

# NoSQL

## O que é NoSQL?

- NoSQL faz referência a sistemas de gerenciamentos de dados que vão além dos bancos de dados relacionais com consulta usando SQL – **Structured Query Language**, antes **Structured English Query Lange** (SEQUEL)
  - “Not-only SQL”, significando que a linguagem a ser usada é irrelevante, o que deve ser enfatizado é a falta da garantia de relacionamento entre dados
  - São conhecidos também como bancos de dados não relacionais
  - Poderiam ser chamados de bancos de dados NoREL ou NoJOIN
  - Várias tentativas de inovar sobre as consultas SQL foram feitas, por exemplo, com OODBMS, mas nenhuma “emplacou” até recentemente
    - Tentativa de manter as restrições e a organização dos bancos relacionais, mas com novas maneiras de recuperação e atualização dos dados
    - Talvez menos seja mais

## O que são SGBD relacionais

- Tradicionalmente, desde meados da década de 1970, os sistemas de gerenciamento de bancos de dados se baseiam na álgebra relacional para a representação de dados
  - Conhecidos como R(elational)DBMS
  - Dados são modelados de acordo com sua natureza específica, independentemente da aplicação em que estão inseridos
  - Dados de naturezas diferentes ficam armazenados em tabelas diferentes, sendo que o mesmo dado não deveria aparecer duas vezes no banco para a manutenção da consistência
  - As tabelas de dados mantêm um relacionamento entre si através das chaves estrangeiras: o tempo de inserção ou recuperação de dados tende a crescer com o volume de dados armazenados e com o número de chaves que o banco precisa checar

## Exemplo de tabelas RDMS

Employee_ID	FirstName	LastName	Email	Deartment_ID
1000	Arun	Jayaram	arun@software developersworld.com	100
1001	Manoj	Shankar	manoj@software developersworld.com	100
1002	Syam	Sundar	syam@software developersworld.com	102

employee table

Deartment_Number	Department_Name
100	HR
101	Software
102	Accounts

department table

### Restrição de mudanças

Department\_Number presentes em todas as tabelas devem ser checadas para evitar quebrar a consistência antes modificar um Department

### Restrição de inserção:

Department\_ID deve ser checado antes de inserir novos Employees

## Bancos de dados transacionais

- Os bancos de dados relacionais são conhecidos também por garantir que suas chamadas são transacionais, ou seja, possuam as características ACID:
  - **Atomicidade:** uma chamada para inserção ou modificação de dados deve ser atômica, ou seja, indivisível do ponto de vista de quem invoca. No caso de falha de execução, o estado do banco não deve ser modificado.
  - **Consistência:** a consistência dos dados deve ser mantida, ou seja, todas as regras de chaves únicas e chaves estrangeiras devem ser verificadas antes da modificação de dados
  - **Isolamento:** chamadas que ocorrem simultaneamente não podem influenciar os resultados umas das outras. Assim, cada chamada só pode acessar o resultado das transações que finalizaram antes de sua invocação. Uma forma de fazer isso é, por exemplo, bloqueando o acesso a chaves primárias que estejam sendo usadas em outras transações
  - **Durabilidade:** o resultado de uma chamada deve ser definitivo, ou seja, se alguma falha de sistema, queda de energia, etc. ocorrer após o término de uma transação, seus resultados não podem ser perdidos

## Contexto dos RDBMS

- RDBMS são, por si só, sistemas complexos, que funcionam de modo autônomo, e que precisam de pessoal qualificado para sua instalação, operação e manutenção
- Os dados a serem usados uma aplicação que usa RDBMS precisam antes de um processo de modelagem para a definição das tabelas e dos relacionamentos, o que engessa o desenvolvimento da aplicação
  - A confiabilidade necessária a um sistema bancário exige que, pelo menos no seu núcleo, sejam empregados RDBMS ou tecnologias já provadas por anos, com metodologias de projetos que prevejam longos ciclos de teste antes que alguma mudança seja posta em produção
  - Novas metodologias ágeis de desenvolvimento de projetos, características das Startups e que começa a ser empregada por empresas mais tradicionais, entram em conflito com a filosofia engessada dos RDBMS, seja por manter uma equipe enxuta, seja pelo engessamento do desenvolvimento de novos trechos de código sem que a modelagem de dados correspondente seja realizada

## Mudanças no cenário do desenvolvimento de software (1)

- Aplicar modificações rápidas no código de uma aplicação requer que a nova configuração dos dados se reflita automaticamente no seu armazenamento
  - RDBMS são baseados em schemas: as tabelas do banco e seus campos são predefinidos, e, de modo geral, não são modificados pela aplicação. Caso a aplicação necessite de mais um campo em uma tabela, todos os dados já inseridos serão modificados
  - Bancos NoSQL são schemaless (sem schema) ou possuem schema flexível, que pode ser modificado sem afetar os dados já presentes no banco de dados

## Exemplo: documentos JSON

Key	Value	Type
▼ {} (1) {_id : 55f6c4c9b9769793c349b45e}	{ 4 fields }	Document
{}_id	55f6c4c9b9769793c349b45e	ObjectId
"firstName"	Joan	String
"lastName"	Park	String
"department"	HR	String
▼ {} (2) {_id : 55f6c4ddb9769793c349b45f}	{ 4 fields }	Document
{}_id	55f6c4ddb9769793c349b45f	ObjectId
"firstName"	Malcom	String
"lastName"	X	String
"department"	Sales	String
▼ {} (3) {_id : 55f6c559b9769793c349b460}	{ 5 fields }	Document
{}_id	55f6c559b9769793c349b460	ObjectId
"firstName"	Willian	String
"lastName"	Gates	String
"email"	gates@ms.com	String
"department"	Council	String



## Mudanças no cenário do desenvolvimento de software (2)

- O uso de uma segunda linguagem para acesso aos dados pode atrapalhar o andamento do processo de modificações ágeis do software, pois exigem uma mudança de paradigma
  - Bancos relacionais são sistemas “stand-alone”, e possuem sua própria linguagem de consulta e inserção de dados (SQL)
  - Alguns frameworks (Hibernate, por exemplo) procuram abstrair a necessidade da programação em SQL, facilitando o processo de desenvolvimento, mas exigem um processo de configuração que pode ser penoso
  - Bancos NoSQL podem ter sua linguagem específica para consulta, mas muitos possuem drivers nas linguagens de desenvolvimento de software (C#, Java, Python) que conectam diretamente ao banco, com pouca configuração necessária
  - **Os dados em bancos NoSQL acabam refletindo diretamente o código que os gerou, liberando o desenvolvedor de amarrar previamente código e modelagem. Isso é BOM ou MAU?**

## Mudanças no cenário do desenvolvimento de software (3)

- Por amarrar as consultas e inserções de dados a schemas rígidos controlados por chaves únicas e chaves estrangeiras, RDBMS são projetados para trabalhar com um número limitado de entradas
  - Sistemas empresariais trabalham com uma quantidade limitada de dados, geralmente referentes a cadastros de funcionários, produtos, processos, etc.
    - **Dados sensíveis, porém finitos**
  - Sistemas profissionais como o da Oracle possuem subterfúgios que envolvem paralelismo e clusterização para ganhar eficiência e aumentar o limite de dados
  - No cenário atual, cresce o número de empresas que atuam diretamente na internet, e portanto não devem impor limites quanto à quantidade de dados que geram ou que capturam de seus usuários, que geralmente são dados que descrevem suas atividades no site (“likes”, postagens, fotos, etc.)
    - **Dados mais relaxados (quanto à consistência!), mas infinitos**

## Como expandir o armazenamento?

- O problema de dados crescendo sem limites só pode ser resolvido com escalabilidade...
  - **Escalabilidade vertical:** corresponderia a aumentar os recursos dos servidores, aumentando a capacidade de armazenamento, o número de discos, número de processadores, memória RAM, etc.
    - **Problema 1:** mudança na configuração dos servidores muitas vezes irá acarretar uma parada no serviço, e algumas vezes a reconfiguração de software pode ser necessária
    - **Problema 2:** no caso de RDBMS, eficiência no acesso e atualizações de dados cai com o número de entradas, limitando o próprio tamanho das tabelas
  - **Escalabilidade horizontal:** corresponde a distribuir os dados entre diversos servidores, bastando acrescentar novos nós ao sistema
    - **Problema:** Teorema CAP (também conhecido como teorema Brewer)

## O Teorema CAP

- Teorema CAP: sistemas distribuídos em rede que compartilham dados só podem oferecer duas dentre três características:
  - Consistência
  - Disponibilidade (*Availability*)
  - Particionamento
- O raciocínio é simples:
  - Se os dados são particionados, dois ou mais servidores devem se comunicar para acesso mútuo
  - Se o servidor A tem os dados alterados, o servidor B ficou inconsistente com A
  - Se a alta disponibilidade for priorizada, o servidor B continua disponível, mas seus dados podem não casar com os dados de A
  - Se a consistência for priorizada, o servidor B deve ficar indisponível até a atualização dos dados para garantia da consistência

## O Teorema CAP e os Bancos NoSQL

- Os requisitos de BigData em geral necessitam de **escalabilidade horizontal** (Volume), privilegiam a **alta disponibilidade** dos dados (Velocidade), e **não** podem estar **restritos a schemas** rígidos (Variedade)
- Por isso, muitos bancos NoSQL possuem a chamada consistência eventual:
  - Não é possível garantir que após atualizações dos dados, todas as chamadas irão refletir o estado final do banco...
  - No entanto, se nenhuma outra atualização for feita, em algum momento os dados convergirão para um estado consistente
  - A consistência é atingida através de processos de resolução de conflitos e de consolidação, pois diversos updates podem ocorrer em dados não consistentes (pense na contagem de acessos de um site, por exemplo)

## Modelagem de dados em bancos NoSQL

- Devido à falta da exigência no relacionamento dos dados, a modelagem dos dados em bancos NoSQL fica livre das regras de normalização que acompanham o modelagem relacional
  - Podemos até construir uma base de dados relacional sem relacionamentos, mas seria como usar o motor de uma Ferrari para mover um Fusca...
    - Caso do engine MyISAM do MySQL, que não usa chaves estrangeiras: consultas e updates mais rápidos, mas não garante consistência entre tabelas
  - Liberdade de regras significa variedade de opções: escolher o tipo de banco de dados envolve conhecer a aplicação para entender qual é a melhor forma de modelagem
  - As aplicações ficam restritas ao tipo de banco de dados escolhidos. No mundo relacional, migrações de sistemas eram possíveis apenas migrando os dados de um banco para outro, mas no mundo NoSQL conversões precisam ser feitas para permitir essa migração

## Estruturas de armazenamento de dados

- As diferentes tecnologias NoSQL em geral usam um dos quatro tipos de estrutura interna para armazenamento de dados:
  - Estrutura Chave-Valor (Key-Value Store)
  - Estrutura Colunar (Column Store)
  - Estrutura por documentos (Document Store)
  - Estrutura em grafo (Graph Store)
- Todas as estruturas acima *permitem* fazer algum tipo de relacionamento entre os dados! O termo **não-relacional** se refere:
  - à maior dificuldade em modelar os relacionamentos (em maior ou menor grau)
  - à falta da verificação desses relacionamentos pelo banco, transferindo à aplicação a responsabilidade sobre a manutenção desses relacionamentos

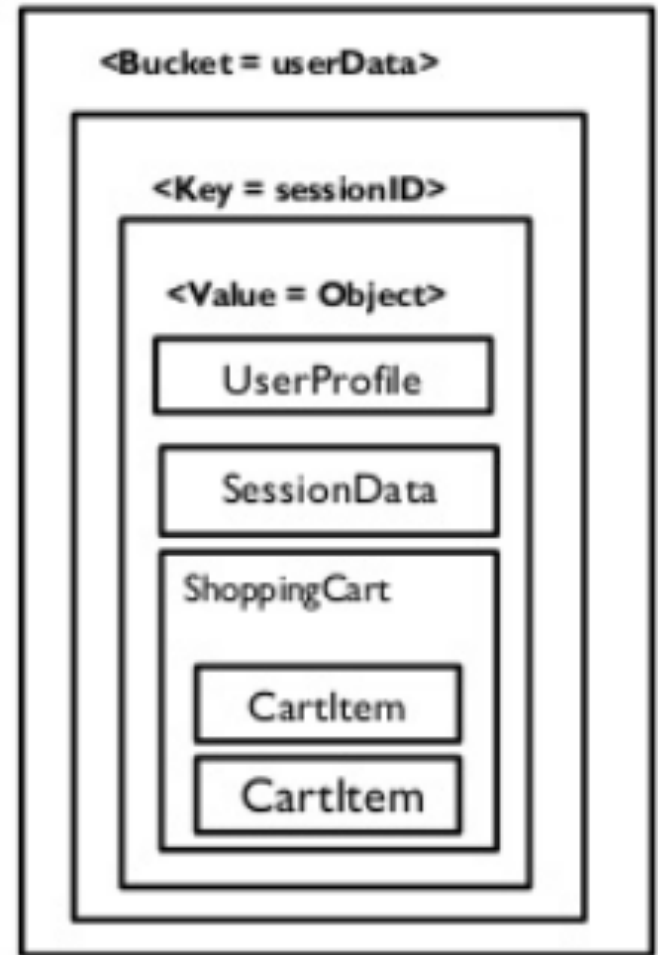
## Estrutura Chave-Valor

- As informações são inseridas como entradas parecidas com um dicionário, possuindo uma chave e um valor:
  - A chave corresponde a uma entrada em um índice de busca
  - O valor corresponde aos dados atrelados a esse índice
  - O próprio modelo MapReduce do Google emprega dados que estão expressos diretamente nessa forma
  - **Vantagem:** as buscas costumam ser muito eficientes, pois a estrutura de armazenamento é muito simples
  - **Desvantagem:** não é possível usar os dados para criar índices de recuperação de informações. Exemplo: (Joaquim, 32), (João, 43), ...
    - Qual a idade de Joaquim? 32. Easy!
    - Quais são as pessoas com mais de 40 anos? Busca extensiva e ineficiente....



## Bancos chave-valor

- Bom para:
  - Informações de sessões
  - Carrinhos de compra
  - Perfis de usuário
- Ruim para:
  - Relacionamentos de dados
  - Recuperação por valores e atributos
  - Operações em conjuntos
- Exemplos: Redis, MemCached, Riak



## Estrutura colunar

- Armazenam as informações organizadas em “column families”, que funcionam como colunas de uma grande tabela
- Evolui do conceito chave-valor para chave-multivalor:
  - Cada linha da família de colunas possui uma chave que aponta para um conjunto de colunas; e cada uma, por sua vez, corresponde a uma par chave/valor
- Não necessita de uma estrutura de tabela previamente criada.
  - Cada registro pode conter diferentes tipos de campos.
  - Cada campo de um registro é considerado uma coluna diferente
  - A column family contém as colunas necessárias para armazenar todos os dados de um registro, e cada coluna pode conter várias entradas vazias (tabela esparsa). Assim, entradas bem diversas são comportadas numa mesma tabela
- As colunas costumam ser agrupadas em famílias, contendo informações que costumam aparecer juntas em um mesmo registro

## Exemplo de chave-multivalor para armazenamento de dados de músicas

a3e64f8f...	title: La Grange	artist: ZZ Top	album: Tres Hombres
8a172618...	title: Moving in Stereo	artist: Fu Manchu	album: We Must Obey
2b09185b...	title: Outside Woman Blues	artist: Back Door Slam	album: Roll Away

## Bancos de dados colunares

- Vantagens:
  - Alta escalabilidade
  - Armazenamento compacto
  - Recuperação de informações por chave e por valor
  - Dados são agrupados por colunas, facilitando a geração de análises estatísticas
- Desvantagens:
  - Recuperação de valores aleatórios de um único registro perde eficiência (cada coluna é procurada separadamente), é necessário planejar os agrupamentos de colunas de acordo com as consultas feitas mais frequentemente
  - Muito ruim para trabalhar com relacionamento de dados, ou garantir transações ACID
- Exemplos:
  - Cassandra, baseado no Google BigTable e DynamoDB;
  - Hbase, parte do ecossistema Hadoop, baseado no Big Table;

## Modelagem por documentos

- Bancos de dados colunares não permitem a construção de sub-colunas
- Para permitir tal aninhamento, precisamos voltar a armazenar cada entrada separadamente, o que permitiria armazenarmos qualquer informação em cada registro
- Em vez de considerar a estrutura de armazenamento como uma tabela, podemos considerá-las como um simples conjunto de documentos, 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

- Considerando que cada documento possui um ID único e indexável no banco, 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

## Exemplos de documentos aninhados

Key	Value	Type
▼ {} (1) { _id : 55f6c4c9b9769793c349b45e }	{ 5 fields }	Document
{} _id	55f6c4c9b9769793c349b45e	ObjectId
{} firstName	Joan	String
{} lastName	Park	String
{} department	HR	String
▼ {} address	{ 2 fields }	Object
{} street	Broadway Ave.	String
{} number	102	String
▼ {} (2) { _id : 55f6c4ddb9769793c349b45f }	{ 4 fields }	Document
{} _id	55f6c4ddb9769793c349b45f	ObjectId
{} firstName	Malcom	String
{} lastName	X	String
{} department	Sales	String



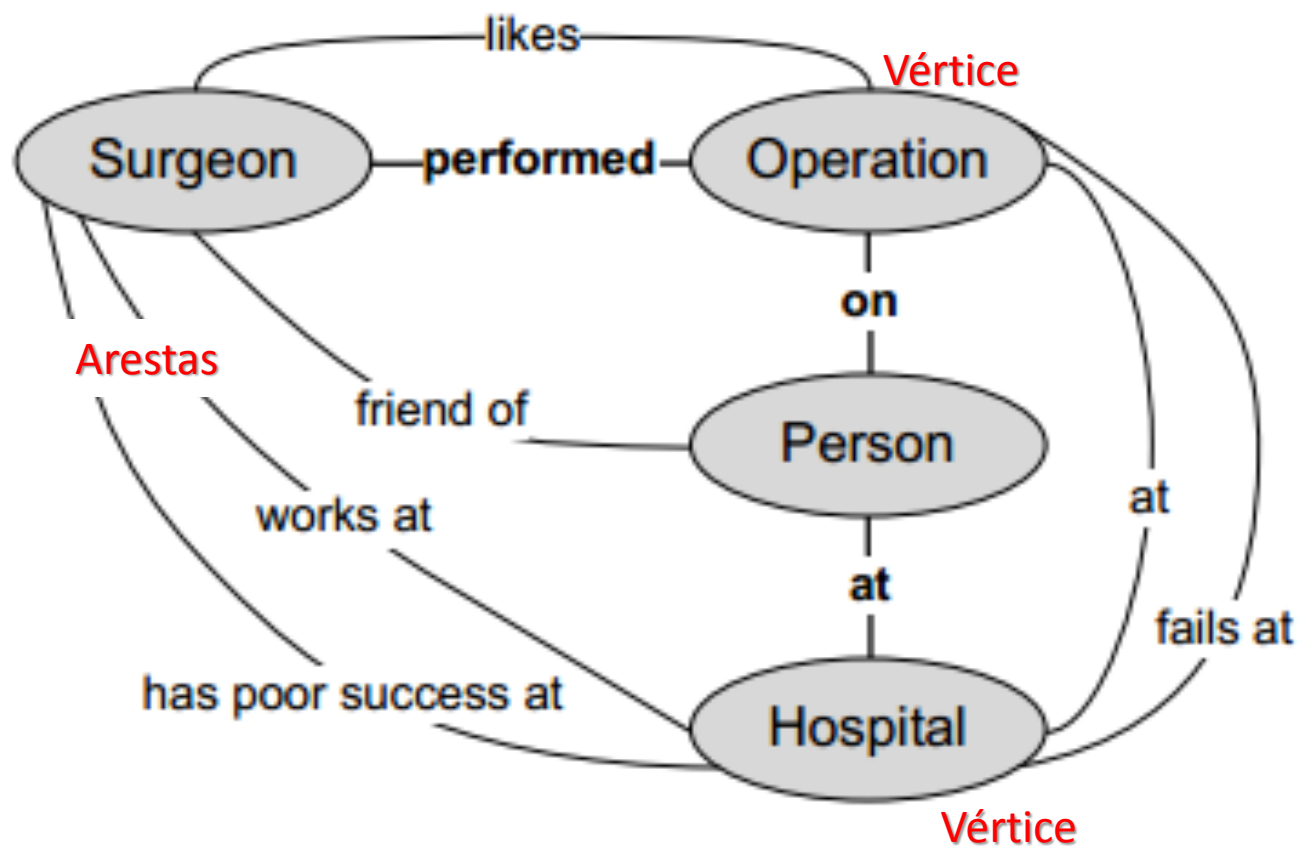
## Bancos de dados por documentos

- Dependendo da frequência em que os dados são observados ou atualizados, um documento pode representar dados não normalizados para efeitos de eficiência de recuperação
  - O endereço pode ser um sub-documento do usuário, por exemplo
  - O maior espaço em disco decorrente da não normalização é compensado pela escalabilidade do banco
- Vantagens:
  - Facilidade de migração do conceito de tabelas em RDBMS para coleções de documentos
  - Facilidade de criação de relacionamentos (verificados pela aplicação)
  - Facilidade de consultas por valores de atributos
- Desvantagens:
  - Uma simples consulta ou alteração resulta na recuperação de todo o documento
  - A garantia de transações ACID torna-se bastante complexa e custosa
- Exemplos:
  - MongoDB, CouchDB, MarkLogic (enterprise), Cloudant

## Modelagem por grafos

- Modelagem de dados que difere das abordagens baseadas em chave-valor por estar focada na relação entre os dados
- A presença de um schema, que determina a forma como as entidades são descritas, é algo opcional.
- As entidades do banco que possuem relações entre si são modeladas através de vértices e arestas de um grafo:
  - **Vértice:** entidade em si, que pode estar representada por qualquer tipo de schema
  - **Aresta:** qualifica a relação entre as entidades
- O interessante da modelagem em grafo é que o *descasamento de impedância* entre a estrutura de dados usados nas tarefas de análise de relacionamentos e o modelo de dados é bem baixa
  - Muitos algoritmos de aprendizado e inferência é baseado em grafos

## Exemplo de dados modelados por grafo



## Bancos de Dados em grafos

- Aplicações
  - Relações em redes sociais
  - Mapas de navegação em sites
  - Web-semântica e Criação de ontologias
- Exemplos:
  - **OrientDB**: combina armazenamento de entidades em documentos com relacionamento em grafos
  - **Neo4J**: poderoso BD de grafo escrito em Java, muito popular e totalmente livre de esquemas.

# 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





# mongoDB

- 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

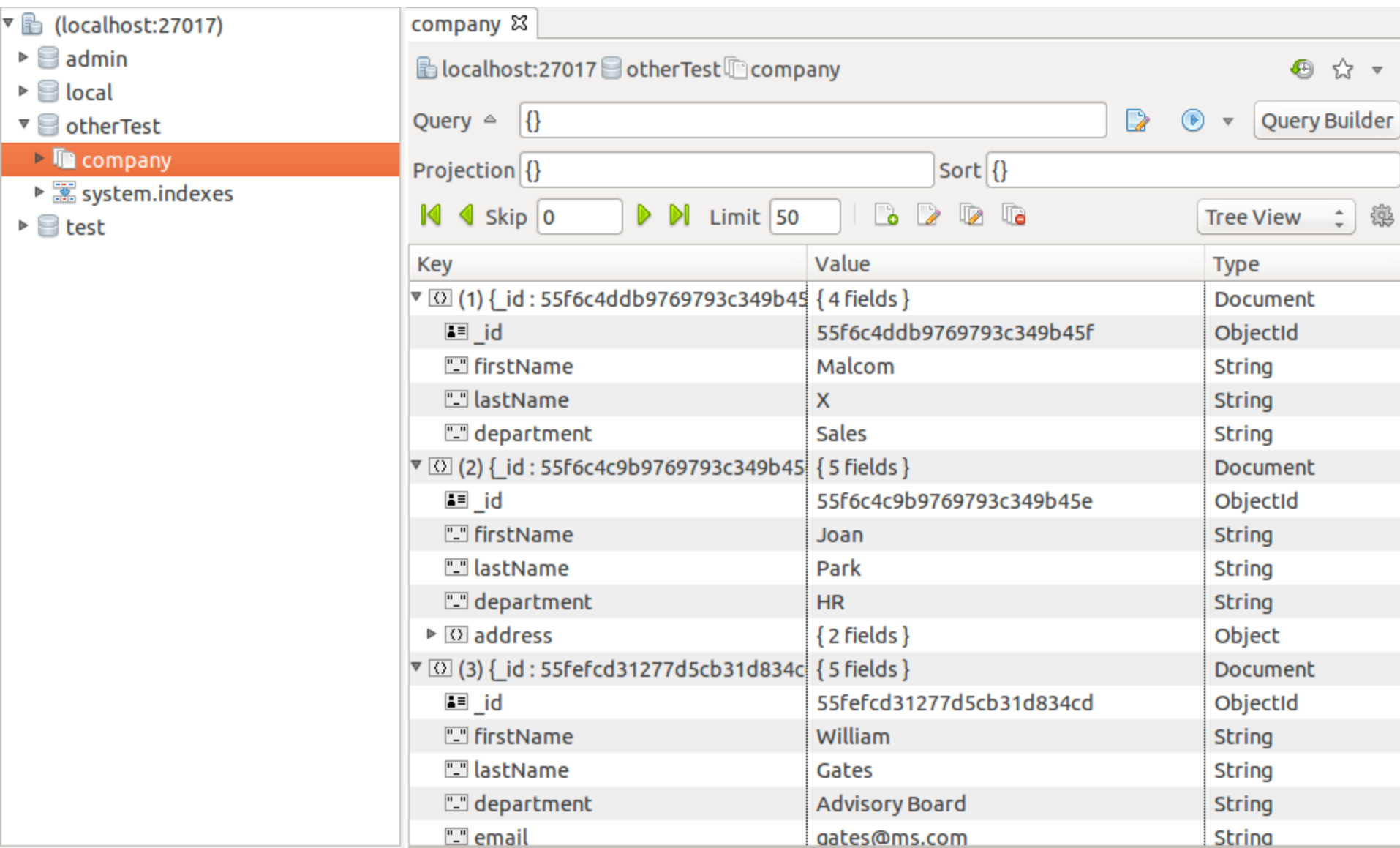
## A arquitetura do MongoDB

- 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 shows the MongoDB Compass interface. On the left, the database structure is displayed, showing a collection named 'company' under the 'otherTest' database. The main area shows the 'company' collection with three documents. The documents are displayed in a table format with columns for Key, Value, and Type.

Query: {}

Projection: {} Sort: {}

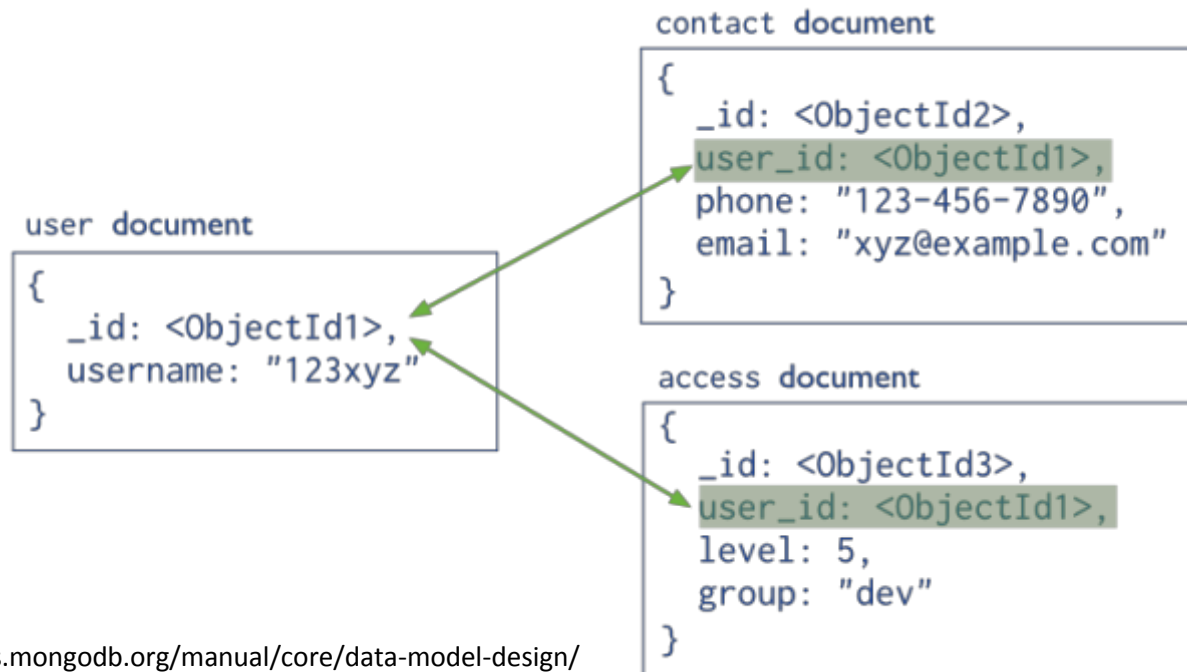
Skip: 0 Limit: 50

Tree View

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;



## 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
<b>Boolean</b>	Verdadeiro ou falso
Double	Armazenamento de valores de ponto flutuante
<b>Min/ Max keys</b>	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
<b>Object ID</b>	Armazenamento da identificação do documento
<b>Binary data</b>	Dados binários
<b>Code</b>	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



**Disciplina:** Arquiteturas Disruptivas e Big Data

**Curso:** Tecnologia em Análise e Desenvolvimento de Sistemas

**Tema – NoSQL e MongoDB**

1 – Quais são os 3 V's do Big Data? E os 5 V's? Explique.

2 – Quais são as quatro camadas tecnológicas de Big Data? Onde se encaixam os conceitos de IaaS, PaaS e DaaS?

3 – Como os bancos NoSQL se encaixam dentro do contexto de Big Data, e como contribuem para o desenvolvimento de aplicações nas metodologias ágeis?

4 – Quais são os quatro tipos de bancos de dados NoSQL? Qual é a estrutura usada por cada um deles para armazenar os dados?

5 – Escreva exemplos de chamadas do cliente padrão do MongoDB para

a) Criar um banco de dados e uma coleção

---

---

b) Inserir um documento em uma coleção no MongoDB

---

c) Encontrar documentos em uma coleção do MongoDB, escolhendo um dos campos para ordenação em ordem crescente / decrescente

---

d) Atualizar documentos em uma coleção, inserindo ou modificando o valor de um campo

---