

GeoReferência no MongoDB

Entendendo o GeoJSON

GeoJSON é um formato de dados baseado em JSON (JavaScript Object Notation) que é utilizado para representar dados geoespaciais. Ele é projetado para ser simples, legível e fácil de usar tanto por humanos quanto por máquinas, e é amplamente utilizado em aplicações de mapeamento e sistemas de informação geográfica (SIG).

Principais Características do GeoJSON

1. Formato Baseado em JSON:

- GeoJSON utiliza a sintaxe JSON, o que o torna fácil de integrar com outras aplicações web e APIs que já utilizam JSON.

2. Representação de Estruturas Geométricas:

- GeoJSON pode representar diferentes tipos de objetos geoespaciais, como pontos, linhas, polígonos, e coleções de geometria.

3. Suporte a Dados de Propriedade:

- Além das informações geoespaciais, cada objeto GeoJSON pode incluir propriedades adicionais que descrevem características do objeto, como nomes, descrições, ou qualquer outro tipo de dado adicional.

Estruturas Geométricas no GeoJSON

O GeoJSON pode representar várias estruturas geométricas, cada uma usada para diferentes tipos de dados espaciais:

1. Point (Ponto):

- Representa uma única localização no espaço. É definido por coordenadas de longitude e latitude.

```
{
  "type": "Point",
  "coordinates": [100.0, 0.0]
}
```

2. LineString (Linha):

- Representa uma sequência de pontos conectados por linhas retas. É usado para representar caminhos ou linhas.

```
{
  "type": "LineString",
  "coordinates": [
    [100.0, 0.0],
    [101.0, 1.0]
  ]
}
```

3. Polygon (Polígono):

- Representa uma área fechada definida por uma sequência de coordenadas. O primeiro e o último ponto devem ser o mesmo para formar um polígono fechado.

```
{
  "type": "Polygon",
  "coordinates": [
    [
      [100.0, 0.0],
      [101.0, 0.0],
      [101.0, 1.0],
      [100.0, 1.0],
      [100.0, 0.0]
    ]
  ]
}
```

4. MultiPoint (Multiponto):

- Representa múltiplos pontos. Pode ser usado para representar vários locais individuais.

```
{
  "type": "MultiPoint",
  "coordinates": [
    [100.0, 0.0],
    [101.0, 1.0]
  ]
}
```

5. **MultiLineString (Multilinha):**

- Representa múltiplas linhas. É útil para descrever conjuntos de linhas conectadas ou desconectadas.

```
{
  "type": "MultiLineString",
  "coordinates": [
    [
      [100.0, 0.0],
      [101.0, 1.0]
    ],
    [
      [102.0, 2.0],
      [103.0, 3.0]
    ]
  ]
}
```

6. **MultiPolygon (Multipolígono):**

- Representa múltiplos polígonos. Pode ser usado para representar áreas complexas compostas por vários polígonos separados.

```
{
  "type": "MultiPolygon",
  "coordinates": [
    [
      [
        [100.0, 0.0],
        [101.0, 0.0],
        [101.0, 1.0],
        [100.0, 1.0],
        [100.0, 0.0]
      ]
    ],
    [
      [
        [102.0, 2.0],
        [103.0, 2.0],
        [103.0, 3.0],
        [102.0, 3.0],
        [102.0, 2.0]
      ]
    ]
  ]
}
```

Banco de Dados Não Relacional - Desenvolvimento de Software Multiplataforma

```
]
}
```

7. GeometryCollection (Coleção de Geometrias):

- Representa uma coleção de diferentes tipos de geometria. É útil quando é necessário agrupar diferentes formas em um único objeto.

```
{
  "type": "GeometryCollection",
  "geometries": [
    {
      "type": "Point",
      "coordinates": [100.0, 0.0]
    },
    {
      "type": "LineString",
      "coordinates": [
        [101.0, 0.0],
        [102.0, 1.0]
      ]
    }
  ]
}
```

Propriedades no GeoJSON

Cada objeto GeoJSON pode conter um campo `"properties"` que é um objeto contendo dados adicionais sobre a geometria. Essas propriedades podem incluir qualquer informação relevante, como o nome de uma cidade, uma descrição ou qualquer outro metadado que você queira armazenar.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [102.0, 0.5]
  },
  "properties": {
    "name": "Example Point"
  }
}
```

Uso de GeoJSON no MongoDB

No MongoDB, o GeoJSON é utilizado em conjunto com o índice geoespacial `2dsphere` para executar consultas eficientes que envolvem dados geoespaciais. A compatibilidade com GeoJSON permite que o MongoDB suporte uma ampla gama de operações geoespaciais, como consultas de proximidade e intersecção, permitindo o uso de dados espaciais complexos.

Carregando os arquivos municipios.json e estados.json no MongoDB

```
const fs = require('fs')

const dadosEstados = fs.readFileSync('json/estados.json')
const jsonEstados = JSON.parse(dadosEstados)

const dadosMunicipios = fs.readFileSync('json/municipios.json')
const jsonMunicipios = JSON.parse(dadosMunicipios)

use('geo')
db.estados.insertMany(jsonEstados)
db.estados.createIndex({uf:1},{unique:true, name:"idx_estados_uf"})

db.municipios.insertMany(jsonMunicipios)
db.municipios.createIndex({codigo_ibge:1},{unique:true,
name:"idx_municipios_codigoIbge"})

use('geo')
db.estados.find().count() // 27

use('geo')
db.municipios.find().count() // 5570
```

Collection Estados

```
{
  "codigo_uf": 11,
  "uf": "RO",
  "nome": "Rondônia",
  "local": {
    "type": "Point",
    "coordinates": [-63.34, -10.83] // Longitude, Latitude
  }
}
```

Efetuar o update na base

```
//Adequando os estados para o formato GeoJSON
use('geo')
db.estados.find().forEach(function(estado) {
  db.estados.updateOne(
    { _id: estado._id },
    {
      $set: {
        local: {
          type: "Point",
          coordinates: [estado.longitude, estado.latitude]
        }
      },
      $unset: {
        latitude: "",
        longitude: ""
      }
    }
  )
})
```

Banco de Dados Não Relacional - Desenvolvimento de Software Multiplataforma

```
//Adequando os municipios para o formato GeoJSON
use('geo')
db.municipios.find().forEach(function(municipio) {
  db.municipios.updateOne(
    { _id: municipio._id },
    {
      $set: {
        local: {
          type: "Point",
          coordinates: [municipio.longitude, municipio.latitude]
        }
      },
      $unset: {
        latitude: "",
        longitude: ""
      }
    }
  )
})
```

Verificando a alteração

```
use('geo')
db.estados.find({}, { local: 1 });
```

Criar índices 2dsphere

```
use('geo')
db.estados.createIndex({ local: "2dsphere" })
db.municipios.createIndex({ local: "2dsphere" })
```

O índice **2dsphere** é um tipo de índice geoespacial no MongoDB que permite armazenar e consultar dados geoespaciais em um sistema de coordenadas esféricas, o que é ideal para representar a superfície da Terra. Este índice suporta tanto dados GeoJSON (**Point**, **LineString**, **Polygon**, etc.) quanto pares de coordenadas legadas (longitude e latitude).

Banco de Dados Não Relacional - Desenvolvimento de Software Multiplataforma

Ele utiliza uma estrutura de dados conhecida como "R-tree", que é otimizada para armazenamento e consulta de dados multidimensionais, como coordenadas geoespaciais. Essa estrutura é altamente eficiente para operações de vizinhança e intersecção, que são comuns em consultas geoespaciais.

Exemplos de consulta

Encontrar um ponto próximo a uma localização específica

```
use('geo')
//https://www.mapcoordinates.net/pt
db.municipios.find({
  local: {
    $near: {
      $geometry: {
        type: "Point",
        coordinates: [-47.4495, -23.5313] //Fatec
      },
      $maxDistance: 50000 // Em metros (50km)
    }
  }
},{nome:1, _id: 0});
```

Encontrar pontos dentro de um círculo

```
use('geo')
db.municipios.find({
  local: {
    $geoWithin: {
      $centerSphere: [[-47.4495, -23.5313],
        20 / 6378.1] // raio em radianos
    }
  }
},{nome:1, _id:0});
```


Banco de Dados Não Relacional - Desenvolvimento de Software Multiplataforma

Aqui, são 20km e 6378.1 é o raio médio da Terra em quilômetros, e a divisão é necessária para converter a distância para radianos.

Por que converter quilômetros para radianos?

O MongoDB usa a projeção geodésica de esferas para calcular distâncias em uma esfera (como a Terra), e a distância em `$centerSphere` é especificada em radianos. Portanto, se quisermos usar uma distância em quilômetros (ou milhas, etc.), precisamos converter essa distância para radianos.

Como funciona a conversão?

1. **Raio da Terra:** Aproximadamente 6,378.1 km. Este valor é uma média, pois a Terra não é uma esfera perfeita, mas sim um esferoide oblato. Ainda assim, esse é um valor amplamente aceito para cálculos de aproximação.
2. **Conversão para Radianos:**
 - No contexto da consulta:
 - A distância que queremos considerar é 100 km.
 - O raio médio da Terra é 6378.1 km.
3. **Aplicação na Consulta:**
 - A expressão $100 / 6378.1$ divide a distância desejada (100 km) pelo raio da Terra (6378.1 km).
 - **Resultado:** $100 / 6378.1 \approx 0.01567$ radianos.

Por que usar radianos?

Radianos são usados em cálculos que envolvem distâncias em esferas porque são uma medida de ângulo diretamente aplicável à geometria da esfera. Quando se trabalha com geolocalização, usar radianos facilita o cálculo de distâncias e áreas ao redor da superfície da Terra.

Encontrar pontos dentro de um polígono

```
use('geo')
db.municipios.find({
  local: {
    $geoWithin: {
      $polygon: [
        [
          -47.52245119307611,
          -23.4746792368816
        ],
        [
          -47.36371415749238,
          -23.485492279823788
        ],
        [
          -47.37014735514302,
          -23.585489127167406
        ],
        [
          -47.52113011339904,
          -23.586575792902778
        ],
        [
          -47.53276287268014,
          -23.497906758689055
        ],
        [
          -47.52244799253009,
          -23.474711718992197
        ]
      ]
    }
  }
},{nome:1, _id:0});
```

Encontrar pontos dentro de uma caixa delimitadora (rectangle/bounding box)

```
use('geo')
db.municipios.find({
  local: {
    $geoWithin: {
      $box: [
        [
          -48.09911197926928,
          -23.079243685024124
        ], //canto inferior esquerdo
        [
          -46.29894805236373,
          -23.871065212159195
        ] //canto superior direito
      ]
    }
  }
}, {nome:1, _id:0});
```

GeoJSON.io

- [GeoJSON.io](https://geojson.io) é uma ferramenta de código aberto que permite criar, editar e visualizar dados em formato GeoJSON.
- Você pode traçar pontos, linhas e polígonos diretamente no mapa, e ele fornece as coordenadas automaticamente.

Geodata-BR

Este projeto contém arquivos [Geojson](https://github.com/tbrugz/geodata-br/) com os perímetros dos municípios brasileiros dividido por estado

<https://github.com/tbrugz/geodata-br/>

Diferença entre \$geoWithin e \$near

No MongoDB, `$geoWithin` e `$near` são operadores geoespaciais que permitem realizar consultas baseadas em localizações, mas eles são usados para propósitos diferentes e retornam resultados distintos. Vamos explorar as principais diferenças entre esses operadores.

1. \$geoWithin

Descrição:

- O operador `$geoWithin` é usado para encontrar documentos onde a localização geoespacial (`Point`, `LineString`, ou `Polygon`) está dentro de uma área geográfica especificada. Essa área pode ser definida por uma forma geométrica como um polígono, círculo ou caixa delimitadora (retângulo).

Uso Comum:

- Utilizado para encontrar documentos que estão contidos dentro de uma área específica.
- É ideal para consultas como "quais locais estão dentro de uma determinada cidade?" ou "quais lojas estão dentro deste polígono?".

Áreas Suportadas pelo `$geoWithin`:

- `$polygon`: Para definir um polígono fechado.
- `$center`: Para definir um círculo.
- `$centerSphere`: Para definir um círculo em radianos sobre a superfície esférica.
- `$box`: Para definir uma caixa delimitadora (retângulo).

2. \$near

Descrição:

- O operador `$near` é usado para encontrar documentos que estão próximos de um ponto geográfico específico. Ele retorna os documentos ordenados pela distância da localização especificada.

Uso Comum:

- Utilizado para consultas de proximidade, como "quais são os locais mais próximos de mim?".

Banco de Dados Não Relacional - Desenvolvimento de Software Multiplataforma

- É ideal para consultas como "encontre os restaurantes mais próximos deste ponto" ou "quais estações de serviço estão mais próximas?".

Principais Características do **\$near**:

- Ordenação por Distância: Os resultados são automaticamente ordenados pela distância, do mais próximo ao mais distante.
- \$maxDistance: Permite definir uma distância máxima (em metros) para limitar os resultados.
- Aplicável a **Point**: O operador **\$near** só pode ser usado em campos de tipo **Point** do GeoJSON.

Diferenças Resumidas

Característica	\$geoWithin	\$near
Propósito	Encontrar documentos dentro de uma área	Encontrar documentos próximos a um ponto
Retorno de Resultados	Documentos contidos na área especificada	Documentos ordenados por proximidade
Área Definida	Polígono, círculo, caixa delimitadora	Ponto (apenas)
Aplicabilidade	Usado para verificar a inclusão em áreas	Usado para calcular proximidade/distância
Uso de Point	Suporta Point , LineString , Polygon	Suporta apenas Point
Distância	Não considera distância para o retorno	Considera e ordena por distância

Quando usar cada um?

- Use `$geoWithin` quando você precisa verificar se uma localização geoespacial está dentro de uma área específica, como para consultar locais dentro de uma cidade ou dentro de um polígono arbitrário.
- Use `$near` quando o interesse é encontrar os locais mais próximos de um ponto específico, como listar as lojas ou serviços mais próximos da sua localização.

Esses operadores permitem que o MongoDB ofereça flexibilidade para diferentes necessidades geoespaciais, dependendo de como você deseja filtrar e classificar seus dados de localização.

Operador \$lookup

O operador `$lookup` no MongoDB é extremamente útil para realizar junções entre coleções, permitindo que você combine dados de diferentes fontes em uma única consulta. Com as coleções `estados` e `municipios`, podemos explorar diversas possibilidades.

Associar cada município ao seu estado

```
use('geo')

db.municipios.aggregate([

  {

    $lookup: {

      from: "estados",

      localField: "codigo_uf",

      foreignField: "codigo_uf",

      as: "estado"

    }

  },

  {

    $project: {

      "_id": 0,

      "nome": 1,

      "local.coordinates": 1,

      "estado.nome": 1

    }

  }

])
```

```
])
```

Encontrar todos os municípios de cada estado

```
use('geo')
db.estados.aggregate([
  {
    $lookup: {
      from: "municipios",
      localField: "codigo_uf",
      foreignField: "codigo_uf",
      as: "relacaoMunicipios"
    }
  },
  {
    $project: {
      "_id": 0,
      "nome": 1,
      "local.coordinates": 1,
      "relacaoMunicipios.nome": 1
    }
  }
])
```

Filtrando os dados de apenas um determinado estado

```
use('geo')
db.estados.aggregate([
  {
    $lookup: {
      from: "municipios",
      localField: "codigo_uf",
      foreignField: "codigo_uf",
      as: "relacaoMunicipios"
    }
  }
])
```


Banco de Dados Não Relacional - Desenvolvimento de Software Multiplataforma

```
}  
},  
{  
  $match: {  
    nome: "Rio de Janeiro"  
  }  
},  
{  
  $project: {  
    "_id": 0,  
    "nome": 1,  
    "local.coordinates": 1,  
    "relacaoMunicipios.nome": 1  
  }  
}  
])
```

Utilizando expressão regular no filtro

```
use('geo')  
db.estados.aggregate([  
  {  
    $lookup: {  
      from: "municipios",  
      localField: "codigo_uf",  
      foreignField: "codigo_uf",  
      as: "relacaoMunicipios"  
    }  
  },  
  {  
    $match: {  
      nome: {$regex: /Rio/i}  
    }  
  },  
  {  
    $project: {  
      "_id": 0,  

```

Banco de Dados Não Relacional - Desenvolvimento de Software Multiplataforma

```
        "nome": 1,  
        "local.coordinates": 1,  
        "relacaoMunicipios.nome": 1  
      }  
    }  
  ])
```

Calcular a média das Latitudes de cada município, agrupando por estado.

```
db.municipios.aggregate([  
  {  
    $lookup: {  
      from: "estados",  
      localField: "codigo_uf",  
      foreignField: "codigo_uf",  
      as: "estado"  
    }  
  },  
  {  
    $group: {  
      _id: "$estado.nome",  
      media_latitude: { $avg: "$latitude" }  
    }  
  }  
])
```

Encontrando municípios com determinada latitude e longitude dentro de um estado específico

```
db.municipios.aggregate([  
  {  
    $lookup: {  
      from: "estados",  
      localField: "codigo_uf",  
      foreignField: "codigo_uf",  
      as: "estado"  
    }  
  }  
])
```

Banco de Dados Não Relacional - Desenvolvimento de Software Multiplataforma

```
    },  
    {  
      $match: {  
        "estado.nome": "Rondônia",  
        "local": {  
          $geoWithin: {  
            $centerSphere: [[-63.34, -10.83], 0.1] //  
Círculo de 100km ao redor de um ponto  
          }  
        }  
      }  
    },  
    {  
      $project: {  
        nome: 1,  
        "estado.nome": 1  
      }  
    }  
  ]  
})
```

Calculando a distância média entre os municípios de um estado e sua capital

```
db.municipios.aggregate([  
  {  
    $lookup: {  
      from: "estados",  
      localField: "codigo_uf",  
      foreignField: "codigo_uf",  
      as: "estado"  
    }  
  },  
  {  
    $match: {  
      "estado.capital": true  
    }  
  },  
  {  
    $project: {  
      distancia: {  
        $geoDistance: {  
          from: "$local",
```

Banco de Dados Não Relacional - Desenvolvimento de Software Multiplataforma

```
        to: "$estado.local"
    }
}
},
{
  $group: {
    _id: null,
    media_distancia: { $avg: "$distancia" }
  }
}
])
```