

Introdução a Banco de Dados Não-Relacional

Os bancos de dados **NoSQL** (ou *não-relacionais*) utilizam um padrão diferente de armazenamento em relação ao **SQL**. O grande diferencial dessa tecnologia é a capacidade de escalabilidade para as operações das empresas de uma forma mais simples e econômica do que no banco relacional.

O **NoSQL** também proporciona uma performance melhor para o gerenciamento de dados das organizações, pois não há necessidade de agrupar os dados em um esquema de tabelas para usar as informações.

Tipos de Banco de Dados Não-Relacionais

Principais bancos de dados não-relacionais utilizados no mercado:

- **REDIS**

O **Redis** é um armazenamento de dados **NoSQL** de chave-valor. Ele vincula um valor a uma chave na sua estrutura, o que facilita o armazenamento e a busca desses dados. Por isso, é muito utilizado pelos desenvolvedores.

- **CASSANDRA**

O **Cassandra** foi desenvolvido no Facebook. Ele usa um banco de dados descentralizado, em que os dados são armazenados em vários datacenters. Ele é otimizado para cluster e fornece baixa latência em suas atualizações.

- **HBASE**

O **Hbase** é um banco de dados que utiliza conjunto de linhas e colunas para armazenar as informações. Ele é utilizado em diferentes plataformas como o LinkedIn, Facebook e Spotify.

- **AMAZON DYNAMODB**

O **AWS DynamoDB** é um banco de dados NoSQL em nuvem, disponibilizado pela Amazon Web Service. Ele tem baixa latência, é rápido e flexível, sendo o modelo ideal para aplicações móveis, jogos na web e outras soluções.

Ele ainda apresenta alto desempenho e escalabilidade automática, características imprescindíveis para negócios que precisam crescer com eficiência.

- **NEO4J**

O [Neo4j](#) é um banco de dados não-relacional que se baseia em grafos (*arestas que se relacionam aos nodes*). Ele é uma implementação de código aberto e pode ser útil para casos de mineração de dados e reconhecimento de padrões.

- **MONGODB**

O [MongoDB](#) também é um banco de dados de código aberto com alta performance. Ele é aceito em diferentes sistemas operacionais e tem como característica ser orientado a documentos.

Sendo assim, ele armazena todas as informações relevantes em um documento e utiliza sistemas avançados de agrupamento e filtragem. Diferentes plataformas e linguagens possuem suporte ao **MongoDB**, entre elas estão o Java, JavaScript, PHP, Python e Ruby.

Esses são os principais exemplos de bancos de dados NoSQL ou não-relacionais. O uso entre eles pode se diferenciar de acordo com as necessidades de cada negócio.

Instalação local MongoDB

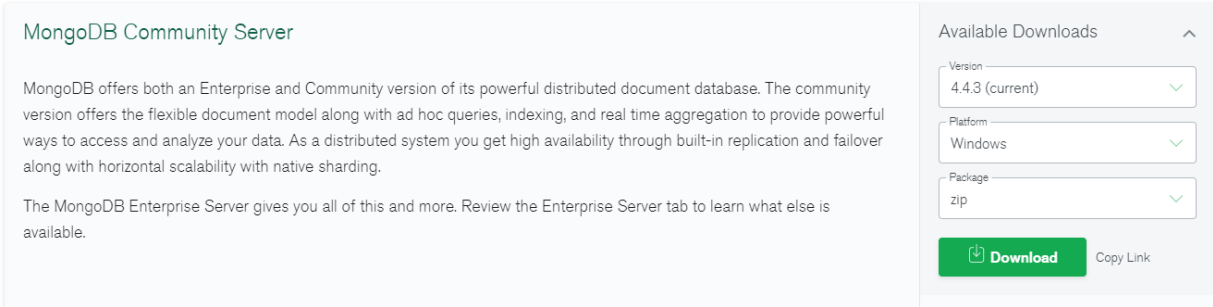
MongoDB não é só um dos bancos **NoSQL** mais populares, mas de modo geral, incluindo os bancos relacionais, um dos bancos mais usados da atualidade.

Nele, os dados são representados em [JSONs](#) : formato largamente usado em APIs, logs, arquivos de configuração, etc (*mais claro e menos verboso do que XML*). No entanto, eles são armazenados como “**BSOns**” ou “**JSONs binários**” : os [JSON](#) são gravados de forma binária, o que otimiza o espaço ocupado e é mais veloz para consultas. Na “*superfície*”, de fato, o que o usuário vê são [JSONs](#) .

Download MongoDB

É necessário realizar [download MongoDB](#). É sempre bom realizar o download da ultima versão gratuita disponível no site.

Podemos utilizar **Package MSI** (para Windows) e **Package ZIP** (para linux).



MongoDB Community Server

MongoDB offers both an Enterprise and Community version of its powerful distributed document database. The community version offers the flexible document model along with ad hoc queries, indexing, and real time aggregation to provide powerful ways to access and analyze your data. As a distributed system you get high availability through built-in replication and failover along with horizontal scalability with native sharding.

The MongoDB Enterprise Server gives you all of this and more. Review the Enterprise Server tab to learn what else is available.

Available Downloads

Version: 4.4.3 (current) ✓

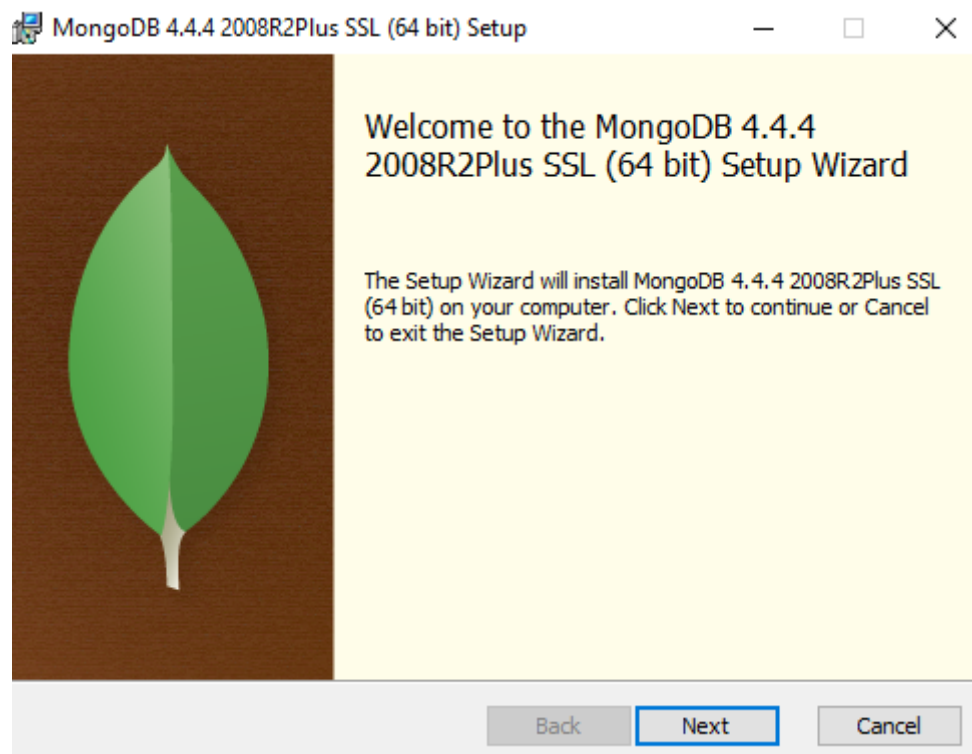
Platform: Windows ✓

Package: zip ✓

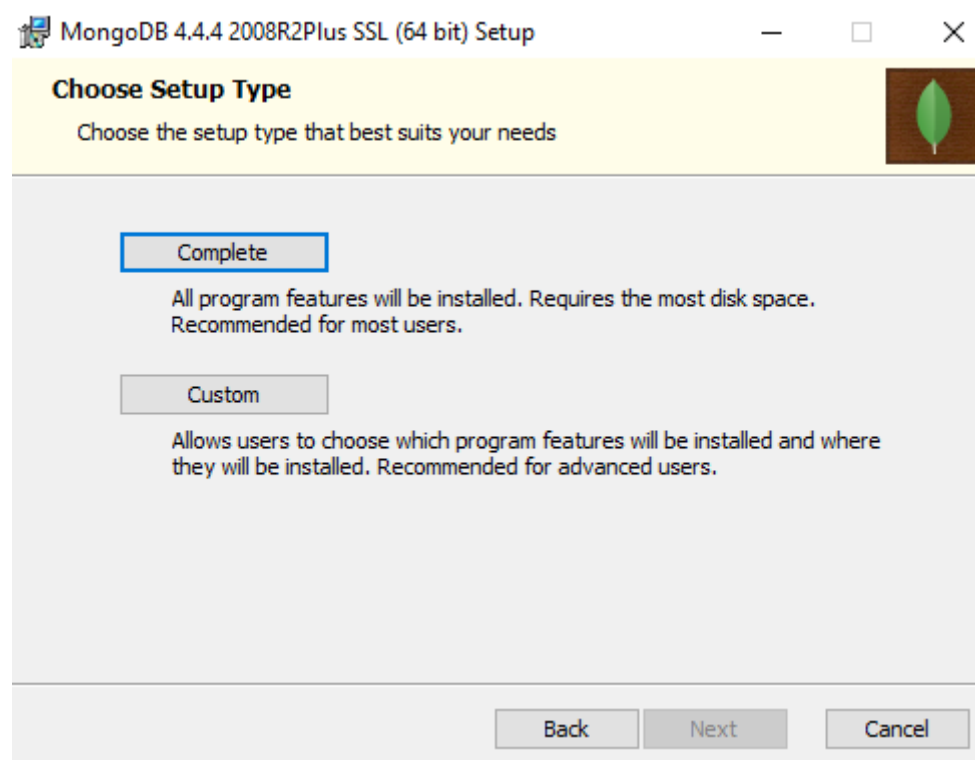
[Download](#) [Copy Link](#)

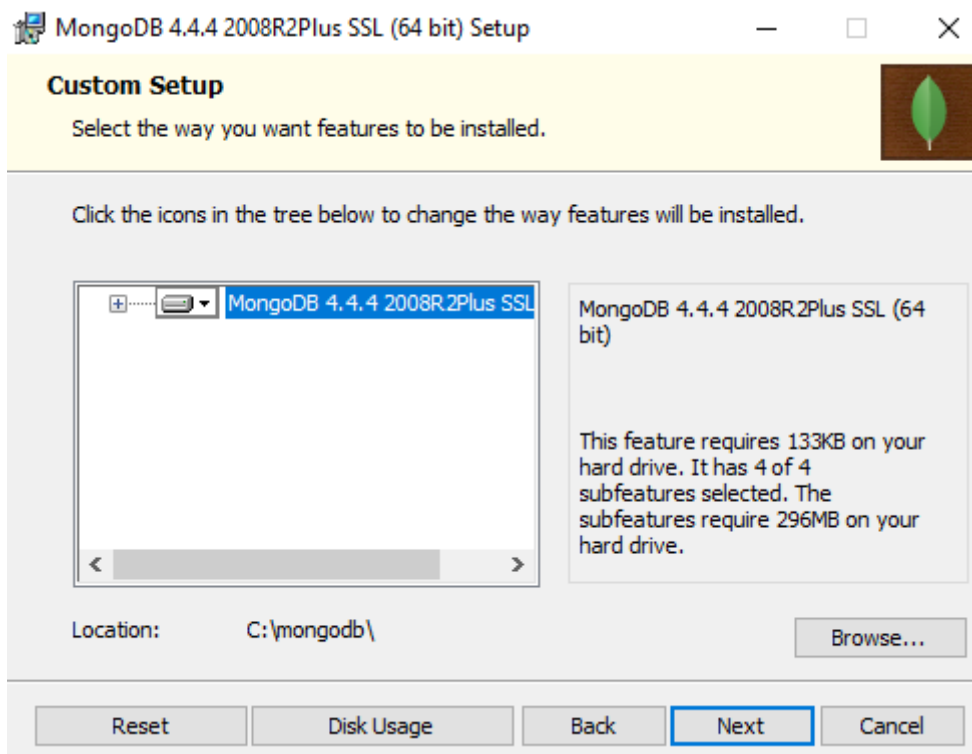
Instalação MongoDB

A instalação é simples, apenas next... next... next.

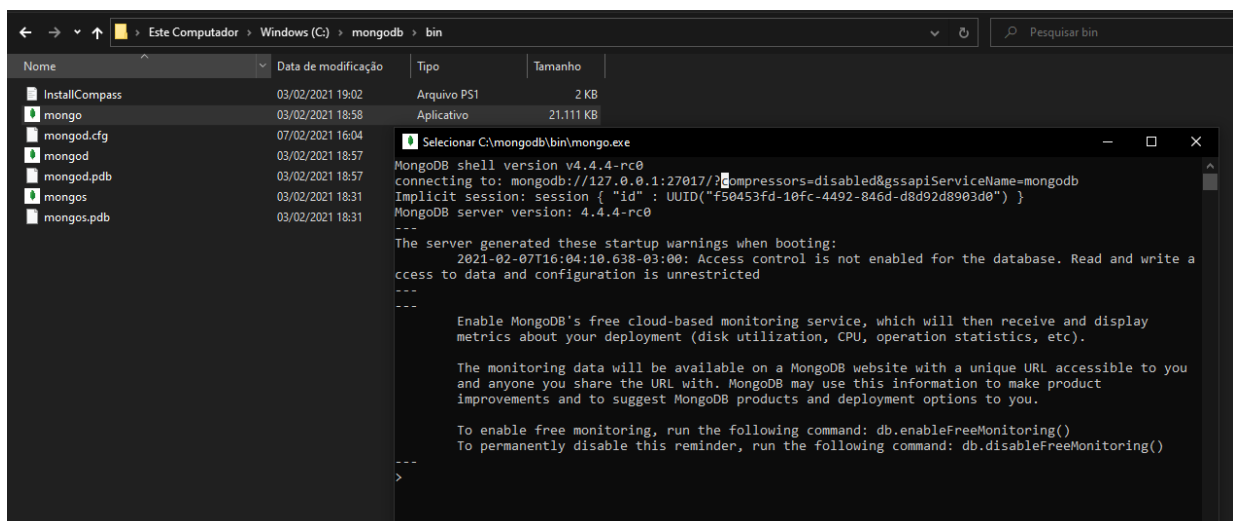


Em **Custom**, altere a pasta de instalação para "*C:\mongodb*".





Para abrir o terminal do **MongoDB** clique duas vezes no arquivo *mongo*, localizado no diretório “C:\mongodb\bin”.



Consultas

No MongoDB, conjuntos de documentos são chamados de “**Coleções**” (**collections**). Seriam os análogos das tabelas no Postgres - com a diferença fundamental de que seus documentos não precisam respeitar um schema rígido de tabela, como nos bancos relacionais.

Como no caso dos bancos relacionais, essas coleções ficam salvas dentro de “**databases**” no mesmo banco.

Criando uma collection

Para criar uma collection, fazemos no shell:

```
db.createCollection('customers')
```

Para deletar uma:

```
db.dropDatabase()
```

Inserindo documentos

Para inserir um documento na nossa **collection**, fazemos:

```
db.customers.insertOne(  
  {  
    nome: "Carlos Alberto",  
    idade: 27  
  }  
)
```

Ao fazer isto, um *id randômico* automaticamente é criado:

```
"insertedId" : ObjectId("602471311f028450f0839cdd")
```

Nada nos impede de inserir um documentos com outro formato na mesma **collection**:

```
db.customers.insertOne(  
  {  
    nome: "Maria",  
    idade: 30,  
    empresas: ['Facebook', 'Google']  
  }  
)
```

Podemos **inserir múltiplos** itens simultaneamente:

```
db.customers.insertMany([  
  {nome: "Maria Clara"},  
  {nome: "Pedro", idade: 21},  
  {nome: "Joao", empresas: ['Apple', 'Samsung']}]  
)
```

Atualizando documentos

Para **atualizar** um documento (*o primeiro apenas!*):

```
db.customers.updateOne(  
  {nome: "Maria"},  
  {$set: {"idade": 31}}  
)
```

O código acima atualiza a idade da **primeira "Maria"** que encontra para **31**. Se quisermos um comportamento com o qual estamos mais acostumados no Postgres, onde atualizamos **todos os itens que se encaixam na busca**, fazemos:

```
db.customers.updateMany(  
  {nome: "Maria"},  
  {$set: {"idade": 31}}  
)
```

Deletando documentos

Como nos outros casos, há o **"One"** e o **"Many"**. Abaixo, **deletamos o primeiro registro cujo nome é "Maria Clara"**

```
db.customers.deleteOne({nome: "Maria Clara"})
```

Para **deletar todos os registros cujos nomes são "Maria Clara"**, fazemos:

```
db.customers.deleteMany({nome: "Maria Clara"})
```

O famoso **"delete sem WHERE"** é:

```
db.customers.deleteMany({})
```

Consultas básicas

Para **contar todos os elementos numa tabela**:

```
db.customers.count()
```

Para **filtrar**:

```
db.customers.count({nome: 'Pedro'})
```

Para **consultar uma coleção**, no equivalente ao SQL para **SELECT * FROM customers**, basta fazer:

```
db.customers.find()
```

As condições para a consulta vão como argumento do **find()**, da mesma forma que com o **count()**:

```
db.customers.find( { <condições> } )
```

Ex.:

```
db.customers.find(  
  {  
    $or: [{nome: 'Pedro'}, {idade: 31}]  
  }  
)
```

Importando CSVs para dentro do MongoDB

Este é o equivalente da operação de **COPY do Postgres**. Para tanto, abra o terminal do Windows e rode:

```
mongoimport -d <nome-database> -c <nome-collection> --type csv --file C:\  
<endereco>\<nome-arquivo>.csv --headerline
```

Referências

[mongo Shell Methods](#)

[Create a Database in MongoDB](#)

[Download do Robo 3T](#)

[JSON and BSON](#)

[MongoDB in 5 Minutes](#)

[Importando CSV no MongoDB pelo Windows](#)

[Consultando documentos no MongoDB](#)

[< Tópico anterior](#)

