06

Criando e inserindo em uma coleção

Introdução

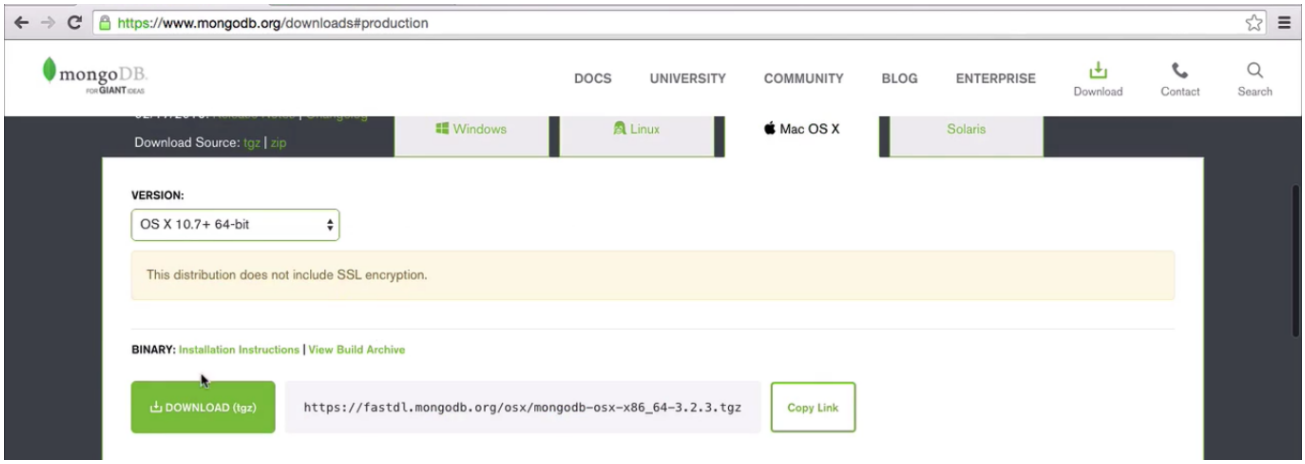
Bem-vindos a nosso curso de *MongoDB*. Ele foi criado pelo Felipe Oliveira, mas vai ser oferecido pelo Guilherme Silveira. Esse curso iniciará pela instalação do *MongoDB* e ao longo das aulas veremos a criação de dados de alunos de uma escola.

Aprenderemos a lidar com esses dados, inserindo, atualizando, removendo e etc. Assim como aprenderemos como os operadores do *MongoDB* aplicam-se nas diversas fases desse processo. Veremos um dos usos mais utilizados do *MongoDB*, que é a busca por proximidade. Veremos também os *features* de banco e ao final se espera que aprendamos a entender como ele funciona e como ele influência nosso trabalho, a observar e ver se um ou outro banco faz mais ou menos sentido em determinados projetos.

Veremos todos esses aspectos buscando fugir da questão de digitar mais ou menos. Ou seja, esquivaremos da competição de digitação. muitas vezes presente no universo da programação. O que busca-se nesse curso é deverás **compreender** essa ferramenta, como ela pode ser usada, como funciona e quais são as vantagens e desvantagens que ela traz.

Criando e inserindo em uma coleção

O primeiro passo antes de começarmos é a instalação do **MongoDB**. Vamos acessar o site www.mongodb.org (http://www.mongodb.org) e logo na primeira página encontramos a opção *download Mongo*. Ao clicarmos nessa opção somos redirecionados a uma nova página onde escolheremos a opção de download conforme o sistema operacional que estivermos utilizando. Aparecem as opções *Mac, Linux, Windows* e etc. Ao clicar em qualquer uma delas podemos ler as suas respectivas instruções para a instalação.



Como nós selecionaremos a opção do *Mac*, teremos que descompactar um arquivo. Repare que quando clicamos na opção do *Mac* ele nos fornece instruções passo a passo de instalação.

O arquivo será baixado. Terminado o *download* iremos descompactar o arquivo. Se ele estiver, por exemplo, na área de transferência, podemos mudá-lo de lugar. Ao abrirmos a pasta do "mongodb" teremos dentro uma pasta do diretório "bin". Dentro desta pasta temos um arquivo para o servidor, o "mongod", e um arquivo para o cliente, o "mongo".

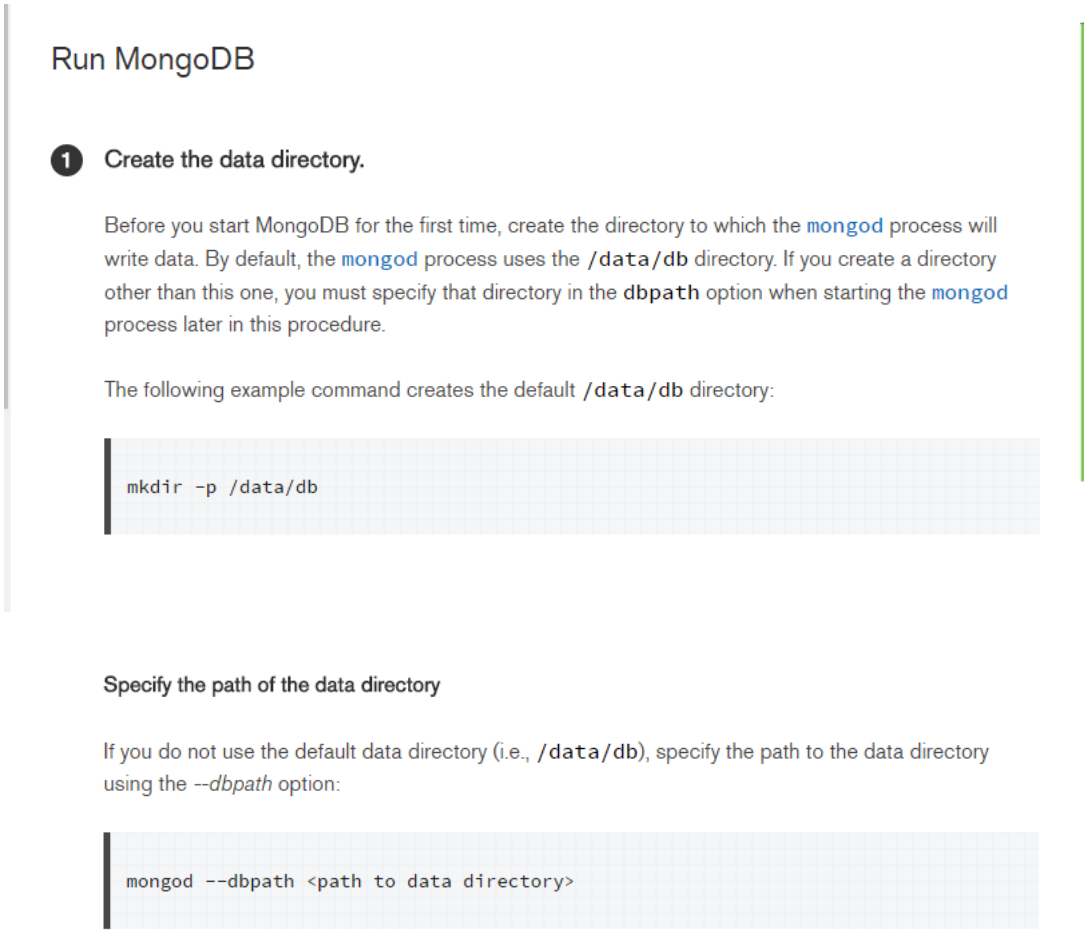
Podemos rodar esses arquivos no Terminal, mas teremos um problema. Se digitarmos:

```
cd Documents/guilherne/cursomongodb/mongodb-osx-x86_64-3.3.3/bin/
```

E tentarmos, dentro disso, executar os arquivos "mongo" e "mongodb", não conseguiremos concluir essa tarefa. Quando descompactamos o "MongoDB", por padrão no *Linux* e no *Mac*, ele fica tentando acessar o diretório `/data/db` que não existe.

Uma opção para resolver esse problema é criar o diretório `/data/db` e dar permissão de acesso ao usuário. A segunda opção é executar o `./mongod` falando para ele que queremos colocar os arquivos do banco de dados em um diretório diferente.

Para isso, digitaremos `./mongod` e passaremos um parâmetro para poder executar o `mongod` com um `PATH` para o banco de dados um pouco diferente. Se voltarmos na documentação, nas instruções de instalação que constam no site, ele nos falará isso:



Run MongoDB

1 Create the data directory.

Before you start MongoDB for the first time, create the directory to which the mongod process will write data. By default, the mongod process uses the /data/db directory. If you create a directory other than this one, you must specify that directory in the dbpath option when starting the mongod process later in this procedure.

The following example command creates the default /data/db directory:

```
mkdir -p /data/db
```

Specify the path of the data directory

If you do not use the default data directory (i.e., /data/db), specify the path to the data directory using the *--dbpath* option:

```
mongod --dbpath <path to data directory>
```

Ele, conforme aparece na primeira imagem, nos avisa que quando tentarmos rodar ele buscará o diretório `/data/db` ou, conforme mostra a segunda imagem, podemos passar o parâmetro `PATH` do banco de dados. Seguiremos a segunda opção! Vamos, primeiramente, criar um diretório para que possamos ver os arquivos separados.

Vamos digitar o seguinte:

```
> ./mongod --dbpath /Users/alura/Documents/guilherme/cursomongodb/db
```

Com esse diretório que criamos o que estamos dizendo é que para esse projeto atual estamos utilizando esse diretório. Temos, agora, um banco de dados para esse projeto:

```
ers/alura/Documents/guilherme/cursomongodb/db" } }
```

Repare que ele nos diz que inicializou o banco de dados dentro do diretório e se formos em nosso navegador veremos que dentro da pasta "db" temos o banco de dados criado.

Agora, com o servidor rodando podemos ir em outra aba e entrar no `bin` para então executar o `mongodb` , entretanto, fazer esse caminho toda vez é um pouco desgastante, pois temos que entrar em diversos diretórios antes de chegarmos ao `bin` .

O que podemos fazer é, ao estarmos no diretório raiz, alterar a variável de ambiente, `PATH` (no Windows) ou editar com `vi` o arquivo `.bash_profile` (no Linux ou no Mac).

Editaremos isso digitando:

```
> vi bash_profile
```

Agora, temos um arquivo editado, ele está em branco. Teclaremos a letra "a" para adicionar em conteúdo novo. Vamos dizer que queremos exportar o `PATH` como `PATH` atual e o diretório do "bin", digitando o seguinte:

```
> export PATH=$PATH:/Users/alura/Documents/guilherme/cursomongodb/mongodb-osx-x86_64-3.2.3/b
```

Podemos apertar "Esc", "w", "q" e ":" para sair. Damos um "Enter" e ele salva o `.bash_profile` .

Agora, podemos voltar a primeira aba, nela deixaremos rodando o banco de dados, com o `db` do `PATH` e na aba nova vamos executar o `mongo` e ele já pega isso do `PATH` . Ficamos com uma terceira aba sobrando e podemos apagá-la.

Repare, estamos rodando um servidor, o *MongoD*, que é algo, por exemplo, como o *Oracle*, o *SQL* e etc. E estamos executando um cliente que é do *Oracle, SQL* e etc , mas que está conectado a nosso banco.

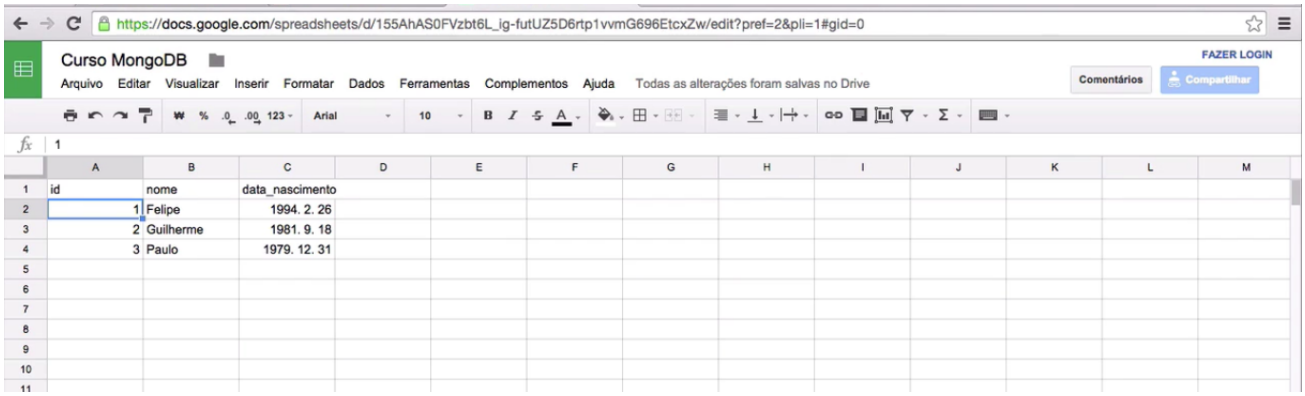
Criando e inserindo em uma coleção

O que faremos agora é trabalhar com o MongoDb como um banco de dados.

O primeiro passo é olharmos uma tabela. Estamos observando uma tabela das ferramentas do Google. Como representamos em um banco de dados, relacional, um conjunto de alunos?

Vamos pensar que essa tabela terá um campo "nome", nele teremos três alunos, "Felipe", "Guilherme" e "Paulo", um campo "data_nascimento" e três datas de nascimento e, por fim, um campo "id", com o identificador desses alunos. É

comum que o "id" sejam números sequenciais, que na tabela tenhamos tudo escrito em minúsculo e que cada linha represente um aluno distinto. Observe:



E como isso funciona no *MongoDB*? Podemos verificar que se fizermos isso no Editor teremos que criar toda a estrutura da tabela, assim, quando inserirmos um novo aluno com seus respectivos dados ele saberá o que é "nome", "idade" e "data de nascimento" pois a estrutura já está pronta. Teremos o seguinte:

```
CREATE TABLE alunos (id INTEGER AUTO_INCREMENT PRIMARY KEY, nome VARCHAR (255), data_nascimento
INSERT INTO alunos  VALUES (1, "FELIPE", "19940226")
```

Vamos ao Terminal. Nele a primeira coisa que gostaríamos que fosse feita é a criação de uma coleção de alunos. Mas, no **MongoDB** não trabalhamos com tabelas com estruturas fixas. O que faremos é falar para o banco de dados, o `db` , como se escrevêssemos um código de **javascript**, para criar uma "Coleção" chamada alunos, digitando

```
db.createCollection("alunos")
```

Após criar a Coleção diremos para ele inserir um aluno. Digitaremos `db.alunos.insert` , com isso, chamamos um *insert* e colocaremos dentro disso um objeto de *javascript*. Abaixo do `db.alunos.insert` vamos colocar entre chaves, o "nome" e o campo "data_nascimento".

Tivemos que escrever os campo "nome" e "data_nascimento" pois, em nenhum momento a estrutura foi mencionada. Agora, como queremos inserir coisas, devemos criar uma estrutura, que segue o padrão de um objeto do *javascript*. Esse objeto possui um "nome" e um valor.

Bom, então, podemos criar objetos. Por exemplo, no **MongoDB** temos o objeto `new DATE` . Assim, temos um objeto `new DATE` que desejamos instanciar e ele é o que queremos inserir. Vamos escrevê-lo abaixo do `db.alunos.insert` :

Teremos o seguinte:

```
> db.alunos.insert(
    {
        "nome" : "Felipe",
        "data_nascimento" : new Date (1994,02,26)
    }
    )
WriteResult ({ "nInserted : 1})
```

Agora que ele inseriu essa informação, gostaríamos que fosse inserido, também, o curso em que o aluno está matrículado na faculdade. Como estamos trabalhando com objetos de *javascript*, podemos ter a descrição de um curso

em nosso Editor. Então, pode ser que o curso seja um novo objeto e nele teremos o nome desse curso. Nesse caso, diremos que Felipe faz "Sistema de Informação".

Podemos inserir esse objeto "curso" e ir muito além. Podemos colocar, por exemplo, as notas de "Felipe" e também informações extras sobre a vida desse aluno, por exemplo, uma habilidade e o nível de desempenho dela.

Repare que não houve preocupação em relação ao acento na chave. E vamos inserir o nome e o nível da habilidade "Taekwondo" e "inglês".

```
{
    "nome" : "Felipe",
    "data_nascimento" : new Date (1994,02,26),
    "curso" : {
        "nome" : "Sistemas de informação"
    },
    "notas" : [10.0, 9.0, 4.5],
    "habilidades": [
        {
            "nome" : "inglês",
            "nível" : "avançado"
        },
        {
            "nome" : "taekwondo",
            "nível" : "básico"
        }
    ]
}
```

Agora, como qualquer objeto de *javascript* podemos copiar isso do Editor e colamos no Terminal abaixo de `db.alunos.insert` :

```
> db.alunos.insert({
    "nome": "Felipe",
    "data_nascimento": new Date(1994, 02, 26),
    "curso": {
        "nome": "Sistemas de informação"
    },
    "notas": [10.0, 9.0, 4.5],
    "habilidades": [{
        "nome": "inglês",
        "nível": "avançado"
    }, {
        "nome": "taekwondo",
        "nível": "básico"
    }]
})
WriteResult({ "nInserted" : 1 })
```

Isso será inserido para nós. Colocamos um documento que possui diversos dados a respeito de nosso aluno e fizemos tudo isso de uma única vez, através de um único *insert*.

Se quisermos conferir os alunos que acabamos de inserir, basta digitarmos `db.alunos.find()` e com isso, é trazido para nós dois objetos:

```
db.alunos.find()
{ "_id" : ObjectId ("56cb0002b6d75ec12f75d3b5"), "nome" : "Felipe", "data_nascimento" : ISO Date
{ "_id" : ObjectId ("56cb0139b6d75ec12f75d3b5"), "nome" : "Felipe", "data_nascimento" : ISO Date
```

O primeiro é um aluno que possui um `ID` e `data_nascimento` e o segundo que possui um `ID` e diversos outros dados.

Vamos aprender como inserir, remover e manipular esses dados na sequência!

Removendo elementos e inserindo mais alunos

Nesse momento, trataremos sobre inserir e remover alunos em nosso banco e abordaremos, também, sobre objetos de *javascript*. Primeiro, vamos remover o objeto que está duplicado. Para remoção, utilizamos o `remove` acompanhado do `Id` , entre chaves, daquilo que desejamos remover. A confirmação do `Id` pode ser feita através do `db.find.alunos()` . Teremos o seguinte:

```
db.alunos.remove({
    "_id" : ObjectId ("56cb0002b6d75ec12f75d3b5")
})
```

Com isso, estamos dizendo que queremos remover o objeto com o `Id` especificado. Agora, se dermos um `db.alunos.find()` teremos um único objeto:

```
db.alunos.find()
{ "_id" : ObjectId ("56cb0139b6d75ec12f75d3b5"), "nome" : "Felipe", "data_nascimento" : ISO Date
```

Vamos incluir mais alunos, no caso, vamos adicionar um aluno que se chamará "Júlio". Para isso, preencheremos os dados dele no Editor:

```
{"nome" : "Julio",
"data_nascimento" : new Date(1972, 08, 30),
"curso" : {
    "nome" : "Medicina"
},
"habilidades" : [
        {
        "nome" : "inglês",
        "nível" : "avançado"
        }
    ]
}
```

Agora, vamos adicionar copiar tudo isso e colar no Terminal. Basta digitarmos `db.alunos.insert` e logo abaixo colar, entre parênteses, o que acabamos de escrever no Editor. Dando um "Enter" o nosso aluno estará inserido. Observe:

```
> db.alunos.insert(
{"nome" : "Julio",
"data_nascimento" : new Date(1972, 08, 30),
```

```
"curso" : {
    "nome" : "Medicina"
},
"habilidades" : [
        {
        "nome" : "inglês",
        "nível" : "avançado"
        }
    ]
}
)
WriteResult({ "nInserted" : 1 })
```

Para adicionar outro aluno, por exemplo, o "Alberto", o procedimento é o mesmo. No editor, abrimos chaves e preenchemos o nome de um aluno e suas informações, escrevendo tudo entre aspas. Mas, atenção, no *javascript* o uso de aspas é opcional. Se estivermos utilizando o modelo `ASCII` podem ser retiradas as aspas dos nomes, data_nascimento, curso, habilidades e nível. Se copiarmos e colarmos as informações de "Alberto" no Terminal e dermos um `db.alunos.find()` , encontraremos ele. Agora, temos três alunos, "Felipe", "Julio" e "Alberto".

Mesmo que tenhamos conseguido inserir esses alunos sem utilizar as aspas, na prática, se usam esses caracteres, justamente, para que o sistema possa lê-los e não cometa erros.

Vamos inserir um quarto aluno, ele se chamará "Daniela". Para adicionar essa aluna reproduzimos o mesmo processo. Preenchemos as informações no Editor, copiamos isso e colamos no Terminal abaixo de `db.alunos.insert` , entre parenteses, e dando um "Enter", teremos o aluno inserido. Se dermos um `db.alunos.find()` teremos os quatro aluno.

O uso das aspas, portanto, é opcional! Se não utilizarmos, isso não influenciará em nada.

```
{
    nome : "Daniela",
    data_nascimento : new Date(1995,09,30),
    curso : {
        nome : "Moda"
    },
    habilidades : [
    {
        nome: "alemão",
        nível: "básico"
    }
    ]
}
```

Na prática, a **dica** é: Evite carácteres que podem dar erro e use as aspas!

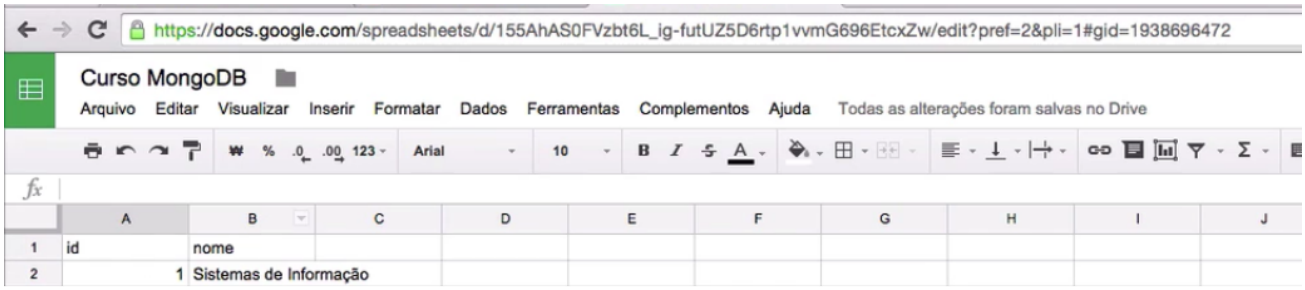
- Limite-se a carácteres simples, por exemplo, letras e o *underline*. Pessoalmente, inclusive, nós tendemos a evitar os acentos.O padrão não é evitar, mas é uma precaução. O molde é, usar letras minúsculas e *underline* para diferenciar duas palavras.
- É importante utilizarmos as aspas, elas são também um meio de precaução. Podemos ter um carácter inválido, mas usando ele entre aspas, podemos utilizá-lo.

A ideia foi passar como inserir diversos alunos, falar sobre objetos do *javascript* e algumas das regras específicas para isso. Vimos também, como remover objetos que foram inseridos.

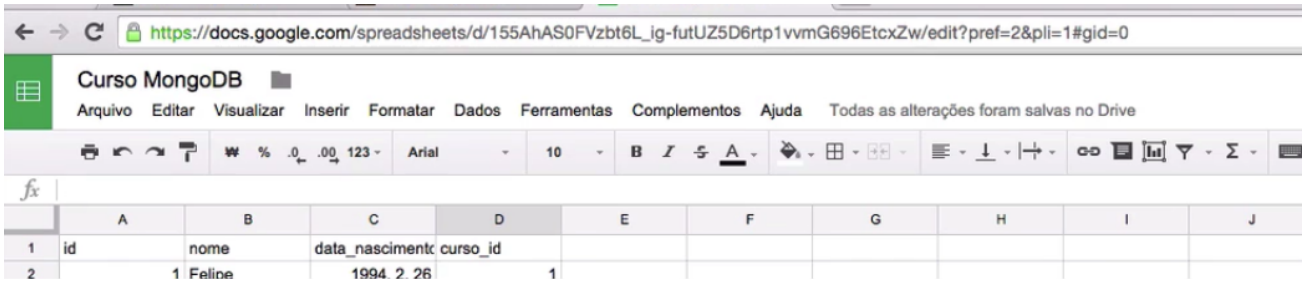
Como ser injusto e justo ao comparar o Mongodb e o SQL

Já aprendemos a inserir novos nome em uma Coleção, agora, vamos aprender como podemos inserir alunos em um banco relacional. Veremos que a enfâse de usar-se uma ou outra opção não deve ser, necessariamente, o quanto digitamos, a motivação de optar-se por uma ou outra tecnologia deve ser sua performance.

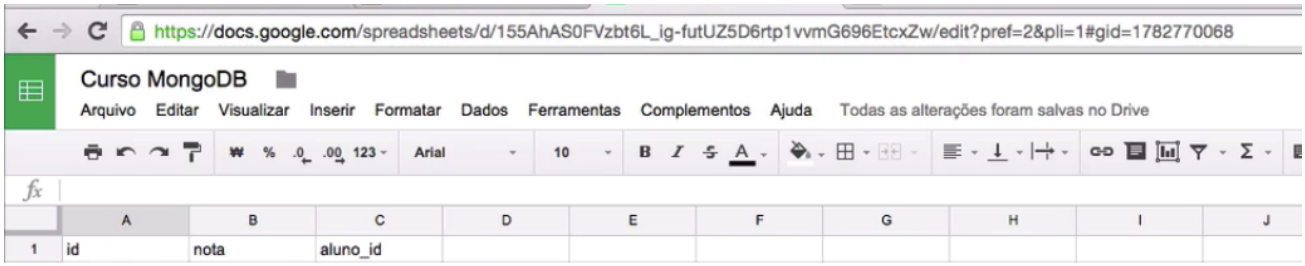
Nesse instante o objetivo é mostrar como podemos representar os dados e informações dos alunos em uma tabela de dados relacional. Usaremos a tabela das ferramentas do Google. Já tínhamos criado uma tabela de alunos, vamos criar, também, uma tabela com os cursos que os alunos estão fazendo. Nessa tabela teremos uma coluna `Id` e o "Nome dos cursos". Observe:

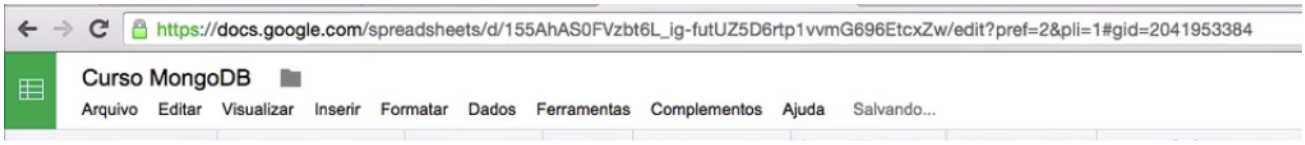


Um aluno só poderá estar em um único curso, ele nunca estará em dois cursos. Esta é a maneira como modelaremos nossa tabela, porém existem diversas possibilidades de fazer isso. A modelagem que utilizaremos, portanto, limita que um aluno pode cursar apenas um curso. Vamos acrescentar uma outra coluna o `curso_id`, que estará na planilha dos nomes dos alunos. O `curso_id` serve para relacionarmos uma tabela com a outra.



Precisamos, ainda, de uma terceira tabela na qual constará as notas dos alunos. Nela criaremos as colunas `nota_id`, `aluno_id` e `id`. Podemos criar,também, a quarta tabela, a "Habilidades" e nela teremos as seguintes colunas: "nome", "nível", "id" e "aluno_id". Observe:





Repare que criamos quatro tabelas e relacionamentos entre `Id`. Temos um banco que está seguindo regras razoavelmente normatizadas, de banco de dados, de SQL. Importante lembrar que essa não é a única maneira de modelar é apenas uma das maneiras. Para criar um estrutura precisamos de quatro *CREATE TABLE*, ou seja, para inserir um aluno com todos os seus dados, temos que dar `insert` em todas as quatro tabelas.

O nível de normatização, mais generalizando ou mais específico, influencia o número final de tabelas. Isso, por sua vez, também influenciará os *inserts*.

Importante!

Uma decisão arquitetural não pode ser feita devido a quantidade de coisas que temos que digitar. Ela deve ser uma escolha inteligente, reconhecendo as vantagens e desvantagens em termos de tecnologia.

Retornando, a ideia foi demonstrar a "Coleção de alunos" em um formato de quatro tabelas. Voltando ao Editor, podemos recriar essas tabelas utilizando o `CREATE TABLE`. Para isso, digitaremos `CREATE TABLE`, o nome da tabela e dentro de parênteses o que estiver dentro de cada uma das tabelas. Por exemplo, sempre teremos distintos itens em cada uma das tabelas, se tivermos um `id` devemos, além disso, preencher seu nome e sua classificação: `id INTEGER PRIMARY KEY`. Como mostrado abaixo:

```
CREATE TABLE cursos (id INTEGER PRIMARY KEY AUTO_INCREMENT, nome VARCHAR(255));
CREATE TABLE alunos (id INTEGER PRIMARY KEY AUTO_INCREMENT nome VARCHAR (255), curso_id INTEGER
CREATE TABLE notas (id INTEGER PRIMARY KEY AUTO_INCREMENT, nota DEIMAL (3,2), aluno_id INTEGER)
CREATE TABLE notas (id INTEGER PRIMARY KEY AUTO_INCREMENT, nome VARCHAR(255), nivel VARCHAR(255
```

Com isso, criamos a coleção que no **MongoDB** criamos utilizando apenas um comando, o `db.createCollection`.

Repare que usar o `db.createCollection("alunos")` é uma coisa e usar o `CREATE TABLE` é totalmente distinto. O conteúdo que a `CREATE TABLE` suporta é exatamente aquilo que está escrito e isso o distingue do `db.createCollection("alunos")`, pois ele não suporta garantias da maneira como elas estão escritas. Assim, usar os `CREATE TABLE` ou o `db.createCollection("alunos")` não é uma questão de um ser melhor que o outro, só que depende da situação que estamos e do que queremos. Se você quer ter a certeza de que as regras serão seguidas, usaremos o `CREATE TABLE`, que valida certas coisas que o `db.createCollection("alunos")` não valida.

A possibilidade de escolha é, na verdade, uma das vantagem que o MongoDB possui. Um dos pontos positivos é, justamente, essa flexibilidade e liberdade de estrutura. Algo que um banco de dados relacional não fornece. A tomada de decisão, portanto, não pode se pautar na maior ou menor quantidade de código escrito. A escolha não pode ser feita baseada na quantidade de palavras digitadas e sim, por vantagens. A escolha deve ser feita reconhecendo nossas necessidades, se é preciso uma estrutura mais flexível usaremos o `db.createCollection("alunos")`, se é mais fixa utilizaremos o `CREATE TABLE`. Cada uma dessas estruturas possuem vantagens próprias. Veremos essas vantagens ao longo do curso.

Vamos observar, empiricamente, as vantagens dessas estruturas. Vamos entrar no site www.ebay.com (http://www.ebay.com). Se olharmos os calçados e clicarmos em um modelo de sapato, teremos informações sobre as características desse sapato, tamanho, material e etc. Se mudarmos o produto, por exemplo, para um *Mac*, teremos outros materiais, tamanhos e informações. Assim, temos algumas informações que sempre vamos querer que apareçam, mais fixas, e outras que sejam mais flexíveis. Exemplo, não faria sentido aparecer o material do *Mac*, pois, está não é uma informação tão importante quanto, por exemplo, memória.

O uso dessas estruturas, mais fixas ou flexíveis, portanto, variam conforme o que queremos.