

BD NoSQL



Profa. Kelly Lais Wiggers

Tipos

Chave/valor	 RocksDB   redis
Documento	 mongoDB  Couchbase
Coluna	  cassandra  amazon DynamoDB  APACHE HBASE 
Grafo	 JanusGraph  neo4j

Tipos

1. **Bancos de Dados de Documentos (Document Store):** Neste tipo de banco de dados, os dados são armazenados em documentos semelhantes a JSON ou BSON (Binary JSON). Cada documento pode conter informações heterogêneas e não precisa seguir um esquema rígido. Exemplos populares são o MongoDB e o Couchbase.
2. **Bancos de Dados de Chave-Valor (Key-Value Store):** Nesse tipo de banco de dados, os dados são armazenados como pares de chave e valor, onde a chave é um identificador único para o valor associado. São altamente eficientes em termos de acesso rápido aos dados. Exemplos incluem o Redis e o Amazon DynamoDB.
3. **Bancos de Dados de Família de Colunas (Column-Family Store):** Esses bancos de dados organizam os dados em famílias de colunas, permitindo que cada linha contenha diferentes colunas. São adequados para cenários em que os dados têm estruturas complexas. O Apache Cassandra é um exemplo popular dessa categoria.
4. **Bancos de Dados de Grafos (Graph Database):** Bancos de dados de grafos são otimizados para armazenar e recuperar dados interconectados. Eles utilizam a teoria dos grafos para representar e processar relações entre os dados. O Neo4j é um exemplo conhecido de banco de dados de grafos.

Tipos

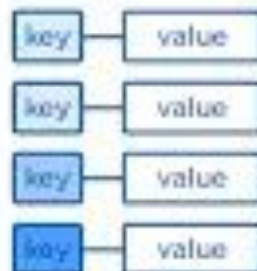
Document



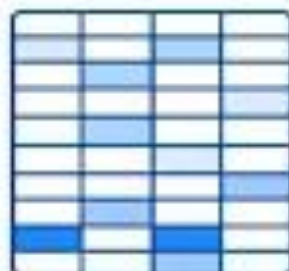
Graph



Key-Value



Wide-column



Atenção

É importante mencionar que a escolha entre um banco de dados relacional ou NoSQL **depende das necessidades específicas do projeto**, do **volume de dados**, dos **requisitos de escalabilidade**, do **modelo de dados** e das **habilidades da equipe de desenvolvimento**.

Banco de dados baseado em documentos

Os bancos de dados NoSQL orientados a documentos podem ser considerados como os mais populares atualmente entre as quatro categorias existentes.

A principal diferença entre MongoDB e os principais sistemas tradicionais de bancos de dados relacionais é que, em vez de tabelas, linhas e colunas, a base para armazenamento no MongoDB é um documento.

Banco de dados baseado em documentos

Segundo Marquesone(2017):

Bancos de dados orientados a documentos são ótimas soluções para armazenamento de registros conter todas as informações relevantes para uma consulta, sem necessitar da criação de joins com atributos variados. Além disso, esses bancos de dados oferecem grande escalabilidade e velocidade de leitura, pois os dados são armazenados em forma desnormalizada. Por esse motivo, um documento armazenado deve conter todas as informações relevantes para uma consulta, sem necessitar da criação de joins.

Banco de dados baseado em documentos

- Os documentos são normalmente modelados usando a formatação JSON e, em seguida, inseridos no banco de dados onde são convertidos em um formato binário para armazenamento.
- Relacionado à base de documentos para armazenamento, está o fato de que os documentos MongoDB não têm esquema fixo. O principal benefício disso é a sobrecarga amplamente reduzida.
- A reestruturação do banco de dados é fácil de ser aplicada e não causa os problemas massivos, travamentos de sites e violações de segurança observados em aplicativos que utilizam banco de dados relacionais.

Banco de dados baseado em documentos



```
1  {  
2    "_id": "5cf0029caff5056591b0ce7d",  
3    "firstname": "Jane",  
4    "lastname": "Wu",  
5    "adress": {  
6      "street": "1 Circle Rd",  
7      "city": "CA",  
8      "zip": "90404"  
9    }  
10   "hobbies": ["surfing", "coding"]  
11 }
```

Banco de dados baseado em documentos

- Outro recurso que faz com que o MongoDB se destaque de outras tecnologias de banco de dados é sua **capacidade de garantir alta disponibilidade** por meio de um processo conhecido como **replicação** ou replication do inglês.
- Um servidor que executa o MongoDB **pode ter cópias de seus bancos de dados duplicadas em mais dois servidores**. Essas cópias são conhecidas como conjuntos de réplicas.
- Outro recurso, entre muitos, é a capacidade do MongoDB de lidar com uma **grande quantidade de dados**.


Banco de dados baseado em documentos

- O modelo orientado a documentos define uma coleção de documentos, sendo cada documento acessado por uma chave única.
- Um documento é composto por atributos com conteúdo estruturado principalmente a partir do uso dos construtores JSON para objeto e array.
- O modelo orientado a documentos é mais adequado para a representação de objetos complexos.

Objeto complexo:

Um Objeto não é apenas uma instância de uma Classe. Ele pode conter instâncias (quase) infinitas de diferentes Classes combinadas.

Banco de dados baseado em documentos

MongoDB	Bancos de dados relacionais	Exemplo
Document	Linha	{ "nome" : "Guilherme Lima", "email" : "gui@alura.com" }
Field	Coluna	{ "nome" : "Guilherme Lima", "email" : "gui@alura.com" }
Collection	Tabela	

Portanto, em vez de colunas, os documentos MongoDB têm campos. Em vez de tabelas, existem coleções de documentos.

Por que usar MongoDB?

- Por ser um banco de dados NoSQL, o ideal é que ele seja utilizado sempre que os bancos tradicionais forem incapazes ou inviáveis para realizar algum tipo de serviço, por exemplo, quando precisamos inserir um grande volume de dados em velocidade muito rápida (cenário comum em aplicações web).
- Quando precisamos escalar uma aplicação, o uso de bancos de dados tradicionais aumenta a complexidade do sistema, obrigando a criar cada vez mais relacionamentos entre as tabelas, com isso perdendo desempenho e dificultando a manutenção.

Compatibilidade

MongoDB é compatível com diversas linguagens e frameworks, por exemplo:

- NodeJS;
- C# e .NET;
- Java;
- Perl;
- PHP e seus frameworks;
- Python;
- Ruby.

Por que usar MongoDB?

- Por não possuir 'esquema', o mongoDB garante um crescimento horizontal do banco de forma muito mais ágil, sendo perfeito para aplicações como:
 - E-Commerces que dependem de grande volume de dados (por exemplo, taxas, valores de produtos, endereços de clientes, métodos de pagamento, etc.);
 - Aplicações que usem NodeJS ou Redis no backend devido à facilidade de manutenção, altíssima performance e integração com Javascript;
 - Sistemas de gerenciamento de conteúdos baseadas em Big Data.

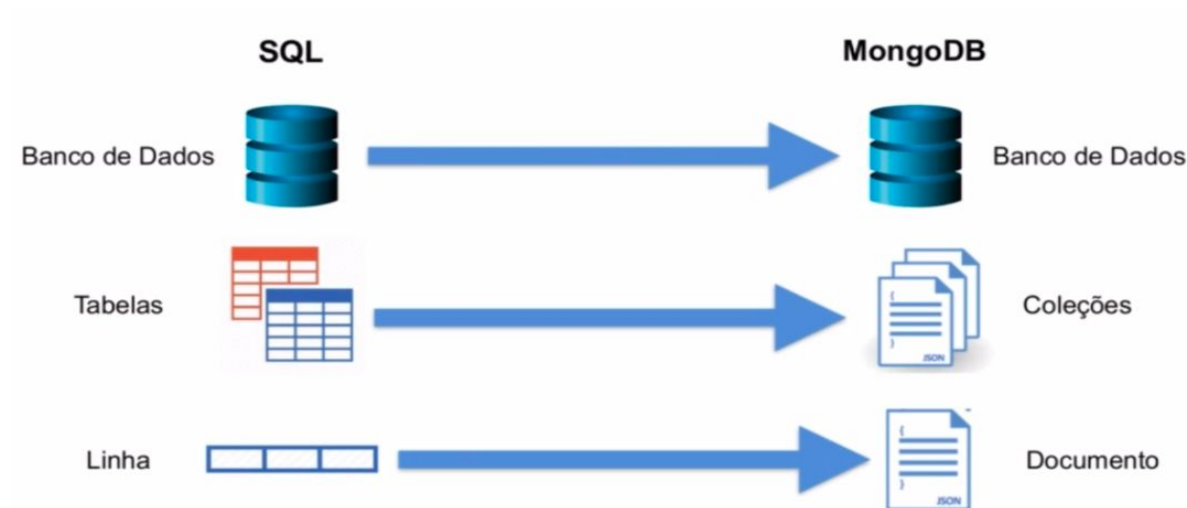
Comparando MongoDB com MySQL

SQL

- *Database* (Base de dados);
- Tabela;
- Linha;
- Índice;
- Coluna;
- União.

MongoDB

- *Database* (Base de dados);
- Coleção (*Collection*);
- Documento;
- Índice;
- Campo (*Field*);
- *Link* e Incorporação.

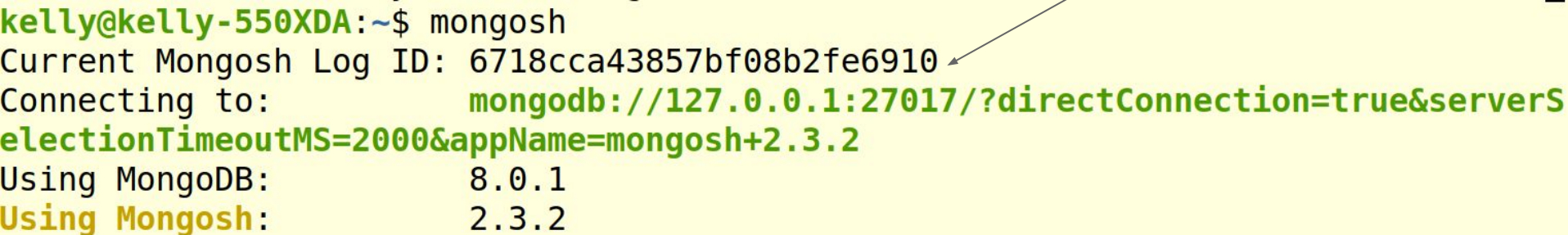


Comparando MongoDB com MySQL

NoSQL	SQL
MongoDB, Redis, CouchDB, Firebase, etc	MySQL, PostgreSQL, SQLite, etc
Não nos força a utilizar um esquema bem definido de dados.	Nos força a utilizar um esquema bem definido de dados.
Não é focado em relações, apesar de permiti-las.	Os banco de dados relacionais tem como característica principal as relações entre tabelas.
Os documentos são independentes.	Os registros são relacionados.
Excelente para grandes quantidades de dados.	Excelentes para trabalhar com aplicações convencionais.

Iniciando com MongoDB

```
kelly@kelly-550XDA:~$ mongosh
Current Mongosh Log ID: 6718cca43857bf08b2fe6910
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverS
electionTimeoutMS=2000&appName=mongosh+2.3.2
Using MongoDB:      8.0.1
Using Mongosh:      2.3.2
```



Veja a porta que está conectado o
Mongo shell
Padrão é 27017

E agora para usar o BD?

Digite o comando: **use meubanco**

Veja que ele deu a mensagem: switched to db meubanco

```
test> use meubanco  
switched to db meubanco
```

Ou seja, mudamos para o banco de dados **meubanco**

Se esse banco de dados existir, você faz acesso a ele, caso não, nós acabamos de criar ele.

Se eu quero descobrir qual banco estamos conectados:

```
meubanco> db  
meubanco
```

Solicitar ajuda - comando help

```
meubanco> help
```

Shell Help:

`use`
`show`
available databases.

all collections for current database.

.

nt database.

nt database.

type is not set uses 'global'

`exit`
`quit`
`Mongo`
ct. Usage: `new Mongo(URI, options [optional])`
`connect`

bject. Usage: `connect(URI, username [optional], password [optional])`

`it`

Set current database

'show databases'/'show dbs': Print a list of all

'show collections'/'show tables': Print a list of

'show profile': Prints system.profile information

'show users': Print a list of all users for curre

'show roles': Print a list of all roles for curre

'show log <type>': log for current connection, if

'show logs': Print all logs.

Quit the MongoDB shell with `exit`/`exit()`/`.exit`

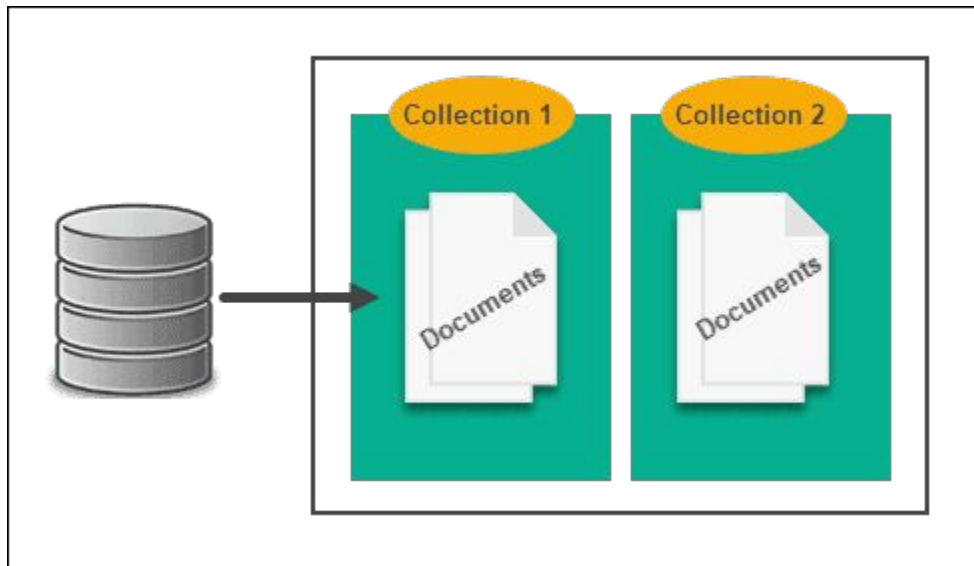
Quit the MongoDB shell with `quit`/`quit()`

Create a new connection and return the Mongo obje

Create a new connection and return the Database o

result of the last line evaluated; use to further

Coleções no MongoDB



Uma coleção no MongoDB já começa quando inserimos o primeiro dado nela.

Coleções no MongoDB

- No SQL tínhamos o conceito de tabelas. Aqui temos coleções.
- Collection é um grupo de documentos MongoDB.
- Uma coleção existe dentro de um único banco de dados.
- Coleções não impõem um esquema.
- Documentos dentro de uma coleção pode ter diferentes campos.
- Normalmente, todos os documentos em uma coleção são propositalmente semelhantes ou afins.

Uma coleção no MongoDB já começa quando inserimos o primeiro dado nela.

Criando coleções

- Criar uma coleção nomeada Pessoas e inserir dados
- **db.<nomeColeção>.insertOne ()**

```
meubanco> db.pessoas.insertOne(  
... {  
...   "nome": "Fulano de Tal",  
...   "email": "fulano@email.com",  
... }  
... )  
{  
  acknowledged: true,  
  insertedId: ObjectId('6718d1843857bf08b2fe6913')  
}
```

Inserimos um documento!
Um documento nada mais é do que uma linha inserida no banco

Inserindo dados

Se for comparar com o relacional, é como inserir um registro! Aqui é documento inserido dentro de uma coleção

- `db.<nomeColeção>.insertMany ()`

```
meubanco> db.pessoas.insertMany(  
... [  
... { "nome": "Beltrano", "email": "beltrano@email.com" },  
... { "nome": "Ciclano", "email": "ciclano@gmail.com" }  
... ]  
... )  
{  
  acknowledged: true,  
  insertedIds: {  
    '0': ObjectId('6718d2523857bf08b2fe6914'),  
    '1': ObjectId('6718d2523857bf08b2fe6915')  
  }  
}
```


E para verificar se o documento foi inserido?

```
db.pessoas.find()
```

```
meubanco> show collections
pessoas
```

```
meubanco> db.pessoas.find()
[
  {
    _id: ObjectId('6718d0ca3857bf08b2fe6911'),
    nome: 'Kelly Lais',
    email: 'kwiggers@utfpr.edu.br'
  },
  {
    _id: ObjectId('6718d1373857bf08b2fe6912'),
    nome: 'Kelly Lais',
    email: 'kwiggers@utfpr.edu.br'
  },
  {
    _id: ObjectId('6718d1843857bf08b2fe6913'),
    nome: 'Fulano de Tal',
    email: 'fulano@email.com'
  },
  {
    _id: ObjectId('6718d2523857bf08b2fe6914'),
    nome: 'Beltrano',
    email: 'beltrano@email.com'
  },
  {
    _id: ObjectId('6718d2523857bf08b2fe6915'),
    nome: 'Ciclano',
    email: 'ciclano@gmail.com'
  }
]
```

E verificando bases de dados

- Antes de criar a coleção pessoas dentro do meubanco

```
meubanco> show dbs
admin      40.00 KiB
config     72.00 KiB
local      72.00 KiB
```

- Depois de criar a coleção pessoas dentro do meubanco

```
meubanco> show dbs
admin      40.00 KiB
config     96.00 KiB
local      72.00 KiB
meubanco   72.00 KiB
```

Visualizando no Mongodb Compass

The screenshot displays the MongoDB Compass web interface. On the left sidebar, the 'CONNECTIONS (1)' section shows a list of connections: 'localhost:27017' with sub-items 'admin', 'config', 'local', and 'meubanco'. The 'meubanco' connection is expanded, showing a collection named 'pessoas' which is highlighted with a blue border.

The main panel shows the 'pessoas' collection selected. The breadcrumb path is 'localhost:27017 > meubanco > pessoas'. A button 'Open MongoDB shell' is in the top right. Below the breadcrumb, there are tabs for 'Documents' (5), 'Aggregations', 'Schema', 'Indexes' (1), and 'Validation'. The 'Documents' tab is active.

A query input field contains the text 'Type a query: { field: 'value' } or [Generate query](#)'. To the right of the input are buttons for 'Explain', 'Reset', 'Find', and a code icon, followed by an 'Options' link.

Below the query field is a row of action buttons: 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. To the right of these buttons is a pagination control showing '25' items per page and '1 - 5 of 5' documents.

The document list displays five entries, each with its JSON representation:

- `{ "_id": ObjectId('6718d0ca3857bf08b2fe6911'), "nome": "Kelly Lais", "email": "kwiggers@utfpr.edu.br" }`
- `{ "_id": ObjectId('6718d1373857bf08b2fe6912'), "nome": "Kelly Lais", "email": "kwiggers@utfpr.edu.br" }`
- `{ "_id": ObjectId('6718d1843857bf08b2fe6913'), "nome": "Fulano de Tal", "email": "fulano@email.com" }`
- `{ "_id": ObjectId('6718d2523857bf08b2fe6914'), "nome": "Beltrano", "email": "beltrano@email.com" }`
- `{ "_id": ObjectId('6718d2523857bf08b2fe6915'), "nome": "Ciclano", "email": "ciclano@gmail.com" }`

Vantagem BD não relacional

Vamos inserir mais uma pessoa, mas agora ela também possui o atributo “idade”.

```
meubanco> db.pessoas.insertOne( { "nome": "Bill Gates", "email": "bill@email.com", "idade": "68" } )
{
  acknowledged: true,
  insertedId: ObjectId('671a47b504895640f3fe6911')
}
```

Se fosse em um BD relacional, isso daria erro.

Vantagem BD não relacional

```
{
  _id: ObjectId('6718d1843857bf08b2fe6913'),
  nome: 'Fulano de Tal',
  email: 'fulano@email.com'
},
{
  _id: ObjectId('6718d2523857bf08b2fe6914'),
  nome: 'Beltrano',
  email: 'beltrano@email.com'
},
{
  _id: ObjectId('6718d2523857bf08b2fe6915'),
  nome: 'Ciclano',
  email: 'ciclano@gmail.com'
},
{
  _id: ObjectId('671a47b504895640f3fe6911'),
  nome: 'Bill Gates',
  email: 'bill@email.com',
  idade: '68'
}
```

Vantagem BD não relacional

- **Modelo de dados flexível:** Permite armazenar dados heterogêneos em documentos BSON, sem a necessidade de um esquema fixo.
- **Escalabilidade horizontal:** Pode escalar facilmente distribuindo dados em vários servidores.
- **Alta performance:** Oferece operações de leitura e gravação rápidas, otimizadas para grandes volumes de dados.
- **Replicação automática:** Fornece alta disponibilidade e tolerância a falhas, com replicação automática de dados em vários servidores.
- **Consultas poderosas:** Suporta consultas complexas e operações de agregação, facilitando a extração de informações dos dados. Índices eficientes: Permite criar índices para melhorar o desempenho de consultas.
- **Transações:** Suporta transações para garantir a integridade dos dados em operações de leitura e gravação.

Dúvidas?

kwiggers@utfpr.edu.br