Banco de Dados II

Bancos de Dados Não-Convencionais

Bancos de Dados Não Convencionais

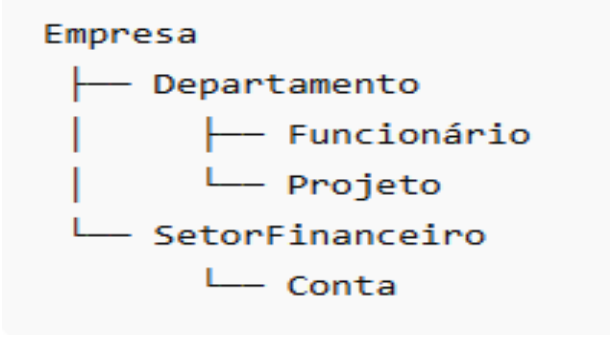
- **Bancos de dados não convencionais** são **sistemas de gerenciamento de dados que fogem do modelo tradicional relacional (baseado em tabelas, linhas e colunas) e que surgiram para atender necessidades que os bancos de dados relacionais não conseguem resolver de forma eficiente** — como:
  - 1) **lidar com grandes volumes de dados**,
  - 2) **dados complexos**,
  - 3) **não estruturados** ou
  - 4) **com relacionamentos dinâmicos**.

Histórico de Desenvolvimento de Bancos de Dados

- Historicamente, **três principais modelos de dados** (ou **paradigmas**) foram desenvolvidos **antes da popularização dos bancos não convencionais:**
  - **Modelo Hierárquico**
  - **Modelo em Rede**
  - **Modelo Relacional**
- **Cada um representa uma forma distinta de organizar, armazenar e acessar os dados.**

Modelo Hierárquico

- **Década de 1960 (início) .**
- **Surgiu nos primeiros grandes sistemas corporativos, quando empresas e governos começaram a usar mainframes para armazenar dados organizados em estruturas simples e fixas.**
- Exemplo: **IBM IMS** (Information Management System, 1968).
- O **modelo hierárquico** organiza os dados em **uma estrutura de árvore** — cada registro (ou nó) tem um único pai e um ou mais filhos.
- Na imagem abaixo, **EMPRESA** corresponde à raiz da árvore. **DEPARTAMENTO** e **SETORFINANCEIRO** são seus filhos. Por sua vez, **FUNCIONARIO** e **PROJETO** são os filhos de **DEPARTAMENTO** enquanto **CONTA** é filho de **SETORFINACEIRO**.
- Para chegar em **PROJETO**, por exemplo, devemos navegar antes por **EMPRESA** e **DEPARTAMENTO**.

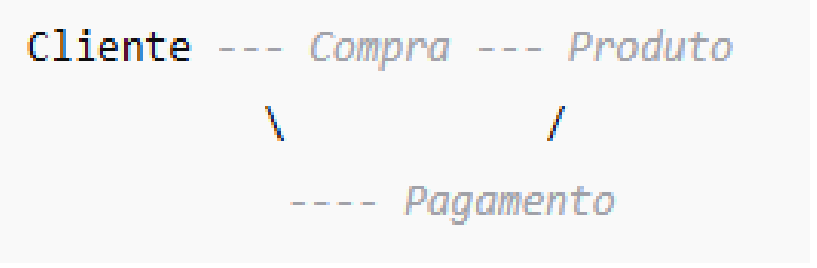


Modelo Hierárquico

- **Características do modelo hierárquico:**
  - **Acesso rápido por caminhos pré-definidos.**
  - **Relação um-para-muitos (1:N).**
  - **Cada dado deve pertencer a uma única hierarquia.**
- **Limitações do modelo hierárquico:**
  - **Dificuldade de representar relacionamentos muitos-para-muitos (N:M).**
  - **Estrutura rígida e difícil de modificar.**
  - **A navegação depende de percorrer toda a hierarquia.**

Modelo Em Rede

- **Final da década de 1960 a 1970.**
- O **modelo em rede** foi **criado para superar as limitações do modelo hierárquico, permitindo relacionamentos mais complexos (N:M).** Formalizado pelo grupo **CODASYL** em **1971**.
- Exemplos: **IDMS (Integrated Database Management System)**, **TurboIMAGE** e **Univac DMS-1100**.
- **Permite que um registro tenha vários pais e vários filhos, formando uma estrutura de grafo (não apenas uma árvore).**



Modelo Em Rede

- **Características do modelo em rede:**
  - **Relações muitos-para-muitos (N:M) possíveis.**
  - **Acesso a dados feito por navegação de ponteiros (links).**
  - **Mais flexível que o modelo hierárquico.**
  - **Baseado na especificação CODASYL (Conference on Data Systems Languages).**

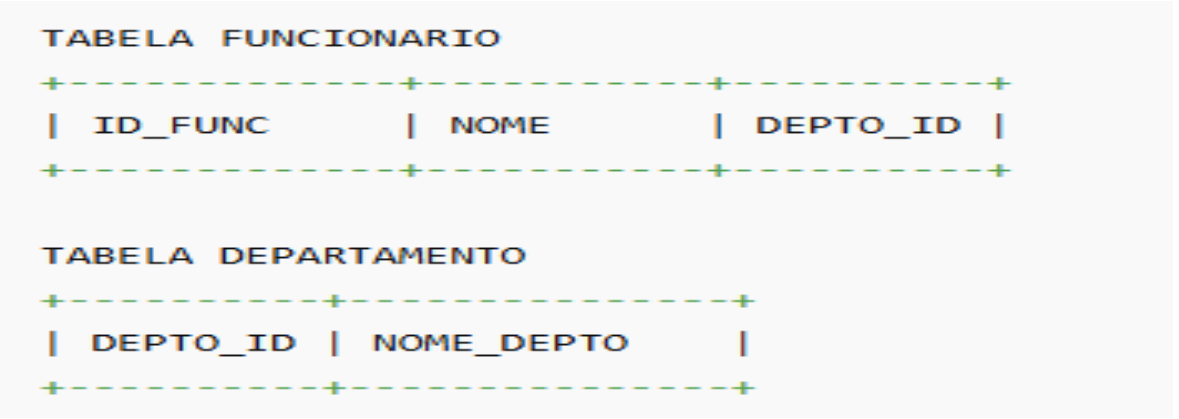
Modelo Em Rede

- **Limitações:**
  - **Linguagem de manipulação de dados complexa e procedural (necessário especificar caminhos).**
  - **Alta dependência da estrutura física dos dados.**

Modelo Relacional

- **Década de 1970 (proposto em 1970, popularizado nos anos 1980) .**
- **Proposto por Edgar F. Codd (IBM) em seu artigo clássico “A Relational Model of Data for Large Shared Data Banks” (1970). Base matemática sólida (Teoria dos Conjuntos), priorizando independência e flexibilidade.**
- **Exemplos: Oracle (1979), IBM DB2 (1983), Ingres.**
- **Organiza os dados em tabelas (relações) compostas por linhas (tuplas) e colunas (atributos), baseando-se em princípios matemáticos da álgebra relacional.**

Modelo Relacional



- **Características do Modelo Relacional:**
  - **Relações entre tabelas por chaves primárias e estrangeiras.**
  - **Linguagem de consulta declarativa (SQL) — o usuário diz o que quer, não como buscar.**
  - **Independência lógica e física dos dados.**
  - **Base teórica sólida (E. F. Codd, 1970).**

Modelo Relacional

- **Limitações do Modelo Relacional:**
  - **Dificuldade em representar dados complexos (imagens, multimídia, objetos).**
  - **Menor desempenho em alguns contextos de Big Data ou relacionamentos altamente conectados.**
- **Exemplo de SGBD: PostgreSQL, Oracle, MySQL, SQL Server.**

Modelo Relacional

- O **DB2** da **IBM** surgiu a partir do **projeto System R** da própria **IBM**, iniciado em **1974** **para testar o modelo relacional proposto por Edgar F. Codd**.
- Em **1977**, o projeto foi **comercializado** como um **banco de dados relacional** e, em **1981**, foi **renomeado** para **IBM SQL/DS**. O nome **DB2** (Database 2) foi dado ao produto lançado oficialmente em **1983** para o **sistema mainframe**.
- 1970s: Edgar F. Codd publica um artigo definindo o modelo relacional de banco de dados.
- 1974: A IBM inicia o projeto System R para implementar as ideias de Codd.
- 1977: O System R é comercializado pela primeira vez.
- 1981: A IBM lança o IBM SQL/DS, uma versão comercializada do projeto.
- 1983: O produto é renomeado para IBM DB2 e lançado oficialmente para mainframes.

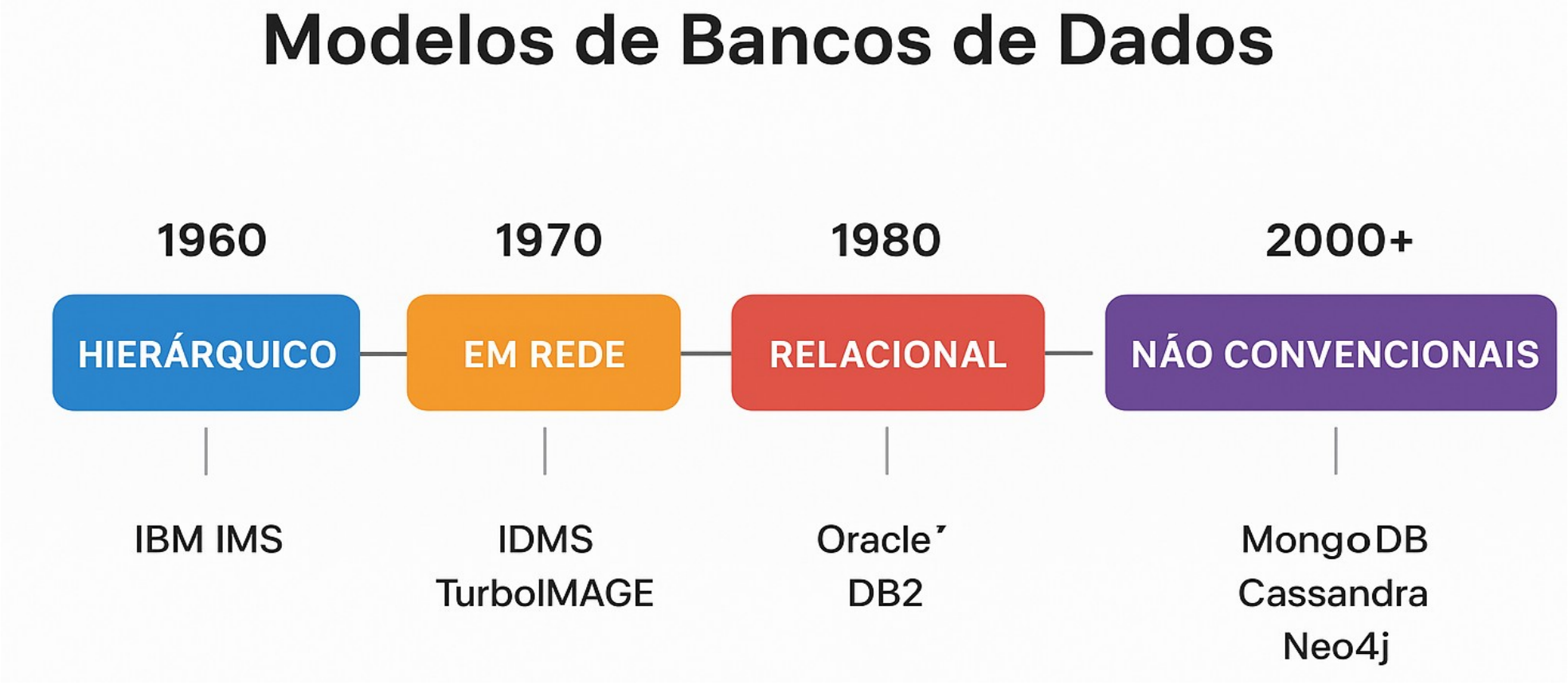
Modelo Orientado a Objetos

- **Década de 1980–1990.**
- **Surgiu com a popularização da Programação Orientada a Objetos (POO), buscando armazenar objetos complexos com atributos e métodos.**
- **Exemplos: ObjectDB, db4o, Versant ODBMS.**
- **Tentativa de integrar bancos com linguagens como Java e C++.**
- **Maior capacidade de representar objetos complexos.**
- **Não substituiu o modelo relacional, mas influenciou o surgimento dos bancos objeto-relacionais (como o PostgreSQL).**

Bancos de Dados Não Convencionais

- Um **banco de dados não convencional** (ou **não tradicional**) é aquele cujo modelo de dados, mecanismos de armazenamento e processamento **não seguem o paradigma relacional clássico**, proposto por ***E. F. Codd***, mas sim **outros paradigmas projetados para lidar com:**
  - **Estruturas de dados complexas ou heterogêneas;**
  - **Altos volumes de dados (Big Data);**
  - **Dados multimídia, geográficos, temporais ou semiestruturados;**
  - **Alta escalabilidade e disponibilidade em sistemas distribuídos.**
- **Surgiram no início do século 21.**

Bancos de Dados Não Convencionais



Bancos de Dados Não Convencionais

- **1. Bancos de Dados Orientados a Documentos:**
- **Exemplo: MongoDB**
- **Modelo de dados: Documentos no formato JSON/BSON (semiestruturado).**
- **Características:**
- **Armazena informações em coleções de documentos, não em tabelas.**
- **Esquema flexível — documentos de uma mesma coleção podem ter campos diferentes.**
- **Alta escalabilidade horizontal (sharding).**
- **Consultas poderosas via linguagem própria semelhante ao JSON.**
- **Aplicações típicas: sistemas web, catálogos de produtos, aplicações em nuvem.**
- **Outros exemplos: CouchDB, RavenDB.**

Bancos de Dados Não Convencionais

- **2. Bancos de Dados Chave–Valor.**
- **Exemplo: Redis**
- **Modelo de dados: Pares chave → valor, geralmente armazenados na memória.**
- **Características:**
- **Extremamente rápido (opera na RAM).**
- **Suporte a tipos de dados simples (string, lista, conjunto, hash, etc.).**
- **Ideal para cache, filas e sessões de usuário.**
- **Pode ser usado como banco persistente, mas é mais comum em funções de alto desempenho.**
- **Aplicações típicas: sistemas de recomendação, controle de sessão, contadores em tempo real.**
- **Outros exemplos: Amazon DynamoDB, Riak, Berkeley DB.**

Bancos de Dados Não Convencionais

- **3. Bancos de Dados em Grafos**
- **Exemplo: Neo4j**
- **Modelo de dados: Nós (entidades) e arestas (relacionamentos).**
- **Características:**
- **Ideal para modelar redes complexas — sociais, logísticas, biológicas, etc.**
- **Consultas feitas por linguagem declarativa Cypher, que expressa facilmente relações como “amigos de amigos”.**
- **Permite consultas profundas com excelente desempenho.**
- **Aplicações típicas: redes sociais, detecção de fraudes, recomendação, roteamento.**
- **Outros exemplos: ArangoDB, OrientDB, JanusGraph.**

Bancos de Dados Não Convencionais

- **4. Bancos de Dados Orientados a Colunas**
- **Exemplo: Apache Cassandra**
- **Modelo de dados: Armazenamento baseado em famílias de colunas, não em linhas.**
- **Características:**
- **Altamente distribuído e tolerante a falhas.**
- **Excelente para grandes volumes de dados em clusters.**
- **Baseado em arquitetura peer-to-peer (sem nó mestre).**
- **Inspirado no Bigtable (Google) e Dynamo (Amazon).**
- **Aplicações típicas: Big Data, telemetria, logs, IoT.**
- **Outros exemplos: HBase, ScyllaDB.**

MongoDB

- Estudo de alguns **Bancos de Dados Não-Convencionais (BDNCs)**, também chamados de:
  - **Bancos de Dados NoSQL,**
  - **Bancos de Dados Não Relacionais e**
  - **Bancos de Dados Pós-Relacionais.**
  - Estes BDs visam atender as necessidades de gerenciamento de dados de aplicações ditas não-convencionais.

MongoDB

- **Conhecer os diferentes tipos de bancos de dados é importante para entender qual infraestrutura é a mais indicada para determinado negócio.**
- **Os bancos de dados NoSQL (não-relacionais) surgiram a partir da necessidade de utilizar soluções mais fáceis de ampliar ou reduzir a infraestrutura conforme as empresas crescem.**

MongoDB

- O termo NoSQL foi utilizado pela primeira vez em 1998 por Carlo Strozzi, ao falar sobre um banco de dados não relacional de código aberto.
- O termo foi novamente utilizado em 2006, quando a empresa Google publicou um artigo sobre o armazenamento de dados.
- Já em 2009, um colaborador do Rackspace organizou um evento para tratar de bancos de dados open source distribuídos e novamente citou o termo NoSQL. Naquela época, começaram a surgir mais tecnologias com bancos de dados não relacionais e o tema foi se popularizando.

- A facilidade para processar informações, escalar a infraestrutura com menor custo foi tornando este tipo de banco de dados popular entre as empresas.

MongoDB

- **Principais Bancos de Dados Não-Convencionais:**
  - **Redis**
  - **Memcached**
  - **Cassandra**
  - **Hbase**
  - **Amazon DynamoDB**
  - **Neo4j**
  - **MongoDB**

MongoDB

- **O MongoDB é um banco de dados orientado a documentos que possui código aberto (*open source*) e foi projetado para armazenar uma grande escala de dados, além de permitir que se trabalhe de forma eficiente com grandes volumes.**
- **Ele é categorizado como um banco de dados NoSQL (*not only SQL*) pois o armazenamento e a recuperação de dados no MongoDB não são feitas no formato de tabelas.**
- **O banco de dados também fornece suporte oficial de driver para todas as linguagens populares como C, C ++, C # e .Net, Go, Java, Node.js, Perl, PHP, Python, Motor, Ruby, Scala, Swift, Mongoid. Assim, pode-se criar um aplicativo usando qualquer uma dessas linguagens.**

MongoDB

- **MongoDB é um banco de dados de código aberto, gratuito, de alta performance, sem esquemas e orientado a documentos .**
- **Foi lançado em fevereiro de 2009 pela empresa 10gen.**
- **Foi escrito na linguagem de programação C++ e seu desenvolvimento durou quase 2 anos, tendo iniciado em 2007.**

MongoDB

- **Por ser orientado a documentos JSON (armazenados em modo binário, apelidado de BSON), muitas aplicações podem modelar informações de modo muito mais natural, pois os dados podem ser aninhados em hierarquias complexas e continuar a ser indexáveis e fáceis de buscar, igual ao que já é feito em JavaScript.**

MongoDB

- **MongoDB foi criado tendo em mente o conceito de Big Data.**
- **Ele suporta tanto escalonamento horizontal quanto vertical usando replica sets (instâncias espelhadas) e sharding (dados distribuídos), tornando-o uma opção muito interessante para grandes volumes de dados, especialmente os desestruturados.**
- **Dados desestruturados (com *Schema* variável) são um problema para a imensa maioria dos bancos de dados relacionais, mas não tanto para o MongoDB.**
- **Os documentos BSON (JSON binário) são *schemaless* e aceitam quase qualquer coisa que você quiser armazenar, sendo um mecanismo de persistência perfeito para uso com tecnologias que trabalham com JSON nativamente, como JavaScript (e consequentemente Node.js).**

MongoDB

- Um documento é um conjunto de pares de valores-chave.
- Documentos têm esquema dinâmico.
- Esquema dinâmico significa que os documentos na mesma coleção não precisam ter o mesmo conjunto de campos ou estrutura e campos comuns em documentos de uma coleção podem conter diferentes tipos de dados.

MongoDB

```
{
   _id:ObjectId(7df78ad8902c)
   title:'MongoDB - Guia Rapido',
   description:'MongoDB - Guia Rapido',by:'MongoDBWise',
   url:'http://www.mongodbwise.worpress.com',
   tags:['mongodb','database','NoSQL'],
   likes:100,
   comments:[{
         user:'user1',
         message:'My first comment',
         dateCreated:newDate(2014,5,21,2,15),
         like:0},{
         user:'user2',
         message:'My second comments',
         dateCreated:newDate(2014,5,21,7,45),
         like:5}]}
```

Exemplo de Documento no MongoDB.

MongoDB

- **Collection é um grupo de documentos MongoDB.**
- **É o equivalente de uma tabela de RDBMS. Assim como Document é equivalente a um registro em bancos de dados relacionais.**
- **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.**

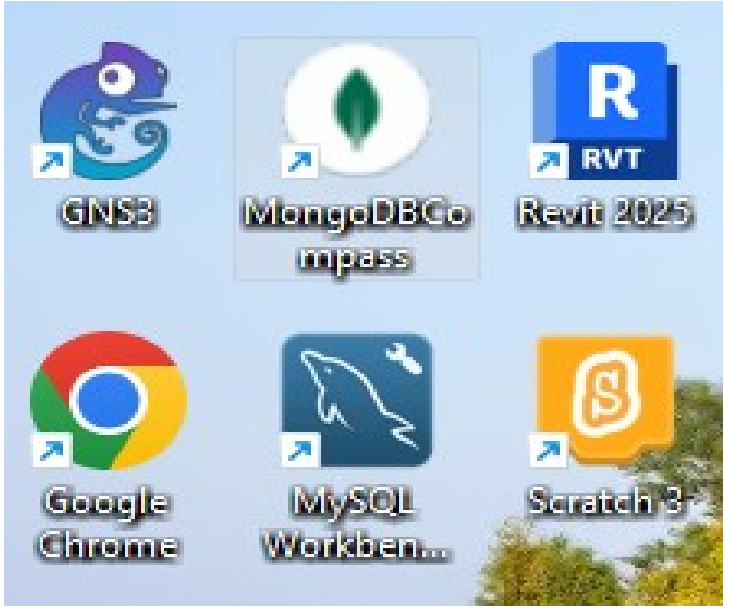
MongoDB

- **Banco de dados é um recipiente físico para coleções.**
- **Cada banco de dados tem o seu próprio conjunto de arquivos no sistema de arquivos.**
- **Um único servidor MongoDB normalmente tem vários bancos de dados.**

MongoDB

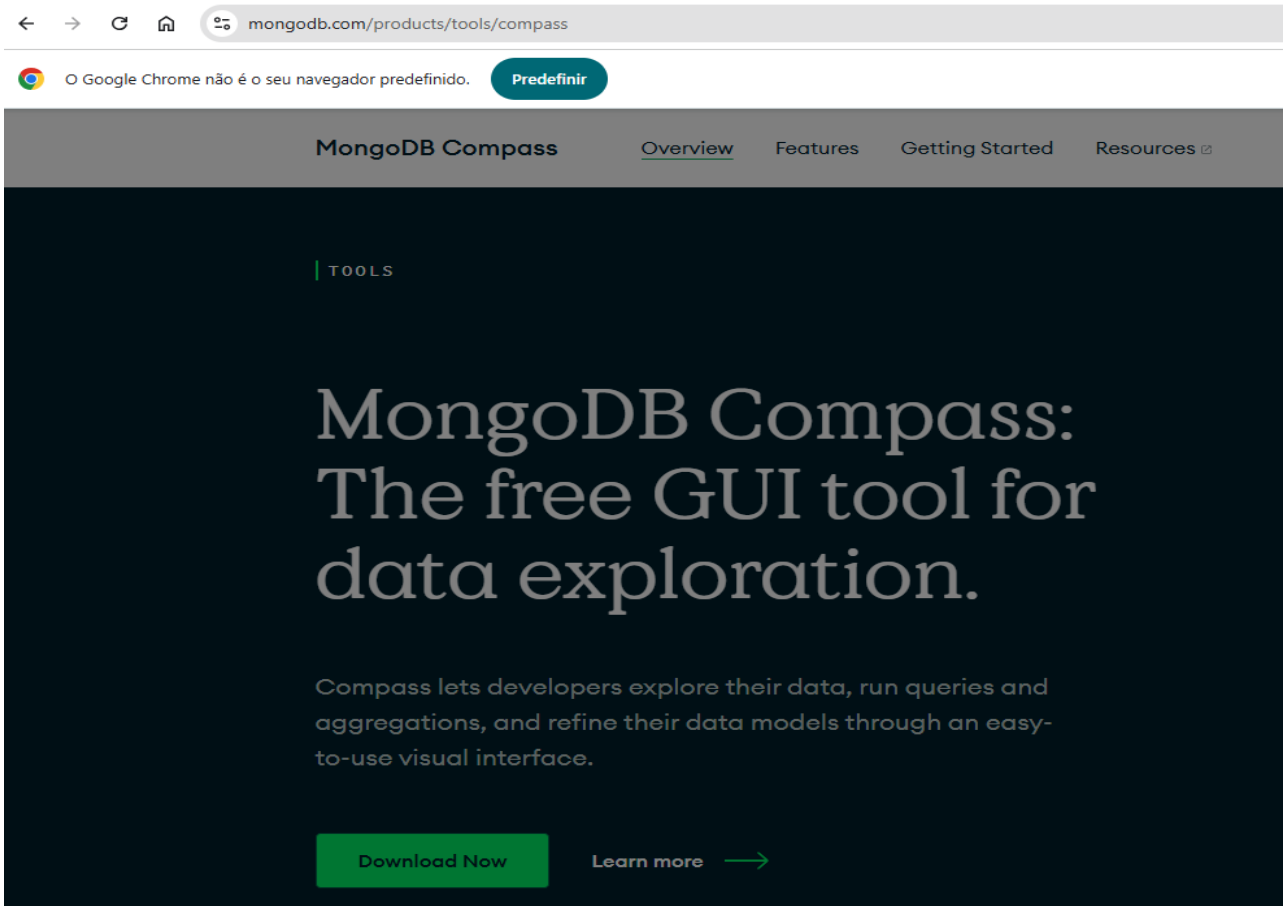
- **MongoDB não deve ser encarado como uma panaceia nem como uma "bala de prata" capaz de resolver qualquer tipo de problema relativo a persistência de dados.**
- **O MongoDB não deve ser utilizado quando relacionamentos entre diversas entidades são importantes para o seu sistema. Se precisar de usar muitas "chaves estrangeiras" e "JOINs", você está usando do jeito errado, ou, ao menos, não do jeito mais indicado.**
- **Além disso, diversas entidades de pagamento (como bandeiras de cartão de crédito) não homologam sistemas cujos dados financeiros dos clientes não estejam em bancos de dados relacionais tradicionais.**
- **Obviamente isso não impede completamente o uso de MongoDB em sistemas financeiros, mas o restringe apenas a certas partes (como dados públicos).**

MongoDB

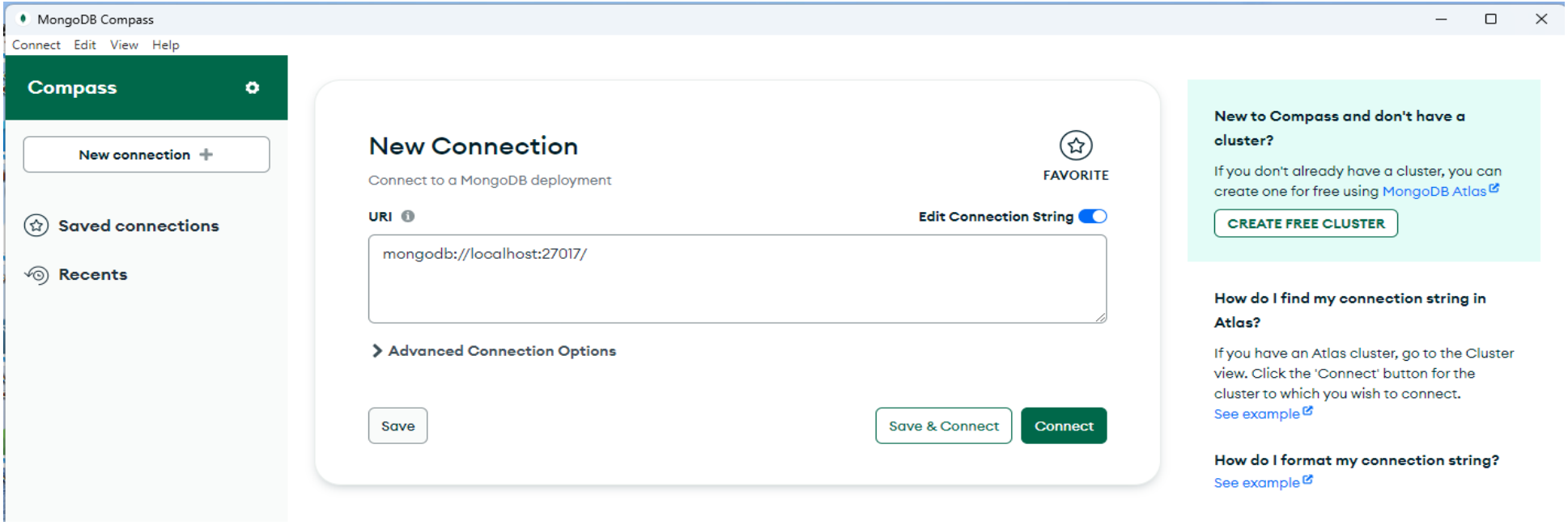


- Utilizaremos o **MongoDB Compass**.
- **O que é MongoDB Compass?**
- O **MongoDB Compass** é uma **interface gráfica** poderosa para **query**, **aggregation** e **análise de seus dados MongoDB** em um **ambiente visual**.
- O **Compass** é **gratuito** para uso e tem fonte disponível e pode ser executado no **macOS**, **Windows** e **Linux**.

MongoDB

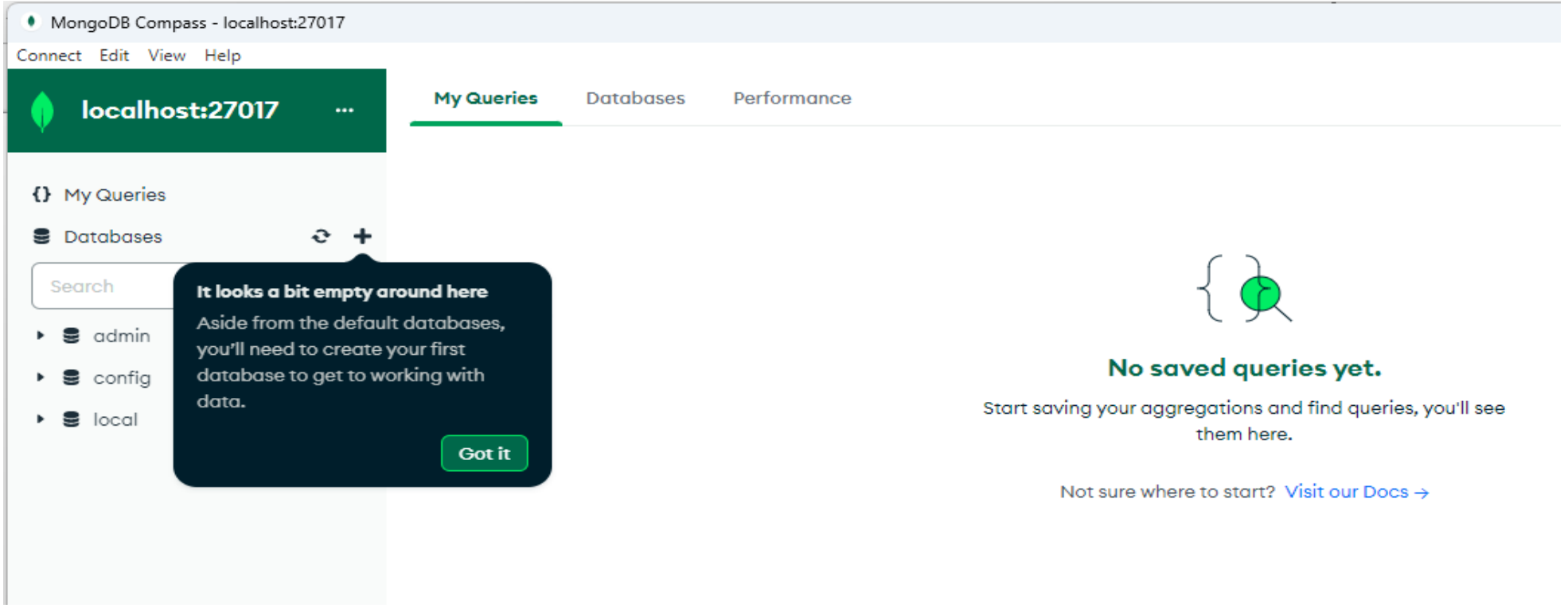

mongodb.com/products/tools/compass
O Google Chrome não é o seu navegador predefinido.
Predefinir
MongoDB Compass
Overview
Features
Getting Started
Resources
TOOLS
MongoDB Compass: The free GUI tool for data exploration.
Compass lets developers explore their data, run queries and aggregations, and refine their data models through an easy-to-use visual interface.
Download Now
Learn more

MongoDB



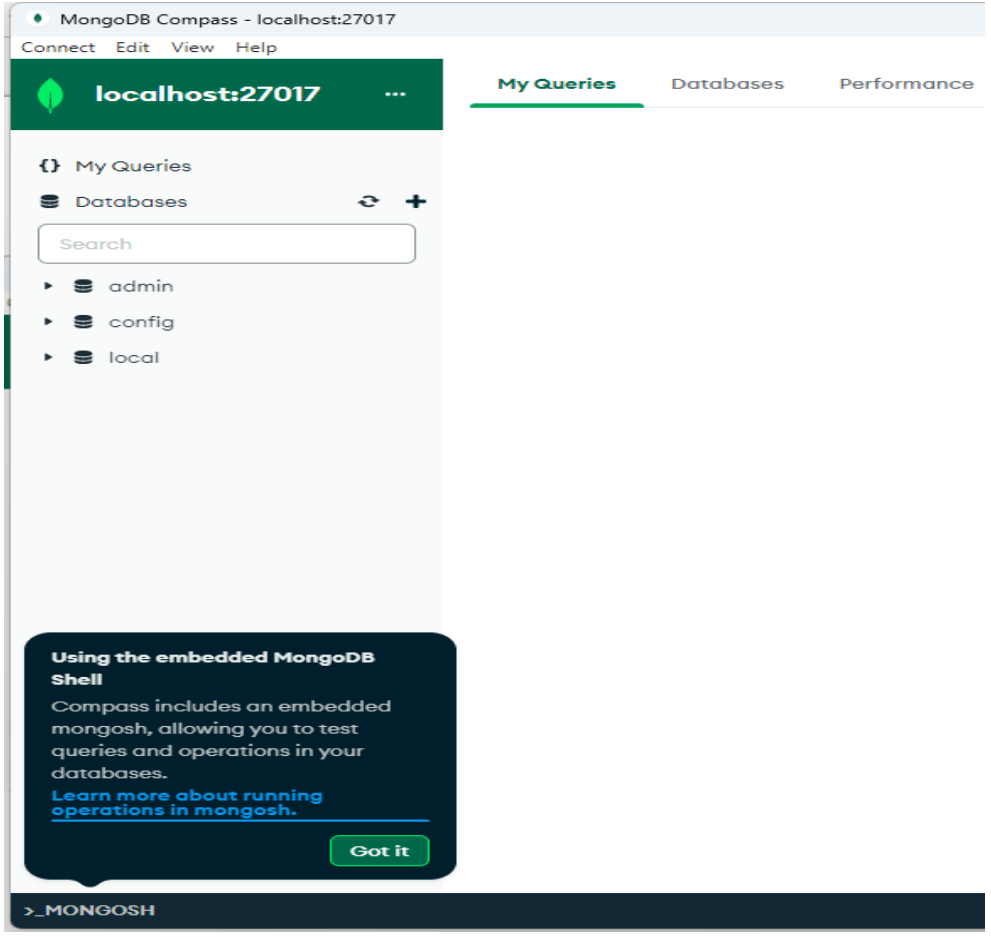
- Informe a **URI** para a conexão com o **servidor MongoDB** e clique no botão **CONNECT**.

MongoDB



- **Clique em “+” à direita de DATABASES.**

MongoDB



- **Mongosh** é o moderno **MongoDB Shell**, **um ambiente interativo JavaScript e Node.js REPL para interagir com implantações do MongoDB.** Ele é **usado para se conectar a um servidor MongoDB, seja uma instância local, um cluster Atlas ou outro host remoto, e então executar operações como:**

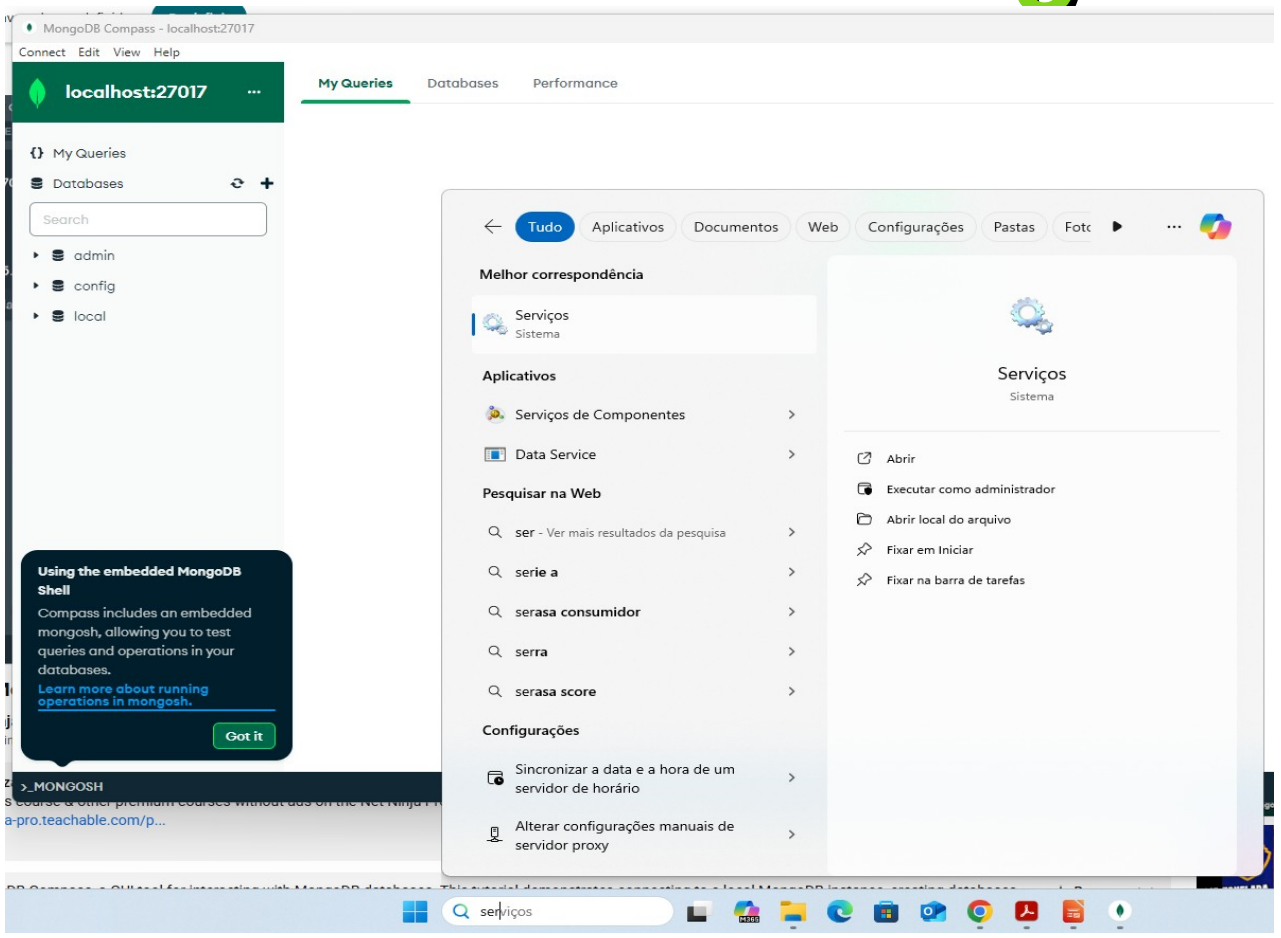
- **Manipulação de dados:** criação, leitura, atualização e exclusão de documentos dentro de coleções.

- **Consulta:** execução de consultas e agregações complexas para analisar dados.

- **Administração de banco de dados:** gerenciar usuários, funções, criar índices e executar outras tarefas administrativas.

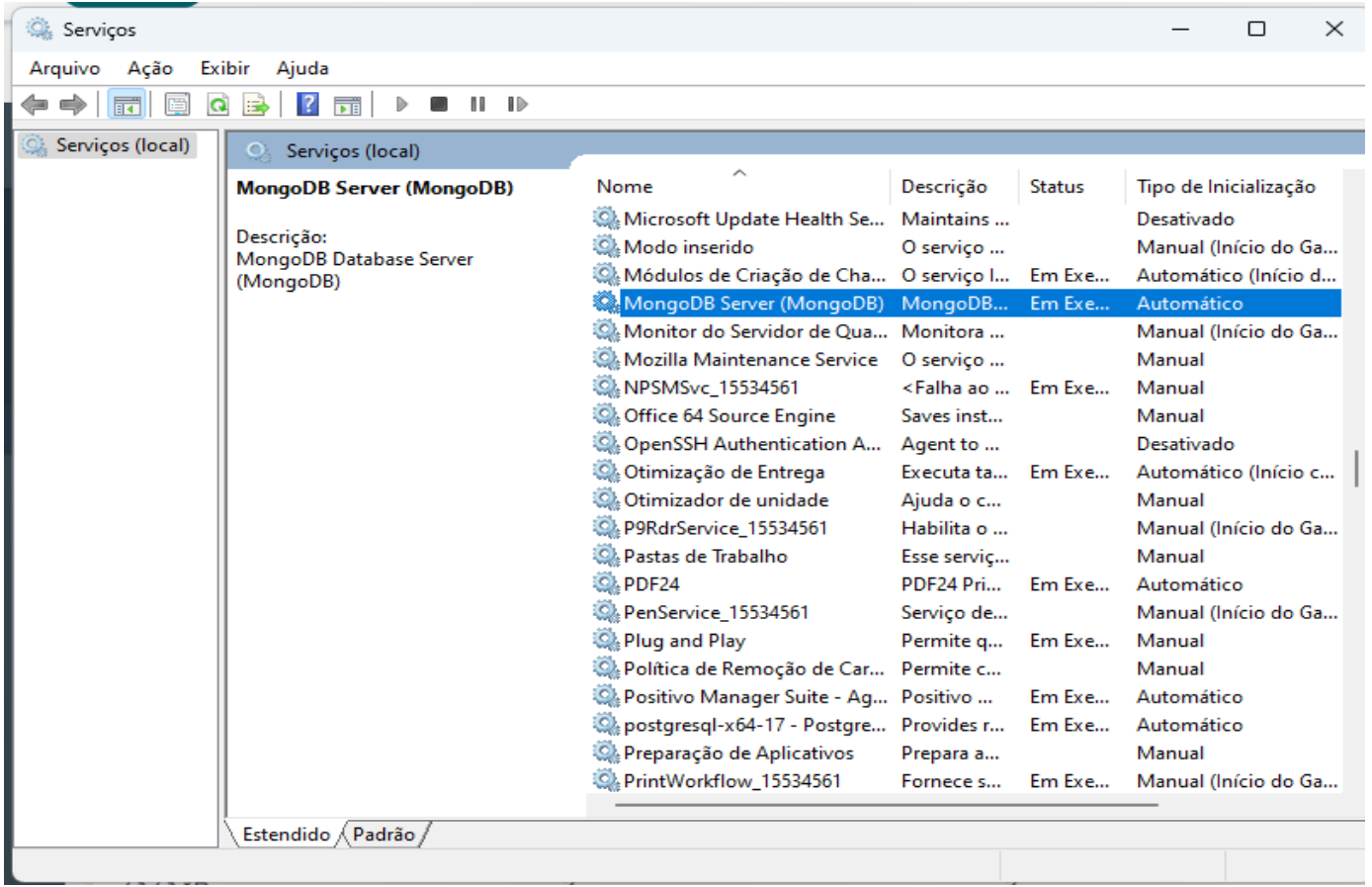
- **Scripting:** automatização de tarefas repetitivas escrevendo e executando scripts JavaScript.

MongoDB



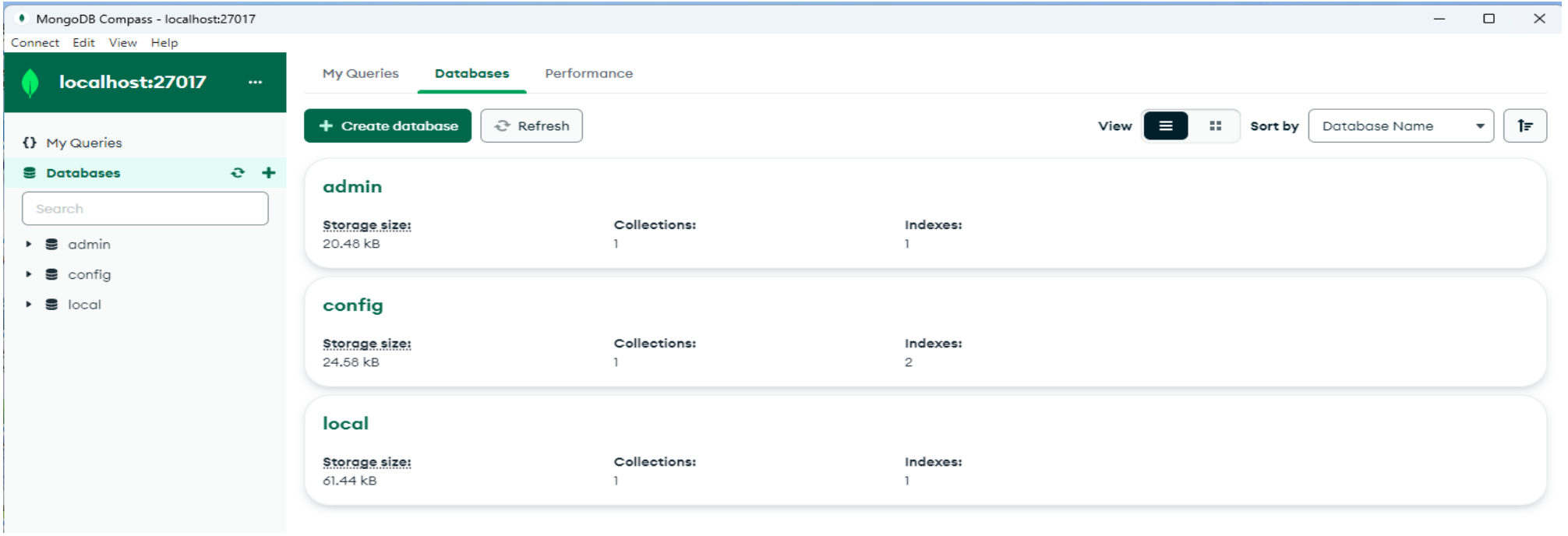
- Procure por **serviços** no **Windows**.
- Isso para nos **certificarmos** de que o **serviço MongoDB** está em **execução**.

MongoDB



- Localizamos o **serviço MongoDB Server**.
- Ele está **em execução** (Status).

MongoDB

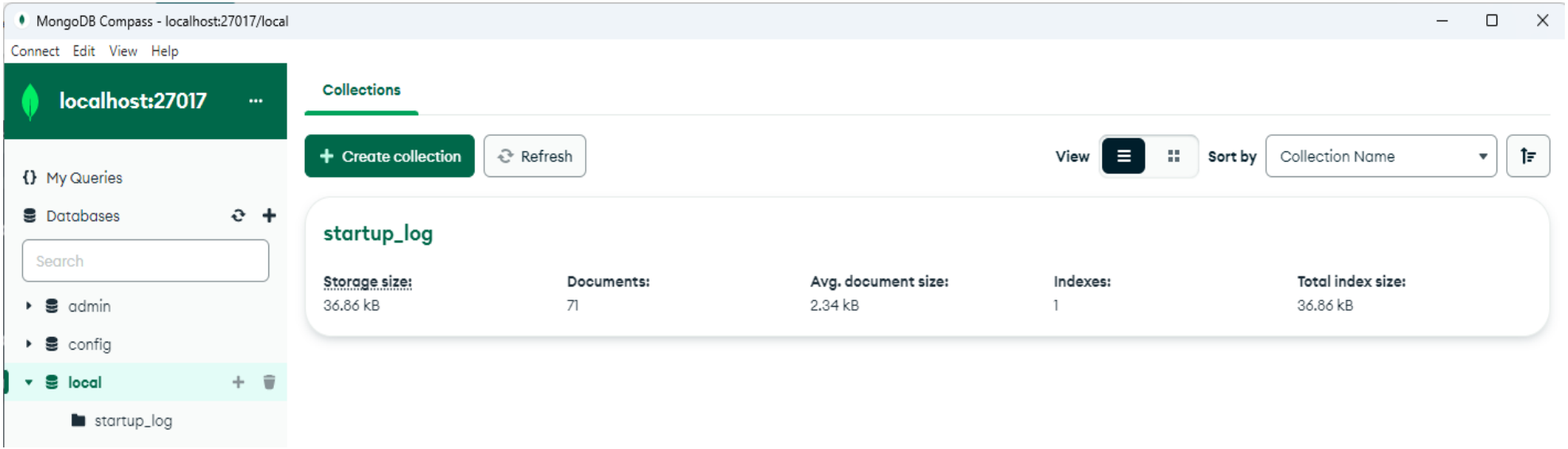


- **Clique** em **Databases** para **visualizar** os **bancos de dados** já existentes em sua **instalação**.

MongoDB

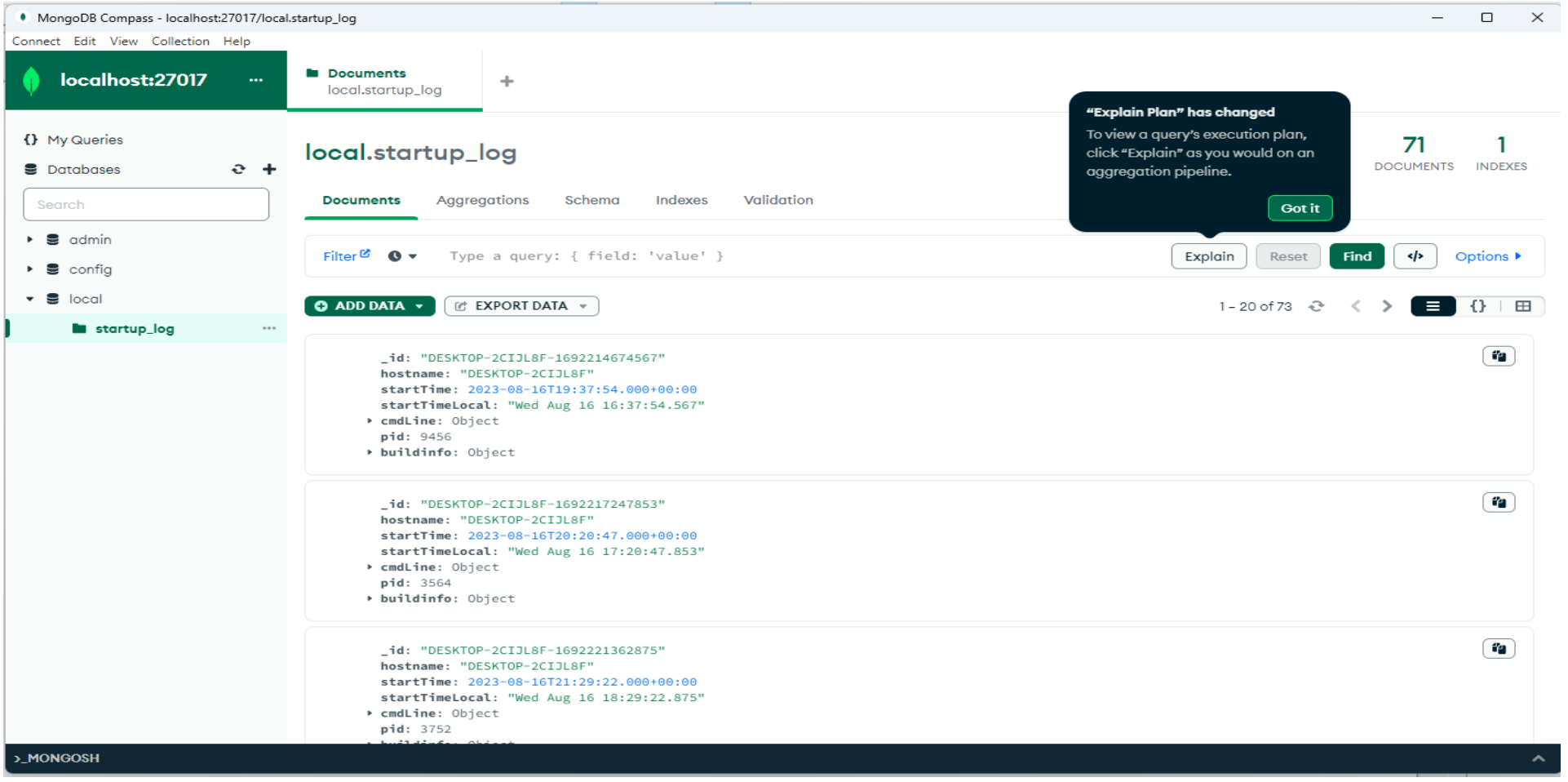
- Os bancos **admin**, **config** e **local** não foram criados por nós.
- São **bancos “pré-fabricados”** pelo próprio **MongoDB**.
- O banco de dados **local**, por exemplo, **contém dados de log de inicialização**.
- Você pode acessá-lo clicando sobre seu nome.

MongoDB

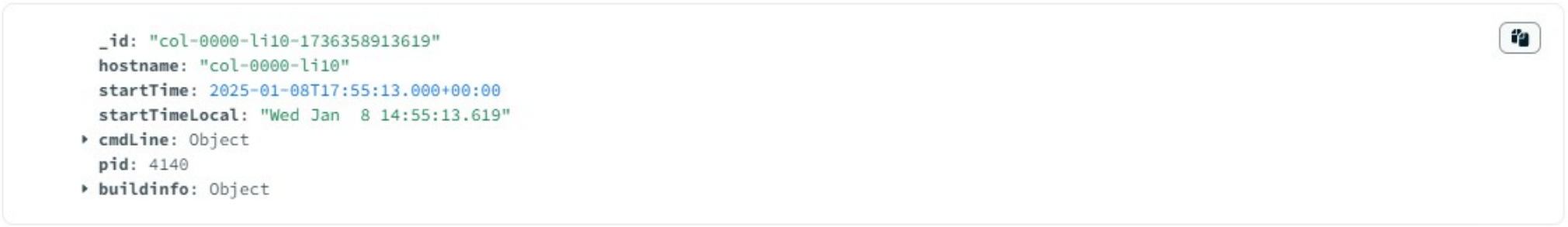


- O **banco de dados local**, conforme apresentado logo acima, **possui uma única coleção: startup_log**. Clique sobre seu nome.

MongoDB

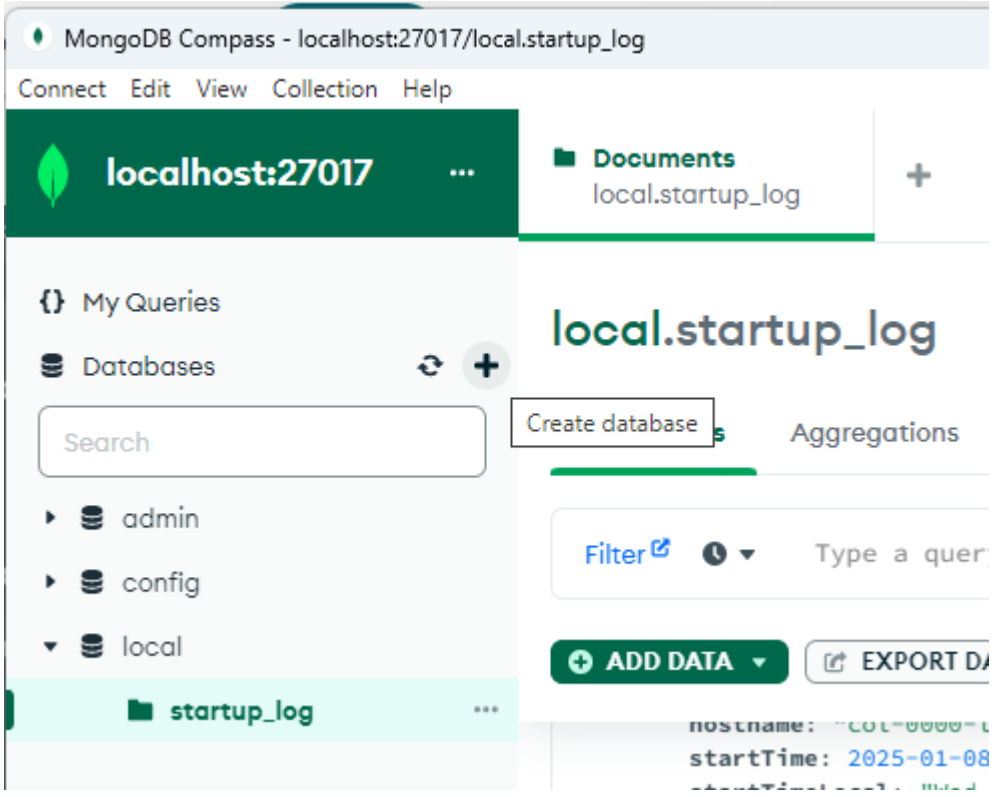

MongoDB Compass - localhost:27017/local.startup_log
Connect Edit View Collection Help
localhost:27017
Documents
local.startup_log
My Queries
Databases
Search
admin
config
local
startup_log
local.startup_log
Documents Aggregations Schema Indexes Validation
"Explain Plan" has changed
To view a query's execution plan, click "Explain" as you would on an aggregation pipeline.
Got it
71 DOCUMENTS
1 INDEXES
Filter
Type a query: { field: 'value' }
Explain Reset Find Options
ADD DATA
EXPORT DATA
1 – 20 of 73
_id: "DESKTOP-2CIJL8F-1692214674567"
hostname: "DESKTOP-2CIJL8F"
startTime: 2023-08-16T19:37:54.000+00:00
startTimeLocal: "Wed Aug 16 16:37:54.567"
cmdLine: Object
pid: 9456
buildinfo: Object
_id: "DESKTOP-2CIJL8F-1692217247853"
hostname: "DESKTOP-2CIJL8F"
startTime: 2023-08-16T20:20:47.000+00:00
startTimeLocal: "Wed Aug 16 17:20:47.853"
cmdLine: Object
pid: 3564
buildinfo: Object
_id: "DESKTOP-2CIJL8F-1692221362875"
hostname: "DESKTOP-2CIJL8F"
startTime: 2023-08-16T21:29:22.000+00:00
startTimeLocal: "Wed Aug 16 18:29:22.875"
cmdLine: Object
pid: 3752
>_MONGOSH

MongoDB



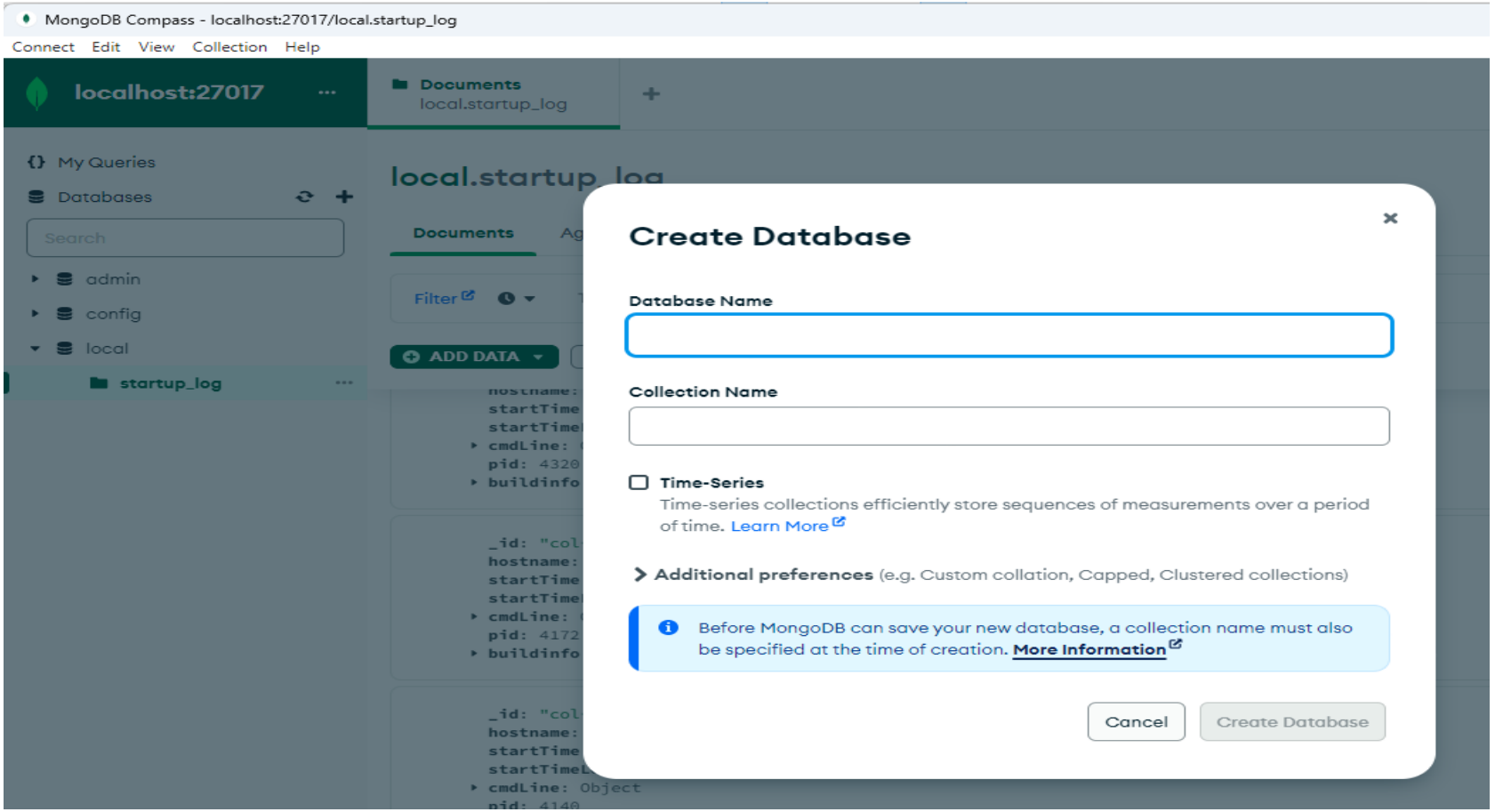
- Dentro desta **coleção** encontramos o seguinte **documento**, apresentado logo acima, que **descreve as ocasiões em que o serviço do MongoDB foi inicializado**.

MongoDB

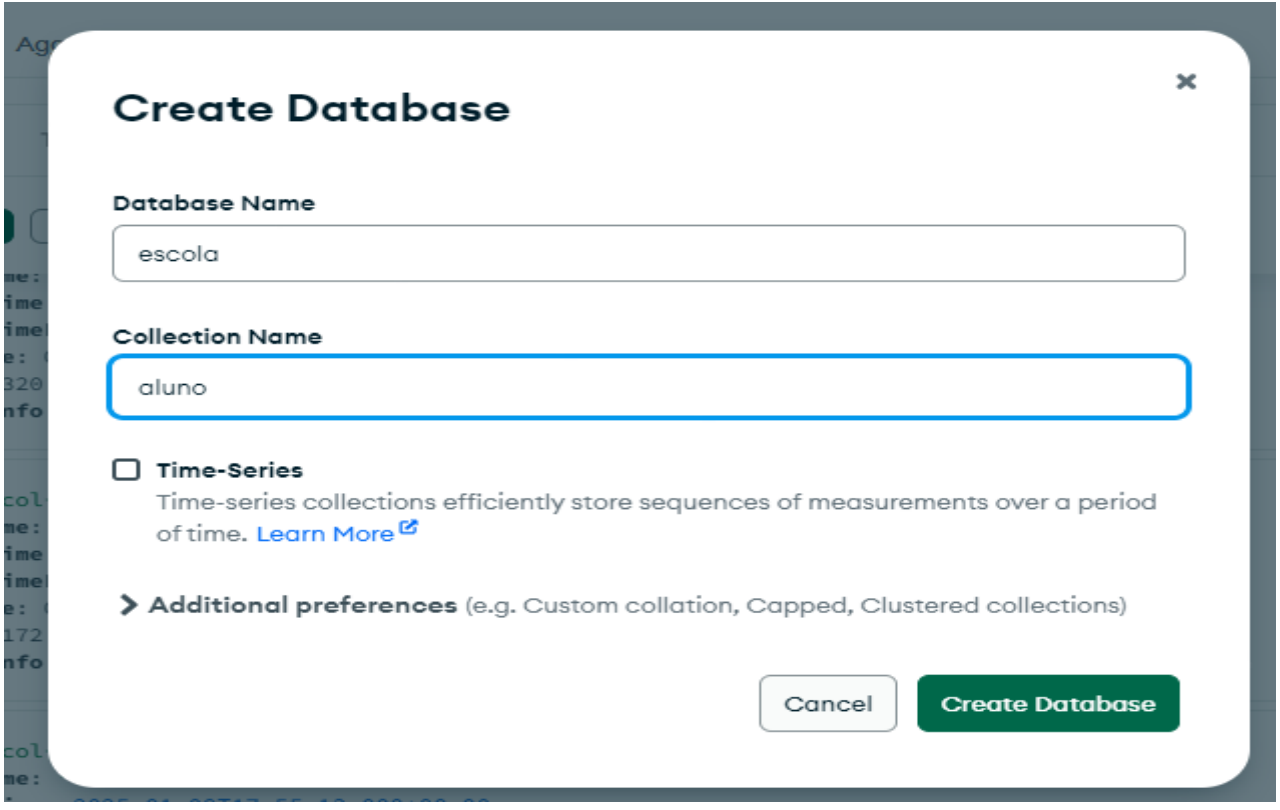


- Para **criar um novo banco de dados** clique no sinal de “**+**” à direita de **Databases**.

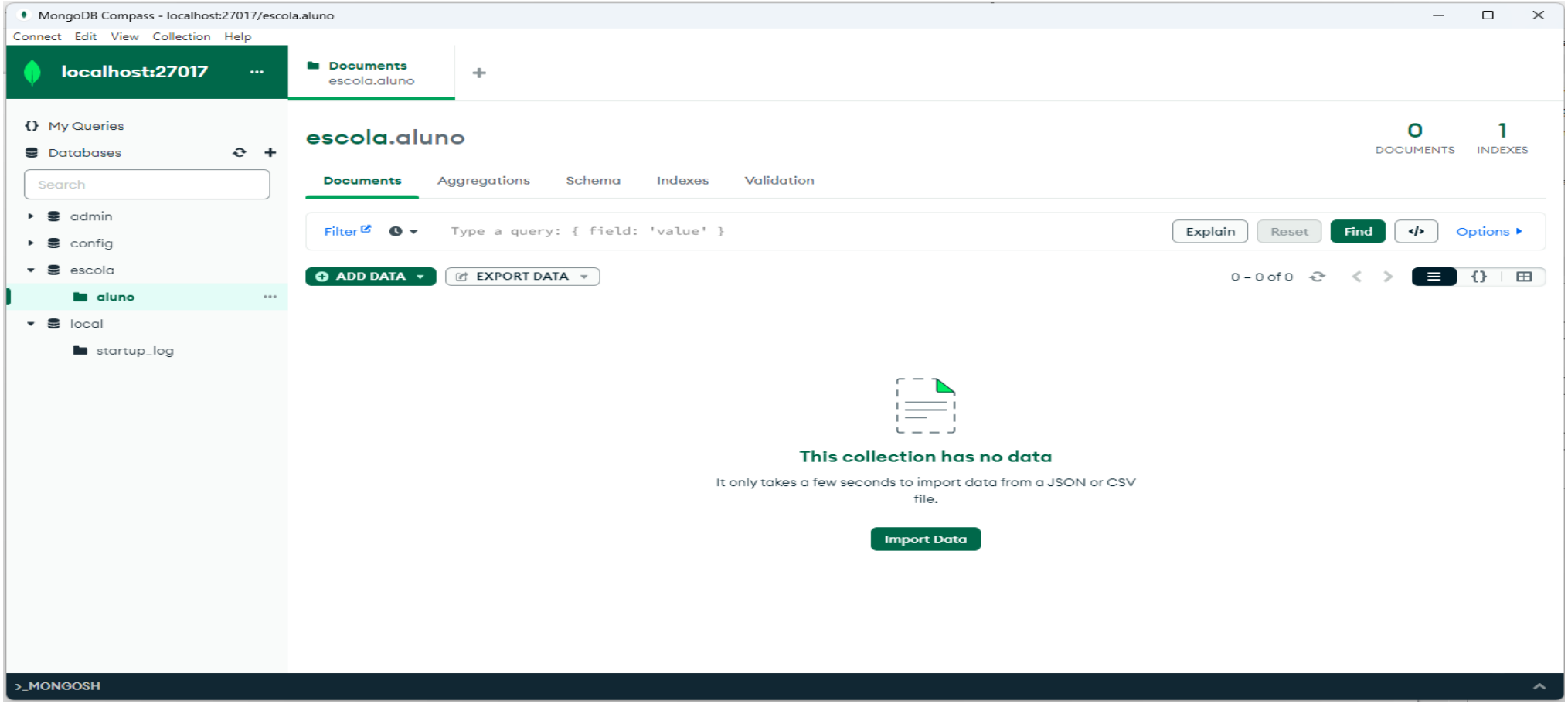
MongoDB

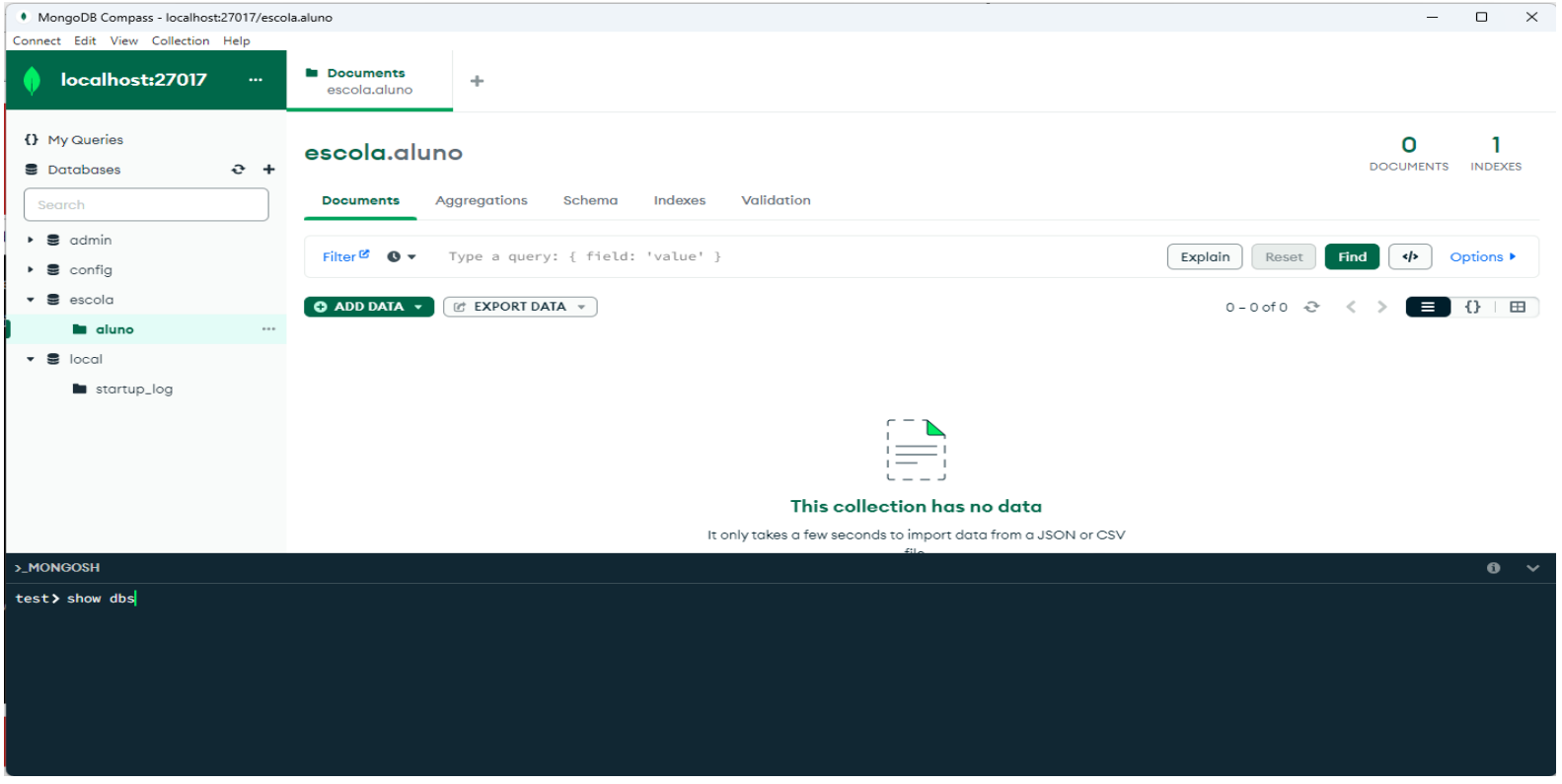

MongoDB Compass - localhost:27017/local.startup_log
Connect Edit View Collection Help
localhost:27017
My Queries
Databases
Search
admin
config
local
startup_log
Documents
local.startup_log
local.startup_log
Documents
Filter
ADD DATA
Create Database
Database Name
Collection Name
Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More
Additional preferences (e.g. Custom collation, Capped, Clustered collections)
Before MongoDB can save your new database, a collection name must also be specified at the time of creation. More Information
Cancel
Create Database

MongoDB


Create Database
Database Name
escola
Collection Name
aluno
Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More
Additional preferences (e.g. Custom collation, Capped, Clustered collections)
Cancel
Create Database

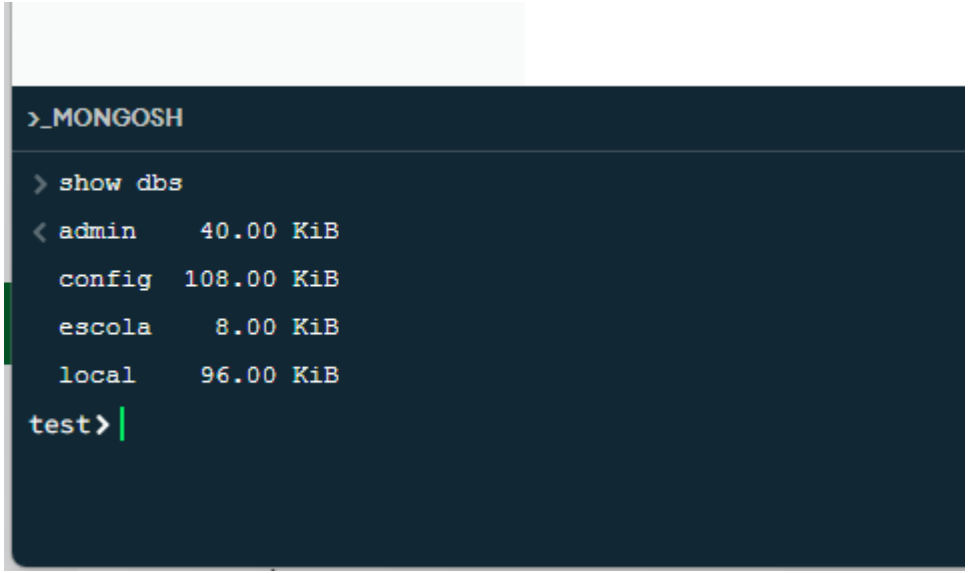
MongoDB


MongoDB Compass - localhost:27017/escola.aluno
Connect Edit View Collection Help
localhost:27017
Documents
escola.aluno
My Queries
Databases
Search
admin
config
escola
aluno
local
startup_log
escola.aluno
0
DOCUMENTS
1
INDEXES
Documents
Aggregations
Schema
Indexes
Validation
Filter
Type a query: { field: 'value' }
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
0 – 0 of 0
This collection has no data
It only takes a few seconds to import data from a JSON or CSV file.
Import Data
>_MONGOSH



- **Clique** sobre o **MONGOSH** no canto inferior do **Compass** e **digite**: **show dbs**. Depois pressione **<ENTER>**.

MongoDB



- Este **comando** **listará todos os databases** (bancos de dados) **existentes** – inclusive o recém criado **escola**.

MongoDB

```
>_MONGOSH
> show dbs
< admin    40.00 KiB
  config  108.00 KiB
  escola    8.00 KiB
  local    96.00 KiB
test> use escola
```

- O **comando use escola fará com que entremos no database escola.**

```
>_MONGOSH
> show dbs
< admin    40.00 KiB
  config  108.00 KiB
  escola    8.00 KiB
  local    96.00 KiB
> use escola
< switched to db escola
escola>
```

MongoDB

- **Se não existir um banco de dados com o nome passado** por **parâmetro** depois do **use**, o **MongoDB** **cria um novo Banco** com o nome referenciado.

MongoDB

```
>_MONGOSH

> show dbs
< admin    40.00 KiB
  config  108.00 KiB
  escola    8.00 KiB
  local    96.00 KiB
> use escola
< switched to db escola
escola> db.aluno.insert({matricula: 1099, nome: 'Sandra Calegari', sexo: 'F', altura: 1.67})
```

- **Comando** para **inserir um novo documento** na **coleção aluno** dentro do **banco de dados escola**.

MongoDB

```
>_MONGOSH
> db.aluno.insert({matricula: 1099, nome: 'Sandra Calegari', sexo: 'F', altura: 1.67})
< DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId("68f8e069933dac4b3d00290e")
    }
  }
escola>
```

- Note que o **comando insert foi depreciado ou desaprovado.** Agora **recomenda-se** o uso de **insertOne**, **insertMany** ou **bulkWrite**.

MongoDB

```
>_MONGOSH
< DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId("68f8e069933dac4b3d00290e")
    }
  }
escola > db.aluno.find()
```

- Agora um **comando** para **listar os documentos da coleção aluno**.

MongoDB

```
>_MONGOSH
> db.aluno.find()
< {
    _id: ObjectId("68f8e069933dac4b3d00290e"),
    matricula: 1099,
    nome: 'Sandra Calegari',
    sexo: 'F',
    altura: 1.67
  }
escola>
```

- Apesar de não recomendar mais o uso do comando insert o **MongoDB** inseriu o registro.

MongoDB

```
>_MONGOSH
}
> db.aluno.insertOne({matricula: 1100, nome: 'Asdrubal Soares Ribeiro', sexo: 'M', altura: 1.72})
< {
    acknowledged: true,
    insertedId: ObjectId("68f90a714c6353124c0ce2eb")
  }
>
>
escola>
```

- Inserindo um **novo documento de aluno** – agora usando **insertOne( )**.

MongoDB

```
>_MONGOSH

> db.disciplina.insertOne({nome: 'Banco de Dados 2', cargaHoraria: 60})
< {
    acknowledged: true,
    insertedId: ObjectId("68f90bbe4c6353124c0ce2ec")
  }
>
>
escola>
```

- **Inserindo um novo documento para a coleção Disciplina (que ainda não foi criada).**

MongoDB

```
>_MONGOSH

> db.disciplina.find()
< {
    _id: ObjectId("68f90bbe4c6353124c0ce2ec"),
    nome: 'Banco de Dados 2',
    cargaHoraria: 60
  }
>
escola>
```

- A coleção disciplina foi criada e o documento foi inserido.

MongoDB

```
>_MONGOSH

> db.aluno.find( )
< {
    _id: ObjectId("68f90a254c6353124c0ce2ea"),
    matricula: 1099,
    nome: 'Sandra Calegari',
    sexo: 'F',
    altura: 1.67
  }
  {
    _id: ObjectId("68f90a714c6353124c0ce2eb"),
    matricula: 1100,
    nome: 'Asdrubal Soares Ribeiro',
    sexo: 'M',
    altura: 1.72
  }
escola>
```

MongoDB

```
>_MONGOSH

> db.aluno.insertOne({matricula: 1101, nome: 'Rowena Santos', sexo: 'F', ama: ['papaia','futebol'], altura: 1.74})
{
  acknowledged: true,
  insertedId: ObjectId("68f90ea94c6353124c0ce2ed")
}

escola>
```

- **Schemaless: o novo documento possui um atributo que nenhum outro documento possuía até aqui (ama) – um vetor com duas ocorrências.**

MongoDB

```
>_MONGOSH

> 
> db.aluno.insertOne({matricula: 1102, nome: 'Carlomano Santos', sexo: 'M', ama: ['vôlei'], altura: 1.77})
{
  acknowledged: true,
  insertedId: ObjectId("68f9104b4c6353124c0ce2ee")
}
escola>
```

MongoDB

```
>_MONGOSH

> db.aluno.insertOne({matricula: 1103, nome: 'Jambira Andrada', sexo: 'F', ama: ['dança','pêssego','rock'], altura: 1.59})

< {
    acknowledged: true,
    insertedId: ObjectId("68f911464c6353124c0ce2ef")
  }

escola>
```

MongoDB

```
>_MONGOSH
> db.aluno.find( )
< {
    _id: ObjectId("68f90a254c6353124c0ce2ea"),
    matricula: 1099,
    nome: 'Sandra Calegari',
    sexo: 'F',
    altura: 1.67
  }
  {
    _id: ObjectId("68f90a714c6353124c0ce2eb"),
    matricula: 1100,
    nome: 'Asdrubal Soares Ribeiro',
    sexo: 'M',
    altura: 1.72
  }
```

MongoDB

```
>_MONGOSH
  {
    _id: ObjectId("68f90ea94c6353124c0ce2ed"),
    matricula: 1101,
    nome: 'Rowena Santos',
    sexo: 'F',
    ama: [
      'papaia',
      'futebol'
    ],
    altura: 1.74
  }
  {
    _id: ObjectId("68f9104b4c6353124c0ce2ee"),
    matricula: 1102,
    nome: 'Carlomano Santos',
    sexo: 'M',
    ama: [
      'vôlei'
    ],
```

MongoDB

```
>_MONGOSH

  {
    _id: ObjectId("68f911464c6353124c0ce2ef"),
    matricula: 1103,
    nome: 'Jambira Andrada',
    sexo: 'F',
    ama: [
      'dança',
      'pêssego',
      'rock'
    ],
    altura: 1.59
  }
escola >
```

MongoDB

- No **MongoDB**, o **comando (ou método) getIndexes( )** é usado para **listar todos os índices existentes em uma coleção.**
- Em outras palavras, ele retorna informações sobre todos os índices criados naquela coleção — incluindo o índice padrão criado automaticamente sobre o campo _id.

```
db.<nome_da_colecao>.getIndexes()
```

MongoDB

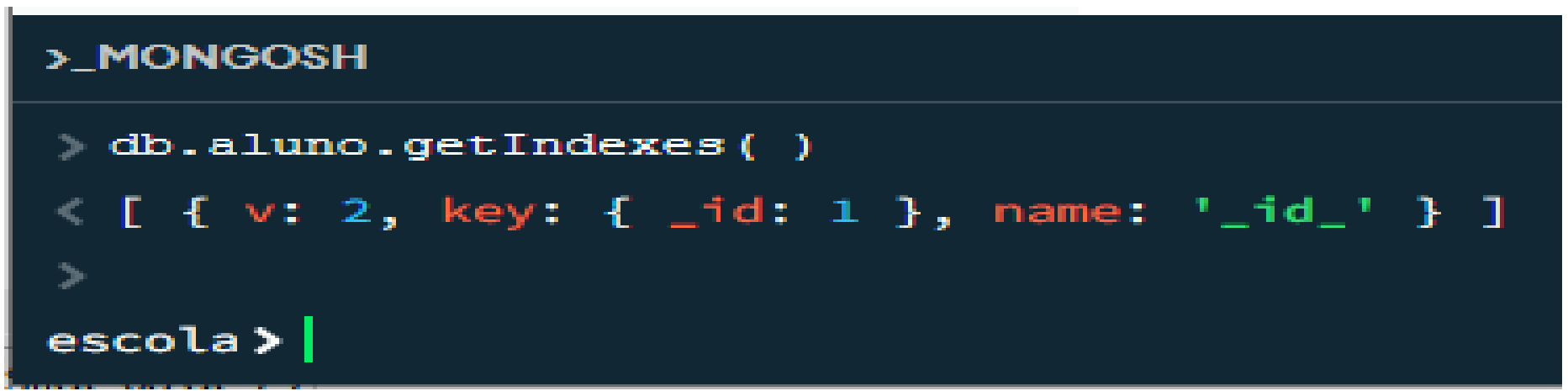
```
>_MONGOSH

> db.aluno.getIndexes( )
< [ { v: 2, key: { _id: 1 }, name: '_id_' } ]
>

escola>
```

Explicando o resultado:

- `v` → versão do formato do índice.
- `key` → campos indexados e sua ordem (`1` para ascendente, `-1` para descendente).
- `name` → nome do índice (gerado automaticamente, a menos que seja definido).
- `unique` → indica se o índice impõe unicidade.
- Outros campos podem aparecer (como `partialFilterExpression`, `sparse`, etc.) dependendo do tipo de índice.



MongoDB

- No **MongoDB**, você cria índices com o comando **createIndex( )** (ou **createIndexes( )** para criar vários de uma vez).
- Eles servem para **acelerar consultas** (**find**, **sort**, **aggregate**, etc.) **sobre campos específicos de uma coleção.**

MongoDB

```
db.<nome_da_colecao>.createIndex({ <campo>: <ordem> }, { <opções> })
```

<campo> → nome do campo a ser indexado

<ordem> → 1 (crescente) ou -1 (decrescente)

<opções> → parâmetros opcionais (ex: unique, sparse, name, etc.)

MongoDB

```
>_MONGOSH

 >
 > db.aluno.createIndex({  nome:  1  })
 < nome_1
escola >
```

Criação de um índice para o atributo nome (em ordem crescente). O nome do índice é nome_1.

MongoDB

```
>_MONGOSH

> use escola
< switched to db escola
> db.aluno.getIndexes()
< [
    { v: 2, key: { _id: 1 }, name: '_id_' },
    { v: 2, key: { nome: 1 }, name: 'nome_1' }
  ]
escola>
```

A **versão (v:)** do índice **indica o formato interno e o algoritmo usado pelo MongoDB para construir e gerenciar aquele índice.** Ela existe porque, ao longo do tempo, o **MongoDB mudou e otimizou o formato dos índices** — então cada índice tem uma "**versão de estrutura**".

MongoDB

Aqui, v: 2 significa que o índice foi criado usando o formato de índice versão 2 (atual nas versões modernas do MongoDB).

Ajuda na compatibilidade entre versões do MongoDB. Importante em processos de migração entre versões. Índices mais novos têm melhorias de desempenho e correção de bugs

Principais valores

v (versão do índice)	Significado
0 / 1	Formatos antigos, usados em versões bem antigas do MongoDB
2	Versão atual dos índices (mais comum a partir do MongoDB 3.2+)

MongoDB

```
>_MONGOSH
> db.aluno.find({nome: 'Asdrubal Soares Ribeiro'})
< {
    _id: ObjectId("68f90a714c6353124c0ce2eb"),
    matricula: 1100,
    nome: 'Asdrubal Soares Ribeiro',
    sexo: 'M',
    altura: 1.72
  }
escola>
```

Efetuando uma **busca** pelos **documentos** com coluna **nome** igual a “**Asdrúbal Soares Ribeiro**”.

MongoDB

```
>_MONGOSH

> db.aluno.find({nome: 'Asdrubal Soares Ribeiro'})
< {
    _id: ObjectId("68f90a714c6353124c0ce2eb"),
    matricula: 1100,
    nome: 'Asdrubal Soares Ribeiro',
    sexo: 'M',
    altura: 1.72
  }
> show collections
< aluno
  disciplina
escola >
```

Show collections exibe todas as coleções do banco de dados em questão: aluno e disciplina.

MongoDB

```
>_MONGOSH
> db.aluno.find({sexo: 'F'})
< {
    _id: ObjectId("68f90a254c6353124c0ce2ea"),
    matricula: 1099,
    nome: 'Sandra Calegari',
    sexo: 'F',
    altura: 1.67
  }
  {
    _id: ObjectId("68f90ea94c6353124c0ce2ed"),
    matricula: 1101,
    nome: 'Rowena Santos',
    sexo: 'F',
    ama: [
      'papaia',
      'futebol'
    ],
    altura: 1.74
  }
```

Busca pelos documentos que tenham sexo = ‘F’.

MongoDB

```
>_MONGOSH
  {
    _id: ObjectId("68f911464c6353124c0ce2ef"),
    matricula: 1103,
    nome: 'Jambira Andrada',
    sexo: 'F',
    ama: [
      'dança',
      'pêssego',
      'rock'
    ],
    altura: 1.59
  }
escola>
```

MongoDB

```
>_MONGOSH
> db.aluno.find({
    sexo: "F",
    altura: { $lt: 1.70 }
  })
< {
    _id: ObjectId("68f90a254c6353124c0ce2ea"),
    matricula: 1099,
    nome: 'Sandra Calegari',
    sexo: 'F',
    altura: 1.67
  }
  {
    _id: ObjectId("68f911464c6353124c0ce2ef"),
    matricula: 1103,
    nome: 'Jambira Andrada',
    sexo: 'F',
    ama: [
      'dança',
      'pêssego',
      'rock'
    ],
    altura: 1.59
  }
escola>
```

- Buscando por alunos de sexo ‘F’ e altura menor que 1.70.
- Observe que vieram apenas ‘Sandra’ e ‘Jambira’.
- ‘Rowena’, com 1.74 m de altura, não aparece.

MongoDB

```
> db.aluno.find(
    { sexo: "F", altura: { $lt: 1.70 } },
    { _id: 0, nome: 1, altura: 1, sexo: 1, ama:1 }
  )
< {
    nome: 'Sandra Calegari',
    sexo: 'F',
    altura: 1.67
  }
  {
    nome: 'Jambira Andrada',
    sexo: 'F',
    ama: [
      'dança',
      'pêssego',
      'rock'
    ],
    altura: 1.59
  }
escola>
```

Agora suponha que desejássemos que a coluna _id não fosse exibida.

MongoDB

```
>_MONGOSH
> db.aluno.find({
    sexo: "F",
    altura: { $lte: 1.67 }
  })
< {
    _id: ObjectId("68f90a254c6353124c0ce2ea"),
    matricula: 1099,
    nome: 'Sandra Calegari',
    sexo: 'F',
    altura: 1.67
  }
  {
    _id: ObjectId("68f911464c6353124c0ce2ef"),
    matricula: 1103,
    nome: 'Jambira Andrada',
    sexo: 'F',
    ama: [
      'dança',
      'pêssego',
      'rock'
    ],
    altura: 1.59
  }
escola>
```

- **Busca pelos alunos do sexo feminino e altura menor ou igual a 1.67.**
- **Uso do operador $lte em vez de $lt.**

MongoDB

```
>_MONGOSH

> db.aluno.find({
    altura: { $gt: 1.60 }
  })
< {
    _id: ObjectId("68f90a254c6353124c0ce2ea"),
    matricula: 1099,
    nome: 'Sandra Calegari',
    sexo: 'F',
    altura: 1.67
  }
  {
    _id: ObjectId("68f90a714c6353124c0ce2eb"),
    matricula: 1100,
    nome: 'Asdrubal Soares Ribeiro',
    sexo: 'M',
    altura: 1.72
  }
```

- Procurando **alunos** com **altura** **superior a** **1.60 m**.

MongoDB

```
>_MONGOSH
  {
    _id: ObjectId("68f90ea94c6353124c0ce2ed"),
    matricula: 1101,
    nome: 'Rowena Santos',
    sexo: 'F',
    ama: [
      'papaia',
      'futebol'
    ],
    altura: 1.74
  }
  {
    _id: ObjectId("68f9104b4c6353124c0ce2ee"),
    matricula: 1102,
    nome: 'Carlomano Santos',
    sexo: 'M',
    ama: [
      'vôlei'
    ],
    altura: 1.77
  }
escola>
```

MongoDB

```
>_MONGOSH

> db.aluno.find({
    altura: { $gte: 1.72 }
  })
< {
    _id: ObjectId("68f90a714c6353124c0ce2eb"),
    matricula: 1100,
    nome: 'Asdrubal Soares Ribeiro',
    sexo: 'M',
    altura: 1.72
  }
  {
    _id: ObjectId("68f90ea94c6353124c0ce2ed"),
    matricula: 1101,
    nome: 'Rowena Santos',
    sexo: 'F',
    ama: [
      'papaia',
      'futebol'
    ],
    altura: 1.74
  }
```

- **Procurando por alunos com altura maior ou igual ($gte) a 1.72 m.**

MongoDB

- Como a coleção aluno possui índices em _id (padrão) e nome, você pode verificar se o índice está sendo utilizado e comparar desempenho com e sem índice.

- Para tanto, utilize o comando explain( ) acoplado ao comando find( ): db.aluno.find(...).explain("executionStats").

MongoDB

```
>_MONGOSH
> db.aluno.find({ nome: "Maria" }).explain("executionStats")
< {
    explainVersion: '1',
    queryPlanner: {
      namespace: 'escola.aluno',
      indexFilterSet: false,
      parsedQuery: {
        nome: {
          '$eq': 'Maria'
        }
      },
      queryHash: '6ADB31C1',
      planCacheKey: 'EEB0C82A',
      maxIndexedOrSolutionsReached: false,
      maxIndexedAndSolutionsReached: false,
      maxScansToExplodeReached: false,
      winningPlan: {
        stage: 'FETCH',
        inputStage: {
          stage: 'IXSCAN',
          keyPattern: {
            nome: 1
          }
```

- Não existe um documento com o nome “Maria”.
- **Na saída, verifique:**
- **indexName** deve aparecer como "**nome_1**".
- **stage** deve ser **IXSCAN** (significa que usou índice), e não **COLLSCAN** (varredura completa).
- Compare **nReturned vs totalDocsExamined:** Boa eficiência: poucos documentos examinados para muitos retornados.

MongoDB

```
>_MONGOSH
> db.aluno.find({ nome: "Rowena Santos" }).explain("executionStats")
< {
    explainVersion: '1',
    queryPlanner: {
      namespace: 'escola.aluno',
      indexFilterSet: false,
      parsedQuery: {
        nome: {
          '$eq': 'Rowena Santos'
        }
      },
      queryHash: '6ADB31C1',
      planCacheKey: 'EEB0C82A',
      maxIndexedOrSolutionsReached: false,
      maxIndexedAndSolutionsReached: false,
      maxScansToExplodeReached: false,
      winningPlan: {
        stage: 'FETCH',
        inputStage: {
          stage: 'IXSCAN',
          keyPattern: {
            nome: 1
```

- **Documento** com **nome** “**Rowena Santos**” **existe**.
- **Stage IXSCAN** (utilizou o índice).

MongoDB

```
>_MONGOSH
winningPlan: {
  stage: 'FETCH',
  inputStage: {
    stage: 'IXSCAN',
    keyPattern: {
      nome: 1
    },
    indexName: 'nome_1',
    isMultiKey: false,
    multiKeyPaths: {
      nome: []
    },
    isUnique: false,
    isSparse: false,
    isPartial: false,
    indexVersion: 2,
    direction: 'forward',
    indexBounds: {
      nome: [
        '["Rowena Santos", "Rowena Santos"]'
      ]
    }
  }
```

- **IndexName: ‘nome_1’.**
- **IndexVersion: 2.**

MongoDB

```
>_MONGOSH
    },
    executionStats: {
      executionSuccess: true,
      nReturned: 1,
      executionTimeMillis: 0,
      totalKeysExamined: 1,
      totalDocsExamined: 1,
      executionStages: {
        stage: 'FETCH',
        nReturned: 1,
        executionTimeMillisEstimate: 0,
        works: 2,
        advanced: 1,
        needTime: 0,
        needYield: 0,
        saveState: 0,
        restoreState: 0,
        isEOF: 1,
        docsExamined: 1,
        alreadyHasObj: 0,
        inputStage: {
          stage: 'IXSCAN',
          nReturned: 1,
```

- ExecutionSuccess: **true**.
- NReturned: **1** (retornou 1 documento).
- TotalKeysExamined: **1**.
- TotalDocsExamined: **1**.

MongoDB

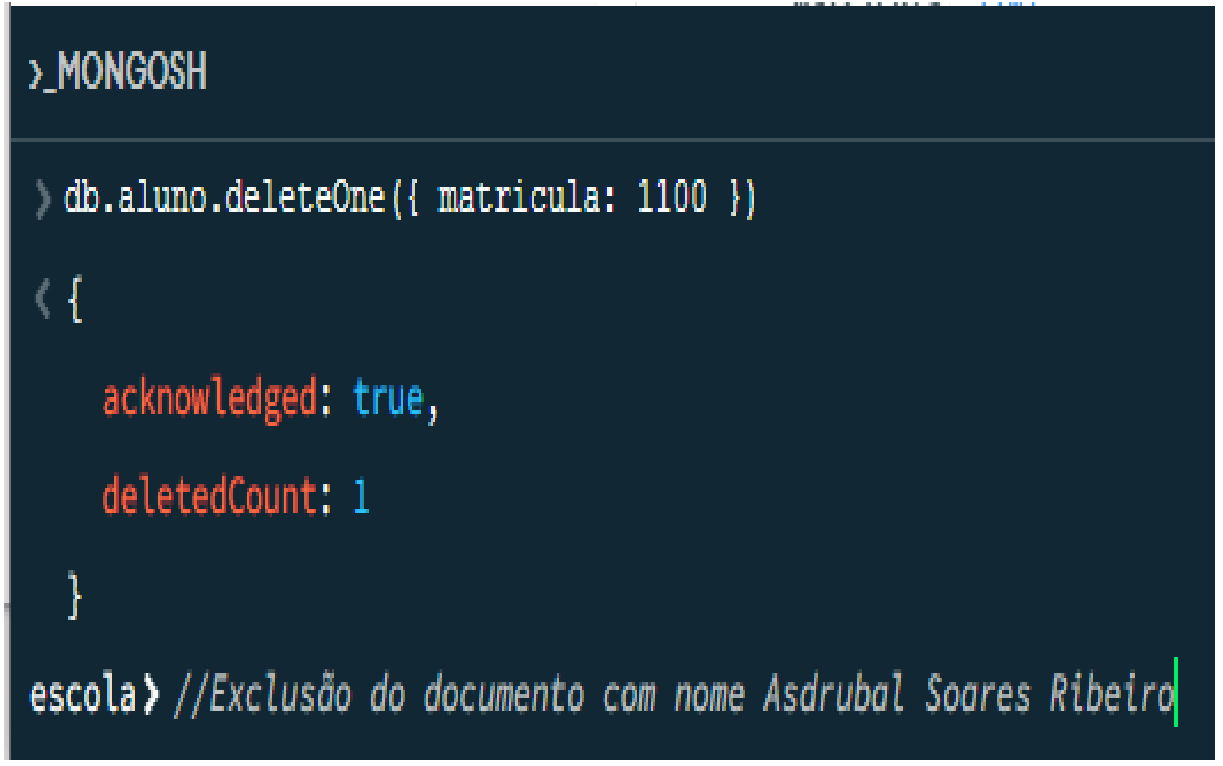
Busca que utiliza o índice:

```
db.aluno.find({ nome: "Asdrubal Soares Ribeiro" }).hint({ nome: 1 })
```

Busca sem a utilização do índice (que efetua a varredura):

```
db.aluno.find({ nome: "Asdrubal Soares Ribeiro" }).hint({ $natural: 1 })
```

MongoDB


›_MONGOSH
› db.aluno.deleteOne({ matricula: 1100 })
{
acknowledged: true,
deletedCount: 1
}
escola› //Exclusão do documento com nome Asdrubal Soares Ribeiro

MongoDB

```
>_MONGOSH

> db.aluno.deleteMany({ sexo: "M" })  //Exclusão de Documentos com Sexo = 'M'
< {
    acknowledged: true,
    deletedCount: 1
  }
escola>
```

MongoDB

```
> db.aluno.find()
< {
    _id: ObjectId("68f90a254c6353124c0ce2ea"),
    matricula: 1099,
    nome: 'Sandra Calegari',
    sexo: 'F',
    altura: 1.67
  }
  {
    _id: ObjectId("68f90ea94c6353124c0ce2ed"),
    matricula: 1101,
    nome: 'Rowena Santos',
    sexo: 'F',
    ama: [
      'papaia',
      'futebol'
    ],
    altura: 1.74
  }
```

MongoDB

```
  {
    _id: ObjectId("68f911464c6353124c0ce2ef"),
    matricula: 1103,
    nome: 'Jambira Andrada',
    sexo: 'F',
    ama: [
      'dança',
      'pêssego',
      'rock'
    ],
    altura: 1.59
  }
escola>
```

MongoDB

Para remover todos os documentos da coleção:

db.aluno.deleteMany({})

Para remover a Coleção:

db.aluno.drop()

MongoDB

Supondo que você queira mudar o nome e a altura do aluno com matrícula 123:

```
db.aluno.updateOne(
  { matricula: 123 },          // Critério de busca
  {
    $set: {                    // Campos a alterar
      nome: "Maria Souza",
      altura: 1.68
    }
  }
)
```

MongoDB

- No **MongoDB**, **agregação** (ou ***aggregation***) é um **mecanismo avançado de processamento de dados que permite realizar operações complexas sobre os documentos de uma coleção** — como filtragem, agrupamento, cálculos, ordenação, transformações e combinações — de forma **semelhante** ao que as **funções de agregação e consultas avançadas fazem em bancos relacionais** (ex.: ***GROUP BY***, ***SUM***, **AVG**, **JOIN**, etc.).

MongoDB

```
db.aluno.aggregate([
  { $group: { _id: "$sexo", total: { $sum: 1 } } }
])
```

Resultado:

```
[
  { "_id": "F", "total": 12 },
  { "_id": "M", "total": 15 }
]
```

- **Buscar a quantidade de alunos por sexo:**

MongoDB

- **Contar quantas alunas (F) têm altura maior que 1.60:**

```
db.aluno.aggregate([
  { $match: { sexo: "F", altura: { $gt: 1.60 } } },
  { $count: "quantidade" }
])
```

MongoDB

- **Agrupa por sexo e soma quantos alunos existem em cada categoria.**

```
db.aluno.aggregate([
  { $group: { _id: "$sexo", total: { $sum: 1 } } }
])
```

MongoDB

- **Média de altura dos alunos**

```
db.aluno.aggregate([
  { $group: { _id: null, media_altura: { $avg: "$altura" } } }
])
```

MongoDB

- **Média de altura separada por sexo:**

```
db.aluno.aggregate([
  { $group: {
      _id: "$sexo",
      media_altura: { $avg: "$altura" }
  }}
])
```

MongoDB

- **Listar apenas alunas com altura menor ou igual a 1.67:**

```
db.aluno.aggregate([
  { $match: { sexo: "F", altura: { $lte: 1.67 } } }
])
```

MongoDB

- **Contar quantas alunas têm altura menor ou igual a 1.67:**

```
db.aluno.aggregate([
  { $match: { sexo: "F", altura: { $lte: 1.67 } } },
  { $count: "quantidade" }
])
```

MongoDB

- **Selecionar apenas o nome e a altura dos alunos:**

```
db.aluno.aggregate([
  { $project: { _id: 0, nome: 1, altura: 1 } }
])
```

MongoDB

- **Ordenar alunos por altura (do maior para o menor)**

```
db.aluno.aggregate([
  { $sort: { altura: -1 } }
])
```

MongoDB

- **Calcular menor e maior altura da turma:**

```
db.aluno.aggregate([
  {
    $group: {
      _id: null,
      menor_altura: { $min: "$altura" },
      maior_altura: { $max: "$altura" }
    }
  }
])
```

MongoDB

- **Criar um campo calculado (IMC hipotético):**

```
db.aluno.aggregate([
  {
    $project: {
      nome: 1,
      altura: 1,
      peso: 1,
      imc: {
        $divide: [ "$peso", { $pow: [ "$altura", 2 ] } ]
      }
    }
  }
])
```

MongoDB

- **Agrupar por faixa de altura (bucket):**

```
db.aluno.aggregate([
  {
    $bucket: {
      groupBy: "$altura",
      boundaries: [1.40, 1.50, 1.60, 1.70, 1.80, 1.90],
      default: "fora_da_faixa",
      output: {
        total: { $sum: 1 },
        media_idade: { $avg: "$idade" }
      }
    }
  }
])
```