

1. Introdução ao MongoDB

Este capítulo apresenta alguns conceitos introdutórios sobre o MongoDB.

1.1 Introdução

MongoDB é um banco de dados NoSQL orientado a objetos, simples, dinâmico e escalonável. É baseado no modelo de armazenamento de documentos NoSQL. NoSql significa Not Only SQL, ou seja, não somente SQL. Desta forma, os objetos de dados são armazenados como documentos, separados dentro de uma coleção - em vez de armazenar os dados nas colunas e linhas de uma tabela no banco de dados relacional. O MongoDB usa documentos JSON ou BSON para armazenar dados. Cabe salientar que a motivação do MongoDB é implementar um armazenamento de dados que forneça alto desempenho, alta disponibilidade e escalonamento automático. Distribuições gerais para MongoDB suportam Windows, Linux, Mac OS X e Solaris.

1.1.1 O que é um banco de dados NoSQL?

O uso do termo NoSQL é resultado de uma reunião realizada em 2009, na cidade de São Francisco, nos Estados Unidos, organizada por Johan Oskarsson. O objetivo desta reunião foi para discutir sobre projetos de bancos de dados de código aberto, não relacionais, sem esquema e distribuídos.

Os bancos de dados NoSQL podem ser divididos em:

- a) Bancos de dados Key-Value: armazena uma chave (usada na consulta) e um valor (retorno da consulta). Costumam ser usados como cache das aplicações, uma vez que armazenam partes críticas de dados na memória para acesso de baixa latência. Os exemplos de bancos de dados desta categoria são: Redis, Voldemort, Memcached e Riak;
- b) Banco de dados orientados a documentos: Nesta categoria os dados normalmente são armazenados no formato JSON. Nesta categoria se encaixa o MongoDB, que recebe os valores no formato JSON e os armazena em um formato binário chamado BSON. Fazem parte desta categoria também: CouchDB, OrientDB, RavenDB e TerraStore.
- c) Bancos de dados colunares: Podemos pensar nesta categoria como uma estrutura agregada de dois níveis, onde existe um identificador da linha e um mapa com valores mais detalhados. Exemplos de bancos de dados colunares são: Cassandra, HBase, Hipertable e Amazon SimpleDB;
- d) Banco de dados orientados a grafos: A grande diferença para o modelo clássico está na representação explícita de relacionamentos entre os dados, através de um modelo com vértices, chamados de nós, e arcos, chamados de relações. Exemplos de bancos de dados orientado a grafos: Neo4J, OrientDB e Infnit Graph.

1.1.2 Bancos de Dados NoSQL não tem schema

Os bancos de dados relacionais exigem que as estruturas (schemas) sejam definidas antes que você possa adicionar dados nas tabelas. Os bancos de dados NoSQL, entretanto, permitem a inclusão dos dados sem um esquema predefinido. Isso facilita fazer alterações nos aplicativos em tempo real, sem se preocupar com interrupções de serviço para atualização da estrutura de uma determinada tabela, de forma que o desenvolvimento tende a ser mais rápido.

1.1.3 Propriedades BASE

Em muitas situações, as transações ACID (acrônimo de Atomicidade, Consistência, Isolamento e Durabilidade) são pessimistas, isto é, estão mais preocupadas com a segurança dos dados do que o domínio realmente exige. Atualmente, a internet cria uma quantidade de dados que precisam ser processados, analisados e entregues aos usuários que os requisitaram. Aplicações de grande escala se preocupam com disponibilidade e redundância, de modo que essas propriedades são difíceis de se alcançar usando as propriedades ACID. Desta forma, quando falamos de NoSQL, pensamos nas propriedades BASE. Em inglês BASE quer dizer:

- Basic Availability: O banco de dados parece funcionar a maior parte do tempo.
- Soft-state: Os dados não precisam ser consistentes com gravação, nem réplicas diferentes precisam ser mutuamente consistentes o tempo todo.
- Eventual consistency: Os dados ficarão consistentes em algum momento

As propriedades BASE, desta forma, são muito mais suaves do que as garantias ACID.

1.1.4 Quando o NoSQL é indicado?

Quando trabalhamos com banco relacionais, percebemos que quanto maior é o tamanho da base de dados, maior será o tempo de resposta na geração de consultas. Isso se torna inviável em situações onde os dados devem ser mostrados em real-time. Imagine, por exemplo uma consulta demorar 5 ou 10 minutos para trazer as informações. Em um ambiente competitivo como o que vivemos atualmente, essa demora não é aceitável.

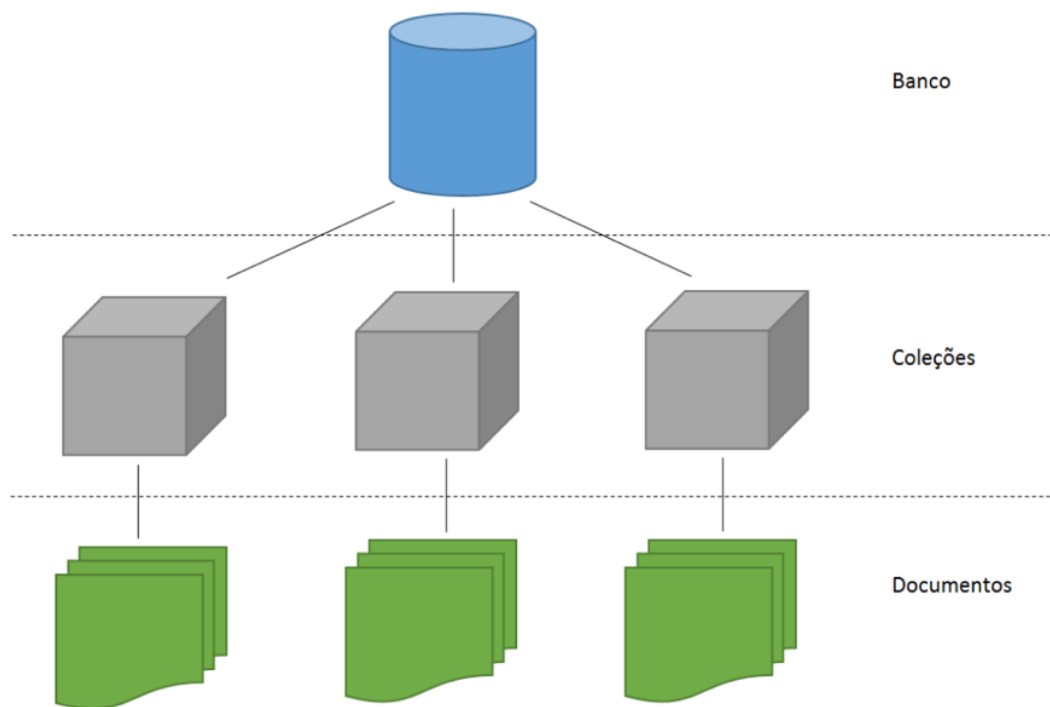
Assim, o modelo de dados NoSQL é indicado para aplicações que irão trabalhar com grandes quantidade de dados, onde um modelo relacional já não atende. O NoSQL veio exatamente para agilizar consultas e buscas de maneira quase que instantânea. Pela sua facilidade em escalar diversos servidores horizontalmente e pela alta performance grandes empresas vem adotando essa tecnologia.

1.1.5 MongoDB não usa a linguagem SQL.

A Structured Query Language é a linguagem mais conhecida e mais usada para consulta de dados. É usada principalmente com os bancos de relacionais, mas tem suas adaptações para bancos de dados não relacionais como o Hive que possui o HQL, o Cassandra que possui o CQL.

Os documentos recebidos pelo MongoDB normalmente estão no formato JSON, e são armazenados em um formato aberto e binário desenvolvido pela equipe do MongoDB, o BSON (Binary JSON).

Mas, como os dados são organizados dentro do MongoDB? Conforme ilustrado abaixo, uma instância do MongoDB pode possuir vários bancos de dados, cada banco de dados possui várias coleções, e cada coleção possui documentos. Esses documentos são conteúdos indexáveis recebidos, normalmente, no formato JSON e armazenados no formato BSON.



Bem, se fizermos um paralelo com um banco de dados relacional teremos o seguinte:

Banco de Dados Relacional	MongoDB
Banco de Dados	Banco de Dados
Tabela	Coleção
Linha	Documento

1.2. Projeto de Exemplo

O MongoDB é um banco de dados de documentos que está [disponível](#) na Nuvem ou como Servidor local. Na Nuvem, vamos nos conectar a um cluster [MongoDB Atlas](#). MongoDB

Atlas é um serviço de banco de dados em nuvem que hospeda seus dados em instâncias do MongoDB. Como servidor local, temos a versão Enterprise e Community. Vamos usar a versão [Community](#). Faça o download e instale em seu computador.

1.2.2 Configurando o projeto

Vamos criar um diretório chamado node-mongo que irá retornar e cadastrar informações sobre os cursos oferecidos no IFRS. Em seguida, vamos iniciar um projeto Node com auxílio do gerenciador de pacotes NPM. Assim, digite os seguintes comandos no terminal:

```
cd node-mongo
npm init -y
```

Este comando cria um arquivo chamado package.json. Cabe lembrar que se você especificar a opção -y no comando, o npm aceita automaticamente os valores padrões.

Em seguida, adicione o MongoDB às dependências do projeto. Use o seguinte comando para instruir o npm a baixar e instalar o pacote mongodb.

```
npm i mongodb
```

Este comando baixa o pacote mongodb e as dependências necessárias para sua instalação. Ele também os salva no diretório chamado node_modules e registra as informações de dependência no arquivo package.json gerado na etapa anterior.

Agora, vamos instalar o Nodemon como dependência de desenvolvimento no projeto. O Nodemon é um utilitário que monitora quaisquer mudanças em seu código fonte e automaticamente reinicia seu servidor.

```
npm i --save-dev nodemon
```

Em seguida, vamos alterar o arquivo package.json para usarmos o nodemon:

```
{
  "name": "mongo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon",

    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
```

```
"license": "ISC",
"dependencies": {
  "mongodb": "^6.2.0"
},
"devDependencies": {
  "nodemon": "^3.0.1"
}
}
```

Pronto!

1.2.3 Conectando-se ao MongoDB

Nesta etapa, nós vamos criar e executar um aplicativo que usa o driver Node.js MongoDB para se conectar à sua instância do MongoDB e, depois, executar uma consulta aos dados.

Passamos instruções ao driver sobre onde e como se conectar à sua instância do MongoDB através de uma string de conexão. Esta string inclui informações sobre o nome do host ou endereço IP e porta de sua instância, mecanismo de autenticação, credenciais do usuário (quando aplicável) e outras opções de conexão.

Como estamos testando em nosso computador, a string de conexão deverá ser algo assim:

```
mongodb://127.0.0.1:27017
```

Após, vamos criar o arquivo responsável por configurar e subir o servidor. Primeiro, crie um arquivo chamado `index.js`. Agora, adicione o seguinte código neste arquivo, substituindo a variável `uri` por sua string de conexão. Certifique-se de substituir a seção `"<password>"` da string de conexão pela senha que você criou para o seu usuário. Substitua `<dbname>` pelo nome do banco de dados que as conexões usarão por padrão.

```
const { MongoClient, ServerApiVersion } = require("mongodb");
// Substitua a string uri pela string de conexão do MongoDB
const uri = "mongodb://127.0.0.1:27017";
// Crie um MongoClient com um objeto MongoClientOptions para definir a versão
estável da API
const client = new MongoClient(uri, {
  serverApi: {
    version: ServerApiVersion.v1,
    strict: true,
    deprecationErrors: true,
  }
})
```

```

);
async function run() {
  try {

    await client.connect();
    const database = client.db('ifrs_db');
    // Envie um ping para confirmar uma conexão bem-sucedida
    await database.command({ ping: 1 });
    const collection = database.collection('courses');

    console.log("Funcionou!");
  } finally {
    // Garante que o client fechará quando você terminar ou der erro
    await client.close();
  }
}
run().catch(console.dir);

```

Execute o código com o seguinte comando no terminal:

```
npm run dev
```

Veja o resultado!

```

PS C:\Users\Mauri\Downloads\node-mongo> npm run dev

> mongo@1.0.0 dev
> nodemon

[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Funcionou!
[nodemon] clean exit - waiting for changes before restart

```

1.2.4 Inserindo Documentos

Para adicionar novos documentos a uma coleção, você pode usar o método `insertOne()` ou `insertMany()`. Esses métodos aceitam documentos únicos ou múltiplos, respectivamente. O driver gera automaticamente um campo `_id` exclusivo para os documentos, a menos que especificado um específico.

Você pode especificar o documento a ser inserido pela operação de gravação `insertOne()` em um objeto JSON da seguinte maneira:

```
// cria um documento a ser inserido
const courseDocument = {
  name: "Análise e Desenvolvimento de Sistemas",
  type: "Superior",
};
```

Assim, para inserir os dados do objeto `courseDocument`, especifique o documento como o primeiro parâmetro de sua chamada para o método `insertOne()` conforme mostrado abaixo:

```
console.log(`Documento inserido com o _id: ${result.insertedId}`);
```

Veja como ficou o arquivo `index.js`:

```
const { MongoClient, ServerApiVersion } = require("mongodb");

// Substitua a string uri pela string de conexão do MongoDB
const uri = "mongodb://127.0.0.1:27017";
// Crie um MongoClient com um objeto MongoClientOptions para definir a versão
estável da API
const client = new MongoClient(uri, {
  serverApi: {
    version: ServerApiVersion.v1,
    strict: true,
    deprecationErrors: true,
  },
});

async function run() {
  try {
    await client.connect();
    const database = client.db("ifrs_db");
    const collection = database.collection("courses");

    // cria um documento a ser inserido
    const courseDocument = {
      name: "Análise e Desenvolvimento de Sistemas",
      type: "Superior",
    };

    const result = await collection.insertOne(courseDocument);
    console.log(`Documento inserido com o _id: ${result.insertedId}`);
```

```

    } finally {
      // Garante que o client fechará quando você terminar ou der erro
      await client.close();
    }
  }
}
run().catch(console.dir);

```

Observe que você pode imprimir o número do documento inserido acessando o campo `InsertedId` do resultado da operação (`result`).

```

Documento inserido com o _id: 654bdb6e8a37cc955ef81929
[nodemon] clean exit - waiting for changes before restart

```

Nós podemos especificar vários documentos a serem inseridos com a operação de gravação `insertMany()` usando uma matriz de objetos JSON da seguinte maneira:

```

const courseDocument = [
  { name: "Agronomia", type: "Superior" },
  { name: "Hospedagem", type: "Técnico" },
  { name: "Informática para Internet", type: "Técnico" },
];

```

Para inserir essa matriz de documentos do objeto `courseDocument`, usamos o método `insertMany()` conforme mostrado abaixo:

```

const options = { ordered: true };
const result = await collection.insertMany(courseDocument,
options);

```

Veja que `{ ordered: true }` impede que documentos adicionais sejam inseridos se um falhar.

Logo, altere o arquivo `index.js` para o seguinte código:

```

const { MongoClient, ServerApiVersion } = require("mongodb");

// Substitua a string uri pela string de conexão do MongoDB
const uri = "mongodb://127.0.0.1:27017";
// Crie um MongoClient com um objeto MongoClientOptions para definir a versão
estável da API
const client = new MongoClient(uri, {

```



```

serverApi: {
  version: ServerApiVersion.v1,
  strict: true,
  deprecationErrors: true,
},
});
async function run() {
  try {
    await client.connect();
    const database = client.db("ifrs_db");
    const collection = database.collection("courses");

    const courseDocument = [
      { name: "Agronomia", type: "Superior" },
      { name: "Hospedagem", type: "Técnico" },
      { name: "Informática para Internet", type: "Técnico" },
    ];
    const options = { ordered: true };
    const result = await collection.insertMany(courseDocument, options);
    console.log(`${result.insertedCount} documentos foram inseridos`);

  } finally {
    // Garante que o client fechará quando você terminar ou der erro
    await client.close();
  }
}
run().catch(console.dir);

```

Execute o aplicativo e veja o resultado no terminal:

```

3 documentos foram inseridos
[nodemon] clean exit - waiting for changes before restart

```

1.2.5 Retornando Documentos

Nós podemos usar operações de leitura para recuperar dados do banco de dados MongoDB. Existem vários tipos de operações de leitura que, por sua vez, acessam os dados de maneiras diferentes. Por exemplo, se quisermos solicitar resultados do nosso banco com base em um conjunto de critérios, podemos usar uma operação de localização, como os métodos `find()` ou `findOne()`.

Nós podemos consultar um único documento em uma coleção com o método `collection.findOne()`. O método `findOne()` permite você informar um documento

de consulta que deverá ser usado corresponder aos resultados de sua consulta. Se você não fornecer um documento de consulta ou se fornecer um documento vazio, o MongoDB corresponderá a todos os documentos da coleção. Cabe dizer que a operação `findOne()` retorna apenas o primeiro documento correspondente.

Assim, altere o arquivo `index.js` da seguinte forma:

```
const { MongoClient, ServerApiVersion } = require("mongodb");

// Substitua a string uri pela string de conexão do MongoDB
const uri = "mongodb://127.0.0.1:27017";
// Crie um MongoClient com um objeto MongoClientOptions para definir a versão
estável da API
const client = new MongoClient(uri, {
  serverApi: {
    version: ServerApiVersion.v1,
    strict: true,
    deprecationErrors: true,
  },
});

async function run() {
  try {
    await client.connect();
    const database = client.db("ifrs_db");
    const collection = database.collection("courses");

    // Pesquisa um curso com o nome de Agronomia
    const query = { name: "Agronomia" };
    const result = await collection.findOne(query);
    console.log(result);
  } finally {
    // Garante que o client fechará quando você terminar ou der erro
    await client.close();
  }
}

run().catch(console.dir);
```

Ao executar este aplicativo, teremos o seguinte resultado:

```
[nodemon] starting `node index.js`
{
  _id: new ObjectId('654bdbf3dca1ad576d9291ad'),
  name: 'Agronomia',
  type: 'Superior'
}
[nodemon] clean exit - waiting for changes before restart
█
```

Já o método `find()` aceita um documento de consulta, que descreve os documentos que você deseja recuperar. Para acessar os resultados, podemos, opcionalmente, passar um callback na chamada do método ou resolver o objeto Promise retornado.

Você também pode definir opções adicionais de consulta, como classificação por ordem alfabética e informar a projeção de quais campos devem ser retornados. Você pode especificá-los no parâmetro `options` em sua chamada de método `find()` em objetos `sort` e `projection`.

Ainda, o método `find()` retorna um `Cursor` que gerencia os resultados de sua consulta. Podemos iterar através do cursor usando métodos de cursor como `next()`, `toArray()` ou `forEach()` para buscar e trabalhar com os documentos retornados. Se nenhum documento corresponder à consulta, `find()` retorna um cursor vazio.

Assim, altere o arquivo `index.js` da seguinte forma

```
const { MongoClient, ServerApiVersion } = require("mongodb");

// Substitua a string uri pela string de conexão do MongoDB
const uri = "mongodb://127.0.0.1:27017";
// Crie um MongoClient com um objeto MongoClientOptions para definir a versão
estável da API
const client = new MongoClient(uri, {
  serverApi: {
    version: ServerApiVersion.v1,
    strict: true,
    deprecationErrors: true,
  },
});

async function run() {
  try {
    await client.connect();
    const database = client.db("ifrs_db");
    const collection = database.collection("courses");

    // Pesquisa os cursos superiores
```

```

const query = { type: "Superior" };
const options = {
  // ordena os documentos em ordem ascendente por nome (A->Z)
  sort: { name: 1 },
  // inclui apenas o campos name e type no documento retornado
  projection: { _id: 0, name: 1, type: 1 },
};

const cursor = collection.find(query, options);
for await (const doc of cursor) {
  console.log(doc);
}

} finally {
  // Garante que o client fechará quando você terminar ou der erro
  await client.close();
}
}
run().catch(console.dir);

```

Depois que a operação retorna, a variável `findResult` faz referência a um `Cursor`. Você pode imprimir os documentos recuperados usando o método `forEach()`.

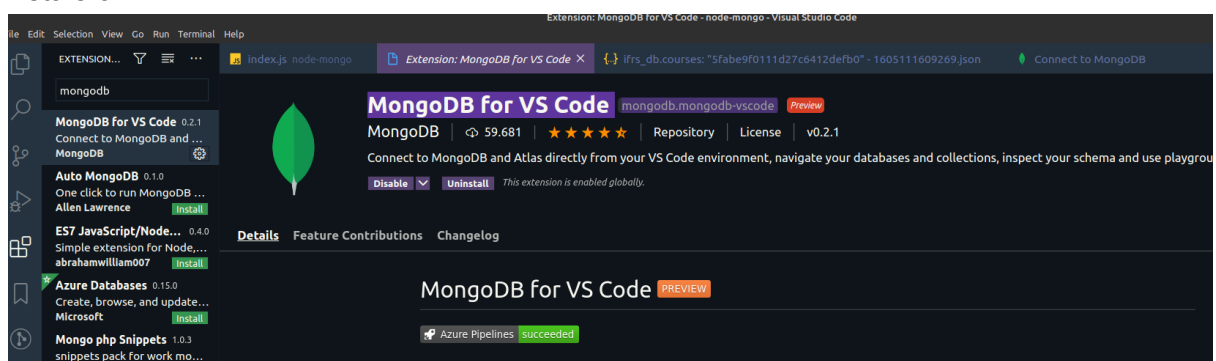
Execute o aplicativo e veja o resultado:

```

[nodemon] starting `node index.js`
{ name: 'Agronomia', type: 'Superior' }
{ name: 'Análise e Desenvolvimento de Sistemas', type: 'Superior' }

```

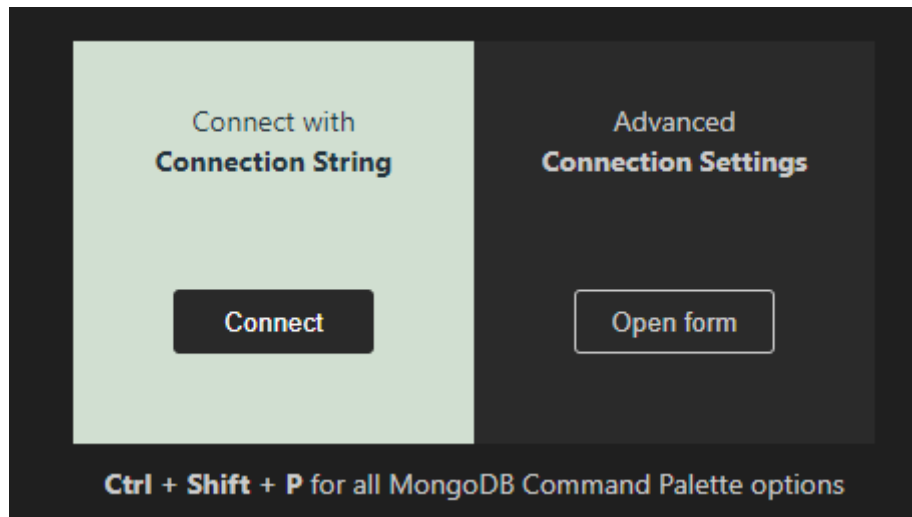
Outra forma de verificar estes documentos do MongoDB é instalar a extensão “MongoDB for VS Code” no Visual Studio Code. Clique no botão da direita referente às extensões do VS Code. Em seguida, pesquise por MongoDB. Clique na extensão “MongoDB for VS Code” e instale-a.



Em seguida, irá aparecer na sua barra lateral da esquerda o ícone em formato de “folha”.

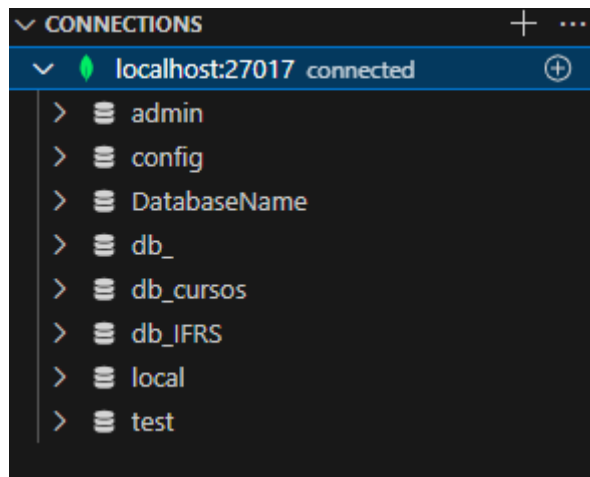


Clique neste ícone. Em “Connections”, clique no botão “Add Connection”.

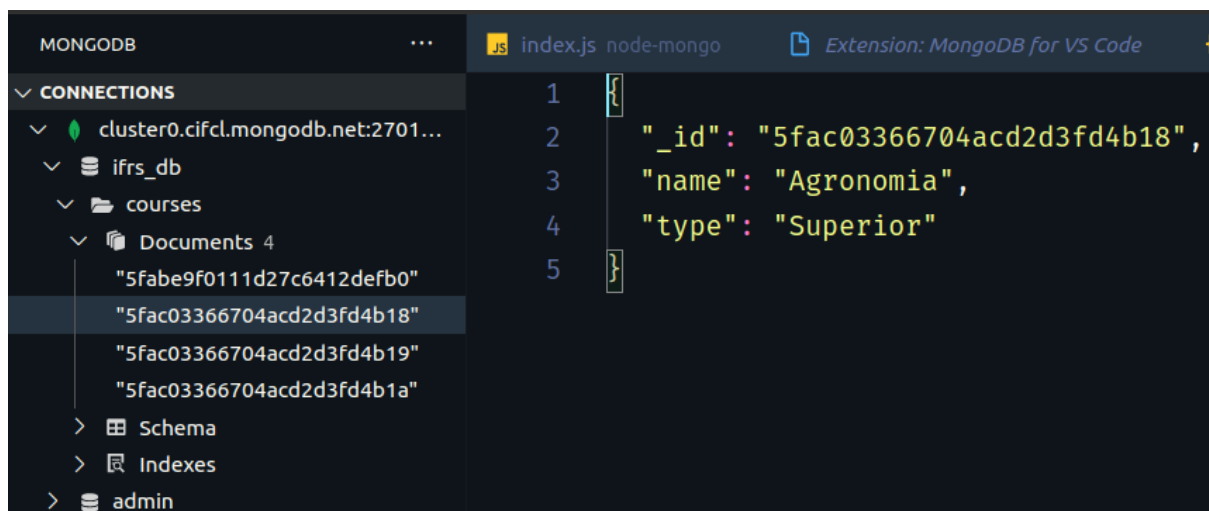


Como estamos com um servidor local, selecione Advanced Connection Settings . Vai abrir a seguinte janela:

Clique em “Connect”. Pronto, na janela Connections deve aparecer a sua conexão com o banco de dados MongoDB



Clique em um documento específico e veja o seu conteúdo.



1.2.6 Alterando Documentos

Nós podemos alterar documentos em uma coleção do MongoDB usando dois tipos de operações distintos: `update` e `replace`. A operação `update` altera os campos especificados em um ou mais documentos e deixam outros campos e valores inalterados. A operação `replace` remove todos os campos existentes em um ou mais documentos e os substituem por campos e valores especificados.

Altere o arquivo `index.js` da seguinte forma:

```
const { MongoClient, ServerApiVersion } = require("mongodb");

// Substitua a string uri pela string de conexão do MongoDB
const uri = "mongodb://127.0.0.1:27017";
// Crie um MongoClient com um objeto MongoClientOptions para definir a versão
estável da API
```

```

const client = new MongoClient(uri, {
  serverApi: {
    version: ServerApiVersion.v1,
    strict: true,
    deprecationErrors: true,
  },
});

async function run() {
  try {
    await client.connect();
    const database = client.db("ifrs_db");
    const collection = database.collection("courses");

    const filter = { name: "Análise e Desenvolvimento de Sistemas" };
    // atualiza o documento
    const updateDocument = {
      $set: { name: "ADS" },
    };
    const result = await collection.updateOne(filter, updateDocument);
    console.log("Atualizou!");
  } finally {
    // Garante que o client fechará quando você terminar ou der erro
    await client.close();
  }
}

run().catch(console.dir);

```

Veja em nosso banco de dados que esta informação foi modificada.

```

{
  "_id": "5fabe9f0111d27c6412defb0",
  "name": "ADS",
  "type": "Superior"
}

```

1.2.7 Excluindo Documentos

Se quisermos remover documentos existentes de uma coleção, podemos usar `deleteOne()` para remover um documento ou `deleteMany()` para um ou mais documentos. Esses métodos aceitam um documento de consulta que corresponda aos documentos que você deseja excluir.

Assim, altere o arquivo index.js da seguinte forma:

```
const { MongoClient, ServerApiVersion } = require("mongodb");

// Substitua a string uri pela string de conexão do MongoDB
const uri = "mongodb://127.0.0.1:27017";
// Crie um MongoClient com um objeto MongoClientOptions para definir a versão
estável da API
const client = new MongoClient(uri, {
  serverApi: {
    version: ServerApiVersion.v1,
    strict: true,
    deprecationErrors: true,
  },
});

async function run() {
  try {
    await client.connect();
    const database = client.db("ifrs_db");
    const collection = database.collection("courses");

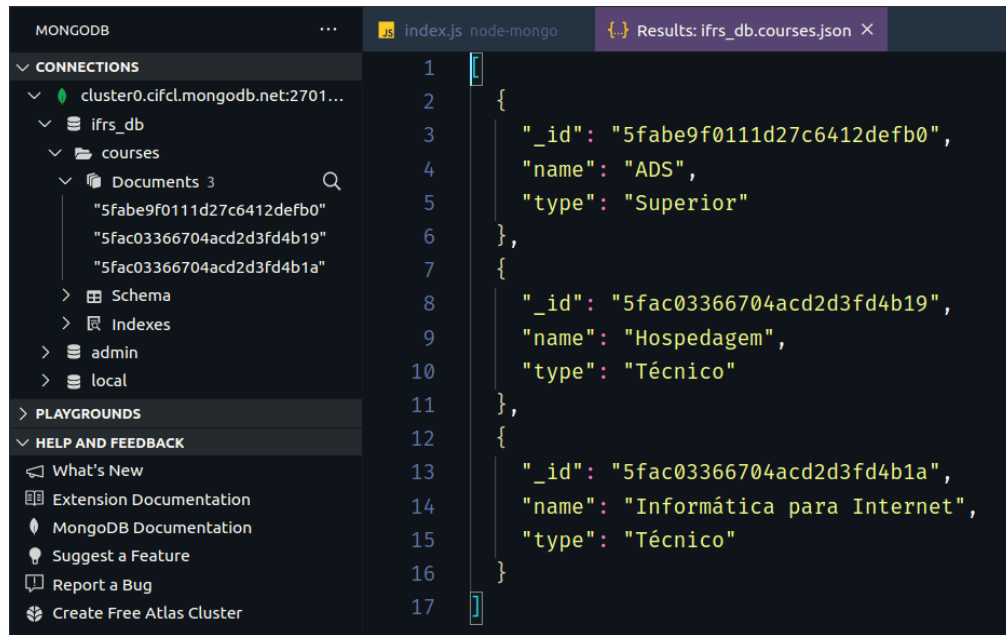
    const query = { name: 'Agronomia' };
    const result = await collection.deleteOne(query);
    if (result.deletedCount === 1) {
      console.dir("Exclusão realizada com sucesso.");
    } else {
      console.log("Não foi possível encontrar um documento");
    }
  } finally {
    // Garante que o client fechará quando você terminar ou der erro
    await client.close();
  }
}

run().catch(console.dir);
```

Ao executar este comando teremos o seguinte resultado:

```
[nodemon] starting `node index.js`
'Exclusão realizada com sucesso.'
[nodemon] clean exit - waiting for changes before restart
█
```


Veja o resultado em nossa base de dados:



```
1 [
2 {
3   "_id": "5fabe9f0111d27c6412defb0",
4   "name": "ADS",
5   "type": "Superior"
6 },
7 {
8   "_id": "5fac03366704acd2d3fd4b19",
9   "name": "Hospedagem",
10  "type": "Técnico"
11 },
12 {
13   "_id": "5fac03366704acd2d3fd4b1a",
14   "name": "Informática para Internet",
15   "type": "Técnico"
16 }
17 ]
```

Referências

- Site Oficial do MongoDB. Disponível em: <https://docs.mongodb.com/>
- Site do Mongo DB Atlas. Disponível em: <https://www.mongodb.com/cloud/atlas>