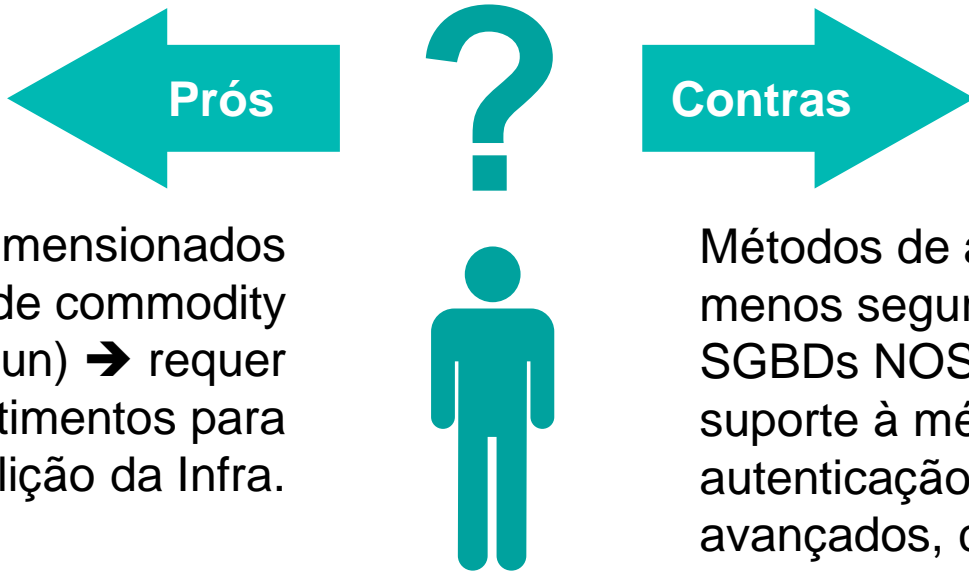


Flexibilidade ou ausência de schema físico → acelera o desenvolvimento.

Capacidade de manipular dados não estruturados, como arquivos do Word, de áudio, vídeo, etc., de forma exímia.

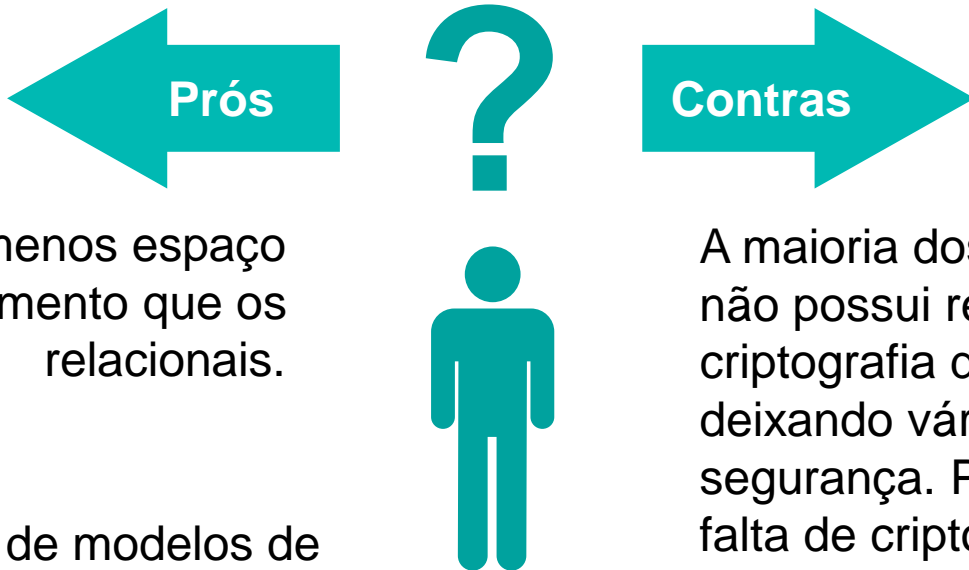
Risco maior de vazamento de informações → mecanismos fracos de armazenamento de senha → expostos a ataques de força bruta de senha.

Possuem mais falhas e limitações → ponto de maturidade não tão alto como o dos relacionais.



Muito bem dimensionados em hardware de commodity (servidor comum) → requer menos investimentos para disponibilização da Infra.

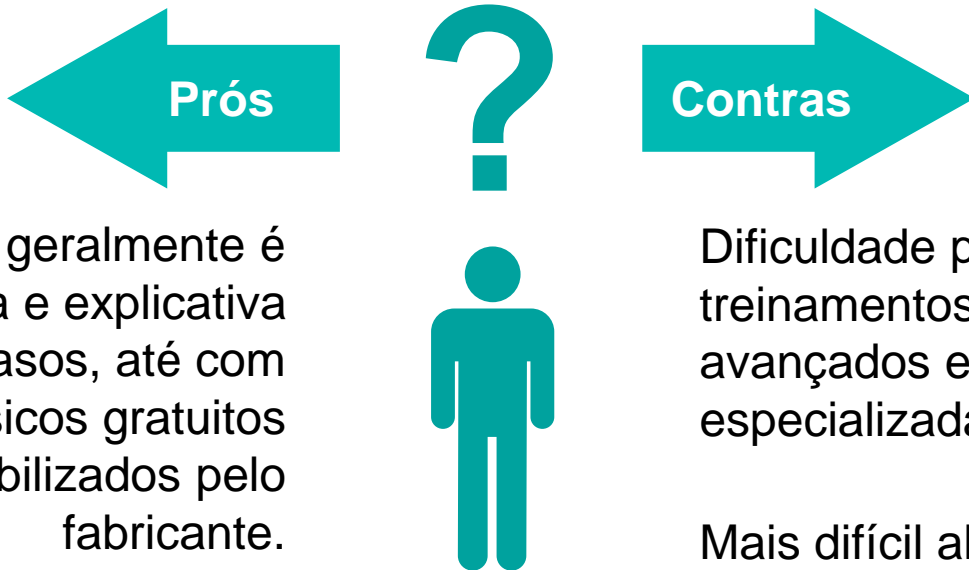
Métodos de autenticação menos seguros, com poucos SGBDs NOSQL fornecendo suporte à métodos de autenticação mais avançados, como por exemplo Kerberos ou integrados com LDAP.



Gastam bem menos espaço de armazenamento que os relacionais.

Várias opções de modelos de armazenamento dos dados, com mais possibilidades de aderência aos requisitos de um sistema.

A maioria dos SGBDs NoSQL não possui recursos de criptografia de dados, deixando várias brechas de segurança. Por exemplo, a falta de criptografia de comunicação entre client-server, deixa-os vulneráveis a **ataques de injeção de SQL ou de negação de serviço.**



Documentação geralmente é bem completa e explicativa e, em alguns casos, até com cursos básicos gratuitos disponibilizados pelo fabricante.

A maioria das soluções NOSQL são de código aberto e com versões gratuitas (*Community*).

Dificuldade para encontrar treinamentos mais avançados e mão-de-obra especializada

Mais difícil alinhar as expectativas de SLAs do negócio com os SLAs fornecidos pelo fabricante ou praticados pela comunidade.

- ☑ É fundamental fazer a escolha da abordagem NoSQL com muito critério, avaliando todos os prós e contras.



# Próxima aula

☐ Modelagem de Dados NoSQL.



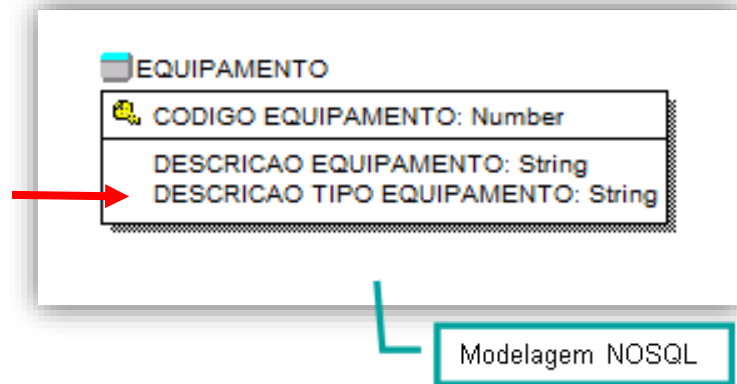
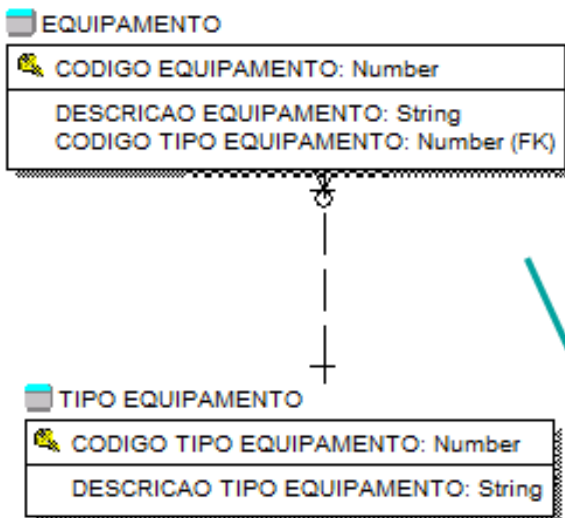
## **Aula 3.5. Modelagem de Dados NoSQL**

- ☐ Introdução à Modelagem de Dados NoSQL.
- ☐ Considerações e técnicas de Modelagem NoSQL.

- **Modelagem conceitual** é independente da abordagem escolhida:
  - Modelo pode ser derivado para um modelo lógico NoSQL ou relacional, de rede ou hierárquico.
- **Modelagem lógica** utilizando a abordagem NoSQL (não-relacional):
  - Definir qual a família (modelo de armazenamento de dados) a ser utilizada;
  - Com exceção da família baseada em grafos, a modelagem lógica para as famílias chave-valor, baseada em documentos e colunar é muito similar.
- **Modelagem física** ➔ SGBD NoSQL específico ➔ esquema físico
  - Se mostra bem diferente entre famílias e até mesmo na mesma família.


- Modelagem NOSQL → abordagem ***top-down*** considerando a visão do desenvolvedor:
  - Primeiro, o que será perguntado (consultado);
  - Depois, como será armazenado;
  - Modelagem relacional → orientada pela estrutura disponível do dado e dos relacionamentos entre eles:
    - Abordagem ***bottom-up***
    - Primeiro, como os dados serão armazenados, de acordo com a teoria relacional
    - Depois, meios de como deve-se responder (consultar os dados).


- **Filosofia da Agregação Atômica:** dados consultados juntos, são armazenados juntos;



Modelagem Relacional

- Estruturas de armazenamento de dados independentes entre si:
  - Desnormalização não é evitada:
    - Imprescindível para a manipulação conjunta de enormes massas de dados;
    - Reduzindo a complexidade do processamento de relacionamentos;
    - Relacionamentos entre estruturas de armazenamento de dados são evitados ao máximo (exceto para o modelo baseado em grafo).
- Redundância (duplicação) de dados:
  - Situação comum decorrente da Filosofia da Agregação Atômica;
  - Incrementa exponencialmente a rapidez das operações de I/O.

FUNCIONARIO	
	CODIGO MATRICULA FUNCIONARIO: Number
NOME FUNCIONARIO: String	
TIPO FUNCIONARIO: String	
PAIS: String	
UF: String	
CIDADE: String	
BAIRRO: String	
LOGRADOURO: String	
NUMERO LOGRADOURO: Number	
CEP: Number	

EQUIPAMENTO	
	CODIGO EQUIPAMENTO: Number
DESCRICAO EQUIPAMENTO: String	
DESCRICAO TIPO EQUIPAMENTO: String	
PAIS: String	
UF: String	
CIDADE: String	
BAIRRO: String	
LOGRADOURO: String	
NUMERO LOGRADOURO: Number	
CEP: Number	

- ☑ A modelagem conceitual continua independente da abordagem NoSQL.
- ☑ Na etapa de modelagem lógica é preciso definir a família (modelo de estrutura de armazenamento de dados).
- ☑ No modelo físico, há diferenças até mesmo entre SGBDs pertencentes à mesma família.
- ☑ Técnicas para modelagem NoSQL, com a filosofia da agregação atômica, sendo aceitas a desnormalização e redundância de dados.



- ❑ Ferramentas Case de Mercado para Modelagem NoSQL.



## **Aula 3.6. Ferramentas Case de Mercado para Modelagem NoSQL**

- ❑ Ferramentas Case de Mercado para Modelagem NoSQL.

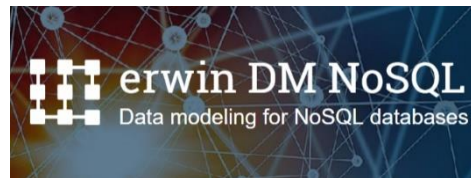
- Para construção do modelo de dados **lógico** e do **físico**;
- Não há uma diversidade de ferramentas case de modelagem NoSQL;
- Grande parte fornece recursos de modelagem para as famílias de bancos NoSQL baseados em documentos e/ou colunares.

## ▪ DBSchema

- Paga;
- Modelo lógico e físico, relacional e não-relacional;
- SGBDs NOSQL:
  - Cassandra, CTreeAce, HBase, Hive, OrientDB, e MongoDB;
- Engenharia reversa e *forward*;
- Gera scripts de criação do schema físico;
- Comparação e sincronização entre modelos e o banco de dados;
- Site: [www.dbschema.com](http://www.dbschema.com).



- **ERWin DM NoSQL**



- Paga;
- Modelo lógico para famílias de bancos baseados em documentos;
- SGBDs NOSQL: CouchDB e MongoDB;
- Engenharia reversa e *forward*;
- Gera scripts de criação do schema físico;
- Comparação e sincronização entre modelos e o banco de dados;
- Conversão de modelo relacional para não-relacional.
- Site: <https://erwin.com/products/erwin-dm-nosql>.

## ▪ Hackolade



- Modelo lógico para **todas** as famílias de bancos NoSQL:
  - Baseados em documento: MongoDB, Couchbase, Elasticsearch, Google Firebase e Firestore;
  - Baseados em grafo: Neo4j;
  - Colunares: HBase e Cassandra;
  - Baseados em chave-valor: DynamoDB;
  - Multimodelos: Cosmos DB.
- Engenharia reversa e *forward* para a maioria dos SGBDs suportados;
- Gera scripts de criação do schema físico;
- Comparação e sincronização entre modelos e o banco de dados;
- Site: <https://hackolade.com/>. (Paga)

- ☑ Baixa diversidade de ferramentas case para modelagem de dados NoSQL, mas com boas opções.



- ❑ Introdução ao MongoDB.



## **Aula 3.7. Introdução ao MongoDB**

- ❑ Introdução ao MongoDB.

# Introdução ao MongoDB

## SGBD e SGBDD

Banco de dados  
centralizado / distribuído  
Multiplataforma  
Escrito em C++



## Lançado em 02/2009

Inicialmente 10gen.  
Atualmente ©MongoDB Inc.



## Gratuito e Open Source

Edição *Community Server*  
\*Edição *Enterprise*  
(subscrição com suporte)



NOSQL

**“Not Only SQL”**  
Não relacional

## Orientado a Documento

Esquema Dinâmico  
*Document Database*  
Formato *JSON* (field-value)  
Armazenado como *BSON*

{JSON}

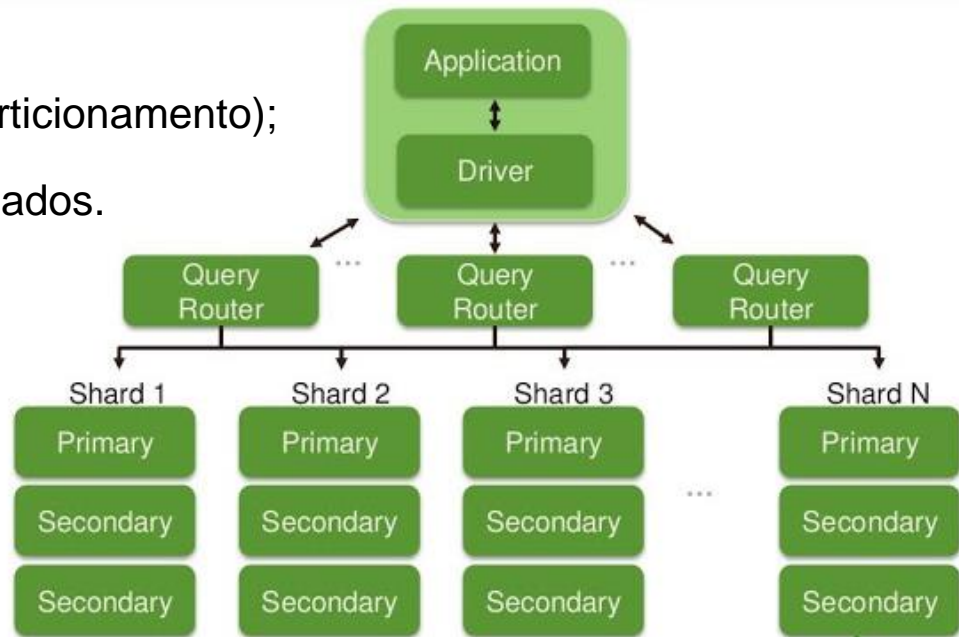


## Universidade MongoDB

Cursos gratuitos e online, em  
inglês, com certificado  
<https://university.mongodb.com>



- SGBD distribuído **nativo**:
  - Alta disponibilidade (cluster);
  - Escalabilidade horizontal (particionamento);
  - Distribuição geográfica dos dados.



- ☑ MongoDB como um SGBD NoSQL baseado em documentos, open source, gratuito e distribuído de natureza.
- ☑ Popularização crescente pela facilidade de uso, qualidade, utilidade, ótima documentação e treinamentos gratuitos na Universidade MongoDB.

## Próxima aula

- ❑ Conceitos básicos e Arquitetura do MongoDB.



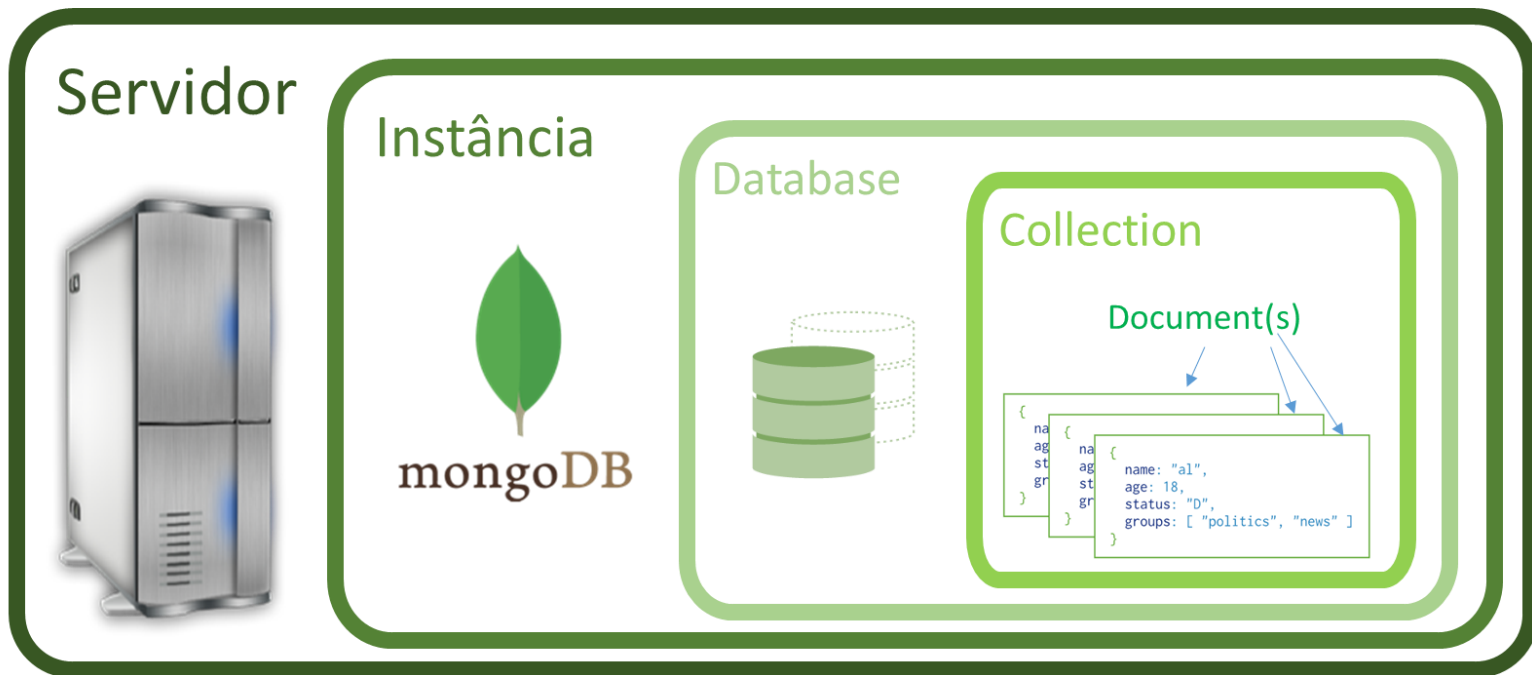
## **Aula 3.8. Conceitos básicos e Arquitetura do MongoDB**



- ☐ Visão geral da Arquitetura do MongoDB.
- ☐ Engines de armazenamento.
- ☐ Autenticação e segurança de acesso.
- ☐ Ferramentas Client.

# Visão geral da Arquitetura do MongoDB

- Organização hierárquica tradicional **servidor** → **instância** → **banco**



- Documento (*Document*) → um registro
  - Conceito similar ao de tupla do mundo relacional;
  - Composto por **pares de campos (*fields*) e valores (*values*)**;

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```



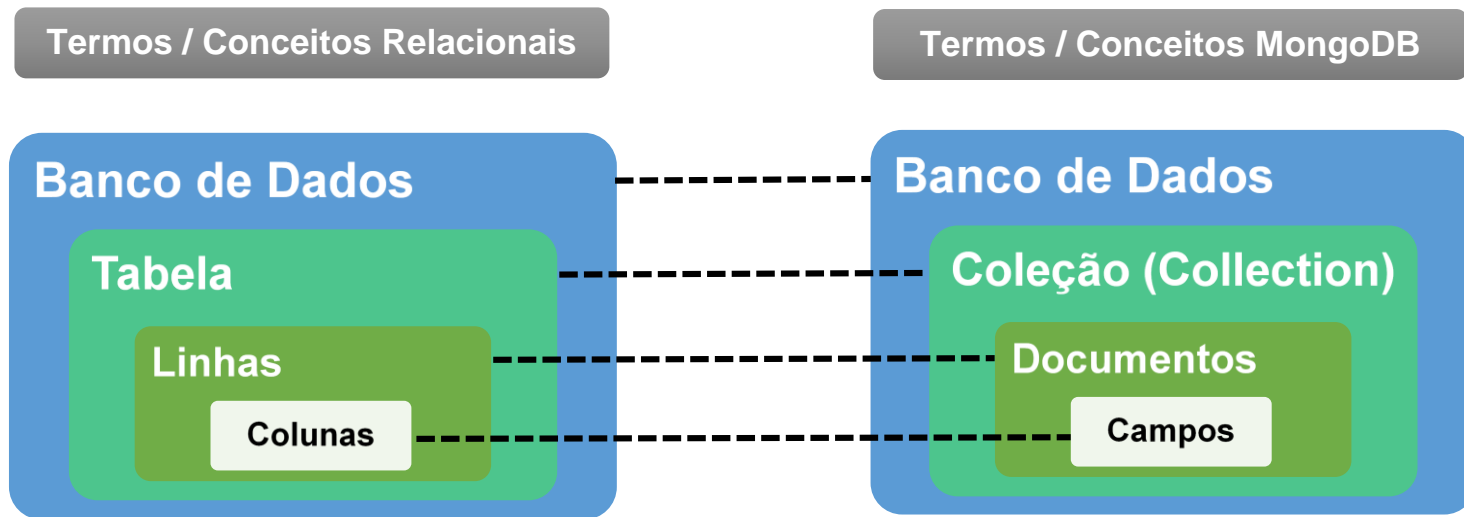
Diagram illustrating the structure of a MongoDB document (JSON-like):

- name: "sue", ← field: value
- age: 26, ← field: value
- status: "A", ← field: value
- groups: [ "news", "sports" ] ← field: value

- Formato *JSON* → armazenado como *BSON*;
- Tamanho máximo → 16 MB (pode ser expandido usando-se GridFS).

# Visão geral da Arquitetura do MongoDB

- Analogia dos termos / conceitos Relacionais x MongoDB



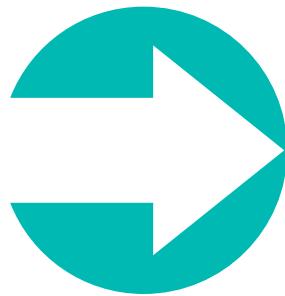
- Mecanismos para gerenciar como os dados são armazenados;
- Três ***engines*** nativas:
  - **WiredTiger:**
    - Engine **padrão** a partir da versão 3.2 do MongoDB;
    - Adequada para a maioria das cargas de trabalho e recomendada para novas implementações;
    - Controle de concorrência a nível de documento, *checkpoint*, compactação, dentre outros recursos;
    - No MongoDB Enterprise, também suporta criptografia.
  - **In-Memory:**
    - Ao invés de armazenar documentos no disco, esse mecanismo os retém na memória para redução da latência de leitura / escrita;
    - Disponível no MongoDB Enterprise.

- **MMAPv1:**
  - Antiga engine padrão → versão 3.0 e anteriores do MongoDB;
  - Foi descontinuada a partir da versão 4.0.
- Além das engines nativas → ***Pluggable Storage Engines:***
  - Externas e de terceiros;
  - Possibilita usar novos recursos e configurações ideais para arquiteturas específicas de hardwares.
- Cada engine apresenta um desempenho diferente para cargas de trabalho específicas → escolher a mais aderente.

## Autenticação Local

### Login Interno do MongoDB

Login e senha são armazenados no MongoDB

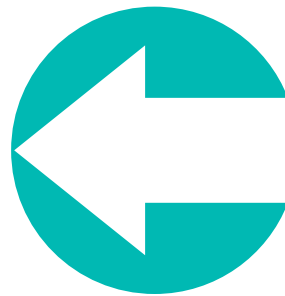


## Autenticação Integrada

### Login Externo ao MongoDB

SCRAM  
x.509 Certificates  
LDAP\*  
Kerberos\*

*\*Só na Enterprise*



- 100% RBAC (***Role-Based Access Control***):

- Privilégios concedidos somente através de Roles → *Action on Resource*

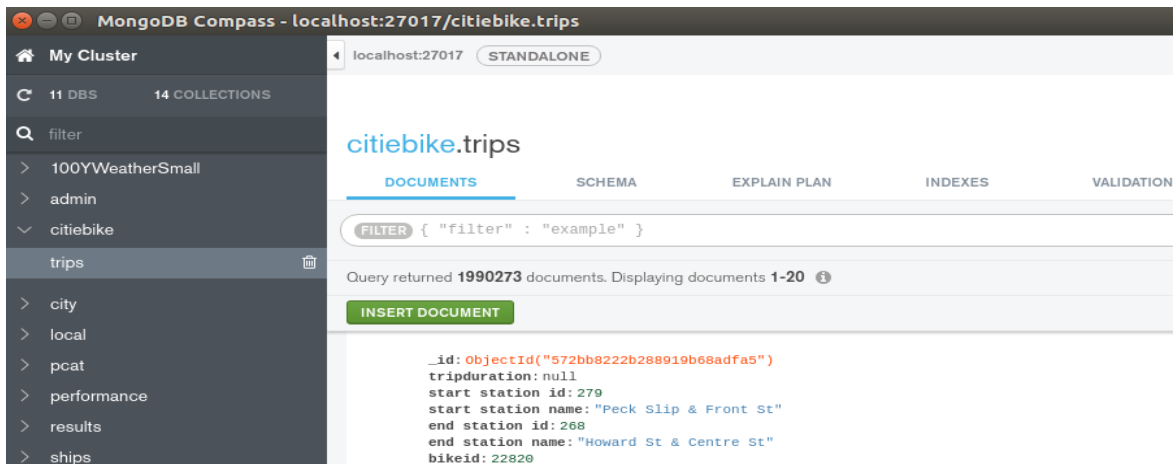
- **Mongo Shell:** ferramenta de linha de comando que possui todos os comandos existentes para configurar e interagir com o MongoDB.

```
gustavaaagl@Gu-Ubuntu: ~  
gustavaaagl@Gu-Ubuntu:~$ mongo  
MongoDB shell version v3.4.6  
connecting to: mongodb://127.0.0.1:27017  
MongoDB server version: 3.4.6  
Server has startup warnings:  
2018-02-20T19:28:36.577-0300 I STORAGE [initandlisten]  
2018-02-20T19:28:36.577-0300 I STORAGE [initandlisten]  
2018-02-20T19:28:36.577-0300 I STORAGE [initandlisten]  
2018-02-20T19:28:37.167-0300 I CONTROL [initandlisten]  
2018-02-20T19:28:37.167-0300 I CONTROL [initandlisten]  
2018-02-20T19:28:37.167-0300 I CONTROL [initandlisten]  
2018-02-20T19:28:37.167-0300 I CONTROL [initandlisten]  
2018-02-20T19:28:37.168-0300 I CONTROL [initandlisten]  
2018-02-20T19:28:37.168-0300 I CONTROL [initandlisten]  
2018-02-20T19:28:37.168-0300 I CONTROL [initandlisten]  
2018-02-20T19:28:37.168-0300 I CONTROL [initandlisten]  
> show dbs  
100YWeatherSmall 0.119GB  
admin             0.000GB  
citiebike         0.261GB  
city              0.002GB  
local             0.000GB  
pcat              0.000GB
```



## ▪ *MongoDB Compass*

- Ferramenta gráfica client (gratuita);
- Download em <https://www.mongodb.com/products/compass>.



- ☑ O conceitos básicos da arquitetura do MongoDB.
- ☑ Três engines nativas de armazenamento, com comportamento e desempenho diferentes em um mesmo cenário.
- ☑ Autenticação via usuário local ou através de forma integrada com algum sistema externo de autenticação.
- ☑ Ferramentas client oficiais sendo o Shell (linha comando) ou o MongoDB Compass.

☐ Replica Set e Sharding.

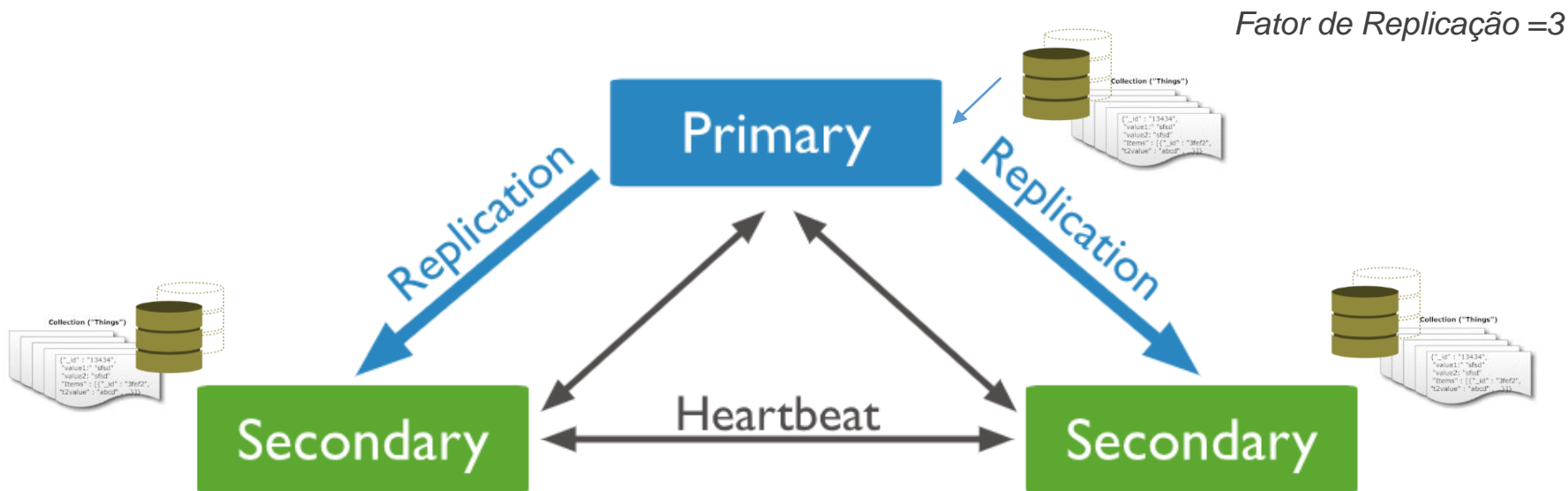


## **Aula 3.8.1. Replica Set e Sharding**

- ☐ Replicação de Dados.
- ☐ Particionamento de Dados.

- MongoDB é um SGBD **distribuído nativo**:
  - Recursos de alta disponibilidade;
  - Particionamento de dados;
  - Replicação de dados.
- Replicação → **Replica Set**
  - Cluster de servidores;
  - Dados sendo replicados entre eles → **oplog**;
  - De acordo com o **fator de replicação** configurado.

- Quanto à topologia: distribuído do **tipo hierárquico (*single-master*)**
- Quanto à consistência: **assíncrono**

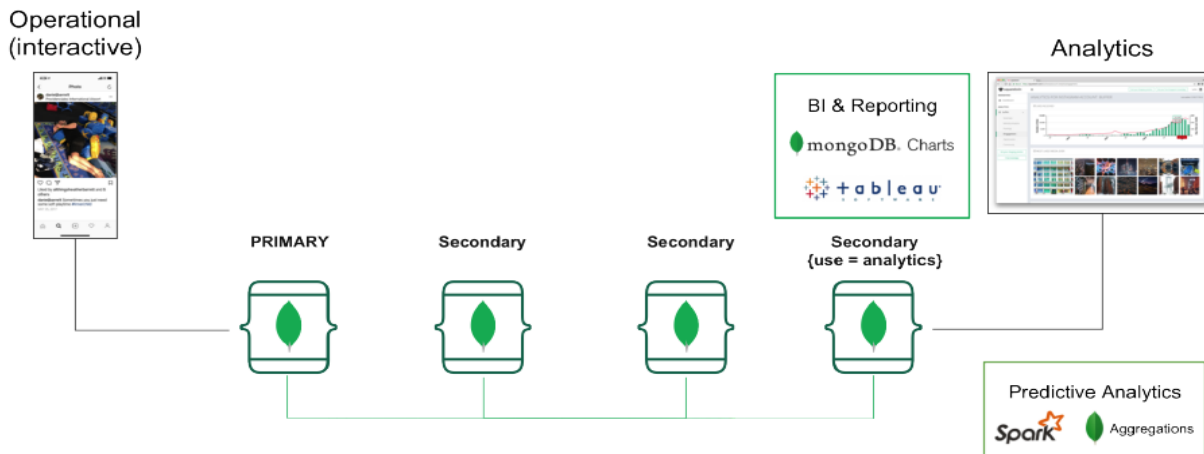


- Possibilita a **distribuição de dados global**;
  - Não há restrições para a localização física das réplicas;

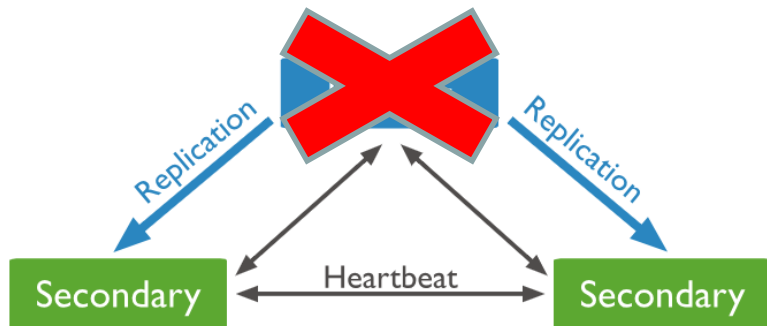




- Permite leituras nas réplicas secundárias:
  - **Distribuição de cargas de leituras;**
  - Coexistência de **workloads distintos** no mesmo ambiente.

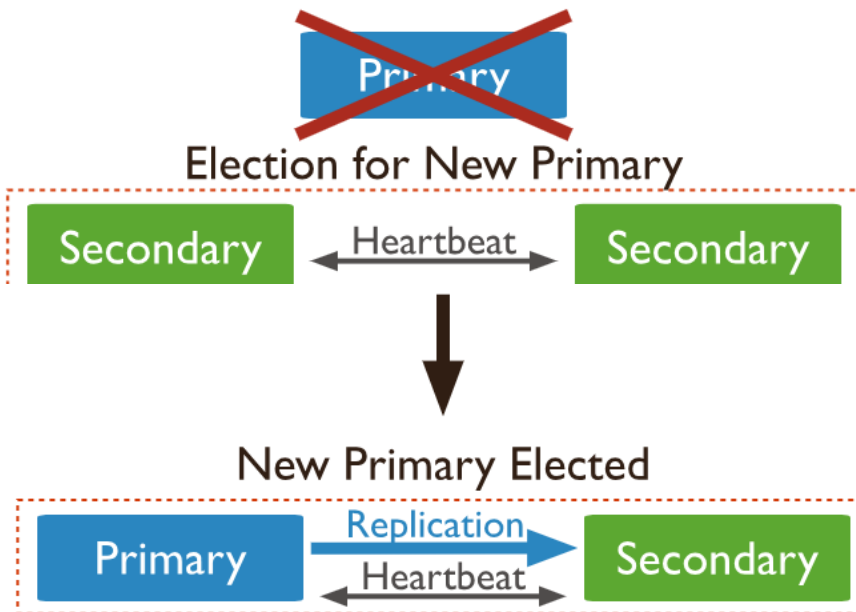


- Alta disponibilidade e tolerância à falhas de hardware:
  - Eleição de uma nova réplica primária** em caso de falhas.

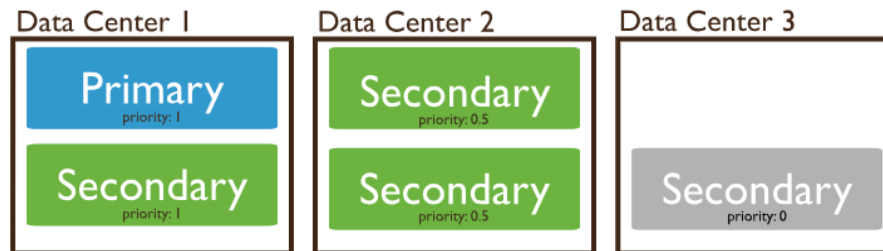


## ! Quantidade de Réplicas:

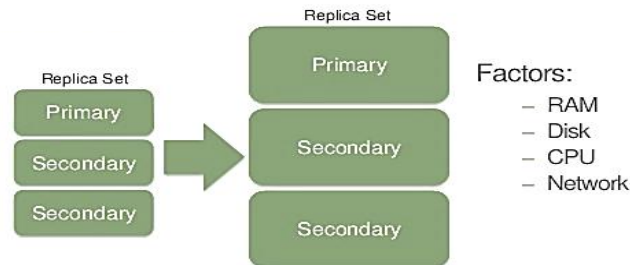
- Ímpar: sem empate;
- Par: usar árbitro para desempate.



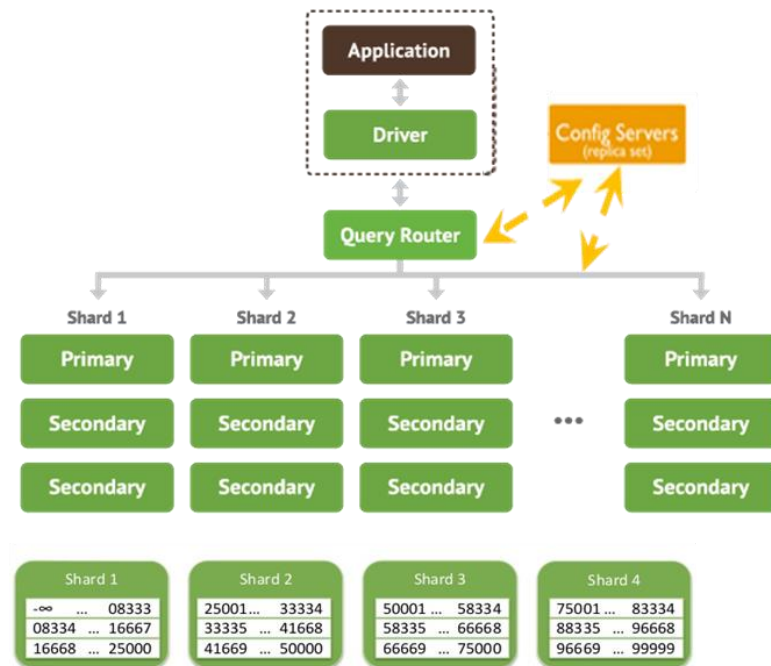
- *Disaster Recovey (DR);*



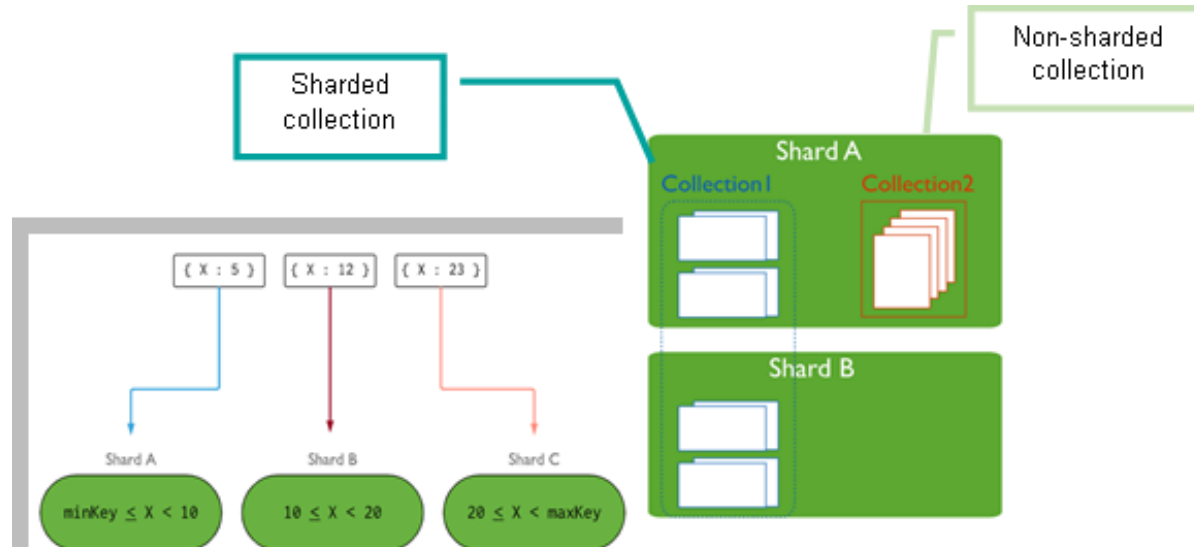
- **Escalabilidade Vertical:**



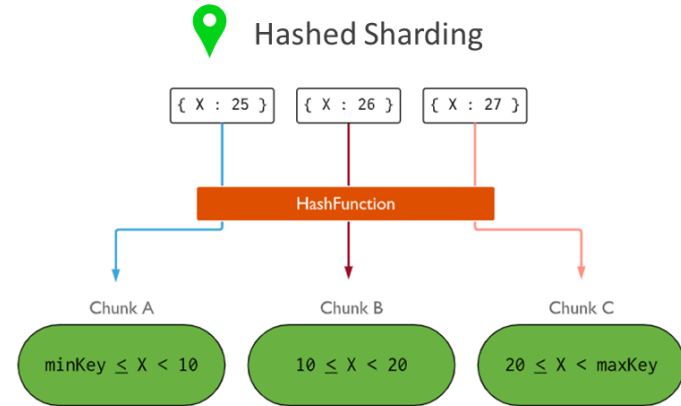
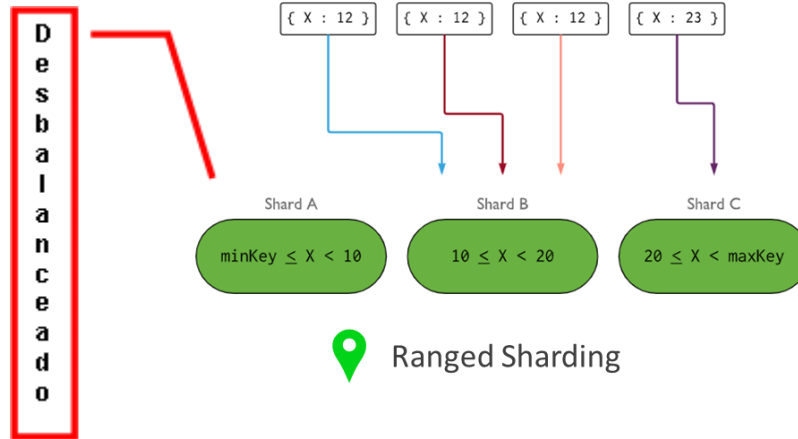
- **Sharding** → dados particionados de forma **horizontal**;
- Em conjunto com replica set:
  - A partir da versão 3.2;
  - Distribuição global de dados;
  - Alta disponibilidade;
  - Tolerância à falhas;
  - Topologia **multi-master**:
    - Leitura e escrita em todos servidores;
    - Escalabilidade horizontal.



- **Sharded Collections** → são divididas em fragmentos (*shards*);
- **Shard Key** → chave de particionamento de cada collection.

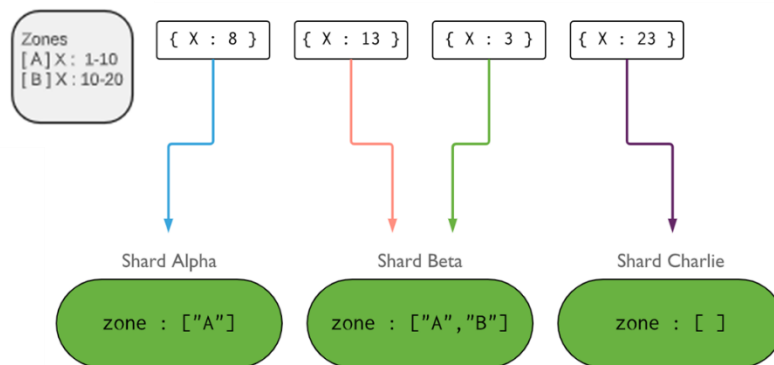


- Shard Key pode ser de dois tipos:
  - **Hash:** valores gerados e gerenciados pela própria engine do MongoDB;
  - **Range:** permitido definir os limites de valores (faixas) para cada shard.

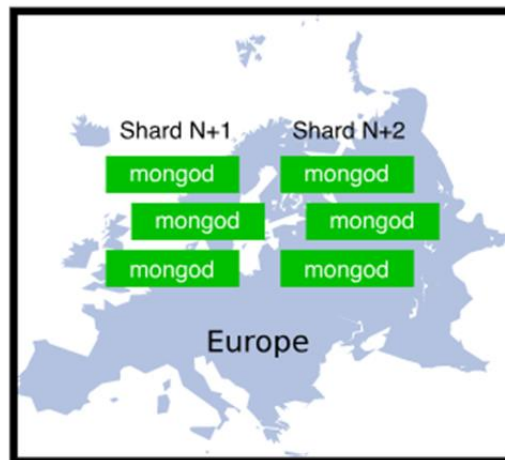
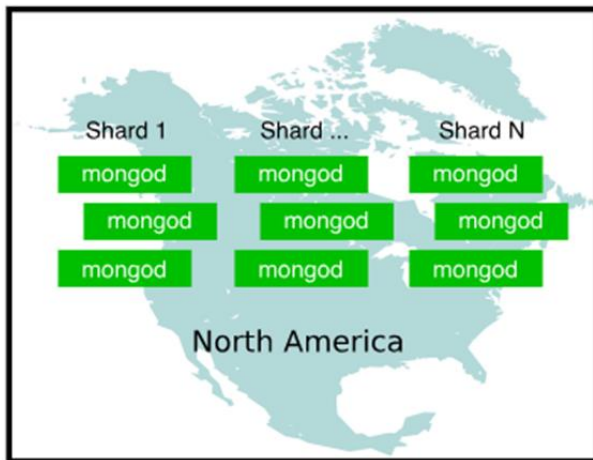


## ▪ Zonas (*zones*)

- Subconjunto de dados (faixas) que podem ser especificadas com base na shard key;
- Cada zona pode ser associada à um ou mais shards do cluster;
- Shard pode se associar a qualquer número de zonas.

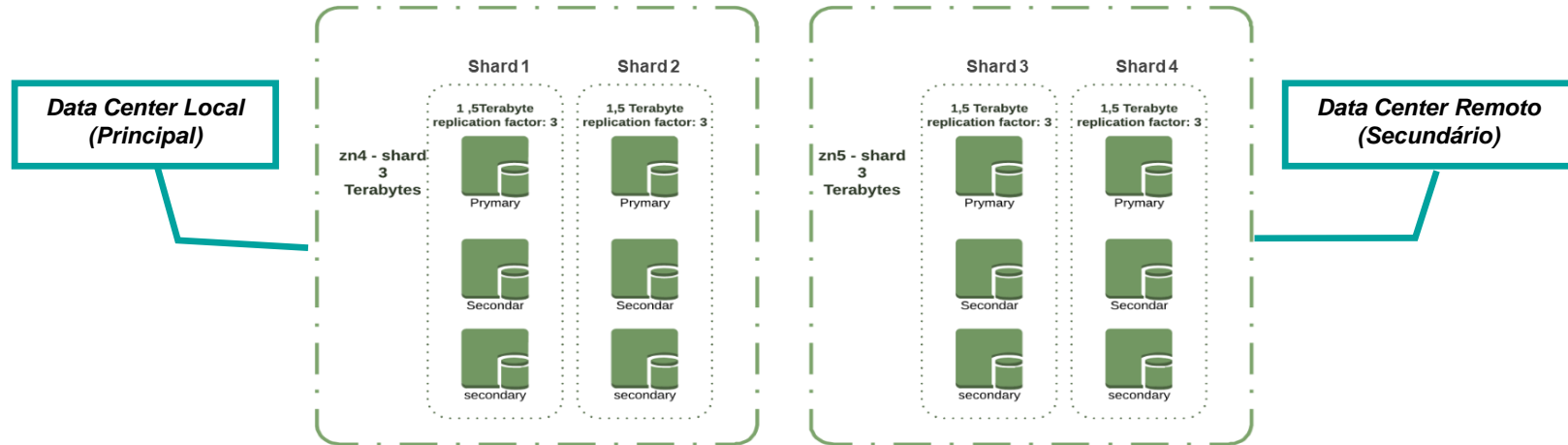


- Zonas + Sharding + Replica Set → **Disaster Recovery para Continuidade do Negócio**

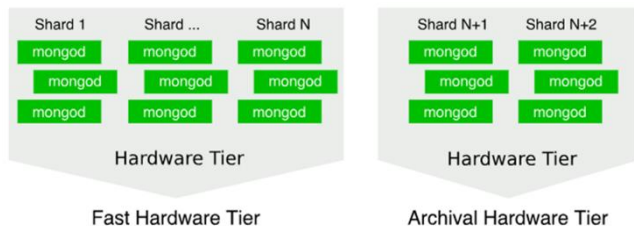




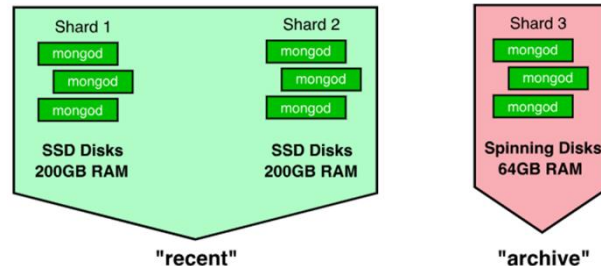
- Zonas + Sharding + Replica Set → **Dados mais relevantes em shards que estão mais próximos geograficamente ou que são mais estratégicos, e distribuição de carga entre data centers:**



- Zonas + Sharding + Replica Set:
  - **Otimização / Segmentação do Uso de Hardware:** shards menos importantes / histórico, em zonas com hardware menos performático.



- **Archive Automático:** isolar dados pouco acessados ou que não possuem requisitos de tempo de resposta baixo.



- ☑ Utilização de replica set para prover alta disponibilidade e distribuição de dados com topologia hierárquica assíncrona.
- ☑ Utilização de sharding para particionamento horizontal de dados, de forma a permitir topologia multi-master e escalabilidade horizontal.

- ❑ Instalação e configuração do MongoDB.



## **Aula 3.9. Instalação e configuração do MongoDB**

- ☐ Plataformas suportadas.
- ☐ Pré-requisitos e boas práticas de Instalação.
- ☐ Instalação Stand Alone.
- ☐ Instalação das ferramentas Client.

- Atualmente na **versão 4.0**, tem suporte para as plataformas:
  - **x86\_64**;
  - **ARM64**;
  - **PPC64LE (somente a edição Enterprise)**;
  - **s390x**;
  - Desde a versão 3.2 não suporta mais a plataforma 32-bits x86.
- **Edição *Community*** (gratuita) e ***Enterprise*** (paga);
- Edição nas nuvens → ***MongoDB Atlas***.

- Em termos de **sistema operacional**:
  - **Linux**, em praticamente quase todas as distribuições (*Amazon Linux, Debian, Red Hat, CentOS, Suse, Solaris e Ubuntu*);
  - **macOS**;
  - **Windows**;
  - **Docker** ➔ somente na edição Enterprise.



- Independentemente da topologia definida (Stand Alone, Replica Set ou Sharding), inicialmente a instalação do MongoDB é feita stand alone;
- Feito isso, realiza-se a configuração em replica set ou sharding, caso necessário;
- Nessa disciplina:
  - Instalação do MongoDB 4.0;
  - Servidor Linux x86\_64 com sistema operacional Red Hat 7;
  - Topologia stand alone.

- Desabilitar o ***SELinux***;
- Desativar o parâmetro de ***Transparent Hugepages*** e persistir para o próximo restart;
- Por default, o serviço do MongoDB roda com o usuário ***mongod***;
- **Diretórios** para separar os arquivos de sistema, dos arquivos dos bancos de dados, índices, log e journal. Ex.:

- *mongodb*
- *mongodb/data*
- *mongodb/log/*
- *mongodb/data/journal*

\* O tipo de file system recomendado para o MongoDB é o XFS, e a opção ***noatime*** deve ser setada.

- **Permissão de *root*** para o usuário que realizará a instalação;
    - Essa permissão não é necessária para administrar o MongoDB, bastando permissão de `sudo` para o usuário `mongod`, com as configurações abaixo:
- `/bin/su - mongod -s /bin/bash`
  - `/bin/vi /etc/yum.repos.d/mongodb-org-*. *.repo`
  - `/bin/rm /etc/yum.repos.d/mongodb-org-*. *.repo`
  - `/bin/vi /etc/yum.conf`
  - `/bin/yum * mongodb-org*`
  - `/bin/vi /etc/mongo*.conf`
  - `/bin/journalctl -xe`
  - `/sbin/service mongo* *`
  - `/bin/systemctl * mongo*`

- Em sistemas operacionais Linux:
  - Através de ***packages .rpm;***
    - Necessária comunicação do servidor onde o MongoDB será instalado, com o repositório de packages.
  - Usando ***Tarballs.***
  - Nessa disciplina → packages RPM.
- Como root, criar o arquivo ***/etc/yum.repos.d/mongodb-org-4.0.repo*** e adicionar as linhas abaixo no mesmo:

```
[mongodb-org-4.0]
name=MongoDB Repository
baseurl= https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.0/x86_64/
gpgcheck=1
enabled=1
gpgkey= https://www.mongodb.org/static/pgp/server-4.0.asc
```

- Repositório é composto por várias packages, que podem ser instaladas separadamente:
  - **mongodb-org-server**: contém o serviço da engine de banco de dados, scripts de inicialização e o arquivo de configuração do MongoDB (*configuration file*), que por padrão é criado em ***/etc/mongod.conf***;
  - **mongodb-org-mongos**: contém o serviço *mongos*, um serviço de roteamento para cenários com sharding, incumbido de processar consultas, determinando o local desses dados no cluster;
  - **mongodb-org-shell**: contém o *mongo shell* (*client linha de comando*);

- ***mongodb-org-tools***: contém ferramentas úteis para se trabalhar com o MongoDB, como *mongoimport*, *bsondump*, *mongodump*, *mongoexport*, *mongofiles*, *mongorestore*, *mongostat*, and *mongotop*;
  - ***mongodb-org***: instala todos os componentes.
- Para instalar, executar como root: ***yum install mongodb-org***;
  - Para instalar a última versão: ***yum install -y mongodb-org***;
  - Sem usar o arquivo de configuração do repositório → especificar a versão desejada para cada package de cada componente. Ex.:
    - ***yum install -y mongodb-org-4.0.6 mongodb-org-server-4.0.6 mongodb-org-shell-4***

- Consultar a versão instalada: ***mongod --version***
- Ajustar owner dos file systems do MongoDB para o usuário *mongod*
- Subir o serviço do MongoDB: *sudo service mongod start*
- Verificar status do serviço: *systemctl status mongod*
- Por default, na instalação:
  - Serviço do MongoDB é configurado para rodar na **porta 27017**;
  - Com o controle de acesso desabilitado: qualquer usuário consegue, localmente, logar na instância:
    - Comando ***mongo*** para iniciar o *mongo shell* e logar na instância.

# Instalação Stand Alone

- Próximo passo é configurar os parâmetros de inicialização do MongoDB de forma apropriada:

- Arquivo **mongod.conf**
- No diretório **/etc**
- Opções importantes:
  - Local do log do MongoDB
  - Local de armazenamento dos dados
  - Ativação do Journal
  - Separação de bancos por diretórios
  - Separação dos índices dos dados
  - Engine a ser utilizada

```
# mongod.conf

# for documentation of all options, see:
# http://docs.mongodb.org/manual/reference/configuration-options/

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# Where and how to store data.
storage:
  dbPath: /var/lib/mongo
  journal:
    enabled: true
# engine:
# mmapv1:
# wiredTiger:

# how the process runs
processManagement:
  fork: true # fork and run in background
  pidFilePath: /var/run/mongodb/mongod.pid # location of pidfile
  timeZoneInfo: /usr/share/zoneinfo

# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1 # Enter 0.0.0.0,:: to bind to all IPv4 and IPv6 addresses or, alternatively, use the net.bindIpAll setting.

#security:

#operationProfiling:

#replication:

#sharding:

## Enterprise-Only Options

#auditLog:
```



- Deve reiniciar o MongoDB para qualquer alteração no arquivo de configuração surtir efeito → *systemctl restart mongod*
- Antes de ativar o controle de acesso → criar usuário administrador

- Logar no MongoDB com o comando *mongo*;
- Executar script de criação o usuário administrador (DBA), colocando o nome (*user*) e a senha (*pwd*) desejada;

```
use admin
db.createUser(  {  user: "mongoadmin",
                  pwd: "s3nha",
                  roles:["root"]
                })
```

- Conferir se o usuário foi criado com o comando *show users*.

- Por fim, para ativar controle de acesso no MongoDB, adicionar no arquivo de configuração:  
*security:*  
*authorization: enabled*
- Para permitir conexões client-server, alterar, no *mongod.conf*, o parâmetro *bindip* de 127.0.0.1 para 0.0.0.0;
- Reiniciar o serviço do MongoDB: *systemctl restart mongod*;
- Após isso, é possível conexão remota, passando usuário e senha:
  - *mongo -host NOME\_SERVIDOR: PORTA -u USUARIO -p SENHA*

- ☑ MongoDB suporta várias plataformas e sistemas operacionais, possuindo edição gratuita, paga e nas nuvens;
- ☑ Os pré-requisitos e configurações de boas práticas são muito importantes para a performance e integridade do ambiente;
- ☑ A instalação, de certa forma é bem simples e amigável, podendo ser feita de modo semi-automatizado, usando-se o repositório de packages RPM.

- ❑ Definindo e criando um Banco de Dados MongoDB.



## **Aula 3.10. Definindo e criando um Banco de Dados MongoDB**

- ☐ Controle de acesso.
- ☐ Criação de Banco de Dados.
- ☐ Criação de coleções.
- ☐ Estrutura dos documentos.
- ☐ Tipos de Dados.
- ☐ Índices.

- MongoDB é 100% **RBAC (Role-Base Access Control)**;
  - Permissões (**actions**) → sempre concedidas via **role**
- Boa prática → gerenciamento centralizado dos usuários
  - Criação de usuários somente no banco *admin*
  - Concessão das roles nos bancos necessários
- Listar Usuários → **show users**
- Listar Roles → **show roles**
  - Existem as roles pré-definidas pelo MongoDB
  - Podem ser criadas roles customizadas

- Roles pré-definidas:
  - **root**: privilégios para administração da instância (todos os bancos);
  - **readAnyDatabase**: leitura em todos os bancos da instância;
  - **readWriteAnyDatabase**: leitura e escrita em todos os bancos;
  - **read**: privilégio de leitura no banco;
  - **readWrite**: privilégio de leitura e escrita no banco;
  - **dbAdmin**: privilégios para tarefas administrativas no banco;
  - **userAdmin**: para gerenciar usuários e permissões no banco;
  - **dbOwner**: readWrite + dbAdmin + userAdmin;
  - **backup**
  - **restore**



# Criação de Banco de Dados

- Dados (registros) → documentos JSON (formato BSON);
- Documentos → collections (coleções) → banco de dados.
- Para listar bancos de dados existentes → ***show dbs***

- Para utilizar um banco de dados (existente ou não)

→ ***use NOME\_BANCO***

- Para exibir banco utilizado pela sessão corrente → ***db***

```
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
> use banco_inexistente
switched to db banco_inexistente
> db
banco_inexistente
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
```

- **Não há um comando *CREATE DATABASE*** → o MongoDB criará o banco de dados quando você armazenar dados nesse banco de dados pela primeira vez.

- Para listar coleções existentes → ***show collections***
- Criação explícita → ***db.createCollection(...)***
- Criação implícita → o MongoDB criará a coleção automaticamente quando você armazenar dados na mesma pela primeira vez, ou quando criar um índice nela.
- **Capped Collection** → coleção com tamanho fixo
  - Funcionam como um buffer circular → conceito FIFO (*First In First Out*);
  - Garante e preserva a ordem de inserção dos documentos: > **throughput**.  
*db.createCollection("Exemplo\_Col\_Capped", {capped:true, size:10000, max:50000})*

- Formato JSON, composto por pares de campos e valores:

```
{  
  Campo1: valor1,  
  Campo2: valor2,  
  ...  
  CampoN: valorN  
}
```

- Pode ter documentos embutidos em outros documentos (***embedded documents***).

```
{  
  _id: <ObjectId>,  
  username: "123xyz",  
  contact: {  
    phone: "123-456-7890",  
    email: "xyz@example.com"  
  },  
  access: {  
    level: 5,  
    group: "dev"  
  }  
}
```

Embedded sub-document

Embedded sub-document

- Armazenados no formato binário BSON → mais tipos para especificação da natureza dos dados armazenado por cada campo.

```
{  _id: ObjectId("3100805ae3c4948bd2f768254"),
    nome: { primeiro_nome: "Gustavo", sobrenome: "Aguilar" },
    nascimento: new Date('Jul 05, 1978'),
    hobby: [ "Futebol", "Filmes", "Viagem" ],
    conexoes : NumberLong(1250000)
}
```

- » Campo **\_id** armazena o *ObjectId*: valor único identificador de cada documento, gerado automaticamente pelo MongoDB;
- » Campo **nome**: armazena um documento embutido, que contém dois campos (primeiro\_nome e sobrenome);
- » Campo **nascimento**: armazena valores do tipo data;
- » Campo **hobby**: armazena um array de strings;
- » Campo **conexões**: armazena um valor do tipo numérico longo.

- Para otimizar as consultas;
- Campo `_id` → MongoDB já cria um índice único automaticamente;
- Podem ser de apenas um campo ou de vários;
- Comando de criação → `db.NOME_COLLECTION.createIndex`

```
db.Pessoa.createIndex( { nome: 1 } )
```

```
db.Pessoa.createIndex( { conexoes: -1 } )
```

OBS.: 1 significa **ordenação crescente** e -1 **ordenação decrescente**.

- **TTL Indexes** ➔ removem automaticamente documentos de uma coleção após um determinado período;
- Remoção é feita em segundo plano ➔ mais performática do que a execução de um comando explícito de deleção de documentos;
- Ideais para certos tipos de informações que precisam persistir em um banco de dados apenas por um período de tempo finito:
  - Dados de eventos gerados por máquinas;
  - Logs;
  - Informações de sessão, etc.

- ☑ MongoDB é 100% RBAC, possuindo roles pré-definidas e permitindo criação de roles personalizadas;
- ☑ Não há comando explícito para criação de um banco, que é criado automaticamente ao se inserir dados no mesmo;
- ☑ Criação de collections pode ser explícita ou implícita;
- ☑ Documentos compostos por pares de campos e valores, armazenados como BSON, que expande os tipos de dados padrões de documentos JSON;
- ☑ Criação de índices como boa prática para otimizar consultas.

- ☐ Trabalhando com Bancos de Dados MongoDB.





## **Aula 3.11. Trabalhando com Bancos de Dados MongoDB**

- ☐ Inserindo Dados.
- ☐ Consultando Dados.
- ☐ Atualizando Dados.
- ☐ Deletando Dados.
- ☐ Operações Massivas de Dados.

- Primeiro passo → selecionar o banco desejado (***use nome\_banco***)
- Dois métodos para criação de documentos (inserção de dados) em collections:

- **db.nome\_da\_colecao.insertOne()**

- Insere apenas um documento;

```
db.users.insertOne(  ← collection
{
  name: "sue",        ← field: value
  age: 26,            ← field: value
  status: "pending"   ← field: value
}                    } document
)
```

- **db.nome\_da\_colecao.insertMany()**

- Insere vários documentos de uma vez.

```
db.inventory.insertMany([
  // MongoDB adds the _id field with an ObjectId if _id is not present
  { item: "journal", qty: 25, status: "A",
    size: { h: 14, w: 21, uom: "cm" }, tags: [ "blank", "red" ] },
  { item: "notebook", qty: 50, status: "A",
    size: { h: 8.5, w: 11, uom: "in" }, tags: [ "red", "blank" ] },
  { item: "paper", qty: 100, status: "D",
    size: { h: 8.5, w: 11, uom: "in" }, tags: [ "red", "blank", "plain" ] },
  { item: "planner", qty: 75, status: "D",
    size: { h: 22.85, w: 30, uom: "cm" }, tags: [ "blank", "red" ] },
  { item: "postcard", qty: 45, status: "A",
    size: { h: 10, w: 15.25, uom: "cm" }, tags: [ "blue" ] }
]);
```

- Método *find()*;
  - Pode ser usado com outras opções
    - Critério de busca (filtro);
    - Limite da quantidade de linhas retornadas (*limit*);
- db.usuario.find ( { idade: { \$gt 18 } } ).limit(5)**
- Formatação mais amigável da saída → *pretty()*

```
> db.movies.find().limit(3).pretty()
{
  "_id" : ObjectId("58c59c6a99d4ee0af9e0c34e"),
  "title" : "Bateau-mouche sur la Seine",
  "year" : 1896,
  "imdbId" : "tt0000042",
  "genre" : "Documentary, Short",
  "viewerRating" : 3.8,
  "viewerVotes" : 17,
  "director" : "Georges M\u00e9li\u00e8s"
}
{
  "_id" : ObjectId("58c59c6a99d4ee0af9e0c340"),
  "title" : "Watering the Flowers",
  "year" : 1896,
  "imdbId" : "tt0000035",
  "genre" : "Short",
  "viewerRating" : 5.3,
  "viewerVotes" : 33,
  "director" : "Georges M\u00e9li\u00e8s"
}
{
  "_id" : ObjectId("58c59c6a99d4ee0af9e0c34a"),
  "title" : "The Boxing Kangaroo",
  "year" : 1896,
  "imdbId" : "tt0000048",
  "genre" : "Short",
  "viewerRating" : 5.2,
  "viewerVotes" : 48,
  "director" : "Birt Acres"
}
>
```

- Operações que só podem ser feitas em uma coleção;
- São atômicas no nível de documento;
- Três métodos disponíveis:
  - **db.collection.updateOne()** → atualiza os campos do primeiro documento que satisfaça ao critério;
  - **db.collection.updateMany()** → atualiza vários documentos que atendam aos critérios, de uma só vez;
  - **db.collection.replaceOne()** → substitui um documento por outro.

- Podem ser especificados critérios (*update action*) ou filtros (*update filter*) → identificam os documentos a serem atualizados
- Filtros possuem a mesma sintaxe das operações de leitura
  - Podem usar os mesmos operadores

```
db.users.updateMany(  
  { age: { $lt: 18 } },  
  { $set: { status: "reject" } }  
)
```

← collection  
← update filter  
← update action

- Disponibilizados dois métodos:
  - **db.collection.deleteOne()** ➔ apaga o primeiro documento da coleção que satisfaça aos critérios;
  - **db.collection.deleteMany()** ➔ apaga todos os documentos da coleção que satisfaçam aos critérios.
  
- *db.movies.deleteMany ( { ano : “2000” } )*



# Operações Massivas de Dados

- *mongoexport*
- *mongoimport*
- *mongodump*
- *mongorestore*



- ☑ Tanto o banco de dados quanto as collections são criados implicitamente ao se inserir documentos pela primeira vez.
- ☑ MongoDB possui métodos para todas operações CRUD.

□ Capítulo 4 - Controle de Versão de Banco de Dados.