

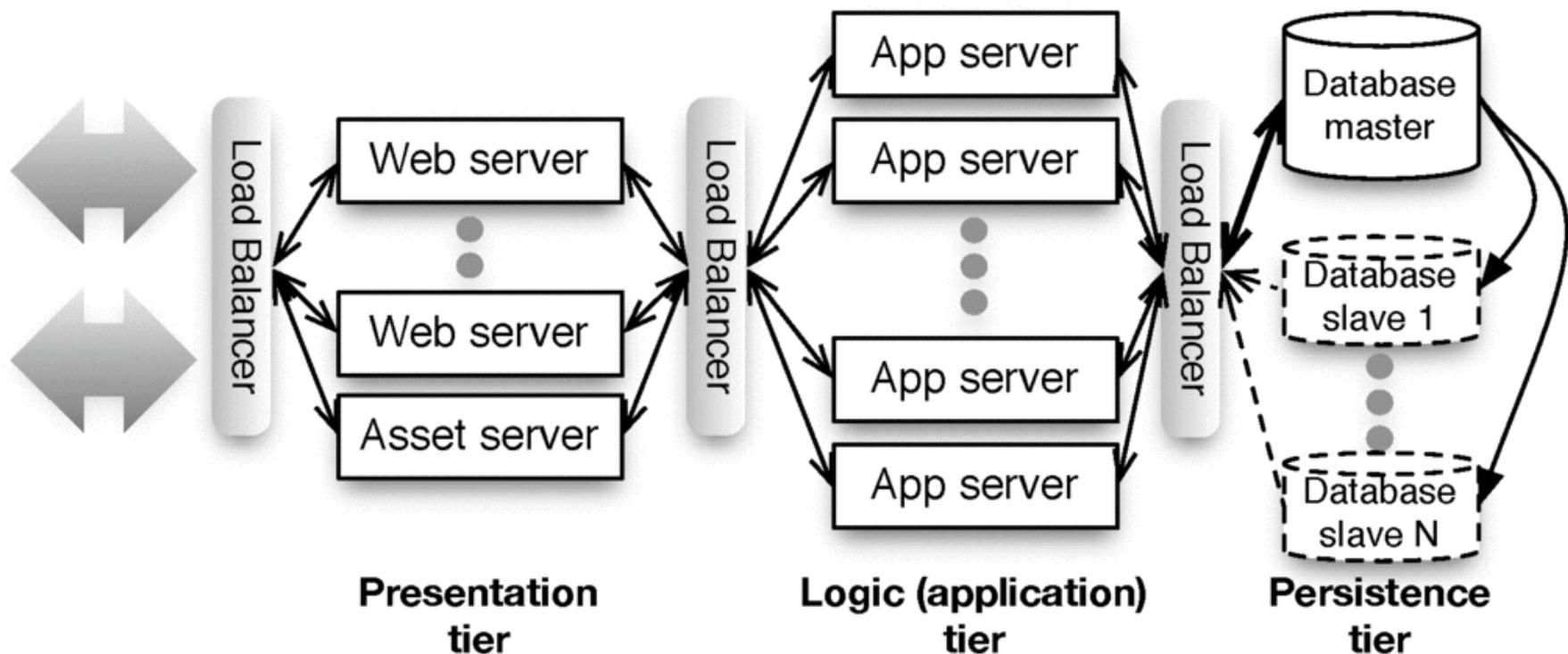
Desenvolvimento de Software para Persistência



Prof. Regis Pires Magalhães
regismagalhaes@ufc.br

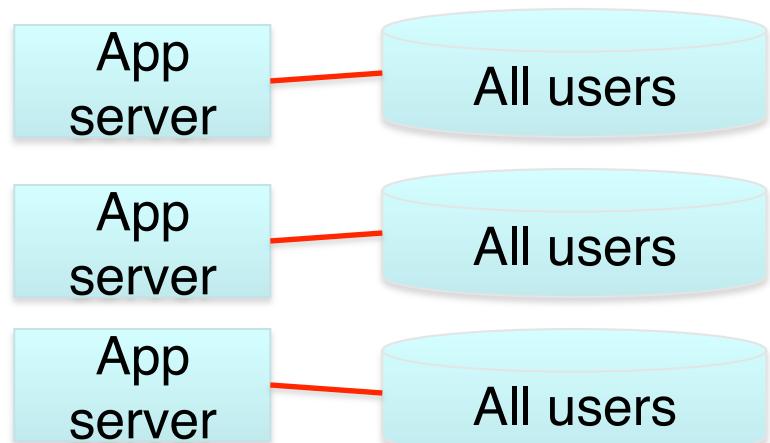
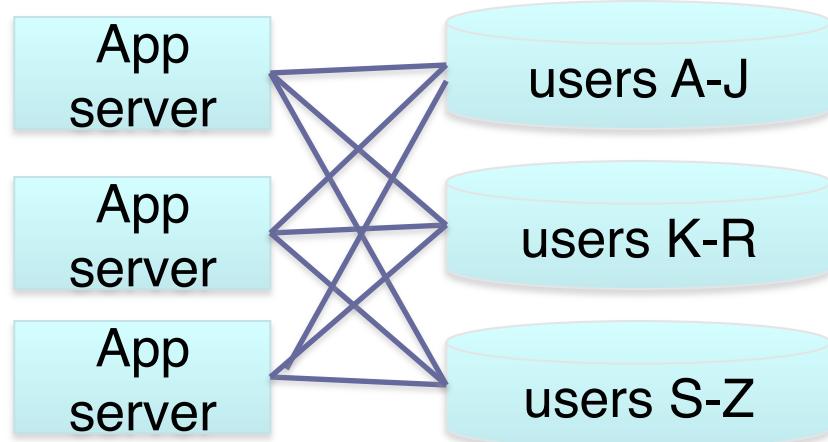


Arquitetura “Shared nothing”



Sharding x Replicação

- Particionamento dos dados em pedaços independentes.
 - Escala bem.
 - Ruim quando operações usam mais de uma tabela.
- Replicar tudo
 - Consultas multi-tabelas rápidas.
 - Difícil escalar
 - Escritas deve ser propagadas para todas as cópias.
 - Dados podem ficar temporariamente inconsistentes.



MongoDB



- Modelo de documento.
- Início do desenvolvimento em Outubro de 2007.
- Primeira versão pública em Fevereiro de 2009.
- Escrito em C++.
- Software livre (open-source):
<https://github.com/mongodb/mongo>
- Mantido por 10gen (suporte comercial).

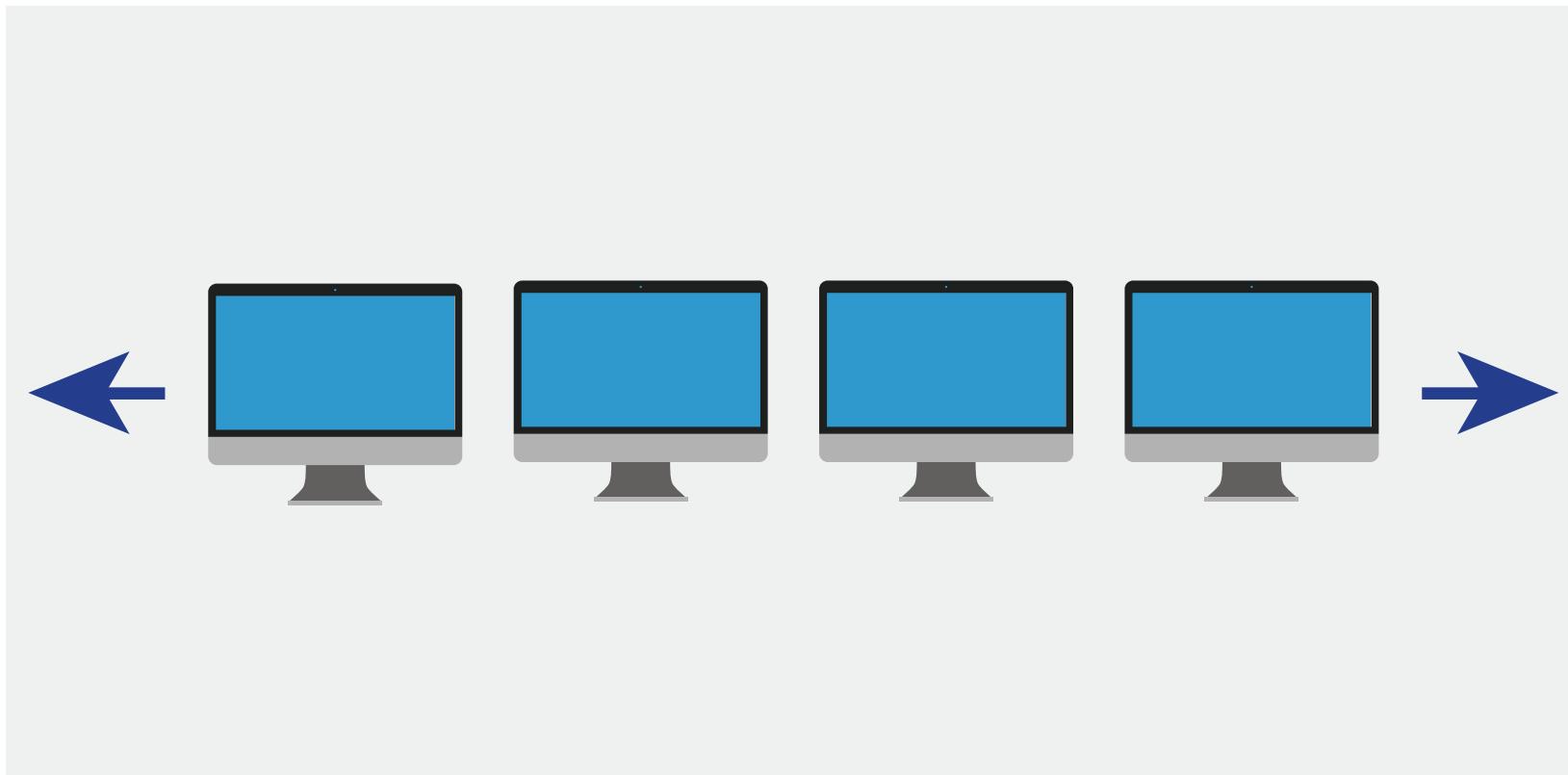
MongoDB



- Preenche a lacuna entre bancos chave-valor altamente rápidos e escaláveis e os bancos relacionais tradicionais.
- Nome derivado do adjetivo *hu(mongo)us*.
- Documentos de uma coleção podem ser heterogêneos.

Escalabilidade horizontal

- Escalonamento horizontal através de auto-sharding.
 - Distribuição dos dados em “milhares de nós” com balanceamento automático de carga e dados e recuperação de falha automática.



MongoDB



- Consultas ad-hoc
- Agregação em tempo real
- Recursos geoespaciais
- Esquema flexível
- Map/Reduce
- Atualizações rápidas in-place
- Índices
- Replicação e alta disponibilidade
- GridFS
- Suporte para a maioria das linguagens
 - Drivers traduzem BSON para tipos nativos.

<http://api.mongodb.org/>



MongoDB - Quando usar?

- MongoDB FAQ
 - <http://docs.mongodb.org/manual/faq/fundamentals/>
 - **What are typical uses for MongoDB?**
 - MongoDB has a general-purpose design, making it appropriate for a large number of use cases. Examples include content management systems, mobile applications, gaming, e-commerce, analytics, archiving, and logging.
 - **Do not use MongoDB for systems that require SQL, joins, and multi-object transactions.**

MongoDB



- BD de documentos JSON.
 - Tecnicamente os dados são armazenados em uma forma binária de JSON, chamado de BSON.
 - Dados serializados como BSON para “*parsing*” rápido.
- Um documento Mongo é semelhante a uma linha de tabela relacional sem esquema e cujos valores podem aninhar uma profundidade arbitrária qualquer.
- Modelo documento = menos trabalho.

Documento BSON

- Conjunto de campos
 - Pares chave-valor.
 - Chave: um nome (string)
 - Valor: um tipo básico
 - String, int, float, date, binary, array, document, ...

Usado em...

- CERN, globo.com, ...

bit.ly

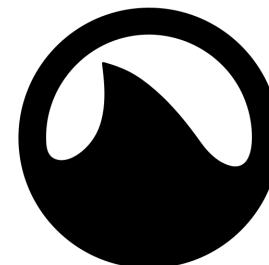
github
SOCIAL CODING

The New York Times

Disney

foursquare™

shutterfly

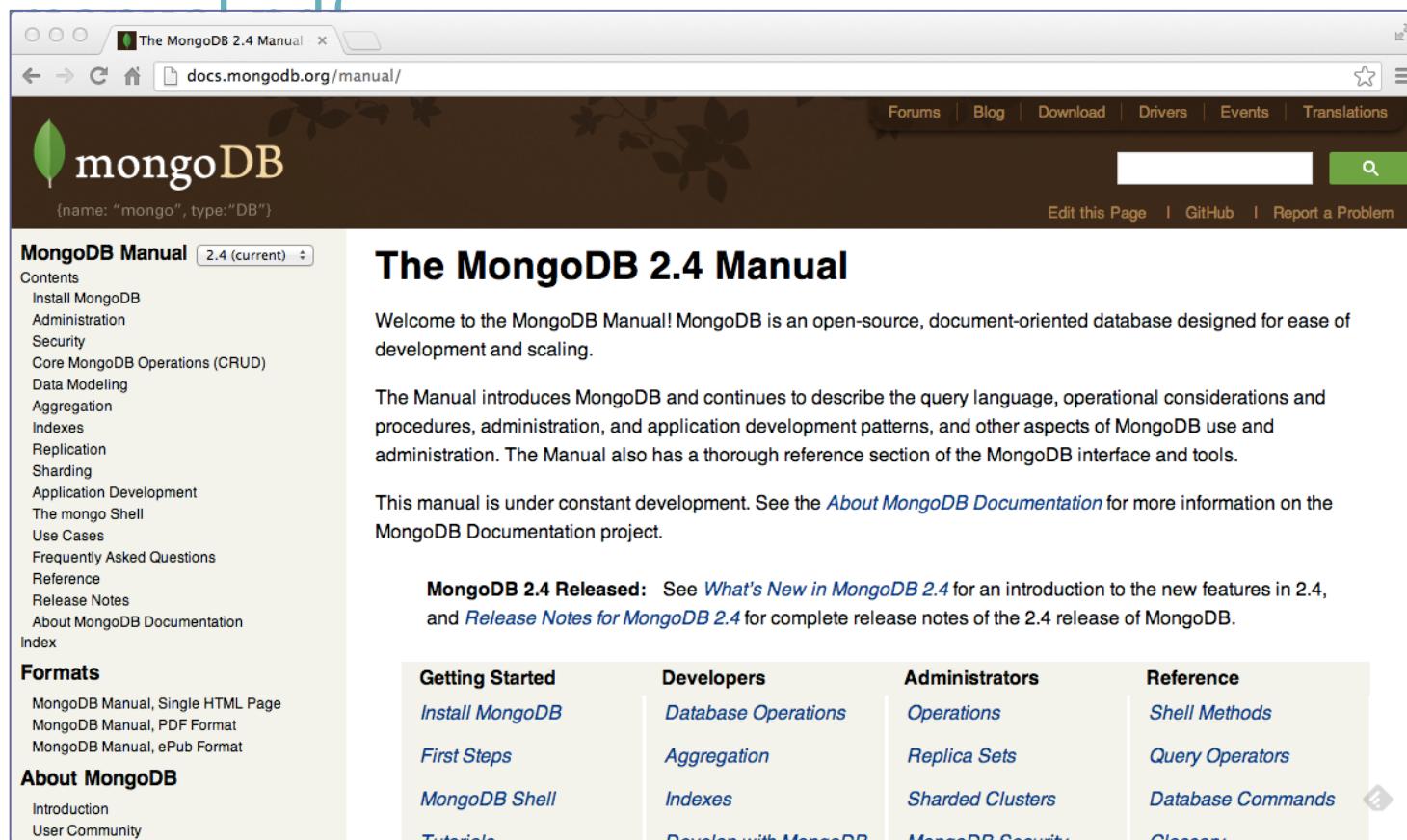


Grooveshark

sourceforge

Documentação

- <http://docs.mongodb.org/manual/>
- PDF:
<http://docs.mongodb.org/manual/MongoDB-Manual.pdf>



The screenshot shows the MongoDB 2.4 Manual website. The header features the MongoDB logo and navigation links for Forums, Blog, Download, Drivers, Events, and Translations. A search bar is also present. The main content area is titled "The MongoDB 2.4 Manual". It includes a welcome message, a section about the manual's purpose, and a note about its development. Below this, there is a "MongoDB 2.4 Released" section with links to "What's New in MongoDB 2.4" and "Release Notes for MongoDB 2.4". At the bottom, there is a footer with links to various MongoDB resources and a "Glossary" link.

The MongoDB 2.4 Manual

Welcome to the MongoDB Manual! MongoDB is an open-source, document-oriented database designed for ease of development and scaling.

The Manual introduces MongoDB and continues to describe the query language, operational considerations and procedures, administration, and application development patterns, and other aspects of MongoDB use and administration. The Manual also has a thorough reference section of the MongoDB interface and tools.

This manual is under constant development. See the [About MongoDB Documentation](#) for more information on the MongoDB Documentation project.

MongoDB 2.4 Released: See [What's New in MongoDB 2.4](#) for an introduction to the new features in 2.4, and [Release Notes for MongoDB 2.4](#) for complete release notes of the 2.4 release of MongoDB.

Getting Started	Developers	Administrators	Reference
Install MongoDB	Database Operations	Operations	Shell Methods
First Steps	Aggregation	Replica Sets	Query Operators
MongoDB Shell	Indexes	Sharded Clusters	Database Commands
Tutorials	Develop with MongoDB	MongoDB Security	Glossary

Terminologia

- Database == Database
- Collection == Table
- Document == Row

Terminologia



RDBMS		MongoDB
Table, View	→	Collection
Row	→	Document
Index	→	Index
Join	→	Embedded Document
Foreign Key	→	Reference
Partition	→	Shard

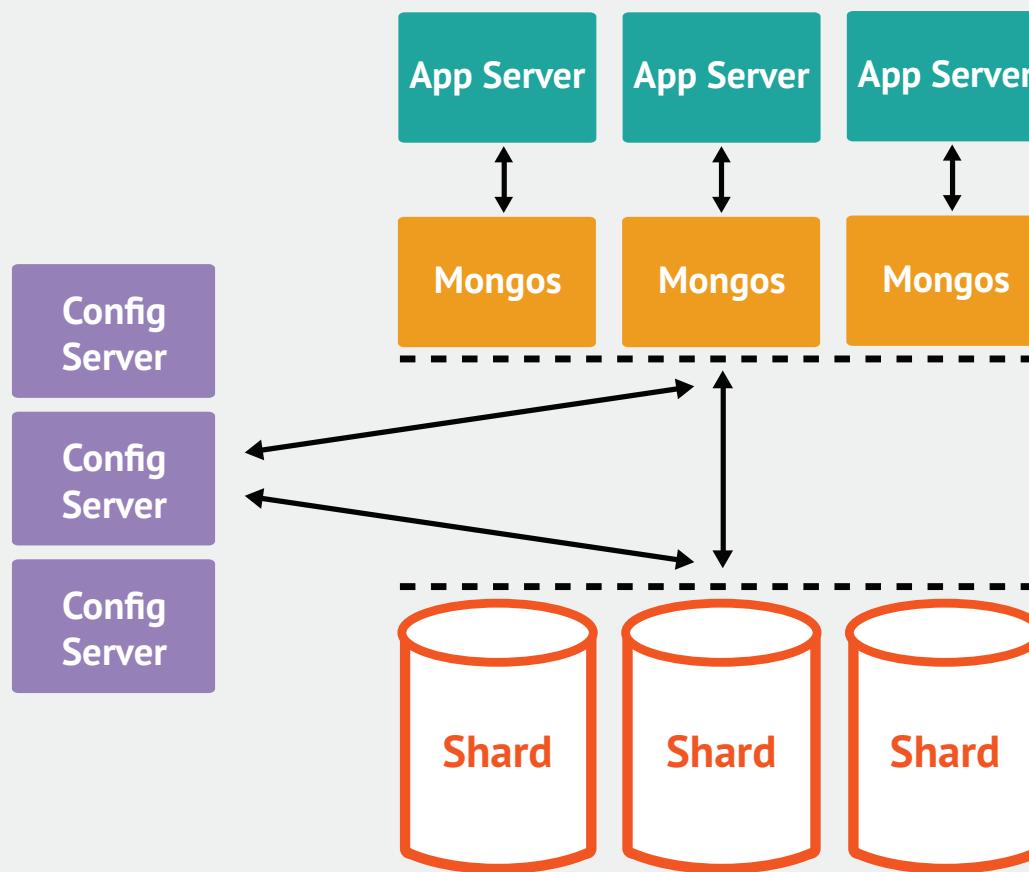
Shard = caco, fragmento, estilhaço.

Sharding



- Doc. MongoDB: “Particionamento dos dados entre múltiplas máquinas de modo a preservar a ordem.”
- Realizado por coleção e não no banco inteiro.
- Ao ser configurado para sharding, o MongoDB automaticamente detecta que coleções crescem mais rápido que a média, de modo que se tornam sujeitas a sharding, enquanto outras coleções podem ainda residir em nós simples.
- MongoDB detecta desbalanceamento na manipulação de shards e pode automaticamente rebalancear os dados para reduzir a distribuição de carga desproporcional.

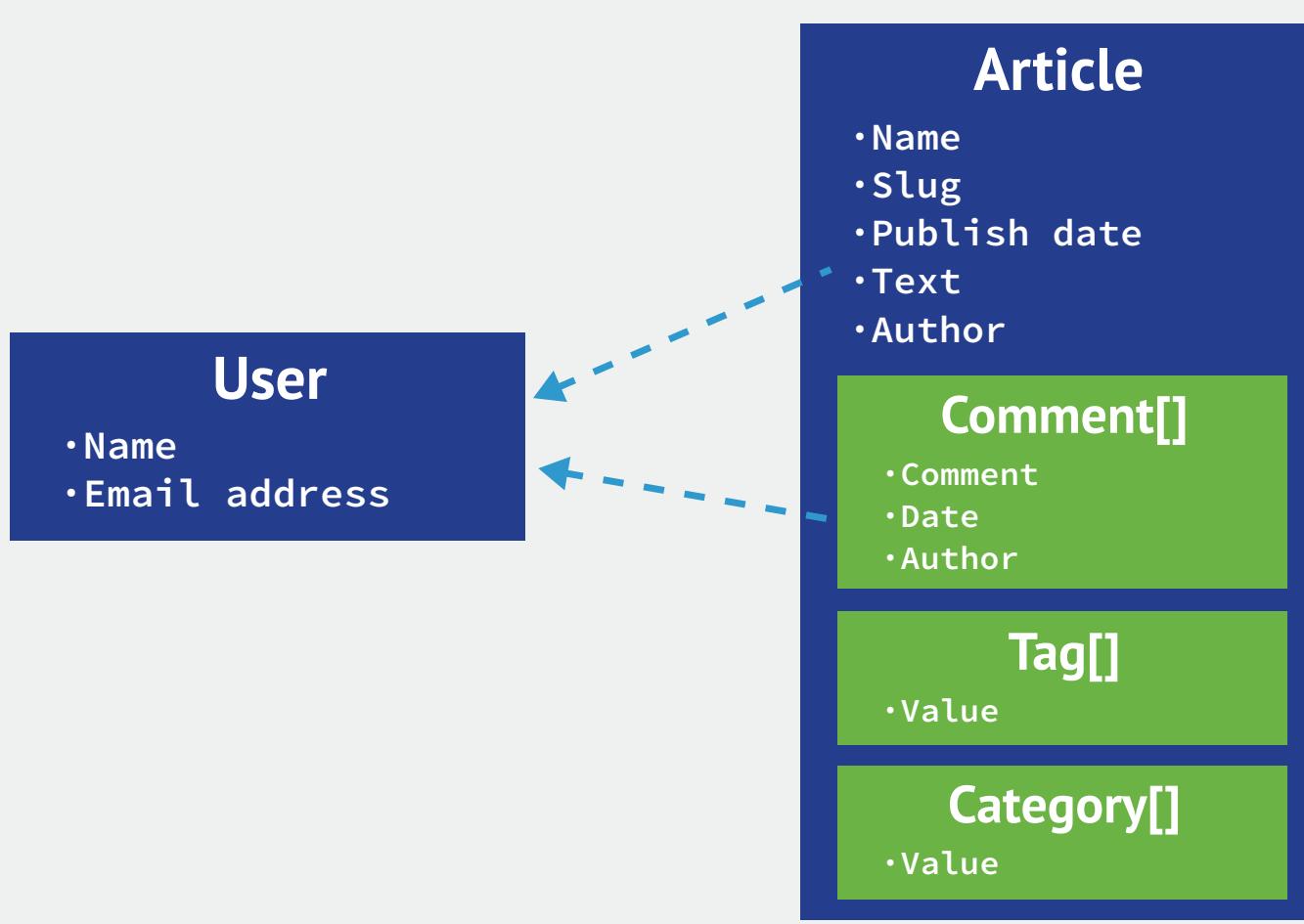
Sharding



Banco de dados de documento

- Um documento é basicamente um array associativo, semelhante a:
 - Objeto JSON
 - Array associativo do PHP
 - Dicionário do Python
 - Hash do Ruby
 - Map do Java

Esquema



Documento exibido como JSON

```
> printjson( db.towns.findOne({"_id" : ObjectId("4d0b6da3bb30773266f39fea")}) )
```

```
{  
    "_id" : ObjectId("4d0b6da3bb30773266f39fea"),  
    "country" : {  
        "$ref" : "countries",  
        "$id" : ObjectId("4d0e6074deb8995216a0309e")  
    },  
    "famous_for" : [  
        "beer",  
        "food"  
    ],  
    "last_census" : "Thu Sep 20 2007 00:00:00 GMT -0700 (PDT)",  
    "mayor" : {  
        "name" : "Sam Adams",  
        "party" : "D"  
    },  
    "name" : "Portland",  
    "population" : 582000,  
    "state" : "OR"  
}
```

Collection

Database

Identifier

Document

Download

- <http://www.mongodb.org/downloads>

The screenshot shows the MongoDB website's download page. At the top, there's a navigation bar with links to Get Involved, User Groups, Forums, Events, Twitter, Facebook, and Blog. Below the navigation is a search bar and a green search button. The main header features the MongoDB logo and the text '{name: "mongo", type:"DB"}'. Below the header, there are four main navigation links: MONGO DOCS (MongoDB Documentation), TRY IT OUT (Try The Online Shell), DOWNLOADS (Download MongoDB), and DRIVERS (Get The Latest Drivers). The DOWNLOADS link is highlighted with a green arrow icon. The page title is 'MongoDB Downloads'. A subtext below the title states: 'This table lists MongoDB distributions by platform and version. We recommend using these binary distributions, but there are also packages available for various package managers.' The main content is a table with columns for Windows, Linux, Mac OS X, and Solaris. It shows two rows: 'Production Release (Recommended)' and 'Nightly'. Each row contains links for 64-bit and 32-bit versions, along with changelog and source links.

	Windows	Linux	Mac OS X	Solaris
Production Release (Recommended)				
2.4.7 - 10/21/2013	64-bit 2008R2+ ? 64-bit ? 32-bit ?	64-bit 32-bit ?	64-bit	64-bit
Changelog Release Notes SOURCE TGZ ZIP				
Nightly	64-bit 2008R2+ ? 64-bit ? 32-bit ?	64-bit 32-bit ?	64-bit	64-bit
Changelog SOURCE TGZ ZIP				

Usando - servidor



```
$ tar xvf mongodb-linux-x86_64-2.4.7.tgz  
$ cd mongodb-linux-x86_64-2.4.7/bin  
$ mkdir -p /data/db  
$ ./mongod
```

Na versão 32 bits recomenda-se usar a opção **--journal** para maior segurança dos dados.

Na versão 64 bits, journaling já é o padrão.

Em caso de problema para iniciar o MongoDB, usar:

./mongod --repair

Usando - cliente



```
$ mongo
MongoDB shell version: 2.4.4
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  http://docs.mongodb.org/
Questions? Try the support group
  http://groups.google.com/group/mongodb-user
> db.test.insert({text: 'Welcome to MongoDB'})
> db.test.find().pretty()
{
    "_id" : ObjectId("51c34130fb5d7261b4cdb55"),
    "text" : "Welcome to MongoDB"
}
```

Usando

- Executando o servidor no Windows:
 - C:\mongodb\bin\mongod.exe
- Executando o cliente no Windows:
 - C:\mongodb\bin\mongo.exe
- Executando de forma geral:
 - mongo
- Usar um bd chamado book:
 - use book
- Exibir bancos:
 - show dbs
- Exibir coleções:
 - show collections
- Ajuda:
 - help

MongoDB como Serviço do Windows

- Instalando:
 - `md C:\mongodb\log`
 - `echo logpath=C:\mongodb\log\mongo.log > C:\mongodb\mongod.cfg`
 - `C:\mongodb\bin\mongod.exe --config C:\mongodb\mongod.cfg --install`
- Iniciando o serviço:
 - `net start MongoDB`
- Parando o serviço:
 - `net stop MongoDB`
- Removendo o serviço:
 - `C:\mongodb\bin\mongod.exe --remove`

Instalação no Ubuntu

- sudo apt-get install mongodb

ou

```
$ mkdir -p /data/db  
$ wget http://downloads.mongodb.org/linux/  
mongodb-linux-x86_64-2.4.6.tgz  
$ tar -xf mongodb-linux-x86_64-2.4.6.tgz  
$ mongodb-linux-x86_64-2.4.6.tgz/bin/mongod
```

Ferramentas

- Shell
 - bin/mongo
- Importar
 - bin/mongoimportjson
- Exportar
 - bin/mongoexport
- Backup
 - bin/mongodump
- Restore
 - bin/mongorestore

Interface Web

- <http://localhost:28017/>
- Para funcionar REST edite o arquivo de configuração (/etc/mongodb.conf no Ubuntu) e acrescente a seguinte linha:
 - `rest=true`
 - Ou ativar a opção `--rest` ao iniciar o mongod
- `http://localhost:28017/`
- `http://localhost:28017/database/collection/?limit=5`
- `http://localhost:28017/database/collection/?filter_key=value`

Inserindo dados



```
db.towns.insert({  
    name: "New York",  
    population: 22200000,  
    last_census: ISODate("2009-07-31"),  
    famous_for: [ "statue of Liberty", "food" ],  
    mayor : {  
        name : "Michael Bloomberg",  
        party : "I"  
    }  
})
```

CRUD simples com MongoDB

```
# inserindo um documento
a = {posicao:1, descricao:"Primeira letra do alfabeto"}
db.alfabeto.save(a)

# inserindo um novo documento
db.alfabeto.save({posicao:5, letra:"e"})

# buscando um documento
documento = db.alfabeto.findOne({posicao:1})

# alterando o documento buscado
documento.bobagem = "Veja, estou incluindo um novo atributo
completamente inútil!"
db.alfabeto.save(documento)

# excluindo um documento
db.alfabeto.remove(documento)
```

Usando...

```
> use testing  
switched to db testing  
  
> db.colors.insert({name: 'red', primary:true})  
> db.colors.insert({name: 'green', primary:true})  
> db.colors.insert({name: 'blue', primary:true})  
> db.colors.insert({name: 'purple', primary:false})  
> db.colors.insert({name: 'orange', primary:false})  
> db.colors.insert({name: 'yellow', primary:false})
```

Consultando e navegando nos resultados

```
> var cursor = db.colors.find()  
> cursor.next()  
  
{  
  "_id" : ObjectId("4bed7aeb0b4acd070c593ba6"),  
  "name" : "red",  
  "primary" : true  
}
```

Cursor

```
{ "_id" : ObjectId("4bed7af40b4acd070c593ba7"), "name" :  
"green", "primary" : true }  
  
{ "_id" : ObjectId("4bed7af80b4acd070c593ba8"), "name" :  
"blue", "primary" : true }  
  
{ "_id" : ObjectId("4bed7b570b4acd070c593ba9"), "name" :  
"purple", "primary" : false }  
  
{ "_id" : ObjectId("4bed7b6a0b4acd070c593baa"), "name" :  
"orange", "primary" : false }  
  
{ "_id" : ObjectId("4bed7b7d0b4acd070c593bab"), "name" :  
"yellow", "primary" : false }
```

Consulta 1

```
SELECT * from colors WHERE name = 'green'
```

```
> db.colors.find({name: 'green'})
```

```
{ "_id" : ObjectId("4bed7af40b4acd070c593ba7"), "name" :  
"green", "primary" : true }
```

Consulta 2

```
SELECT name from colors WHERE primary = 1
```

```
> db.colors.find({primary:true}, {name:true})
{ "_id" : ObjectId("4bed7aeb0b4acd070c593ba6"), "name" : "red" }
{ "_id" : ObjectId("4bed7af40b4acd070c593ba7"), "name" : "green" }
{ "_id" : ObjectId("4bed7af80b4acd070c593ba8"), "name" : "blue" }
```

Consulta 3

```
> db.colors.find({name:/l/})
{ "_id" : ObjectId("4bed7af80b4acd070c593ba8"), "name" : "blue",
"primary" : true }
{ "_id" : ObjectId("4bed7b570b4acd070c593ba9"), "name" : "purple",
"primary" : false }
{ "_id" : ObjectId("4bed7b7d0b4acd070c593bab"), "name" : "yellow",
"primary" : false }
```

Ordenação / Paginação

```
> db.colors.find({primary:true}).sort({name:1}).limit(1)
{ "_id" : ObjectId("4bed7af80b4acd070c593ba8"), "name" : "blue", "primary" : true }

> db.colors.find({primary:true}).sort({name:-1}).limit(1)
{ "_id" : ObjectId("4bed7aeb0b4acd070c593ba6"), "name" : "red", "primary" : true }

> db.colors.find({primary:true}).sort({name:1}).skip(1).limit(1)
{ "_id" : ObjectId("4bed7af40b4acd070c593ba7"), "name" : "green", "primary" : true }
```

Inserção

```
> db.people.insert({name:'John', age:28})  
> db.people.insert({name:'Steve', age:29})  
> db.people.insert({name:'Steph', age:27})
```

insert x save/update

- A diferença é sutil.
- save de objeto **sem `_id`** = insert.
- save
 - de obj. com `_id` → Atualiza (update)
 - de obj. sem `_id` → Insere (insert)
- Ou seja: save pode cumprir as funções de insert e update.
- insert
 - de obj. com `_id` → lança um erro de chave duplicada.

Consulta

```
SELECT * from people WHERE age > 27
```

```
> db.people.find({age: {$gt: 27}})  
{ "_id" : ObjectId("4bed80b20b4acd070c593bac"), "name" : "John", "age" : 28 }  
{ "_id" : ObjectId("4bed80bb0b4acd070c593bad"), "name" : "Steve", "age" : 29 }
```

```
SELECT * from people WHERE age <= 27
```

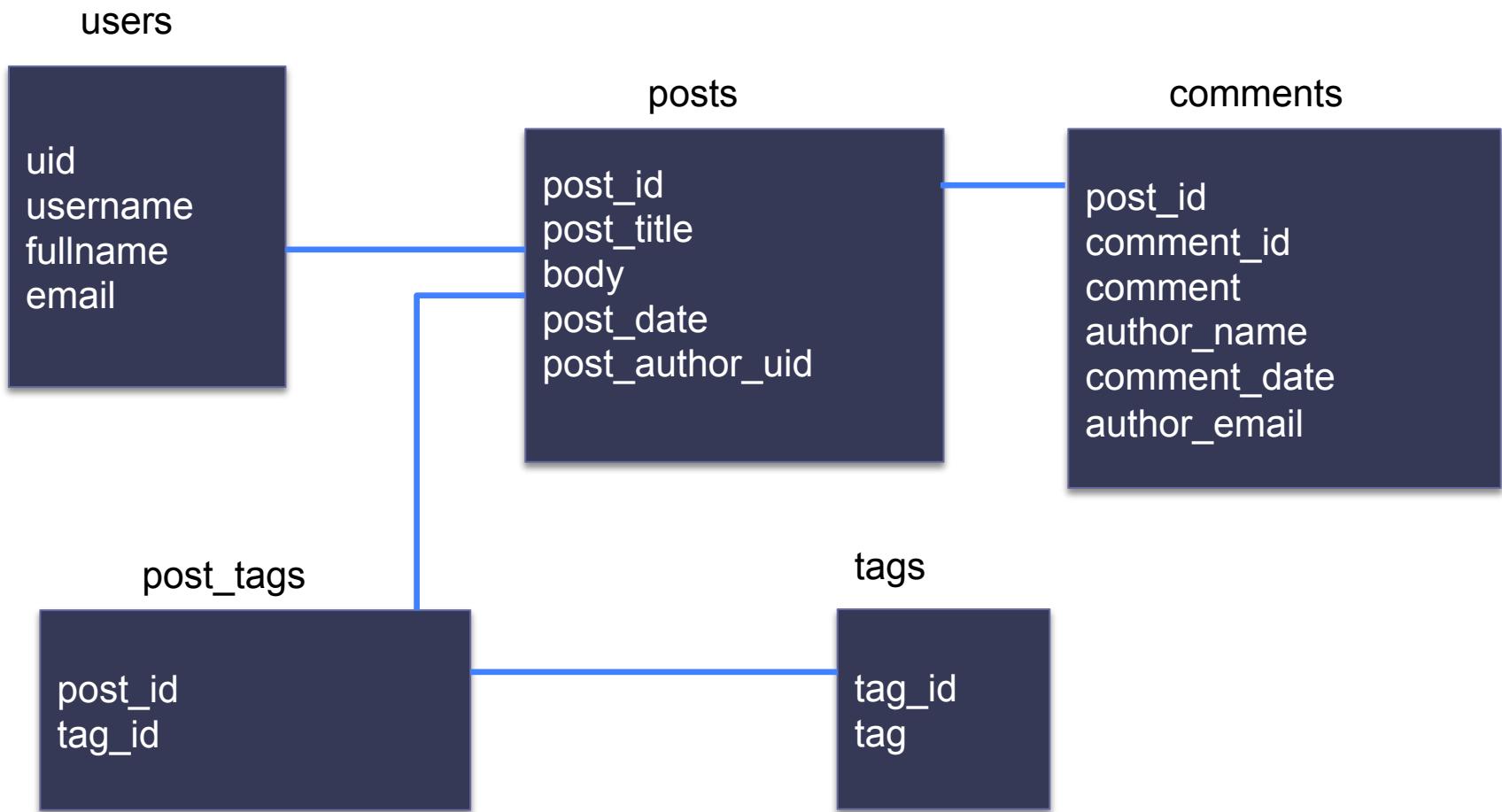
```
> db.people.find({age: {$lte: 27}})  
{ "_id" : ObjectId("4bed80c10b4acd070c593bae"), "name" : "Steph", "age" : 27 }
```

Exemplo - Blog

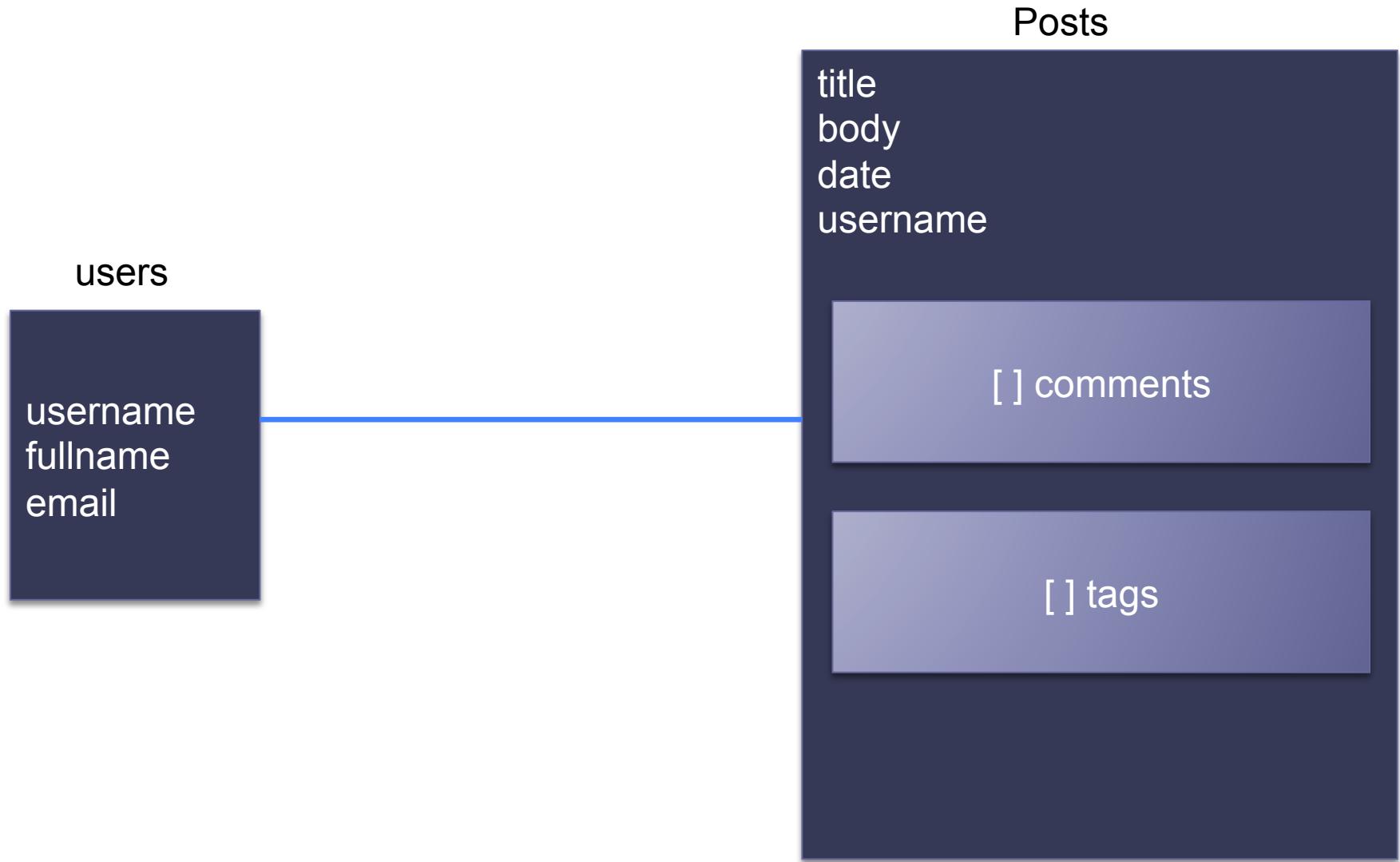


- Users (autores de posts)
- Posts
- Comments
- Tags

Exemplo - Blog - Relacional



Exemplo - Blog - MongoDB



Inicie com um objeto...

```
user = {  
    username: 'ngraham',  
    first_name: 'Norman',  
    last_name: 'Graham'  
}
```

Insira no MongoDB...

```
> db.users.insert(user)
```

Buscando o usuário

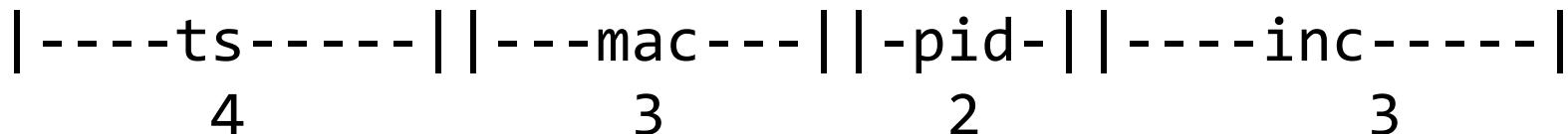
```
> db.users.findOne()
{
  "_id" : ObjectId("50804d0bd94ccab2da652599"),
  "username" : "ngraham",
  "first_name" : "Norman",
  "last_name" : "Graham"
}
```

_id

- É a chave primária no MongoDB
- Automaticamente indexado
- Automaticamente criado como um ObjectId se não fornecido.

ObjectId

- Valor especial de 12 bytes.
- Valor único em todo o cluster.
- Gerado no cliente do mongo.
- ObjectId("50804d0bd94ccab2da652599")



ts – timestamp

mac – identificação da máquina cliente (hash do hostname)

pid – process id que realizou a escrita.

inc – distingue os inserts feitos pelo mesmo processo na mesma máquina cliente.

Criando um post de um blog

```
> db.posts.insert({  
    title: "Hello World",  
    body:  "This is my first blog post",  
    date: new Date("2013-06-20"),  
    username: "ngraham",  
    tags: ["adventure", "mongodb"],  
    comments: []  
})
```

Buscando o post...

```
➤ db.posts.find().pretty()
  "_id" : ObjectId("51c3bafafbd5d7261b4cdb5a"),
  "title" : "Hello World",
  "body" : "This is my first blog post",
  "date" : ISODate("2013-06-20T00:00:00Z"),
  "username" : "ngraham",
  "tags" : [
    "adventure",
    "mongodb"
  ],
  "comments" : [ ]
}
```

Consultando um array

```
> db.posts.find({tags:'adventure'}).pretty()
{
  "_id" : ObjectId("51c3bcddfbd5d7261b4cdb5b"),
  "title" : "Hello World",
  "body" : "This is my first blog post",
  "date" : ISODate("2013-06-20T00:00:00Z"),
  "username" : "ngraham",
  "tags" : [
    "adventure",
    "mongodb"
  ],
  "comments" : [ ]
}
```

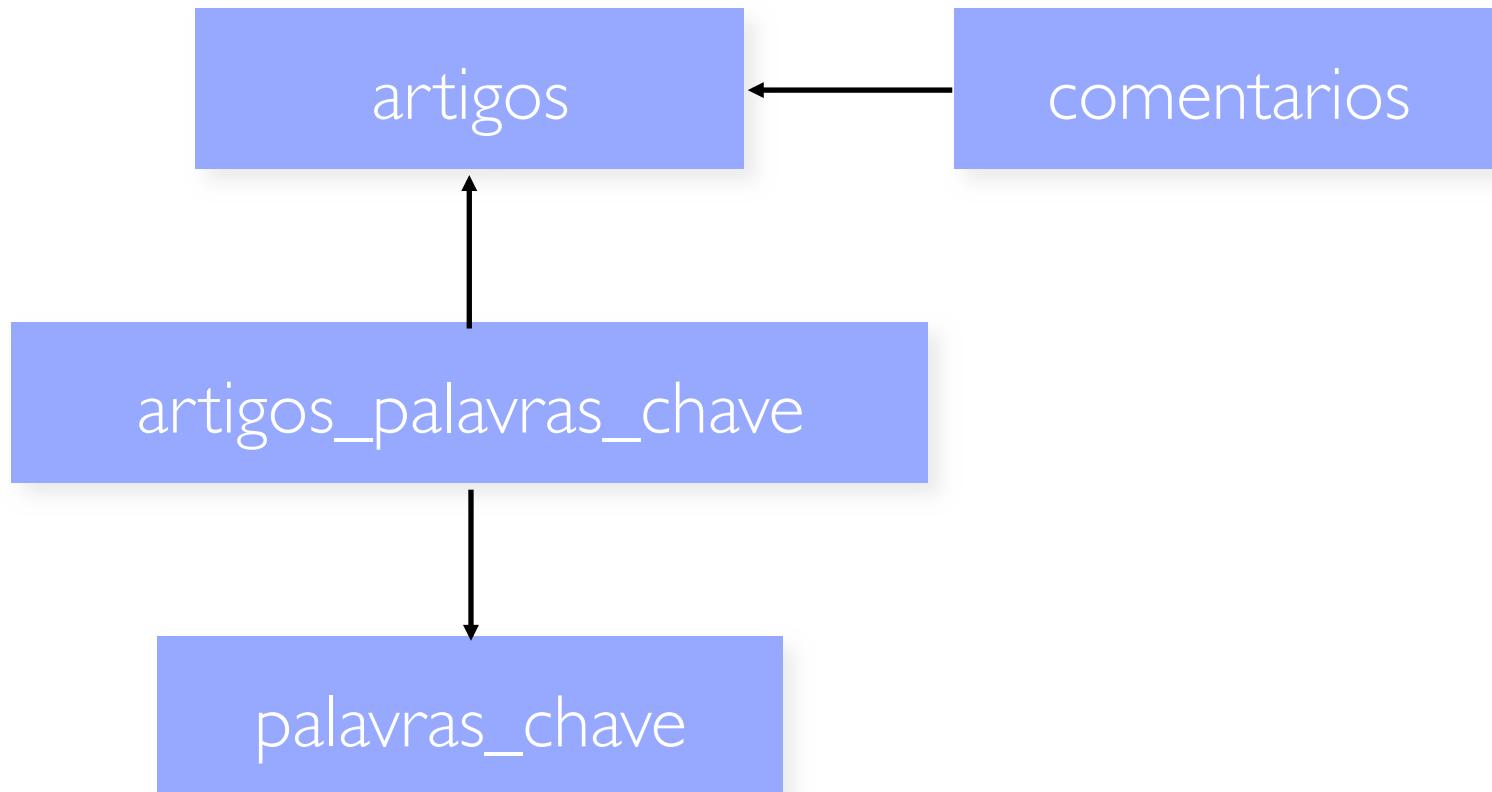
Usando update para adicionar um comentário...

```
> db.posts.update({_id:  
  new ObjectId("51c3bcddfbd5d7261b4cdb5b")},  
  {$push:{comments:  
    {name: 'Steve Blank', comment: 'Awesome Post'}}})  
>
```

Consultando o post atualizado

```
> db.posts.findOne({_id: new ObjectId("51c3bcddfb5d7261b4cdb5b")})  
{  
  "_id" : ObjectId("51c3bcddfb5d7261b4cdb5b"),  
  "body" : "This is my first blog post",  
  "comments" : [  
    {  
      "name" : "Steve Blank",  
      "comment" : "Awesome Post"  
    }  
,  
  "date" : ISODate("2013-06-20T00:00:00Z"),  
  "tags" : [  
    "adventure",  
    "mongodb"  
,  
  "title" : "Hello World",  
  "username" : "ngraham"  
}
```

Outro exemplo - No modelo relacional



Outro exemplo - No MongoDB

artigos

comentarios

palavras_chave

Documento no MongoDB

```
{  
  "_id" : ObjectId("4c03e856e258c2701930c091"),  
  "titulo" : "Semana da Computacao UFC",  
  "atelho" : "semana-da-computacao-ufc",  
  "conteudo" : "A Semana da Computacao da UFC foi um sucesso...",  
  "publicado" : true,  
  "criado_em" : "Mon Oct 4 2010 16:00:00 GMT-0300 (BRT)",  
  "atualizado_em" : "Mon Oct 4 2010 16:00:00 GMT-0300 (BRT)",  
  "comentarios" : [  
    {  
      "autor" : "Julio",  
      "email" : "julio@monteiro.eti.br",  
      "conteudo" : "Gostei muito da semana!",  
      "criado_em" : "Mon Oct 4 2010 17:00:00 GMT-0300 (BRT)"  
    },  
    ],  
  "palavras_chave" : [ "ufc", "computacao" ]  
}
```

Consultas

- ... por uma palavra inteira?

```
db.artigos.find("palavras-chave" : "semana")
```

- ... por parte de uma palavra?

```
db.artigos.find({"titulo" : /udesc/i})
```

- ... por uma palavra dentro de um embutido?

```
db.posts.find({ "comentarios.email" :  
    "julio@monteiro.eti.br" })
```

- Por uma expressão regular

- `db.users.find({ 'lastname': /eckmann$/i })`;

Consultas

```
db.users.find({ 'lastname': /eckmann$/i }, { 'age': true });
```

```
db.users
  .find({ 'lastname': /eckmann$/i })
  .sort({ 'lastname': -1 })
  .limit(10)
  .skip(10*(pageNumber - 1));
```

```
db.users
  .find({ 'lastname': /eckmann$/i })
  .count();
```

Consultas

```
db.scores.find({a: {'$in': [2, 3, 4]}});
```

```
db.scores.find({a: {'$gte': 2, '$lte': 4}});
```

```
db.alunos.count({nome: /C/});
```

ou

```
db.alunos.find({nome: /C/}).count();
```

Comparando com SQL

SQL Statement	Mongo Statement
CREATE TABLE USERS (a Number, b Number)	implicit; can also be done explicitly with <code>db.createCollection("mycoll")</code>
ALTER TABLE users ADD ...	implicit
INSERT INTO USERS VALUES(3,5)	<code>db.users.insert({a:3,b:5})</code>
SELECT a,b FROM users	<code>db.users.find({}, {a:1,b:1})</code>
SELECT * FROM users	<code>db.users.find()</code>

Comparando com SQL

```
SELECT * FROM users WHERE age=33
```

```
db.users.find({age:33})
```

```
SELECT a,b FROM users WHERE age=33
```

```
db.users.find({age:33}, {a:1,b:1})
```

```
SELECT * FROM users WHERE age=33  
ORDER BY name
```

```
db.users.find({age:33}).sort({name:1})
```

```
SELECT * FROM users WHERE age>33
```

```
db.users.find({age:{$gt:33}})
```

```
SELECT * FROM users WHERE age!=33
```

```
db.users.findt.find({age:{$ne:4}})
```

```
SELECT * FROM users WHERE name LIKE  
"%Joe%"
```

```
db.users.find({name:/Joe/})
```

```
SELECT * FROM users WHERE name LIKE  
"Joe%"
```

```
db.users.find({name:/^Joe/})
```

Comparando com SQL

```
SELECT * FROM users WHERE age>33 AND  
age<=40
```

```
db.users.find({ 'age': { $gt: 33, $lte: 40 } })
```

```
SELECT * FROM users ORDER BY name  
DESC
```

```
db.users.find().sort({ name: -1 })
```

```
SELECT * FROM users WHERE a=1 and  
b='q'
```

```
db.users.find({ a: 1, b: 'q' })
```

```
SELECT * FROM users LIMIT 10 SKIP 20
```

```
db.users.find().limit(10).skip(20)
```

```
SELECT * FROM users WHERE a=1 or b=2
```

```
db.users.find( { $or : [ { a : 1 } , { b :  
2 } ] } )
```

```
SELECT * FROM users LIMIT 1
```

```
db.users.findOne()
```

```
SELECT DISTINCT last_name FROM users
```

```
db.users.distinct('last_name')
```

Comparando com SQL

```
SELECT COUNT(*y)  
FROM users where AGE > 30
```

```
db.users.find({age: {'$gt': 30}}).count()
```

```
SELECT COUNT(AGE) from users
```

```
db.users.find({age: {'$exists':  
true}}).count()
```

```
CREATE INDEX myindexname ON  
users(name)
```

```
db.users.ensureIndex({name:1})
```

```
CREATE INDEX myindexname ON  
users(name,ts DESC)
```

```
db.users.ensureIndex({name:1,ts:-1})
```

```
EXPLAIN SELECT * FROM users WHERE z=3
```

```
db.users.find({z:3}).explain()
```

Comparando com SQL

```
UPDATE users SET a=1 WHERE b='q'
```

```
db.users.update({b:'q'}, {$set:{a:1}},  
false, true)
```

```
UPDATE users SET a=a+2 WHERE b='q'
```

```
db.users.update({b:'q'}, {$inc:{a:2}},  
false, true)
```

```
DELETE FROM users WHERE z="abc"
```

```
db.users.remove({z:'abc'});
```

Comparando com SQL

- `SELECT * FROM TABLE WHERE Cond1 = 1 AND Cond2 = "A"`
`db.Collection.find({Cond1: 1, Cond2: "A"})`
- `SELECT * FROM TABLE WHERE Cond1 > 1`
`db.Collection.find({Cond1: {$gt: 1}})`
- `SELECT * FROM TABLE LIMIT 100,10`
`db.Collection.find().skip(100).limit(10)`
- `SELECT count(1) FROM TABLE`
`db.Collection.count()`
- `SELET * FROM TABLE WHERE Cond1 IS NOT NULL`
`db.Collection.find({Cond1: {$exists: true}})`

Atualização

- ... determinado atributo de um registro?

```
db.artigos.update(  
  { "comentarios.email" : "julio@monteiro.eti.br" },  
  { $set:  
    {  
      "comentarios.$.email" : "julio@awegen.com"  
    }  
  }  
)
```

Atualização

Dado original:

```
db.users.save({name: 'Johnny', languages: ['ruby',  
'c']});
```

Atualização total:

```
db.users.update({name: 'Johnny'}, {name: 'Cash',  
languages: ['english']});
```

Atualização parcial:

```
db.users.update({name: 'Cash'}, {'$set': {'age':  
50}});
```

Atualização

Adicionando e removendo itens de array interno:

```
db.users.update({name: 'Sue'}, {'$pull':  
{'languages': 'scala'}});
```

```
db.users.update({name: 'Sue'}, {'$push':  
{'languages': 'ruby'}});
```

Atualização

```
job = db.jobs.findAndModify({  
    query: {inprogress: false, task: "calculateProfile"},  
    sort: { priority: -1 },  
    update: { $set: {inprogress: true, started: new Date()} }  
    new: true  
});
```

Remoção

```
db.scores.remove();
```

```
db.users.remove({name: 'Sue'});
```

Operadores

- \$gt
- \$gte
- \$lt
- \$lte
- \$ne
- \$in
- \$nin
- \$mod
- \$all
- \$size
- \$exists
- \$type
- \$elemMatch
- \$not
- \$where

Usando operadores

- Maior que (\$gt):

```
terceiraIdade = db.pessoas.find({ "age":  
{ $gt: 30 } })
```

- Incluindo (\$in):

```
interessante = db.artigos.find({ "tags":  
{ $in : ["mongodb", "interessante"] } })
```

- Não incluindo (\$nin):

```
todo = db.tarefas.find({ "status": { $nin :  
[ "em execucao", "terminado" ] } })
```

Funções

- Usando funções arbitrárias (com \$where):

```
db.artigos.find({ $where : function() {  
    return this.acessos % 2 == 0  
} })
```

- Usando agrupamento (com \$group):

```
db.artigos.group({  
    "key" : { "hits" : true },  
    "initial" : { "count": 0 },  
    "reduce" : function(obj, prev) {  
        prev.count++;  
    }  
})
```

Operações atômicas

- \$set
- \$unset
- \$inc
- \$push
- \$pushAll
- \$pop
- \$pull
- \$pullAll
- \$addToSet
- \$rename

Operações atômicas

- Incrementando com \$inc

```
db.artigos.update(  
  { _id : new ObjectId("4c041...")},  
  { $inc: {"hits": 1} }  
)
```

- Atualizando

```
db.posts.update({}, { $set : { "hits" : 0 }})
```

Tipos de dados

- Array, Binary, Boolean, DateTime, DB Reference, Embedded Object, Integer, Null, ObjectId, RegExp, String, Symbol, Timestamp.

Tipos de dados

```
> db.people.insert({  
  name : 'John',  
  awesome : true,  
  shows : ['Dexter', 'LOST', 'How I Met Your Mother'],  
  info : {  
    age : 28,  
    home: 'South Bend, IN',  
    dob : (new Date('November 25, 1981'))  
  }  
})
```

Tipos de dados

```
> var me = db.people.findOne({name: 'John'})  
> me.name  
John  
> me.awesome  
true  
> me.shows[1]  
LOST  
> me.info.age  
28  
> me.info.dob.getFullYear()  
1981
```

Tipos de dados

```
> db.people.find({'info.age': 28})  
{ "_id" : ObjectId("4bed9cba0b4acd070c593bc5"),  
  "name" : "John" }  
  
> db.people.find({shows:'Dexter'})  
{ "_id" : ObjectId("4bed9cba0b4acd070c593bc5"),  
  "name" : "John" }  
  
> db.people.find({shows:{$in:['Dexter', 'LOST']}})  
{ "_id" : ObjectId("4bed9cba0b4acd070c593bc5"),  
  "name" : "John" }
```

Bulk inserts

- Para melhorar o desempenho em caso de inserção de muito documentos, pode-se criar um array de documentos e usar o método insert() para inserir o array de documentos e não apenas um documento de cada vez.

Índices



Índices

- Lento para escrita, mas rápido para leitura.
- Crie índices por onde você busca.
- Qualquer propriedade.
- Sobre documentos aninhados.
- Sobre arrays.

Índices

// crescente

```
> db.colors.ensureIndex({name: 1})
```

// composto

```
> db.colors.ensureIndex({name: 1, created_at: -1})
```

// único (unique)

```
> db.colors.ensureIndex({email: 1}, {unique: true})
```

// decrescente

```
> db.colors.ensureIndex({created_at: -1})
```

// non-blocking in background

```
> db.colors.ensureIndex({name: 1}, {background: true})
```

Sem índice

```
db.items.find({tags: "dog"}).explain();
```

```
{
```

```
  "cursor" : "BasicCursor",
```



```
  "nscanned" : 6,
```

```
  "nscannedObjects" : 6,
```

```
  "n" : 6,
```

```
  "millis" : 18,
```



```
  "indexBounds" : {
```

```
}
```

```
}
```

Aplicando índice

- db.items.ensureIndex({tags: 1})
- db.users.ensureIndex({ lastname: 1 })
- db.users.ensureIndex({ 'body.odor': 1 })
- db.users.ensureIndex({ 'body.odor': 1, age: 1 })

Com índice

```
db.items.find({tags: "dog"}).explain();
```

```
{  
  "cursor" : "BtreeCursor tags_1", ←  
  "nscanned" : 6,  
  "nscannedObjects" : 6,  
  "n" : 6,  
  "millis" : 0, ←  
  "indexBounds" : {  
    "tags" : [  
      [  
        "dog",  
        "dog"  
      ]  
    ]  
  }  
}
```

Agregação



Count

```
> db.colors.count()
```

```
6
```

```
> db.colors.count({primary:true})
```

```
3
```

Count

```
{ a: 1, b: 0 }
{ a: 1, b: 1 }
{ a: 1, b: 4 }
{ a: 2, b: 2 }
```

```
db.records.count()
db.records.count( { a: 1 } )
```

Distinct

Collection

```
db.orders.distinct( "cust_id" )
```

```
{  
  cust_id: "A123",  
  amount: 500,  
  status: "A"  
}
```

```
{  
  cust_id: "A123",  
  amount: 250,  
  status: "A"  
}
```

```
{  
  cust_id: "B212",  
  amount: 200,  
  status: "A"  
}
```

```
{  
  cust_id: "A123",  
  amount: 300,  
  status: "D"  
}
```

distinct → ["A123", "B212"]

orders

Distinct

```
> db.colors.distinct('name')
[ "blue", "green", "orange", "purple",
"red", "yellow" ]
```



```
> db.people.distinct('name', {age:28})
[ "John" ]
```

Group

```
{ a: 1, count: 4 }
{ a: 1, count: 2 }
{ a: 1, count: 4 }
{ a: 2, count: 3 }
{ a: 2, count: 1 }
{ a: 1, count: 5 }
{ a: 4, count: 4 }
```

```
db.records.group( {
  key: { a: 1 },
  cond: { a: { $lt: 3 } },
  reduce: function(cur, result) { result.count += cur.count },
  initial: { count: 0 }
} )
```

```
[
  { a: 1, count: 15 },
  { a: 2, count: 4 }
]
```

Aggregation Pipeline

Collection
↓
db.orders.aggregate(
 \$match phase → { \$match: { status: "A" } },
 \$group phase → { \$group: { _id: "\$cust_id", total: { \$sum: "\$amount" } } }
)

```
{  
  cust_id: "A123",  
  amount: 500,  
  status: "A"  
}  
  
{  
  cust_id: "A123",  
  amount: 250,  
  status: "A"  
}  
  
{  
  cust_id: "B212",  
  amount: 200,  
  status: "A"  
}  
  
{  
  cust_id: "A123",  
  amount: 300,  
  status: "D"  
}
```

\$match →

```
{  
  cust_id: "A123",  
  amount: 500,  
  status: "A"  
}  
  
{  
  cust_id: "A123",  
  amount: 250,  
  status: "A"  
}  
  
{  
  cust_id: "B212",  
  amount: 200,  
  status: "A"  
}
```

\$group →

Results

```
{  
  _id: "A123",  
  total: 750  
}  
  
{  
  _id: "B212",  
  total: 200  
}
```

Aggregate

```
select sum(idade) as total from contatos;  
  
db.contatos.aggregate( [  
  { $group: { _id: null,  
              total: { $sum: '$idade' } } }  
] )
```

```
SELECT cust_id, ord_date, SUM(price) AS total  
FROM orders  
GROUP BY cust_id, ord_date
```

```
db.orders.aggregate( [  
  { $group: { _id: { cust_id: "$cust_id", ord_date: "$ord_date" },  
              total: { $sum: "$price" } } }  
] )
```

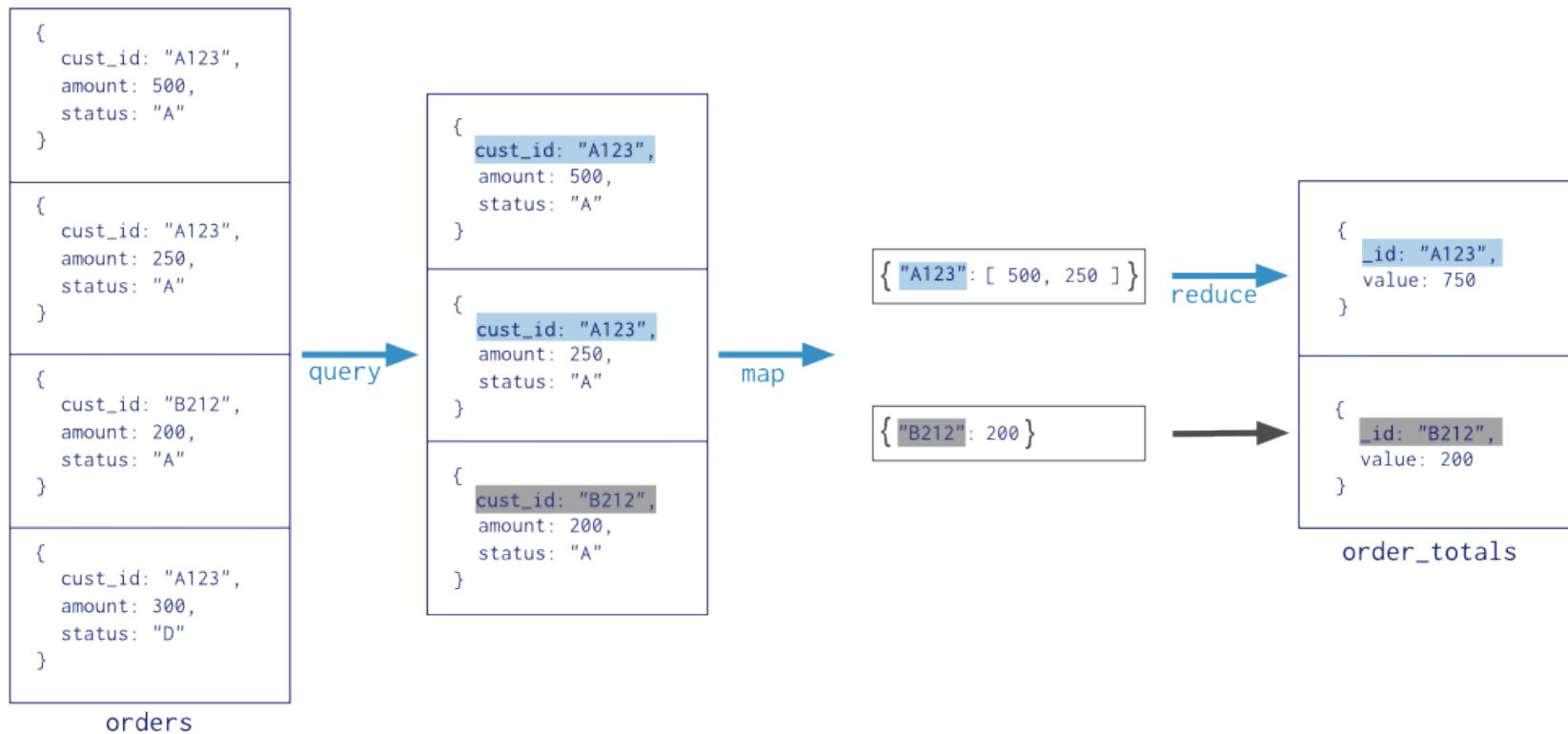
Aggregate

```
SELECT state, SUM(pop) AS totalPop  
FROM zipcodes  
GROUP BY state  
HAVING totalPop >= (10*1000*1000)
```

```
db.zipcodes.aggregate(  
{ $group: { _id : "$state",  
            totalPop: { $sum : "$pop" } } },  
{ $match: {totalPop: { $gte :  
10*1000*1000 } } } )
```

Map-Reduce

```
Collection  
db.orders.mapReduce(  
  map → function() { emit( this.cust_id, this.amount ); },  
  reduce → function(key, values) { return Array.sum( values ) },  
  {  
    query → { status: "A" },  
    output → {  
      out: "order_totals"  
    }  
  }  
)
```



Agregação - dados

```
> db.items.insert({title: 'Home', template: 'home'})  
> db.items.insert({title: 'What We Do', template: 'page'})  
> db.items.insert({title: 'Our Writing', template: 'page'})  
> db.items.insert({title: 'Who We Are', template: 'page'})  
> db.items.insert({title: 'Hire Us', template: 'page'})
```

Agregação

```
> var key = {template: true};  
> var initial = {count:0};  
> var reduce = function(obj, prev) {  
    prev.count += 1;  
};  
  
> db.items.group({key:key, initial:initial,  
reduce:reduce})  
[  
{"template" : "home", "count" : 1},  
 {"template" : "page", "count" : 4}  
]
```

Consulta - agrupamento por idade

```
db.alunos.group({  
    key: {  
        idade: true  
    },  
    initial: {  
        alunos: 0  
    },  
    reduce: function (aluno, grupo) {  
        grupo.alunos++;  
    }  
});
```

Resultado:

```
[{"idade": 9, "alunos": 2}, {"idade": 8, "alunos": 1}]
```

Listar usando Javascript

```
db.alunos.find().sort({  
    nome: 1  
}).forEach(function (aluno) {  
    print("Aluno: " + aluno.nome)  
});
```

```
db.accounts.find().forEach(function(doc) {  
    print(tojson(doc));  
});
```

Relacionamentos



Referenciando documento de outra coleção

- Sintaxe:
 - { \$ref : "collection_name", \$id : "reference_id" }
- Exemplo:

```
db.towns.update(  
  { _id : ObjectId("4d0ada87bb30773266f39fe5") },  
  { $set : { country: { $ref: "countries", $id: "us" } } }  
)
```

Relacionamento One to Many

- Duas estratégias
 1. Normalizado
 2. Embutido

One to Many - Normalizado

```
// insert post
> db.posts.insert({title:'Why Mongo Rocks'});
> var post = db.posts.findOne({title:'Why Mongo
Rocks'});
```



```
// insert comment
> db.comments.insert({
name : 'John',
body : 'Because...',
post_id : post._id
});
```



```
> var comment = db.comments.findOne({name: 'John'});
```

One to Many - Normalizado

```
SELECT * FROM comments WHERE post_id = #{post.id}
```

```
> db.comments.find({post_id: post._id}) {  
  "_id" : ObjectId("4bee1cc79e89db4e12bf78de"),  
  "name" : "John",  
  "body" : "Because...",  
  "post_id" : ObjectId("4bee1c519e89db4e12bf78dd")  
}
```

One to Many - Normalizado

```
SELECT * FROM posts WHERE id = #{comment.id}
```

```
> db.posts.find({_id: comment.post_id}) {  
  "_id" : ObjectId("4bee1c519e89db4e12bf78dd"),  
  "title" : "Why Mongo Rocks"  
}
```

One to Many - Embutido

```
// insert post AND comments
> db.posts.insert({
  title: 'Why Mongo Rocks',
  comments: [
    {name: 'John', body: 'Because...'},
    {name: 'Steve', body: 'Uh huh!'}
  ]
})
```

One to Many - Embutido

```
> var post = db.posts.find({title: 'Why Mongo  
Rocks'});  
  
> post  
{  
  "_id" : ObjectId("4bee21259e89db4e12bf78df"),  
  "title" : "Why Mongo Rocks",  
  "comments" : [  
    {"name": "John", "body": "Because..."},  
    {"name": "Steve", "body": "Uh huh!"}  
  ]  
}
```

One to Many - Embutido

```
> db.posts.find({'comments.name': 'John'})
```

```
> db.posts.find({  
  comments: {  
    $elemMatch: {name: 'John'}  
  }  
})
```

One to Many

- Embutido = junção prévia
- Embutir quando o documento sempre aparecer com pai.
- Limite máximo de tamanho de documento: 4MB

Many to Many

```
> db.sites.insert({domain: 'orderedlist.com'})  
> db.sites.insert({domain: 'railstips.org'})  
> db.sites.find()  
{  
  "_id" : ObjectId("4bee280f9e89db4e12bf78e2"),  
  "domain": "orderedlist.com"  
}  
{  
  "_id" : ObjectId("4bee283c9e89db4e12bf78e3"),  
  "domain": "railstips.org"  
}
```

Many to Many

```
> db.users.insert({  
  name: 'John',  
  authorizations: [  
    ObjectId('4bee280f9e89db4e12bf78e2'),  
    ObjectId('4bee283c9e89db4e12bf78e3')  
  ]  
})  
  
> db.users.insert({  
  name: 'Steve',  
  authorizations: [  
    ObjectId('4bee280f9e89db4e12bf78e2')  
  ]  
})
```

Many to Many

```
> var orderedlist =
db.sites.findOne({domain: 'orderedlist.com'})
> db.users.find({authorizations:orderedlist._id})
// john and steve

> var railstips =
db.sites.findOne({domain: 'railstips.org'})
> db.users.find({authorizations:railstips._id})
// john

> var john = db.users.findOne({name: 'John'})
> db.sites.find({_id: {$in: john.authorizations}})
// orderedlist.com and railstips.org
```

Map-Reduce



Map-Reduce

```
> db.items.insert({tags: ['dog', 'cat']})  
> db.items.insert({tags: ['dog']})  
> db.items.insert({tags: ['dog', 'mouse']})  
> db.items.insert({tags: ['dog', 'mouse', 'hippo']})  
> db.items.insert({tags: ['dog', 'mouse', 'hippo']})  
> db.items.insert({tags: ['dog', 'hippo']})
```

Map-Reduce

```
> var map = function() {
  this.tags.forEach(function(t) {
    emit(t, {count: 1});
  });
}
```

```
var reduce = function(key, values) {
  var count = 0;
  for(var i=0, len=values.length; i<len; i++) {
    count += values[i].count;
  }
  return {count: count};
}
```

Map-Reduce

```
> var result = db.items.mapReduce(map, reduce);
```

```
> result
{
  "ok" : 1,
  "timeMillis" : 86,
  "result" : "tmp.mr.mapreduce_1273861517_683",
  "counts" : {
    "input" : 6,
    "emit" : 13,
    "output" : 4
  }
}
```

Map-Reduce

```
> db[result.result].find()
```

```
{ "_id" : "cat", "value" : { "count" : 1 } }
{ "_id" : "dog", "value" : { "count" : 6 } }
{ "_id" : "hippo", "value" : { "count" : 3 } }
{ "_id" : "mouse", "value" : { "count" : 3 } }
```

Média de notas por matéria

```
mapReduce = db.alunos.mapReduce(function () {  
    this.materias.forEach(function (materia) {  
        materia.notas.forEach(function (nota) {  
            emit(materia.materia, nota);  
        })  
    })  
}, function (key, vals) {  
    var somaDasNotas = 0;  
    vals.forEach(function (nota) {  
        somaDasNotas += nota;  
    });  
    var media = somaDasNotas / vals.length;  
    return media;  
});
```

Resultado

```
{"result" : "tmp.mr.mapreduce_1266446901_6",
"timeMillis" : 74, "counts" : {"input" : 3,
"emit" : 18, "output" : 2},
"ok" : 1,}
```

```
db[mapReduce.result].find()
{ "_id" : "Inglês", "value" :
8.055555555555555 }
{ "_id" : "Matemática", "value" :
6.888888888888889 }
```

Geoposicionamiento



Geoposicionamento

- Simples.
- Apenas adicione um índice:
`db.lugares.ensureIndex({ localizacao: "2d" })`
- Lugares mais próximos

```
db.lugares.find({  
  localizacao: { $near : [  
    21.123456789, -20.123456789  
  ]}  
})
```

20 lugares mais próximos

```
db.lugares.find({  
    localizacao: { $near : [  
        21.123456789, -20.123456789  
    ]}  
}).limit(20)
```

Em uma área

```
db.lugares.find({
```

```
    localizacao: { $within: { $box: {
```

```
[
```

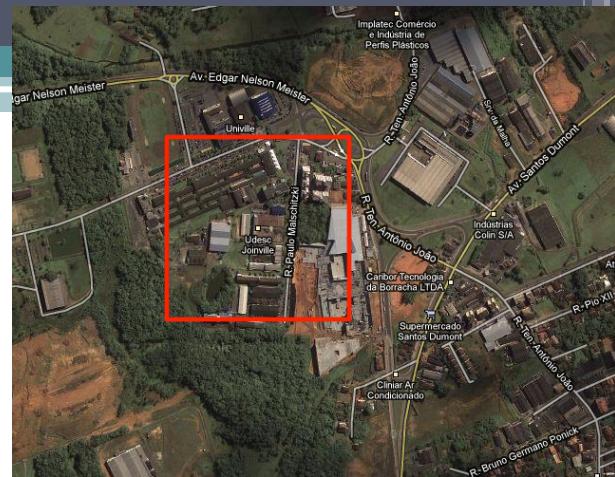
```
    [21.123456789, -20.123456789],
```

```
    [22.123456789, -21.123456789]
```

```
]
```

```
}}}
```

```
})
```



Referências



Referências



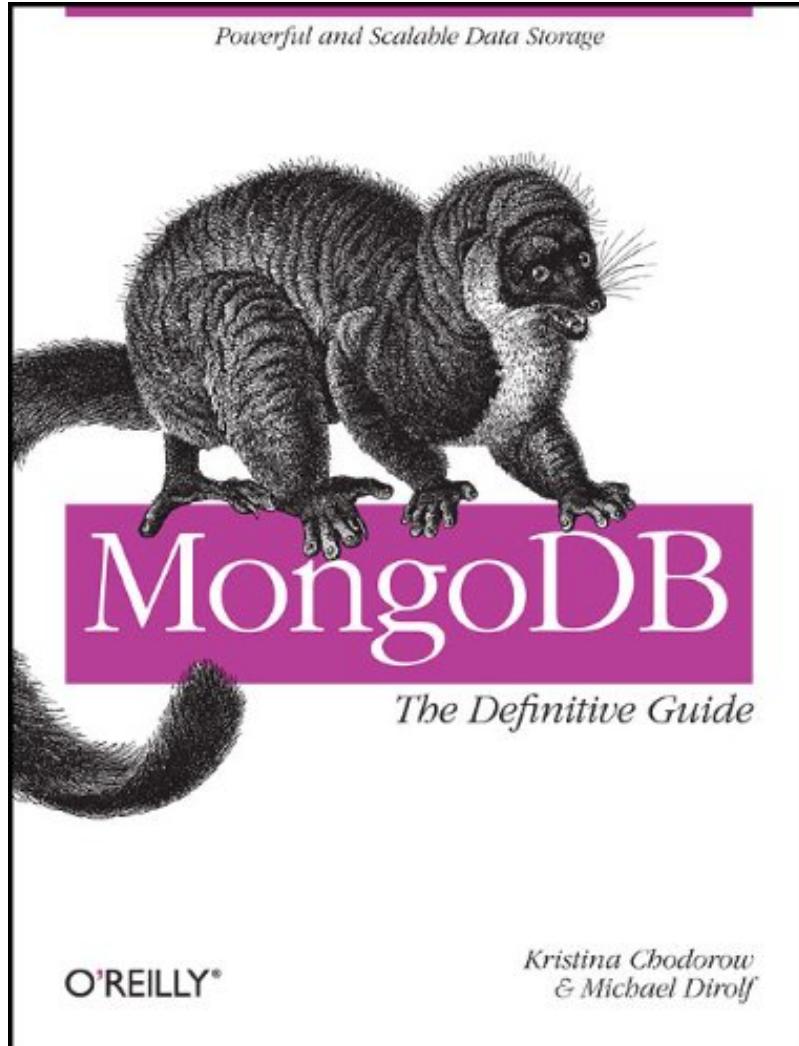
- MongoDB – Site oficial
 - <http://www.mongodb.com>
- MongoDB Manual
 - <http://docs.mongodb.org/manual/>
 - <http://docs.mongodb.org/manual/MongoDB-manual.pdf>
- Slides: Building your first app: an introduction to MongoDB. Norman Graham. Consulting Engineer, 1ogen.
- Slides: mongoDB. Júlio Monteiro (julio@monteiro.eti.br).

Referências



- Slides Why MongoDB Is Awesome
 - John Nunemaker - Ordered List
 - (john@orderedlist.com)
- Mkyong.com
 - <http://www.mkyong.com/tutorials/java-mongodb-tutorials/>
- The Little MongoDB Book by Karl Seguin
 - <http://openmymind.net/mongodb.pdf>

Referências



Obrigado!

Dúvidas, comentários, sugestões?



mongoDB

Regis Pires Magalhães
regismagalhaes@ufc.br

