

mongoDB

Profa. Andrêza Leite

MongoDB

- Modelo de documento
- Tipo de dado
- Características e Recursos
- Casos apropriados e inapropriados
- Atividade Prática

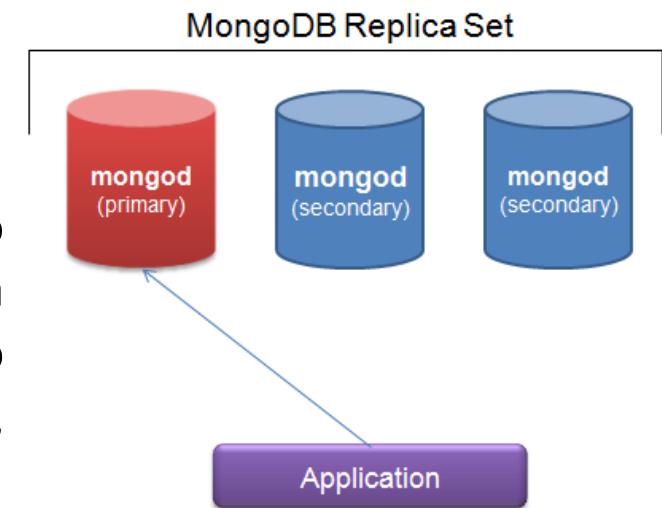
MongoDB

- Uma base de dados possui várias coleções;
- Uma coleção possui vários documentos;
- Cada documento pode ter qualquer campo. Não há esquema!!

| Termos/conceitos do SQL | Termos/conceitos do MongoDB |
|--|---|
| Database | Database |
| Tabela | Coleção |
| Linha | Documento ou documento BSON |
| Coluna | Campo |
| Index | Index |
| Table join | Doc aninhado (embedded) e vinculados |
| Chave primária —qualquer coluna única ou uma combinação de colunas | Chave primária — automaticamente definida como campo <code>_id</code> |

Características e Recursos

- Disponibilidade
 - Conjunto de réplicas Mestre-escravo
 - Redundância de dados, recuperação automática de falhas, ampliação da capacidade de leitura, manutenção do servidor sem tirar o app do ar, recuperação pós desastres



Características e Recursos

- Consistência
 - Réplicas db.runCommand({getlasterror:1,w:"majority"})
 - w="majority" em 3 nós a gravação replicará para pelo menos 2
 - Aumentar w aumenta a consistência mas perde desempenho.
 - rs.slaveOk() permite leituras a partir de escravos na conexão;
 - .setWriteConcern(ACKNOWLEDGED) garante gravações mestre+escravos, espera por acknowledgement

```
Mongo mongo = new Mongo("localhost:27017");
mongo.slaveOk();
```

Características e Recursos

- Transações
 - Não possui comandos commit ou rollback
 - Gravação bem sucedida ou falha
 - Transações atômicas(1 documento)
 - + de uma operação não são possíveis
 - WriteConcern(W2) =gravações em + de 1 nó

```
DBCollection shopping = database.getCollection("shopping");
shopping.setWriteConcern(REPLICAS_SAFE);
```

ACKNOWLEDGED

Características e Recursos

- Escalabilidade
 - Horizontal para leituras: + escravos
 - Horizontal para gravações: fragmentação
 - Dados divididos por campo entre os nodos graváveis
 - Ex: fragmentar pela localização do cliente. Dados em nodos mais próximos.

Casos apropriados

<esquemas flexíveis>

- Registros de eventos (log)
- Sistemas de Gerenciamento de Conteúdo, blog
- Análises web ou em tempo real (analytics)
- Apps de Comércio eletrônico

Casos Inapropriados

- Transações complexas que abranjam diferentes operações
 - Operações atômicas em múltiplos documentos.
 - RavenDB
- Consultas em estruturas agregadas variáveis
 - Esquemas flexíveis: consultas mudarão
 - Necessitaria gravar no nível mais baixo de granularidade: normalizar os dados

Exemplo de um documento

```
{  
  "_id" : ObjectId("541f30d992a2ee25fedaa652"),  
  "nome" : "Andreza",  
  "twitter" : "andreza_paju"  
}  


Chave



Valor


```

Exemplo de um documento

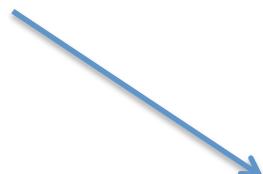
```
{  
  "_id" : ObjectId("541f30d992a2ee25fedaa652"),  
  "nome" : "Andreza",  
  "twitter" : "andreza_paju"  
  "linguagens" : [  
    "Python",  
    "C",  
    "JavaScript",  
    "C++"  
  ]  
}
```



Uma lista ou array

Exemplo de um documento

```
{  
  "_id" : ObjectId("541f30d992a2ee25fedaa652"),  
  "nome" : "Andreza",  
  "redes_sociais" : {  
    "Twitter" : "andreza_paju",  
    "Facebook" : "andreza.leite1",  
    "LinkedIn" : "andrezaleite",  
  }  
}
```

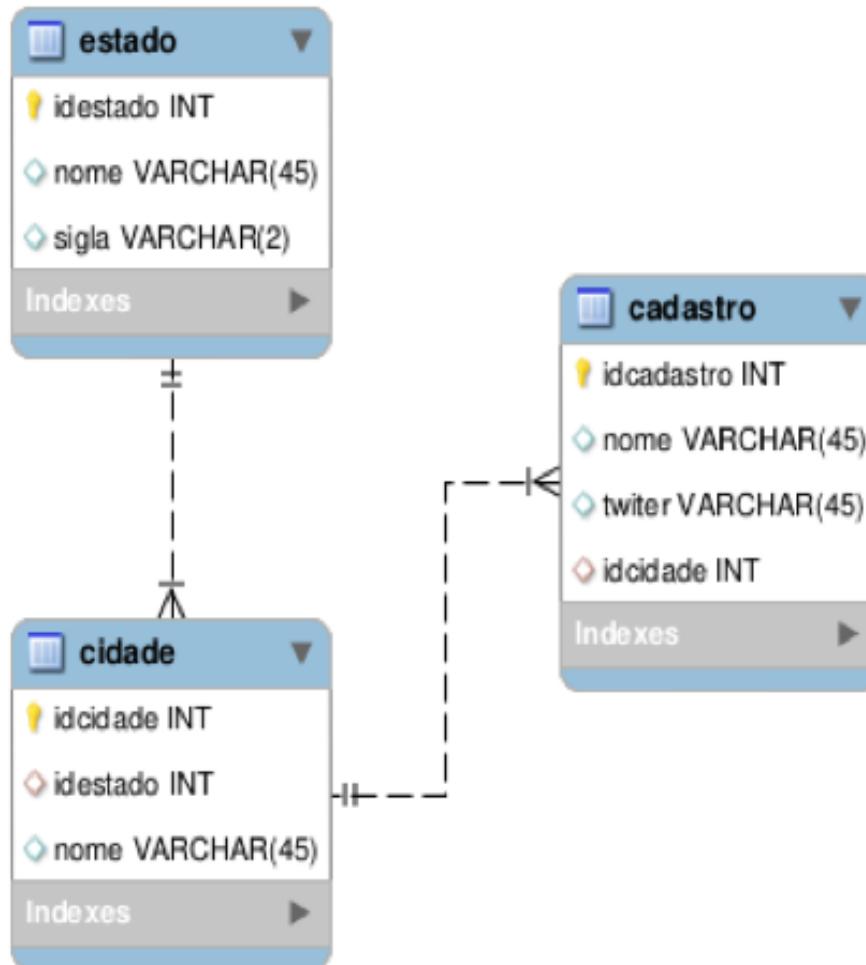


**Uma lista de valores ou
Um documento aninhado**

MongoDB X Relacional

- Dados:
 - Nome: Andreza Leite
 - Twitter: andreza_paju
 - Cidade: Recife
 - Estado: PE

Relacional



```
mysql> select * from cadastro;
+----+-----+-----+-----+
| id | nome      | twitter | cidade |
+----+-----+-----+-----+
| 1  | Christiano | dump   | 1       |
+----+-----+-----+-----+
1 row in set (0.00 sec)
```

Informação normalizada,
Relaciona com a tabela de cidade que
relaciona com a tabela de estado

MongoDB

```
{  
  "_id" : ObjectId("541f64d092a2ee25fedaa654"),  
  "nome" : "Andreza",  
  "twitter" : "andreza",  
  "cidade" : "Recife",  
  "estado" : "PE"  
}
```

- Coleções não relacionadas - dados desnormalizados
- Pode ocorrer inconsistências – controle feito via código;

Outro exemplo de schema design:

Coleção: livraria

Coleção: alunos

Os alunos podem retirar um ou mais livros da livraria.

Como fazer esse controle com MongoDB?

Outro exemplo de schema design:

- Coleção alunos

```
> db.alunos.find()  
{ "_id" : 1, "nome" : "Pedrinho", "sala" : "200" }  
{ "_id" : 2, "nome" : "Zezinho", "sala" : "404" }  
{ "_id" : 3, "nome" : "Luizinho", "sala" : "500" }
```

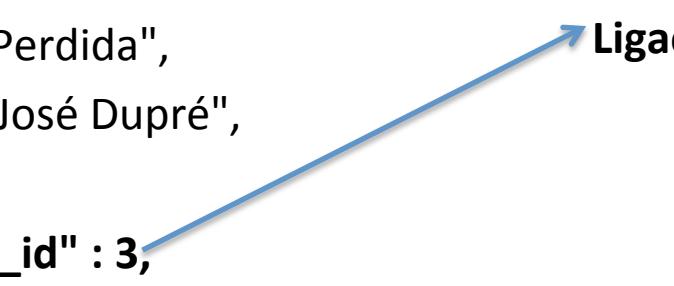
- Coleção livros

```
> db.livros.find().pretty()  
{ "_id" : 1, "titulo" : "A Ilha Perdida", "autor" : "Maria José Dupré" }  
{ "_id" : 2, "titulo" : "Éramos Seis", "autor" : "Maria José Dupré" }  
{ "_id" : 3, "titulo" : "Sozinha no Mundo", "autor" : "Marcos Rey" }
```

Cenários

- Luizinho quer alugar o livro “A Ilha Perdida”
 - Criar uma chave “aluguel” na coleção de livros

```
{  
  "_id" : 1,  
  "titulo" : "A Ilha Perdida",  
  "autor" : "Maria José Dupré",  
  "aluguel" : {  
    "aluno_id" : 3,  
    "data" : ISODate("2014-09-21T00:00:00Z")  
  }  
}
```



Ligaçao com o id do aluno, Luizinho

Atualização

```
>db.livros.update({'_id': 1}, {$set:{'aluguel':{'aluno_id':  
3, 'data':ISODate('2014-09-21')}}})
```

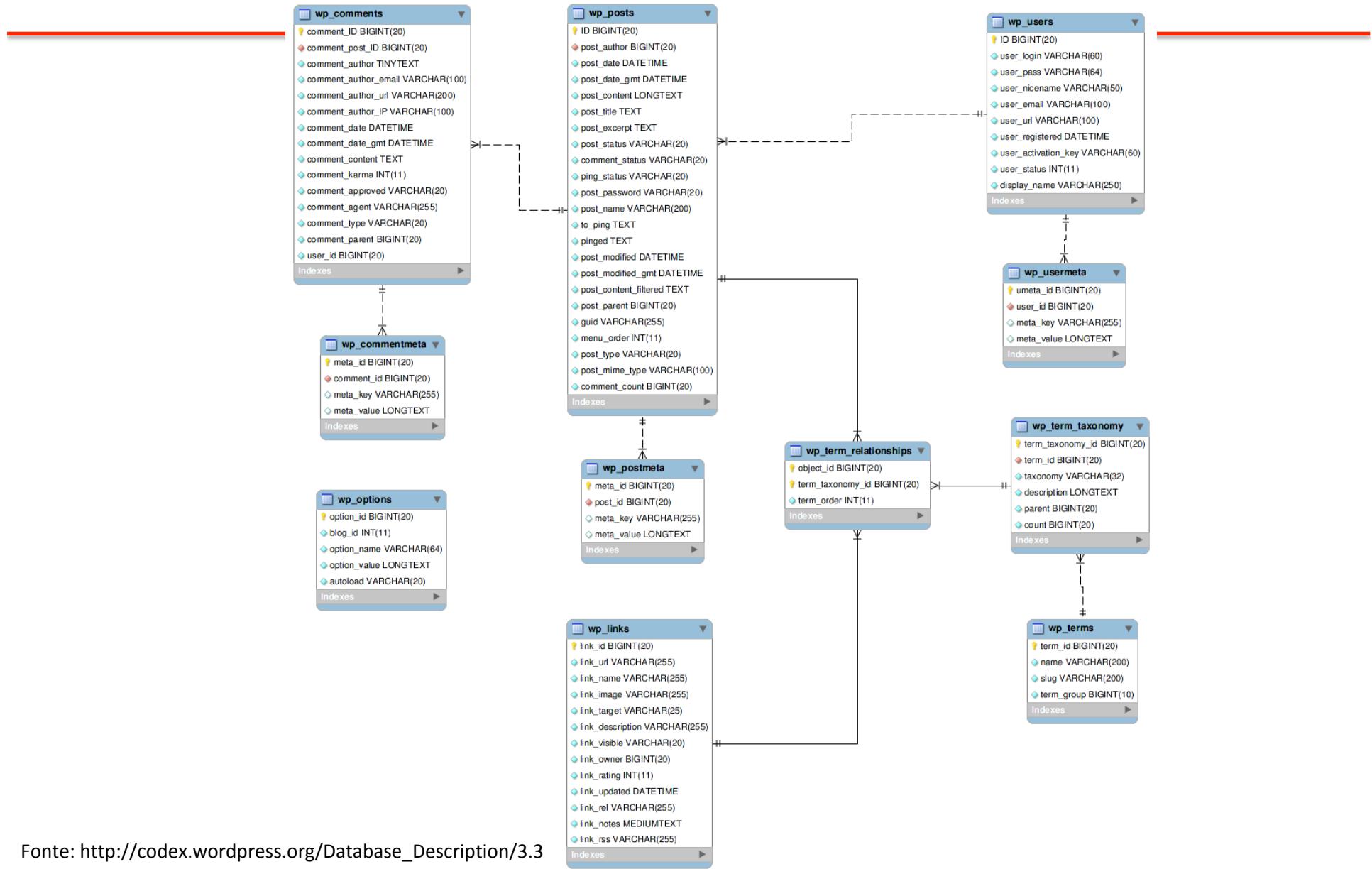
Ver todos os livros alugados

```
> db.livros.find({'aluguel':{$exists:true}}).pretty()
{'aluguel':
  "_id" : 1,
  "titulo" : "A Ilha Perdida",
  "autor" : "Maria José Dupré",
  "aluguel" : {
    "aluno_id" : 3,
    "data" : ISODate("2014-09-21T00:00:00Z")
  }
}
```

Modelagem::Schema Design

- O arquiteto precisa entender como as informações serão **inseridas e consultadas** no sistema;
- O sucesso está em extrair o maior número de informações em uma única consulta

Exemplo Blog



Fonte: http://codex.wordpress.org/Database_Description/3.3

Posts e Comentários



- Um post = Vários comentários
- Tabela de Post possui a PK de cada post
- Tabela de Comentário possui FK ligando a cada post
- A cada acesso no Post, é necessário fazer uma consulta na tabela de Comentários para buscar todos

MongoDB

```
{  
    "_id" : ObjectId("541f6a9092a2ee25fedaa655"),  
    "titulo" : "Aqui é o título",  
    "tags" : [  
        "teste",  
        "exemplo",  
        "mongodb"  
    ],  
    "conteudo" : "Aqui vem o Lorem Ipsum básico",  
    "comentarios" : [  
        {  
            "usuario" : "Usuario Troll",  
            "email" : "troll@trololand.com",  
            "comentario" : "Vim aqui só trollar"  
        },  
        {  
            "usuario" : "Usuario Sério"  
            "email" : "serio@serioland.com",  
            "comentario" : "Parabéns pelo post"  
        }  
    ]  
}
```

Deixar posts e comentários na mesma coleção

Os comentários ficam embarcados no mesmo documento que o post

Uma única query retorna o post e todos seus comentários



Prática

Iniciando

```
$ mongo
```

```
MongoDB shell version: 2.6.7
```

```
connecting to: test
```

```
> show dbs;
```

```
...
```

```
> use ligado;
```

```
switched to db ligado
```

```
> show dbs;
```

```
ligado
```

```
> db.albums.insert({});
```

```
WriteResult({ "nInserted" : 1 })
```

Coleções

```
> show collections;
```

```
> db.albuns.insert({});
```

```
WriteResult({ "nInserted" : 1 })
```

```
> db.albuns.find({});
```

```
{ "_id" : ObjectId("54c023bf09ad726ed094e7db")}
```

Inserindo documentos

```
> use ligado
```

```
> db.albuns.insert(  
  {"nome" : "Master of Puppets",  
   "dataLancamento" : new Date(1986, 2, 3),  
   "duracao" : 3286})
```

```
> db.albuns.insert(  
  {"nome" : "...And Justice for All",  
   "dataLancamento" : new Date(1988, 7, 25),  
   "duracao" : 3929})
```

```
> db.albuns.insert(  
  {"nome" : "Among the Living",  
   "produtor" : "Eddie Kramer"})
```

Buscando documentos

```
>db.albuns.find({"nome" : "Master of Puppets"});
```

Equivalente a:

```
SELECT *  
FROM albuns a  
WHERE a.nome = "Master of Puppets"
```

```
>db.albuns.findOne({"nome" : "Master of Puppets"})
```

```
null
```

- db.albuns.find({"nome":/m/})
- db.albuns.find({"nome":.*[Mm].*/})
- db.albuns.find({"nome":.*M|m.*/})
- db.albuns.find({"nome":^m\$/})
 - Começa e termina com m

← Equivalente a ao
"Like" em SQL

Buscando documentos

```
> db.albuns.find({"nome" : "Master of Puppets"});  
[{"_id" : ObjectId("590e0e6d657ba1f53f44581d"), "nome" : "Master of Puppets", "dataLancamento" : ISODate("1986-03-03T02:00:00Z"), "duracao" : 3286, "artista_id" : ObjectId("590e1e02657ba1f53f445820") }]  
>  
>  
>  
>  
> db.albuns.findOne({"nome" : "Master of Puppets"});  
{  
    "_id" : ObjectId("590e0e6d657ba1f53f44581d"),  
    "nome" : "Master of Puppets",  
    "dataLancamento" : ISODate("1986-03-03T02:00:00Z"),  
    "duracao" : 3286,  
    "artista_id" : ObjectId("590e1e02657ba1f53f445820")  
}  
> █
```

```
[> db.albuns.find({"nome" : "Master"});  
[> db.albuns.findOne({"nome" : "Master"});  
null  
> █
```

Buscando documentos

| Nome | Descrição |
|-------|--|
| \$gt | Corresponde a valores que são maiores que o valor específico na query. |
| \$gte | Corresponde a valores que são maiores ou iguais ao valor específico na query. |
| \$in | Corresponde a quaisquer valores que existem em um array específico em uma query. |
| \$lt | Corresponde a valores que são menores que o valor específico na query. |
| \$lte | Corresponde a valores que são menores ou iguais que o valor específico na query. |
| \$ne | Corresponde a todos os valores que não são iguais ao valor específico na query. |
| \$nin | Corresponde a valores que não existem em um array específico da query. |

Buscando documentos

Sintaxe : {"nomeDoCampo" : {"operador" : " valor "}}

```
> db.albuns.find({"duracao" : {"$lt" : 1800}})
```

Equivalente a:

```
SELECT *
```

```
FROM albuns a
```

```
WHERE a.duracao < 1800
```



menor que 1800

Buscando documentos

| Nome | Descrição |
|-------|--|
| \$and | Junta <i>query clauses</i> com uma lógica E retorna todos os documentos que combinam com ambas condições. |
| \$nor | Junta <i>query clauses</i> com uma lógica NEM retorna todos os documentos que falham em combinar ambas as condições. |
| \$not | Inverte o efeito de uma <i>query expression</i> e retorna os documentos que não combinam com a condição. |
| \$or | Junta <i>query clauses</i> com uma lógica OU retorna todos os documentos que combinam com ambas condições. |

Buscando documentos::Exercício

- Montar a query que retorna todos os álbuns lançados em 1986 usando intervalos (dois filtros) e operadores lógicos
- Sintaxe:
 - {operador : [expressão 1, expressão 2, expressão n]}.
- Filtros:
 - *data de lançamento maior ou igual que 01/01/1986*
 - {"dataLancamento" : {\$gte : new Date(1986, 0, 1)}}
 - *data de lançamento menor que 01/01/1987*
 - {"dataLancamento" : {\$lt : new Date(1987, 0, 1)}}

Buscando documentos::Exercício

```
> db.albuns.find(  
  {$and : [{"dataLancamento" : {$gte : new Date(1986, 0, 1)}},  
            {"dataLancamento" : {$lt : new Date(1987, 0, 1)}}]}  
)
```

Equivalente a:

```
SELECT *  
FROM albuns a  
WHERE a.dataLancamento >= '1986-01-01 00:00:00'  
AND a.dataLancamento < '1987-01-01 00:00:00'
```

Buscando documentos::Exercício

- Podemos usar uma variação da sintaxe da seguinte maneira:
{"nomeDoCampo" : {comparador1 : "valor1", comparador2 : "valor2"} }.
- A consulta seria:
> db.albuns.find({"dataLancamento" : {"\$gte" : new Date(1986, 1, 1), "\$lt" : new Date(1987, 1, 1)}})

Removendo Documentos

- Primeiro consulte:
> db.albuns.find({"nome": "...And Justice for All"})
- Se ok, mude para a função *remove*
> db.albuns.remove({"nome": "...And Justice for All"})

PS: CUIDADO PARA NÃO REMOVER TODOS OS DADOS
> db.albuns.remove({})

Alterando documento

```
>db.albuns.update({"nome" : "Among the Living"},  
 {"duracao" : 3013})  
> db.albuns.find({})  
...{"_id" : ObjectId("54c828b5342dda43e93bee1e"),  
 "duracao" : 3013 } ...
```

Sintaxe:

```
db.albuns.update({"nome" : "Among the Living"},  
 { $set : {"duracao" : 3013}}).
```

Alterando documento

```
> db.albums.update(  
  {"_id" : ObjectId("590e0e7f657ba1f53f44581f")},  
  {$set : {"nome" : "Among the Living",  
           "produtor" : "Eddie Kramer"}})
```



query

Excluindo BD e Collections

```
>use ligado
```

```
switched to db ligado
```

```
>db.dropDatabase()
```

```
>{"dropped" : "ligado", "ok" : 1 }
```

```
>db.albuns.drop({})
```

Criando relacionamentos

<ObjectID>

```
> db.artistas.insert([ {"nome" : "Metallica"},  
                      {"nome" : "Megadeath"},  
                      {"nome" : "Slayer"},  
                      {"nome" : "Anthrax"} ])
```

```
> db.artistas.find({})
```

Criando relacionamentos

<definindo id>

```
> db.artistas.insert([ {"nome" : "Metallica",  
  "_id":1},  
                      {"nome" : "Megadeath", "_id":2},  
                      {"nome" : "Slayer", "_id":3},  
                      {"nome" : "Anthrax", "_id":4} ])
```

```
> db.artistas.find({})
```

Criando relacionamentos

```
> db.albuns.update( {"nome" : "Master of Puppets"},  
{$set : {"artista_id" : ObjectId("580b9533acd84e0d20e2a85c")}}  
);
```

Ou

```
>db.albuns.update( {"nome" : "Master of Puppets"},  
{$set : {"artista_id" : 1}} );
```

... Repetir para cada album

Consultando

```
> var artista = db.artistas.findOne({"nome" : "Metallica"});
```

```
> artista
```

```
> var albuns = db.albuns.find({"artista_id" : artista._id})
```

```
> albuns.forEach( function(albuns) {  
  print(albuns["nome"]);  
});
```

Aninhando documentos

```
> db.albuns.insert(  
  {"nome" : "Somewhere Far Beyond",  
   "dataLancamento" : new Date(1992, 5, 30),  
   "duracao" : 3328,  
   "artista" : {"nome" : "Blind Guardian"}  
 }  
);
```

Mongolimport

- JSON, CSV ou TSV
- A primeira linha (cabeçalho) será usada como chave dos campos

```
$ cat celulares.txt
marca,modelo,valor
Samsung,SIII,1500.00
Apple,iPhone 5,2500.00
Geeksphone,Keon,300.00
```

```
$ mongoimport -d produtos -c celulares --
type csv --headerline --file celulares.txt
```

Mongolimport

- Exercício Banco “Movies”

```
$ mongoimport -d movies -c movie --type csv  
--headerline --file movie_metadata.csv
```