

BANCOS DE DADOS II

Cap.2 - Bancos de Dados Orientados a Documentos



Prof. Me. Renzo P. Mesquita

Objetivos

- Compreendermos características de BDs NoSQL Orientados a Documentos;
- Trabalhar este conceito por meio do BD MongoDB e sua ferramenta para visualização, busca e manipulação de dados: o MongoDB Compass;
- Aprender recursos fundamentais da linguagem de programação Python para conectarmos aplicações feitas nesta linguagem a BDs NoSQL;



Capítulo 2

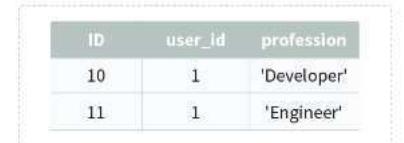
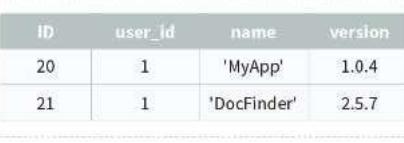
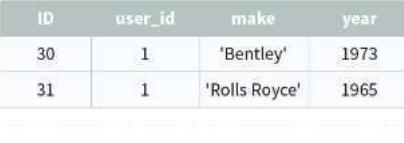
Bancos de Dados Orientados a Documentos

- 2.1. Bancos Orientados a Documentos;**
- 2.2. Introdução ao MongoDB;**
- 2.3. MongoDB Query Language (MQL);**
 - 2.3.1. Buscas Simples;**
 - 2.3.2. Buscas em Documentos Aninhados;**
 - 2.3.3. Buscas em Arrays;**
- 2.4. A linguagem de programação Python;**
- 2.5. Conexão Python + MongoDB;**
 - 2.5.1. Configurando uma Conexão;**
 - 2.5.2. Buscando Documentos;**
 - 2.5.3. Inserindo Documentos;**
 - 2.5.4. Atualizando Documentos;**
 - 2.5.5. Removendo Documentos.**

2.1. Bancos Orientados a Documentos

Diferente dos BDs Relacionais, BDs Orientados a Documentos possuem como filosofia conter todas as informações importantes sobre algo em um único documento, ao invés de quebrá-las em parte.

Exemplo:

Relacional								Documentos			
ID	first_name	last_name	cell	city	year_of_birth	location_x	location_y				
1	'Mary'	'Jones'	'516-555-2048'	'Long Island'	1986	'-73.9876'	'40.7574'				
											
											
											
											
											
"full_name"	: "Rahul",										
"age"	: 24,										
"designation"	: "Software Developer",										
"city"	: "Hyderabad",										
"state"	: "AndhraPradesh"										
}											

2.1. Bancos Orientados a Documentos

Um ponto muito importante nestes BDs é identificar quais campos a serem armazenados podem exigir múltiplos valores. Estes campos serão candidatos a serem documentos mais internos (Embedded Sub-documents) ou Lista de Valores.

Exemplo:

```
{  
    first_name: "Mary",  
    last_name: "Jones",  
    cell: "516-555-2048",  
    city: "Long Island",  
    year_of_birth: 1986,  
    location: {  
        type: "Point",  
        coordinates: [-73.9876, 40.7574] > Embedded sub-document  
    },  
    profession: ["Developer", "Engineer"], > lista  
    apps: [  
        { name: "MyApp", > Embedded sub-document  
            version: 1.0.4 },  
        { name: "DocFinder", > Embedded sub-document  
            version: 2.5.7 }  
    ],  
    cars: [  
        { make: "Bentley", > Embedded sub-document  
            year: 1973 },  
        { make: "Rolls Royce", > Embedded sub-document  
            year: 1965 }  
    ]  
}
```

Veja que esta abordagem não seria possível em um BD SQL, pois lá seria necessário utilizar do conceito de relacionamentos para se fazer algo parecido.

2.1. Bancos Orientados a Documentos

O Esquema (Schema) de um documento é flexível e autodescritivo. Não é necessário defini-lo antes de usá-lo, ou seja, campos podem variar de documento para documento e podem ter suas estruturas modificadas a qualquer momento.

Exemplo:

```
{officeName:"3Pillar Noida",
{Street: "B-25, City:"Noida", State:"UP", Pincode:"201301"}
}
{officeName:"3Pillar Timisoara",
{Boulevard:"Coriolan Brediceanu No. 10", Block:"B, Ist Floor", City: "Timisoara", Pinc
}
{officeName:"3Pillar Cluj",
{Latitude:"40.748328", Longitude:"-73.985560"}
}
```

Isso acaba abrindo portas para os chamados **dados Semiestruturados** (ou seja, dados que não possuem um esquema fixo a seguir, mas que possuem tags que ajudam a armazenar dados importantes).

É importante ressaltar que, apesar da definição de um esquema fixo não ser um procedimento obrigatório geralmente em BDs NoSQL, alguns oferecem este recurso a fim de deixar os dados mais estruturados, como acontecem nos BDs Relacionais.

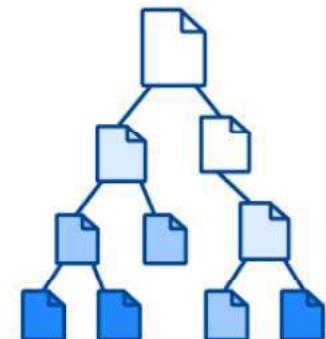
2.1. Bancos Orientados a Documentos

Exercício Proposto

Agora que já possui uma noção de como um Documento é estruturado, crie um Documento na sua ferramenta de texto predileta para guardar informações pessoais de uma pessoa da sua escolha.

Sugestão de alguns campos e abordagens que o Documento possa usar:

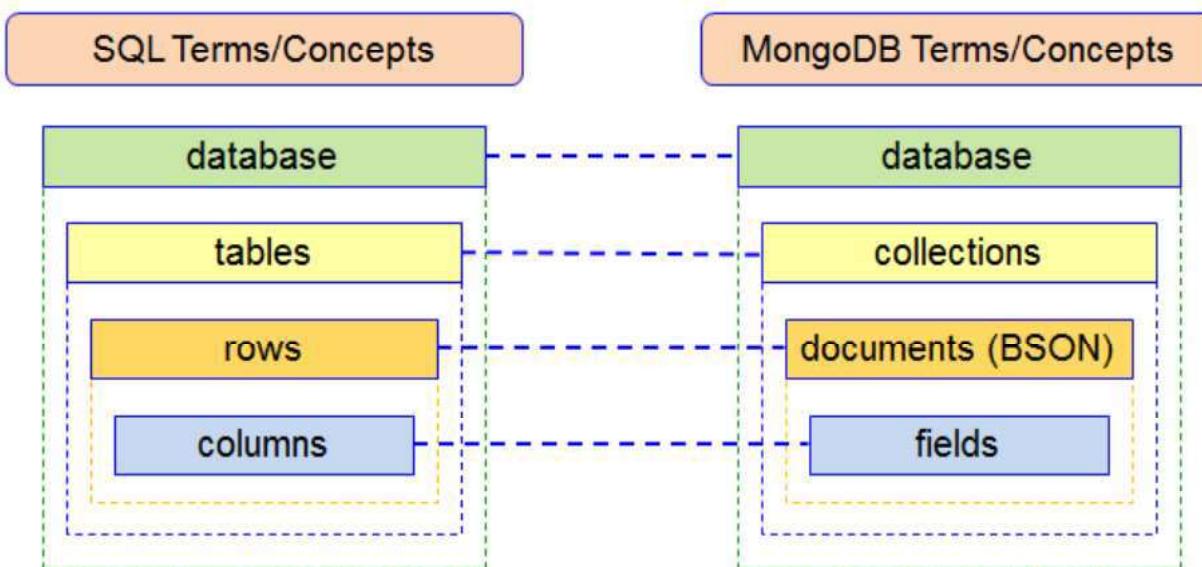
- nome;
- idade;
- profissão;
- parentescos (como um documento mais interno que mostre em detalhes o tipo e nome do parentesco);
- usar de listas para guardar o nome de múltiplos parentescos do mesmo tipo;



2.2. Introdução ao MongoDB

Gratuito e de código aberto, ele utiliza do modelo baseado em Documentos, e consequentemente, de todas as características que vimos deste tipo de BD.

Apesar de não ser um BD Relacional (SQL), suas estruturas possuem algumas equivalências, são elas:



2.2. Introdução ao MongoDB

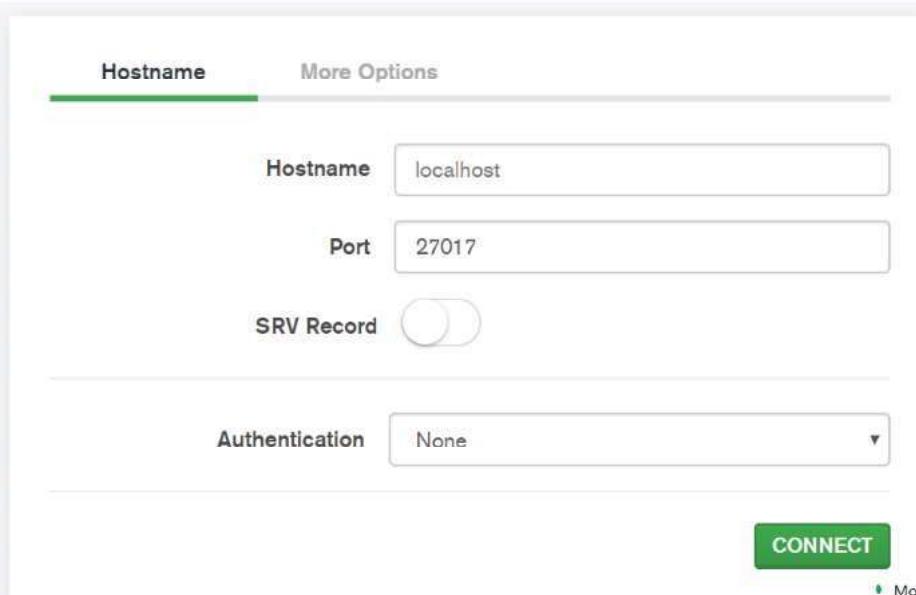
A forma mais fácil de explorar e manipular os dados do MongoDB é por meio do client oficial distribuido pela empresa, neste caso, o MongoDB Compass.

- Permite visualizar, compreender e trabalhar com os dados de forma gráfica e intuitiva;
- Insira, valide e modifique dados do banco com facilidade;
- Permite compreender questões voltadas à performance do sistema de forma visual;
- Possui grande compatibilidade com outros plugins para deixar a ferramenta mais completa quando necessário;
- Permite o acesso a bases de dados locais ou remotas;

Vamos explorá-lo mais em detalhes?



2.2. Introdução ao MongoDB



Uma vez conectado (conexão localhost e clicando em CONNECT), será possível visualizar as bases de dados já criadas ou criar uma nova com o botão "CREATE DATABASE".

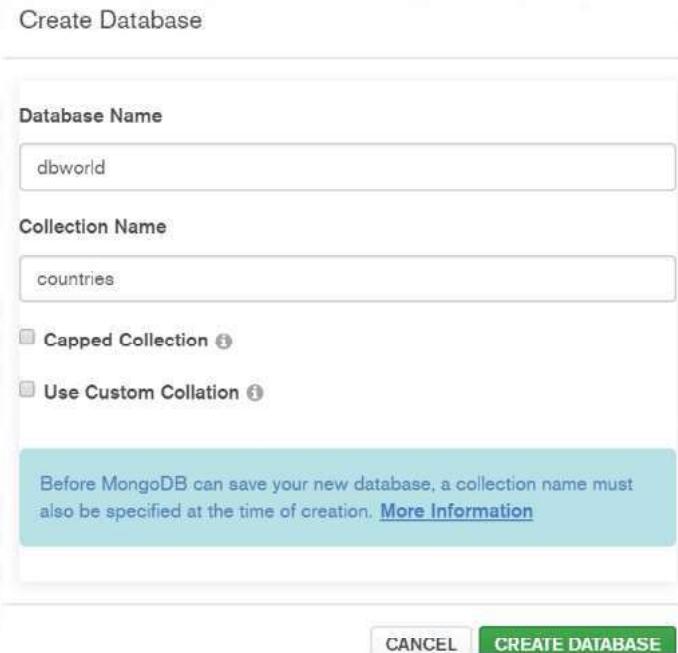
Ao executar o MongoDB Compass, é possível configurar uma conexão local ou remota.

The screenshot shows the MongoDB Compass interface after connecting to 'localhost:27017'. The left sidebar shows 'Local' with '3 DBS' and '1 COLLECTIONS'. The right panel has tabs for 'Databases' (selected) and 'Performance'. Under 'Databases', there is a 'CREATE DATABASE' button and a table:

Database Name	Storage Size
admin	32.0KB
config	20.0KB
local	36.0KB

2.2. Introdução ao MongoDB

Para compreendermos melhor como manipular um Banco de Dados por meio do MongoDB Compass, vamos fazer um exemplo juntos.



- No canto inferior esquerdo, vá em "+" -> "CREATE DATABASE";
- Insira como nome do Banco de Dados "dbworld" e para a primeira coleção "countries".

1. Capped Collection -> cria coleções de tamanhos finitos. Funciona como um buffer circular, que quando o tamanho de uma coleção acaba, dados mais antigos começam a ser sobreescritos.
2. Use Custom Collation -> permite configurar o Banco para melhor trabalhar com casos muito específicos, como caracteres de uma linguagem específica.

2.2. Introdução ao MongoDB

Alguns tipos de dados populares para criação de campos em Documentos do MongoDB:

- **String:** tipo de dado mais largamente utilizado. Guarda palavras formadas por caracteres em UTF-8;
- **Integer:** usado para guardar valores numéricos de 32 ou 64 bits;
- **Boolean:** usada para guardar valores true ou false;
- **Double:** usado para guardar valores de ponto flutuante (64 bits);
- **Date:** usado para guardar data e hora no formato: YYYY-MM-DDThh:mm:ss.000+00:00
- **Arrays:** usado para armazenar uma lista de valores;
- **Object:** usado para armazenar documentos aninhados.



Para maiores detalhes, acesse: <https://docs.mongodb.com/manual/reference/bson-types/>

2.2. Introdução ao MongoDB

Ao clicar sobre o database que criamos, e em seguida sobre a sua única coleção, vamos adicionar alguns perfis diferentes de documentos.

1. Utilizando apenas tipos de dados comuns;

```
_id: ObjectId("600ace06934831e65eb7c686") ObjectId
name : "Brazil //"
area : 8516000 Int32
capitalist : true Boolean
register : 2020-09-08T00:16:21.000+00:00 Date
```

```
_id: ObjectId("600acfde934831e65eb7c687") ObjectId
name : "Panama //"
population : 4250000 Int32
currencies : Array
  0: "PAB //"
  1: "USD //"
String
String
```

2.Utilizando tipos de dados comuns e Arrays;

3. Utilizando documentos aninhados.

```
_id: ObjectId("600ad25b934831e65eb7c688") ObjectId
name : "Canada //"
languages : Object String
  EN : 57 Int32
  FR : 22 Int32
  currency : "CAD //"
Object
```

2.3. MongoDB Query Language (MQL)

Bancos de Dados Relacionais utilizam do SQL como linguagem para estruturação, manipulação e consulta de dados. O MongoDB usa do MongoDB Query Language (MQL).

The screenshot shows the Compass MongoDB interface for the dbworld.countries collection. The top navigation bar displays the database and collection names. Below the navigation are tabs for 'Documents', 'Aggregations', 'Schema', 'Explain Plan', 'Indexes', and 'Validation'. The 'Documents' tab is currently active. Key statistics are shown: DOCUMENTS 0, TOTAL SIZE 0B, AVG. SIZE 0B, INDEXES 1, TOTAL SIZE 36.0KB, and AVG. SIZE 36.0KB. A large search bar below the tabs contains buttons for 'FILTER', 'PROJECT', 'SORT', and 'COLLATION', along with 'OPTIONS', 'FIND', 'RESET', and a three-dot menu. To the right of the search bar are buttons for 'MAX TIME MS' (set to 60000), 'SKIP' (0), and 'LIMIT' (0). At the bottom of the interface are buttons for 'ADD DATA', 'VIEW', and various refresh and search controls.

No Compass, ao selecionar um database e uma coleção, a tela para realização de CRUDs irá se abrir.

- FILTER -> permite filtrar documentos baseando em regras específicas (como se fosse o WHERE);
- PROJECT -> permite especificar quais campos de um documento o usuário tem interesse de ver (funcionando como se fosse o SELECT);
- SORT -> permite que uma consulta seja ordenada se baseando em um critério (como se fosse o ORDER BY);
- COLLATION -> especifica regras para se trabalhar com palavras de uma linguagem específica;
- ADD DATA -> permite criar um novo documento ou inserir um conjunto deles;
- RESET -> limpa os critérios usados em uma busca anterior.

2.3. MongoDB Query Language (MQL)

Com a opção ADD DATA, conseguimos de forma prática e rápida realizarmos o UPLOAD de um arquivo JSON dentro do MongoDB.

- Exclua os registros que criamos anteriormente dentro da coleção countries;
- Faça upload do arquivo **countries.json** fornecido.

ADD DATA -> Import File

em VIEW conseguimos visualizar os documentos de diferentes formas.

_id	ObjectId	altSpellings	Array	area	Int32	borders	Array	callingCodes	Actions
1	55a0f42f20a4d760b5fc305e	[]	1 elements	91		[]	0 elements	[]	
2	55a0f42f20a4d760b5fc305f	[]	3 elements	468		[]	2 elements	[]	
3	55a0f42f20a4d760b5fc3060	[]	3 elements	1246700		[]	4 elements	[]	
4	55a0f42f20a4d760b5fc3061	[]	3 elements	83600		[]	2 elements	[]	
5	55a0f42f20a4d760b5fc3062	[]	4 elements	1580		[]	0 elements	[]	
6	55a0f42f20a4d760b5fc3063	[]	2 elements	652230		[]	6 elements	[]	
7	55a0f42f20a4d760b5fc3064	[]	1 elements	14000000		[]	0 elements	[]	
8	55a0f42f20a4d760b5fc3065	[]	1 elements	7692024		[]	0 elements	[]	
9	55a0f42f20a4d760b5fc3066	[]	2 elements	7747		[]	0 elements	[]	
10	55a0f42f20a4d760b5fc3067	[]	4 elements	199		[]	0 elements	[]	

2.3. MongoDB Query Language (MQL)

2.3.1. BUSCAS SIMPLES

- Selecionando todos os documentos:

Ex: FILTER -> {}

- Filtrando com igualdade:

Padrão: { <field1>: <value1>, ... }

Ex: FILTER -> {capital: "Brasília"}

- Filtrando com operadores:

Padrão: { <field1>: { <operator1>: <value1> }, ... }

Ex: FILTER -> {region: {\$in: ["Americas", "Oceania"]}}

Mas quais os operadores mais populares do MongoDB?

2.3. MongoDB Query Language (MQL)

Para realização de Filtros, o MongoDB oferece uma série de operadores. Os mais populares são:

- **\$in** : busca valores que são iguais aos valores especificados dentro de [];
- **\$gt** : busca valores que são maiores a um valor especificado;
- **\$gte** : busca valores que são maiores ou iguais a um valor especificado;
- **\$lt** : busca valores que são menores a um valor especificado;
- **\$lte** : busca valores que são menores ou iguais a um valor especificado;
- **\$ne** : busca valores que são diferentes a um valor especificado;
- **\$and** : realiza a lógica 'e' em buscas;
- **\$not** : realiza a lógica 'não' em buscas;
- **\$or** : realiza a lógica 'ou' em buscas;
- **\$exists** : busca os documentos que possuem um campo específico;

Para mais operadores, acesse:

<https://docs.mongodb.com/manual/reference/operator/query/>

2.3. MongoDB Query Language (MQL)

- Filtrando com operadores e realizando Projeção:

Ex: FILTER -> {region: "Europe", area:{\$gt:300000}}
PROJECT -> {name: 1}

ps: observe que a , (vírgula) realiza a lógica AND nas condições do FILTER.

- Filtrando com operadores, realizando Projeção e Ordenação:

Ex: FILTER -> {subregion: "South America", landlocked: true}
PROJECT -> {name:1,capital:1,area:1}
SORT -> {capital:-1}

ps: observe que o -1 no SORT indica a ordem decrescente de ordenação.

Para mais opções de buscas simples, acesse:

<https://docs.mongodb.com/manual/tutorial/query-documents/>

2.3. MongoDB Query Language (MQL)

2.3.2. BUSCAS EM DOCUMENTOS ANINHADOS

- Selecionando em documentos aninhados:

Ex: FILTER -> {"name.common":"Austria"}

PROJECT -> {"name.common":1,"name.official":1}

ps: observe a notação de . (ponto) para acesso a campos de documentos aninhados.

- Selecionando em documentos aninhados e com operadores:

Padrão: { <field1>: { <operator1>: <value1> }, ... }

Ex: FILTER -> {"languages.eng":{\$exists:true}}

Ex: FILTER -> {\$expr:{\$eq:["\$name.common", "\$name.official"]}}

Para mais opções de buscas em documentos aninhados, acesse:

<https://docs.mongodb.com/manual/tutorial/query-embedded-documents/>

Para mais opções de buscas com Aggregation Expressions, acesse:

<https://docs.mongodb.com/manual/reference/operator/query/expr/>

2.3. MongoDB Query Language (MQL)

2.3.3. BUSCAS EM ARRAYS

- Selecionando em Arrays:

Ex: FILTER -> {currency:"USD"}

ps: aqui buscaremos todos os documentos que possuam o valor como um de seus elementos.

- Selecionando em Arrays com operadores:

Ex: FILTER -> {borders:{\$all:["BRA","ARG"]}}

ps: aqui buscaremos todos os documentos que possuam pelo menos os seguintes valores como seus elementos.

Ex: FILTER -> {borders:{\$size: 3, \$all:["RUS"]}}

ps: aqui buscaremos todos os documentos que possuam somente 3 elementos dentro do array especificado, seguido de outra condição.

Para mais opções de buscas com Arrays, acesse:

<https://docs.mongodb.com/manual/tutorial/query-arrays/>

2.3. MongoDB Query Language (MQL)

2.3.4. BUSCAS POR PADRÕES TEXTUAIS

No MongoDB, Expressões Regulares (REGEX) podem ser usadas para buscas de padrões textuais. Isso pode ser feito por meio do uso do operador \$regex.

Padrão: { <field>: { \$regex: /pattern/<options> } }

Ex: FILTER -> {"name.common":{\$regex: /un/i}}

ps: aqui buscaremos todos os países que possuem em seu nome os caracteres "un" juntos. O "i" depois da / é uma opção que significa case insensitive (não diferencia maiúsculas de minúsculas).

Ex: FILTER -> {"name.common":{\$regex: /^U.*s\$/}}

ps: neste exemplo, cada uma das partes significa:

- ^ -> força a String começar com algo (neste caso, "U");
- .* -> padrão que indica qualquer caracter e em qualquer quantidade;
- \$ -> força a String terminar com algo (neste caso, "s");

Para mais opções de buscas com REGEX no MongoDB, acesse:

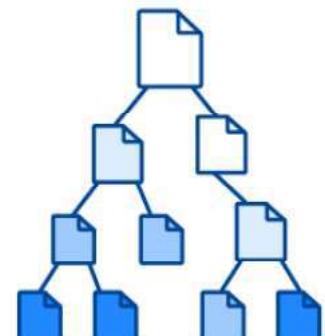
<https://docs.mongodb.com/manual/reference/operator/query/regex/>

2.3. MongoDB Query Language (MQL)

Exercício Proposto

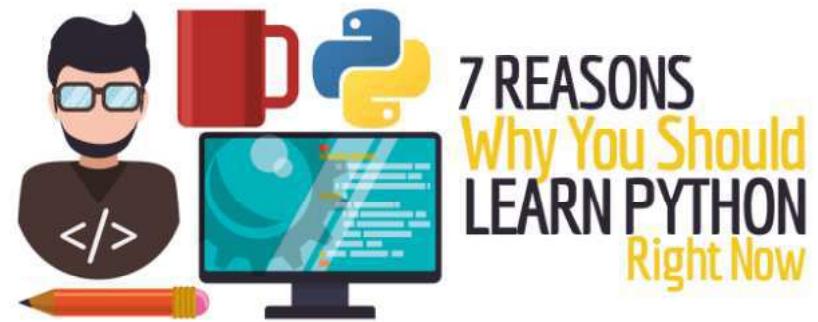
Baseado no que vimos até o momento de MQL, crie queries que respondam às seguintes questões:

1. Mostre apenas o nome oficial dos países que são da região "Americas" e possuem área menor que 100;
2. Mostre o nome comum, a latitude e longitude apenas dos países que falam português (por);
3. Mostre apenas o nome comum dos países que começam com "B" e ordene os resultados por ordem crescente de area;



2.4. A Linguagem de Programação Python

Com tantas linguagens de programação disponíveis no mercado, por que usar Python?



- É uma **linguagem de propósito geral**, ou seja, você pode usar Python para criar qualquer tipo de programa;
- É considerada uma das **linguagens mais fáceis e intuitivas** já criadas (comandos muito simples);
- É uma linguagem **multiplataforma** (um mesmo programa roda em diferentes tipos de sistemas operacionais e dispositivos);
- Vem com um **conjunto de bibliotecas** muito completa;
- É um **software livre** (não é necessário pagar para usá-lo ou distribuí-lo);
- É **MUITO organizada!** Ela força o programador a deixar o código organizado;
- É **ORIENTADA A OBJETOS**, ou seja, faz uso do paradigma de programação mais popular do mercado;

2.4. A Linguagem de Programação Python

Uma IDE (Integrated Development Environment) de desenvolvimento nada mais é que uma ferramenta que facilita a vida do desenvolvedor na hora de utilizar uma linguagem de programação.

- Nesta disciplina, utilizaremos o Python 3.x e o PyCharm, uma IDE de desenvolvimento da empresa JetBrains, para desenvolver e continuar nossos estudos com BDs NoSQL;
- Dentre as várias vantagens oferecidas por essa IDE, pode-se destacar:
 - Editor intuitivo e de fácil manuseio;
 - Função auto-complete de códigos;
 - Auto identação;
 - Vários atalhos para diferentes funções;
 - Possibilidade de adicionar e organizar plugins de forma simplificada;
 - Geração de código automático, e muito mais ;).



2.4. A Linguagem de Programação Python

Para criar um novo projeto no PyCharm é muito simples: depois de abri-lo, vá em File -> New Project.

- Vale ressaltar que um projeto pode ter uma série de arquivos Python dentro dele. Futuramente vamos compreender melhor a vantagem de criar vários arquivos dentro de um mesmo projeto;
- Para criar pelo menos um arquivo Python no seu projeto e começar a programar, clique com o Botão direito em cima do seu projeto -> New -> Python File;
- O PyCharm oferece muitas facilidades e recursos. Caso queira customizar a IDE pra ficar mais 'a sua cara' (como deixar a letra maior, etc.), vá em File -> Settings;
- Para executar um programa em um arquivo específico, clique com o Botão direito sobre seu código -> Run (setinha verde);



PyCharm

2.4. A Linguagem de Programação Python

Todo comando em Python 3 é uma Função!

- Função nada mais é que uma espécie de comando que é escrito dentro de um bloco que começa com (e termina com);
- Para escrevermos algo na tela, usamos a função print, que deve ter seu conteúdo colocado dentro do (). Caso o conteúdo seja uma mensagem, então ela também deve estar circundada com aspas simples ou duplas;

Ex: `print ("Olá Python!")`

- Caso o conteúdo seja um número ou uma operação matemática, não é necessário colocar as aspas, mas sim o número ou a operação em si dentro do parêntesis;

Ex: `print (7 + 7)`

- Caso queira colocar mensagens juntamente com números ou operações de forma conjunta, basta utilizar da , (vírgula);

Ex: `print('O resultado de 7 + 7 é', 7+7)`

2.4. A Linguagem de Programação Python

Variáveis são espaços de memória capazes de armazenar informações de diferentes tipos.

Para criar uma variável em Python basta escrever o nome dela, seguido do operador = (recebe) e do valor que deseja armazenar:

```
Ex: nome = 'Vincent'  
     idade = 30  
     peso = 83.5  
     print(nome,idade,peso)
```

Importante:

Observe que no Python para se criar uma variável não há necessidade de colocar o seu tipo. Python é uma linguagem dinamicamente tipada.

Os tipos primitivos mais básicos utilizados no Python são:

- **int**: usado para representar números inteiros (Ex: 7, -5, 0, 999);
- **float**: usado para representar números reais (Ex: 4.5, 0.75, -15.5);
- **bool**: só aceita dois valores (Ex: True e False);
- **str**: usado para representar palavras (Ex: 'Oi', 'Python é legal');

2.4. A Linguagem de Programação Python

• Estruturas de Decisão

Blocos de comandos que podem mudar o fluxo de execução de um software dependendo do resultado de uma variável ou de uma condição, como por exemplo, o if-else (se-senão).

Ex:

```
1     idade = int(input('Entre com sua idade: '))
2
3     if idade < 18:
4         print('Você é menor de idade!')
5     else:
6         print('Você é maior de idade!')
```

• Laços de Repetição

Um loop (ou laço de repetição) permite executar o mesmo bloco de código enquanto uma condição é atendida, como por exemplo, o for (para).

Ex:

```
1     for c in range(0, 10):
2         print('Python é legal! :)')
```

2.4. A Linguagem de Programação Python

Os programas em Python inicialmente operam com um conjunto básico de recursos, porém, caso haja necessidade, podemos IMPORTAR novas bibliotecas ou módulos a fim de adicionarmos mais funcionalidades.

Existem duas formas de importar bibliotecas externas no Python, são elas:

- `import novaBiblioteca` -> importa uma biblioteca inteira de recursos;
- `from novaBiblioteca import parteDaNovaBiblioteca` -> importa apenas um recurso específico de uma biblioteca;

Neste Módulo usaremos da biblioteca `pymongo` para permitir criarmos aplicações em Python capazes de se comunicarem com o Banco de Dados MongoDB.



Uma vez dentro do PyCharm, para baixarmos esta biblioteca, basta seguirmos os seguintes passos:

`File -> Settings -> Project: NomeDoSeuProjeto -> Project Interpreter -> +`

Feito, isso, basta procurar por `pymongo` e clicar em "Install Package"

Espere a instalação e pronto! agora ela já pode ser usada no projeto.

2.5. Conexão Python + MongoDB

2.5.1. CONFIGURANDO UMA CONEXÃO

Para se conectar o Python com o MongoDB localmente, devemos possuir, no mínimo, as seguintes configurações:

1.  # Importando o módulo MongoClient do pymongo
from pymongo import MongoClient

2. # Abrindo e guardando uma conexão com o MongoDB server
client = MongoClient('mongodb://localhost:27017')

3. # Criando/Guardando o acesso ao meu Banco de Dados
db = client['dbworld']

4. # Criando/Guardando o acesso a uma colecao chamada countries
paises = db.countries

2.5. Conexão Python + MongoDB

2.5.2. BUSCANDO DOCUMENTOS

Para se buscar todos documentos de um database, um exemplo seria:

```
# Buscando Documentos
# Buscando tudo
results = paises.find()

# Mostrando os resultados da busca
for aux in results:
    pprint.pprint(aux)
```

Dica: importe a biblioteca pprint para poder mostrar os resultados de forma mais adequada.

Vamos ver alguns exemplos de buscas com filtros igual aprendemos mas agora dentro do Python?

2.5. Conexão Python + MongoDB

2.5.2. BUSCANDO DOCUMENTOS

Ex: `results = paises.find({"region": {"$in": ["Americas", "Oceania"]}})`

Ex: `results = paises.find(
 {"region": "Europe", "area": {"$gt": 300000}},
 {"name": 1}
)`

Ex: `results = paises.find(
 {"subregion": "South America", "landlocked": True},
 {"name": 1, "capital": 1, "area": 1}
).sort("capital", -1)`

Observe que já tivemos a oportunidade de executar estes exemplos anteriormente, mas agora estamos executando diretamente de dentro do método `find()` do `pymongo`, que executa seus comandos obedecendo os parâmetros `find(FILTER, PROJECT)`. Para se ordenar, usa-se o método `sort()` após o `find()`.

Para mais opções de buscas de dentro do Python, acesse:

<https://docs.mongodb.com/manual/reference/method/db.collection.find/>

2.5. Conexão Python + MongoDB

2.5.3. INSERINDO DOCUMENTOS

Se uma coleção ainda não existe, então o método `insert_one()` a cria automaticamente, senão, simplesmente adiciona um novo documento na coleção.

Ex:

```
newdoc = {  
    "name": "Brazil",  
    "temp": {  
        "SP": 26,  
        "RJ": 32,  
        "MG": 26  
    }  
}
```

```
result = db.temperatures.insert_one(newdoc)  
  
if result.acknowledged:  
    print('Documento adicionado!')  
else:  
    print('Erro ao inserir Documento!')
```

Criando um novo Documento

Adicionando o novo Documento e
verificando se deu certo.

Se o novo documento não possui um campo chamado `_id`, o MongoDB adicionará um valor automático a ele. E claro, se o campo `_id` for preenchido, ele deve ser único para cada documento.

Para inserção de múltiplos documentos ao mesmo tempo, acesse:

<https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/>

2.5. Conexão Python + MongoDB

2.5.4. ATUALIZANDO DOCUMENTOS

Da mesma forma que acontece na inserção de documentos, a atualização possui os métodos `update_one()` e `update_many()`.

Ex:

```
# Acessando a Colecao no BD
temperaturas = db.temperatures
# Realizando uma atualizacao
result = temperaturas.update_one(
    {"temp.MG": {"$exists": True}},
    {"$set": {'temp.MG': 30}}
)
```

Atualizando um Documento

```
if result.acknowledged:
    print('Documento atualizado!')
else:
    print('Erro ao atualizar Documento!')
```

Verificando se deu certo

Para atualização de múltiplos documentos ao mesmo tempo, acesse:

<https://docs.mongodb.com/manual/reference/method/db.collection.updateMany/>



2.5. Conexão Python + MongoDB

2.5.5. REMOVENDO DOCUMENTOS

Da mesma forma que acontece na inserção de documentos, a remoção possui os métodos `delete_one()` e `delete_many()`.

Ex:

```
# Acessando a Colecao no BD
temperaturas = db.temperatures
# Realizando a remocao
result = temperaturas.delete_one({"name": "Brazil"})
```

Removendo um único Documento

OU

```
# Acessando a Colecao no BD
temperaturas = db.temperatures
# Realizando a remocao
result = temperaturas.delete_many({"name": "Brazil"})
```

```
if result.acknowledged:
    print('Documento removido!')
else:
    print('Erro ao remover Documento!')
```

Verificando se deu certo

Removendo um conjunto de Documentos que satisfaça a condição.

<https://docs.mongodb.com/manual/crud/#delete-operations>

**FIM
DO
CAPÍTULO 2**



EXERCÍCIOS