Universidade do Minho
Mestrado Integrado em Engenharia Informática
Licenciatura em Ciências da Computação

Unidade Curricular de Bases de Dados

Ano Letivo de 2017/2018

Bilheteira de Cinema

NoSQL - MongoDB

Diogo Emanuel da Silva Nogueira (a78957)

Diogo Francisco Araújo Leite da Costa (a78485)

Mariana Lino Lopes Costa (a78824)

Sarah Tifany da Silva (a76867)

Janeiro, 2018

BD

Data de Receção	
Responsável	
Avaliação	
Observações	

Bilheteira de Cinema

NoSQL - MongoDB

Diogo Emanuel da Silva Nogueira (a78957)

Diogo Francisco Araújo Leite da Costa (a78485)

Mariana Lino Lopes Costa (a78824)

Sarah Tifany da Silva (a76867)

Janeiro, 2018

Resumo

Esta fase final do projeto consiste em desenvolver uma BD Não-Relacional com base na BD Relacional anteriormente desenvolvida. O que se espera desta nova BD é que a mesma possa dar suporte a um Sistema que permita o correto funcionamento de uma Bilheteira de Cinema, acabando assim por cumprir todos os requisitos inicialmente estabelecidos, mas agora de uma forma mais elástica aos futuros requisitos, com um esquema mais flexível e com dados semi-estruturados e independentes.

Área de Aplicação: Desenho e Administração de BD para uma Bilheteira de um Cinema.
Palavras-Chave: Cinema, Bilheteira, Bilhetes, Entretenimento, Simplicidade, Organização, Rapidez, Descomplicação, Fluidez, Assistência, BD, Entidades, Atributos, NoSQL, MongoDB.

Índice

Índice de Figuras

1. Introdução

Nesta segunda fase do projeto iremos focar num sistema de Base de Dados Não-Relacional como suporte para o sistema de armazenamento de toda a informação referente à Bilheteira dum Cinema. Neste caso, dos variadíssimos sistemas *NoSQL* que existem em mercado foi utilizado o *MongoDB*, que se apoia no armazenamento de documentos em forma de *JSON/BSON*. Dessa forma a Base de Dados não tem de ter uma estrutura pré-definida (como as tabelas/relações em *MySQL*), sendo a estrutura definida, alterada e moldada à medida que se junta documentos às coleções da Base de Dados do *MongoDB*. Neste trabalho iremos destacar todas as transições e modificações que foram necessárias implementar por forma a conseguir obter o mesmo nível de requisitos e funcionalidades pedidas pelo empregador na primeira parte do projeto/relatório, na qual incidia a utilização de MySQL, ou seja, um sistema de Base de Dados Relacional.

O relatório está focado nestas áreas sendo que se divide em numa primeira parte na explicação ampla de como foi o pensamento para remodelar a estrutura Relacional para se adaptar às bases de dados, coleções e documentos que são o suporte de dados que o *MongoDB* assenta e, por fim, terá uma análise aos processos e pensamentos da transformação supracitada neste relatório bem como no projeto em geral.

2. Estrutura do modelo Não-Relacional por forma de documentos

2.1. Conversão do sistema Relacional para Não-Relacional por documentos JSON

O nosso projeto de Base de Dados tinha como base uma gestão Relacional, baseando-se no modelo proposto por E. F. Codd em 1970, que consiste na ideia de a informação (*data*) ser armazenada em tabelas (relações) com as suas linhas e colunas, sendo que cada coluna corresponde a um atributo e cada linha uma tupla, ou seja uma instância pertencente àquela tabela com os atributos respetivos. Para existir uma conversão correta baseou-se nos manuais e documentos que *MongoDB* disponibiliza para ajudar no processo.

Segundo os documentos, a conversão baseia-se fundamentalmente em que uma tabela/relação torna-se uma coleção, uma tupla/linha torna-se um documento e as colunas/atributos são campos (*fields*) desses mesmos documentos. Dessa forma, a nossa estrutura passou a ter três coleções, sendo Bilhete, Filme e Cliente as selecionadas entre as tabelas anteriormente existentes. As tabelas Género e InfoBilhete foram retiradas, dado que se pode usar uma das vantagens de *MongoDB* que deixa colocar *arrays* e/ou documentos embutidos noutros documentos. Daí a InfoBilhete pode estar introduzida no próprio Bilhete e os géneros do Filme foram introduzidos como um "*array* de *strings"* no documento do mesmo Filme.

Como visto nas imagens seguintes temos a comparação e migração das estruturas relacionais para não-relacionais:

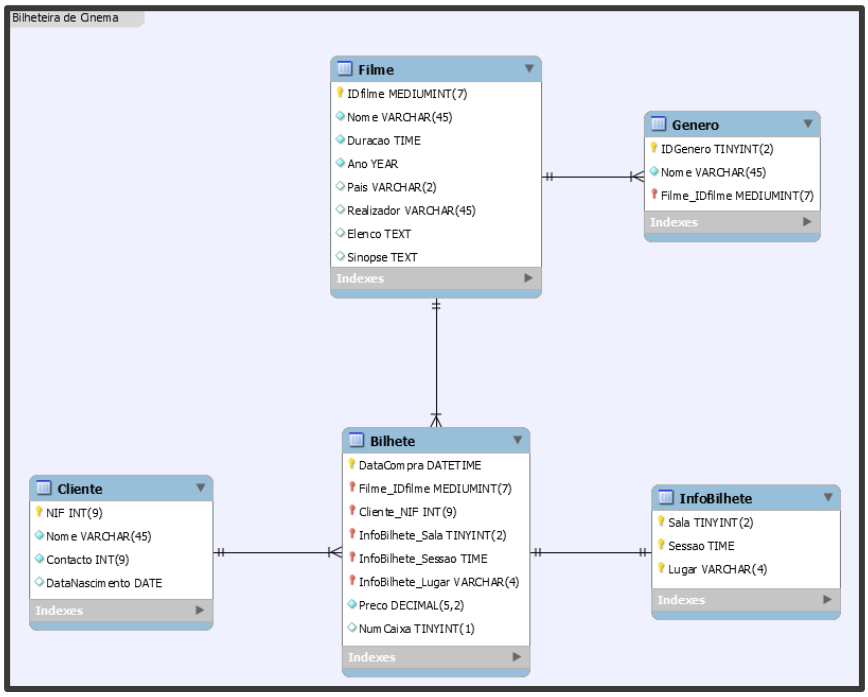

Figura 1: Modelo Lógico Relacional da Bilheteira de Cinema

Cinema.Bilhete


```
_id: ObjectId("5a5e425817723b32906b864a")
Filme: "The Avengers"
NIF: 123456789
Sala: 7
Sessão: 2017-11-23 00:40:00.000
Lugar: "F15"
Preco: 6.6
DataCompra: 2017-11-23 17:15:00.000
InfoCaixa: Object
    Num: 3
    Funcionario: "Joana Sousa"
```


Cinema.cliente


```
_id: 253356369
Nome: "Diogo Nogueira"
Contacto: 913392656
DataDeNascimento: 1997-12-13 00:00:00.000
```



```
_id: 848228
Titulo: "The Avengers"
Duracao: "02:23"
DataEstreia: 2012-01-01 00:00:00.000
Pais: "Estados Unidos"
Realizador: "Joss Whedon"
Elenco: Array
    0: "Robert Downey Jr."
    1: "Chris Evans"
    2: "Scarlett Johansson"
Sinopse: "Superheróis a lutar aliens"
Generos: Array
    0: "Ação"
    1: "Familiar"
```


Cinema.Filme

Figura 2: Estrutura Não-Relacional da Bilheteira de Cinema

Para a coleção Filme fizemos a transição de todos os atributos/colunas para os campos de cada documento introduzido, sendo que mantivemos o *id* como o nº identificativo daquele Filme na maior Base de Dados de entretenimento (IMDB), para o *MongoDB* fazer o índice próprio usando o nosso *id* personalizado. De seguida mantivemos o Título, Duração, Data de Estreia, País, Realizador e Sinopse com os domínios de dados já usados anteriormente. A única diferença, que usou o estilo próprio de documentos JSON, foram as introduções do elenco e género como *arrays* da própria entrada do Filme/documento, em vez de atributos multivalor (tabelas separadas).

Para a coleção Cliente foram migrados todos os mesmos atributos para os campos de documento, sendo que o documento ficou com os campos: *id* (personalizado dado que continua a ser o NIF "Número de Identificação Fiscal"), Nome, Contacto e Data de Nascimento. Nestes campos JSON manteve-se também o domínio de dados que cada um tinha, para melhor conservação de espaço e controlo.

Para a coleção Bilhete foram migrados os atributos Filme, NIF do Cliente, Sala, Sessão, Lugar, Preço, Data da Compra e agora colocamos a Informação referente à caixa como um documento embutido, que contém o número de caixa e o funcionário que atendeu o cliente em causa. Assim toda a informação do Bilhete fica presente no mesmo, sendo possível alterar a estrutura Não-Relacional, colocando um documento dentro de outro, para melhor gestão e procura sobre a informação que possamos querer saber dos funcionários, como qual foi o funcionário do mês ou qual a caixa que é menos utilizada.

Usando os recursos disponíveis desta organização Não-Relacional é possível moldar melhor a informação, não só para um incremento exponencial de dados, mas também responder com maior rapidez a certas questões/requisitos que o empregador nos dá. Em última instância torna-se fundamental ter este nível de flexibilidade porque não se sabe caso precisemos de mudar a estrutura de dados e com o *MongoDB* isso torna-se algo trivial e acessível.

2.2. Migração e povoamento dos dados relacionais para os documentos *JSON* do Sistema *MongoDB*

Para procedermos à migração e povoamento dos dados relacionais para os vários documentos *JSON* do nosso Sistema de Base de Dados Não-Relacional, começamos por criar a nossa BD no Sistema *MongoDB* que dará suporte às várias coleções de documentos futuros. De seguida, geramos todas essas coleções necessárias em concordância com o nosso modelo Não-Relacional definido sendo que, por fim, e indo de acordo com o povoamento do modelo antigo, produzimos os vários documentos que dizem respeito a cada uma destas coleções.

Para demonstrar todo o povoamento no sistema vamos anexar os vários excertos de código criados para o efeito. Para tal, apenas iremos exibir, por cada coleção, a inserção de dois documentos de modo a não tornar toda esta amostra demasiado extensa. Para o suporte destes excertos vamos também anexar imagens que dirão respeito à representação destas coleções e dos seus documentos no *software MongoDB Compass* que nos serviu de aliado visual para melhor compreensão e manipulação da nossa Base de Dados *NoSQL*.

- **Criação da Base de Dados:**

```
use Cinema;
```

- **Criação de todas as Coleções:**

```
db.createCollection("Bilhete");
db.createCollection("cliente");
db.createCollection("Filme");
```

- **Povoamento da Coleção Bilhete:**

```
db.Bilhete.insertMany( [
    {
    "Filme" : "The Avengers",
    "NIF" : NumberInt(123456789),
    "Sala" : NumberInt(7),
    "Sessao" : new Date("2017-11-23 00:40:00"),
    "Lugar" : "F15",
    "Preco" : 6.6,
    "DataCompra" : new Date("2017-11-23 17:15:00"),
    "InfoCaixa" : {
```

```
            "Num" : NumberInt(3),
            "Funcionario" : "Joana Sousa"}
        },
        {
        "Filme" : "Beauty and the Beast",
        "NIF" : NumberInt(321654987),
        "Sala" : NumberInt(1),
        "Sessao" : new Date("2017-11-24 21:40:00"),
        "Lugar" : "G14",
        "Preco" : 10.6,
        "DataCompra" : new Date("2017-11-24 17:15:00"),
        "InfoCaixa" : {
            "Num" : null,
            "Funcionario" : null}
        },
    (…)
    ] );
```

Figura 3: Quatro dos seis documentos da coleção Bilhete representados no Software MongoDB Compass

- **Povoamento da Coleção Cliente:**

```
db.cliente.insertMany( [
    {
    "_id": 253356369,
    "Nome" : "Diogo Nogueira",
    "Contacto" : NumberInt(913392656),
    "DataDeNascimento" : new Date("1997-12-13")
    },
    {
    "_id": 249929589,
    "Nome" : "Diogo Araújo",
    "Contacto" : NumberInt(914215212),
    "DataDeNascimento" : new Date("1997-10-08")
    },
    {
    "_id": 123456789,
    "Nome" : "Tifany Silva",
    "Contacto" : NumberInt(910040099),
    "DataDeNascimento" : new Date("1994-01-25")
    },
(…)
] );
```

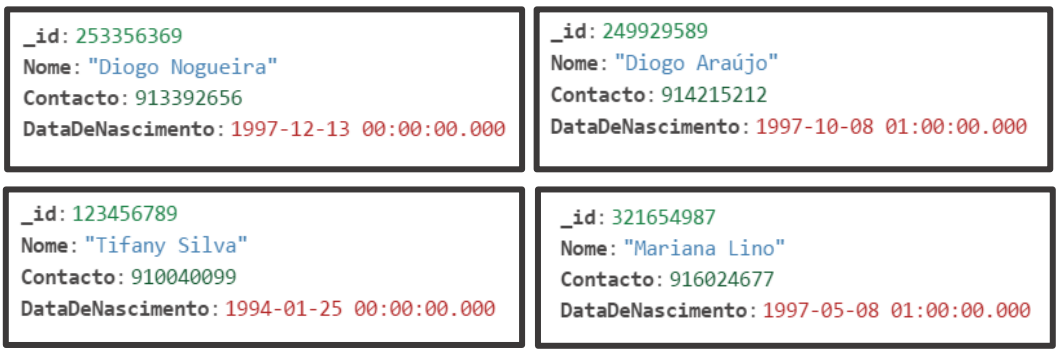

Figura 4: Quatro dos cinco documentos da coleção Cliente representados no Software MongoDB Compass

- **Povoamento da Coleção Filme:**

```
db.Filme.insertMany( [
    {
    "_id": 848228,
    "Titulo" : "The Avengers",
    "Duracao" : "02:23",
    "DataEstreia" : new Date("2012-01-01"),
    "Pais" : "Estados Unidos",
    "Realizador" : "Joss Whedon",
    "Elenco" : ["Robert Downey Jr.", "Chris Evans", "Scarlett Johansson"],
    "Sinopse" : "Superheróis a lutar aliens",
    "Generos" : ["Ação", "Familiar"]
    },
   {
    "_id": 2771200,
    "Titulo" : "Beauty and the Beast",
    "Duracao" : "02:09",
    "DataEstreia" : new Date("2017-01-01"),
    "Pais" : "Estados Unidos",
    "Realizador" : "Bill Condon",
    "Elenco" : ["Emma Watson", "Dan Stevens","Luke Evans"],
    "Sinopse" : "Uma adaptação do livro e Filme Bela e o Monstro",
    "Generos" : ["Familiar", "Musical", "Romance"]
    },
(…)
] );
```

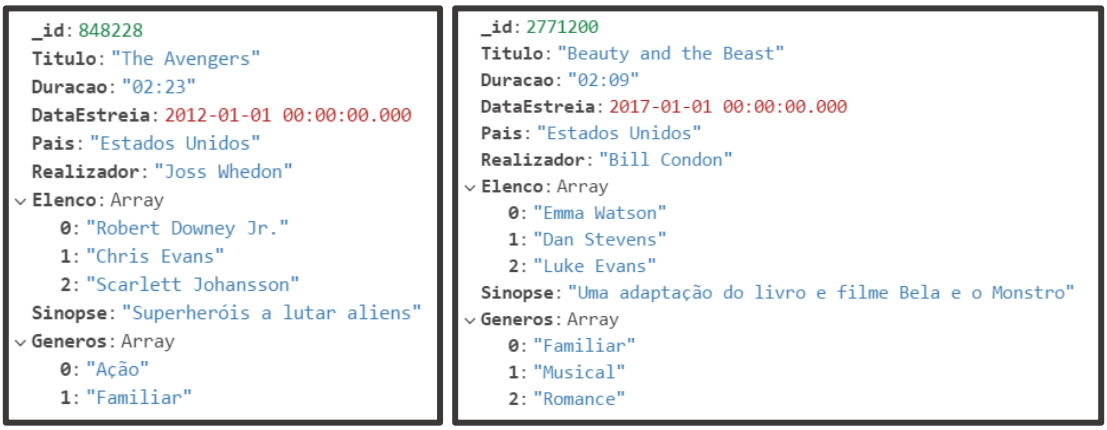

Figura 5: Dois dos cinco documentos da coleção Filme representados no Software MongoDB Compass

2.3. *Queries* equivalentes às utilizadas no meio Relacional (SQL) para *MongoDB*

- **Saber a receita total de um certo Filme como, por exemplo, *Beauty and the Beast*:**

```
db.Bilhete.aggregate(
    [
        // Fase do Match
        {
            $match: {Filme: "Beauty and the Beast"}
        },
        // Fase do Group e Sum
        {
            $group: {
            _id: "$Filme",
            ReceitaTotal: {$sum: "$Preco"}
            }
        },
    ]
);
```

- **Listar os Nomes dos Clientes e respetivos gastos totais nas compras do Bilhete, por ordem decrescente de preços totais:**

```
db.Bilhete.aggregate(
    [
        // Fase de Outer Join com Cliente
        {
            $lookup: {
                from: "cliente",
                localField: "NIF",
                foreignField: "_id",
                as: "ClienteInfo"
            }
        },
        // Transformar o array ClienteInfo em apenas um field documento
        {
            $unwind: {path : "$ClienteInfo"}
        },
        // Agrupar por NIF, somando o gasto total e mostrando o nome também
        {
            $group: {
             _id: "$NIF",
```

```
            Nome: {$first: "$ClienteInfo.Nome"},
            GastoTotal: {$sum: "$Preco"}
           }
       },
       // Colocando por ordem descendente do GastoTotal
       {
           $sort: {GastoTotal: -1}
       }
   ]
);
```

Para estas duas foi utilizado a *framework* de agregação de dados do *MongoDB*. Esta *framework* foi utilizado dado que é muito mais estável e tem uma interface mais coesiva que a *framework* mais antiga *(Map-Reduce)*. Dessa forma a agregação funciona através dum *pipeline* em que os documentos são submetidos para depois dar o resultado pretendido pela *query*.

No caso da primeira interrogação do relatório foi usado o *aggregation pipeline* em que usa todos os documentos presentes na coleção Bilhete e na primeira fase faz um *match* para filtrar apenas os Bilhetes que sejam do Filme pretendido no exemplo. De seguida fez-se um agrupamento, forçando a soma do atributo Preço de todos os Filmes após a filtração anterior, o que é equivalente a dizer para somar a receita do Filme pretendido.

Em termos da segunda interrogação é já uma agregação com 4 fases, dado que não existe, de forma nativa ao *MongoDB,* a junção natural de coleções, tal como em SQL. Dessa forma usou-se um operador "*$lookup"* que faz um *outer join* consoante os campos do documento em que estamos e o documento que queremos juntar. Assim juntamos à informação dos Bilhetes, a informação completa relativa ao cliente, dado que igualamos o NIF do Bilhete ao *id* dos documentos da coleção Cliente. Essa junção é colocada num *array* ClienteInfo, que neste caso fica apenas com um elemento porque só há um documento com o NIF, dado que é um *id*, logo não pode haver repetições (clientes iguais). Na segunda fase usou-se o operador *"$unwind"* que "abre" o array, logo transformou o único elemento lá dentro (a informação do cliente) num campo agora do documento Bilhete. Na terceira fase agrupou-se consoante o NIF, podendo agora também aparecer o nome da pessoa associada (através de ClienteInfo.Nome) e somou-se os valores chamando Gasto Total. Aqui utilizou-se o agrupamento por NIF e não pelo nome do cliente, dado que podem existir nomes iguais na coleção e assim induzia toda a *query* para valores errados. Na quarta fase, apenas se disse para ordenar o Gasto Total descendentemente que equivale a dar o *top* de clientes da Bilheteira.

3. Análise crítica do novo sistema de Base de Dados

À primeira vista este novo sistema de Base de Dados Não-Relacional é tão ambíguo que se torna complicado de entender até que ponto as vantagens são assim tão significativas ao outro método Relacional que está estabelecido e conhecido há tantos anos. Não existir a noção de chaves primárias e estrangeiras e todo o sistema de juntar tabelas e dados para conseguir obter as informações está tão enraizada na nossa mente que de repente deixa de fazer sentido todo o sistema de documentos. Toda a ideia das chaves primárias e estrangeiras e a junção de tabelas serve para todos os dados terem o máximo de consistência e o mínimo de redundância possível na agregação da Base de Dados por completo. Ao transformarmos para NoSQL, e em caso mais específico uma Base de Dados por documentos (*MongoDB*) toda essa consistência e não-redundância perde-se a favor da rapidez de resposta a uma *query* e a flexibilidade de trabalhar com a Base de Dados em si.

No caso da nossa Base de Dados, apesar que criamos três coleções (Cliente, Bilhete e Filme), na verdade, toda a nossa informação podia estar apenas num documento que seria o Bilhete de Cinema. Dentro dele iria ter uma gama de documentos embutidos que nos dariam a informação do cliente e do Filme associado a ele, sendo que assim literalmente toda a informação estaria disponível no próprio, o que se tornaria conveniente, rápido e eficiente. Obviamente que assim a ideia da consistência como falado anteriormente seria perdida, dado que numa coleção estariam *n*-vezes a mesma informação sobre o mesmo Filme ou mesmo a mesma informação sobre um cliente frequente. Toda a ideia de normalização e que os atributos deixam de ser dependentes apenas da chave composta/primária se perde, aliás nem existe essa noção, para facilitar as *querys*. Em detrimento teríamos (por exemplo) que a informação de contacto dum cliente podia mudar dum Bilhete para outro e a informação que recebíamos podia não ser a mais constante, ou seja, a mais recente e correta.

Para finalizar e analisando as nossas duas partes do projeto para a Bilheteira de Cinema é que, em última instância, nos seria benéfico usar um sistema de Base de Dados Relacional, como o MySQL dado que para uma Bilheteira o fundamental é que os dados estejam corretos, atualizados, consistentes e nunca redundantes (poupando assim também espaço). Não existe problema em demorar mais a processar *n queries*, mas sim que o *output* final seja o mais correto. Mesmo tratando-se no eventual caso futuro de os dados serem exponencialmente crescentes, até que exista um ponto de biliões de documentos e/ou entradas, isso é raro acontecer no nosso empregador, logo no caso deste projeto é mais importante a consistência que a rapidez e/ou flexibilidade que o *MongoDB* oferece.

4. Conclusão

Com a realização deste trabalho foi possível desenvolver uma Base de Dados não Relacional e orientada aos documentos a partir de um modelo Relacional de Base de Dados desenvolvido anteriormente.

Podemos, assim, inferir que habitualmente é implementado o modelo Não-Relacional quando pretendemos manipular quantidades massivas de dados, pois é neste aspeto que o modelo Relacional falha. Portanto, o modelo *NoSQL* é capaz de oferecer inúmeras vantagens nestas situações, tais como, bom desempenho, facilidade para consultas de dados, flexibilidade e escalabilidade.

Para além destas características do modelo Não-Relacional, a Base de Dados orientada aos documentos é livre de esquemas, permite consultar os documentos através de métodos de agrupamento e filtragem, é dotada de identificadores únicos e universais (*UUID*), contém todas as informações importantes num único documento e permite redundância e a inconsistência nos dados. Como já foi referido ao longo deste trabalho, o padrão utilizado para o desenvolvimento desta Base de Dados foi o *JSON* que nos proporcionou inúmeras vantagens tais como a sua flexibilidade, disponibilidade e o facto de ser uma linguagem de consulta simples e eficaz.

Neste trabalho foi utilizado o sistema de Base de Dados *MongoDB*, sendo formada por um conjunto de documentos JSON, que permitiu desenvolver o modelo Não-Relacional pretendido. A utilização desta ferramenta foi bastante intuitiva e acessível quando comparada com o *MySQL* utilizado no primeiro trabalho, pois com este as coleções podem ser dinâmicas, ou seja, não existe um *schema* pré-definido dentro das coleções logo existe uma liberdade *a priori* para os dados serem inseridos e atualizados.

Assim, podemos concluir que apesar das pequenas dificuldades ultrapassadas ao longo do desenvolvimento deste trabalho em relação à implementação de um sistema Não-Relacional orientado aos objetos a partir de um modelo Relacional, este pode acarretar inúmeras vantagens, que já foram referidas anteriormente. Para além disto, foi possível desenvolver capacidades relativas à linguagem JSON em relação à resolução das interrogações propostas no trabalho anterior, tendo sido nesta parte que surgiram maiores dificuldades. Contudo, a resolução deste trabalho acabou por ser mais acessível e intuitiva e permitiu consolidar conhecimentos não relativos a esta matéria, mas também como a anterior.

5. Referências Bibliográficas

MongoDB. (2017). Obtido de MongoDB and MySQL Compared: https://www.mongodb.com/compare/mongodb-mysql

6. Lista de Siglas e Acrónimos

BD: Base de Dados

JSON: JavaScript Object Notation

BSON: Binary JSON

NoSQL: Um termo usado maioritariamente para definir todos os sistemas de Base de Dados que não assentam num modelo Relacional parcial ou inteiramente.