
UNITAU
Universidade de Taubaté

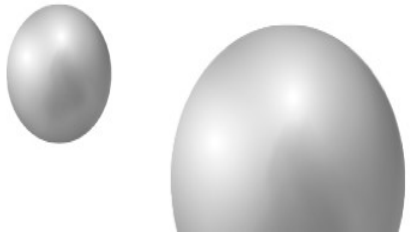

MongoDB
Introdução

Formato para Troca de Dados

Uma das mais importantes questões na área de informática é o armazenamento das informações e a troca destas informações entre diferentes sistemas.

O uso de banco de dados relacionais para o armazenamento das informações permite que as mesmas sejam manipuladas de forma rápida e segura por diferentes aplicações. Porém, é necessário uma definição do layout dessas informações para se encaixar no esquema do banco de dados.

Além disso, frequentemente é necessário trocar informações diretamente entre aplicações, sem que as mesmas precisem se conectar ao mesmo banco de dados.



Formato para Troca de Dados

Para resolver essas questões, foram criados formatos para troca de dados entre aplicações sem a exigência de um formato fixo. Dois dos formatos mais conhecidos são o **XML** e o **JSON**.

Várias linguagens de programação passaram a contar com recursos para geração, análise (parsing) e conversão dos dados nesses formatos. Tornando muito mais fácil a troca de informações entre sistemas, mesmo que escritos em diferentes linguagens.

Dada a popularidade desses formatos e a independência de um esquema fixo, diversos bancos de dados **NoSQL** se utilizam desses formatos para armazenar e/ou manipular os dados. Atualmente, mesmo bancos de dados relacionais possuem recursos para manipular informações nesses formatos.

XML

XML (Extensible Markup Language) é uma linguagem de marcação utilizada para descrição de dados estruturados.

É um dos subtipos da SGML (Standard Generalized Markup Language ou Linguagem Padronizada de Marcação Genérica) capaz de descrever diversos tipos de dados.

XML

Exemplo de dados em formato XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<alunos>
  <aluno nome="Maria">
    <notas>
      <nota>8</nota>
      <nota>10</nota>
      <nota>7</nota>
    </notas>
  </aluno>
  <aluno nome="José">
    <notas>
      <nota>10</nota>
      <nota>10</nota>
      <nota>9</nota>
    </notas>
  </aluno>
</alunos>
```

JSON

JSON (JavaScript Object Notation) é um formato compacto, de padrão aberto independente, de troca de dados entre sistemas, especificado por Douglas Crockford em 2000, que utiliza texto legível a humanos, no formato atributo-valor (natureza auto-descritiva).

Exemplo de dados em formato JSON:

```
{"Alunos":[
  { "nome": "João", "notas": [ 8, 9, 5 ] },
  { "nome": "Maria", "notas": [ 8, 10, 7 ] },
  { "nome": "José", "notas": [ 10, 10, 9 ] }
]}
```

MongoDB

MongoDB é um gerenciador de banco de dados orientado a documentos, de código aberto e gratuito.

O MongoDB armazena os dados em coleções ao invés de relações. Essas coleções não tem esquema fixo como as relações dos bancos de dados relacionais.

As coleções são um conjunto de registros semelhantes a um documento JSON.

As principais características do MongoDB são:

- Alta performance
- Alta disponibilidade
- Escalabilidade Horizontal

BSON

O MongoDB armazena os dados no formato BSON (Binary JSON), que é uma representação binária para documentos JSON.

O formato BSON define os seguintes tipos de dados:

double
string
object *(BSON object)*
array *(BSON array)*
binData *(byte array)*
objectId
bool
date *(Inteiro de 64 bits para milissegundos desde 01/01/1970)*
null
regex *(Expressão regular)*
javascript
javascriptWithScope

BSON

int *(Inteiro de 32 bits)*
timestamp
long *(Inteiro de 64 bits)*
decimal
minKey
maxKey

Shell

Para acessar o banco de dados é necessário se conectar ao gerenciador do banco de dados com algum programa que permita a interação com o servidor.

Com a instalação do gerenciador do MongoDB, também é instalado o shell do MongoDB, o programa mongo.

Shell

Para se conectar com o banco de dados usando esse shell, deve se executar o comando:

```
mongo [--port <port>] [--host <hostname>] [-u
<username>] [-p <password>]
[--authenticationDatabase <authdabase>]
[<database>]
```

onde:

<port> - porta de conexão do gerenciador de banco de dados
<hostname> - hostname do servidor de banco de dados
<username> - nome do usuário da conexão
<password> - senha do usuário
<authdatabase> - banco de dados onde estão as credenciais do usuário
<database> - nome do banco de dados

Shell

É possível se conectar ao servidor sem especificar qual usuário está fazendo a conexão.

Também é possível se conectar ao servidor sem especificar o banco de dados da conexão, nesse caso, o banco de dados corrente da conexão será o banco de dados padrão de nome test.

```
mongo
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.2
>
```

Shell

Para se exibir o nome do banco de dados corrente, se utiliza o método db.getName()

```
> db.getName()
test
> exit
bye
```

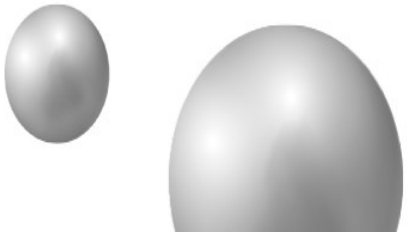
DataBases

As coleções são armazenadas em databases (bancos de dados) e toda conexão ao gerenciador do banco de dados tem um database corrente.

Para se alterar o banco de dados corrente da conexão deve se usar o comando:

```
use <database>
```

Nome de databases não diferenciam maiúsculas e minúsculas, devem ser menores que 64 caracteres.

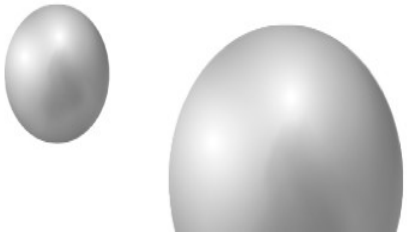


DataBases

É possível se alterar o banco de dados corrente para um banco de dados não existente e o banco de dados será gravado no gerenciador quando for inserido algum dado no banco de dados.

Por exemplo, é possível ser alterar o banco de dados corrente para um banco de dados chamado aluno, mesmo que esse banco ainda não exista:

```
mongo
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.2
> use aluno
switched to db aluno
```

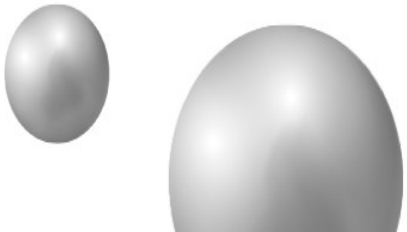


Usuários

Para acessar o MongoDB é necessária a criação dos usuários que acessarão o servidor.

Um usuário deve ter uma senha, os papéis de segurança atribuídos aos usuários e o banco ao qual o papel se aplica.

Existem diversos papéis pré-definidos no MongoDB, sendo também possível a criação de papéis customizados com os privilégios necessários.



Usuário Administrador

Se ainda não existir um usuário administrador, é necessário criar esse usuário administrador:

```
mongo
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.2
> use admin
switched to db admin
> db.createUser(
      {
        user: "root",
        pwd: "mongoadmin",
        roles: [ { role: "userAdminAnyDatabase",
db: "admin" } ]
      }
    )
```

Usuário Administrador

Para se conectar ao banco de dados com o usuário administrador definido anteriormente, deve ser executado o comando:

```
mongo -u root -p mongoadmin --
authenticationDatabase admin
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.2
>
```

Usuários

Para se criar os demais usuários que terão acesso aos bancos de dados, deve se utilizar o método createUser indicando quais os papéis que o usuário terá nos bancos de dados.

Usuários

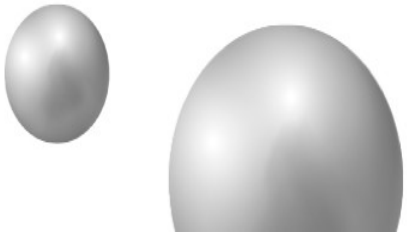
Por exemplo, para criar um usuário que possa ler e gravar informações no banco aluno, seguiremos os seguintes passos:

1) Conectar no gerenciador de banco de dados com o usuário administrador:

```
mongo -u root -p mongoadmin --
authenticationDatabase admin
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.2
```

2) Defina o banco "aluno" como o banco corrente

```
> use aluno
switched to db aluno
```



Usuários

3) Crie o usuário aluno

```
> db.createUser(
      {
        user: "aluno", pwd: "aluno",
        roles: [ { role: "readWrite", db: "aluno"
} ]
      })
Successfully added user: {
        "user" : "aluno",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "aluno"
                }
        ]
}
```

Usuários

Para se conectar ao banco de dados aluno com o usuário criado, deve ser executado o comando:

```
mongo -u aluno -p aluno --authenticationDatabase
aluno aluno
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27017/aluno
MongoDB server version: 3.6.2
```

Usuários

Para que a senha seja solicitada no prompt, o parametro -p deve ser o ultimo parametro do comando:

```
mongo -u aluno --authenticationDatabase aluno
aluno -p
MongoDB shell version v3.6.2
Enter password:
connecting to: mongodb://127.0.0.1:27017/aluno
MongoDB server version: 3.6.2
```

Coleções

O MongoDB armazena os dados em coleções. Uma coleção é um conjunto de documentos, sendo que cada documento tem sua própria estrutura.

Para inserir documentos em uma coleção, não é necessário que a coleção exista previamente. Ao inserir documentos em uma coleção ainda não existente, a coleção é criada automaticamente.

Coleções

```
>  db.contatos.insertOne({nome:  "Luis",  email:
"luis@gmail.com"})
{
        "acknowledged" : true,
        "insertedId" :
ObjectId("5b606af49d70aee9d902be3a")
}
> db.contatos.insertOne( {nome: "Ana", email:
"ana@gmail.com"} )
{
        "acknowledged" : true,
        "insertedId" :
ObjectId("5b606b1e9d70aee9d902be3b")
}
```

Coleções

```
> db.contatos.find()
{ "_id" : ObjectId("5b606af49d70aee9d902be3a"),
"nome" : "Luis", "email" : "luis@gmail.com" }
{ "_id" : ObjectId("5b606b1e9d70aee9d902be3b"),
"nome" : "Ana", "email" : "ana@gmail.com" }
```

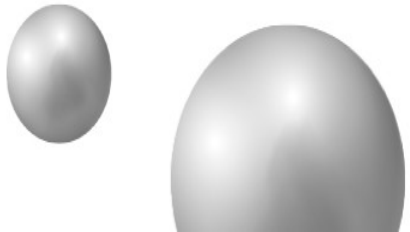
Campo _id

Todo documento armazenado em um coleção tem um campo _id que é a chave primária do documento. O campo _id pode ser especificado explicitamente pelo usuário ou gerado automaticamente pelo MongoDB.

O campo _id pode conter valores de qualquer tipo de dados BSON exceto arrays.

Se não for especificado o _id do documento, o MongoDB usará a função ObjectId() para gerar a chave primária do documento.

Quando cria uma coleção, O MongoDb também cria um índice pelo campo _id para a coleção.



Campo _id

O campo _id sempre é o primeiro campo do documento. Se for inserido um documento com o campo _id em outra posição que não o primeiro campo, o MongoDB moverá o campo _id para o primeiro campo do documento.

Se o usuário especificar o valor do _id do documento, deve garantir que este valor não seja repetido.

Campo _id

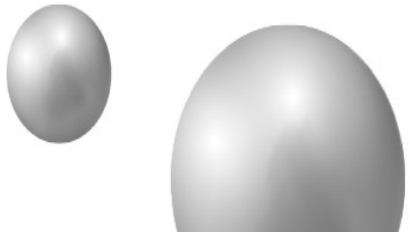
```
> db.contatos.insertOne( {_id: 1, nome: "Marcia",
email: "marcia@gmail.com"} )
{ "acknowledged" : true, "insertedId" : 1 }
> db.contatos.find()
{ "_id" : ObjectId("5b606af49d70aee9d902be3a"),
"nome" : "Luis", "email" : "luis@gmail.com" }
{ "_id" : ObjectId("5b606b1e9d70aee9d902be3b"),
"nome" : "Ana", "email" : "ana@gmail.com" }
{ "_id" : 1, "nome" : "Marcia", "email" :
"marcia@gmail.com" }
```

Execução de Comandos

Os comandos executados sobre um banco de dados também podem ser executados usando o método db.runCommand(). Nesta forma, deve ser indicado o comando e os parametros na forma de um documento JSON.

Execução de Comandos

```
> db.runCommand(
      {
        find: "contatos",
        projection: { _id: 0, nome: 1 },
        sort: { nome: 1 },
        limit: 1
      }
    )
```



Execução de Comandos

```
{
        "cursor" : {
                "firstBatch" : [
                        {
                                "nome" : "Ana"
                        }
                ],
                "id" : NumberLong(0),
                "ns" : "aluno.contatos"
        },
        "ok" : 1
}
```

Execução de Comandos

Para executar comandos sobre o banco de dados "admin", pode ser usado o método db.adminCommand().

```
mongo -u root -p mongoadmin --
authenticationDatabase admin test
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.2
> db.getName()
test
```

Execução de Comandos

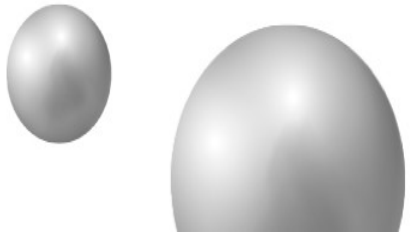
```
> db.adminCommand(
      {
        createUser: "teste",
        pwd: "teste123",
        roles: [
          "readWrite"
        ]
      }
    )
{ "ok" : 1 }
> use admin
switched to db admin
```

Execução de Comandos

```
> show users
{
        "_id" : "admin.root",
        "user" : "root",
        "db" : "admin",
        "roles" : [
                {
                        "role" :
"userAdminAnyDatabase",
                        "db" : "admin"
                }
        ]
}
```

Execução de Comandos

```
{
        "_id" : "admin.teste",
        "user" : "teste",
        "db" : "admin",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "admin"
                }
        ]
}
```



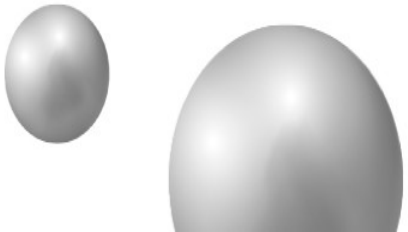
Deleção de Usuários

Para remover um usuário deve ser utilizado o método dropUser().

```
mongo -u root -p mongoadmin --
authenticationDatabase admin admin
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27017/admin
MongoDB server version: 3.6.2
> db.dropUser("teste")
true
```

Deleção de Usuários

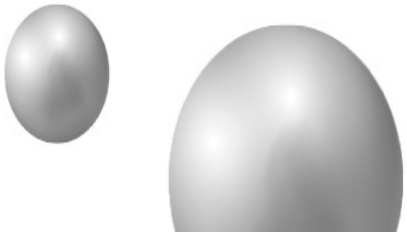
```
> show users
{
        "_id" : "admin.root",
        "user" : "root",
        "db" : "admin",
        "roles" : [
                {
                        "role" :
"userAdminAnyDatabase",
                        "db" : "admin"
                }
        ]
}
```



Criação Explícita de Coleções

Apesar de não ser necessário, é possível criar explicitamente uma coleção. A criação explícita de uma coleção permite definir características da coleção como coleções limitadas, validação de esquema, codificação de caracteres e outras características.

```
mongo -u aluno -p aluno --authenticationDatabase
aluno aluno
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27017/aluno
MongoDB server version: 3.6.2
> db.createCollection( "teste" )
{ "ok" : 1 }
> show collections
contatos
teste
```



Criação Explícita de Coleções

Se necessário alterar as opções de uma coleção após a criação da coleção, pode ser utilizado o command collMod:

```
db.runCommand( { collMod: <colecao>, <opcao1>:
<valor1>, <opcao2>: <valor2> ... } )
```

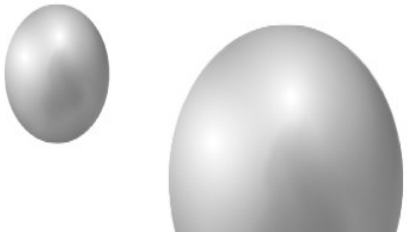
Coleções Limitadas

Coleções limitadas são coleções com um limite de tamanho. Essas coleções obrigatoriamente tem um tamanho máximo em bytes e opcionalmente um número máximo de documentos.

Coleções limitadas preservam a ordem de inserção dos documentos.

Quando um novo documento é inserido em uma coleção limitada, se a coleção ultrapassar os limites estabelecidos, o documento mais antigo da coleção é deletado para permitir a inclusão do novo documento.

Não é permitido deletar documentos de uma coleção limitada.



Coleções Limitadas

```
> db.createCollection("nomes", {capped: true,
size: 5000, max: 3} )
{ "ok" : 1 }
> db.getCollectionNames()
[ "contatos", "nomes", "teste" ]
> db.nomes.insertOne( {nome: "Ana"} )
{
        "acknowledged" : true,
        "insertedId" :
ObjectId("5b7ba53935bcf552041c73d8")
}
```

Coleções Limitadas

```
> db.nomes.insertOne( {nome: "Pedro"} )
{
        "acknowledged" : true,
        "insertedId" :
ObjectId("5b7ba56a35bcf552041c73d9")
}
> db.nomes.insertOne( {nome: "Paula"} )
{
        "acknowledged" : true,
        "insertedId" :
ObjectId("5b7ba5cc35bcf552041c73da")
}
```

Coleções Limitadas

```
> db.nomes.find()
{ "_id" : ObjectId("5b7ba53935bcf552041c73d8"),
"nome" : "Ana" }
{ "_id" : ObjectId("5b7ba56a35bcf552041c73d9"),
"nome" : "Pedro" }
{ "_id" : ObjectId("5b7ba5cc35bcf552041c73da"),
"nome" : "Paula" }
```

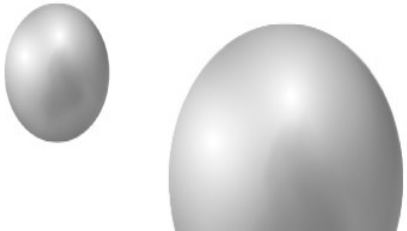
Coleções Limitadas

```
> db.nomes.insertOne( {nome: "Carlos"} )
{
        "acknowledged" : true,
        "insertedId" :
ObjectId("5b7ba64e35bcf552041c73db")
}
> db.nomes.find( {},{_id: 0, nome: 1} )
{ "nome" : "Pedro" }
{ "nome" : "Paula" }
{ "nome" : "Carlos" }
```

Nome de Coleções

O nome de uma coleção deve começar com o carácter "_" ou com uma letra. Não pode conter o carácter $ ou null e não deve começar com system. que é um prefixo reservado para uso interno do MongoDB

O tamanho máximo do namespace das coleções é 120 caracteres. O namespace da coleção é o nome do banco de dados mais o separador . e o nome da coleção (isto é, <database>.<collection>), como por exemplo, aluno.contatos.

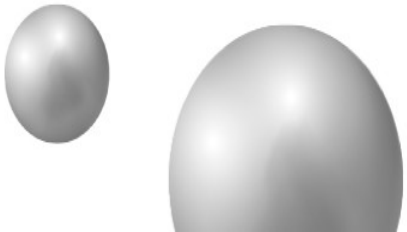


Nome dos Campos

O nome de um campo não pode conter o carácter null e em versões anteriores a 3.x não era permitido conter o carácter . ou iniciar com o carácter $.

A partir da versão 3.x é possível nome de campos com o carácter . ou iniciando com $ desde que estejam dentro de outro campo. Nomes de campos com o carácter . devem ser colocados entre aspas.

```
> db.teste.insertOne( {contact: {"first.name":
"Ana", "last.name": "Silva", $age: 21}} )
{
        "acknowledged" : true,
        "insertedId" :
ObjectId("5b7baabd35bcf552041c73e0")
}
```



Nome dos Campos

```
> db.teste.find()
{ "_id" : ObjectId("5b7baabd35bcf552041c73e0"),
"contact" : { "first.name" : "Ana", "last.name" :
"Silva", "$age" : 21 } }
```

Nome dos Campos

Documentos BSON podem ter mais de um campo com o mesmo nome. Porém a maioria das interfaces do MongoDB utilizam uma tabela hash para representar os documentos e não suportam múltiplos campos com o mesmo nome.

```
> db.teste.insertOne( {name: "Ana", name:
"Silva"} )
{
        "acknowledged" : true,
        "insertedId" :
ObjectId("5b7bab6d35bcf552041c73e1")
}
```

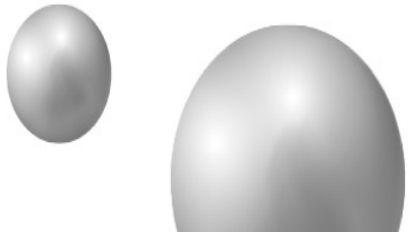
Nome dos Campos

```
> db.teste.find()
{ "_id" : ObjectId("5b7baabd35bcf552041c73e0"),
"contact" : { "first.name" : "Ana", "last.name" :
"Silva", "$age" : 21 } }
{ "_id" : ObjectId("5b7bab6d35bcf552041c73e1"),
"name" : "Silva" }
```

Deleção de Coleções

Para deletar uma coleção, deve ser usado o método drop().

```
> db.getCollectionNames()
[ "contatos", "nomes", "teste" ]
> db.nomes.drop()
true
> db.getCollectionNames()
[ "contatos", "teste" ]
```



Informações das Coleções

Para obter informações sobre as coleções existentes, pode ser usado o método getCollectionInfos().

```
> db.getCollectionInfos()
[
        {
                "name" : "contatos",
                "type" : "collection",
                "options" : {

                },
                "info" : {
                        "readOnly" : false,
                        "uuid" : UUID("a6ce81d9-
53c9-4085-b224-7a13549a5b3a")
                },
```

Informações das Coleções

```
        "idIndex" : {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "aluno.contatos"
        }
},
{
        "name" : "teste",
        "type" : "collection",
        "options" : {

        },
```

Informações das Coleções

```
                "info" : {
                        "readOnly" : false,
                         "uuid" : UUID("290d7ea0-
a204-4063-b86a-018b4619b75d")
                },
                "idIndex" : {
                        "v" : 2,
                        "key" : {
                                "_id" : 1
                        },
                        "name" : "_id_",
                        "ns" : "aluno.teste"
                }
        }
]
```

Coleções de Sistema

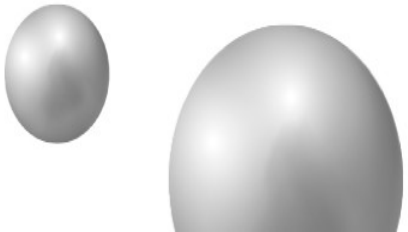
O MongoDB armazena informações do sistema em coleções que usam o namespace <database>.system.*.

Não é permitido criar coleções com esse padrão de nome.

Coleções de Sistema

Exemplos de coleções do sistema armazenadas no banco admin:

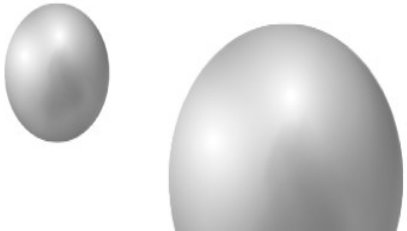
- **admin.system.roles** - armazena os papéis customizados criados pelos administradores do banco
- **admin.system.users** - armazena as credenciais de autenticação dos usuários



Coleções de Sistema

Exemplos de coleções do sistema armazenadas em cada banco de dados:

- **<database>.system.profile - armazena informações sobre as operações executadas no banco de dados para análise de performance**
- **<database>.system.js - armazena código JavaScript**



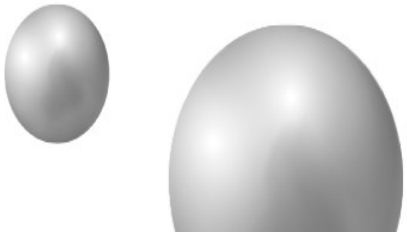
Validação de Esquema

Definindo uma regra de validação para uma coleção, a estrutura dos documentos será verificada quando houver inserções ou alterações de documentos da coleção.

A regra de validação pode ser definida na criação da coleção com o método createCollection() ou alterada em uma coleção já existente com o comando collMod.

A partir da versão 3.6, o MongoDB suporta a validação de esquema do JSON.

A validação de esquema se aplica na inserção ou alteração de um documento. Documentos já existentes na coleção não são verificados quando a regra de validação é criada ou alterada.



Validação de Esquema

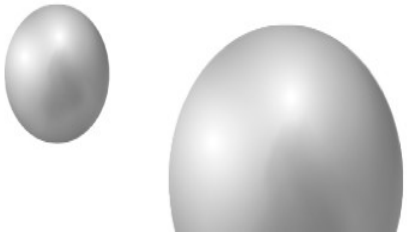
A opção **validationLevel** da regra de validação determina como será a validação de documentos já existentes no momento de definição da regra:

- **strict** - (valor default) determina que a regra se aplica a todas inserções e alterações
- **moderate** - a validação é aplicada a todas inserções mas apenas alteração de documentos que já atendiam a regra antes da alteração, documentos que não atendem a regra de validação não são verificados quando alterados

Validação de Esquema

A opção **validationAction** determina se a validação impede ou não a inserção e alteração de documentos que não atendem a regra de validação:

- **error** - (valor default) determina que o MongoDB rejeite inserções e alterações que violem a regra de validação
- **warn** - violações da regra de validação são apenas registradas mas a inserção ou alteração é executada

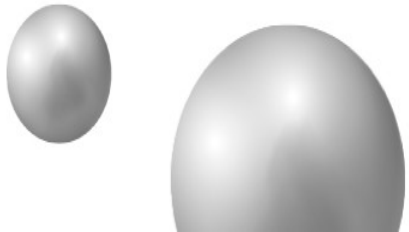


Validação de Esquema

Não é permitido criar regras de validação para as coleções do sistema ou coleções nos bancos de dados **admin**, **local** e **config**.

Alguns métodos possuem a opção **bypassDocumentValidation** para contornar a regra de validação.

```
> db.contatos.drop()
true
```



Validação de Esquema

```
> db.createCollection( "contatos", {
      validator: { $jsonSchema: {
         bsonType: "object",
         required: [ "nome", "tipo" ],
         properties: {
            nome: {
               bsonType: "string",
               description: "Obrigatorio, Deve
ser uma string"
               },
```

Validação de Esquema

```
                email: {bsonType : "string",
                   pattern : "@unitau\\.com\\.br$",
                   description: "Deve ser uma string
compativel com a expressao regular"
                },
                 tipo: {
                   enum: [ "Professor", "Aluno" ],
                   description: "Apenas valores
permitidos"
                }
             }
          } }
      } )
{ "ok" : 1 }
```

Validação de Esquema

```
> db.contatos.insertOne( {nome: "Ana"} )
WriteError({
  "index" : 0,
  "code" : 121,
  "errmsg" : "Document failed validation",
  "op" : {
    "_id" : ObjectId("62298b015f9ca34b2c88ccb7"),
    "nome" : "Ana"
  },
  "errInfo" : {
    "failingDocumentId" :
ObjectId("62298b015f9ca34b2c88ccb7"),
    "details" : {
      "operatorName" : "$jsonSchema",
      "schemaRulesNotSatisfied" : [
        {
          "operatorName" : "required",
          "specifiedAs" : {
            "required" : [
              "nome",
              "tipo"
            ]
```

Validação de Esquema

```
            },
            "missingProperties" : [
              "tipo"
            ]
          }
        ]
      }
    }
})  :
WriteError({
    "index" : 0,
    "code" : 121,
    "errmsg" : "Document failed validation",
    "op" : {
      "_id" : ObjectId("62298b015f9ca34b2c88ccb7"),
      "nome" : "Ana"
    },
    "errInfo" : {
      "failingDocumentId" :
ObjectId("62298b015f9ca34b2c88ccb7"),
```

Validação de Esquema

```
    "details" : {
      "operatorName" : "$jsonSchema",
      "schemaRulesNotSatisfied" : [
        {
          "operatorName" : "required",
          "specifiedAs" : {
            "required" : [
              "nome",
              "tipo"
            ]
          },
          "missingProperties" : [
            "tipo"
          ]
        }
      ]
    }
  }
})
```

Validação de Esquema

```
WriteError@src/mongo/shell/bulk_api.js:465:48
mergeBatchResults@src/mongo/shell/bulk_api.js:871:49
executeBatch@src/mongo/shell/bulk_api.js:940:13
Bulk/this.execute@src/mongo/shell/bulk_api.js:1182:21
DBCollection.prototype.insertOne@src/mongo/shell/
crud_api.js:264:9
@(shell):1:1
```

Validação de Esquema

```
> db.contatos.insertOne( {nome: "Paula", email:
"paula@gmail.com", tipo: "Aluno"} )
WriteError({
  "index" : 0,
  "code" : 121,
  "errmsg" : "Document failed validation",
  "op" : {
    "_id" : ObjectId("62298e0c5f9ca34b2c88ccb8"),
    "nome" : "Paula",
    "email" : "paula@gmail.com",
    "tipo" : "Aluno"
  },
  "errInfo" : {
    "failingDocumentId" :
ObjectId("62298e0c5f9ca34b2c88ccb8"),
    "details" : {
      "operatorName" : "$jsonSchema",
      "schemaRulesNotSatisfied" : [
        {
          "operatorName" : "properties",
```

Validação de Esquema

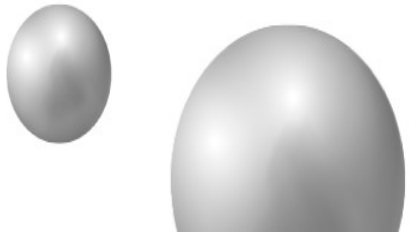
```
          "propertiesNotSatisfied" : [
            {
              "propertyName" : "email",
              "details" : [
                {
                  "operatorName" : "pattern",
                  "specifiedAs" : {
                    "pattern" : "@unitau\\.com\\.br$"
                  },
                  "reason" : "regular expression did not
match",
                  "consideredValue" : "paula@gmail.com"
                }
              ]
            }
          ]
        }
      ]
    }
  }
}) :
```

Validação de Esquema

```
WriteError({
  "index" : 0,
  "code" : 121,
  "errmsg" : "Document failed validation",
  "op" : {
    "_id" : ObjectId("62298e0c5f9ca34b2c88ccb8"),
    "nome" : "Paula",
    "email" : "paula@gmail.com",
    "tipo" : "Aluno"
  },
  "errInfo" : {
    "failingDocumentId" :
ObjectId("62298e0c5f9ca34b2c88ccb8"),
    "details" : {
      "operatorName" : "$jsonSchema",
      "schemaRulesNotSatisfied" : [
        {
          "operatorName" : "properties",
          "propertiesNotSatisfied" : [
            {
              "propertyName" : "email",
```

Validação de Esquema

```
            "details" : [
              {
                "operatorName" : "pattern",
                "specifiedAs" : {
                  "pattern" : "@unitau\\.com\\.br$"
                },
                "reason" : "regular expression did not
match",
                "consideredValue" : "paula@gmail.com"
              }
            ]
          }
        ]
      }
    ]
  }
}
```



Validação de Esquema

```
WriteError@src/mongo/shell/bulk_api.js:465:48
mergeBatchResults@src/mongo/shell/bulk_api.js:871:49
executeBatch@src/mongo/shell/bulk_api.js:940:13
Bulk/this.execute@src/mongo/shell/bulk_api.js:1182:21
DBCollection.prototype.insertOne@src/mongo/shell/
crud_api.js:264:9
@(shell):1:1
```

Validação de Esquema

```
> db.contatos.insertOne( {nome: "Paula", email:
"paula@unitau.com.br", tipo: "Aluno"} )
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5b7bbdd735bcf552041c73ea")
}
```

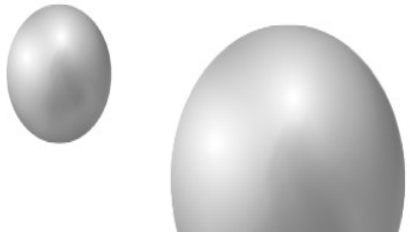
Roles

O MongoDB utiliza um sistema de papéis (roles) para gerenciar o acesso dos usuários a recursos e operações. O usuário deve ter acesso a um ou mais papéis para permitir o acesso do usuário aos privilégios atribuídos aos papéis.

Privilégio é a permissão para executar uma ação (action) sobre um recurso (resource) do banco de dados.

Existem diversos papéis pré-definidos para gerenciar diversas tarefas ou situações como administração do banco de dados, administração de clusters, backup e restauração de dados.

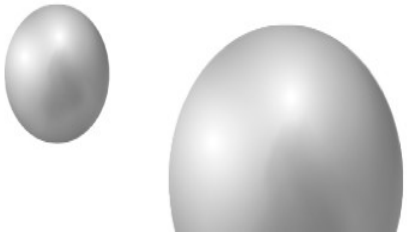
As ações e papéis pré-definidos existentes podem ser verificadas na documentação do MongoDB.



Roles

Além dos papéis específicos para essas tarefas, todos bancos de dados incluem papéis para permitir o acesso de usuários comuns aos objetos do banco:

- **read** - permite a leitura de todas as coleções que não são coleções do sistema e também nas coleções do sistema system.indexes, system.js e system.namespaces.
- **readWrite** - permite os acessos do papel **read** e também a modificação de todas as coleções que não são coleções do sistema e da coleção de sistema system.js.



Roles

Além dos papéis pré-definidos, é possível criar papéis customizados com privilégios específicos.

O controle de acesso por papeis devem ser habilitado no arquivo de configuração ou na linha de comando de execução do MongoDB.

Inserindo Documentos

O MongoDB permite a inserção individual de documentos ou a inserção de múltiplos documentos em uma coleção

Para inserção individual de documentos deve ser usado o método **db.insertOne()**.

```
db.collection.insertOne(
    <document>,
    {
        writeConcern: <document>
    }
)
```

Inserindo Documentos

```
> db.produtos.insertOne( {_id: 1, item:
"caderno", quantidade: 20} )
{ "acknowledged" : true, "insertedId" : 1 }
> db.produtos.find()
{ "_id" : 1, "item" : "caderno", "quantidade" :
20 }
```

Inserindo Documentos

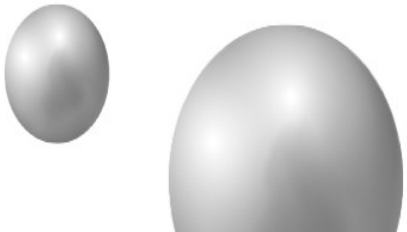
Para inserir múltiplos documentos deve ser usado o método db.insertMany().

```
db.collection.insertMany(
   [ <document 1> , <document 2>, ... ],
   {
      writeConcern: <document>,
      ordered: <boolean>
   }
)
```

Inserindo Documentos

Se o parâmetro ordered tem valor true (valor default), os documentos serão inseridos na ordem do array de documentos e se ocorrer um erro na inserção de algum documento, os demais documentos não serão inseridos.

Se o parâmetro ordered tem valor false, o MongoDB pode reordenar os documentos para melhorar a performance das inserções e se ocorrer um erro na inserção de algum documento, o MongoDB continuar a processar a inserção dos demais documentos.



Inserindo Documentos

```
> db.produtos.insertMany( [
    {_id: 2, item: "cartao", quantidade: 15,
minimo: 5},
    {_id: 3, item: "envelope", quantidade: 20,
minimo: 10},
    {_id: 4, item: "selos" , quantidade: 30,
minimo: null}
  ] )
{ "acknowledged" : true, "insertedIds" : [ 2, 3,
4 ] }
```

Inserindo Documentos

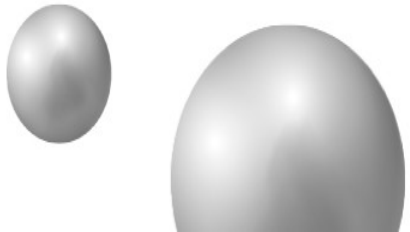
```
> db.produtos.find()
{ "_id" : 1, "item" : "caderno", "quantidade" :
20 }
{ "_id" : 2, "item" : "cartao", "quantidade" :
15, "minimo" : 5 }
{ "_id" : 3, "item" : "envelope", "quantidade" :
20, "minimo" : 10 }
{ "_id" : 4, "item" : "selos", "quantidade" : 30,
"minimo" : null }
```

Inserindo Documentos

Além dos métodos db.insertOne() e db.insertMany() também é possível usar o método db.insert() para inserir documentos.

O método db.insert() permite a inserção individual ou de múltiplos documentos.

```
db.collection.insert(
    <document or array of documents>,
    {
      writeConcern: <document>,
      ordered: <boolean>
    }
)
```



Inserindo Documentos

```
> db.produtos.insert( [
    {_id: 5, item: "lapis", quantidade: 50,
minimo: 30},
    {_id: 6, item: "borracha", quantidade: 25,
minimo: 10}
  ] )
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

Inserindo Documentos

```
> db.produtos.find()
{ "_id" : 1, "item" : "caderno", "quantidade" :
20 }
{ "_id" : 2, "item" : "cartao", "quantidade" :
15, "minimo" : 5 }
{ "_id" : 3, "item" : "envelope", "quantidade" :
20, "minimo" : 10 }
{ "_id" : 4, "item" : "selos", "quantidade" : 30,
"minimo" : null }
{ "_id" : 5, "item" : "lapis", "quantidade" : 50,
"minimo" : 30 }
{ "_id" : 6, "item" : "borracha", "quantidade" :
25, "minimo" : 10 }
```

Atomicidade

No MongoDB as operações de escrita são atômicas para cada documento e não por operação, ou seja, em uma instrução que escreve em vários documentos, a gravação de cada documento é uma operação isolada, não há a relação de atomicidade entre os diferentes documentos afetados pela operação.

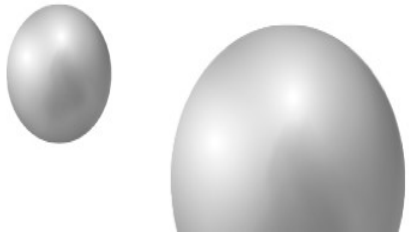
A partir da versão 4.0 o MongoDB implementa transação multi-documento.

Importando Documentos

O utilitário **mongoimport** permite inserir múltiplos documentos a partir de um arquivo texto.

Atualmente é possível importar documentos de arquivos nos formatos **json**, **csv** (comma-separated values) e **tsv** (tab-separated values).

Os parâmetros do utilitário **mongoimport** devem ser verificados na documentação.



Importando Documentos

```
mongo -u aluno -p aluno --authenticationDatabase
aluno aluno
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27017/aluno
MongoDB server version: 3.6.2
> show collections
contatos
produtos
teste
> exit
bye
```

Importando Documentos

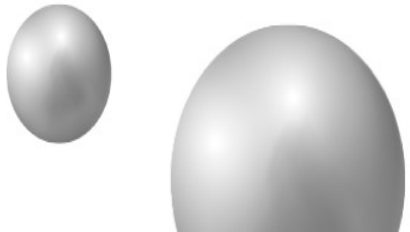
```
mongoimport -d aluno -u aluno -p aluno -c megasena megasena.json
2018-08-25T02:42:16.879-0300    connected to: localhost
2018-08-25T02:42:18.773-0300    imported 2071 documents
mongo -u aluno -p aluno --authenticationDatabase aluno aluno
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27017/aluno
MongoDB server version: 3.6.2
```

Importando Documentos

```
> show collections
contatos
megasena
produtos
teste
```

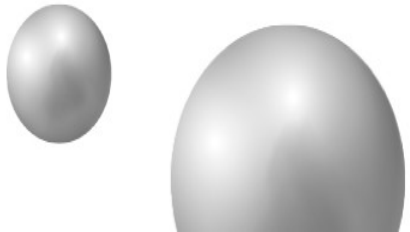
Importando Documentos

```
> db.megasena.find( {_id:1532} ).pretty()
{
        "_id" : 1532,
        "data" : "21/09/2013",
        "dezenas" : [
                42,
                2,
                11,
                23,
                31,
                44
        ],
```



Importando Documentos

```
        "sena_ganhadores" : 3,
        "uf" : [
                "MS",
                "SP",
                "SP"
        ],
        "sena_rateio" : 1934416.36,
        "quina_ganhadores" : 91,
        "quina_rateio" : 19455.14,
        "quadra_ganhadores" : 5676,
        "quadra_rateio" : 445.58
}
```



Exportando Documentos

Para exportar documentos para um arquivo texto pode ser utilizado o utilitário mongoexport.

Atualmente é possível exportar documentos de arquivos nos formatos json e csv.

```
mongoexport -d aluno -u aluno -p aluno -c
megasena --type csv --fields _id,dezenas --out
megasena.csv
2018-08-25T02:45:46.271-0300    connected to:
localhost
2018-08-25T02:45:46.296-0300    exported 2071
records
```