Table of Contents

Learn CRUD Java With MongoDB

Introdução ao MongoDB com Java

Bem-vindo a este tutorial prático onde você aprenderá a realizar operações CRUD (Create, Read, Update, Delete) no MongoDB usando o driver Java. Ao final, você será capaz de construir aplicações Java que interagem com bancos de dados MongoDB.

Pré-requisitos

- Conhecimento básico de Java
- MongoDB instalado localmente (porta padrão 27017)
- Java Development Kit (JDK) 8 ou superior
- Maven ou Gradle para gerenciamento de dependências

Passo 1: Configurar Seu Projeto

Primeiro, adicione o driver Java do MongoDB ao seu projeto:

Maven:

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>4.9.0</version>
</dependency>
```

Gradle:

```
implementation 'org.mongodb:mongodb-driver-sync:4.9.0'
```

Passo 2: Conectar ao MongoDB

Crie uma nova classe Java `MongoCRUD.java` e adicione o código de conexão:

```
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;

public class MongoCRUD {
    private MongoClient mongoClient;

    public MongoCRUD() {
        // Conectar ao MongoDB rodando localmente
        this.mongoClient =
MongoClients.create("mongodb://localhost:27017");
        System.out.println("Conectado ao MongoDB com sucesso!");
    }

    public void close() {
        mongoClient.close();
        System.out.println("Conexão encerrada.");
    }

    public static void main(String[] args) {
        MongoCRUD mongoApp = new MongoCRUD();
        try {
            // Nossas operações virão aqui
        } finally {
            mongoApp.close();
        }
    }
}
```

Passo 3: Criar Documentos

Vamos adicionar um método para inserir dados na coleção "pessoas":

```
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import org.bson.Document;
import java.util.Arrays;
```

```
public class MongoCRUD {
    // ... código anterior ...

    private MongoDatabase database;
    private MongoCollection<Document> collection;

    public MongoCRUD() {
        this.mongoClient =
MongoClients.create("mongodb://localhost:27017");
        this.database = mongoClient.getDatabase("lab_mongodb");
        this.collection = database.getCollection("pessoas");
    }

    public void criarDocumentos() {
        System.out.println("\n=== Criando Documentos ===");

        // Inserção de documento único
        Document pessoa1 = new Document("nome", "João Silva")
                .append("idade", 28)
                .append("cidade", "São Paulo");
        collection.insertOne(pessoa1);
        System.out.println("Documento 1 inserido: " + pessoa1);

        // Inserção de múltiplos documentos
        Document pessoa2 = new Document("nome", "Maria Oliveira")
                .append("idade", 32)
                .append("cidade", "Rio de Janeiro");

        Document pessoa3 = new Document("nome", "Carlos Souza")
                .append("idade", 25)
                .append("cidade", "Belo Horizonte");

        collection.insertMany(Arrays.asList(pessoa2, pessoa3));
        System.out.println("Mais 2 documentos inseridos");
    }

    public static void main(String[] args) {
        MongoCRUD mongoApp = new MongoCRUD();
```

```
        try {
            mongoApp.criarDocumentos();
        } finally {
            mongoApp.close();
        }
    }
}
```

Passo 4: Ler Documentos

Adicione capacidades de consulta à sua aplicação:

```
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Projections;
import org.bson.conversions.Bson;

public class MongoCRUD {
    // ... código anterior ...

    public void lerDocumentos() {
        System.out.println("\n=== Lendo Documentos ===");

        // 1. Encontrar todos os documentos
        System.out.println("Todos os documentos:");
        collection.find().forEach(doc ->
System.out.println(doc.toJson()));

        // 2. Encontrar com filtro (idade > 25)
        System.out.println("\nPessoas com mais de 25 anos:");
        Bson filtroIdade = Filters.gt("idade", 25);
        collection.find(filtroIdade)
                .forEach(doc ->
System.out.println(doc.toJson()));

        // 3. Projeção (apenas nome e cidade)
        System.out.println("\nApenas nomes e cidades:");
        Bson projecao = Projections.fields(
```

```
            Projections.include("nome", "cidade"),
            Projections.excludeId()
        );
        collection.find()
                .projection(projecao)
                .forEach(doc ->
System.out.println(doc.toJson()));
    }

    public static void main(String[] args) {
        MongoCRUD mongoApp = new MongoCRUD();
        try {
            mongoApp.criarDocumentos();
            mongoApp.lerDocumentos();
        } finally {
            mongoApp.close();
        }
    }
}
```

Passo 5: Atualizar Documentos

Implemente atualizações de documentos:

```
import com.mongodb.client.model.Updates;
import com.mongodb.client.result.UpdateResult;

public class MongoCRUD {
    // ... código anterior ...

    public void atualizarDocumentos() {
        System.out.println("\n=== Atualizando Documentos ===");

        // 1. Atualizar um documento
        Bson filtro = Filters.eq("nome", "João Silva");
        Bson atualizacao = Updates.set("profissao", "Engenheiro");
        UpdateResult resultado = collection.updateOne(filtro,
```

```
atualizacao);
        System.out.println(resultado.getModifiedCount() + "
documento(s) modificado(s)");

        // 2. Atualizar múltiplos documentos
        Bson filtroIdade = Filters.gt("idade", 25);
        Bson atualizacaoSalario = Updates.inc("salario", 500.00);
        UpdateResult resultadoMultiplo =
collection.updateMany(filtroIdade, atualizacaoSalario);
        System.out.println(resultadoMultiplo.getModifiedCount() +
" documentos atualizados com aumento salarial");

        // 3. Atualização complexa (definir se não existir)
        Bson atualizacaoComplexa = Updates.combine(
            Updates.set("profissao", "Desenvolvedor"),
            Updates.setOnInsert("data_criacao", new
java.util.Date())
        );
        collection.updateOne(Filters.eq("nome", "Carlos Souza"),
atualizacaoComplexa);
    }

    public static void main(String[] args) {
        MongoCRUD mongoApp = new MongoCRUD();
        try {
            mongoApp.criarDocumentos();
            mongoApp.lerDocumentos();
            mongoApp.atualizarDocumentos();
            mongoApp.lerDocumentos(); // Verificar atualizações
        } finally {
            mongoApp.close();
        }
    }
}
```

Passo 6: Deletar Documentos

Adicione métodos de exclusão:

```
import com.mongodb.client.result.DeleteResult;

public class MongoCRUD {
    // ... código anterior ...

    public void deletarDocumentos() {
        System.out.println("\n=== Deletando Documentos ===");

        // 1. Deletar um documento
        Bson filtro = Filters.eq("nome", "Carlos Souza");
        DeleteResult resultado = collection.deleteOne(filtro);
        System.out.println(resultado.getDeletedCount() + "
documento(s) deletado(s)");

        // 2. Deletar múltiplos documentos
        Bson filtroIdade = Filters.lt("idade", 30);
        DeleteResult resultadoMultiplo =
collection.deleteMany(filtroIdade);
        System.out.println(resultadoMultiplo.getDeletedCount() + "
documentos deletados onde idade < 30");
    }

    public static void main(String[] args) {
        MongoCRUD mongoApp = new MongoCRUD();
        try {
            mongoApp.criarDocumentos();
            mongoApp.lerDocumentos();
            mongoApp.atualizarDocumentos();
            mongoApp.deletarDocumentos();
            mongoApp.lerDocumentos(); // Verificar exclusões
        } finally {
            mongoApp.close();
        }
```

```
    }
}
```

Passo 7: Recursos Avançados

Indexação

```
import com.mongodb.client.model.Indexes;
import com.mongodb.client.model.IndexOptions;

public class MongoCRUD {
    // ... código anterior ...

    public void gerenciarIndices() {
        System.out.println("\n=== Gerenciando Índices ===");

        // 1. Criar índice único no nome
        collection.createIndex(
            Indexes.ascending("nome"),
            new IndexOptions().unique(true)
        );
        System.out.println("Índice único criado em 'nome'");

        // 2. Criar índice composto
        collection.createIndex(
            Indexes.compoundIndex(
                Indexes.ascending("cidade"),
                Indexes.descending("idade")
            )
        );
        System.out.println("Índice composto criado em cidade e
idade");

        // 3. Listar todos os índices
        System.out.println("\nÍndices atuais:");
        collection.listIndexes().forEach(idx ->
System.out.println(idx.toJson()));
```

```
    }
}
```

Agregação

```
import com.mongodb.client.model.Accumulators;
import com.mongodb.client.model.Aggregates;
import com.mongodb.client.model.Sorts;

public class MongoCRUD {
    // ... código anterior ...

    public void exemplosAgregacao() {
        System.out.println("\n=== Exemplos de Agregação ===");

        // 1. Média de idade por cidade
        System.out.println("\nMédia de idade por cidade:");
        collection.aggregate(Arrays.asList(
            Aggregates.group("$cidade",
                Accumulators.avg("mediaIdade", "$idade"),
                Accumulators.sum("contagem", 1)
            ),
            Aggregates.sort(Sorts.ascending("_id"))
        ).forEach(doc -> System.out.println(doc.toJson()));

        // 2. Estatísticas salariais por profissão
        System.out.println("\nEstatísticas salariais por
profissão:");
        collection.aggregate(Arrays.asList(
            Aggregates.group("$profissao",
                Accumulators.avg("salarioMedio", "$salario"),
                Accumulators.min("salarioMinimo", "$salario"),
                Accumulators.max("salarioMaximo", "$salario")
            ),
            Aggregates.sort(Sorts.descending("salarioMedio"))
        )).forEach(doc -> System.out.println(doc.toJson()));
```

```
    }
}
```

Exemplo Completo

Aqui está a classe completa com todas as operações:

```
import com.mongodb.client.*;
import com.mongodb.client.model.*;
import com.mongodb.client.result.*;
import org.bson.Document;
import org.bson.conversions.Bson;
import java.util.Arrays;

public class MongoCRUD {
    private MongoClient mongoClient;
    private MongoDatabase database;
    private MongoCollection<Document> collection;

    public MongoCRUD() {
        this.mongoClient =
MongoClients.create("mongodb://localhost:27017");
        this.database = mongoClient.getDatabase("lab_mongodb");
        this.collection = database.getCollection("pessoas");
        System.out.println("Conexão com MongoDB estabelecida");
    }

    // Todos os métodos que criamos acima...
    public void criarDocumentos() { /* ... */ }
    public void lerDocumentos() { /* ... */ }
    public void atualizarDocumentos() { /* ... */ }
    public void deletarDocumentos() { /* ... */ }
    public void gerenciarIndices() { /* ... */ }
    public void exemplosAgregacao() { /* ... */ }

    public void close() {
        mongoClient.close();
```

```
        System.out.println("Conexão encerrada");
    }

    public static void main(String[] args) {
        MongoCRUD mongoApp = new MongoCRUD();
        try {
            mongoApp.criarDocumentos();
            mongoApp.lerDocumentos();
            mongoApp.atualizarDocumentos();
            mongoApp.lerDocumentos();
            mongoApp.deletarDocumentos();
            mongoApp.lerDocumentos();
            mongoApp.gerenciarIndices();
            mongoApp.exemplosAgregacao();
        } finally {
            mongoApp.close();
        }
    }
}
```

Boas Práticas

1. **Gerenciamento de Conexão**: Sempre feche seu MongoClient quando terminar

2. **Operações em Massa**: Para múltiplas escritas, use operações em massa

3. **Projeções**: Recupere apenas os campos que você precisa

4. **Tratamento de Erros**: Adicione blocos try-catch adequados em código de produção

5. **Indexação**: Crie índices para seus padrões de consulta comuns

Como Tudo Funciona

Este tutorial explica passo a passo como realizar operações CRUD (Create, Read, Update, Delete) no MongoDB usando Java, desde a conexão até consultas avançadas com agregação.

1. Conexão com o MongoDB

Como Funciona?

O MongoDB usa o MongoClient para gerenciar conexões. Quando você cria uma instância dele, ele abre um pool de conexões (conexões reutilizáveis para melhor desempenho).

Código Explicado

```
MongoClient mongoClient =
MongoClients.create("mongodb://localhost:27017");
MongoDatabase database = mongoClient.getDatabase("lab_mongodb");
MongoCollection<Document> collection =
database.getCollection("pessoas");
```

- **MongoClients.create()** - Conecta ao servidor MongoDB (rodando em localhost:27017)
- **getDatabase()** - Pega (ou cria, se não existir) o banco de dados lab_mongodb
- **getCollection()** - Acessa (ou cria) a coleção pessoas

Importante: Sempre feche a conexão com mongoClient.close() para evitar vazamentos.

2. Inserindo Dados (Create)

Como Funciona?

- insertOne() - Insere um único documento (registro)
- insertMany() - Insere vários documentos de uma vez (mais eficiente)

Código Explicado

```
// Inserindo um documento
Document pessoa = new Document("nome", "João Silva")
    .append("idade", 30)
    .append("cidade", "São Paulo");

collection.insertOne(pessoa);

// Inserindo vários documentos
List<Document> pessoas = Arrays.asList(
    new Document("nome", "Maria").append("idade", 25),
    new Document("nome", "Carlos").append("idade", 40)
);

collection.insertMany(pessoas);
```

- Document - Representa um registro no MongoDB (como um JSON)
- append() - Adiciona campos ao documento

3. Consultando Dados (Read)

Como Funciona?

- find() - Retorna todos os documentos
- find(Filtro) - Retorna apenas documentos que atendem a um critério

Filtros Comuns

Método	Exemplo	Descrição
eq	Filters.eq("nome", "João")	Igual a "João"
gt	Filters.gt("idade", 25)	Maior que 25
lt	Filters.lt("idade", 30)	Menor que 30
and	Filters.and(eq, gt)	Combina filtros

Código Explicado

```
// Busca todos
collection.find().forEach(doc ->
System.out.println(doc.toJson()));

// Busca com filtro (idade > 25)
Bson filtro = Filters.gt("idade", 25);
collection.find(filtro).forEach(doc -> System.out.println(doc));
```

4. Atualizando Dados (Update)

Como Funciona?

- updateOne() - Atualiza o primeiro documento que bate com o filtro
- updateMany() - Atualiza todos os documentos que batem com o filtro

Operações de Atualização

Método	Exemplo	Descrição
set	Updates.set("idade", 31)	Altera o valor
inc	Updates.inc("idade", 1)	Incrementa (+1)
unset	Updates.unset("cidade")	Remove o campo

Código Explicado

```
// Atualiza o salário do João
Bson filtro = Filters.eq("nome", "João Silva");
Bson update = Updates.set("salario", 5000.00);
collection.updateOne(filtro, update);

// Aumenta salário de todos com +500
Bson filtroGeral = Filters.gt("salario", 3000);
Bson updateGeral = Updates.inc("salario", 500);
collection.updateMany(filtroGeral, updateGeral);
```

5. Deletando Dados (Delete)

Como Funciona?

- deleteOne() - Remove o primeiro documento que bate com o filtro
- deleteMany() - Remove todos os documentos que batem com o filtro

Código Explicado

```
// Deleta o João
collection.deleteOne(Filters.eq("nome", "João Silva"));
```

```
// Deleta todos com salário < 3000
collection.deleteMany(Filters.lt("salario", 3000));
```

6. Índices (Para Consultas Rápidas)

Como Funciona?

Índices aceleram buscas, mas ocupam espaço extra.

Tipos de Índices

- Simples - Indexes.ascending("nome")
- Único - new IndexOptions().unique(true)
- Composto - Indexes.compoundIndex(asc, desc)

Código Explicado

```
// Cria índice único no nome
collection.createIndex(
    Indexes.ascending("nome"),
    new IndexOptions().unique(true)
);

// Cria índice composto (cidade + idade)
collection.createIndex(
    Indexes.compoundIndex(
        Indexes.ascending("cidade"),
        Indexes.descending("idade")
    )
);
```

7. Agregações (Análise de Dados)

Como Funciona?

Agregações permitem filtrar, agrupar e calcular dados em pipelines.

Operações Comuns

Método	Exemplo	Descrição
group	Aggregates.group("$cidade")	Agrupa por cidade
avg	Accumulators.avg("media", "$salario")	Calcula média
sort	Aggregates.sort(Sorts.descending("idade"))	Ordena resultados

Código Explicado

```
// Média salarial por profissão
collection.aggregate(Arrays.asList(
    Aggregates.group("$profissao",
        Accumulators.avg("mediaSalarial", "$salario")
    ),
    Aggregates.sort(Sorts.descending("mediaSalarial"))
).forEach(doc -> System.out.println(doc));
```

Conclusão

Agora você sabe:

- Como conectar Java ao MongoDB
- Como fazer CRUD (inserir, ler, atualizar, deletar)
- Como acelerar consultas com índices
- Como analisar dados com agregações

Próximos passos:

- Transações (para operações complexas)
- Change Streams (monitorar alterações em tempo real)

- Driver Reativo (para programação assíncrona)

Dica: Experimente modificar os exemplos e criar suas próprias consultas!