

Banco de dados NoSQL

BCD29008 – Engenharia de Telecomunicações

Prof. Emerson Ribeiro de Mello

mello@ifsc.edu.br

Licenciamento



Slides licenciados sob [Creative Commons "Atribuição 4.0 Internacional"](https://creativecommons.org/licenses/by/4.0/)

Motivação e conceitos

Banco de dados transacional – propriedades ACID

Garante que todas operações de consulta ou de alteração são **atômicas**, **consistentes**, **isoladas** e **duráveis**.

■ **A**tomicidade

- Todas operações (leitura/escrita) em uma transação são executadas com sucesso ou tudo é desfeito

■ **C**onsistência

- A execução de uma transação leva o banco de um estado consistente para um outro estado consistente

■ **I**solamento

- Transações podem acontecer de forma concorrente sem qualquer interferência

■ **D**urabilidade

- Ao concluir uma transação, todas modificações geradas serão persistentes

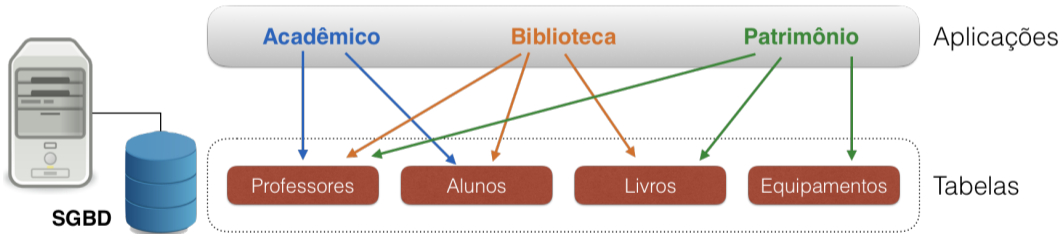
Banco de dados relacionais

- Modelo cliente-servidor
- Garante que todas transações são atômicas, consistentes, isoladas e duráveis (ACID)
- Linguagem padronizada para consulta (SQL)
- Dados estruturados
- Esquema estático
 - É possível alterar a modelagem posteriormente, porém pode ser custoso
- Não foram projetados para possibilitar o escalonamento horizontal
- *Frameworks* para mapeamento objeto-relacional (ORM) trazem facilidades para uso de linguagens de programação que seguem a POO

Abordagens para compartilhamento por diversas aplicações

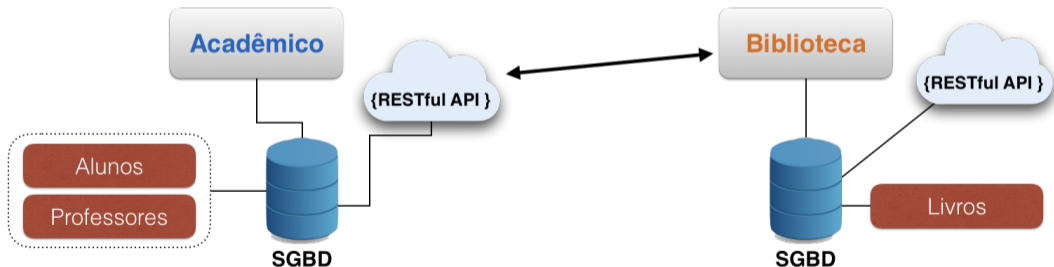
■ Servidor centralizado e com um único esquema

- Diferentes aplicações armazenam dados em um único local, facilitando assim a integração entre essas



Abordagens para compartilhamento por diversas aplicações

- Um esquema por aplicação, podendo cada um estar em um único servidor ou distribuído por vários servidores
 - Cada esquema é acessado somente por uma única aplicação
 - *Web Services* são usados para permitir a troca de dados entre aplicações



Motivação para um novo paradigma

- **Migração do modelo centralizado para o modelo com um banco de dados por aplicação**
 - integração por meio de *Web Services*
- **Ser escalável é algo crucial para soluções na era da *Big Data***
 - Grande volume de dados, gerado com grande velocidade e grande variedade
- **Algumas aplicações requerem um esquema de dados flexível**
 - Mudar o esquema de um banco relacional é custoso
- **Escalabilidade horizontal é mais viável que a escalabilidade vertical**
 - SGBD relacionais não operam bem com escalonamento horizontal

Banco de dados NoSQL

NoSQL fundamentado sobre modelo de consistência eventual (BASE)

■ **Basic Availability**

- Todo pedido terá uma resposta, porém a resposta pode indicar uma falha na tentativa de obter o dado ou que o dado retornado está em um estado inconsistente

■ **Soft-state**

- O estado do sistema pode alterar ao longo do tempo, mesmo durante intervalo de tempo que não houve qualquer escrita

■ **Eventual consistency**

- Sistema se tornará consistente ao longo do tempo uma vez que não se tenha novas operações de escrita

Sistema Gerenciador de Banco de Dados NoSQL

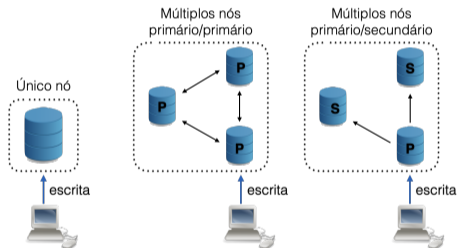
- **Não possui esquema de dados** ou possui um esquema flexível
 - Dados, seus relacionamentos, semântica e restrições de consistência
- **Não usa linguagem SQL**
 - Algumas implementações possuem linguagem de consulta semelhante à linguagem SQL
- **Simplicidade de projeto, facilidade para escalonamento horizontal**
 - Estrutura de dados propicia um desempenho melhor de algumas operações se comparado com os banco de dados relacionais
 - Dados podem ser particionados por diferentes nós

Como fazer a distribuição dos dados

- **Um único servidor de banco de dados** (sem distribuição)
 - Simplicidade na administração do BD e no desenvolvimento da aplicação que o usa

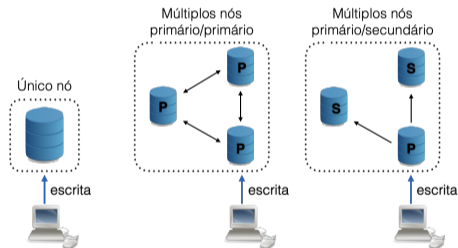
Como fazer a distribuição dos dados

- **Um único servidor de banco de dados** (sem distribuição)
 - Simplicidade na administração do BD e no desenvolvimento da aplicação que o usa
- **Replicação por diversos nós**
 - Replica o mesmo conjunto de dados em diferentes nós



Como fazer a distribuição dos dados

- **Um único servidor de banco de dados** (sem distribuição)
 - Simplicidade na administração do BD e no desenvolvimento da aplicação que o usa
- **Replicação por diversos nós**
 - Replica o mesmo conjunto de dados em diferentes nós



■ Fragmentação por diversos nós

- diferentes conjuntos de dados por diferentes nós, sendo cada nó o único responsável pelo conjunto de dados que armazena

Tipos e exemplos de bancos de dados NoSQL

■ chave-valor

- Redis, MemcacheDB, Riak

■ orientado a documento

- MongoDB, Apache CouchDB

■ orientado a coluna

- Apache HBase, Apache Cassandra

■ baseado em grafos

- Neo4j

APACHE
HBASE

 Cassandra


CouchDB
relax



 riak

 mongoDB

HYPERTABLE^{INC}

 Neo4j



redis

Armazenamento: chave-valor

Armazenamento: chave-valor

- Dados representados como uma coleção de pares chave e valor

```
codigo : "BCD29008"  
nome   : "Banco de dados"
```

- **Consultas são limitadas às chaves**

- get – obter um valor a partir de uma chave
- put – inserir um valor para uma dada chave
- post – atualizar um valor para uma dada chave
- delete – remover um chave
- Valor pode ser texto ou binário (BLOB)
 - A aplicação é responsável por interpretar o valor
- Casos de uso
 - Informações de sessão de aplicação web, carrinho de compras

**Armazenamento:
orientado a documento**

Sintaxe do JSON – <http://json.org/>

- Um **objeto** é um conjunto não ordenado de pares chave/valor
- Os pares são encapsulados por chaves { e }
- Dois pontos é usado para separar chave de valor
- Vírgula é usada para separar pares

```
{ "nome" : "Alice", "sobrenome" : "dos Santos" }
```

- Um **vetor** é uma coleção ordenada de valores
 - Os valores são encapsulados por colchetes [e]
 - Vírgula é usada para separar valores

```
[ "poo", "std", "bcd" ]
```

Sintaxe do JSON – <http://json.org/>

- Strings são delimitadas por aspas
- Números são representados obrigatoriamente na forma decimal e seguem a sintaxe de representação do C/C++/Java
- Um **vetor** pode conter string, número, true, false, null, objeto ou vetor

```
{ "telefones": [  
  {  
    "tipo": "residencial",  
    "valor": "48 3381 2800"  
  },  
  {  
    "tipo": "celular",  
    "valor": "48 93381 2800"  
  }  
]  
}
```

Armazenamento: orientado a documento

- **Dados são representados como uma coleção de pares chave e valor**
 - **No valor se tem documentos** que representam informações de forma estruturada (ex: objeto JSON)
- **Documento consiste de um conjunto de pares chave e valor**
 - **Cada documento tem sua própria estrutura** (semi-estruturado), o que contrasta com os banco de dados de relacionais onde se tem um esquema rígido

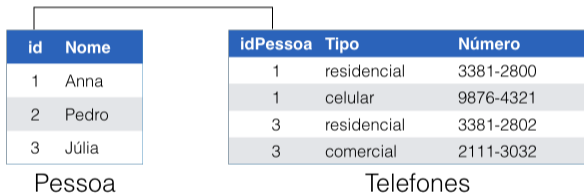
```
{
  "_id": "1234",
  "nome": "Alice",
  "endereco": {
    "rua": "Rua José Lino Kretzer, 608",
    "cidade": "São José",
    "uf": "SC"
  }
}
```

Modelo de dados – modelo relacional

id	Nome	Residencial	Celular	Comercial
1	Anna	3381-2800	9876-4321	
2	Pedro			
3	Júlia	3381-2802		2111-3032

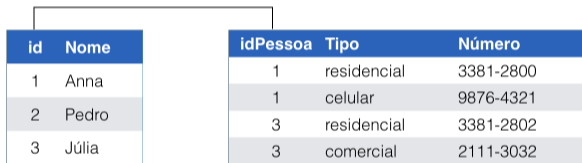
Modelo de dados – modelo relacional

id	Nome	Residencial	Celular	Comercial
1	Anna	3381-2800	9876-4321	
2	Pedro			
3	Júlia	3381-2802		2111-3032



Modelo de dados – modelo relacional

id	Nome	Residencial	Celular	Comercial
1	Anna	3381-2800	9876-4321	
2	Pedro			
3	Júlia	3381-2802		2111-3032



Modelo de dados – orientado a documento

- **Cada documento pode ter seu próprio conjunto de pares**
 - Um par pode estar presente em um documento, porém não em outro
 - Uma chave pode ter um nome diferente
- **Documentos não permitem chaves com valor vazio**
 - Se não tem valor, então não se deve ter chave
- **É possível adicionar novos pares a qualquer instante**
 - Não é necessário alterar documentos que já foram persistidos anteriormente

Modelo de dados – orientado a documento

id	Nome	Residencial	Celular	Comercial
1	Anna	3381-2800	9876-4321	
2	Pedro			
3	Júlia	3381-2802		2111-3032

Modelo de dados – orientado a documento

id	Nome	Residencial	Celular	Comercial
1	Anna	3381-2800	9876-4321	
2	Pedro			
3	Júlia	3381-2802		2111-3032

```
{
  _id: "1",
  nome: "Anna",
  telefones: ["3381-2800", "9876-4321"]
}
{
  _id: "2",
  nome: "Pedro"
}
{
  _id: "3",
  nome: "Júlia",
  telefones: ["3381-2802", "2111-3032"]
}
```

Modelo de dados – orientado a documento

id	Nome	Residencial	Celular	Comercial
1	Anna	3381-2800	9876-4321	
2	Pedro			
3	Júlia	3381-2802		2111-3032

```
{
  _id: "1",
  nome: "Anna",
  telefones: ["3381-2800", "9876-4321"]
}
{
  _id: "2",
  nome: "Pedro"
}
{
  _id: "3",
  nome: "Júlia",
  telefones: ["3381-2802", "2111-3032"]
}
```

```
{
  _id: "1",
  nome: "Anna",
  telefones: [
    {residencial: "3381-2800"},
    {celular: "9876-4321"}
  ]
}
{
  _id: "2",
  nome: "Pedro"
}
{
  ...
}
```

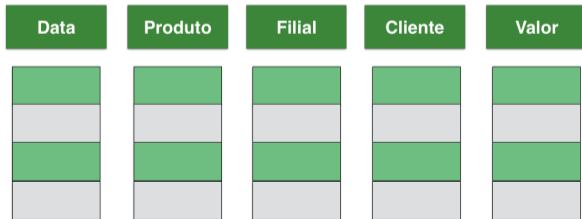
Orientado a columna

Orientado a coluna



orientado a colunas

- Armazena colunas de forma contínua no disco
- Requer **múltiplas operações de escrita** para armazenar um **único registro**
- Adequado para processamento analítico de dados
 - Leituras frequentes e intensas em uma grade base de dados



Relacional orientado a linhas

- Armazena registros de forma contínua no disco
- **Todos os dados de um registro (linha)** são armazenados com uma **única operação de escrita**
- **Eficiente** quando se deseja **obter todos os dados de um registro**, porém **menos eficiente** quando se deseja **ler poucas colunas de muitos registros**

Orientado a coluna – algumas características

■ Eficiência na leitura

- Apresentar 10 colunas de um registro que possui 200 colunas
- Banco de dados relacional lê do disco todas as colunas de uma linha e depois apresenta somente as 10
- No orientado a coluna serão lidas somente as 10 colunas do disco

■ Eficiência na compressão dos dados

- Colunas possuem uma taxa de compressão maior do que linhas
- Linhas contém valores de diferentes domínios

■ Algumas implementações

- Google Bigtable, Apache Hadoop, Apache Cassandra, Amazon SimpleDB

Banco de dados baseado em grafos

Banco de dados baseado em grafos

- **Chave-valor, documentos e orientado a colunas**

- Focado no armazenamento dos dados

- **Baseado em grafos**

- Tem como base a teoria dos grafos e os dados são representados como um grafo dirigido
- Adequado quando as informações de relacionamento entre os dados ou mesmo a topologia dos dados é mais importante que os próprios dados
- A aplicação não precisa inferir sobre os relacionamentos entre os dados, usando, por exemplo, chave estrangeira
 - Em quais restaurantes Juca já almoçou em São José? - no modelo relacional seria necessário fazer uso de junções

Baseado em grafos - Componentes

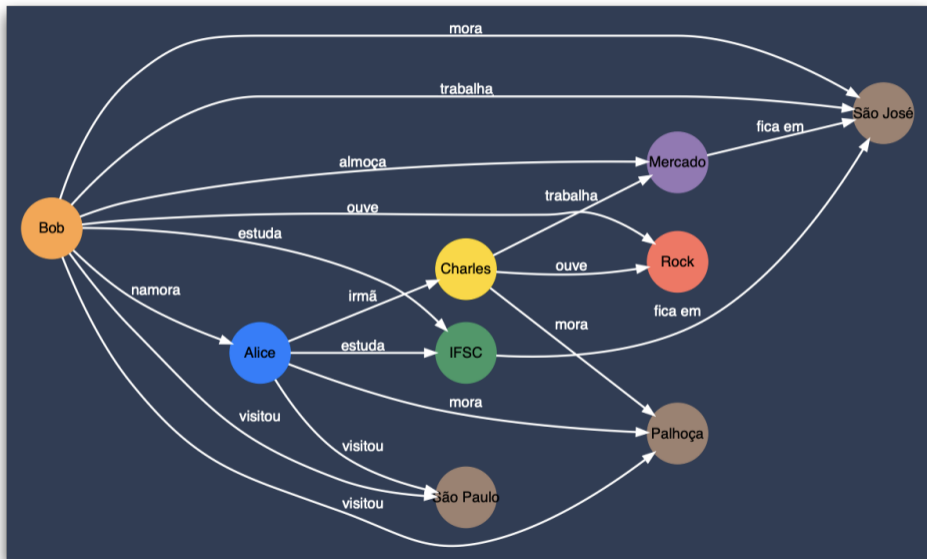
- **Nós** (vértices)

- Representa uma entidade. Ex: pessoa, lugar, categoria, disciplina, etc
- Pode conter propriedades (atributos). Ex: idade, cidade

- **Arestas** (arcos)

- Representa o relacionamento entre dois nós. Ex: amigo, aluga, possui, matrícula
- Pode conter propriedades. Ex: desde 2019

Baseado em grafos - Componentes



Cenários de uso típicos

■ Detecção de fraudes

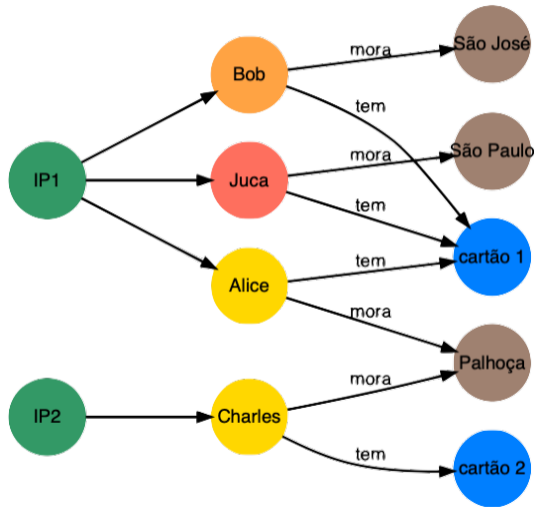
- Várias pessoas compartilhando um mesmo endereço IP, porém residentes em diferentes cidades

■ Sistema de recomendação *online*

- Apresentar outros acessórios para combinar com a roupa que foi colocada no carrinho

■ Gestão de identidade e controle de acesso

- Sistema Operacional, navegador, IP de origem



Prática

Redis: chave-valor

Redis

- Servidor de estrutura de dados
 - Provê acesso a estrutura de dados mutáveis e compartilhada
 - Segue o modelo cliente-servidor sobre o TCP
 - Possui funcionalidades como replicação, clustering e alta disponibilidade
- Por padrão mantém informações em memória, porém é possível persistir todos os dados de uma única vez em disco
- Provê suporte para diferentes estruturas de dados
 - string, hash, listas, conjuntos, imagens, logs, coordenadas geoespaciais, etc



Instalação e execução do servidor e cliente Redis

■ Instalação local

■ Servidor

- Baixe o código fonte em <https://redis.io/download>, compile e execute o comando `ONDE-VOCÊ-SALVOU/src/redis-server`
- Ou instale via gerenciador de pacotes de sua distribuição

■ Cliente

- `ONDE-VOCÊ-SALVOU/src/redis-cli`

■ Uso do docker

■ Servidor

```
docker run --name meu-redis -p 6379:6379 --rm -d redis
```

■ Cliente

```
docker exec -it meu-redis redis-cli
```


Interações com o cliente redis

- Adicionando e recuperando valor associado a uma chave

```
set codigo "BCD29008"  
  
get codigo
```

- Para exigir que os cliente se autentiquem, é necessário configurar o parâmetro `requirepass` dentro do arquivo `redis.conf`¹
 - `requirepass senha-muito-facil`

```
# executando o servidor e indicando o arquivo de configuração  
./src/redis-server redis.conf  
  
# no cliente Redis  
auth senha-muito-facil  
get codigo
```

¹Com Docker será necessário criar um Dockerfile, veja em https://hub.docker.com/_/redis

Cliente Java usando a biblioteca Jedis

<https://redis.io/clients#java>

- Adicionar linha abaixo dentro da seção de dependências no arquivo build.gradle

```
dependencies {  
    implementation 'redis.clients:jedis:4.1.1'  
}
```

```
import redis.clients.jedis.Jedis;  
  
public class App {  
    public static void main(String[] args) {  
  
        Jedis jedis = new Jedis("localhost", 6379);  
  
        jedis.set("campus", "São José");  
        System.out.println(jedis.get("campus"));  
    }  
}
```

Cliente Python3 usando a biblioteca redis-py

<https://redis.io/clients#python>

- Criar ambiente virtual e instalar a biblioteca redis-py

```
python3 -m venv venv
source venv/bin/activate
pip install redis
```

```
import redis

r = redis.Redis(host='localhost', port=6379, db=0, password='senha')

r.set('campus', 'São José')
valor = r.get('campus')
```

MongoDB: orientado a documentos

Instalação do servidor MongoDB com Docker

■ Servidor

```
docker run --name meu-mongodb -p 27017:27017 --rm -d mongo
```

■ Cliente - MongoDB Shell

```
docker exec -it meu-mongodb mongosh
```

Documentação oficial



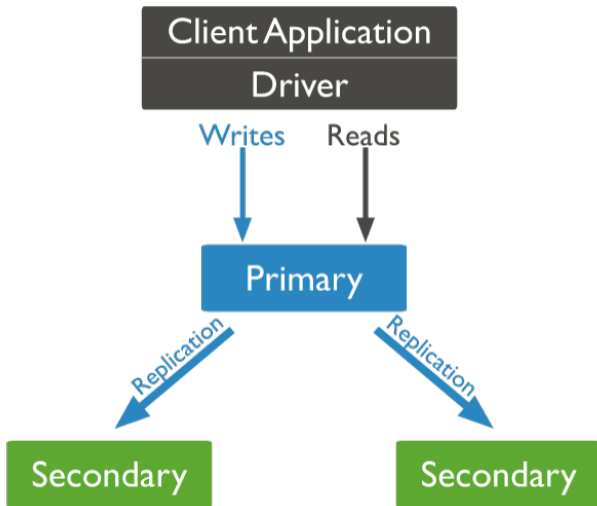
- <https://docs.mongodb.com/manual>
- https://hub.docker.com/_/mongo

Distribuição e consistência por várias instâncias MongoDB

- Para as operações de escrita é possível configurar como essa será propagada pelas réplicas
 - Define-se o número de réplicas que devem ser atualizadas antes da operação ser considerada como bem-sucedida
 - Em um ambiente com uma única réplica, a gravação será confirmada imediatamente
 - Em um ambiente com 3 réplicas e optar pela maioria, então só será confirmada depois que estiver escrita em pelo menos 2 réplicas
- Tais parâmetros podem ser configurados na conexão ou na criação da coleção ou mesmo para cada operação de escrita
 - Transações no nível de um único documento são consideradas atômicas

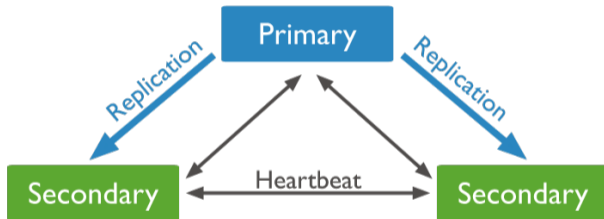
MongoDB com várias réplicas

<https://docs.mongodb.com/manual/replication/>



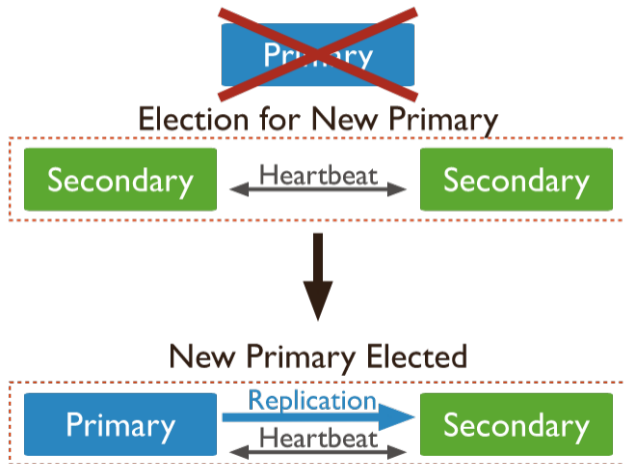
MongoDB com várias réplicas

<https://docs.mongodb.com/manual/replication/>



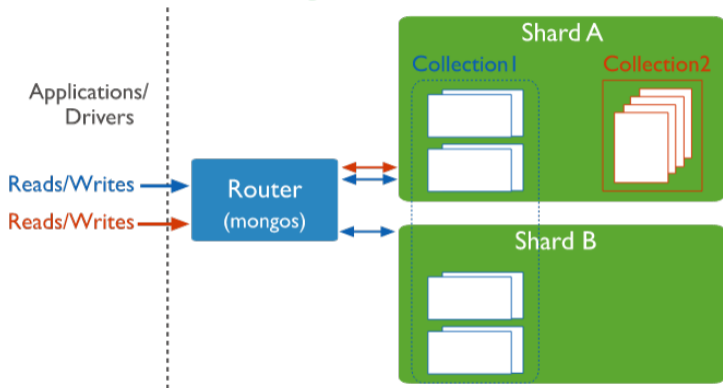
MongoDB com várias réplicas

<https://docs.mongodb.com/manual/replication/>



Fragmentação

<https://docs.mongodb.com/manual/sharding/>



- Uma coleção pode ter seus documentos espalhados por diferentes nós em um cluster MongoDB
- **mongos** atua como um roteador entre as aplicações clientes e o *cluster*

Modelo de dados

MySQL	MongoDB
Esquema	Database
Tabela	Coleção
Registro	Documento
Chave primária	campo <code>_id</code>
Junção	<code>\$lookup</code>

- Uma instância do MongoDB pode conter vários **databases**
- Um **database** é composto por um conjunto de **coleções**
- Um **documento** é um conjunto não ordenado de pares (chave,valor) sem um esquema rígido e que fica armazenado em uma **coleção**
 - Cada **documento** em uma **coleção** pode possuir um número diferente de **atributos**
 - Documentos BSON (*an extended Binary form of JSON*)

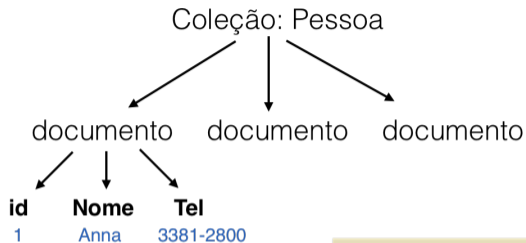
Modelo de dados

Tabela: Pessoa

Colunas

Linhas

id	Nome	Tel
1	Anna	3381-2800
2	Pedro	3381-2801
3	Júlia	3381-2802



```
{
  _id: "1"
  nome: "Anna"
  tel: "3381-2800"
}
{
  _id: "2"
  nome: "Pedro"
  tel: "3381-2801"
}
```

Coleção: Pessoa

Alguns comandos com o MongoDB Shell

<https://www.mongodb.com/basics/examples>

```
# Listando os bancos de dados
show dbs;

# Listando as coleções
use meuBanco;
show collections;

# Adicionando um documento em uma coleção chamada pessoas
db.pessoas.insertOne({"nome": "Juca", "email": "juca@email.com"})

# Listando a quantidade de documentos em uma coleção chamada pessoas
db.pessoas.countDocuments()

# Obtendo todos os documentos contidos em uma coleção chamada pessoas
db.pessoas.find()

# Obtendo um documento por meio de seu id
db.pessoas.find({_id: ObjectId("622a21d31108c0fd7f72fcf8")})

# Obtendo um documento por meio de um valor associado a chave 'nome'
db.pessoas.find({nome: "Juca"})
```

Comparando algumas instruções SQL com o MongoDB

■ Criando uma tabela

```
CREATE TABLE pessoa(  
  idPessoa INT NOT NULL AUTO_INCREMENT,  
  nome VARCHAR(40),  
  email VARCHAR(40),  
  PRIMARY KEY (idPessoa)  
)
```

```
db.pessoas.insertOne(  
  {  
    nome: "Juca",  
    email: "juca@email.com"  
  }  
)
```

■ Alterando uma tabela

```
ALTER TABLE pessoa  
  ADD data DATETIME  
  
ALTER TABLE pessoa  
  DROP COLUMN data
```

```
db.pessoas.updateMany(  
  { },  
  {$set: {data: new Date()}}  
)  
db.pessoas.updateMany(  
  { },  
  {$unset: {data: ""}}  
)
```

Comparando algumas instruções SQL com o MongoDB

■ Inserindo registros / documentos

```
INSERT INTO pessoa(nome, email)
VALUES ("José", "jo@em.com")
```

```
db.pessoas.insertOne(
  {
    nome: "José",
    email: "jo@em.com"
  }
)
```

■ Listando dados de uma tabela / documento

```
SELECT * FROM pessoa

SELECT nome FROM pessoa

SELECT * FROM pessoa
WHERE nome = "Juca" AND uf = "SC"
```

```
db.pessoa.find()

db.pessoa.find({}, {nome: 1})

db.pessoa.find({nome: "Juca", uf: "SC"})
```

MongoDB - CRUD

<https://docs.mongodb.com/manual/crud/>

■ Inserir

```
db.users.insertOne(
  {
    name: "sue",
    age: 26,
    status: "pending"
  }
)
```

← collection

← field: value
← field: value
← field: value } document

■ Obter

```
db.users.find(
  { age: { $gt: 18 } },
  { name: 1, address: 1 }
).limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

MongoDB - CRUD

<https://docs.mongodb.com/manual/crud/>

■ Atualizar

```
db.users.updateMany(                                     ← collection
  { age: { $lt: 18 } },                                 ← update filter
  { $set: { status: "reject" } }                       ← update action
)
```

■ Excluir

```
db.users.deleteMany( ← collection
  { status: "reject" } ← delete filter
)
```

Inserção

■ Inserindo um único documento

```
db.produtos.insertOne(  
  { item: "água", qtd: 10, tipo: "sem gás", ml: 500 } );
```

■ Inserindo múltiplos documentos

```
db.produtos.insertMany([  
  { item: "televisor", qtd: 25, tags: ["lcd", "led"],      dimensao: { a: 14, l: 21,  
    unidade: "cm" } },  
  { item: "microondas", qtd: 15, tags: ["embutir", "inox"], dimensao: { a: 19, l:  
    22.85, unidade: "cm" } }  
]);
```

Recuperação

<https://docs.mongodb.com/manual/tutorial/query-documents/>

■ Recuperando todos documentos em uma coleção

```
db.produtos.find( { } );
```

■ Recuperando um documento

```
db.produtos.find( { item: "água" } );
```

■ Usando operadores lógicos AND e OR

```
db.produtos.find( { item: "água", qtd: { $lt: 100 } } );
```

```
db.produtos.find( { $or: [ { item: "água" }, { qtd: { $gt: 5 } } ] } );
```

Atualização

<https://docs.mongodb.com/manual/tutorial/update-documents/>

■ Métodos para atualização

```
db.collection.updateOne(<filter>, <update>, <options>)  
db.collection.updateMany(<filter>, <update>, <options>)  
db.collection.replaceOne(<filter>, <update>, <options>)
```

■ Atualizando um único documento

```
db.produtos.updateOne( {item: "microondas"},  
                      {$set: {"dimensao.1" : 23}})
```

Exclusão

<https://docs.mongodb.com/manual/tutorial/remove-documents/>

■ Métodos para exclusão

```
db.collection.deleteMany()  
db.collection.deleteOne()
```

■ Excluindo no máximo um único documento (vai excluir o primeiro)

```
db.produtos.deleteOne( {qtd : 100})
```

MongoDB com Java

- Dependência no arquivo `build.gradle` para o MongoDB Synchronous Driver
- *MongoDB Java Driver* é uma versão legada que não deve mais ser usada

```
dependencies {  
    implementation 'org.mongodb:mongodb-driver-sync:4.5.0'  
}
```



MongoDB com Java

```
public static void main(String[] args) {  
  
    String uri = "mongodb://localhost:27017";  
  
    try (MongoClient mongoClient = MongoClient.create(uri)) {  
  
        MongoDBDatabase database = mongoClient.getDatabase("meuBanco");  
  
        // database.createCollection("clientes");  
        MongoCollection<Document> collection = database.getCollection("clientes");  
  
        Document document = new Document();  
  
        document.append("codigo", "123");  
        document.append("nome", "José");  
        document.append("dataNasc", LocalDate.of(1980, 2, 10));  
        document.append("saldo", 102.30);  
  
        collection.insertOne(document);  
    } catch (Exception e) {System.err.println(e.toString());}  
}
```

MongoDB com Java

```
public static void main(String[] args) {  
  
    String uri = "mongodb://localhost:27017";  
  
    try (MongoClient mongoClient = MongoClient.create(uri)) {  
  
        MongoDBDatabase database = mongoClient.getDatabase("meuBanco");  
  
        MongoCollection<Document> collection = database.getCollection("clientes");  
  
        // https://docs.mongodb.com/drivers/java/sync/current/fundamentals/crud/read-operations/retrieve/  
        Bson filtro = Filters.all("nome", "José");  
  
        collection.find(filtro).forEach(doc-> System.out.println(doc.toJson()));  
  
    } catch (Exception e) {System.err.println(e.toString());}  
}
```


Aulas baseadas em

-  Sullivan, David G.
Computer Science E-66 – Harvard University
-  MongoDB Official Documentation
<https://docs.mongodb.com/manual/>