

FIAP GRADUAÇÃO

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Arquiteturas Disruptivas e Big Data

PROF. ANTONIO SELVATICI

SHORT BIO



É engenheiro eletrônico formado pelo Instituto Tecnológico de Aeronáutica (ITA), com mestrado e doutorado pela Escola Politécnica (USP), e passagem pela Georgia Institute of Technology em Atlanta (EUA). Desde 2002, atua na indústria em projetos nas áreas de robótica, visão computacional e internet das coisas, aliando teoria e prática no desenvolvimento de soluções baseadas em Machine Learning, processamento paralelo e modelos probabilísticos. Desenvolveu projetos para Avibrás, IPT e Systax.

PROF. ANTONIO SELVATICI

profantonio.selvatici@fiap.com.br

MongoDB

Relembrando: modelagem por documentos

- Armazenam as entradas de dados como um documento, a princípio no formato XML, JSON ou BSON
 - A noção de documento está intrinsecamente relacionada com um “objeto”
- Em vez de considerar a estrutura de armazenamento como uma tabela, podemos considerá-las como um simples conjunto de documentos (ou objetos), permitindo máxima flexibilidade:
 - Cada documento poderia conter dados arbitrários
 - Permite uma estrutura de aninhada, pois um documento pode estar dentro de outro documento, e assim por diante
- Qualquer estrutura complexa e arbitrária pode se tornar um documento
 - Pense em um objeto Java ou JSON sendo serializado

Exemplos de armazenamento por documentos

Documento XML

```
<objects>
  <object>
    <id> 100 </id>
    <nome> Astolfo </nome>
    <sobrenome> Silva </sobrenome>
    <endereco>
      <rua> Rua das Orquideas </rua>
      <no>23</no>
    </endereco>
  </object>
  <object>
    <id> 101 </id>
    <nome> Maria </nome>
    <sobrenome> Teresa </sobrenome>
    <idade> 49 </idade>
  </object>
</objects>
```

Documento JSON

```
[
  {
    "id": 100,
    "nome": "Astolfo",
    "sobrenome": "Silva",
    "endereco": {
      "rua": "Rua das Orquideas",
      "no": 23
    }
  },
  {
    "id": 101,
    "nome": "Maria",
    "sobrenome": "Teresa",
    "idade": 49
  }
]
```

Considerações sobre modelagem por documentos

- As entradas relativas ao mesmo documento estão armazenadas de forma contígua em disco, de forma que recuperar o documento inteiro é tão custoso quanto recuperar parte do documento
- Considerando que cada documento possui um ID único no banco e indexável, é possível construir relacionamentos de forma fácil usando esse ID para apontar para outros documentos
- São uma extensão do conceito chave-valor, pois cada documento possui um índice que traz como valor o corpo do documento, que por sua vez é constituído de vários pares chave-valor
- Há uma fácil relação entre um documento e a entrada de uma tabela:
 - Cada campo do documento pode ser comparado à coluna de uma tabela, mas com a possibilidade de abrigar sub-documentos
- Possui um schema mais flexível do que o colunar, sendo bastante apropriado para o log de eventos distintos, com a fácil criação de atributos sob demanda
 - Interessante para guardar dados de sensores e outras informações da Internet das Coisas (pense no acréscimo de novos sensores, não previstos originalmente)
 - A interpretação dos campos pode ser feita posteriormente, na aplicação



- Banco de dados NoSQL orientado a documentos
- É um projeto multiplataforma, open-source, escrito em C++, projetado para oferecer
 - Alta performance nas escritas e consultas
 - Alta disponibilidade
 - Fácil escalabilidade;
 - Liberdade de formatos (schema-less)
- Obs: alguns estudos mostram tempos de escrita e consultas 10x menores com relação a bancos relacionais como o MySQL, enquanto outros mostram poucas diferenças. Essas comparação são despropositadas, uma vez que MongoDB e MySQL possuem preocupações diferentes

Um pouco sobre a empresa

- O banco de dados MongoDB foi desenvolvido pela empresa MongoDB Inc. em outubro de 2007;
 - Inicialmente pensado para ser um componente de uma plataforma PaaS (Platform as a Service);
- Seu código foi aberto em 2009 e a companhia passou a oferecer serviço comercial de suporte;
- Passou a ser adotado como software de infraestrutura em várias empresas de grande expressão;
 - Uma extensa lista de empresas que usam o software comercialmente pode ser encontrada em: <https://www.mongodb.com/who-uses-mongodb>
 - No entanto, não está clara a forma como essas empresas usam MongoDB
- Em 2014 se tornou o Sistema de Banco de Dados NoSQL mais utilizado.
- Atualmente fornecem o serviço de banco de dados em nuvem, o MongoDB Atlas

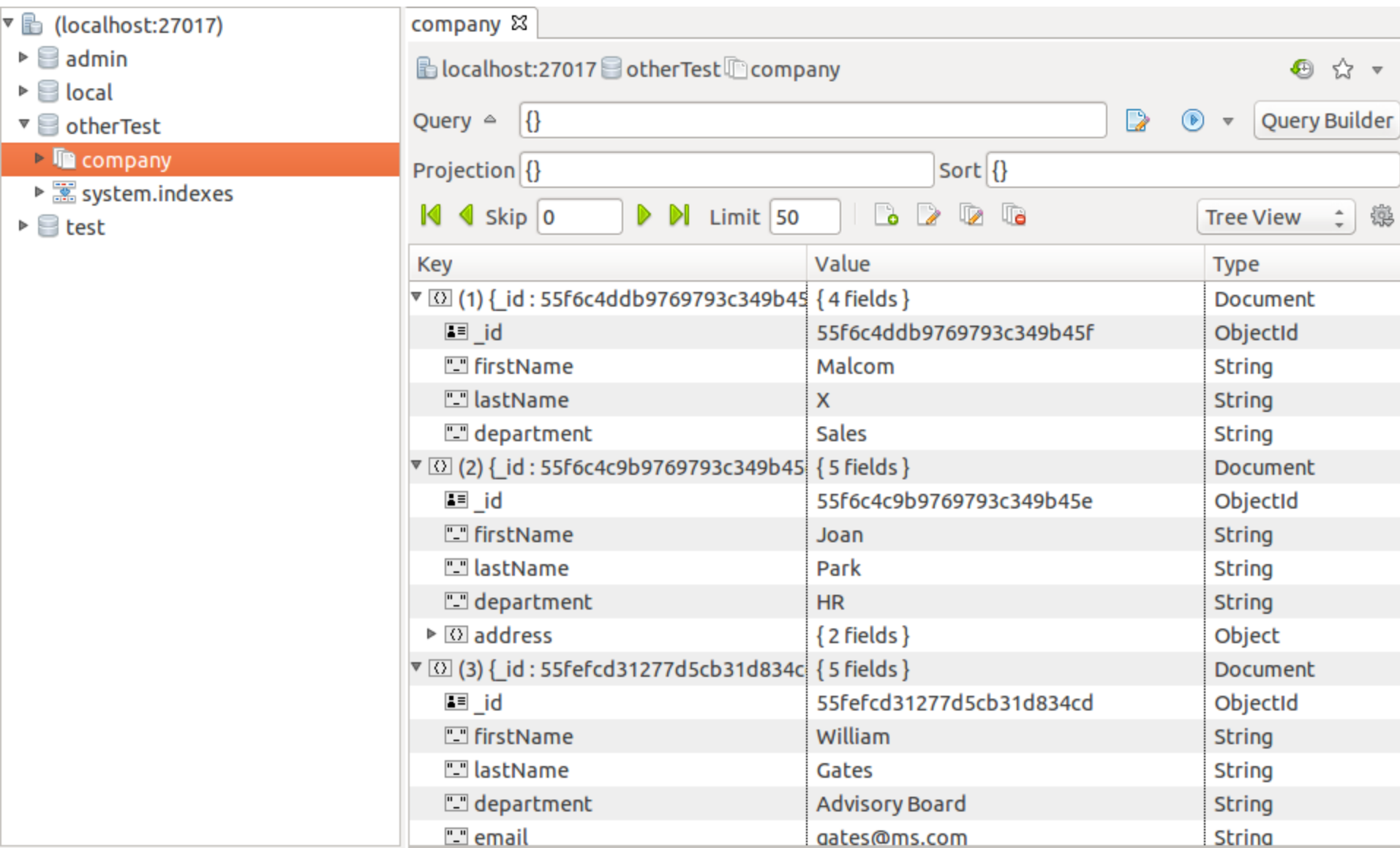
A arquitetura do MongoDB

- Possui uma arquitetura cliente-servidor
 - Nome do servidor: `mongod` (daemon)
 - Nome do cliente padrão: `mongo`
- Podemos usar MongoDB em um único servidor (single-node) ou em vários servidores (cluster)
- Modos de clusterização (podem ser combinados):
 - Replica Set: replicação do conteúdo de um servidor para outros servidores secundários, a fim de evitar perda de dados
 - Sharding: permite a escalabilidade horizontal, fazendo com que os dados sejam divididos entre vários servidores; requer a instânciação do serviço `mongos`
- Autenticação: permite o uso de usuários com autenticação para acesso aos bancos de dados

■ Conceitos básicos empregados pelo MongoDB

- Banco de Dados (database): equivalente ao database dos bancos relacionais
 - É um container físico para armazenar as coleções
 - Cada BD possui seu próprio conjunto de arquivos no sistema operacional
 - Um servidor mongoDB costuma ter vários Bancos de Dados
- Coleção: equivalente ao conceito de tabela do banco relacional
 - É um grupo de documentos do mongoDB
 - Pertence a apenas um Banco de Dados
 - Não requer um esquema; cada documento pode conter diferentes campos
- Documento: equivalente ao registro de um BD relacional
 - É um conjunto de pares chave/valor, representado por um objeto JSON
 - Cada chave equivale ao conceito de coluna ou atributo dos BDs SQL
 - Possui esquema dinâmico, ou seja, não precisam possuir os mesmos campos; campos homônimos podem conter diferentes tipos de dados
- Índice: equivalente ao índice dos bancos relacionais
 - Serve para realizar uma busca rápida a partir de um valor indexado
 - Não possui conceito de chave estrangeira

Exemplos de componentes do MongoDB

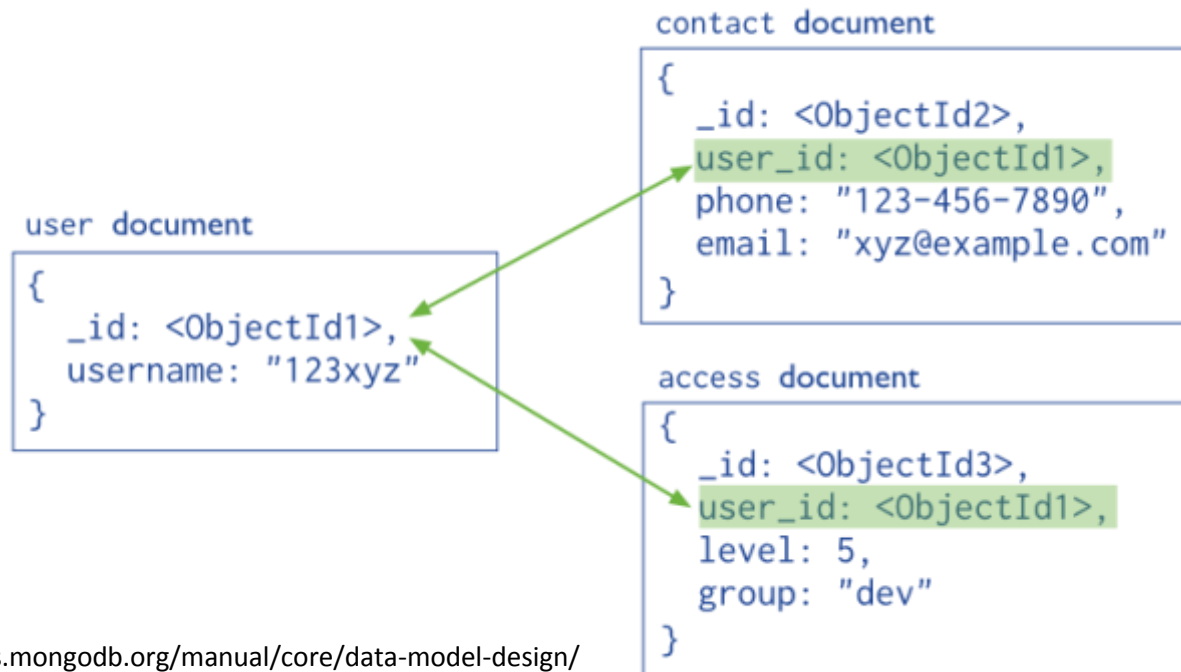


The screenshot displays the MongoDB Compass interface. On the left, a sidebar shows the database structure for 'localhost:27017', including collections 'admin', 'local', 'otherTest', and 'company' (highlighted). The main area shows the 'company' collection with a query of '{}'. The projection is also '{}'. The results are displayed in a table with columns 'Key', 'Value', and 'Type'.

Key	Value	Type
(1) { _id : 55f6c4ddb9769793c349b45f }	{ 4 fields }	Document
_id	55f6c4ddb9769793c349b45f	ObjectId
firstName	Malcom	String
lastName	X	String
department	Sales	String
(2) { _id : 55f6c4c9b9769793c349b45e }	{ 5 fields }	Document
_id	55f6c4c9b9769793c349b45e	ObjectId
firstName	Joan	String
lastName	Park	String
department	HR	String
address	{ 2 fields }	Object
(3) { _id : 55fefcd31277d5cb31d834cd }	{ 5 fields }	Document
_id	55fefcd31277d5cb31d834cd	ObjectId
firstName	William	String
lastName	Gates	String
department	Advisory Board	String
email	wgates@ms.com	String

Modelos de dados

- Modelo de Dados Normalizado:
 - Descreve os relacionamentos através de referências entre os documentos
 - Utilizado para evitar redundância de dados, representar relacionamentos muito-para-muitos complexos e modelar hierarquia de dados;



Modelos de dados

- Modelo de Dados Aninhado (ou Embarcado):
 - Agrega vários documentos em um único;
 - Geralmente garante ótima performance de leitura, mas pode fazer com que os documentos fiquem muito grandes, ocasionando efeitos colaterais;

```
{
  _id: <ObjectId>,
  username: "123xyz",
  contact: {
    phone: "123-456-7890",
    email: "xyz@example.com"
  },
  access: {
    level: 5,
    group: "dev"
  }
}
```

Embedded sub-document

Embedded sub-document

Executando o MongoDB

- O serviço `mongod` deve estar rodando. Caso não esteja, executar:
 - `$> mongod --config <arquivo-de-configuração>`
- Essencialmente, o arquivo de configuração (`mongodb.config`) precisa conter pelo menos uma linha:
 - `dbpath=<PASTA-ARQUIVOS-DB>`
- Alternativamente, podemos executar:
 - `$> mongod --dbpath <PASTA-ARQUIVOS-DB>`
- Atualmente, a empresa MongoDB Inc. desenvolve um cliente de texto, o `mongo`.
 - Para executá-lo na porta padrão no localhost, sem autenticação, chamar:
 - `mongo`
 - Há diversos drivers que permitem a comunicação com o MongoDB diretamente na aplicação, através da linguagem de programação original:
 - <http://docs.mongodb.org/master/applications/drivers/>
- Oficialmente, não há uma interface gráfica suportada pela empresa, mas existem diversas disponíveis open source ou comerciais
 - Vamos usar a `mongochef`: <http://3t.io/mongochef/>
 - Vamos executar os comandos do MongoDB no IntelliShell

■ Executando o MongoDB no laboratório

- Caso o caminho de instalação do MongoDB não esteja configurado no PATH (caminho de execução dos programas) do sistema operacional Windows do lab, é necessário indicar qual é o caminho dos executáveis no sistema de arquivos

- Para executar o servidor, devemos então fazer:

```
C:\opensource\MongoDB-3.0\bin\mongod.exe --dbpath=c:\
```

- Para executar o cliente, ou informar a localização do cliente do MongoDB para outros programas, como o MongoChef, usamos o caminho:

```
C:\opensource\MongoDB-3.0\bin\mongo.exe
```


Manipulando bancos de dados

- Vamos criar um database para testes
 - `use meuprimeirodb`
 - O comando `use` alterna para o DB, que, se não existir, é criado
- Para checar as databases existentes:
 - `show dbs`
 - Atualizar a árvore de componentes com `Ctrl-Shift-R`
- Para mostrar as coleções no DB atual:
 - `db.getCollectionNames()`
- Para apagar o database atual, executar:
 - `db.dropDatabase()`
 - `db` é o objeto JavaScript que permite acessar o banco atual
- Vamos criar o banco de dados teste para importar o seguinte arquivo: unicorns.json
 - Retirado do tutorial: <http://openmymind.net/mongodb.pdf>

Manipulando Coleções

- Para criar uma coleção podemos utilizar o método `db.createCollection(<nome>, <opções>)`, onde:
 - `nome`: string que define o nome da coleção;
 - `opções`: não mandatório, especifica opções como espaço em disco e indexação.
- O MongoDB cria uma coleção automaticamente ao inserir nela uma entrada, sem a necessidade de criá-la explicitamente antes
- O objeto `db.colecao` dá acesso aos métodos de manipulação da coleção `colecao`.
 - **insert**: cria um novo documento
 - **update**: substitui dados de um documentos existentes
 - **find**: encontra entradas de uma coleção através de critérios de busca
 - **remove**: apaga entradas de acordo com o critério de busca
 - **drop**: apaga a coleção
- Assim, para excluir uma coleção basta chamar `db.colecao.drop()`

Inserindo uma nova entrada em uma nova coleção

- Vamos inserir o documento de campo **nome** igual a "João" e **idade** igual a 28 na coleção `idades` do database `primeiroteste`
 - `use primeiroteste`
 - `db.idades.insert({nome:"João", idade: 28})`
- Observar que todos os documentos recebem um campo de índice `_id`, do tipo `ObjectId`
- Vamos apagar a coleção:
 - `db.idades.drop()`
- Vamos apagar o banco
 - `db.dropDatabase()`

Tipos de dados que podem ser usados no MongoDB

Tipo de Dado	Descrição
String	Cadeia de caracteres ASCII
Integer	Armazenamento de valores numéricos
Boolean	Verdadeiro ou falso
Double	Armazenamento de valores de ponto flutuante
Min/ Max keys	Usado para comparar valores em um BSON
Arrays	Armazenamento de vetores, ou lista de valores
Timestamp	Data / Hora (Usado internamente)
Object	Utilizado para documentos aninhados
Null	Armazenamento de valores nulos
Symbol	Como string, utilizado para certas linguagens
Date	Data
Object ID	Armazenamento da identificação do documento
Binary data	Dados binários
Code	Armazenamento de códigos java script
Regular expression	Armazenamento de expressões regulares

Exemplos de uso dos principais tipos de dados

- `use teste`
- ```
db.unicorns.insert({
 name: 'Ambari',
 dob: new Date (1983, 0, 2, 15, 23),
 loves :['apple ', 'carrot ', 'mango'],
 weight :540.5,
 gender:'m',
 vampires :102
})
```
- Observação sobre datas:
  - Uso: `new Date(yyyy, m-1,dd,h,M,s)`
  - O mês do ano começa em 0 (janeiro)
  - A hora é salva sempre com relação ao UTC (Universal Time Clock)

## Encontrando dados: **find**

- Método de `db` que encontra documentos com base em uma query, que nada mais é do que um objeto JSON possuindo seletores
- Um seletor corresponde um atributo JSON que pode estar combinado com palavras-chave (operadores) para montar expressões
- Cada seletor dentro da query é combinado da mesma forma que os campos de consulta SQL com a palavra chave “AND”
  - Quanto mais seletores, mais restritiva é a consulta
- Para encontrar nosso unicórnio, fazemos:
  - `db.unicorns.find({name: "Ambari"})`
- Para encontrar todos os unicórnios fêmeas unicórnio, fazemos:
  - `db.unicorns.find({gender: 'f'})`

## Uso de operadores em queries

- Os operadores `$lt`, `$lte`, `$gt`, `$gte` e `$ne` são usados para operações lógicas:
  - `$lt`, `$lte`: menor do que, menor ou igual a
  - `$gt`, `$gte`: maior que, maior ou igual a
  - `$ne`: diferente de
- Para encontrar todos os unicórnios machos com mais de 700 libras:

```
db.unicorns.find({ gender: 'm', weight: {$gt: 700}})
```
- Neste caso específico, também funcionaria

```
db.unicorns.find({ gender: {$ne: 'f'}, weight: {$gte: 701}})
```
- O operador `$exists` é usado para encontrar documentos com base na existência ou não de algum atributo. Exemplos:
  - `db.unicorns.find ({vampires: { $exists: false }})`
  - `db.unicorns.find ({vampires: { $exists: true }})`

## Buscando em arrays

- Um seletor pode se referir a um valor isolado ou procurá-lo dentro de um array
- O operador `$in` é usado para encontrar múltipla opções de valores dentro de arrays. Por exemplo:
  - `db.unicorns.find({loves: {$in:['apple ','orange ']}})`
  - Encontra quaisquer unicórnios com 'apple' ou 'orange' dentro do atributo loves
- Para usar seletores para ampliar as opções de busca ao invés de restringir, podemos usar o operador `$or` e passar um array de seletores
  - `db.unicorns.find ({  
 gender: 'f',  
 $or: [{ loves: 'apple '}, { weight: {$lt: 500}}]  
})`
  - Encontra todas as unicórnio fêmeas que gostam de 'apple' ou pesam menos de 500 libras



## MongoDB: buscas especiais

- MongoDB pode fazer buscas a partir de expressões regulares e buscas a partir de informações geométricas
- Se não se souber o texto exato, inclusive para buscas indiferentes à caixa (maiúscula ou minúscula), podemos usar expressões regulares para busca em texto através do operador **regex**:
  - `db.unicorns.find({name:{ $regex:'ooo'} })`
  - `db.unicorns.find({loves:{ $regex:/apple/i'}})//ignora caixa`
- Dentre as várias formas do MongoDB fazer buscas por localidade, podemos usar um campo composto pelos atributos x e y como sendo a sua posição.
  - `{ <location field>:{$geoWithin:{$center: [[<x>,<y>] , <radius>]] }`
- Considere o dado:
  - `db.unicorns.update({name:{$regex:'Van'}},{ $set: {loc:[1.5,2.0]}})`
- Para localizá-lo, fazemos
  - `db.unicorns.find({loc:{$geoWithin:{$center:[[1.5,1],1]} })`

## Filtrando as buscas: projection e sort

- Além dos seletores, podemos passar para os métodos `find` e `remove` um segundo argumento, conhecido como **projection**
- Esse parâmetro corresponde à lista de atributos que queremos recuperar ou excluir
- Por exemplo, para recuperar apenas os nomes de todos os unicórnios
  - `db.unicorns.find ({}, {name: 1});`
- Por padrão, o atributo `_id` é sempre retornado. Para excluí-lo explicitamente, fazemos
  - `db.unicorns.find ({}, {name:1, _id: 0})`
- Podemos também ordenar as saídas do método `find` usando o comando `sort`:
- Especificamos os campos que queremos ordenar, usando `1` para ordem crescente e `-1` para decrescente
  - `db.unicorns.find().sort ({ name: 1, vampires: -1})`
  - Ordena por nome primeiro, e então por número de vampiros caçados em ordem decrescente

## Atualizando documentos: update

- Para atualizar ou inserir um novo campo em um documento, usamos o comando `update`
  - `db.unicorns.update ({ name: 'Roooooodles ' }, { $set: { weight: 590 } })`
  - Sem o operador `$set`, a entrada inteira é removida, e dá lugar a um documento com apenas o campo `weight`
- Podemos ainda incrementar um campo numérico com o operador `$inc`, em valores positivos ou negativos
  - `db.unicorns.update ({ name: 'Pilot ' }, { $inc: { vampires: -2 } })`
- Podemos também acrescentar um novo elemento a um array através do operador `$push`
  - `db.unicorns.update ({ name: 'Aurora ' }, { $push: { loves: 'sugar ' } })`

## MongoDB: Opções do comando update

- O comando update do MongoDB permite duas opções importantes: múltiplas atualizações e **upsert** (**update** or **insert**)
- Essas opções devem aparecer em um terceiro argumento de update, que podem ter o valor **true** ou **false**:

```
db.unicorns.update (
 { gender: 'f'},
 {$push: {loves: 'sugar'}},
 {multi: true})
```

- Upserts: caso não exista o documento correspondente, ele é criado

```
db.unicorns.update (
 { name: 'Vanevar'},
 {$inc: {vampires: 10}},
 {upsert: true})
```

## Criando índices para facilitar a busca

- Índices no MongoDB funcionam de modo similar aos de bancos relacionais, aumentando a performance de busca e ordenação
- Criamos índices através de `ensureIndex`:
  - `db.unicorns.ensureIndex({ name: 1});` //índice em ordem crescente
  - `db.unicorns.ensureIndex({ name: 1},{ unique: true });`
- Removemos índices com `dropIndex`:
  - `db.unicorns.dropIndex ({ name: 1});`
- Podemos criar também índices (ou chaves) compostos
  - `db.unicorns.ensureIndex({name:1,vampires: -1});`
- A direção do índice pode não importar para chaves simples, mas pode ser importante em chaves compostas, quando ambos os campos são usados para busca ordenada

## Exercício com MongoDB

- Crie um banco de dados chamado 'cadastro'
- Crie uma coleção chamada 'clientes', e crie os índices "nome" e "cpf"
- Crie dez entradas na coleção 'clientes', contendo os campos
  - nome (String)
  - nasc (Date)
  - endereco : documento com os campos
    - logradouro (String)
    - numero (inteiro)
    - cidade (String)
    - cep (String)
  - cpf: (String)
- Faça as seguintes buscas:
  - Busca por cpf
  - Busca por todos os clientes que nasceram antes de 1990, ordenados pela data de nascimento
  - Busca por todos os clientes que moram em São Paulo, ordenados pelo nome

## REFERÊNCIAS

- MongoDB. <https://www.mongodb.com>
- Karl Seguin. The Little MongoDB Book. <http://openmymind.net/mongodb.pdf>
- Ricardo Rezende. Aula 11: MongoDB. Arquivo: Aula\_11\_NoSQL-Document-MongoDB\_TendenciasBD\_2TBD\_RicardoRezende.pdf





Copyright © 2017 Prof. Antonio Selvatici

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).