



**DESENVOLVIMENTO DE SOFTWARE MULTIPLATAFORMA**

**Disciplina: IBD-016 – BANCO DE DADOS - NÃO RELACIONAL**

**Aula 03: Tipos de Bancos de Dados NoSQL**

Data 29/02/2024

Prof. Me. Anderson Silva Vanin

# TIPOS DE BANCOS DE DADOS NÃO RELACIONAIS

- **Chave-valor:** todos os registros fazem parte da mesma coleção de elementos, e a única coisa que todos eles têm em comum é uma chave única;
- **Colunar:** todos os registros fazem parte da mesma tabela, mas cada um deles pode ter colunas diferentes;
- **Documento:** cada registro fica armazenado em uma coleção específica, mas mesmo dentro de uma coleção, não existe um esquema fixo para os registros;
- **Grafo:** os registros são nós em um grafo interligados por relacionamentos.

# TIPOS DE BANCOS DE DADOS NÃO RELACIONAIS

Cada um deles tem suas vantagens e desvantagens, principalmente na hora de consultar e recuperar os registros. Porém, mesmo dentro do mesmo tipo de banco, existem diferentes implementações que podem ter diferenças de performance e escalabilidade.

# Banco de Dados Chave-Valor

Consiste em uma modelagem que **indexa os dados a uma chave**. Ao se armazenar os dados, sua forma de procura se dá por uma base similar a um **dicionário**, onde estes possuem uma **chave**. Esta forma de armazenamento é **livre de “schema”**, **permite a inserção de dados em tempo de execução**, sem conflitar o banco e não influenciando na disponibilidade, pois seus **valores são isolados e independentes entre si**.

Alguns exemplos são: Oracle NoSQL, Riak, Azure Table Storage, BerkeleyDB e Redis.

# Banco de Dados Chave-Valor

Chave	Valor
carro_3345_cor	preto
carro_3345_pneu	17
carro_3365_cor	branco
carro_3365_pneu	15
carro_4560_peso	1215
carro_4715_ano	2016

# Banco de Dados Colunar

Os bancos de dados colunares, como o Amazon Redshift e o Google BigQuery, armazenam a informação coluna-a-coluna, o que permite:

- **Maior compressão** – já que dados de tipos iguais são armazenados juntos, há uma otimização de espaço utilizado.
- **Eliminação da necessidade de índices** – não é necessário reorganizar como as cores de blocos estão ordenadas (existem outras opções de otimização como sharding).
- **Alta-performance** para operações de agregação.

# Banco de Dados Colunar

Para facilitar o entendimento imagine um conjunto de dados como o seguinte:

Id	Nome	Sobrenome	Idade
1	João	Pereira	32
2	Carlos	Gonçalves	41
3	Kondado	Inteligência	13

Em um banco de dados tradicional, essas informações seriam armazenadas em um formato muito parecido com a tabela abaixo:

1	João	Pereira	32	2	Carlos	Gonçalves	41	3	Kondado	Inteligência	13
---	------	---------	----	---	--------	-----------	----	---	---------	--------------	----

# Banco de Dados Colunar

Veja só como os valores são reordenados em um banco de dados Colunar:



Se você quiser fazer a soma das idades, o banco de dados fará apenas uma operação de leitura para a cor laranja e fará o cálculo.



# Banco de Dados Documentos

Consiste em uma estrutura baseada em uma **coleção de documentos**, sendo um documento um **objeto** que contém um código único com um conjunto de informações, podendo ser strings, documentos aninhados ou ainda listas. Inicialmente pode ser semelhante ao modelo de chave-valor (Key-value), no entanto, diferencia-se em ter um conjunto de documentos e cada um destes recebe um identificador único, assim como as chaves, dentro da coleção. Ao se armazenar os dados em **JSON**, o desenvolvimento é facilitado, pois há suporte a vários tipos de dados. Exemplos destes são o MongoDB e CouchBase.

# Banco de Dados Documentos

```
{
  "id": 49,
  "País": "Alemanha",
  "Região": "Europa",
  "População":
  "PrincipaisCidades": [
    {
      "NomeCidade": "Berlin",
      "População": 3610156,
    },
    {
      "NomeCidade": "Hamburg",
      "População": 1746342,
    }
  ]
}
```

# Banco de Dados Grafos

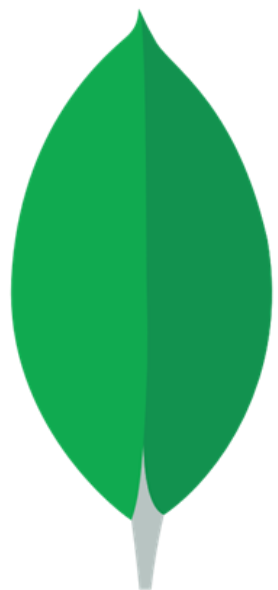
Este modelo armazenamento utiliza **três componentes básicos**: um **grafo** para representar um dado, **arestas** ou ligações para representar a associação entre os grafos e os **atributos** (ou propriedades) dos nós e relacionamentos. Modelo altamente usado onde exijam dados fortemente ligados. Este modelo é vantajoso onde há consultas complexas frente aos outros modelos, pois seu diferencial é o **ganho de performance**. Alguns exemplos são: Neo4J, OrientedDB, GraphBase e InfiniteGraph.

# Banco de Dados Grafos



# MongoDB

MongoDB é um software de **banco de dados orientado a documentos** livre, de código aberto e multiplataforma, escrito na linguagem C++. Classificado como um programa de **banco de dados NoSQL**, o MongoDB usa documentos semelhantes a **JSON com esquemas**.



mongoDB®

# MongoDB

- **MongoDB Community Server:** é totalmente gratuita e disponível para Windows, Linux e MacOS.
- **MongoDB Enterprise Server:** é a edição comercial do MongoDB, disponível como parte da inscrição MongoDB Enterprise Advanced.
- **MongoDB Atlas:** está disponível como um serviço sob-demanda totalmente gerenciável. MongoDB Atlas funciona em diversas plataformas de computação em nuvem, como AWS, Microsoft Azure, e Google Cloud Platform.

# UM PROBLEMA DA VIDA REAL

Durante as próximas aulas, vamos usar um projeto exemplo no qual enfrentaremos alguns dos problemas, em que um banco de dados relacional pode não ser exatamente a melhor solução para persistir nossos dados. O projeto que vamos usar é o ***ligado***, uma aplicação que contém o perfil de músicos, bandas, álbuns e músicas.

# Ligado: REQUISITOS

Para começar a explorar um pouco mais do projeto, vamos focar primeiro no cadastro dos álbuns. Na nossa aplicação, além da banda e das músicas, um álbum também possui alguns dados extras, como: ano de lançamento, ilustrador da capa, produtor ou qualquer outra informação que desejarmos.



# Ligado: REQUISITOS

Porém, cada disco pode ter mais ou menos informações que os outros. Os usuários devem poder adicionar qualquer informação a um disco, mesmo que seja um tipo de informação que somente um disco terá. Por exemplo, um disco pode ter um campo com o número de semanas que ficou em primeiro lugar na billboard, e o número de singles que atingiu a posição — mas a maior parte dos discos não consegue nem um single sequer na primeira posição e conseqüentemente nunca terá dados nestes campos.

# Ligado: REQUISITOS

Dado este cenário, o que precisamos é que cada álbum possua uma estrutura diferente, não necessariamente única. Mas enquanto alguns álbuns possuirão apenas nome, artista e músicas, outros possuirão data de lançamento, estúdio e produtor. Porém, poderemos ter álbuns apenas com dados básicos e o estúdio, e outros apenas com os dados básicos e o produtor, e assim por diante. O que o nosso modelo de dados exige é que cada álbum possa ter todos os dados que conhecemos sobre ele, mas sem afetar os outros álbuns dos quais conhecemos um conjunto de dados diferentes.

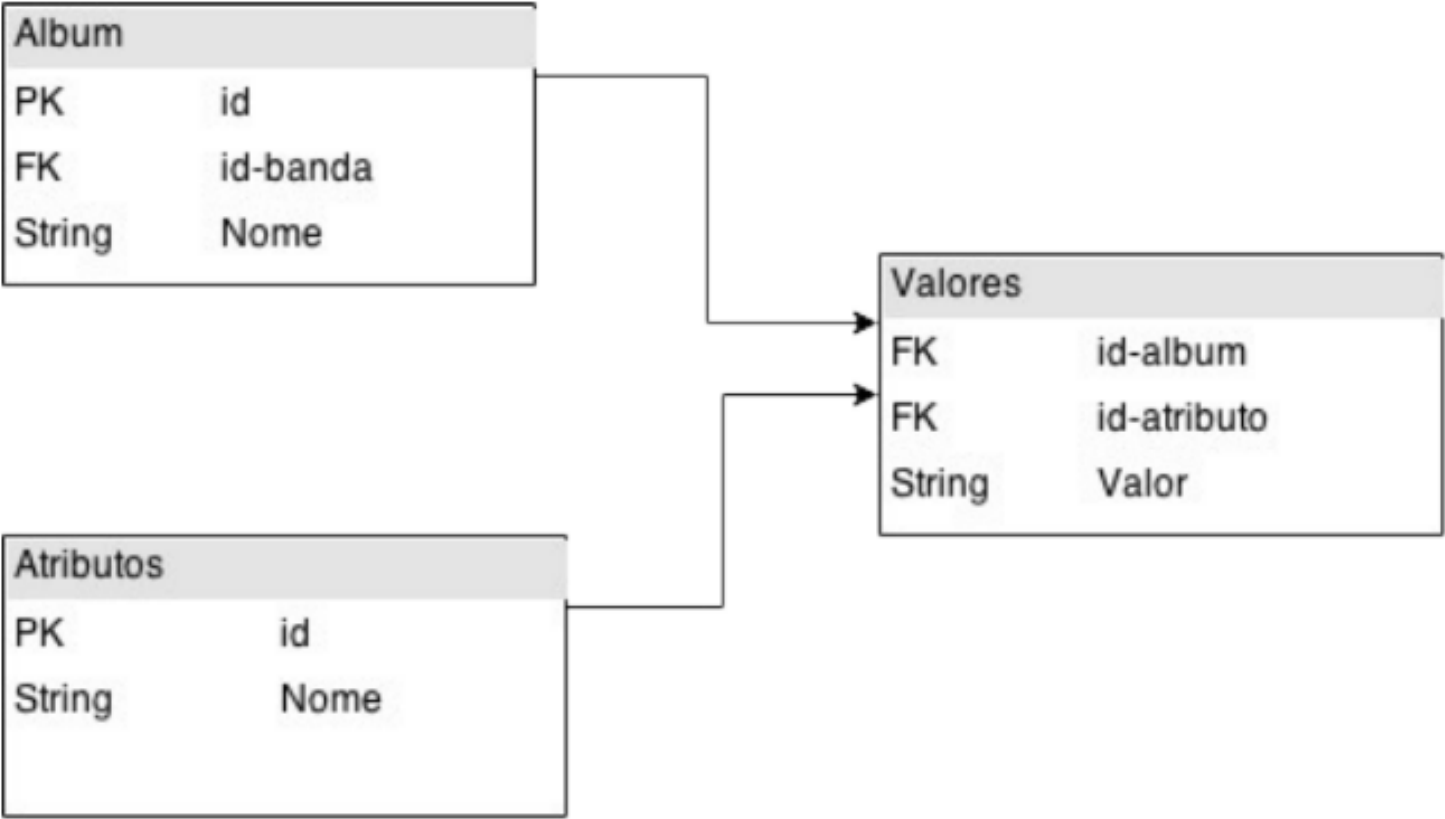
# Ligado: REQUISITOS

Em um **banco de dados relacional tradicional**, onde existe o conceito de tabela, **todos os registros de uma tabela devem possuir a mesma estrutura**. Cada tabela possui uma definição de colunas e tipos de dados, colunas obrigatórias, valores padrões e qualquer outra regra que o banco de dados usado suporte. Este conjunto de regras que define uma tabela é chamado de ***schema***.

**Mas com o nosso cadastro de álbuns, é exatamente o oposto disso que precisamos.**

# Ligado: REQUISITOS

Veja como ficaria a modelagem de dados para uma dessas variações mais simples:



# Ligado: REQUISITOS

Ao inserir os dados, teríamos algo como:

Albuns

id	id-banda	nome
1	1	Master of Puppets
2	1	...And Justice for All

Atributos

id	nome
1	Data de lançamento
1	Estudio onde foi gravado

Valores

id-album	id-atributo	valor
1	1	03/03/1086
2	1	25/08/1088
1	2	Sweet Silence Studios

# Ligado: REQUISITOS

Embora esta solução resolva nossa necessidade, reparem que para a simples tarefa de exibir os detalhes de um álbum precisaremos de uma query um pouco mais complexa, como a seguinte:

```
SELECT atr.nome, val.valor  
FROM atributos atr  
INNER JOIN valores val  
    ON val.id-atributo = atr.id  
WHERE val.id-album = 1;
```

Talvez o exemplo dos álbuns possa parecer um pouco exagerado para você, mas pense nas necessidades do catálogo de produtos de um e-commerce , no qual existem produtos como roupas que têm tamanhos P, M e G, e sapatos com tamanhos 33, 34 etc., além de TVs com atributos como voltagem e tamanho de tela. Mesmo dentro da categoria de TVs, podemos ter um atributo como número de óculos na caixa, para o caso de TVs 3D. Este tipo de problema está mais próximo do que imaginamos.

Como descrito anteriormente, o tipo de banco NoSQL baseado em documento permite que cada documento de uma coleção tenha um esquema único. Ou seja, podemos cadastrar vários álbuns, sendo que cada um deles pode ter um conjunto de dados diferentes dos outros.

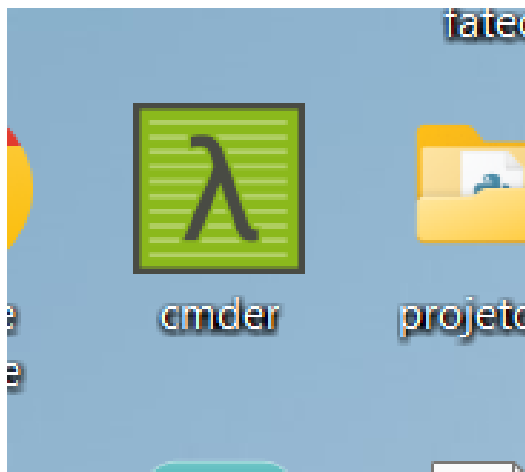


# COMEÇANDO COM NOSQL

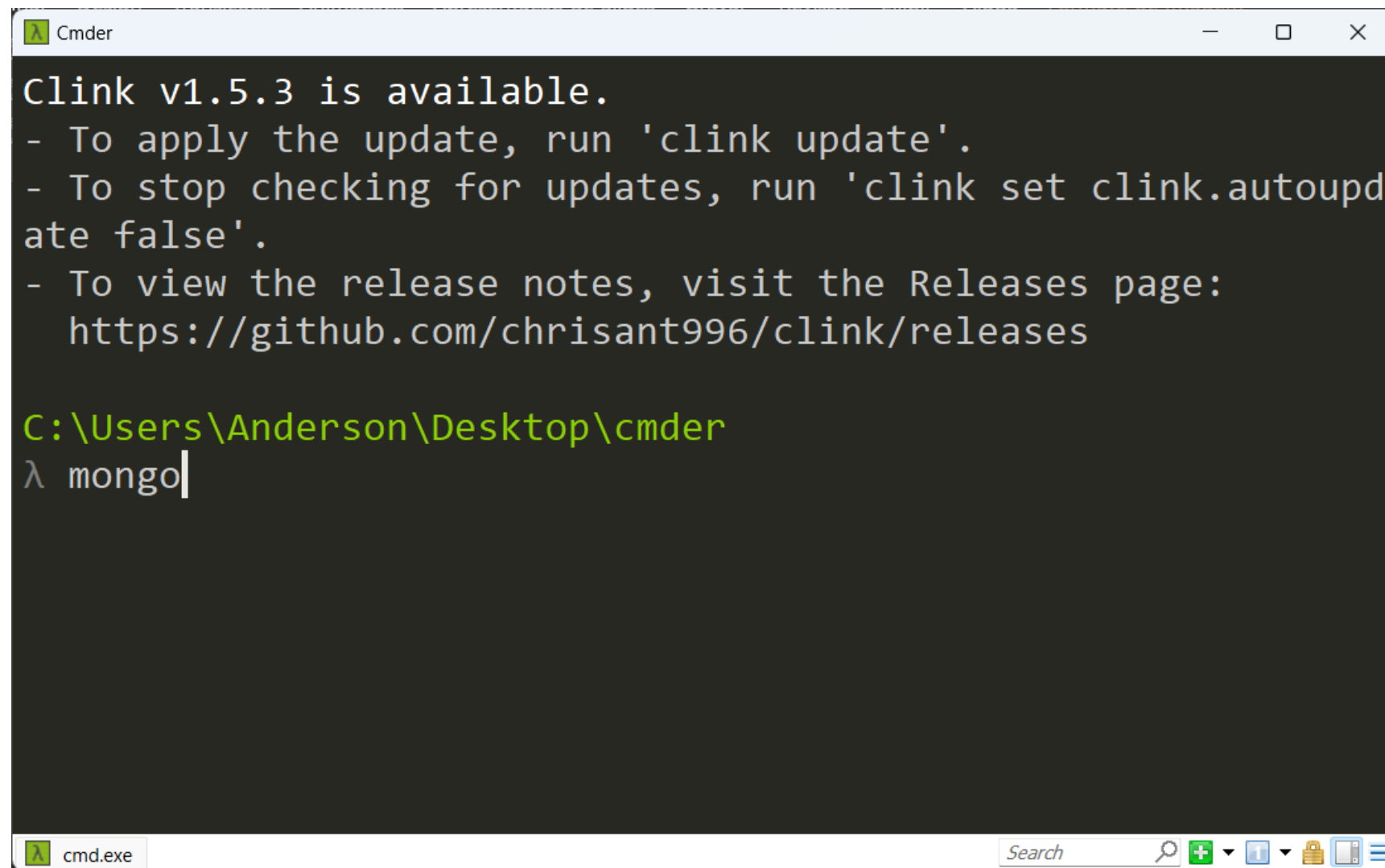
Termos/conceitos do SQL	Termos/conceitos do MongoDB
Database	Database
Tabela	Coleção
Linha	Documento ou documento BSON
Coluna	Campo
Index	Index
Table join	Documentos aninhado (embedded) e vinculados
Chave primária – especifica qualquer coluna única ou uma combinação de colunas como chave primária	Chave primária – No MongoDB, a chave primária é automaticamente definida como campo _id
Agregação (group by)	Agregação de pipeline

# CRIANDO NOSSO PRIMEIRO DOCUMENTO

Abra o CMDER

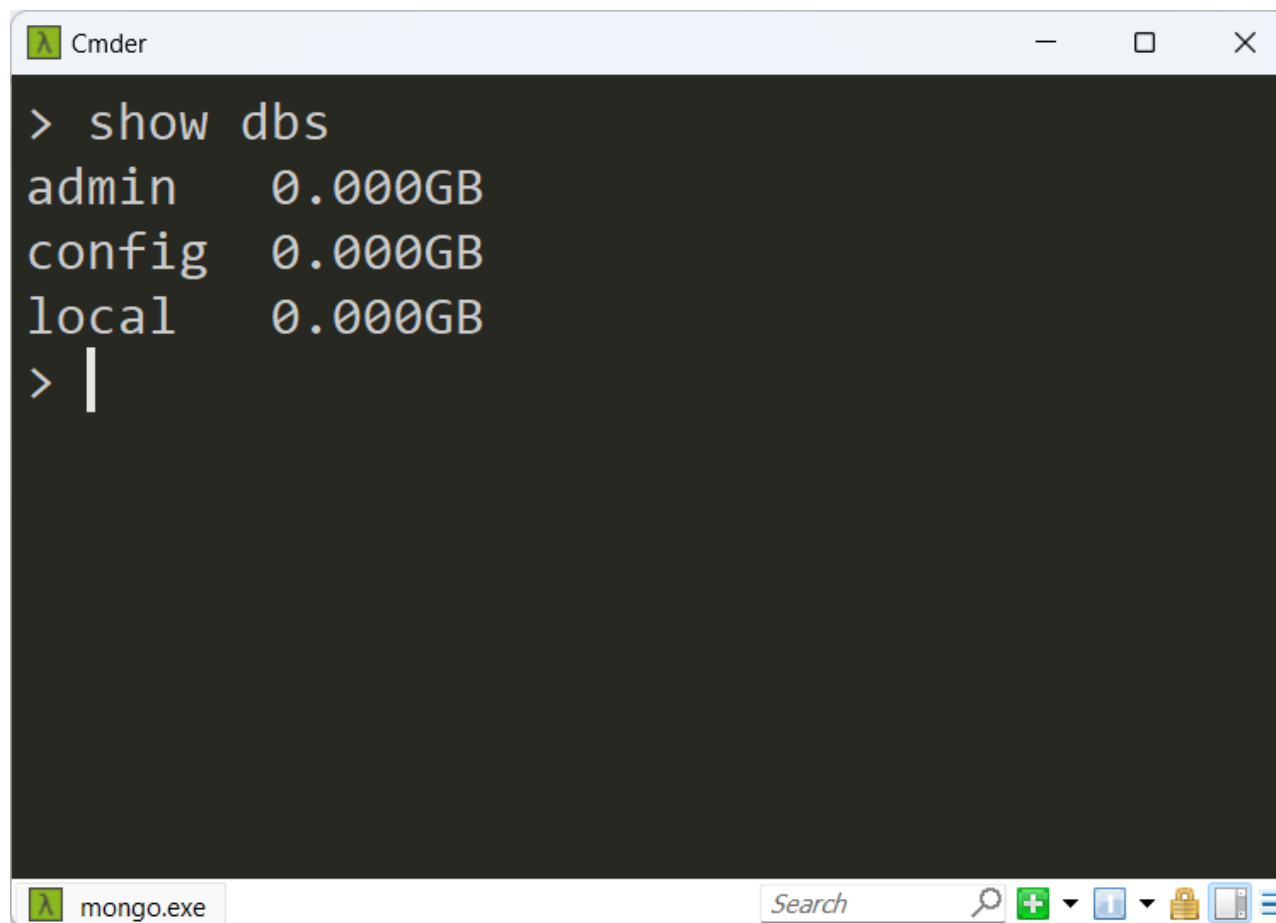


Digite **mongo**

A screenshot of a Windows Command Prompt window titled 'Cmder'. The window has a dark background and white text. The text inside the window reads: 'Clink v1.5.3 is available.' followed by three bullet points: '- To apply the update, run \'clink update\'.', '- To stop checking for updates, run \'clink set clink.autoupdate false\'.', and '- To view the release notes, visit the Releases page: https://github.com/chrisant996/clink/releases'. Below this, the current directory is shown as 'C:\Users\Anderson\Desktop\cmdr' in green text. The prompt 'λ mongo' is visible at the bottom of the window. The taskbar at the bottom shows the 'cmd.exe' process and a search bar.

# CRIANDO NOSSO PRIMEIRO DOCUMENTO

Podemos listar todos os bancos de dados que temos na nossa instalação usando o comando **show dbs**

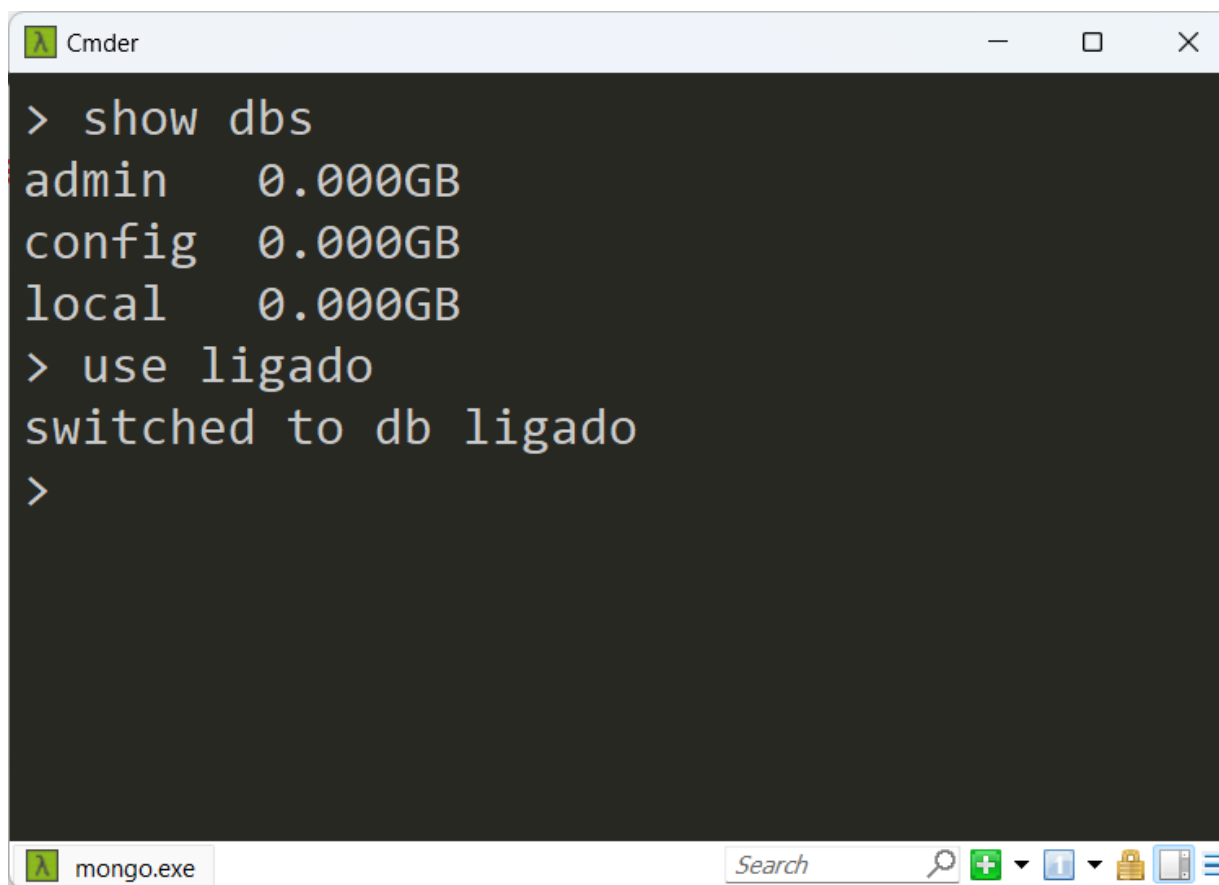


```
Cmder
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
> |
```

The screenshot shows a terminal window titled 'Cmder' with a dark background. The command prompt shows the execution of 'show dbs', which lists three databases: 'admin' (0.000GB), 'config' (0.000GB), and 'local' (0.000GB). Below the output, there is a prompt character '>' followed by a vertical bar cursor '|'. At the bottom of the window, a taskbar shows 'mongo.exe' as the active application, along with a search bar and several system icons.

# CRIANDO NOSSO PRIMEIRO DOCUMENTO

Para a nossa aplicação ligado, criaremos um novo banco de dados homônimo, usando o comando **use ligado**.



```
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
> use ligado
switched to db ligado
>
```

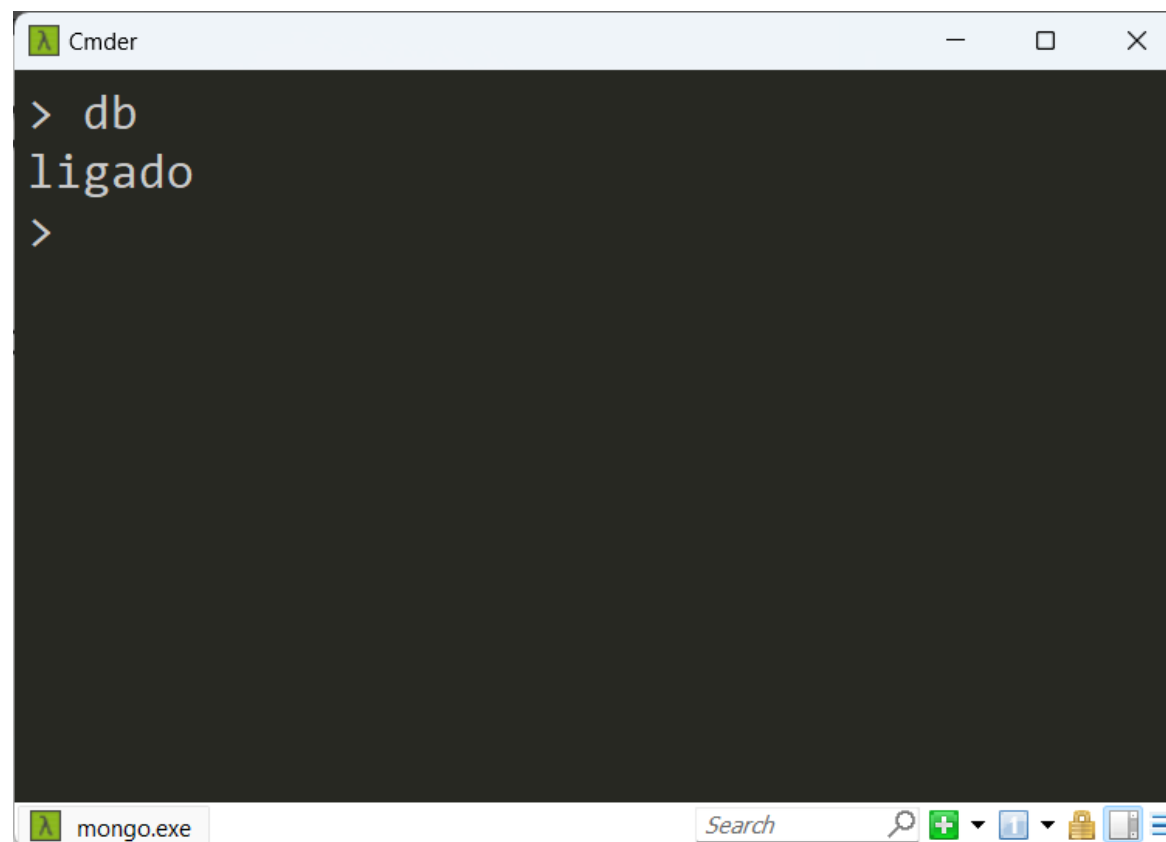
# CRIANDO NOSSO PRIMEIRO DOCUMENTO

Para inserir um documento, usamos a função chamada **insert** . A função insert deve ser chamada a partir do nome de uma coleção, que deve ser chamada a partir do nosso objeto base **db** . No final, a chamada que faremos para criar um documento será **db.nome\_da\_colecao.insert(CORPO\_DO\_DOCUMENTO)** .

O primeiro documento que criaremos será um álbum vazio na nossa nova coleção chamada de albuns .

# CRIANDO NOSSO PRIMEIRO DOCUMENTO

Use o comando **db** para verificar qual banco de dados está selecionado antes de iniciar a construção do seu primeiro documento.

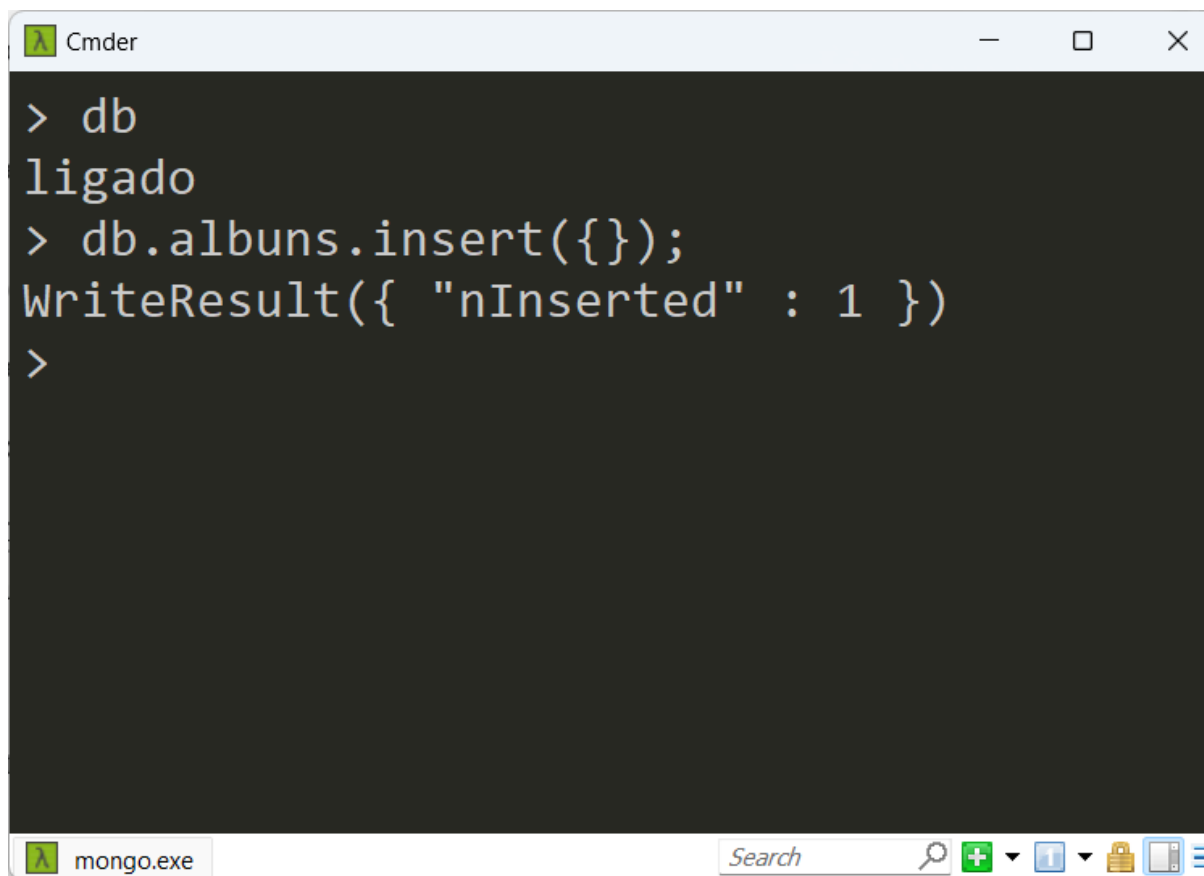


```
Cmder
> db
ligado
>
```

# CRIANDO NOSSO PRIMEIRO DOCUMENTO

Criando a primeira collection e inserindo um documento vazio.

Comando: ***db.albuns.insert({});***

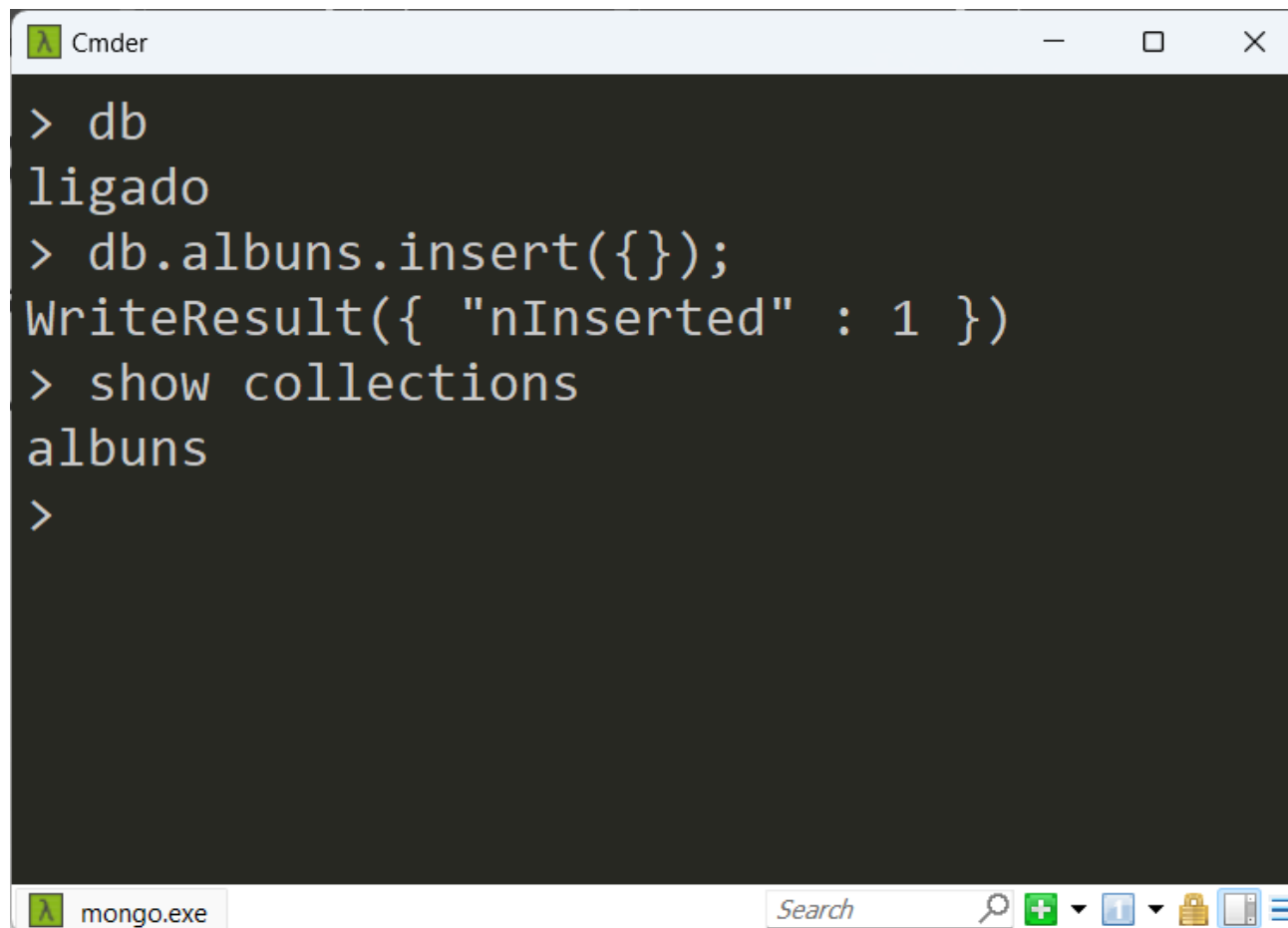


```
> db
ligado
> db.albuns.insert({});
WriteResult({ "nInserted" : 1 })
>
```

# CRIANDO NOSSO PRIMEIRO DOCUMENTO

Visualizando as collections.

Comando: *show collections*



```
Cmdr
> db
ligado
> db.albuns.insert({});
WriteResult({ "nInserted" : 1 })
> show collections
albuns
>
```

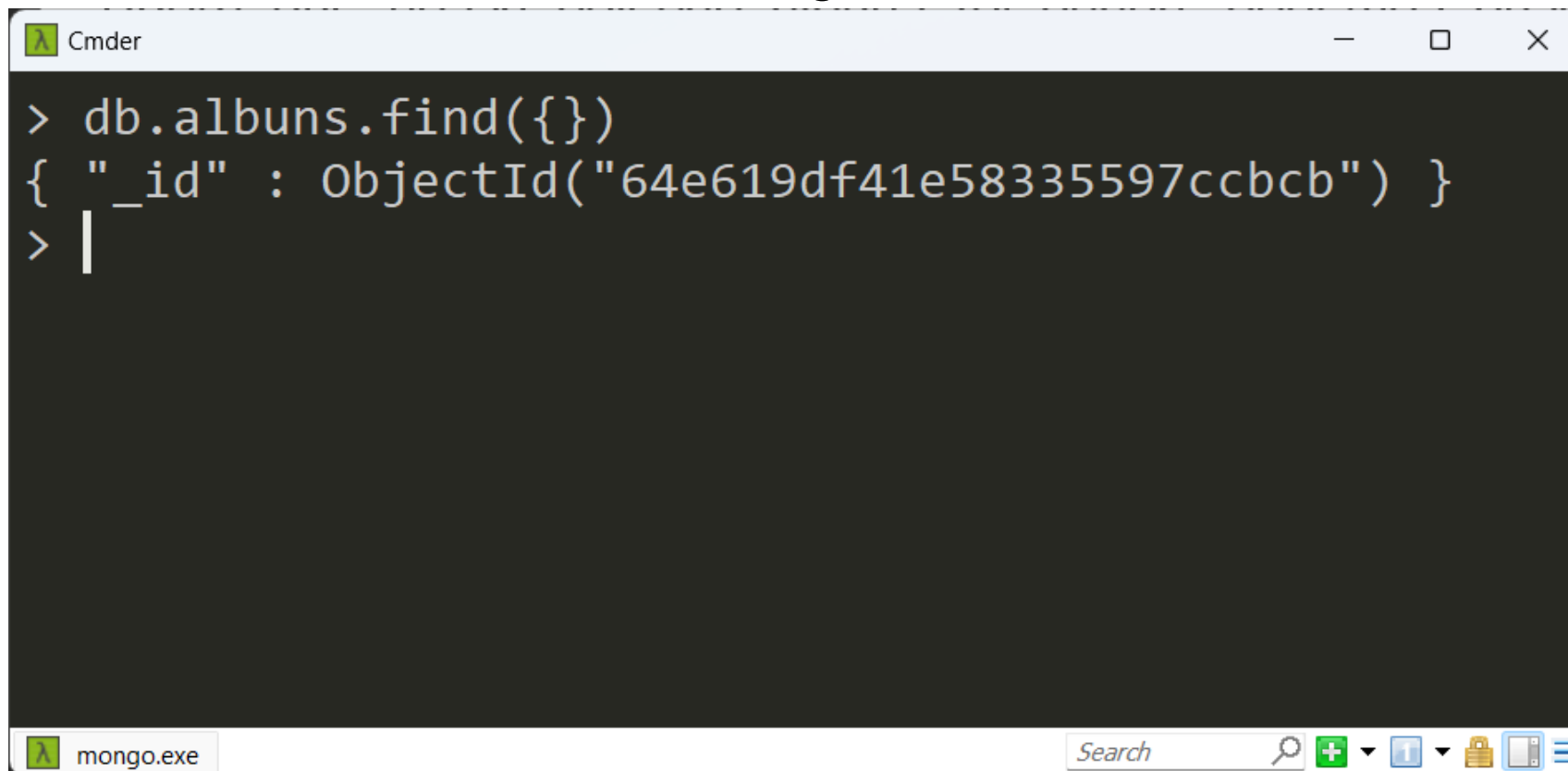
The screenshot shows a terminal window titled 'Cmdr' with a dark background. The text is white and yellow. The commands entered are: '> db', 'ligado', '> db.albuns.insert({});', 'WriteResult({ "nInserted" : 1 })', '> show collections', and 'albuns'. The prompt '>' is shown at the end of the last line. At the bottom of the window, there is a taskbar with a 'mongo.exe' icon, a search bar, and several icons including a magnifying glass, a plus sign, a document, a lock, and a mobile phone.



# CRIANDO NOSSO PRIMEIRO DOCUMENTO

Agora que nossa coleção albuns foi criada, podemos buscar um documento dentro dela.

No nosso caso, é albuns , e faremos da seguinte maneira: **db.albuns.find({})** .



```
Cmder
> db.albuns.find({})
{ "_id" : ObjectId("64e619df41e58335597ccbcdb") }
> |
```

mongo.exe

# CRIANDO NOSSO PRIMEIRO DOCUMENTO

Toda vez que inserirmos um documento, ele automaticamente gera um ObjectId , um tipo especial de BSON com 12 bytes. No caso do exemplo, o valor gerado foi **54c023bf09ad726ed094e7db** .

# INSERINDO DOCUMENTOS

Para quem não está acostumado com o formato, bem resumidamente, definimos um objeto usando as chaves {} e, dentro dela, definimos um grupo de chaves e seus respectivos valores separados pelo caractere : . E usamos , entre cada um desses pares de chave/valor. Veja um exemplo:

```
{"nome" : "MongoDB",  
  "tipo" : "Documento"}
```

```
  {"nome"           : "Master of Puppets",  
   "dataLancamento" : new Date(1986, 2, 3),  
   "duracao"        : 3286}
```

# INSERINDO DOCUMENTOS

Agora que já entendemos um pouco melhor o insert , vamos criar alguns álbuns para vermos como efetuar buscas no MongoDB.

# INSERINDO DOCUMENTOS

```
db.albums.insert({  
  "nome" : "Master of Puppets",  
  "dataLancamento" : new Date(1986, 2, 3),  
  "duracao" : 3286  
})
```

```
db.albums.insert({  
  "nome" : "...And Justice for All",  
  "dataLancamento" : new Date(1988, 7, 25),  
  "duracao" : 3929  
})
```

```
db.albums.insert({  
  "nome" : "Peace Sells... but Who's Buying?",  
  "duracao" : 2172,  
  "estudioGravacao" : "Music Grinder Studios",  
  "dataLancamento" : new Date(1986, 8, 19)  
})
```

```
db.albums.insert({  
  "nome" : "Reign in Blood",  
  "dataLancamento" : new Date(1986, 9, 7),  
  "artistaCapa" : "Larry Carroll",  
  "duracao" : 1738  
})
```

```
db.albums.insert({  
  "nome" : "Among the Living",  
  "produtor" : "Eddie Kramer"  
})
```

# INSERINDO DOCUMENTOS



```
> db.albuns.insert({  
...   "nome" : "Master of Puppets",  
...   "dataLancamento" : new Date(1986, 2, 3),  
...   "duracao" : 3286  
... })  
WriteResult({ "nInserted" : 1 })  
>
```

The screenshot shows a terminal window with a dark background and light-colored text. The window title bar is light blue and contains the text 'Cmder'. The terminal content shows a MongoDB shell command to insert a document into the 'albuns' collection. The document has fields for 'nome' (Master of Puppets), 'dataLancamento' (a date object for February 3, 1986), and 'duracao' (3286). The output shows 'WriteResult({ "nInserted" : 1 })', indicating the document was successfully inserted. The terminal window is part of a taskbar at the bottom, which also shows a taskbar icon for 'mongo.exe' and a search bar.

# BUSCANDO DOCUMENTOS NO MONGODB

Assim como o **insert** , agora há pouco utilizamos a função **find** , passando como argumento `{}` . Este argumento é chamado de *criteria* , e é uma forma de definir filtros para a busca que desejamos fazer. A primeira busca que faremos é buscar em um campo específico por um valor específico. Podemos, por exemplo, buscar pelo álbum que se chama *Master of Puppets* utilizando **`db.albums.find({"nome" : "Master of Puppets"})`** .

# BUSCANDO DOCUMENTOS NO MONGODB



```
> db.albums.find({"nome" : "Master of Puppets"})
{ "_id" : ObjectId("64e67a0d5a2788c5ea9c0287"), "nome" : "Master of Puppets", "dataLancamento" : ISODate("1986-03-03T03:00:00Z"), "duracao" : 3286 }
> |
```

The screenshot shows a Windows command prompt window titled "Cmder". The command prompt is running a MongoDB query: `db.albums.find({"nome" : "Master of Puppets"})`. The output of the query is a JSON document: `{ "_id" : ObjectId("64e67a0d5a2788c5ea9c0287"), "nome" : "Master of Puppets", "dataLancamento" : ISODate("1986-03-03T03:00:00Z"), "duracao" : 3286 }`. The command prompt is running the `mongo.exe` command. The taskbar at the bottom shows the `mongo.exe` application running, along with a search bar and several icons.



# BUSCANDO DOCUMENTOS NO MONGODB

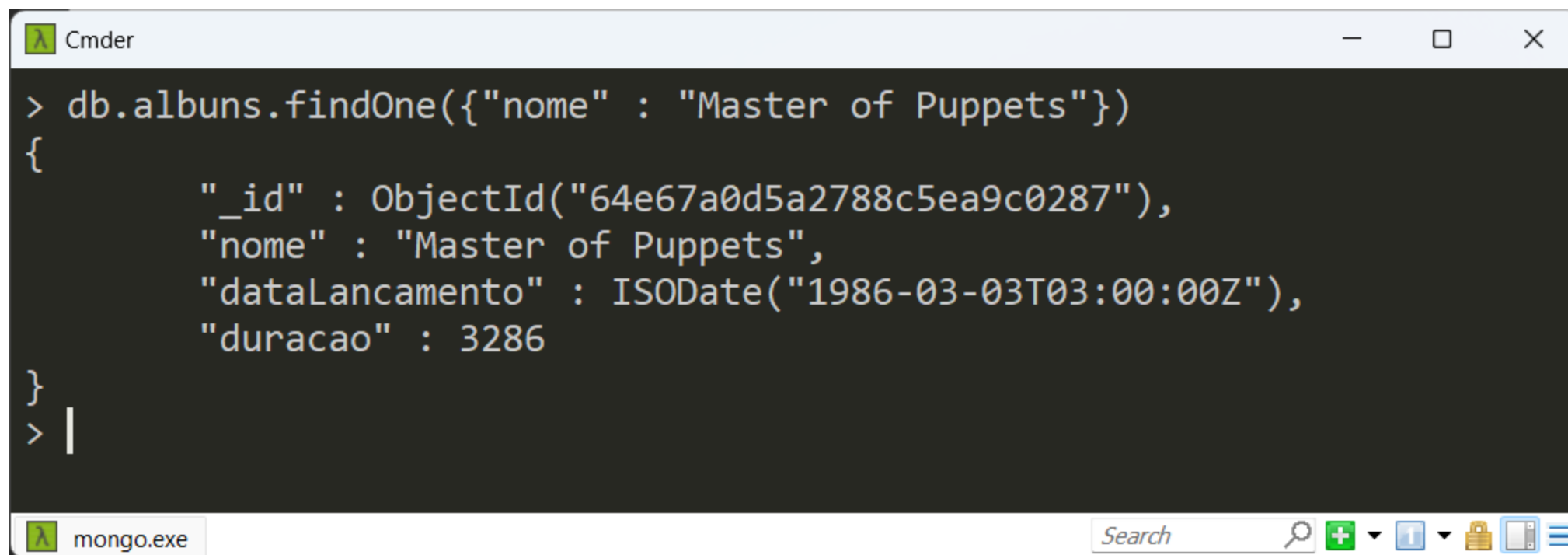
Isso é equivalente a seguinte consulta se estivéssemos em um Banco de Dados Relacional:

```
SELECT *  
FROM albuns a  
WHERE a.nome = "Master of Puppets"
```

# BUSCANDO DOCUMENTOS NO MONGODB

Além do **find** que retorna uma lista de documentos que satisfazem os critérios passados como argumento, o MongoDB também disponibiliza a função **findOne**, que retorna apenas um documento: o primeiro que satisfaça as condições.

```
db.albuns.findOne({"nome" : "Master of Puppets"})
```



```
> db.albuns.findOne({"nome" : "Master of Puppets"})
{
  "_id" : ObjectId("64e67a0d5a2788c5ea9c0287"),
  "nome" : "Master of Puppets",
  "dataLancamento" : ISODate("1986-03-03T03:00:00Z"),
  "duracao" : 3286
}
> |
```

# BUSCANDO DOCUMENTOS NO MONGODB

Podemos também pedir para retornar uma consulta com todos os documentos da coleção utilizando o comando:

```
db.albuns.find({})
```

```
> db.albuns.find({})
{ "_id" : ObjectId("64e619df41e58335597ccbcb") }
{ "_id" : ObjectId("64e67a0d5a2788c5ea9c0287"), "nome" : "Master of Puppets", "dataLancamento" : ISODate("1986-03-03T03:00:00Z"), "duracao" : 3286 }
{ "_id" : ObjectId("64e67a365a2788c5ea9c0288"), "nome" : "...And Justice for All", "dataLancamento" : ISODate("1988-08-25T03:00:00Z"), "duracao" : 3929 }
{ "_id" : ObjectId("64e67a395a2788c5ea9c0289"), "nome" : "Peace Sells... but Who's Buying?", "duracao" : 2172, "estudioGravacao" : "Music Grinder Studios", "dataLancamento" : ISODate("1986-09-19T03:00:00Z") }
{ "_id" : ObjectId("64e67a485a2788c5ea9c028a"), "nome" : "Reign in Blood", "dataLancamento" : ISODate("1986-10-07T03:00:00Z"), "artistaCapa" : "Larry Carroll", "duracao" : 1738 }
{ "_id" : ObjectId("64e67a4a5a2788c5ea9c028b"), "nome" : "Among the Living", "produtor" : "Eddie Kramer" }
```

# BUSCANDO DOCUMENTOS NO MONGODB

Podemos melhorar o retorno da consulta com uma função chamada **pretty()**.

**db.albuns.find({}).pretty()**

Equivalente a:

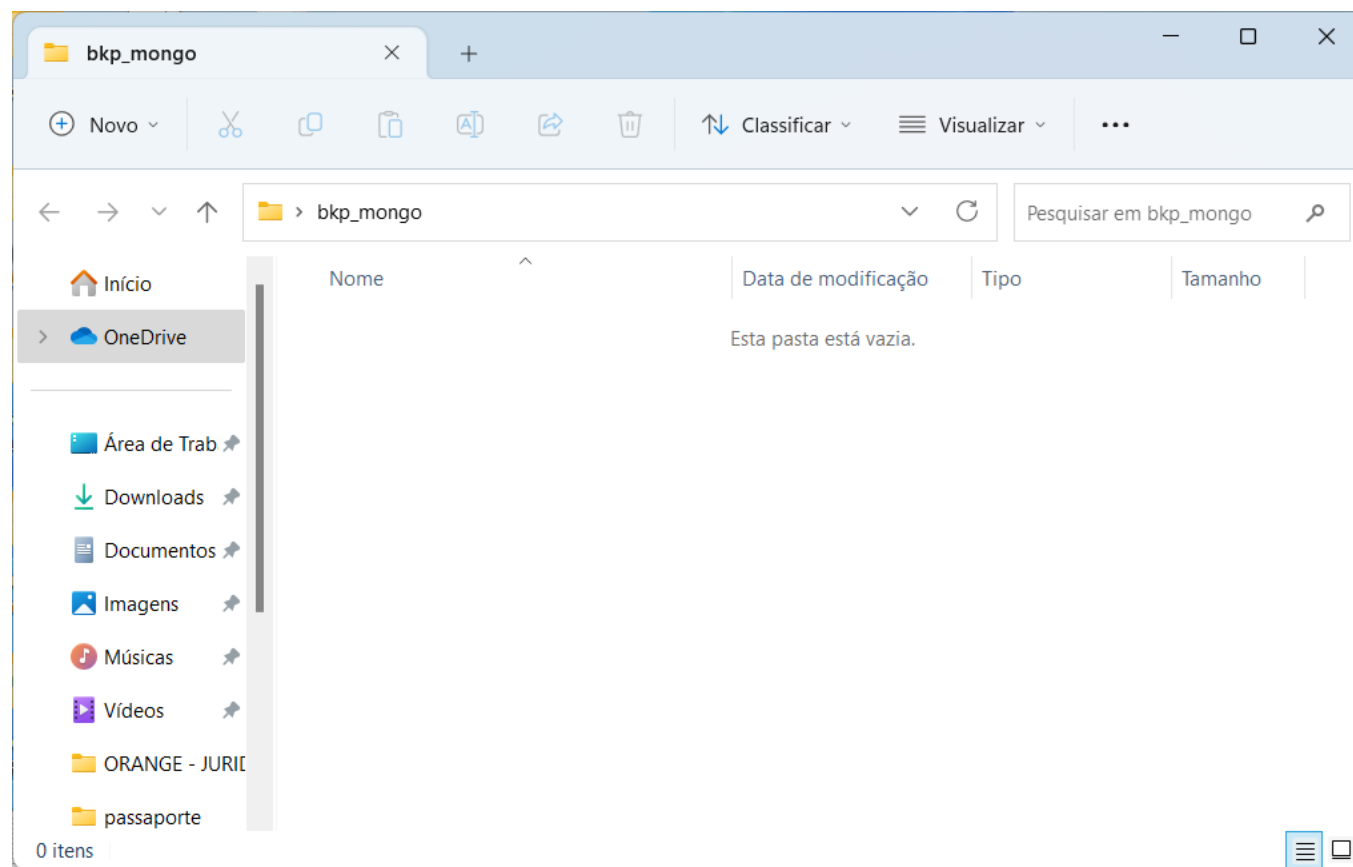
```
SELECT *
```

```
FROM albuns
```

```
> db.albuns.find({}).pretty()
{ "_id" : ObjectId("64e619df41e58335597ccbcdb"),
  "nome" : "Peace Sells... but Who's Buying?",
  "duracao" : 3286,
  "dataLancamento" : ISODate("1986-03-03T03:00:00Z"),
  "estudioGravacao" : "Music Grinder Studios",
  "produtor" : "Eddie Kramer" }
{ "_id" : ObjectId("64e67a0d5a2788c5ea9c0287"),
  "nome" : "Master of Puppets",
  "duracao" : 3286,
  "dataLancamento" : ISODate("1986-03-03T03:00:00Z"),
  "estudioGravacao" : "Music Grinder Studios",
  "produtor" : "Eddie Kramer" }
{ "_id" : ObjectId("64e67a365a2788c5ea9c0288"),
  "nome" : "...And Justice for All",
  "duracao" : 3929,
  "dataLancamento" : ISODate("1988-08-25T03:00:00Z"),
  "estudioGravacao" : "Music Grinder Studios",
  "produtor" : "Eddie Kramer" }
{ "_id" : ObjectId("64e67a395a2788c5ea9c0289"),
  "nome" : "Peace Sells... but Who's Buying?",
  "duracao" : 2172,
  "dataLancamento" : ISODate("1986-09-19T03:00:00Z"),
  "estudioGravacao" : "Music Grinder Studios",
  "produtor" : "Eddie Kramer" }
{ "_id" : ObjectId("64e67a485a2788c5ea9c028a"),
  "nome" : "Peace Sells... but Who's Buying?",
  "duracao" : 2172,
  "dataLancamento" : ISODate("1986-09-19T03:00:00Z"),
  "estudioGravacao" : "Music Grinder Studios",
  "produtor" : "Eddie Kramer" }
```

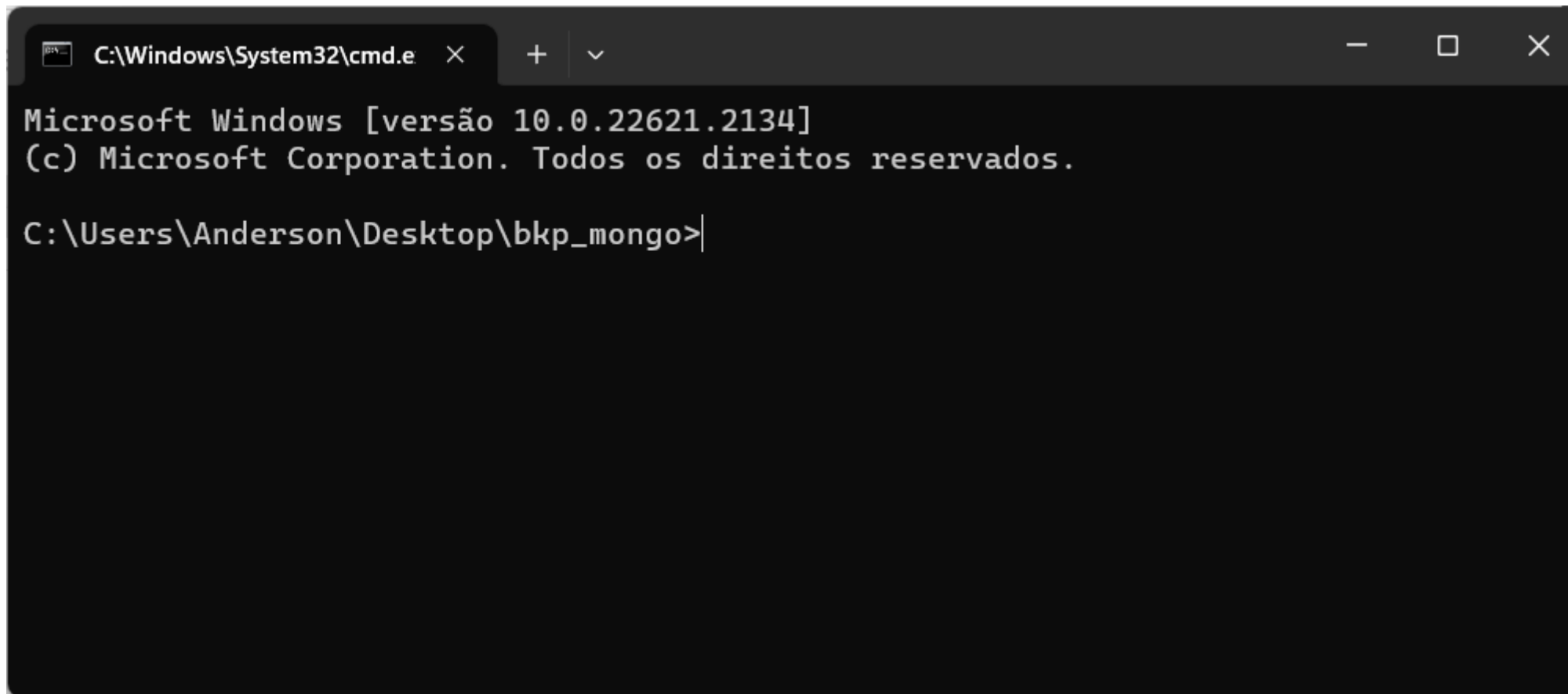
# FAZENDO BACKUP NO MONGODB

Para realizar um backup do que fizemos até agora, crie uma pasta, por exemplo na área de trabalho e renomeie para **bkp\_mongo**.



# FAZENDO BACKUP NO MONGODB

Acesse esta pasta pelo **cmd** do Windows.



```
C:\Windows\System32\cmd.e × + ∨  
Microsoft Windows [versão 10.0.22621.2134]  
(c) Microsoft Corporation. Todos os direitos reservados.  
C:\Users\Anderson\Desktop\bkp_mongo>|
```

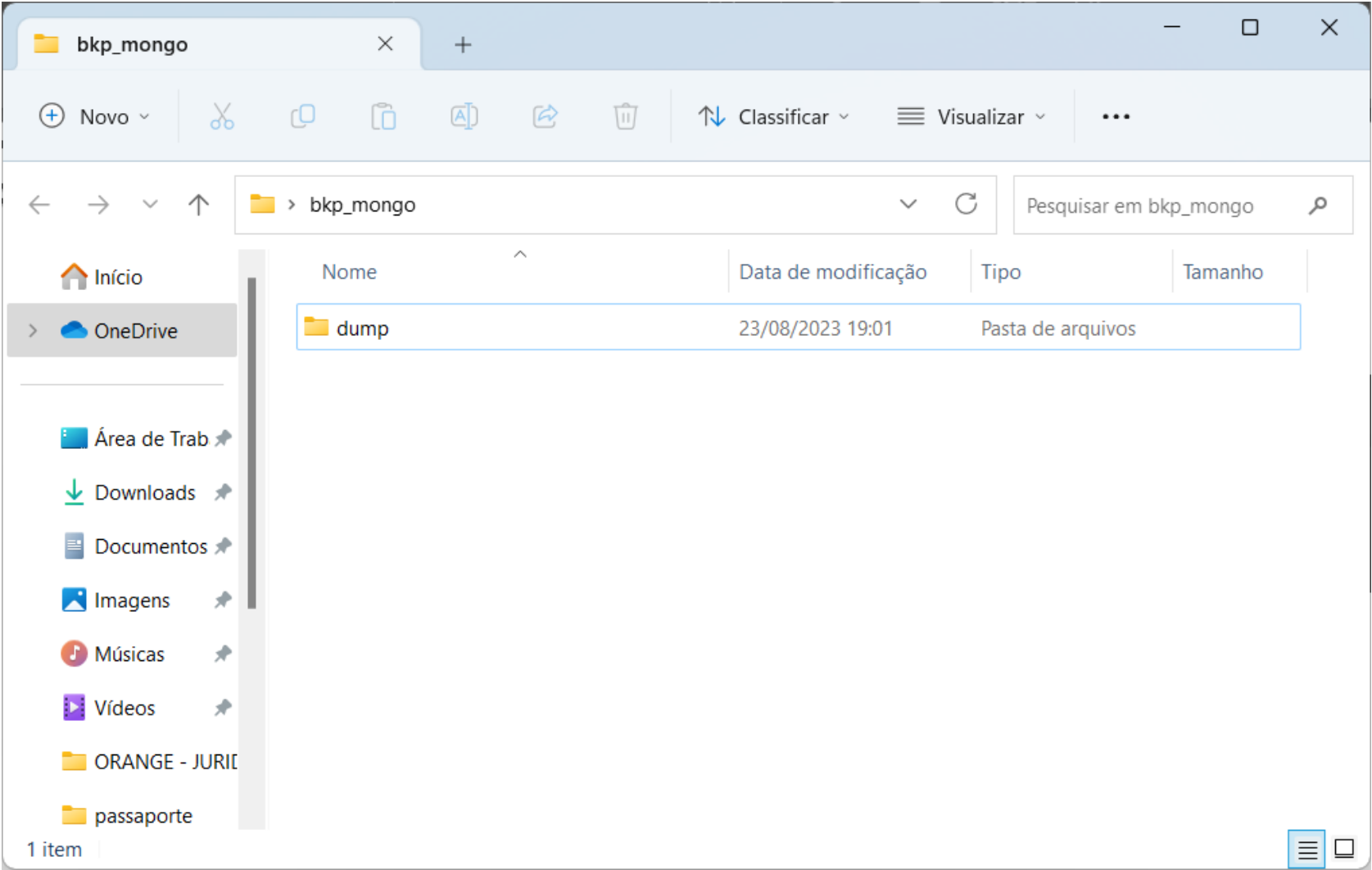
# FAZENDO BACKUP NO MONGODB

Digite o comando: **mongodump --db ligado**

```
C:\Users\Anderson\Desktop\bkp_mongo>mongodump --db ligado
2023-08-23T19:01:37.276-0300    writing ligado.albuns to dump\ligado\albuns.bson
2023-08-23T19:01:37.333-0300    done dumping ligado.albuns (6 documents)

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

# FAZENDO BACKUP NO MONGODB



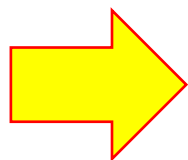


# APAGANDO UM BANCO DE DADOS NO MONGODB

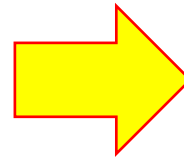
Certifique-se de estar no banco de dados correto (use o comando **db** e verifique se o banco de dados listado é o que deseja realmente apagar).

Para apagar digite o comando: **db.dropDatabase()**

```
> db  
ligado
```



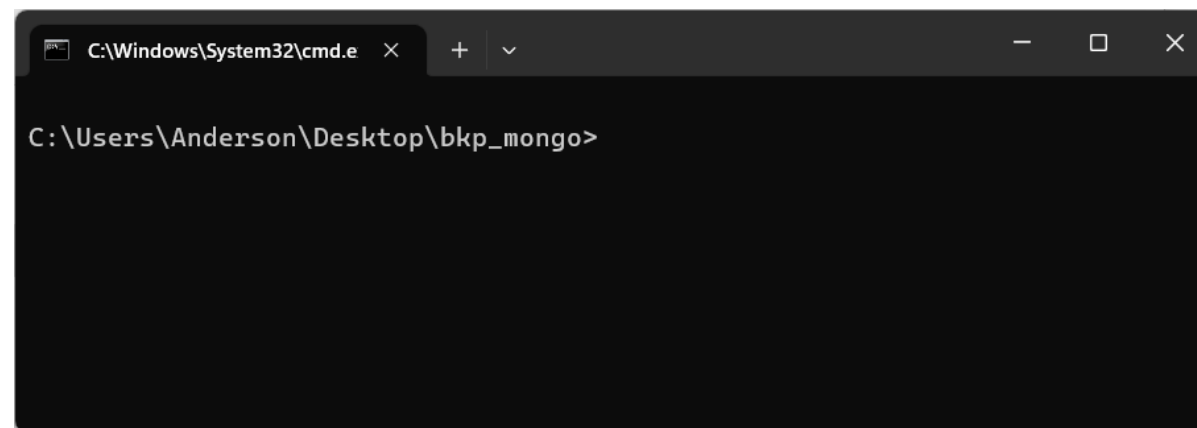
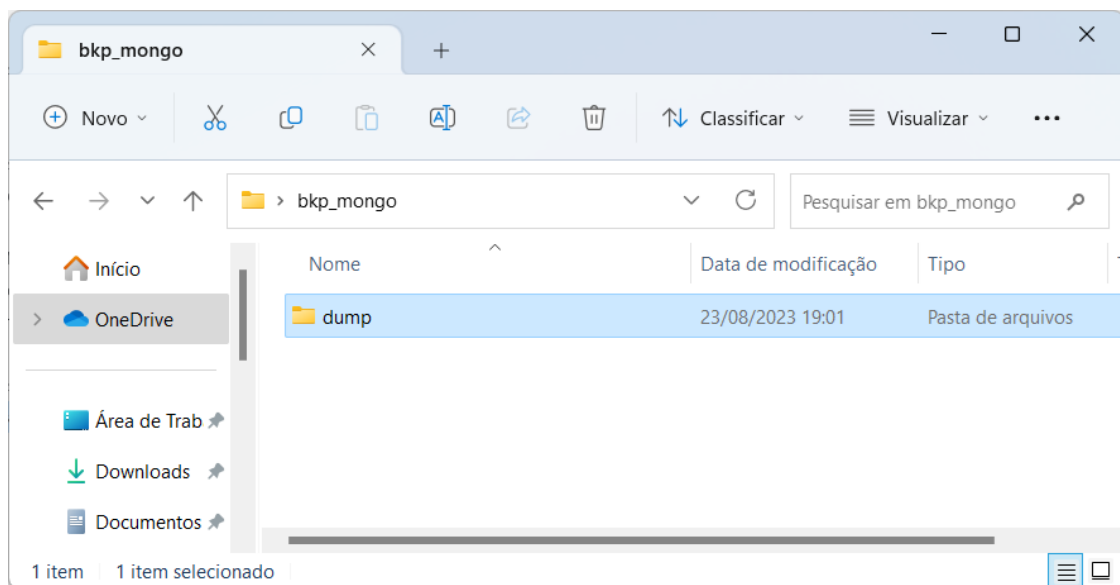
```
Cmder  
> db.dropDatabase()  
{ "ok" : 1 }  
> |  
mongo.exe
```



```
Cmder  
> show dbs  
admin      0.000GB  
config     0.000GB  
local      0.000GB  
>  
mongo.exe
```

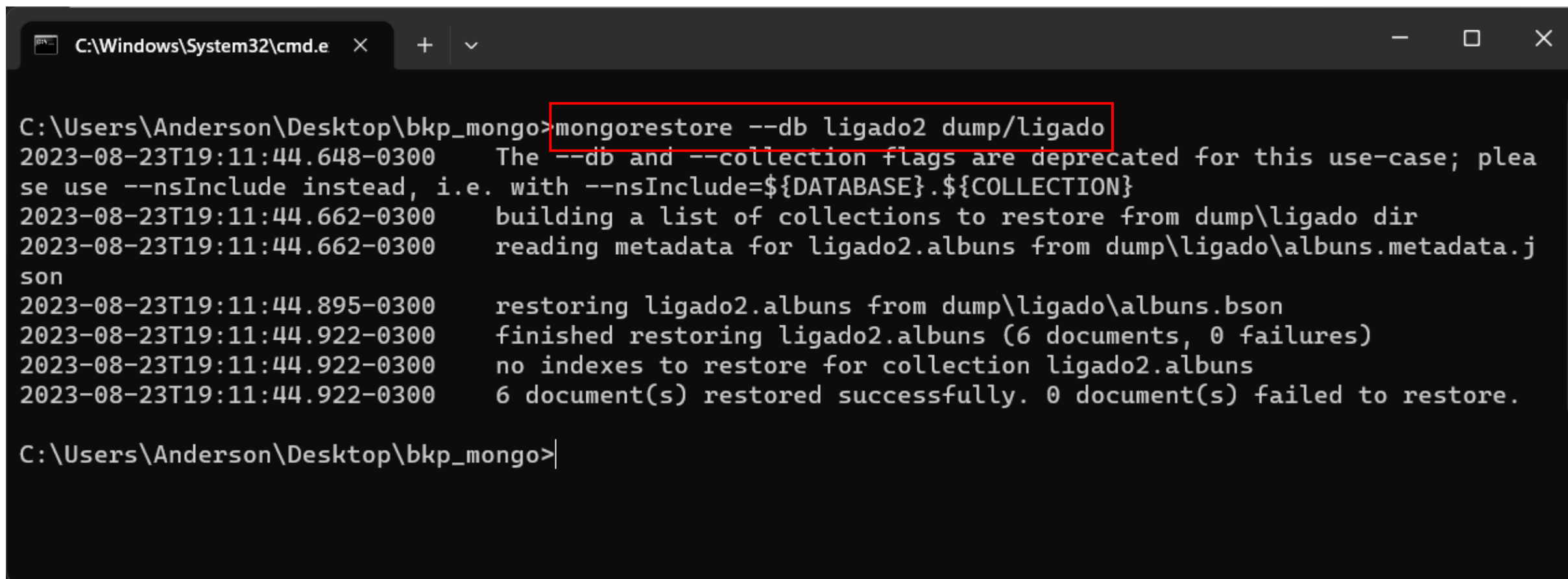
# IMPORTANDO UM BANCO DE DADOS NO MONGODB

Abra a pasta na qual estão os arquivos do banco de dados exportado anteriormente e abra um prompt de comando (cmd).



# IMPORTANDO UM BANCO DE DADOS NO MONGODB

Digite o comando: **mongorestore --db ligado2 dump/ligado**



```
C:\Windows\System32\cmd.e  x  +  v

C:\Users\Anderson\Desktop\bkp_mongo>mongorestore --db ligado2 dump/ligado
2023-08-23T19:11:44.648-0300   The --db and --collection flags are deprecated for this use-case; please use --nsInclude instead, i.e. with --nsInclude=${DATABASE}.${COLLECTION}
2023-08-23T19:11:44.662-0300   building a list of collections to restore from dump\ligado dir
2023-08-23T19:11:44.662-0300   reading metadata for ligado2.albuns from dump\ligado\albuns.metadata.json
2023-08-23T19:11:44.895-0300   restoring ligado2.albuns from dump\ligado\albuns.bson
2023-08-23T19:11:44.922-0300   finished restoring ligado2.albuns (6 documents, 0 failures)
2023-08-23T19:11:44.922-0300   no indexes to restore for collection ligado2.albuns
2023-08-23T19:11:44.922-0300   6 document(s) restored successfully. 0 document(s) failed to restore.

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

# VERIFICANDO A IMPORTAÇÃO

Digite o comando: **show dbs**



```
λ Cmder
> show dbs
admin      0.000GB
config     0.000GB
ligado2    0.000GB
local      0.000GB
>
```

The screenshot shows a Windows command prompt window titled 'Cmder' running the MongoDB command 'show dbs'. The output lists four databases: 'admin' (0.000GB), 'config' (0.000GB), 'ligado2' (0.000GB), and 'local' (0.000GB). The 'ligado2' entry is highlighted with a red rectangular box. The taskbar at the bottom shows the 'mongo.exe' application is open, and the system tray includes a search bar and several utility icons.

# RESUMO DE COMANDOS DA AULA

- **MOSTRA BANCOS**

```
show dbs
```

- **CRIANDO UM BANCO**

```
use bancoteste
```

- **MOSTRA BANCOS**

```
show dbs
```

(o banco criado não aparece pois não tem nenhuma collection criada nele)

- **VERIFICANDO EM QUE BD ESTOU**

```
db
```

- **CRIANDO UMA COLLECTION**

uma collection é como se fosse cada registro de um BD relacional. Aqui chamamos de documentos

```
db.documentos.insertOne({nome:"Anderson",idade:48})
```

- **FAZENDO UMA PESQUISA SIMPLES**

```
db.documentos.find()
```

# RESUMO DE COMANDOS DA AULA

- CRIANDO OUTRA COLLECTION

```
db.documentos.insertOne({nome:"Fulano",idade:48,profissao:"PROFESSOR"})
```

- FAZENDO UMA PESQUISA SIMPLES

```
db.documentos.find()
```

- FAZENDO UMA PESQUISA POR NOME

```
db.documentos.find({nome:"Fulano"})
```

- VISUALIZANDO OS DADOS INSERIDOS

```
db.documentos.find()
```

- MELHORANDO A VISUALIZAÇÃO

```
db.documentos.find().pretty()
```

- FAZENDO UM BKP DO BANCO DE DADOS

```
criar uma pasta, entrar pelo cmd nesta pasta e executar o comando:  
mongodump --db bancoteste
```

# RESUMO DE COMANDOS DA AULA

## - APAGANDO UM BANCO DE DADOS

entrar no prompt do mongo e selecionar o banco desejado  
use bancoteste

em seguida usar o comando  
`db.dropDatabase()`

## - IMPORTANDO UM BANCO DE DADOS

entrar em um prompt de comando do windows na pasta onde esta o BKP do Banco  
`mongorestore --db bancoteste2 dump/bancoteste`

**REALIZAR AS ETAPAS ANTERIORES E VERIFICAR OS ARQUIVOS IMPORTADOS NOVAMENTE**