P.PORTO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

JSON e MongoDB

Bruno Oliveira (bmo@estg.ipp.pt)

PEI Processamento Estruturado de Informação

Agenda

- JSON
- Introdução a MongoDB:
  - Gestão de bases de dados e coleções
  - Gestão de documentos
  - Consultas

JSON - JavaScript Object Notation

- JSON é uma sintaxe para armazenar e transportar dados;
- Um ficheiro JSON não é mais do que texto que pode ser convertido a partir de um objeto JavaScript;
- Apesar de utilizar sintaxe proveniente da linguagem JavaScript, JSON é um formato independente;

JSON - JavaScript Object Notation

- Dados são representados através de associações: chave/valor;
  - Cada associação é composta por um campo identificado com “” e um valor:"name":"John"
- Cada associação é separada por vírgula;
- Chavetas representam objetos;
- Parêntesis retos representam arrays;
- Ver: https://www.json.org/

JSON Values

- Os valores JSON podem ser dos seguintes tipos:
  - String: "name": "John"
  - Número: "age": 30
  - Booleano: "isStudent": false
  - Objeto: "address": {"city": "New York", "zip": "10001"}
  - Array: "fruits": ["apple", "banana", "orange"]
  - Null: "middleName": null

Exemplo de documento JSON

```
{
 "name": "Maria Silva",
 "isStudent": false,
 "email": "maria.silva@example.com",
 "phoneNumbers": [
  {
   "type": "home",
   "number": "123-456-789"
  },
  {
   "type": "work",
   "number": "987-654-321"
  }
 ],
 "address": {
  "street": "Rua dos Lírios",
  "city": "Lisboa"
 },
 "hobbies": ["reading", "traveling", "yoga"]
}
```

JSON e XML

- Partilham algumas características em comum:
  - São “self-describing” e por isso facilmente interpretáveis;
  - Representam dados de forma hierárquica

JSON e XML

- Diferem:
  - JSON não utiliza “tags” de fecho;
  - JSON é mais “leve” e rápido;
  - JSON suporta arrays;
  - XML tem de ser processado por um parser específico enquanto que JSON pode ser interpretado por uma função JavaScript;

JSON e XML

- XML é mais complicado de processar;
- XML permito desenvolvimento de vocabulários de forma standard;
- JSON pode ser facilmente mapeado para primitivas de programação;
- JSON é mais próximo de uma linguagem de programação e como tal é mais poderoso ao nível do processamento mas menos expressivo que o XML;

MongoDB

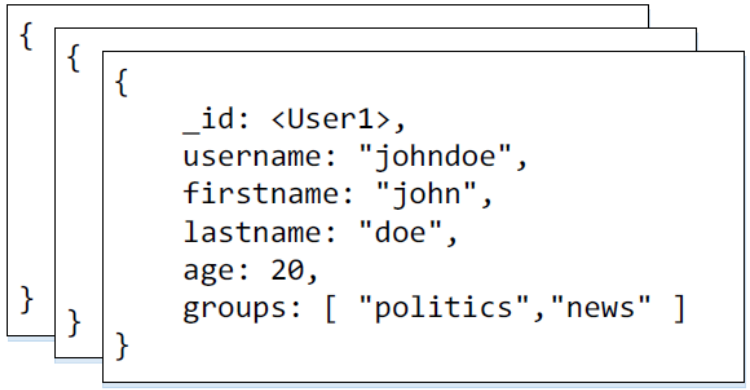
- O MongoDB é um sistema de base de dados orientada a documentos;
- Uma BD consiste em dois itens: documento, que contém dados e coleções que representa conjuntos de documentos.

Documentos BSON

- MongoDB armazena documentos no formato BSON;
- O JSON é utilizado para transportar dados e o BSON para armazenar dados;
- O documento é basicamente um Objeto JSON que o MongoDB armazena no disco em formato binário (BSON);
- O campo _id, é o identificador do documento;
- Com BSON, podemos também representar datas (Date).

https://docs.mongodb.com/manual/reference/bson-types/

Documentos BSON




```
{
  name: "sue",                          ← field: value
  age: 26,                              ← field: value
  status: "A",                          ← field: value
  groups: [ "news", "sports" ]          ← field: value
}
```

http://bsonspec.org/

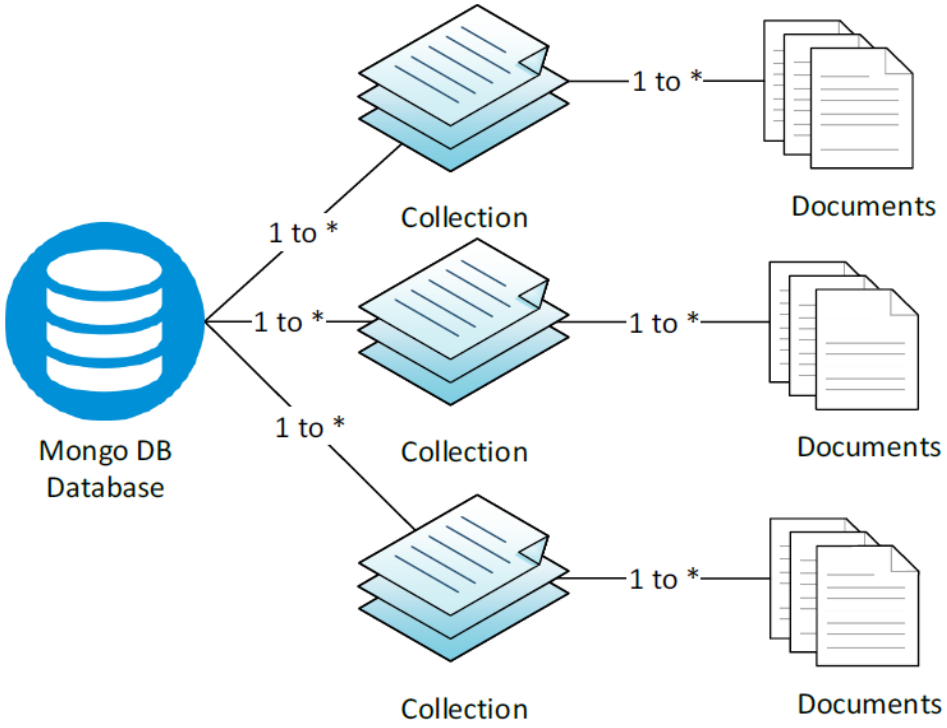
MongoDB

- O tamanho máximo dos documentos é de 16 megabytes.
- A limitação de tamanho garante que um único documento não possa usar uma quantidade excessiva de RAM ou largura de banda.

Coleções

- O MongoDB armazena documentos em coleções.
- Todos os documentos na coleção, a menos que especificado de outra forma, tem um _id automaticamente atribuído.
- Uma coleção não tem uma estrutura pré-definida.

Visão geral – conceitos MongoDB



Modelação de Dados

- Em bases de dados orientadas por documentos, representamos entidades como documentos;
- Antes de proceder à construção do documento é necessário identificar as principais entidades que pretendemos descrever e como estes se relacionam;

Modelação de Dados

- A técnica de embutir documentos é uma abordagem comum, permitindo concentrar todo o conteúdo num único documento.
- Vamos estudar outras abordagens futuramente.

Modelação de Dados

```
user document:
{
    _id: <User1>,
    username: "johndoe",
    firstname: "john",
    lastname: "doe"
    address: {
                city: "Zürich",
                country: "Switzerland"
             },
    contact: {
                phone: "+41 123-123-123",
                email: "jd@jd.com"
              }
}
```

Embedded Address

Embedded contact

Utilização do MongoDB

- Localmente: É necessário instalar o servidor de base de dados (https://www.mongodb.com/try/download/community);
- Cloud: Criar uma conta no Mongo Atlas, configurar um cluster gratuito (https://www.mongodb.com/cloud/atlas);
- Após a configuração do servidor podemos utilizar o Mongo Compass (já incluído na instalação do servidor Mongo) para ligar ao servidor e interagir com a base de dados.

Consultas

- O MongoDB disponibiliza mecanismos para consultar qualquer campo de um documento;
- Disponibiliza também vários tipos de índices para otimizar uma grande variedade de consultas (índices para texto, dados geográficos, etc);

Comandos para gestão de base de dados

- Para além do Compass, a interação com o mongo pode ser realizada através da Shell (também acessível a partir do Mongo Compass):
  - Utilizar uma base de dados:
  - Excluir uma base de dados:
  - Visualizar as coleções existentes:

```
use usersdb
```

```
db.dropDatabase()
```

```
show collections
```

Comandos para gestão de base de dados

- Utilizando a Shell do MongoDB:
- Para criar a coleção chamada users executámos o seguinte comando:

```
db.createCollection("users")
```

- Para excluir uma coleção podemos usar o comando drop:

```
db.person.drop()
```

Gestão de documentos - Inserir

- Inserir uma entrada na BD (usersdb) utilizando a coleção: users;

```
db.users.insert(
        {firstname: 'john', lastname: 'doe', job:'designer'}
)
```

- Para inserir múltiplos documentos

```
db.users.insertMany([
        {firstname: 'john', lastname: 'doe' , job:'designer'},
        {firstname: 'alanis', lastname: 'patsy' , job:'programmer'}
])
```

Gestão de documentos - Atualizar

- Atualizar um documentao na BD (usersdb) utilizando a coleção: users;

```
db.users.updateOne(
      {'name' : 'john' },
      { $set: {'job' : 'software engineer'}}
   );
```

- Para atualizar múltiplos documentos

```
db.users.updateMany(
      {'name' : 'john' },
      { $set: {'job' : 'software engineer'}}
   );
```

Gestão de documentos - Atualizar

- Os seguintes métodos também podem ser utilizados:
  - db.collection.replaceOne(): Substitui um documento que corresponda a um filtro especificado;
  - db.collection.findAndModify(): Modifica e devolve um único documento. Por padrão, o documento devolvido não inclui as modificações feitas na atualização.
  - db.collection.bulkWrite(): Executa múltiplas operações de escrita considerando uma ordem de execução.
  - Mais informação: https://www.mongodb.com/docs/manual/reference/update-methods/

Gestão de documentos - Atualizar

- Também é possível adicionar novos campos:

```
db.users.updateMany(
      { },
      { $set: {'birthdate': new Date('1970-08-02')}}
   );
```

- Eliminar campo:

```
db.users.updateMany(
      { },
      { $unset: {'birthdate': ''}}
   );
```

Gestão de documentos - Remover

- Podemos remover todos os documentos utilizando:

```
db.users.deleteMany({});
```

- Podemos remover vários documentos com base numa condição:

```
db.users.deleteMany({'name' : 'john' });
```

- Remover um documento

```
db.users.deleteOne({'name' : 'john' });
```

```
db.users.deleteOne({ _id: ObjectId("id_do_documento") });
```

https://www.mongodb.com/docs/manual/tutorial/remove-documents/

Consultas com mongo: filtrar

- Considere o exemplo fornecido: primer-dataset;
- Ver todos os documentos na coleção:
  - db.restaurants.find({})
  - db.restaurants.find()
  - Para visualizar os dados de forma mais “simpática” utilize o método: pretty(): db.restaurants.find().pretty()
- Devolver todos os restaurantes com o tipo de cozinha: “Bakery”:
  - db.restaurants.find({ cuisine: "Bakery" })

Consultas com mongo: filtrar

- Mostrar todos os restaurantes de cozinha: “Bakery” ou “American”:

```
db.restaurants.find(
        {
           cuisine:
                {$in: ["Bakery", "American"]}
        }
)
```

Consultas com mongo: projetar

- Também podemos projetar apenas alguns elementos do documento: Apresentar a cozinha e nome de cada restaurante:

```
db.restaurants.find({},{name:1, cuisine:1})
```

- Apresentar a cozinha e nome de cada restaurante de “Bronx”:

```
db.restaurants.find({borough:"Bronx"},{name:1, cuisine:1})
```

- Por defeito é sempre incluído o campo: _id, que podemos remover através:

```
db.restaurants.find({borough:"Bronx"},{name:1, cuisine:1, _id:0})
```

Índices

- Considere os seguintes cenários:
  - Precisamos frequentemente procurar utilizadores por ID (single field index). Podemos criar um índice no campo ID do utilizador;
  - Precisamos procurar informações do utilizador por morada. A morada é armazenada num documento embutido com campos como cidade e código postal. Podemos criar um índice (Single field index on a object) no documento que armazena a morada;
  - Precisamos de procurar os utilizadores com a profissão “designer” e por data de nascimento. Podemos criar um único índice (compound index) nos campos job e birthdate.

Índices

- Criar um índice descendente para o caso 1 (Para um índice de campo único, a ordenação (crescente ou decrescente) da chave do índice não importa porque o MongoDB pode percorrer o índice em qualquer direção:

```
db.users.createIndex( { id: -1 } )
```

- Para o caso 2:

```
db.users.createIndex( { morada: -1 } )
```

- Para o caso 3:

```
db.users.createIndex( { job: -1, birthdate:1 } )
```

- O caso 3 não suporta pesquisas que consultem documentos com ambos os campos por order ascendente ou descendente;

https://www.mongodb.com/docs/manual/indexes/

Schemas

- Os schemas permitem impor regras sobre a estrutura dos documentos.
- Numa fase inicial de desenvolvimento, a validação de esquema pode impor restrições desnecessárias;
- A validação com um *schema* é mais útil para uma aplicação estabelecida, onde há um entendimento claro de como organizar os dados.

Schemas

- Cenários de utilização:
  - Para uma coleção com dados de utilizadores, garantir que a senha de acesso seja armazenado apenas como string.
  - Para uma coleção de vendas, garantir que o campo do produto pertença a uma lista de produtos existente.

Schemas

- O JSON Schema é uma linguagem que permite anotar e validar documentos JSON.
- Podemos especificar regras de validação para os campos utilizando um formato legível.
  - https://json-schema.org
- No entanto, JSON Schema ainda é um draft e o MongoDB suporte a o draft 4.

Exemplo para o dataset: primerdataset

- Title: O esquema tem o título "restaurante", mas não afeta a validação em si.
- Properties: Esta secção define os campos esperados em cada documento e os seus correspondentes tipos de dados (bsonType).
- Required: Esta secção especifica os campos obrigatórios na inserção de documentos.

```
{
  "title": "restaurant",
  "properties": {
    "_id": {
      "bsonType": "objectId"
    },
    "address": {
      "bsonType": "object",
      "properties": {
        "building": {
          "bsonType": "string"
        },
        "street": {
          "bsonType": "string"
        }
      }
    },
    "borough": {
      "bsonType": "string"
    },
    "cuisine": {
      "bsonType": "string"
    }
  },
  "required": [
    "_id"
  ]
}
```

Associar um schema a uma coleção

```
db.runCommand({
  collMod: "restaurants",
  validator: {
    $jsonSchema: {
      "title": "restaurant",
      "properties": {
        "_id": {
          "bsonType": "objectId"
        },
        (...)
})
```

Coleção a aplicar o schema

Schema

Bibliografia/referências

- Referências Web:
  - https://www.json.org/json-en.html
  - https://docs.mongodb.com/manual/
  - Phaltankar, A. Ahsan, J., Harrison, M., Nedov, L. (2020). MongoDB Fundamentals: A hands-on guide to using MongoDB and Atlas in the real world. Packt Publishing

P.PORTO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

JSON e MongoDB

Bruno Oliveira (bmo@estg.ipp.pt)

PEI Processamento Estruturado de Informação