

MongoDB

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Divisão de Computação
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande

Agenda I

- 1 MongoDB
 - Introdução
 - Instalação
 - Principais Comandos
 - Query Conditions

- 2 mongodb-driver-sync
 - Instalação
 - Como Usar?
 - Como Usar - POJO?

Agenda

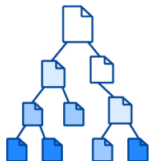
- 1 MongoDB
 - Introdução
 - Instalação
 - Principais Comandos
 - Query Conditions

- 2 mongodb-driver-sync

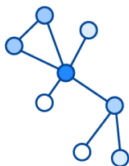
Introdução

- MongoDB é um banco de dados NoSQL baseado em documentos

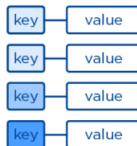
Document



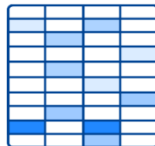
Graph



Key-Value

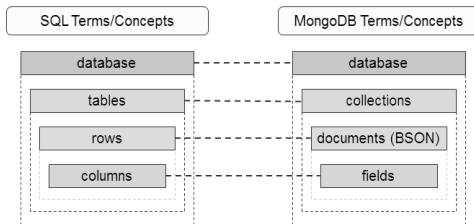


Wide-column



Introdução


Diferentemente dos BD's relacionais, o MongoDB trabalha com:
Coleções/Collections e **Documentos/Documents**



- Cada documento pode ter uma estrutura diferente mesmo estando em uma mesma coleção
- Por outro lado, é possível definir o que cada documento pode ter mediante a criação de Schemas

Introdução

Embora documentos não pode ser vistos como tabelas, colunas e linhas, eles podem ser vistos no formato JSON (mais especificamente BSON - *Binary JSON*):



```
1  person = {  
2    _id: "jo",  
3    name: "Jo Bloggs",  
4    age: 34,  
5    address: {  
6      street: "123 Fake St",  
7      city: "Faketon",  
8      state: "MA",  
9      zip: "12345"  
10   }  
11   books: [ 27464, 747854, ...]  
12 }
```

Instalação

Instalação - Ubuntu/Linux Mint/PopOS!:

- <https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/>

Docker

```
-- instalando o docker
sudo apt install docker.io
-- baixando o mongodb
sudo docker run --name mongodb -p 27017:27017 -d mongo:latest
-- iniciando o client
sudo docker exec -it mongodb mongo
```

Tutorial - MongoDB com Docker - Youtube (Código Fonte):

- <https://youtu.be/4dTI1mVLX3I?t=239>

Alternativa de *client* gráfico:

- <https://robomongo.org/>

Principais Comandos

```
-- Criando/Selecionar um BD
use minhaBase
-- retornar collection users
1) db.users
2) db.getCollection("users")
-- inserir
db.users.insert({"name": "codigo"}) -- legacy
db.users.insertOne({"name": "codigo"}) -- current
-- listar todos
db.users.find()
-- Update
db.users.update({"name": "codigo"}, {"name": "novo"}) -- legacy
db.users.updateOne({"name": "codigo"}, {"name": "novo"}) -- current
-- Delete
db.users.remove({"name": "codigo"}) -- legacy
db.users.deleteOne({"name": "codigo"}) -- current
```


Query Conditions

Da *collection inventory* os documentos com status **IGUAL** a D:

```
db.inventory.find({status:"D"})
```

Da *collection inventory* os documentos com status **IGUAL** a A **OU** D:

```
db.inventory.find({status:{$in:["A", "D"]}})
```

Da *collection inventory* os documentos com status **IGUAL** a A E **qty** é **MENOR QUE** 30:

```
db.inventory.find({status:"A", qty:{$lt:30}})
```

Da *collection inventory* os documentos com status **IGUAL** a A **OU** **qty** é **MENOR QUE** 30:

```
db.inventory.find({$or:[{status: "A"}, {qty:{$lt:30}}]})
```

Agenda

1 MongoDB

2 mongodb-driver-sync

- Instalação
- Como Usar?
- Como Usar - POJO?

Instalação

Instalação via Maven

```
<dependencies>
  <dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>4.6.1</version>
  </dependency>
</dependencies>
```

Como usar?

- **Quick-Reference:**

- <https://www.mongodb.com/docs/drivers/java/sync/current/quick-reference/>

- **Find:**

- <https://www.mongodb.com/docs/drivers/java/sync/current/usage-examples/find/>

- **Insert:**

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

- **POJO:**

- <https://www.mongodb.com/developer/languages/java/java-mapping-pojos/>

Como usar?

```
1 import static com.mongodb.client.model.Filters.eq;
2 import org.bson.Document;
3 import com.mongodb.client.MongoClient;
4 import com.mongodb.client.MongoClients;
5 import com.mongodb.client.MongoCollection;
6 import com.mongodb.client.MongoDatabase;
7 public static void main( String[] args ) {
8     // Replace the uri string with your MongoDB deployment's connection string
9     String uri = "<connection string uri>";
10    try (MongoClient mongoClient = MongoClients.create(uri)) {
11        MongoDatabase database = mongoClient.getDatabase("sample_mflix");
12        MongoCollection<Document> collection = database.getCollection("movies");
13        Document doc = collection.find(eq("title", "Back to the Future")).first();
14        System.out.println(doc.toJson());
15    }
16 }
```

```
String uri = "mongodb://localhost:27017";
```

```

1 public class MappingPOJO {
2     public static void main(String[] args) {
3         ConnectionString connectionString = new ConnectionString(System.getProperty("mongodb.uri"));
4         CodecRegistry.pojoCodecRegistry = fromProviders(PojoCodecProvider.builder().automatic(true).build());
5         CodecRegistry codecRegistry = fromRegistries(MongoClientSettings.getDefaultCodecRegistry(),.pojoCodecRegistry);
6         MongoClientSettings clientSettings = MongoClientSettings.builder()
7             .applyConnectionString(connectionString)
8             .codecRegistry(codecRegistry)
9             .build();
10        try (MongoClient mongoClient = MongoClient.create(clientSettings)) {
11            MongoDB db = mongoClient.getDatabase("sample_training");
12            MongoCollection<Grade> grades = db.getCollection("grades", Grade.class);
13
14            // create a new grade.
15            Grade newGrade = new Grade().setStudent_id(10003d)
16                .setClass_id(10d)
17                .setScores(new ArrayList<Score>() {
18                    add(new Score().setType("homework").setScore(50d));
19                });
20            grades.insertOne(newGrade);
21
22            // find this grade.
23            Grade grade = grades.find(eq("student_id", 10003d)).first();
24            System.out.println("Grade found:\t" + grade);
25
26            // update this grade: adding an exam grade
27            List<Score> newScores = new ArrayList<>(grade.getScores());
28            newScores.add(new Score().setType("exam").setScore(42d));
29            grade.setScores(newScores);
30            Document filterByGradeId = new Document("_id", grade.getId());
31            FindOneAndReplaceOptions returnDocAfterReplace = new FindOneAndReplaceOptions().returnDocument(ReturnDocument.AFTER);
32            Grade updatedGrade = grades.findOneAndReplace(filterByGradeId, grade, returnDocAfterReplace);
33            System.out.println("Grade replaced:\t" + updatedGrade);
34
35            // delete this grade
36            System.out.println(grades.deleteOne(filterByGradeId));
37        }
38    }
39 }

```

Figura: POJO - Plain Old Java Object

Como Usar - POJO?



```
1 public class Grade {  
2     private ObjectId id;  
3     @BsonProperty(value = "student_id")  
4     private Double studentId;  
5     @BsonProperty(value = "class_id")  
6     private Double classId;  
7     private List<Score> scores;  
8 }
```

- *@BsonProperty* foi utilizado para evitar violação da convenção de nomes do Java para variáveis, *getters* e *setters*.
- Isto permite indicar que o campo do BSON **student_id** seja mapeado pelo atributo **studentId** da classe Java.

MongoDB

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Divisão de Computação
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande