

# ***Big Data***



Cruzeiro do Sul Virtual  
Educação a distância





## Banco de Dados Não Estruturado

**Responsável pelo Conteúdo:**

Prof. Dr. Cleber Silva Ferreira da Luz

**Revisão Textual:**

Prof.<sup>a</sup> Dra. Selma Aparecida Cesarin



# UNIDADE

## Banco de Dados Não Estruturado



- Banco de Dados Não Estruturado



### OBJETIVOS DE APRENDIZADO

- Compreender o princípio de Banco de Dados não Estruturados;
- Aprender conceitos introdutórios e avançados sobre Banco de Dados não Estruturados;
- Compreender o funcionamento do Banco mongodb.





# Orientações de estudo

Para que o conteúdo desta Disciplina seja bem aproveitado e haja maior aplicabilidade na sua formação acadêmica e atuação profissional, siga algumas recomendações básicas:



## Assim:

- ✓ Organize seus estudos de maneira que passem a fazer parte da sua rotina. Por exemplo, você poderá determinar um dia e horário fixos como seu "momento do estudo";
- ✓ Procure se alimentar e se hidratar quando for estudar; lembre-se de que uma alimentação saudável pode proporcionar melhor aproveitamento do estudo;
- ✓ No material de cada Unidade, há leituras indicadas e, entre elas, artigos científicos, livros, vídeos e sites para aprofundar os conhecimentos adquiridos ao longo da Unidade. Além disso, você também encontrará sugestões de conteúdo extra no item **Material Complementar**, que ampliarão sua interpretação e auxiliarão no pleno entendimento dos temas abordados;
- ✓ Após o contato com o conteúdo proposto, participe dos debates mediados em fóruns de discussão, pois irão auxiliar a verificar o quanto você absorveu de conhecimento, além de propiciar o contato com seus colegas e tutores, o que se apresenta como rico espaço de troca de ideias e de aprendizagem.

# Introdução

*Big Data* é um termo que se refere a um grande volume de dados, tanto estruturados quanto não estruturados, que inundam uma Organização em uma base diária. Mas não é a quantidade de dados que é importante. O importante é o que as Organizações fazem com os dados.

A análise de *Big Data* permite que as Organizações obtenham *insights* valiosos sobre seus negócios, clientes e operações. Com a análise de *Big Data*, as Organizações podem identificar padrões, tendências e correlações que seriam difíceis ou impossíveis de detectar de outra forma.

As Empresas podem usar *Big Data* para uma variedade de fins, incluindo:

- **Melhorar a tomada de decisão:** ao analisar grandes volumes de dados, as Empresas podem tomar decisões mais informadas e embasadas em dados. Identificar novas oportunidades de negócios: a análise de *Big Data* pode ajudar as Empresas a identificar novos Mercados, produtos ou serviços que possam ser lucrativos;
- **Personalização:** a análise de *Big Data* pode ajudar as Empresas a personalizar sua comunicação e *marketing* para atender às necessidades específicas dos clientes;
- **Prevenção de fraudes:** a análise de *Big Data* pode ajudar as Empresas a identificar e a prevenir fraudes, reduzindo, assim, os riscos e as perdas financeiras.

No entanto, o processamento de grandes volumes de dados pode ser um desafio técnico. As Empresas precisam investir em infraestrutura de *hardware* e *software* para gerenciar grandes volumes de dados. Além disso, as Empresas precisam de pessoal qualificado para realizar análises de *Big Data* e tomar decisões informadas com base nos *insights* obtidos.

Em resumo, *Big Data* oferece oportunidades significativas para as Empresas que desejam melhorar a tomada de decisões, identificar novas oportunidades de negócios, personalizar sua comunicação e *marketing* e prevenir fraudes.

Tendo em vista o tamanho da importância do *Big Data*, nesta Unidade de ensino, iremos estudar conceitos de Banco de Dados não Estruturados, mais especificamente, iremos estudar um SGBD (Sistema de Gerenciamento de Banco de Dados) para Bancos não Estruturados, o MongoDB.

# Banco de Dados Não Estruturado

*Big Data* é um termo que está relacionado à análise de grandes quantidades de dados para obtenção de informações valiosas. O objetivo da análise de *Big Data* é identificar padrões, tendências e *insights* que possam ser usados para melhorar os processos e a tomada de decisão das Empresas. Essa análise pode ser feita a partir de diferentes fontes de dados, incluindo Redes Sociais, dispositivos móveis, sensores e transações financeiras, entre outros.

O uso de *Big Data* permite que as Empresas personalizem suas operações, reduzam custos e aumentem a eficiência dos seus processos. No entanto, a análise de *Big Data* requer conhecimentos avançados em tecnologia, estatística e programação, além de investimentos em infraestrutura e pessoal qualificado.

No *Big Data*, precisamos usar tecnologias para manipular os dados de forma eficiente. Atualmente, Aplicações de *Big Data* trabalham em conjunto com NoSQL.

NoSQL é uma abreviação de “*Not Only SQL*” e é uma alternativa ao modelo relacional de Bancos de Dados. Enquanto o modelo relacional utiliza tabelas para armazenar informações, o NoSQL utiliza estruturas diferentes, como documentos, gráficos e colunas.

Uma das principais vantagens do NoSQL é a escalabilidade horizontal, o que significa que é possível adicionar mais servidores ao Banco de Dados para aumentar a capacidade de armazenamento e processamento. Isso é particularmente importante em aplicações que precisam lidar com grandes quantidades de dados e demandas de acesso simultâneo.

Outra vantagem é a flexibilidade no esquema de dados. Enquanto no modelo relacional é necessário definir previamente a estrutura das tabelas e suas relações, no NoSQL, é possível adicionar campos e documentos sem precisar fazer alterações no esquema do Banco de Dados.

É importante ressaltar que NoSQL e Banco de Dados não Estrutural são termos equivalentes. Continuemos a estudar um pouco mais sobre Banco de Dados não Estrutural.

Os Bancos de Dados não Estruturais são uma alternativa para armazenar grandes quantidades de dados que não se enquadram em modelos de Bancos de Dados relacionais tradicionais. Esse tipo de Banco de Dados é projetado para lidar com dados não estruturados, como arquivos de texto, imagens, vídeos e outros tipos de dados que não podem ser organizados em tabelas.

Ao contrário dos Bancos de Dados relacionais, que têm um esquema definido que determina como os dados são armazenados e organizados, os Bancos de Dados não Estruturais são mais flexíveis e adaptáveis. Eles permitem que os dados sejam armazenados em sua forma original, sem a necessidade de uma estrutura pré-definida.

Os Bancos de Dados não estruturais podem ser usados para uma ampla gama de aplicações, incluindo análise de Mídia Social, gerenciamento de conteúdo, armazenamento de dados de sensores e análise de logs de servidor, entre outros.

Ainda sobre as vantagens de Banco de Dados não Estrutural, podemos citar a escalabilidade. Como esses Bancos de Dados não têm um esquema fixo, eles podem facilmente lidar com grandes quantidades de dados sem precisar de muita manutenção.

No entanto, os Bancos de Dados não Estruturais também têm algumas desvantagens. Uma das maiores é que podem ser mais difíceis de consultar e analisar do que os Bancos de Dados Relacionais. Além disso, eles podem exigir mais recursos de hardware e software para gerenciamento e manutenção.

Dessa forma, os Bancos de Dados não Estruturais são uma opção atraente para Empresas que precisam armazenar e gerenciar grandes quantidades de dados não estru-

turados. Embora apresentem alguns desafios em termos de consulta e manutenção, sua escalabilidade e flexibilidade tornam essa opção muito interessante em um mundo cada vez mais dependente de dados.

Atualmente, o Banco de Dados MongodB é uma boa opção para trabalhar com Banco de Dados não Estruturados. MongoDB é um Sistema de Gerenciamento de Banco de Dados NoSQL, criado em 2007, pela Empresa MongoDB Inc. Ele difere de Bancos de Dados Relacionais tradicionais, pois não usa tabelas, linhas e colunas, mas sim documentos e coleções.

Um documento no MongoDB é uma estrutura de dados semelhante a um *JSON (JavaScript Object Notation)*, que pode conter campos e valores. Esses documentos são armazenados em coleções, que podem ser comparadas a tabelas em um Banco de Dados Relacional. No entanto, as coleções no MongoDB são muito mais flexíveis e dinâmicas, permitindo que os desenvolvedores adicionem novos campos e estruturas sem a necessidade de atualizar esquemas ou fazer migrações complexas.

Outra característica importante do MongoDB é sua capacidade de escalar horizontalmente, o que significa que ele pode lidar com grandes quantidades de dados e tráfego de usuários adicionando mais servidores ao *cluster*. Isso torna o MongoDB uma ótima escolha para aplicativos de grande escala e Empresas que precisam lidar com grandes volumes de dados.

Além disso, o MongoDB tem uma ampla comunidade de desenvolvedores, o que significa que há uma grande quantidade de recursos, tutoriais e documentação disponíveis para os desenvolvedores. Ele também suporta várias Linguagens de Programação, como Java, Python e Node.js.

Dessa forma, o MongoDB é uma opção poderosa e flexível para Empresas e desenvolvedores que precisam lidar com grandes quantidades de dados e querem aproveitar as vantagens do modelo NoSQL. Com sua escalabilidade horizontal e documentação fácil de usar, ele é uma escolha popular para muitos projetos de software modernos.



Para instalar o MongoDB, acesse o site: <https://www.mongodb.com/>. Para baixar o MongoDB, vá a produtos e selecione *Community Server*. Posteriormente, selecione o arquivo para *Download* conforme o seu sistema operacional.

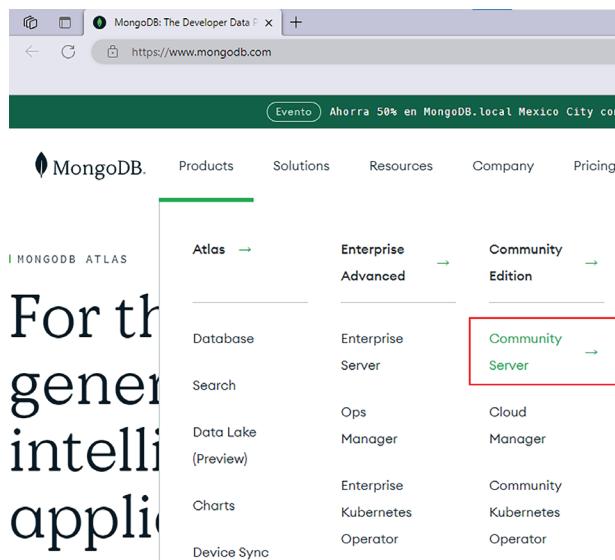


Figura 1 – Tela de *Download* do MongoDB

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo branco com letras verdes mostrando o caminho que deve ser seguido para encontrar o arquivo do MongoDB para *download*. Fim da descrição.

Após baixar o arquivo, devemos instalá-lo. A instalação é “padrão”. É só clicar em “Next” até o final.

Após finalizar a instalação, iremos voltar para o site do MongoDB e baixar para o “shell”. Para isso, vá novamente a Produtos, ferramentas e clique em *Shell*.

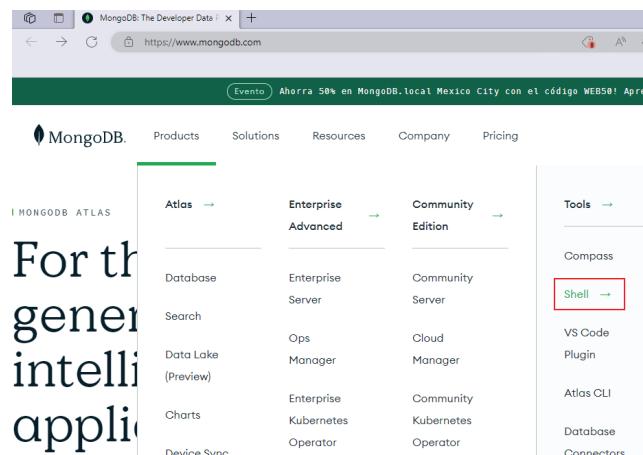


Figura 2 – Tela de *Download* do Shell

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo branco com letras verdes mostrando o caminho que deve ser seguido para encontrar o arquivo do *Shell* para *download*. Fim da descrição.

Após baixar o arquivo, você terá de extrair a pasta. Abra a pasta e vá até a pasta *bin*. Nela, abra o mongosh.

O Mongosh é um *shell* para o MongoDB. Após abrir, coloque o seguinte comando: `mongodb://localhost:27017`. Nesse comando, irá abrir o MongoDB.

```
Please enter a MongoDB connection string (Default: mongodb://localhost/): mongo  
db://localhost:27017
```

Figura 3 – Tela inicial do MongoDB inicial

Fonte: Reprodução

#ParaTodosVerem: foto com fundo preto com letras brancas e com o caminho para abrir o servidor do MongoDB. Fim da descrição.

Com o servidor do MongoDB aberto, a primeira coisa é criar um Banco de Dados. Para criá-lo, usamos o seguinte comando:

```
use xpto
```

O comando use cria um Banco de Dados e já diz para o servidor que agora todos os comandos executados serão executados nesse Banco.

```
test> use xpto  
switched to db xpto  
xpto>
```

Figura 4 – Comando para cria um banco no MongoDB

Fonte: Reprodução

#ParaTodosVerem: foto com fundo preto com letras brancas e com o comando para criar um Banco de Dados no MongoDB. Fim da descrição.

Perceba que o nome do Banco de Dados ficou à esquerda.

Agora, a missão é criar uma *collection*. Uma *collection* é semelhante a uma tabela no Banco de Dados RELACIONAL. Nela, conseguimos armazenar os dados.

O comando para criar uma *collection* é:

```
db.createCollection()
```

Sempre que formos executar um comando, devemos colocar a palavra db. Essa palavra vai falar para o servidor que o comando a seguir será executado no Banco de Dados em questão.

Vamos criar uma *collection* chamada funcionário.

```
xpto> db.createCollection('Funcionarios')  
{ ok: 1 }  
xpto>
```

Figura 5 – Comando para criar uma *collection*

Fonte: Reprodução

#ParaTodosVerem: foto com fundo preto com letras brancas e com o comando para criar uma collection. Fim da descrição.

O comando:

```
show dbs
```

lista todos os bancos criados no servidor.

E o comando:

```
show collections
```

lista todas as *collections* criadas em um Banco de Dados específico.

Agora, iremos inserir dados na *collection* Funcionarios. Para isso, usaremos o comando `insertOne()`, que insere apenas 1 documento por vez:

```
db.funcionarios.insertOne()
```

```
xpto> db.funcionarios.insertOne({nome: "cleber", idade: 17, profissao: "programador"})
{
  acknowledged: true,
  insertedId: ObjectId("645e9f0d8858efdf68bd138b")
}
xpto>
```

Figura 6 – Comando para inserir dados em uma *collection*

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo preto com letras brancas e com o comando para inserir documento. Fim da descrição.

A Figura 6 insere 1 documento na *collection* funcionario. Perceba que após de executar o comando, abaixo, aparece uma mensagem. Essa mensagem diz que foi inserido com sucesso, e também aparece o *id* do documento.

Podemos inserir, também, diversos documentos de uma vez só, usando o comando:

```
insertMany()
```

```
xpto> db.funcionarios.insertMany([ {nome: "Joao", idade: 18, profissao: "Dev"}, 
{nome: "Maria", idade:19, profissao: "dev2"} ])
{
  acknowledged: true,
  insertedIds: [
    '0': ObjectId("645ea1f98858efdf68bd138c"),
    '1': ObjectId("645ea1f98858efdf68bd138d")
  ]
}
xpto> -
```

Figura 7 – Comando para inserir mais de 1 documento em uma *collection*

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo preto com letras brancas e com o comando para inserir mais de um documento em uma collection. Fim da descrição.

Na Figura 7, inserimos 2 documentos de uma vez só.

Um comando interessante e muito usado é o comando:

```
find()
```

Esse comando é responsável por selecionar e retornar um documento em uma *collection*.

```
test> db.funcionarios.find()
[  
  {  
    _id: ObjectId("645ff3ab6af7cd5f72152fd1"),  
    nome: 'cleben',  
    idade: 17,  
    profissao: 'programador'  
  },  
  {  
    _id: ObjectId("645ff3b36af7cd5f72152fd2"),  
    nome: 'Joao',  
    idade: 18,  
    profissao: 'Dev'  
  },  
  {  
    _id: ObjectId("645ff3b36af7cd5f72152fd3"),  
    nome: 'Maria',  
    idade: 19,  
    profissao: 'dev2'
```

Figura 8 – Comando para selecionar documentos

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo preto, com letras brancas e com o comando para selecionar mais de um documento em uma *collection*. Fim da descrição.

Se você desejar, pode selecionar apenas um documento específico. Para isso, coloque o campo que você selecionar.

Por exemplo:

```
db.funcionarios.find({idade:18})
```

No comando acima, estamos filtrando todos os funcionários com idade exata de 18 anos.

```
test> db.funcionarios.find({idade:18})  
[  
  {  
    _id: ObjectId("645ff3b36af7cd5f72152fd2"),  
    nome: 'Joao',  
    idade: 18,  
    profissao: 'Dev'  
  }  
]
```

Figura 9 – Comando para selecionar um documento específico

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo preto com letras brancas e com o comando para selecionar um documento específico em uma *collection*. Fim da descrição.

Agora, vamos supor que você queira atualizar um dado em um documento. Para isso, você pode utilizar o comando:

```
db.funcionarios.update()
```

```
test> db.funcionarios.update({idade:18} , {$set:{nome:'Paulo'}})  
DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 1,  
  modifiedCount: 1,  
  upsertedCount: 0  
}
```

Figura 10 – Comando para alterar um dado em um documento

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo preto com letras brancas e com o comando para alterar um dado em um documento de uma *collection*. Fim da descrição.

Na Figura 10, estamos alterando o nome de João para Paulo. Para atualizarmos muitos dados de uma vez só em um documento, usamos o comando:

```
db.funcionarios.updateMany()
```

```
test> db.funcionarios.updateMany( { } , { $set:{profissao:'Prog'}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
test>
```

Figura 11 – Comando para alterar vários dados em um documento

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo preto com letras brancas e com o comando para alterar vários dados em um documento de uma *collection*. Fim da descrição.

Para apagar um dado específico em um documento, usamos o comando:

```
db.funcionarios.deleteOne()
```

```
test> db.funcionarios.deleteOne( { idade:17 } )
{ acknowledged: true, deletedCount: 1 }
test>
```

Figura 12 – Comando para apagar um único documento em uma *collection*

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo preto com letras brancas e com o comando para apagar um único documento de uma *collection*. Fim da descrição.

Na Figura 12, apagamos um único documento que contém idade igual a 17.

Para apagar vários documentos que contém idade 17, usamos o comando:

```
db.funcionarios.deleteMany()
```

```
test> db.funcionarios.deleteOne( { idade:17 } )
{ acknowledged: true, deletedCount: 1 }
test>
```

Figura 13 – Comando para apagar vários documentos em uma *collection*

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo preto com letras brancas e com o comando para apagar vários documentos em uma *collection*. Fim da descrição.

O comando anterior apaga todos os documentos que contém idade igual a 17.

Para selecionar um documento específico, podemos usar o operador gt. Por exemplo, podemos fazer:

```
db.funcionarios.find({idade: {$gt: 17}})

test> db.funcionarios.find({idade: {$gt: 17}})
[
  {
    _id: ObjectId("645ff3b36af7cd5f72152fd2"),
    nome: 'Paulo',
    idade: 18,
    profissao: 'Prog'
  },
  {
    _id: ObjectId("645ff3b36af7cd5f72152fd3"),
    nome: 'Maria',
    idade: 19,
    profissao: 'Prog'
  }
]
test>
```

Figura 14 – Comando para selecionar documentos em uma *collection*

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo preto com letras brancas e com o comando para selecionar documentos em uma collection. Fim da descrição.

Observe que a função gt sempre seleciona dados acima do valor colocado, para pegar valores iguais e superiores, usamos a função gte. Como, por exemplo:

```
db.funcionarios.find({idade: {$gte: 18}})

test> db.funcionarios.find({idade: {$gte: 18}})
[
  {
    _id: ObjectId("645ff3b36af7cd5f72152fd2"),
    nome: 'Paulo',
    idade: 18,
    profissao: 'Prog'
  },
  {
    _id: ObjectId("645ff3b36af7cd5f72152fd3"),
    nome: 'Maria',
    idade: 19,
    profissao: 'Prog'
  }
]
test> ■
```

Figura 15 – Comando para selecionar documentos em uma *collection*

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo preto com letras brancas e com o comando para selecionar documentos em uma *collection*. Fim da descrição.

O comando apresentado na Figura 15 seleciona todos os documentos que contém idade igual a 18 e superior a 18.

Para filtrar todos os documentos que tenham idade inferior a 17, usamos o operador `lt`.

Por exemplo:

```
db.funcionarios.find({idade: {$lt: 19}})

test> db.funcionarios.find({idade: {$lt: 19}})
[
  {
    _id: ObjectId("645ff3b36af7cd5f72152fd2"),
    nome: 'Paulo',
    idade: 18,
    profissao: 'Prog'
  }
]
test>
```

Figura 16 – Comando para selecionar documentos em uma *collection*

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo preto com letras brancas e com o comando para selecionar documentos em uma *collection*. Fim da descrição.

Na Figura 16, selecionamos todos os documentos que contêm idade menor que 19.

Para selecionar os documentos que contêm idade igual ou inferior a 1,9 temos de usar a função `lte`, tal como a seguir:

```
db.funcionarios.find({idade: {$lte: 17}})

test> db.funcionarios.find({idade: {$lte: 19}})
[
  {
    _id: ObjectId("645ff3b36af7cd5f72152fd2"),
    nome: 'Paulo',
    idade: 18,
    profissao: 'Prog'
  },
  {
    _id: ObjectId("645ff3b36af7cd5f72152fd3"),
    nome: 'Maria',
    idade: 19,
    profissao: 'Prog'
  }
]
```

Figura 17 – comando para selecionar documentos em uma *collection*

Fonte: Reprodução

**#ParaTodosVerem:** foto com fundo preto com letras brancas e com o comando para selecionar documentos em uma *collection*. Fim da descrição.

Na Figura 17, selecionamos documentos com idade igual a 19 ou inferior.

# Considerações Finais

Em resumo, o NoSQL oferece uma alternativa flexível e escalável aos Bancos de Dados Relacionais tradicionais, mas é importante entender as suas vantagens e limitações antes de escolher uma solução de armazenamento de dados. MongoDB é uma excelente alternativa para armazenar e manipular dados em *Big Data*.

# Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta Unidade:

## ▶ Vídeos

### **0 que é *Big Data* – Conceitos Básicos**

Essa referência faz um estudo sobre *Big Data*.

<https://youtu.be/JPC5mE9il0I>

### **A Melhor Maneira de Acessar seus Dados no MongoDB (Guia Completo do COMPASS)**

Esta referência apresenta um estudo sobre mongodb.

<https://youtu.be/k9ZhV4k9ULI>

### **Aprenda *deploy* com MongoDB e Digital Ocean**

Essa referência traz um estudo sobre mongodb.

[https://youtu.be/6u5\\_kpr6Xbc](https://youtu.be/6u5_kpr6Xbc)

### **Criando uma API com Spring Boot e MongoDB**

Estudo sobre mongodb.

<https://www.youtube.com/live/JOSS-pHB9iQ?feature=share>

# Referências

HOWS, D.; MEMBREY, P.; PLUGGE, E. **Introdução ao MongoDB**. S.l.: Novatec, 2019.

BANKER, K. et al. **MongoDB in action: covers MongoDB version 3.0**. S.l.: Simon and Schuster, 2016.





**Cruzeiro do Sul**  
Educacional