IBD-016 – BANCO DE DADOS - NÃO RELACIONAL

Prof. Me. Anderson Vanin

CRUD

- **CRUD** é uma sigla para Create, Read, Update e Delete;
- É onde inserimos dados, resgatamos/lemos dados, atualizamos dados e deletamos/removemos dados;
- As **quatro operações básicas de banco de dados**, tanto relacionais quanto não relacionais;
- **Praticamente toda aplicação** terá um CRUD;

Tudo é documento

- Sempre que vamos trabalhar com MongoDB é comum adicionarmos várias entidades com chaves **{ }**
- Sempre que adicionamos chave a algum local, chamamos de **document** (documento)
- Ou seja, é bem comum ouvir: **inserir um document na collection**
- Onde em SQL seria inserir o dado na tabela

Inserindo dados

- Para inserir um document utilizamos o método **insertOne**
- Desta maneira: **db.<collection>.insertOne({<dados>})**
- Onde **collection** é o nome da collection que vamos inserir dados
- E dados representa o **conjunto de chaves e valores** do dado em questão

Prática

- Crie um banco de dados chamado **crud**

```
> use crud
switched to db crud
```

- Crie uma collection chamada **create** e adicione alguns dados

```
> db.create.insertOne({nome:"Anderson",idade:48, trabalhando:true,hobbies:["Jogar","Programar"]})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("643030bf4d952311beadd6de")
}
```

Prática

- Visualizando os dados criados na **collection create**

```
> db.create.find()
{ "_id" : ObjectId("643030bf4d952311beadd6de"), "nome" : "Anderson", "idade" : 48, "trabalhando" : true,
"hobbies" : [ "Jogar", "Programar" ] }
```

- Podemos inserir mais documents com uma estrutura diferente da última criada.

```
> db.create.insertOne({nome:"Maria",idade:35})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("643031e14d952311beadd6df")
}
> db.create.find()
{ "_id" : ObjectId("643030bf4d952311beadd6de"), "nome" : "Anderson", "idade" : 48, "trabalhando" : true,
"hobbies" : [ "Jogar", "Programar" ] }
{ "_id" : ObjectId("643031e14d952311beadd6df"), "nome" : "Maria", "idade" : 35 }
>
```

Para a execução dos exercícios a seguir

- Para todos os exercícios desta aula, crie um documento em branco no word, e **faça os prints de tela dos comandos executados e resultados obtidos**.
- No término de todos os exercícios **exporte as bases de dados criadas e/ou utilizadas**.
- **Compacte tudo (prints de tela e exports das bases) em um arquivo zip com seu nome completo.**
- **Links Base de Dados (Todas):** https://github.com/ProfAndersonVanin/FATEC_Banco_de_Dados_NoSQL/tree/main/material%20de%20apoio/base%20de%20dados

Exercício 01

- Crie uma **collection** chamada **provas**
- Crie 2 dados com **nome** do aluno e um **array** chamdo **notas**, com notas de provas
- Exiba todos os dados.

Não há relação entre os dados

- Em uma collection não precisamos respeitar as chaves dos outros documents;
- Ou seja, em um BD relacional precisamos adicionar dados das colunas e em MongoDB não existe essa regra;
- Então podemos ter documents totalmente diferentes em uma collection;

Inserindo vários dados

- Podemos também inserir vários dados de uma vez só, com o método **insertMany**
- A sintaxe é a seguinte: **db.<collection>.insertMany([<dados...>])**
- Note que vamos precisar de um **array**, que vamos inserir os **documents**, separados por vírgula.

Inserindo vários dados

```
> db.create.insertMany([{nome:"Pedro",trabalhando:false},{nome:"Cintia",idade:41}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("643034574d952311beadd6e0"),
                ObjectId("643034574d952311beadd6e1")
        ]
}
>
```

```
> db.create.find()
{ "_id" : ObjectId("643030bf4d952311beadd6de"), "nome" : "Anderson", "idade" : 48, "trabalhando" : true,
"hobbies" : [ "Jogar", "Programar" ] }
{ "_id" : ObjectId("643031e14d952311beadd6df"), "nome" : "Maria", "idade" : 35 }
{ "_id" : ObjectId("643034574d952311beadd6e0"), "nome" : "Pedro", "trabalhando" : false }
{ "_id" : ObjectId("643034574d952311beadd6e1"), "nome" : "Cintia", "idade" : 41 }
>
```

Exercício 02

- Crie uma **collection chamada mercado**
- Com o comando de inserção de dados múltiplos, insira produtos com **nome, preço e disponibilidade**.

Voltando ao _id

- Já sabemos que o **_id é único** e criado em todos os documents da collection;
- Porém não necessariamente precisamos deixar a cargo do MongoDB isso, **podemos criar o nosso próprio id**;
- Exemplo: **db.<collection>.insertOne({_id:”meuid”,nome:”Anderson”})**
- Desta forma é possível personalizar este campo!

Voltando ao _id

```
> db.create.insertOne({_id:"2023",nome:"Fulana"})
{ "acknowledged" : true, "insertedId" : "2023" }
> db.create.find()
{ "_id" : ObjectId("643030bf4d952311beadd6de"), "nome" : "Anderson", "idade" : 48, "trabalhando" : true,
"hobbies" : [ "Jogar", "Programar" ] }
{ "_id" : ObjectId("643031e14d952311beadd6df"), "nome" : "Maria", "idade" : 35 }
{ "_id" : ObjectId("643034574d952311beadd6e0"), "nome" : "Pedro", "trabalhando" : false }
{ "_id" : ObjectId("643034574d952311beadd6e1"), "nome" : "Cintia", "idade" : 41 }
{ "_id" : "2023", "nome" : "Fulana" }
>
```

Exercício 03

- Crie um banco de dados chamado dadosDeCarros;
- Crie uma collection chamada carros;
- O id deve ser personalizado!
- Insira 4 carros com os seguintes dados: marca, modelo, ano fabricação, quilometragem rodada.
- Visualize todos os dados com pretty().

Mais sobre Read...

- Importe a base de dados chamada: **books.json**
- Abra um prompt de comando na mesma pasta que esta o arquivo books.json

mongoimport **books.json** -d **booksCollection** -c **books**

```
C:\Users\Anderson\Desktop\livros>mongoimport books.json -d booksCollection -c books
2023-04-07T13:01:27.186-0300    connected to: mongodb://localhost/
2023-04-07T13:01:27.338-0300    431 document(s) imported successfully. 0 document(s) failed to import.

C:\Users\Anderson\Desktop\livros>
```

Verificando o banco importado

```
> show dbs
admin          0.000GB
bancofatec     0.000GB
booksCollection  0.000GB
config         0.000GB
crud           0.000GB
empresa        0.000GB
local          0.000GB
megasena       0.000GB
primeirobanco  0.000GB
segundobanco   0.000GB
>
```

Selecionando o bd e visualizando Collections

```
> use booksCollection
switched to db booksCollection
```

```
> show collections
books
>
```

```
> db.books.find().pretty()
{
        "_id" : 4,
        "title" : "Flex 3 in Action",
        "isbn" : "1933988746",
        "pageCount" : 576,
        "publishedDate" : ISODate("2009-02-02T08:00:00Z"),
        "thumbnailUrl" : "https://s3.amazonaws.com/AKIAJC5RLADLUMVRPFDQ.book-thumb-images/ahmed.jpg",
        "longDescription" : "New web applications require engaging user-friendly interfaces   and the cooler
. With Flex 3, web developers at any skill level can create high-quality, effective, and interactive Rich In
cations (RIAs) quickly and easily. Flex removes the complexity barrier from RIA development by offering soph
ols and a straightforward programming language so you can focus on what you want to do instead of how to do
that the major components of Flex are free and open-source, the cost barrier is gone, as well!    Flex 3 in
 easy-to-follow, hands-on Flex tutorial. Chock-full of examples, this book goes beyond feature coverage and
t Flex to work in real day-to-day tasks. You'll quickly master the Flex API and learn to apply the technique
your Flex applications stand out from the crowd.    Interesting themes, styles, and skins  It's in there.  W
databases  You got it.  Interactive forms and validation  You bet.  Charting techniques to help you visualiz
  The expert authors of Flex 3 in Action have one goal   to help you get down to business with Flex 3. Fast.
x books are overwhelming to new users   focusing on the complexities of the language and the super-specializ
in the Flex eco-system; Flex 3 in Action filters out the noise and dives into the core topics you need every
numerous easy-to-understand examples, Flex 3 in Action gives you a strong foundation that you can build on a
xity of your projects increases.",
        "status" : "PUBLISH",
        "authors" : [
                "Tariq Ahmed with Jon Hirschi",
                "Faisal Abid"
        ],
```

Encontrar dado com valor específico

- Para encontrar um dado específico podemos **definir um document dentro do find**.
- O primeiro argumento da opção também é chamado de **filtro**;
- Exemplo: **db.books.find({ pageCount: 362})**
- Todos os livros com 362 páginas serão retornados!

Encontrar dado com valor específico

```
> db.books.find({ pageCount: 362})
{ "_id" : 20, "title" : "Taming Jaguar", "isbn" : "1884777686", "pageCount" : 362, "publishedDate" : ISODate("2000-07-01
T07:00:00Z"), "thumbnailUrl" : "https://s3.amazonaws.com/AKIAJC5RLADLUMVRPFDQ.book-thumb-images/barlotta3.jpg", "longDes
cription" : "Taming Jaguar is part of the PowerBuilder Developer's series, which includes Distributed Application Develo
pment with PowerBuilder 6 and Jaguar Development with PowerBuilder 7.    An application server is the heart of your ente
rprise computing architecture, centralizing your web content, business logic, and access to your data and legacy applica
tions. Sybase's application server, Jaguar CTS, delivers performance, scalability, and flexibility running CORBA , COM,
Java/EJB, C++, and PowerBuilder components.    If you are looking to adopt Jaguar in your enterprise, look no further. T
aming Jaguar shows you how to solve the real-world problems of installing, trouble-shooting, designing, developing, and
maintaining a Jaguar application. Topical chapters are organized in a Q & A format making it easy for you to quickly fin
d the solution to your problem. They also provide foundational and background information as well as detailed technical
how-tos.    Although designed so you can find your problems easily, this book is meant to be read cover-to-cover with ea
ch chapter discussing its topic exhaustively.    What's inside:    J2EE development  Java Servlets  Jaguar administratio
n & code balancing  EJBs  Web development with PowerDynamo  Advanced component design ", "status" : "PUBLISH", "authors"
 : [ "Michael J. Barlotta", "Jason R. Weiss" ], "categories" : [ "PowerBuilder" ] }
>
```

Encontrar dado com valor específico

```
> db.books.find({ title: "Hibernate in Action (Chinese Edition)"}).pretty()
{
        "_id" : 23,
        "title" : "Hibernate in Action (Chinese Edition)",
        "pageCount" : 400,
        "publishedDate" : ISODate("1999-06-01T07:00:00Z"),
        "thumbnailUrl" : "https://s3.amazonaws.com/AKIAJC5RLADLUMVRPFDQ.book-thumb-images/bauer-cn.jpg",
        "status" : "PUBLISH",
        "authors" : [
                "Christian Bauer",
                "Gavin King"
        ],
        "categories" : [
                "Java"
        ]
}
>
```

Exercício 04

- Selecione o livro com o título de: MongoDB in Action
- Selecione os livros do autor: Jason R. Weiss

Encontrar dado entre valores

- Para esta função vamos utilizar o operador **$in**;
- Exemplo: **db.books.find({categories:{$in:[“Java”,”Internet”]}})**
- Precisamos criar uma **lista de valores** que queremos buscar;
- Todos estes registros que contiverem um destes valores será retornado;

Múltiplas condições

- Dados podem ser encontrados baseados em múltiplas condições;
- Basta **adicionar um vírgula no document** e inserir o próximo requisito;
- Exemplo: **db.books.find({pageCount: 592,_id:63}).pretty()**
- Neste caso buscamos por um livro com 592 páginas e que tenha o id igual a 63;
- **Obs: esta consulta também é semelhante ao operador AND em SQL**.

Múltiplas condições

```
> db.books.find({pageCount: 592,_id:63}).pretty()
{
        "_id" : 63,
        "title" : "POJOs in Action",
        "isbn" : "1932394583",
        "pageCount" : 592,
        "publishedDate" : ISODate("2006-01-01T08:00:00Z"),
        "thumbnailUrl" : "https://s3.amazonaws.com/AKIAJC5RLADLUMVRPFDQ.book-thumb-images/crichardson.jpg",
        "shortDescription" : "\"POJOs in Action is required reading for battle-weary EJB developers and for new develope
rs who want to avoid the sins of the fathers by using lightweight frameworks.    -- C# Online.NET",
        "longDescription" : "There is agreement in the Java community that EJBs often introduce more problems than they
solve. Now there is a major trend toward lightweight technologies such as Hibernate, Spring, JDO, iBATIS, and others, al
l of which allow the developer to work directly with the simpler Plain Old Java Objects, or POJOs. Bowing to the new con
sensus, EJB 3 now also works with POJOs.    POJOs in Action describes these new, simpler, and faster ways to develop ent
erprise Java applications. It shows you how to go about making key design decisions, including how to organize and encap
sulate the domain logic, access the database, manage transactions, and handle database concurrency.    Written for devel
opers and designers, this is a new-generation Java applications guide. It helps you build lightweight applications that
are easier to build, test, and maintain. The book is uniquely practical with design alternatives illustrated through num
erous code example",
        "status" : "PUBLISH",
        "authors" : [
                "Chris Richardson"
        ],
        "categories" : [
                "Java"
        ]
}
> |
```

Atualizando um dado

- Para atualizar um dado utilizamos o método **updateOne**;
- Primeiro realizamos o filtro e depois inserimos o que precisa ser atualizado;
- Exemplo:
  db.books.updateOne({_id:314},{$set:{pageCount:1000}})
- Aqui atualizamos as páginas do livro com id 314 para 1000;
- O operador **$set** é onde ficam os valores a serem atualizado.

Atualizando um dado

```
> db.books.findOne({_id:314})
{
        "_id" : 314,
        "title" : "SQL Server 2005 Reporting Services in Action",
        "isbn" : "1932394761",
        "pageCount" : 600,
        "publishedDate" : ISODate("2006-11-01T08:00:00Z"),
        "thumbnailUrl" : "https://s3.amazonaws.com/AKIAJC5RLADLUMVRPFDQ.book-thumb-images/updegraff.jpg",
        "longDescription" : "Reports are the lifeline of business, so a good reporting environment is a big deal. With a
 powerful tool like Microsoft Reporting Services, .NET developers can add reporting to any type of application, regardle
ss of its target platform or development language. Greatly improved for SQL Server 2005, Reporting Services now provides
 tighter integration with SQL Server, improved developer tools, and an expanded array of options to empower end users.
  SQL Server 2005 Reporting Services in Action helps you build and manage flexible reporting solutions and develop repor
t-enabled applications. In this clear, well-illustrated book, you  ll follow a report from creation to publication. Alon
g the way you  ll explore flexible delivery options like web-based, on-demand, and subscribed reports  complete with co
ol new features like direct printing and client-side sorting.  For applications that require custom reporting, you'll
learn to define reports with RDL and push them to the Report Server using the Report Manager Web Service API. You  ll al
so see how to write server extensions to expand the range of data processing and report delivery options.  Written for
 developers with a solid foundation in .NET and SQL Server.",
        "status" : "PUBLISH",
        "authors" : [
                "Bret Updegraff"
        ],
        "categories" : [
                "Microsoft"
        ]
}
> |
```

Atualizando um dado

```
> db.books.updateOne({_id:314},{$set:{pageCount:1000}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

```
> db.books.findOne({_id:314})
{
        "_id" : 314,
        "title" : "SQL Server 2005 Reporting Services in Action",
        "isbn" : "1932394761",
        "pageCount" : 1000,
        "publishedDate" : ISODate("2006-11-01T08:00:00Z"),
        "thumbnailUrl" : "https://s3.amazonaws.com/AKIAJC5RLADLUMVRPFDQ.book-t
```

Exercício 05

- Altere o titulo do livro com id 20 para “Meu primeiro Update”;
- Encontre o registro e veja se foi corretamente modificado.

Atualizando vários ítens

- Para atualizar diversos itens utilizamos updateMany;
- Este método tem a mesma lógica de execução de que updateOne;
- Exemplo:
  db.books.updateMany({categories:”Java”},{$set:{status:”UNPUBLISHED”}})
- Neste update atualizamos todos os dados da categoria Java, alteramos o status destes registros.

Adicionando dados com update

- O update pode servir para adicionar um dado ao document;
- Basta **inserir um valor para uma chave** que não existe no mesmo;
- Exemplo: **db.books.updateMany({authors:”Vikram Goyal”,{$set:{downloads:1000}}})**
- Aqui adicionamos a chave downloads com o valor de 1000 a todos os livros de Vikram;

Exercício 06

- Adicione a todos os livros com mais de 500 páginas o dado:

bestseller: true

- Depois faça uma seleção para verificar se houve a modificação

Trocando todo o documento

- Podemos trocar todos os dados do documento com o **replaceOne**.
- Ou seja, haverá uma substituição de dados;
- Exemplo: **db.books.replaceOne({_id:120},{foi: “Substituído”})**
- Neste caso trocamos todos os dados do registro com id 120 para o document do segundo argumento.

Adicionar item a um array

- Se tentarmos atualizar um array diretamente vamos substituir ele;
- Para adicionarmos um item vamos precisar do operador **$push**
- Exemplo: **db.books.updateOne({_id:210},{$push: {categories:”PHP”}})**
- Neste caso adicionamos a categoria PHP ao livro com id 201

Remover um ítem

- A remoção de itens é bem parecida com a atualização;
- Baseado em um filtro, podemos deletar um elemento;
- Exemplo: **db.books.deleteOne({_id:20})**
- Neste caso deletamos o item de id igual a 20

Remover mais de um ítem

- Para remover mais de um item utilizamos deleteMany;
- Exemplo: **db.books.deleteMany({categories:”Java”})**
- Neste caso removemos todos os livros da categoria Java;

Exercício 07

- Remova o livro com o isbn igual a 1933988320
- Remova todos os livros que pertençam a categoria PowerBuilder;
- Faça uma seleção de dados para verificar se estes registros foram removidos de fato.

Remover todos os itens

- Para remover todos os itens basta utilizar o **deleteMany**;
- Porém passando um **filtro vazio**;
- Exemplo: **db.books.deleteMany({})**

CUIDADO AQUI!

Tipos de dados no Mongo

- **Categorias** para os dados inseridos no sistema;
- Escolher os melhores tipos de dados nos ajuda a ter um **projeto melhor estruturado**;
- **O MongoDB tem alguns tipos diferentes do SQL**, por isso a importância de conhecer os mais relevantes;
- Alguns deles são: **strings, numbers, boolean, array, etc**

Exercício 08

- Para estes exercícios utilize a base de dados chamada: **restaurant.json**
- **mongoimport restaurant.json –d restaurantData –c restaurants**
- Para os próximos exercícios vamos utilizar os operadores $eq, $gt, $gte, $lt, $lte, $in, $ne e $exists.

Exercício 08

- Selecione restaurantes que tenham nota maior ou igual a 3 e que tipo de comida seja Breakfast;
- Adicione um campo que qualifica os restaurantes ruins, ou seja, os que tem nota menor ou igual a 2. Depois faça uma seleção dos mesmos com exists, baseado neste novo campo;

Operador $text

- O **$text** faz uma busca sobre o texto do campo que foi informado no filtro;
- Exemplo: **db.restaurants.find({$text:{$search: “pizza”}}).pretty()**
- Porém é preciso criar um índice em algum dos campos, se não ele não funciona;

Operador $text

- Criando um índice chamado text

db.restaurants.createIndex({ name: "text" })

```
> db.restaurants.createIndex({ name: "text" })
{
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "createdCollectionAutomatically" : false,
        "ok" : 1
}
>
```

Operador $text

- Agora fazemos a busca por pizza

db.restaurants.createIndex({ name: "text" })

Operador $text

```
> db.restaurants.find({$text:{$search: "pizza"}}).pretty()
{
        "_id" : ObjectId("55f14313c7447c3da70524ea"),
        "URL" : "http://www.just-eat.co.uk/restaurants-bitzapizza-bl2/menu",
        "address" : "824 Bury Road",
        "address line 2" : "Bolton",
        "name" : "Bitza Pizza",
        "outcode" : "BL2",
        "postcode" : "6PA",
        "rating" : 5,
        "type_of_food" : "Pizza"
}
{
        "_id" : ObjectId("55f14313c7447c3da70524e9"),
        "URL" : "http://www.just-eat.co.uk/restaurants-bitzapizza-bl2/menu",
        "address" : "824 Bury Road",
        "address line 2" : "Bolton",
        "name" : "Bitza Pizza",
        "outcode" : "BL2",
        "postcode" : "6PA",
        "rating" : 5,
        "type_of_food" : "Pizza"
}
{
        "_id" : ObjectId("55f14313c7447c3da70524d9"),
        "URL" : "http://www.just-eat.co.uk/restaurants-bistropizza-bh1/menu",
        "address" : "242 Old Christchurch Road",
        "address line 2" : "Bournemouth",
        "name" : "Bistro Pizza",
```

```
> db.restaurants.find({$text:{$search: "pizza"}}).count()
326
>
```