



# PRW3 - Programação para a WEB III

## Demonstração Spring Data JPA

Conteúdo 11



# Avaliação da disciplina



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# AVALIAÇÃO DA DISCIPLINA

- Questionário em

**<https://forms.gle/Rj6Mp8cKp1PdQ58X6>**

- Completamente anônimo!
- Respostas enviadas para o Coordenador do Curso
- **RESPONDAM AGORA, POR FAVOR!!**



# Alteração do calendário

- 29.out Desenvolvimento de uma API REST (parte 4). Introdução Spring Security
- 5.nov Conteúdo extra: Spring Data.
- 12.nov Spring Security parte 2.
- 19.nov Spring Security parte 3.
- 26.nov Dia para trab. na **Avaliação 4. Entrega até 23:59.**
- 3.dez Demonstração: Microservices com Spring (vídeo-aula, s/ controle presença)
- 10.dez Revisão. Preparação para IFA...





# Spring Data JPA



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

O Spring Data JPA é um módulo do ecossistema Spring que simplifica drasticamente o trabalho com persistência de dados em aplicações Java que utilizam JPA (Java Persistence API).

Ele funciona como uma camada de abstração sobre o JPA tradicional, fornecendo uma interface mais simples e produtiva para realizar operações de banco de dados. Em vez de escrever código boilerplate repetitivo para operações CRUD básicas, o Spring Data JPA permite que você defina repositórios através de interfaces simples.

O conceito central são os **repositórios**. Você cria uma interface que estende `JpaRepository` ou `CrudRepository`, e o Spring Data JPA automaticamente gera a implementação dessas operações em tempo de execução. Por exemplo, métodos como `save()`, `findById()`, `findAll()`, `delete()` ficam disponíveis automaticamente.



Uma das funcionalidades mais poderosas são os **query methods** - você pode criar métodos seguindo convenções de nomenclatura e o Spring Data JPA gera automaticamente as consultas SQL correspondentes. Um método como `findByNomeAndIdade(String nome, int idade)` é automaticamente traduzido para uma query que busca registros pelo nome e idade.

Para consultas mais complexas, você pode usar a anotação `@Query` para escrever JPQL ou SQL nativo diretamente nos métodos do repositório.

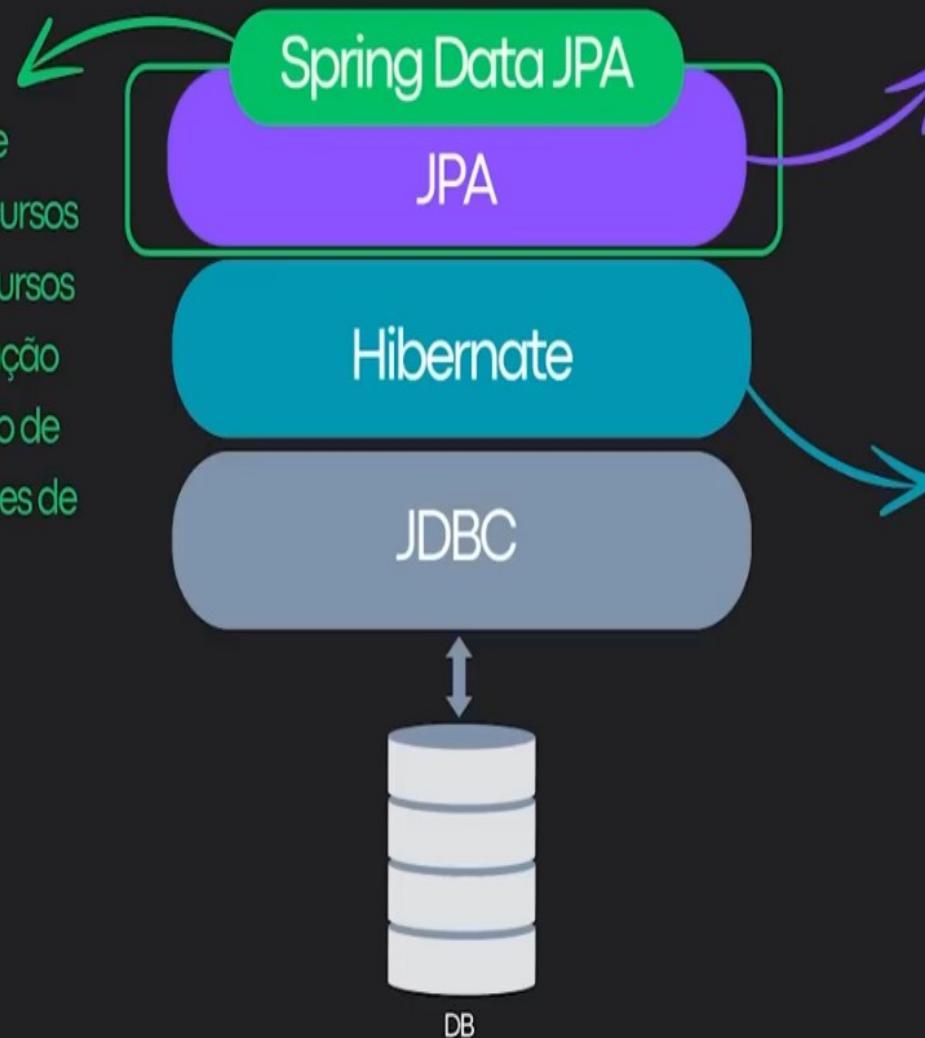
O Spring Data JPA também oferece recursos avançados como paginação, ordenação, auditorias automáticas e specifications para consultas dinâmicas, além de integração perfeita com o ecossistema Spring, incluindo transações declarativas e injeção de dependência.

Isso resulta em código mais limpo, menos propenso a erros e muito mais produtivo para desenvolvimento de aplicações que precisam persistir dados.



JPA (Java Persistence API) é uma especificação para mapeamento objeto-relacional em Java, incluindo anotações, consultas JPQL e APIs para interações com a base de dados.

Spring Data JPA é camada de abstração adicional, utilizando recursos da especificação JPA, além de recursos próprios, como uma implementação do padrão de repositories, criação de consultas na base a partir de nomes de atributos...



O Hibernate é uma das implementações mais comuns da especificação JPA.



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

## JPA Annotations

@JoinColumn

@Entity

@JoinTable

@Table

@OneToOne

@Id

@OneToMany

@GeneratedValue

@ManyToOne

@Column

@ManyToMany

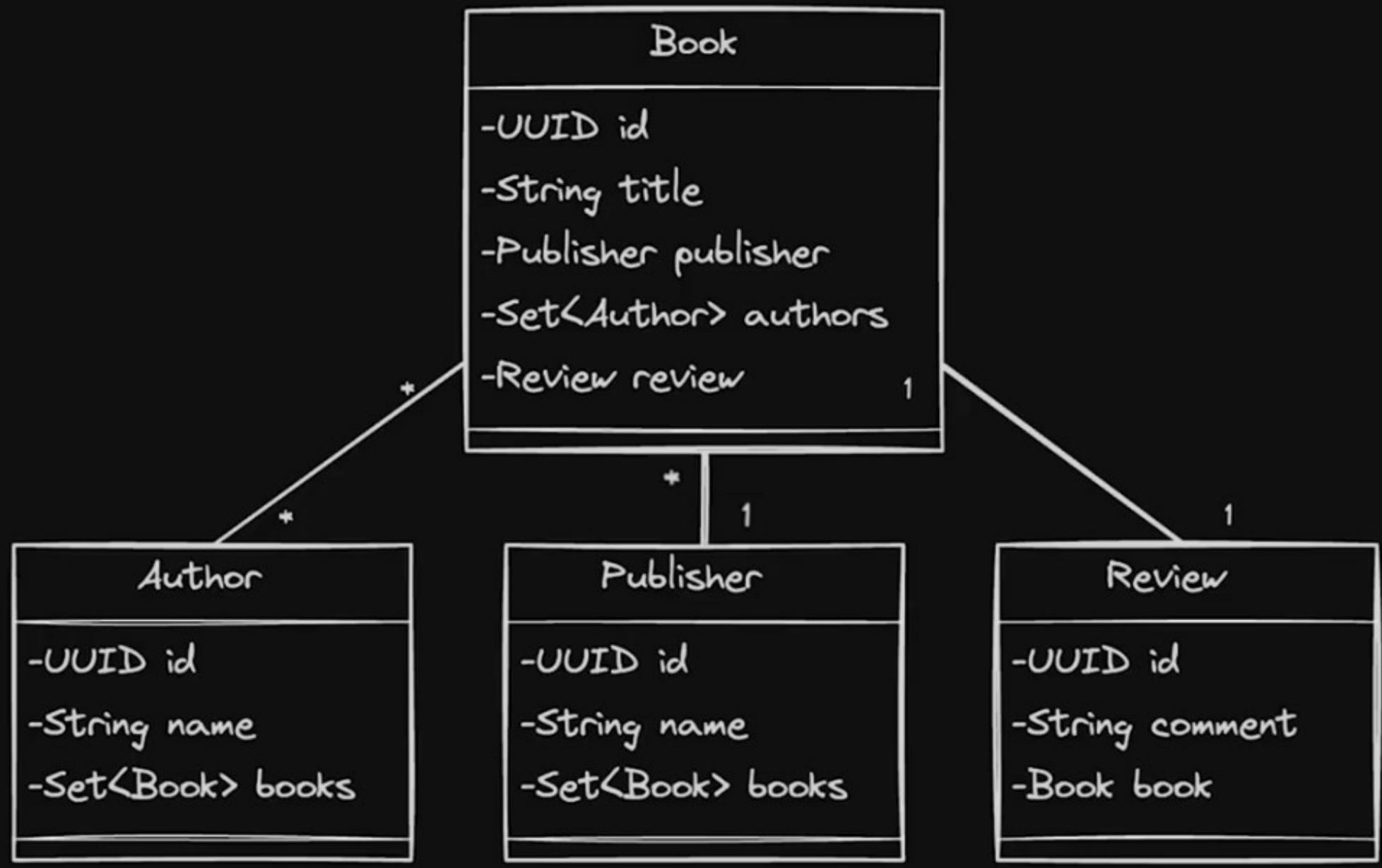
@Transactional

# Projeto que servirá de exemplo:



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# Projeto de exemplo:



Entenda esta aula mais como uma demonstração e apresentação de conceitos básicos, do que um estudo aprofundado.

A idéia aqui é  
"empurrar na ladeira" !!



# Criando o projeto:



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

Spring Initializr +

start.spring.io

spring initializr

Project

Gradle - Groovy  
 Gradle - Kotlin  Maven

Language

Java  Kotlin  
 Groovy

Spring Boot

4.0.0 (SNAPSHOT)  4.0.0 (M3)  3.5.7 (SNAPSHOT)  
 3.5.6  3.4.11 (SNAPSHOT)  3.4.10

Project Metadata

Group carlao2005

Artifact livraria

Name livraria

Description Aula sobre Spring Data

Package name carlao2005.livraria

Packaging  Jar  War

Java  25  21  17

Dependencies ADD DEPENDENCIES... CTRL + B

**Spring Web** WEB  
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Spring Data JPA** SQL  
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

**H2 Database** SQL  
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

GENERATE CTRL + ↵ EXPLORE CTRL + SPACE ...

# application.properties



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

```
spring.application.name=livraria
```

```
# Configuração do banco de dados H2
```

```
spring.datasource.url=jdbc:h2:file:./DATA/livraria
```

```
spring.datasource.username=sa
```

```
spring.datasource.password=
```

```
spring.h2.console.enabled=true
```

```
spring.h2.console.path=/h2-console
```

```
spring.jpa.show-sql=true
```

```
spring.jpa.properties.hibernate.format_sql=true
```

```
# cria as tabelas automaticamente com base nas entidades
```

```
spring.jpa.hibernate.ddl-auto=create
```



```
spring.application.name=livraria
```

```
# Configuração do banco de dados H2
```

```
spring.datasource.url=jdbc:h2:file:./DATA/livraria  
spring.datasource.username=sa
```

```
spring.datasource.password=
```

```
spring.h2.console.enabled=true
```

```
spring.h2.console.path=/h2-console
```

```
spring.jpa.show-sql=true
```

```
spring.jpa.properties.hibernate.format_sql=true
```

```
# cria as tabelas automaticamente com base nas entidades
```

```
spring.jpa.hibernate.ddl-auto=create
```

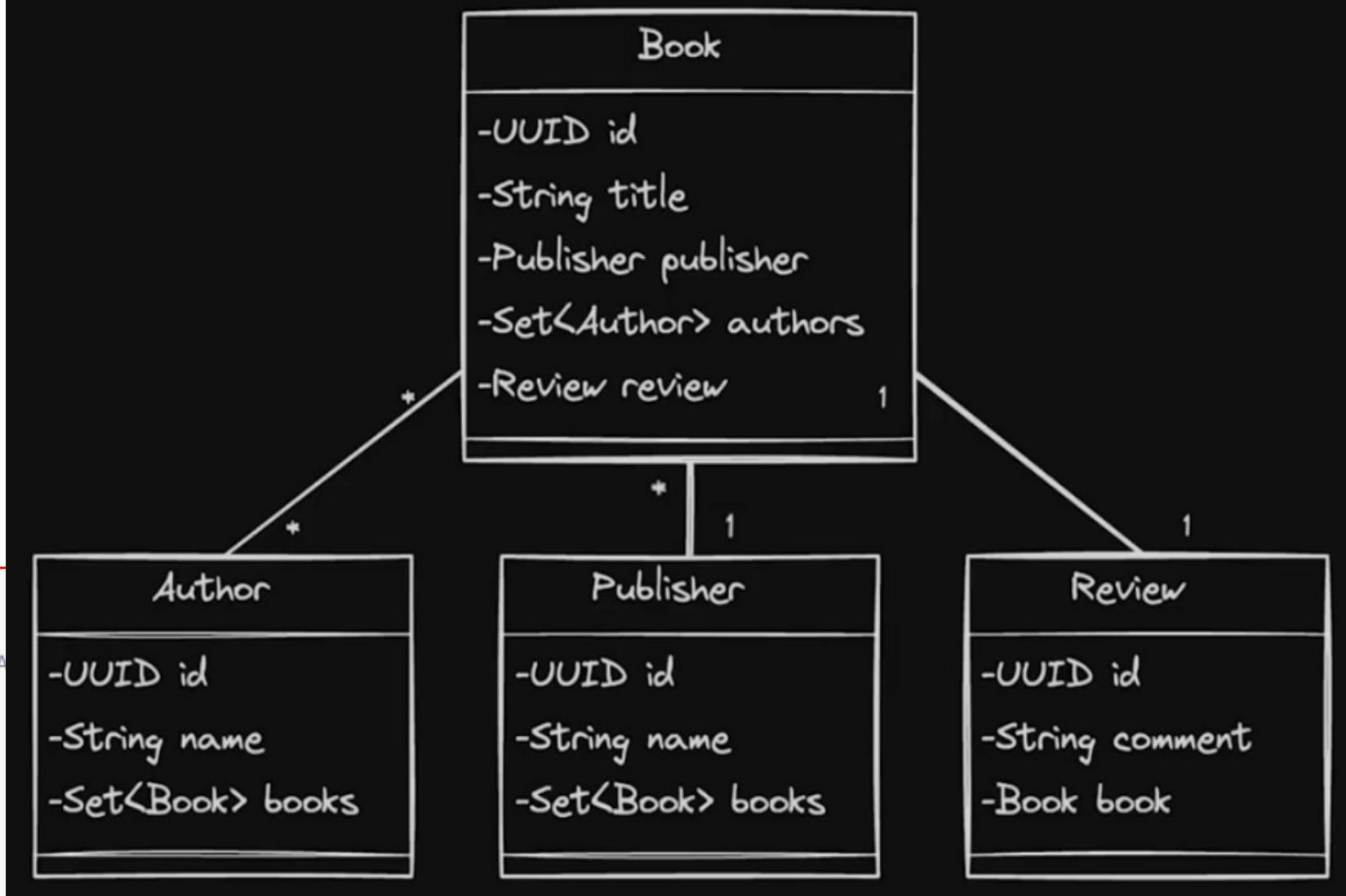


Criando os models...

OBS: incompletos!  
Vamos desenvolvendo passo a passo!



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos



### Project

```

livraria ~/HD_PRINCIPAL
  .idea
  .mvn
  DATA
    livraria.mv.db
  src
    main
      java
        carlao2005.livraria
          LivrariaApplication
        carlao2005.livraria.models
          AuthorModel
          BookModel
          PublisherModel
          ReviewModel
  
```



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# BookModel



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# BookModel

```
@Entity
@Table(name = "TB_BOOK")
public class BookModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false, unique = true)
    private String title;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }
}
```

# BookModel

```
@Entity
@Table(name = "TB_BOOK")
public class BookModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false, unique = true)
    private String title;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }
}
```

 O que é `Serializable`

A interface `Serializable` (do pacote `java.io`) serve para indicar que **um objeto pode ser convertido em uma sequência de bytes** — ou seja, pode ser **serializado**.

Essa conversão é necessária quando o objeto precisa:

- ser salvo em **disco** (por exemplo, em um arquivo);
- ser enviado pela **rede** (como em uma API remota);
- ou ser **armazenado em cache**, transferido entre camadas, ou **guardado em sessão** (no caso de aplicações web).

O processo inverso — transformar os bytes de volta em um objeto Java — chama-se **desserialização**.

# BookModel

```
@Entity
@Table(name = "TB_BOOK")
public class BookModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false, unique = true)
    private String title;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }
}
```



Por que o BookModel implementa Serializable?

No caso do JPA/Hibernate, embora não seja estritamente obrigatório, é **fortemente recomendado** que as entidades implementem Serializable.

Isso ocorre porque:

- O Hibernate pode precisar serializar entidades internamente (por exemplo, ao armazená-las em cache de segundo nível ou ao passá-las entre contextos de persistência);
- Servidores de aplicação podem serializar objetos para manter estado em sessão (em aplicações web com Spring MVC, por exemplo).

# BookModel

```
@Entity
@Table(name = "TB_BOOK")
public class BookModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false, unique = true)
    private String title;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }
}
```



O que é serialVersionUID

A linha:

java

Copiar código

```
private static final long serialVersionUID = 1L;
```

é um identificador de versão da classe.

Ela serve para garantir que, ao deserializar um objeto, a classe que está lendo seja compatível com a classe que foi usada para gerar o objeto originalmente.

# Resumo

## Item

## Explicação

`implements Serializable`

Permite que o objeto seja convertido em bytes (**serializado**)

`serialVersionUID`

Identificador de versão da classe para garantir compatibilidade na desserialização

Por que é útil no JPA

Hibernate e servidores podem serializar entidades internamente

Boa prática

Sempre declarar o `serialVersionUID` manualmente



# BookModel

```
@Entity
@Table(name = "TB_BOOK")
public class BookModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false)
    private String title;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }
}
```

## 2. A anotação @GeneratedValue

Ela indica como o valor da chave primária será gerado.

O JPA permite várias estratégias, controladas pelo parâmetro `strategy`.

No seu caso:

java

Copiar código

```
@GeneratedValue(strategy = GenerationType.AUTO)
```

O `AUTO` significa que o próprio provedor JPA (Hibernate) vai escolher automaticamente a melhor estratégia, dependendo do banco de dados utilizado.

Por exemplo:

- Em bancos H2 ou PostgreSQL, normalmente será usada uma `sequence`;
- Em bancos MySQL, geralmente será usada uma `auto-increment column`;
- Em bancos Oracle, ele pode criar uma `sequence` específica para a tabela.

Assim, o Hibernate escolhe o modo mais adequado para gerar o valor do ID.

# BookModel

```
@Entity
@Table(name = "TB_BOOK")
public class BookModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false, unique = true)
    private String title;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }
}
```

private UUID id;

Aqui, o tipo do ID é `java.util.UUID` — um identificador **universalmente único**, de 128 bits, como:

550e8400-e29b-41d4-a716-446655440000

Copiar código

Ele é amplamente usado em sistemas distribuídos, pois garante unicidade **sem depender do banco de dados**.

Mesmo que o banco gere os IDs (por causa do `@GeneratedValue`), o tipo `UUID` é uma escolha moderna e segura, especialmente quando se quer evitar problemas com colisões de IDs ou migrações entre bancos.

# BookModel

```
@Entity
@Table(name = "TB_BOOK")
public class BookModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false, unique = true)
    private String title;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }
}
```

Atributo	Função	SQL gerado
nullable = false	Impede valores nulos	NOT NULL
unique = true	Garante valores únicos	UNIQUE

# BookModel

```
@Entity
@Table(name = "TB_BOOK")
public class BookModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false, unique = true)
    private String title;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }
}
```

# PublisherModel



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# PublisherModel

```
@Entity
@Table(name = "TB_PUBLISHER")
public class PublisherModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false, unique = true)
    private String name;

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

# PublisherModel

```
@Entity
@Table(name = "TB_PUBLISHER")
public class PublisherModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false, unique = true)
    private String name;

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

# AuthorModel



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# AuthorModel

```
@Entity
@Table(name = "TB_AUTHOR")
public class AuthorModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false, unique = true)
    private String name;

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

# AuthorModel

```
@Entity
@Table(name = "TB_AUTHOR")
public class AuthorModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false, unique = true)
    private String name;

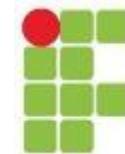
    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

# ReviewModel



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# ReviewModel

```
@Entity
@Table(name = "TB REVIEW")
public class ReviewModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false)
    private String comment;

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }

    public String getComment() {
        return comment;
    }

    public void setComment(String comment) {
        this.comment = comment;
    }
}
```

# ReviewModel

```
@Entity
@Table(name = "TB_REVIEW")
public class ReviewModel implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @Column(nullable = false)
    private String comment;

    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }

    public String getComment() {
        return comment;
    }

    public void setComment(String comment) {
        this.comment = comment;
    }
}
```

Executando para  
criar as tabelas  
no banco...



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

Run LivrariaApplication x



/usr/lib/jvm/java-1.21.0-openjdk-amd64/bin/java ...

```
  .   ---      -      -- - 
/\ / _-'_ _ -_(_)_ -- _ - \ \ \
( ( )\__| '_| '_| '_\ \ ` | \ \ \
\W _--|_|_|_|_|_|_|(|_| ) ) )
' |____| .--|_|_|_|_\_,| / / / /
=====|_|=====|___/=/_/_/_/
```

:: Spring Boot :: (v3.5.6)

```
2025-10-24T14:30:11.483-03:00 INFO 16444 --- [livraria] [
2025-10-24T14:30:11.485-03:00 INFO 16444 --- [livraria] [
2025-10-24T14:30:12.022-03:00 INFO 16444 --- [livraria] [
2025-10-24T14:30:12.038-03:00 INFO 16444 --- [livraria] [
2025-10-24T14:30:12.402-03:00 INFO 16444 --- [livraria] [
2025-10-24T14:30:12.414-03:00 INFO 16444 --- [livraria] [
2025-10-24T14:30:12.415-03:00 INFO 16444 --- [livraria] [
2025-10-24T14:30:12.448-03:00 INFO 16444 --- [livraria] [
2025-10-24T14:30:12.448-03:00 INFO 16444 --- [livraria] [
2025-10-24T14:30:12.585-03:00 INFO 16444 --- [livraria] [
2025-10-24T14:30:12.646-03:00 INFO 16444 --- [livraria] [
2025-10-24T14:30:12.682-03:00 INFO 16444 --- [livraria] [
2025-10-24T14:30:12.939-03:00 INFO 16444 --- [livraria] [
```

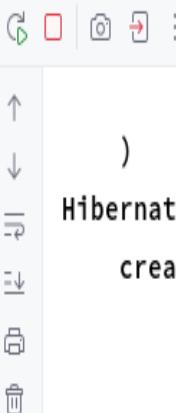
```
main] c.livraria.LivrariaApplication : Starting LivrariaApplic
main] c.livraria.LivrariaApplication : No active profile set,
main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Da
main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data re
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with
main] o.apache.catalina.core.StandardService : Starting service [Tomca
main] o.apache.catalina.core.StandardEngine : Starting Servlet engine
main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring emb
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationCont
main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing F
main] org.hibernate.Version : HHH000412: Hibernate OR
main] o.h.c.internal.RegionFactoryInitiator : HHH000026: Second-level
main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup
```

Run LivrariaApplication x



```
2025-10-24T14:30:13.181-03:00 INFO 16444 --- [livraria] [main] org.hibernate.orm.connections.pooling : HHH10001005: Database JDBC URL [Connecting through datasource 'HikariDataSource (HikariPool-1)']
Database driver: undefined/unknown
Database version: 2.3.232
Autocommit mode: undefined/unknown
Isolation level: undefined/unknown
Minimum pool size: undefined/unknown
Maximum pool size: undefined/unknown
2025-10-24T14:30:13.800-03:00 INFO 16444 --- [livraria] [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA pla
Hibernate:
    drop table if exists tb_author cascade
Hibernate:
    drop table if exists tb_book cascade
Hibernate:
    drop table if exists tb_publisher cascade
Hibernate:
    drop table if exists tb_review cascade
Hibernate:
    create table tb_author (
        id uuid not null,
        name varchar(255) not null unique,
        primary key (id)
    )
```

Run LivrariaApplication x



```
        primary key (id)
```

```
)
```

Hibernate:

```
create table tb_book (
```

```
    id uuid not null,
```

```
    title varchar(255) not null unique,
```

```
    primary key (id)
```

```
)
```

Hibernate:

```
create table tb_publisher (
```

```
    id uuid not null,
```

```
    name varchar(255) not null unique,
```

```
    primary key (id)
```

```
)
```

Hibernate:

```
create table tb_review (
```

```
    id uuid not null,
```

```
    comment varchar(255) not null,
```

```
    primary key (id)
```

```
)
```

2025-10-24T14:30:13.838-03:00 INFO 16444 --- [livraria] [

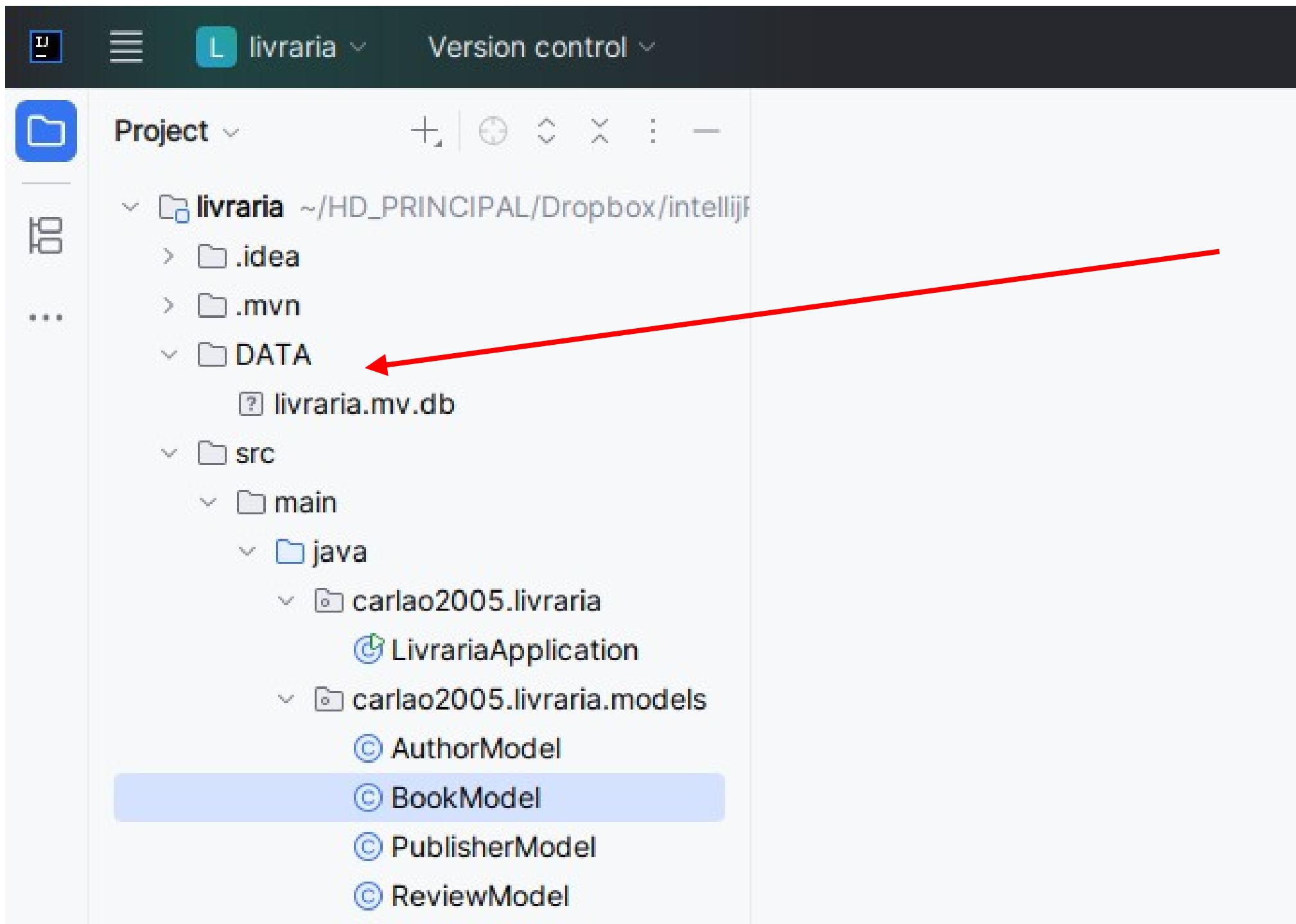
2025-10-24T14:30:13.880-03:00 WARN 16444 --- [livraria] [

2025-10-24T14:30:14.126-03:00 INFO 16444 --- [livraria] [

main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA Entit

main] JpaBaseConfiguration\$JpaWebConfiguration : spring.jpa.open-in-vi

main] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available



jdbc:h2:file:/DATA/livraria

TB_AUTHOR
ID
NAME
Indexes
TB_BOOK
ID
TITLE
Indexes
TB_PUBLISHER
ID
NAME
Indexes
TB REVIEW
ID
COMMENT
Indexes
INFORMATION_SCHEMA
Users
H2 2.3.232 (2024-08-11)

Run

Run Selected

Auto complete

Clear

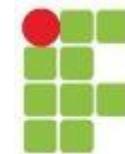
SQL statement:

## Important Commands

		Displays this Help Page
		Shows the Command History
	Ctrl+Enter	Executes the current SQL statement
	Shift+Enter	Executes the SQL statement defined by the text selection
	Ctrl+Space	Auto complete
		Disconnects from the database

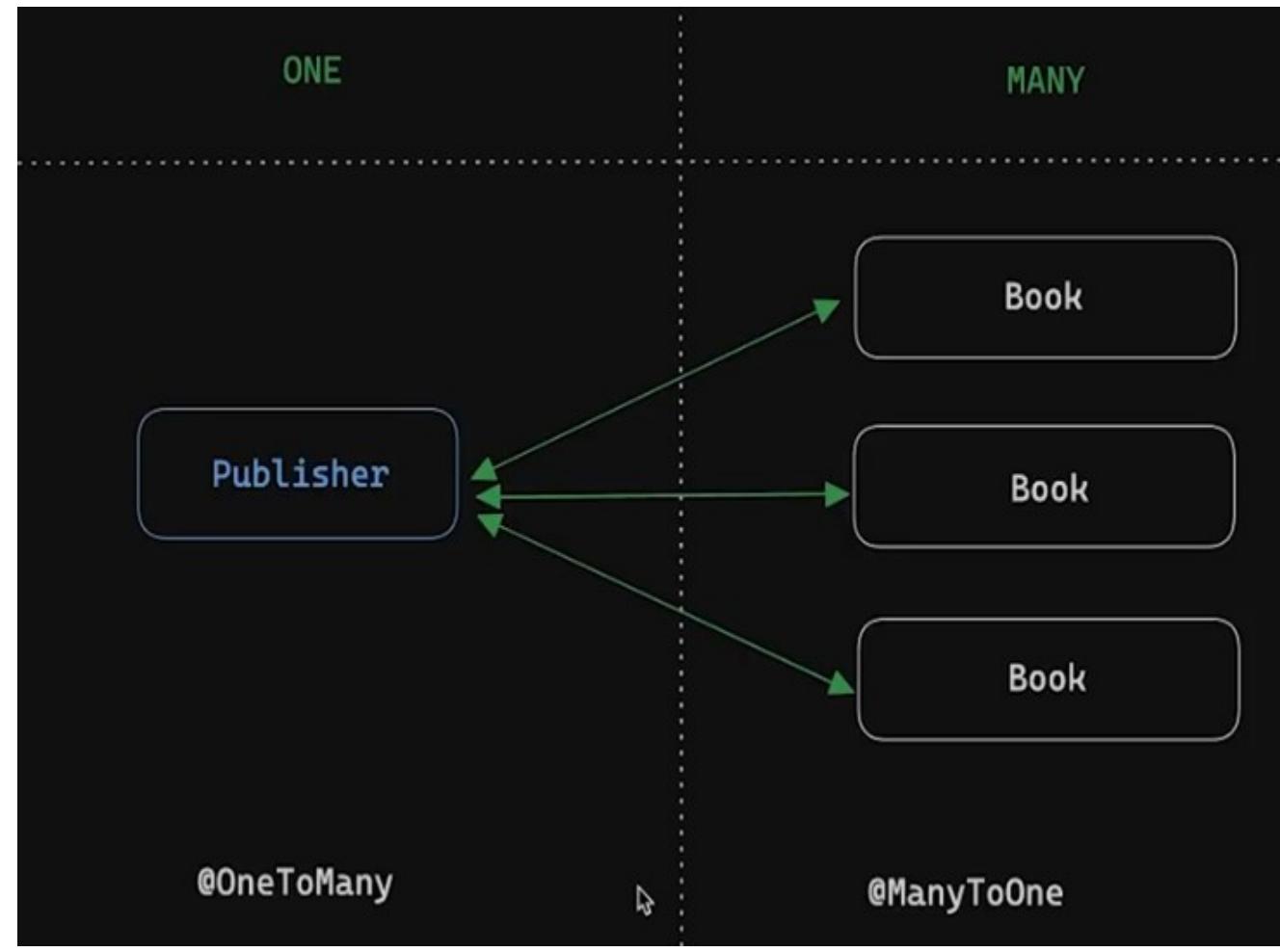
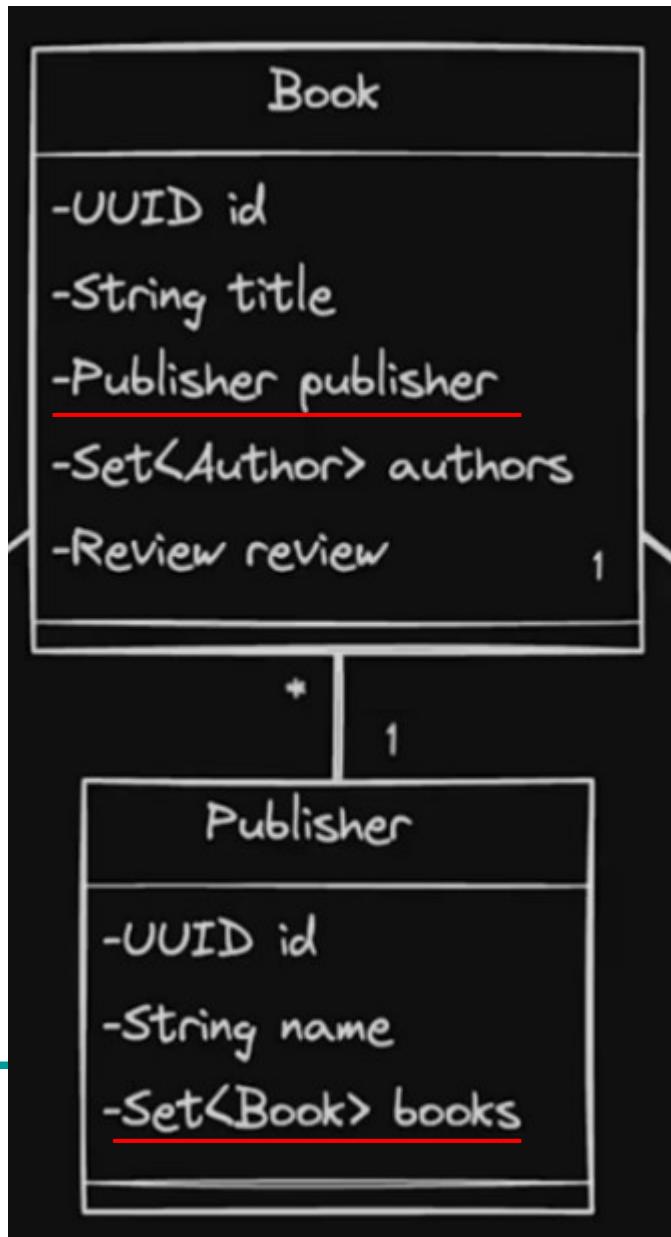


# Relacionamentos entre entidades



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# Relacionamento entre Book e Publisher



# Adição em BookModel

```
@ManyToOne  
@JoinColumn(name = "publisher_id")  
private PublisherModel publisher;  
  
public PublisherModel getPublisher() {  
    return publisher;  
}  
  
public void setPublisher(PublisherModel publisher) {  
    this.publisher = publisher;  
}
```



# Adição em BookModel

```
@ManyToOne
@JoinColumn(name = "publisher_id")
private PublisherModel publisher;

public PublisherModel getPublisher() {
    return publisher;
}

public void setPublisher(PublisherModel publisher) {
    this.publisher = publisher;
}
```

## ✳️ 1. A anotação @ManyToOne

Essa anotação diz ao JPA/Hibernate que o atributo `publisher` representa um **relacionamento de muitos-para-um** com outra entidade, no caso, `PublisherModel`.

Em outras palavras:

- Muitos livros podem pertencer ao mesmo editor.
- Mas cada livro tem apenas um editor.



# Adição em BookModel

```
@ManyToOne  
@JoinColumn(name = "publisher_id")  
private PublisherModel publisher;  
  
public PublisherModel getPublisher() {  
    return publisher;  
}  
  
public void setPublisher(PublisherModel publisher) {  
    this.publisher = publisher;  
}
```

## 💡 2. A anotação `@JoinColumn`

A anotação `@JoinColumn` define **como o relacionamento será mapeado na tabela do banco de dados**.

java

 Copiar código

```
@JoinColumn(name = "publisher_id")
```

Aqui, você está dizendo ao JPA:

- Que a tabela `TB_BOOK` (da entidade `BookModel`) terá uma **coluna chamada `publisher_id`**;
- Essa coluna armazenará o **identificador (`id`) do editor**, servindo como **chave estrangeira (foreign key)** para a tabela de `PublisherModel`.

# Adição em PublisherModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@OneToMany(mappedBy = "publisher", fetch = FetchType.LAZY)
private Set<BookModel> books = new HashSet<>();

public Set<BookModel> getBooks() {
    return books;
}

public void setBooks(Set<BookModel> books) {
    this.books = books;
}
```

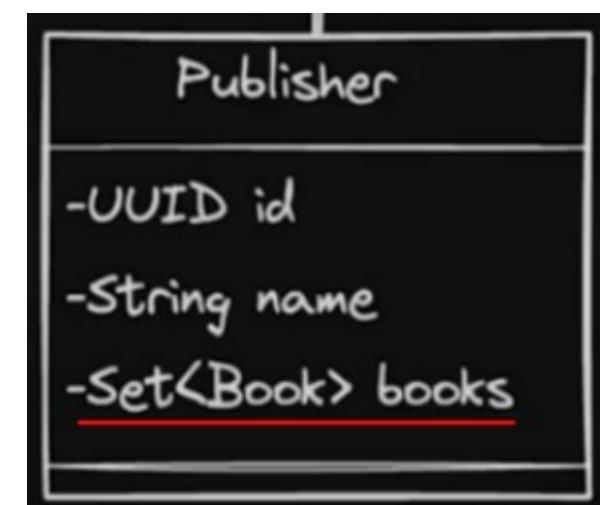


# Adição em PublisherModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@OneToMany(mappedBy = "publisher", fetch = FetchType.LAZY)
private Set<BookModel> books = new HashSet<>();
```

```
public Set<BookModel> getBooks() {
    return books;
}
```

```
public void setBooks(Set<BookModel> books) {
    this.books = books;
}
```



# Adição em PublisherModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@OneToMany(mappedBy = "publisher", fetch = FetchType.LAZY)
private Set<BookModel> books = new HashSet<>();

public Set<BookModel> getBooks() {
    return books;
}

public void setBooks(Set<BookModel> books) {
    this.books = books;
}
```



1. O JPA trabalha melhor com `Set` do que com `List` em `@OneToMany`

- Em um relacionamento `@OneToMany`, o conjunto de entidades filhas (`books`) geralmente **não precisa ter ordem definida**, apenas precisa **garantir unicidade** — ou seja, um mesmo livro não deve aparecer duas vezes na coleção do mesmo publisher.
- `Set` garante exatamente isso: **não permite elementos duplicados**.
- Já `List` permite duplicatas e depende de **índice**, o que **só faz sentido** se você quiser representar uma **ordem explícita** entre os elementos (por exemplo, capítulos numerados).



# Adição em PublisherModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@OneToMany(mappedBy = "publisher", fetch = FetchType.LAZY)
private Set<BookModel> books = new HashSet<>();
```

```
public Set<BookModel> getBooks() {
    return books;
}
```

```
public void setBooks(Set<BookModel> books) {
    this.books = books;
}
```



## 2. HashSet evita problemas com duplicidade e performance

- O `HashSet` usa uma **tabela hash** para garantir unicidade e busca rápida ( $O(1)$  na média).
- `ArrayList`, por outro lado, permite **elementos duplicados** e sua busca é linear ( $O(n)$ ).
- Quando o Hibernate precisa sincronizar a coleção (por exemplo, ao comparar entidades carregadas com o que está no banco), ele precisa verificar se um objeto já está presente.
  - Se a coleção for um `HashSet`, essa verificação é **muito mais eficiente**.
  - Se for uma `List`, pode haver duplicatas e a verificação é **mais custosa**.



# Adição em PublisherModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@OneToMany(mappedBy = "publisher", fetch = FetchType.LAZY)
private Set<BookModel> books = new HashSet<>();

public Set<BookModel> getBooks() {
    return books;
}

public void setBooks(Set<BookModel> books) {
    this.books = books;
}
```

 3. O Hibernate, por padrão, usa `Set` em relacionamentos `@OneToMany`

A documentação do Hibernate e várias boas práticas (como nas especificações da JPA e guias da Red Hat) recomendam:

Prefira `Set` para coleções `@OneToMany`, a menos que a ordem seja relevante.



# Adição em PublisherModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@OneToMany(mappedBy = "publisher", fetch = FetchType.LAZY)
private Set<BookModel> books = new HashSet<>();

public Set<BookModel> getBooks() {
    return books;
}

public void setBooks(Set<BookModel> books) {
    this.books = books;
}
```

```
@ManyToOne
@JoinColumn(name = "publisher_id")
private PublisherModel publisher;
```

`mappedBy = "publisher"` indica que:

👉 o lado dono do relacionamento é o atributo `publisher` dentro da classe `BookModel`.

Ou seja:

- A tabela `TB_BOOK` tem a chave estrangeira (`publisher_id`).
- A entidade `PublisherModel` apenas “espelha” a relação, não cria coluna no banco.
- Em resumo:

`mappedBy = "publisher"` → diz que a ligação é mapeada pelo campo `publisher` da outra entidade (`BookModel`), e este é o lado que realmente possui a FK no banco.

# Adição em PublisherModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@OneToMany(mappedBy = "publisher", fetch = FetchType.LAZY)
private Set<BookModel> books = new HashSet<>();

public Set<BookModel> getBooks() {
    return books;
}

public void setBooks(Set<BookModel> books) {
    this.books = books;
}
```

fetch = FetchType.LAZY significa que:

👉 os livros ( books ) só serão carregados do banco quando realmente forem acessados no código.

Ou seja:

- Ao buscar um PublisherModel , os livros não vêm juntos imediatamente.
- Eles só são buscados quando você chama, por exemplo, publisher.getBooks() .
- Em resumo:

LAZY = carregamento sob demanda (melhor desempenho).

EAGER = carregamento imediato (pode ser mais pesado).

# Adição em PublisherModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@OneToMany(mappedBy = "publisher", fetch = FetchType.LAZY)
private Set<BookModel> books = new HashSet<>();

public Set<BookModel> getBooks() {
    return books;
}

public void s
    this.books
}
```



## O que significa

Essa anotação é do **Jackson**, a biblioteca que o Spring usa para converter objetos Java em JSON e vice-versa.

A propriedade

java

Copiar código

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
```

diz ao Jackson que o atributo:

- ✓ pode ser usado ao desserializar (ou seja, quando o JSON é recebido numa requisição `POST` ou `PUT`);
- ✗ não deve aparecer ao serializar (ou seja, quando o objeto é convertido em JSON para retornar numa resposta `GET`).

# Adição em PublisherModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@OneToMany(mappedBy = "publisher", fetch = FetchType.LAZY)
private Set<BookModel> books = new HashSet<>();

public Set<BookModel> getBooks() {
    return books;
}

public void setBooks(Set<BookModel> books) {
    this.books = books;
}
```

Isso quer dizer que:

- Quando você **recebe um JSON** para criar ou atualizar um `Publisher`, o campo `books` **pode ser lido** (ex.: incluir IDs de livros, se desejar).
- Mas quando você **retorna um Publisher** em uma resposta (`GET /publishers`), o campo `books` **não será mostrado no JSON de saída**.



## Exemplo prático

### 👉 JSON recebido (entrada)

Você pode enviar isso para criar uma editora:

json

```
{  
    "name": "Editora Alpha",  
    "books": [  
        { "title": "Livro 1" },  
        { "title": "Livro 2" }  
    ]  
}
```

Copiar código

O Spring vai **ler** o campo `books`, porque ele é “**write only**”.

### 👉 JSON retornado (saída)

Mas ao buscar a mesma editora com:

bash

```
GET /publishers/1
```

Copiar código

O retorno será algo como:

json

```
{  
    "id": "b7d6b2b4-32a8-45e3-9c6b-8c93b4b1c6df",  
    "name": "Editora Alpha"  
}
```

Copiar código

Note que o campo `books` não aparece — mesmo que existam livros associados.

# Executando...



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

Hibernate:

```
drop table if exists tb_author cascade
```

Hibernate:

```
drop table if exists tb_book cascade
```

Hibernate:

```
drop table if exists tb_publisher cascade
```

Hibernate:

```
drop table if exists tb_review cascade
```

Hibernate:

```
create table tb_author (
```

```
    id uuid not null,
```

```
    name varchar(255) not null unique,
```

```
    primary key (id)
```

```
)
```



**Hibernate:**

```
create table tb_book (
    id uuid not null,
    publisher_id uuid,
    title varchar(255) not null unique,
    primary key (id)
)
```

**Hibernate:**

```
create table tb_publisher (
    id uuid not null,
    name varchar(255) not null unique,
    primary key (id)
)
```

**Hibernate:**

```
create table tb_review (
    id uuid not null,
    comment varchar(255) not null,
    primary key (id)
)
```



**Hibernate:**

```
alter table if exists tb_book
    add constraint FKnirnq5sunln2aixln0wfrlx1o
        foreign key (publisher_id)
            references tb_publisher
```

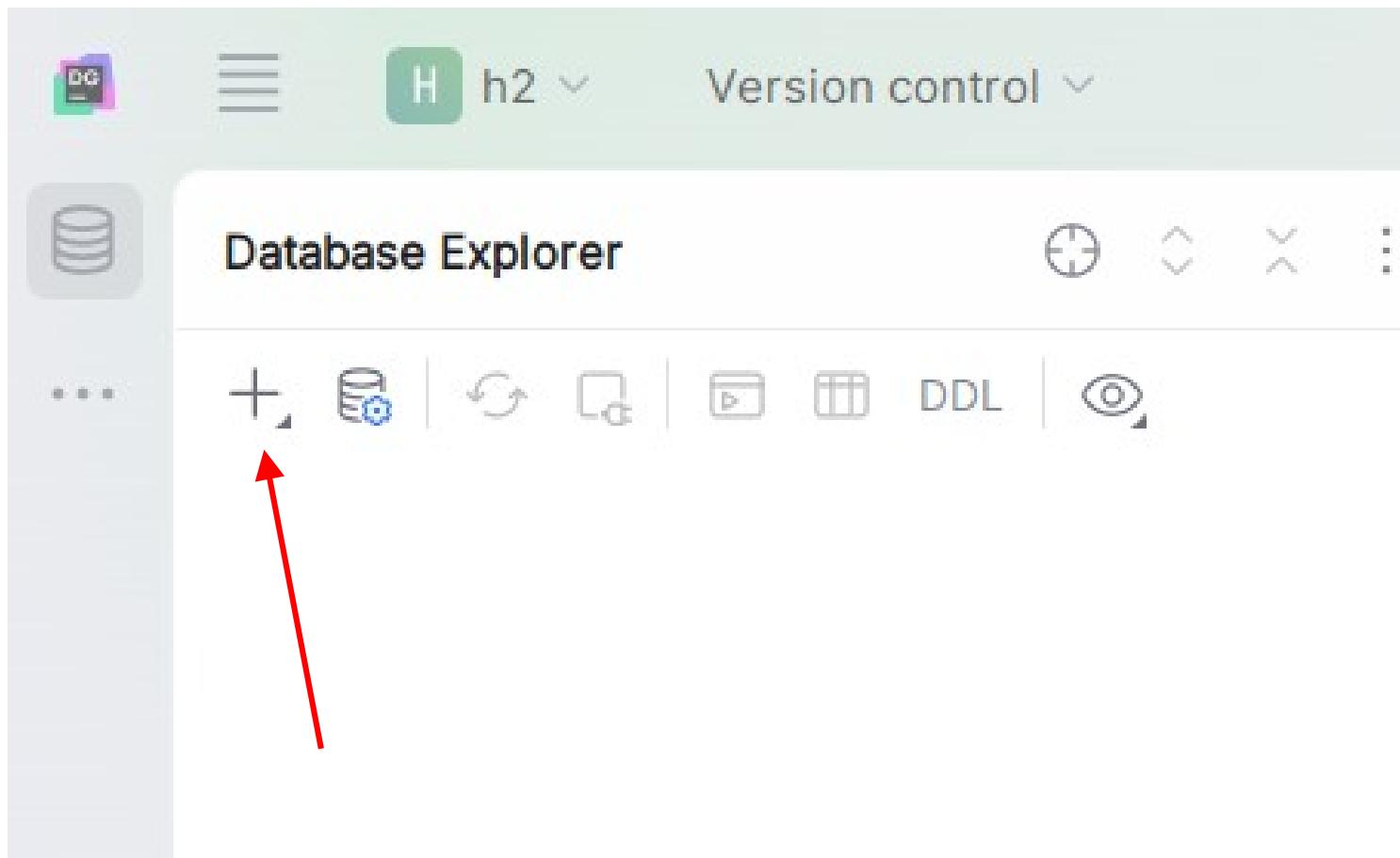


INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

Durante a preparação desta aula,  
descobri que o DataGrip  
agora é gratuito para uso não comercial !



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

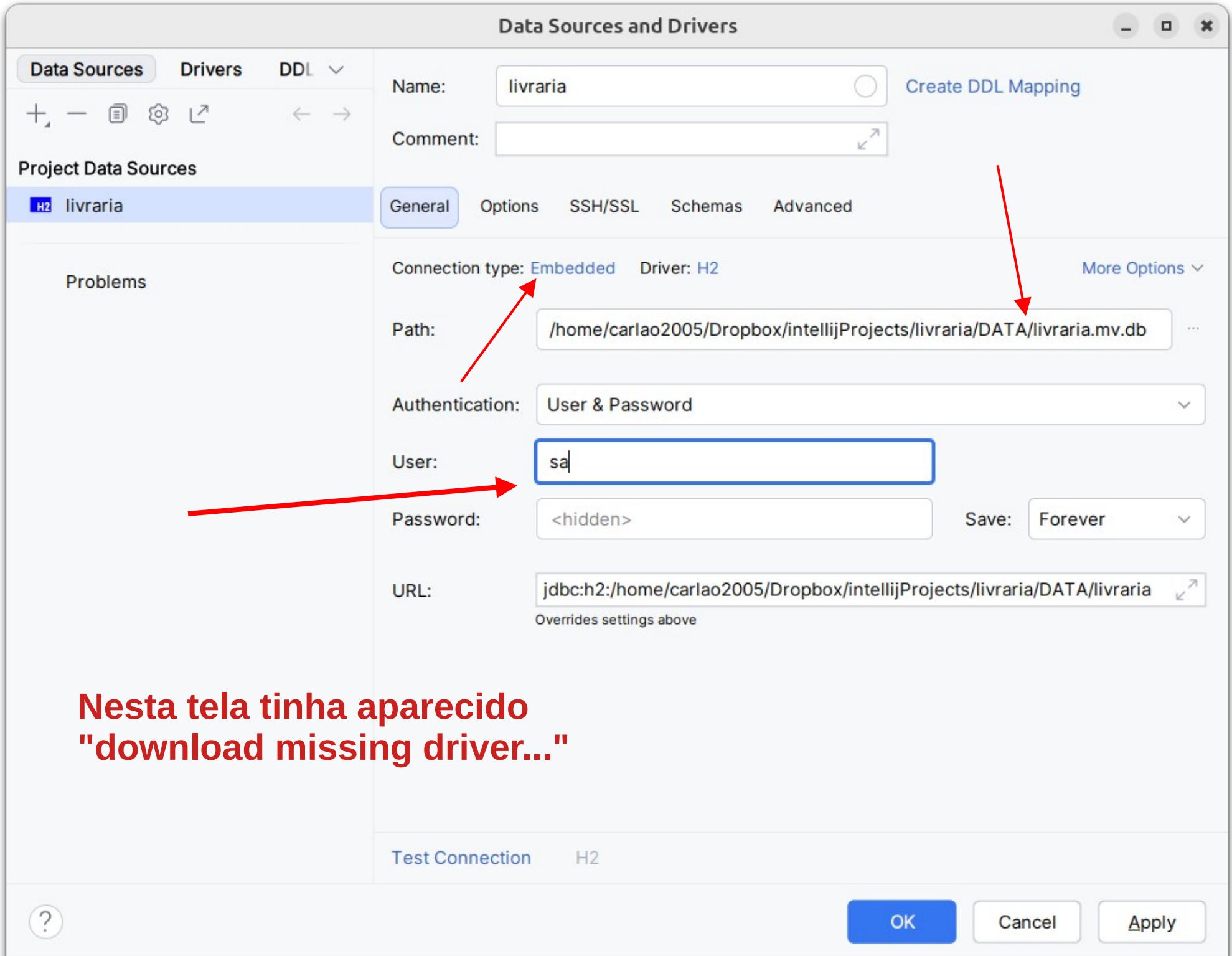


INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

The screenshot shows the Database Explorer panel of a code editor. At the top, there are icons for file operations (New, Open, Save, etc.) and a dropdown showing "h2". Below the header is a toolbar with icons for adding a new source (+), selecting a driver (gear), refreshing (refresh), and other database-related functions. A sub-menu is open under the "Data Source" button, listing options: Data Source (selected and highlighted in blue), DDL Data Source, Data Source from URL, Data Source from Path, and Driver. To the right of the sub-menu is a list of recent databases, with "H2 H2" selected and highlighted. Further down the list are "Complete Support" and several other database entries with their respective icons.

- Recent
- H2 H2
- Complete Support
- Amazon Aurora MySQL
- Amazon Redshift
- Apache Cassandra
- Apache Derby
- Apache Hive





## Data Sources and Drivers

**Data Sources** Drivers DDL

+ -

**Project Data Sources**

H2 livraria	General Options SSH/SSL Schemas Advanced
-------------	--

Problems

Connection type: **Embedded** Driver: **H2** More Options

Path: /home/carla02005/Dropbox/intelliJProjects/livraria/DATA/livraria.mv.db

Authentication: User & Password

User: sa

Password: <hidden> Save: Forever

URL: jdbc:h2:/home/carla02005/Dropbox/intelliJProjects/livraria/DATA/livraria

Overrides settings above

**Succeeded**

DBMS: H2 (ver. 2.3.232 (2024-08-11))   
Case sensitivity: plain=upper, delimited=exact  
Driver: H2 JDBC Driver (ver. 2.3.232 (2024-08-11), JDBC4.3)  
  
Ping: 44 ms

**Test Connection** H2

OK Cancel Apply

**Atenção!!**  
**Aplicação não pode estar rodando!!!**

## Data Sources and Drivers

**Data Sources**   **Drivers**   **DDL** ▾

+ - ⌂ ⚙ ⌛ ↻ ↽

**Project Data Sources**

H2	livraria
----	----------

**General**   Options   SSH/SSL   Schemas   Advanced

Connection type: **Embedded**   Driver: **H2**   [More Options](#) ▾

Path:  ...

Authentication:

User:

Password:  Save:  ▾

URL:  [Overrides settings above](#) ↻ ↽

Test Connection ✓ H2

?

OK Cancel Apply

## livraria 1

### LIVRARIA 1 of 2

#### PUBLIC

##### tables 5

> TB\_AUTHOR

✓ TB\_BOOK

##### columns 3

ID UUID

PUBLISHER\_ID UUID

TITLE CHARACTER VARYING(255)

##### keys 2

##### foreign keys 1

FKNIRNQ5SUNLN2AIXLN0WFRLX1O (PUBLISHER\_ID) → TB\_PUBLISHER (ID)

##### indexes 3



### TB\_PUBLISHER

##### columns 2

ID UUID

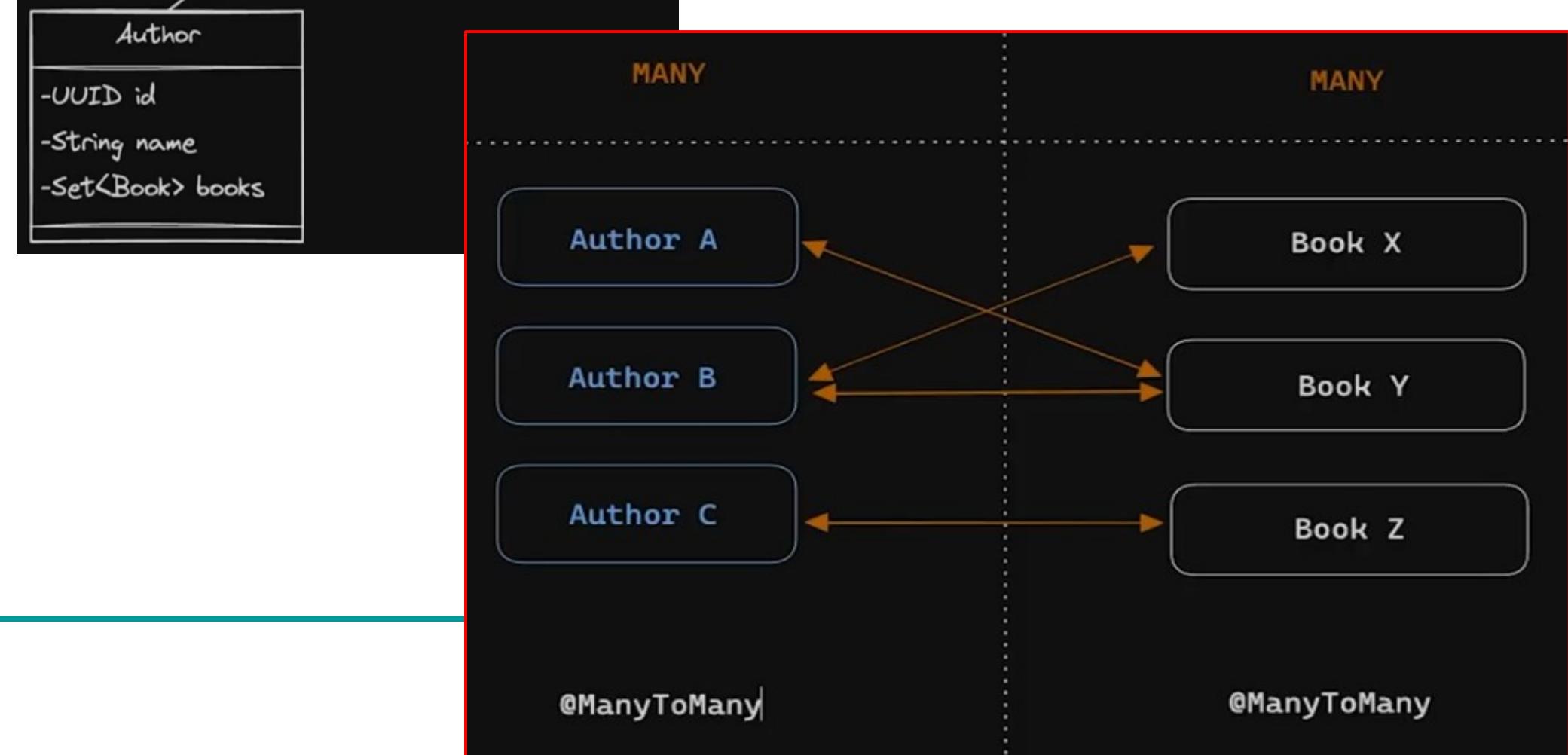
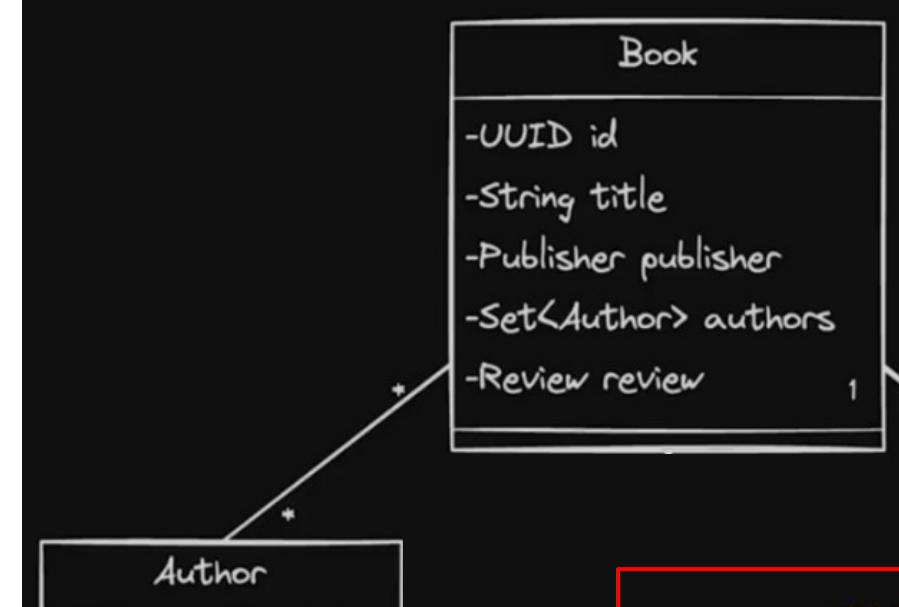
NAME CHARACTER VARYING(255)

##### keys 2

##### indexes 2

> TB\_REVIEW

# Relacionamento entre Livro e Autor

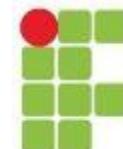


## 1. Conceito do relacionamento Many-to-Many

Um relacionamento **muitos-para-muitos** significa que:

- Um livro pode ter vários autores.
- Um autor pode ter vários livros.

Como o modelo relacional **não suporta diretamente** relacionamentos muitos-para-muitos, o JPA (e o H2) resolvem isso criando uma tabela intermediária (de junção), também chamada de **tabela associativa** ou join table.



# Adição em BookModel

```
@ManyToMany
```

```
@JoinTable(
```

```
    name = "tb_book_author",
```

```
    joinColumns = @JoinColumn(name = "book_id"),
```

```
    inverseJoinColumns = @JoinColumn(name = "author_id"))
```

```
private Set<AuthorModel> authors = new HashSet<>();
```

```
public Set<AuthorModel> getAuthors() {  
    return authors;
```

```
}
```

```
public void setAuthors(Set<AuthorModel> authors) {  
    this.authors = authors;  
}
```



# Adição em BookModel

@ManyToMany

@JoinTable(

name = "tb\_book\_author",

joinColumns = @JoinColumn(name = "book\_id"),

inverseJoinColumns = @JoinColumn(name = "author\_id"))

private Set<AuthorModel> authors = new HashSet<>();

```
public Set<AuthorModel> getAuthors() {  
    return authors;  
}
```

```
public void setAuthors(Set<AuthorModel> authors) {  
    this.authors = authors;  
}
```

Essa linha declara um **conjunto** ( `Set` ) de autores dentro da entidade `BookModel`:

- `Set<AuthorModel>` : representa uma coleção **sem elementos duplicados** de objetos `AuthorModel`.
- `authors` : nome do atributo que guarda os autores do livro.
- `= new HashSet<>()` : inicializa a coleção vazia para evitar `NullPointerException` ao adicionar autores.

Por que `Set` é melhor aqui:

Em relacionamentos muitos-para-muitos, usar `Set` evita **duplicatas automaticamente**, o que é mais adequado do que `List`, que permite elementos repetidos.

# Adição em BookModel

```
@ManyToMany ←  
@JoinTable(  
    name = "tb_book_author",  
    joinColumns = @JoinColumn(name = "book_id"),  
    inverseJoinColumns = @JoinColumn(name = "author_id"))  
private Set<AuthorModel> authors = new HashSet<>();
```

A anotação `@ManyToMany` indica ao JPA que há um **relacionamento muitos-para-muitos** entre duas entidades — neste caso, entre `BookModel` e `AuthorModel`.

👉 Isso significa que:

- Um livro pode ter **vários autores**, e
- Um autor pode ter **vários livros**.

Como bancos relacionais não suportam esse tipo de relação diretamente, o JPA cria (ou utiliza) **uma tabela intermediária** para armazenar as associações.



# Adição em BookModel

@ManyToMany  
@JoinTable(

```
    name = "tb_book_author",  
    joinColumns = @JoinColumn(name = "book_id"),  
    inverseJoinColumns = @JoinColumn(name = "author_id"))
```

```
private Set<AuthorModel> authors = new HashSet<>();
```

A anotação `@JoinTable` define a **tabela intermediária** que o JPA usa para representar um relacionamento, geralmente em casos de `@ManyToMany`.

Ela especifica:

- O **nome da tabela** que vai armazenar as associações entre as entidades.
- As colunas que serão **chaves estrangeiras**, ligando a tabela intermediária às tabelas das entidades envolvidas.

Resumindo:

`@JoinTable` indica qual tabela e quais colunas serão usadas para mapear o vínculo entre duas entidades relacionadas.



# Adição em BookModel

@ManyToMany

@JoinTable(

```
    name = "tb_book_author",
    joinColumns = @JoinColumn(name = "book_id"),
    inverseJoinColumns = @JoinColumn(name = "author_id"))
```

```
private Set<AuthorModel> authors = new HashSet<>();
```

Na anotação `@JoinTable`, os principais parâmetros são:

- `name` : define o **nome da tabela intermediária** que armazenará a relação entre as entidades.
- `joinColumns` : especifica a **coluna que referencia a entidade dona do relacionamento** (aquele onde a anotação está declarada).
- `inverseJoinColumns` : especifica a **coluna que referencia a outra entidade** envolvida na relação.

Resumindo:

Esses três parâmetros determinam **o nome da tabela de junção e quais colunas ligam cada entidade a ela**.



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# AuthorModel



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# Adição em AuthorModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@ManyToMany(mappedBy = "authors", fetch = FetchType.LAZY)
private Set<BookModel> books = new HashSet<>();

public Set<BookModel> getBooks() {
    return books;
}

public void setBooks(Set<BookModel> books) {
    this.books = books;
}
```



# Adição em AuthorModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@ManyToMany(mappedBy = "authors", fetch = FetchType.LAZY)
private Set<BookModel> books = new HashSet<>();
```

```
public Set<BookModel> getBooks() {
    return books;
}
```

```
public void setBooks(Set<BookModel> books) {
    this.books = books;
}
```



# Adição em AuthorModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@ManyToMany(mappedBy = "authors", fetch = FetchType.LAZY)
private Set<BookModel> books = new HashSet<>();
```

Esse trecho declara o atributo que representa os **livros associados a um autor**, com algumas configurações importantes:

- `@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)` : indica que o campo só será usado na **deserialização** (entrada de dados JSON) e **não será exibido na saída** — evita **recursão infinita** ou excesso de dados ao converter objetos em JSON.
- `@ManyToMany(mappedBy = "authors")` : define o **relacionamento muitos-para-muitos** com `BookModel`, informando que o **lado dono da relação** está em `BookModel` (no atributo `authors`).
- `fetch = FetchType.LAZY` : indica **carregamento sob demanda**, ou seja, os livros só serão buscados no banco quando realmente forem acessados.
- `Set<BookModel> books = new HashSet<>()` : armazena os livros associados, sem duplicatas e já inicializado.

**Resumindo:**

Esse trecho mapeia a relação entre autores e livros, evita ciclos na conversão para JSON e otimiza o carregamento dos dados.

# Executando...



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

Hibernate:

```
create table tb_book_author (
    author_id uuid not null,
    book_id uuid not null,
    primary key (author_id, book_id)
)
```



Hibernate:

```
alter table if exists tb_book
    add constraint FKnirnq5sunln2aixln0wfrlx1o
        foreign key (publisher_id)
            references tb_publisher
```

Hibernate:

```
alter table if exists tb_book_author
    add constraint FKbo2nc1syneieprfficcl25yvq
        foreign key (author_id)
            references tb_author
```

Hibernate:

```
alter table if exists tb_book_author
    add constraint FK3w593lyk61mg3qgo60se015v4
        foreign key (book_id)
            references tb_book
```

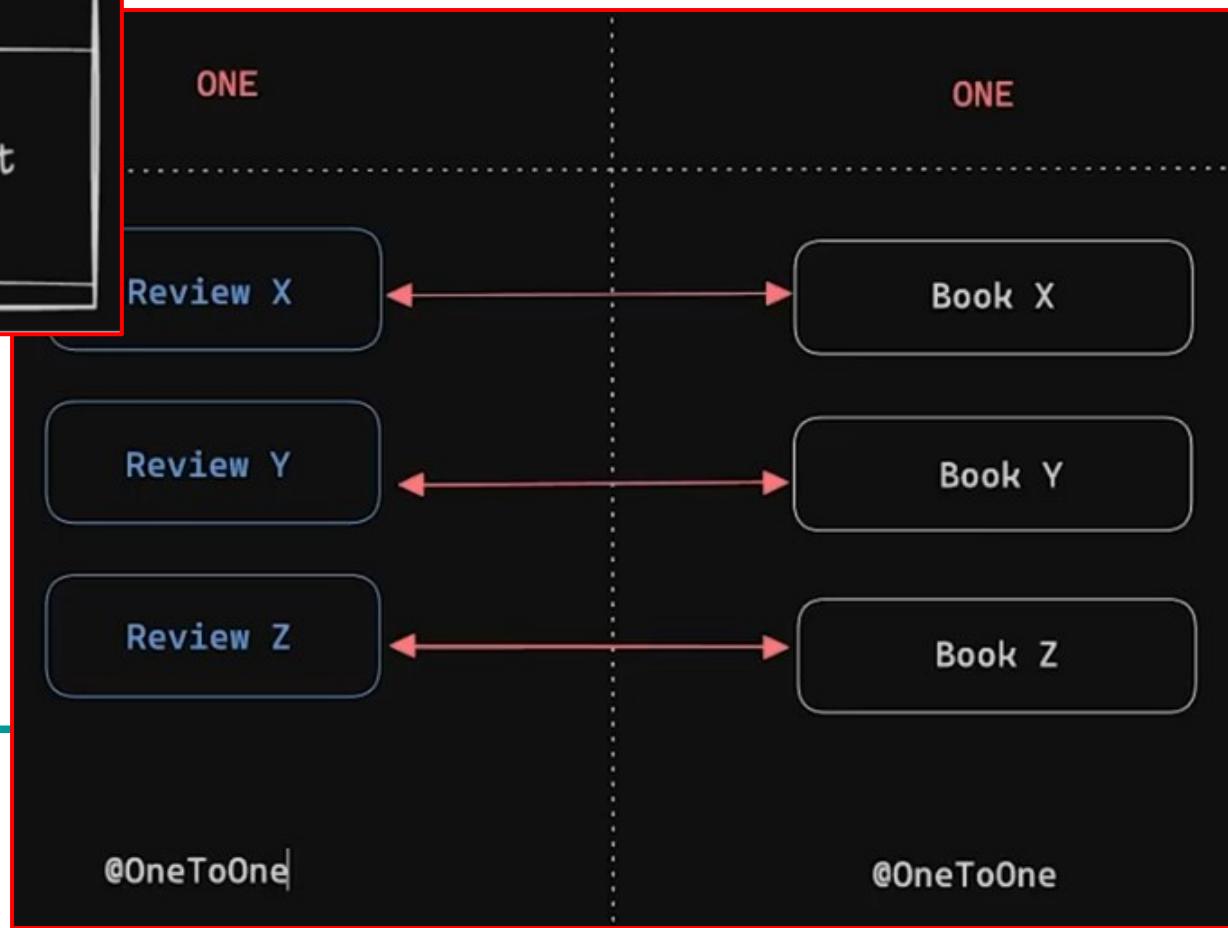
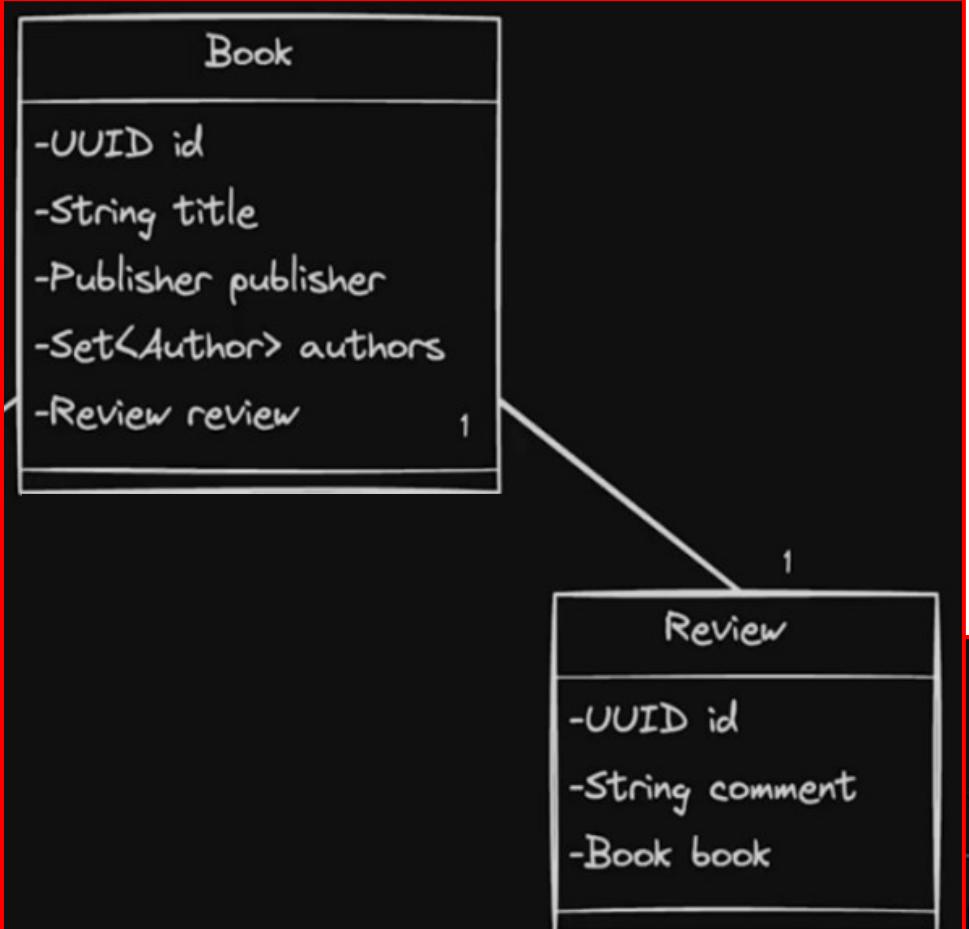


The screenshot shows a database schema for a 'LIVRARIA' database. The schema includes three main tables: TB\_AUTHOR, TB\_BOOK, and TB\_BOOK\_AUTHOR. Red arrows point from the left margin to each of these three tables.

**LIVRARIA** (1 of 2)

- PUBLIC**
- tables** 5
  - TB\_AUTHOR**
    - columns** 2
      - ID UUID
      - NAME CHARACTER VARYING(255)
    - keys** 2
    - indexes** 2
  - TB\_BOOK**
    - columns** 3
      - ID UUID
      - PUBLISHER\_ID UUID
      - TITLE CHARACTER VARYING(255)
    - keys** 2
    - foreign keys** 1
    - indexes** 3
  - TB\_BOOK\_AUTHOR**
    - columns** 2
      - AUTHOR\_ID UUID
      - BOOK\_ID UUID
    - keys** 1
    - foreign keys** 2
      - FK3W593LYK61MG3QGO60SE015V4 (BOOK\_ID) → TB\_BOOK (ID)
      - FKBO2NC1SYNEIEPRFFICCL25YVQ (AUTHOR\_ID) → TB\_AUTHOR (ID)

# Relacionamento entre Livro e Resenha



# Adição em BookModel

```
@OneToOne(mappedBy = "book", cascade = CascadeType.ALL)
private ReviewModel review;

public ReviewModel getReview() {
    return review;
}

public void setReview(ReviewModel review) {
    this.review = review;
}
```



# Adição em BookModel

```
@OneToOne(mappedBy = "book", cascade = CascadeType.ALL)  
private ReviewModel review;
```

```
public ReviewModel getReview() {  
    return review;  
}
```

```
public void setReview(ReviewModel review) {  
    this.review = review;  
}
```

```
private ReviewModel review;
```

Indica que um livro ( BookModel ) tem uma única resenha ( ReviewModel ) associada.



# Adição em BookModel

```
@OneToOne(mappedBy = "book", cascade = CascadeType.ALL)  
private ReviewModel review;
```

- A anotação `@OneToOne(mappedBy = "book", cascade = CascadeType.ALL)`

Essa anotação define o relacionamento **um-para-um (1:1)** com a entidade `ReviewModel`.

## 1. `@OneToOne`

Indica que a relação é **um para um** — ou seja:

- Cada livro tem **uma** resenha;
- Cada resenha pertence a **um** livro.

---

## 2. `mappedBy = "book"`

Esse parâmetro diz que **esta entidade ( BookModel ) não é a dona da relação**.

A coluna de chave estrangeira que faz o vínculo está **na outra entidade, ReviewModel**.

# Adição em BookModel

```
@OneToOne(mappedBy = "book", cascade = CascadeType.ALL)  
private ReviewModel review;
```

## Adição em ReviewModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)  
@OneToOne  
@JoinColumn(name = "book_id")  
private BookModel book;
```

- A anotação `@OneToOne(mappedBy = "book", cascade = CascadeType.ALL)`

Essa anotação define o relacionamento **um-para-um (1:1)** com a entidade `ReviewModel`.

### 1. `@OneToOne`

Indica que a relação é **um para um** — ou seja:

- Cada livro tem **uma** resenha;
- Cada resenha pertence a **um** livro.

---

### 2. `mappedBy = "book"`

Esse parâmetro diz que **esta entidade ( BookModel ) não é a dona da relação**.

A coluna de chave estrangeira que faz o vínculo está **na outra entidade, ReviewModel**.

# Adição em BookModel

```
@OneToOne(mappedBy = "book", cascade = CascadeType.ALL)  
private ReviewModel review;
```

3. cascade = CascadeType.ALL

Define que **todas as operações** feitas em `BookModel` também serão **propagadas automaticamente** para `ReviewModel`.

Por exemplo:

- Se você **salvar** um `BookModel`, a `ReviewModel` associada será salva junto.
- Se você **atualizar** o livro, a resenha será atualizada.
- Se você **deletar** o livro, a resenha também será removida.

Isso é útil para manter a consistência entre objetos **fortemente acoplados**.

# Adição em BookModel

```
@OneToOne(mappedBy = "book", cascade = CascadeType.ALL)  
private ReviewModel review;
```



3. cascade = CascadeType.ALL

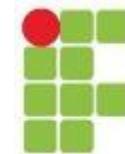
Define que todas as operações feitas no BookModel serão automaticamente realizadas no ReviewModel.

Por exemplo:

- Se você **salvar** um BookModel, a ReviewModel associada será salva junto.
- Se você **atualizar** o livro, a resenha será atualizada.
- Se você **deletar** o livro, a resenha também será removida.

Isso é útil para manter a consistência entre objetos fortemente acoplados.

# ReviewModel



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# Adição em ReviewModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@OneToOne
@JoinColumn(name = "book_id")
private BookModel book;

public BookModel getBook() {
    return book;
}

public void setBook(BookModel book) {
    this.book = book;
}
```



# Adição em ReviewModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@OneToOne
@JoinColumn(name = "book_id")
private BookModel book;
```

```
private BookModel book;
```

Esse campo representa o livro ao qual a resenha ( ReviewModel ) pertence.



# Adição em ReviewModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@OneToOne
@JoinColumn(name = "book_id")
private BookModel book;
```



## Parte 2 — @JoinColumn(name = "book\_id")

Essa anotação diz ao Hibernate **onde armazenar a chave estrangeira** no banco de dados.



Neste caso:

- A tabela `TB REVIEW` terá uma **coluna chamada `book_id`**.
- Essa coluna referencia a **chave primária (`id`) da tabela `TB_BOOK`**.



# Adição em ReviewModel

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@OneToOne
@JoinColumn(name = "book_id")
private BookModel book;
```



## Parte 3 — @JsonProperty(access = JsonProperty.Access.WRITE\_ONLY)

Essa anotação é do **Jackson**, o mecanismo de serialização JSON usado pelo Spring Boot.

Elá controla **como esse campo é tratado durante o envio e recebimento de JSONs na API REST**.

- **WRITE\_ONLY** → o campo pode ser recebido no **JSON de entrada** (quando você cria ou atualiza uma resenha),  
mas **não será incluído no JSON de resposta** (quando a resenha é retornada para o cliente).



# Executando...



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

--

  └─ tables 5

    > TB\_AUTHOR

    └─ TB\_BOOK

      └─ columns 3

         ID UUID

         PUBLISHER\_ID UUID

         TITLE CHARACTER VARYING(255)

      > TB\_BOOK\_AUTHOR

      > TB\_PUBLISHER

      └─ TB\_REVIEW

        └─ columns 3

           BOOK\_ID UUID

           ID UUID

           COMMENT CHARACTER VARYING(255)

        > keys 2

        └─ foreign keys 1

           FKP5TB2JVWEEB394IPB13B1MY8V (BOOK\_ID) → TB\_BOOK (ID)

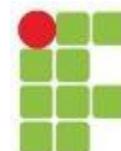
        > indexes 2

# Uma observação...

- Preparando esta aula, toda vez que eu alterava o código e rodava a aplicação, notei que ele apagava as tabelas e recriava do zero.
- No material que eu estava consultando, notei que ele não apagava as tabelas, apenas alterava (alter table) as tabelas, para implementar as mudanças efetuadas no código das entidades.
- Descobri que esta característica foi definida no arquivo

**application.properties**

no atributo **spring.jpa.hibernate.ddl-auto**



- O motivo: a configuração `spring.jpa.hibernate.ddl-auto`

Quando o Spring Boot inicializa com o Hibernate, ele lê o parâmetro:

properties

Copiar código

```
spring.jpa.hibernate.ddl-auto=...
```

Esse parâmetro controla **como o Hibernate gerencia o esquema (as tabelas)** no banco de dados.

- Valores possíveis (e o que cada um faz)

Valor	Descrição
<code>create</code>	Apaga todas as tabelas e recria do zero a cada inicialização. (Executa <code>DROP TABLE</code> e depois <code>CREATE TABLE</code> .)
<code>create-drop</code>	Igual a <code>create</code> , mas também apaga tudo ao encerrar a aplicação.
<code>update</code>	Tenta ajustar o schema existente para refletir as mudanças nas entidades. Não apaga dados existentes.
<code>validate</code>	Verifica se o schema atual está de acordo com as entidades, mas não altera nada. Gera erro se houver diferenças.
<code>none</code>	Desativa completamente a manipulação automática do schema. Você é responsável por criar/alterar as tabelas manualmente.



## • Como resolver

Se você quer que ele **apenas altere as tabelas existentes** (sem apagar dados), mude para:

properties

 Copiar código

```
spring.jpa.hibernate.ddl-auto=update
```

Isso fará com que o Hibernate:

- Crie novas tabelas se necessário;
- Adicione colunas novas;
- Mantenha dados existentes;
- **Mas não apague tabelas ou colunas.**

**⚠ Atenção:** o modo `update` é prático para **ambiente de desenvolvimento**, mas **não é recomendado em produção**, pois pode causar inconsistências dependendo da complexidade das alterações (por exemplo, mudança de tipo de coluna).





## application.properties

```
1     spring.application.name=livraria  
2  
3     # Configuração do banco de dados H2  
4     spring.datasource.url=jdbc:h2:file:./DATA/livraria  
5     spring.datasource.username=sa  
6     spring.datasource.password=  
7     spring.h2.console.enabled=true  
8     spring.h2.console.path=/h2-console  
9  
10    spring.jpa.show-sql=true  
11    spring.jpa.properties.hibernate.format_sql=true  
12  
13    # cria as tabelas automaticamente com base nas entidades  
14    # ESTAVA: spring.jpa.hibernate.ddl-auto=create  
15    spring.jpa.hibernate.ddl-auto=update
```



## Pequeno teste...

- Rodei a aplicação, pra ter certeza de ter uma versão final das tabelas no banco.
- Inseri um atributo qualquer numa classe (teste).
- Rodei novamente a aplicação.  
Se funcionou, devemos ter no log apenas um alter table.



# Nenhuma mensagem relacionada ao banco de dados...

```
Run LivrariaApplication x

2025-10-30T11:21:29.509-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:29.521-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:29.786-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:29.796-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:29.797-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:29.823-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:29.823-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:29.938-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:29.963-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:29.978-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:30.115-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:30.133-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:30.233-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:30.234-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:30.263-03:00 INFO 12348 --- [livraria] [
Database JDBC URL [Connecting through datasource 'HikariDataSource (HikariPool-1)']
Database driver: undefined/unknown
Database version: 2.3.232
Autocommit mode: undefined/unknown
Isolation level: undefined/unknown
Minimum pool size: undefined/unknown
Maximum pool size: undefined/unknown
2025-10-30T11:21:30.683-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:30.712-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:30.733-03:00 WARN 12348 --- [livraria] [
2025-10-30T11:21:30.930-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:31.010-03:00 INFO 12348 --- [livraria] [
2025-10-30T11:21:31.021-03:00 INFO 12348 --- [livraria] [
main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode
main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 7 ms. Found 0 repository interfaces
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.46]
main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 6 ms
main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
main] org.hibernate.Version : HHH000412: Hibernate ORM core version 6.6.29.Final
main] o.h.c.internal.RegionFactoryInitiator : HHH000026: Second-level cache disabled
main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA class transformer
main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection conn0: url=jdbc:h2:file:./db/livraria
main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
main] org.hibernate.orm.connections.pooling : HHH10001005: Database info:
main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platform available (set 'hibernate.transaction.platform_class' or use 'platformTransactionManager')
main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit
main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering
main] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:livraria'
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
main] c.livraria.LivrariaApplication : Started LivrariaApplication in 2.304 seconds (process running for 2.304)
```



**INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO**  
Câmpus São Carlos

# Inserção de atributo em ReviewModel

```
@Entity  
@Table(name = "TB REVIEW")  
public class ReviewModel implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private UUID id;  
  
    @Column(nullable = false)  
    private String comment;  
  
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)  
    @OneToOne  
    @JoinColumn(name = "book_id")  
    private BookModel book;  
  
    // atributo só para teste na aula  
    private String teste; ←
```

# Nova execução...

```
2025-10-30T11:32:46.831-03:00  INFO 12786 --- [livraria] [           main]
  Database JDBC URL [Connecting through datasource 'HikariDataSource (Hi
  Database driver: undefined/unknown
  Database version: 2.3.232
  Autocommit mode: undefined/unknown
  Isolation level: undefined/unknown
  Minimum pool size: undefined/unknown
  Maximum pool size: undefined/unknown
2025-10-30T11:32:47.235-03:00  INFO 12786 --- [livraria] [           main]
  Hibernate:
    alter table if exists tb_review
      add column teste varchar(255)
2025-10-30T11:32:47.269-03:00  INFO 12786 --- [livraria] [           main]
2025-10-30T11:32:47.291-03:00  WARN 12786 --- [livraria] [           main]
```



# Nova execução...

```
2025-10-30T11:32:46.831-03:00  INFO 12786 --- [livraria] [           main]
  Database JDBC URL [Connecting through datasource 'HikariDataSource (Hi
  Database driver: undefined/unknown
  Database version: 2.3.232
  Autocommit mode: undefined/unknown
  Isolation level: undefined/unknown
  Minimum pool size: undefined/unknown
  Maximum pool size: undefined/unknown
```

```
2025-10-30T11:32:47.235-03:00  INFO 12786 --- [livraria] [           main]
```

```
Hibernate:
```

```
  alter table if exists tb_review
    add column teste varchar(255)
```

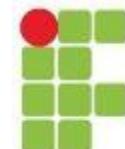
```
2025-10-30T11:32:47.269-03:00  INFO 12786 --- [livraria]
```

```
2025-10-30T11:32:47.291-03:00  WARN 12786 --- [livraria]
```



Entidades e seus relacionamentos  
finalizadas!

Próximo passo:  
**Repositories**



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# BookRepository

```
public interface BookRepository extends JpaRepository<BookModel, UUID> {  
}
```



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# AuthorRepository

```
public interface AuthorRepository extends JpaRepository<AuthorModel, UUID> {  
}
```



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# PublisherRepository

```
public interface PublisherRepository extends JpaRepository<PublisherModel, UUID> {  
}
```



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# ReviewRepository

```
public interface ReviewRepository extends JpaRepository<ReviewModel, UUID> {  
}
```



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

Vamos inserir algumas informações nas tabelas Publisher e Author para criar o livro posteriormente.

Vamos usar o arquivo  
**data.sql**



Vamos inserir algumas informações nas tabelas Publisher e Author para criar o livro posteriormente.

Vamos usar o arquivo  
**data.sql**



## 1 O que é o `data.sql`

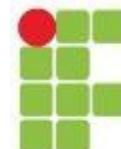
O arquivo `data.sql` é um **script SQL** que o Spring Boot executa automaticamente **após criar as tabelas** (ou seja, depois que o Hibernate gera o schema).

 Ele deve ficar em:

css

 Copiar código

```
src/main/resources/data.sql
```



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos



livraria

Version control



Project



...

- ✓ livraria ~/HD\_PRINCIPAL/Dropbox/intellijProj...
- > .idea
- > .mvn
- ✓ DATA
  - ?] livraria.mv.db
  - ≡ livraria.trace.db
- ✓ src
  - ✓ main
    - ✓ java
      - > carlao2005.livraria
      - > carlao2005.livraria.models
      - > carlao2005.livraria.repositories
    - ✓ resources
      - ✓ static
      - ✓ templates
      - ⚙️ application.properties
    - ≡ data.sql
  - > test

data.sql

1

2



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

## 2 Como ele é executado

Por padrão, o Spring Boot procura automaticamente por arquivos:

- `schema.sql` → cria as tabelas (não é necessário se você usa `@Entity`)
- `data.sql` → insere os dados

👉 Se você usa `spring.jpa.hibernate.ddl-auto=create` ou `update`, o Hibernate cria as tabelas, e logo depois o Spring roda o `data.sql`.



## 3 Exemplo prático para seu caso

Dado o seu `PublisherModel`, seu `data.sql` pode ser:

sql

 Copiar código

```
INSERT INTO tb_publisher (id, name) VALUES (RANDOM_UUID(), 'Alta Books');
INSERT INTO tb_publisher (id, name) VALUES (RANDOM_UUID(), 'Alta Pearson');
```

 Dica: sempre informe as colunas (`id` , `name`) — isso evita erro se a ordem dos campos mudar.



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

### 3 Exemplo prático para seu caso

Dado o seu `PublisherModel`, seu `data.sql` pode ser:

sql

 Copiar código

```
INSERT INTO tb_publisher (id, name) VALUES (RANDOM_UUID(), 'Alta Books');  
INSERT INTO tb_publisher (id, name) VALUES (RANDOM_UUID(), 'Alta Pearson');
```

 Dica: sempre informe as colunas (`id` , `name`) — isso evita erro se a ordem dos campos mudar.

Específico do H2 !

Banco	Função para UUID
PostgreSQL	<code>gen_random_uuid()</code>
H2	<code>RANDOM_UUID()</code>
MySQL	<code>UUID()</code>
SQL Server	<code>NEWID()</code>



Project

- livraria ~/HD\_PRINCIPAL/Dropbox/intelliJProject  
  > .idea  
  > .mvn  
  DATA  
    livraria.mv.db  
    livraria.trace.db  
  src  
    main  
      java  
        carlao2005.livraria  
        carlao2005.livraria.models  
        carlao2005.livraria.repositories  
      resources  
        static  
        templates  
        application.properties  
  data.sql

data.sql

```
1  
2      -- publishers  
3  
4      INSERT INTO TB_PUBLISHER (id, name) VALUES (RANDOM_UUID(), 'Alta Books');  
5      INSERT INTO TB_PUBLISHER (id, name) VALUES (RANDOM_UUID(), 'Alta Pearson');  
6  
7      -- authors  
8  
9      INSERT INTO TB_AUTHOR (id, name) VALUES (RANDOM_UUID(), 'Eric Evans');  
10     INSERT INTO TB_AUTHOR (id, name) VALUES (RANDOM_UUID(), 'Paul Deitel');  
11     INSERT INTO TB_AUTHOR (id, name) VALUES (RANDOM_UUID(), 'Harvey Deitel');  
12  
13 |
```

-- publishers

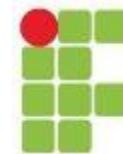
```
INSERT INTO TB_PUBLISHER (id, name) VALUES (RANDOM_UUID(), 'Alta Books');  
INSERT INTO TB_PUBLISHER (id, name) VALUES (RANDOM_UUID(), 'Alta Pearson');
```

-- authors

```
INSERT INTO TB_AUTHOR (id, name) VALUES (RANDOM_UUID(), 'Eric Evans');  
INSERT INTO TB_AUTHOR (id, name) VALUES (RANDOM_UUID(), 'Paul Deitel');  
INSERT INTO TB_AUTHOR (id, name) VALUES (RANDOM_UUID(), 'Harvey Deitel');
```

Executando...

Vendo os dados no banco...



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

booky

Limpar ZAP

WhatsApp Web

Grupo GEN: C...

(2103) FILOS...



jdbc:h2:file:/DATA/livraria

- + TB\_AUTHOR
- + TB\_BOOK
- + TB\_BOOK\_AUTHOR
- + TB\_PUBLISHER
- + TB\_REVIEW
- + INFORMATION\_SCHEMA
- + Users
- i H2 2.3.232 (2024-08-11)**

Run

Run Selected

Auto complete

Clear

SQL statement:

```
SELECT * FROM TB_AUTHOR
```

```
SELECT * FROM TB_AUTHOR;
```

ID	NAME
----	------

(no rows, 3 ms)

Edit



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

booky

Limpar ZAP

WhatsApp Web

Grupo GEN: C...

(2103) FILOS...



Auto commit



Max rows:

1000



Auto complete

jdbc:h2:file:/DATA/livraria

+ TB\_AUTHOR

+ TB\_BOOK

+ TB\_BOOK\_AUTHOR

+ TB\_PUBLISHER

+ TB REVIEW

+ INFORMATION\_SCHEMA

+ Users

i H2 2.3.232 (2024-08-11)

Run

Run Selected

Auto complete

Clear

SQL statement:

SELECT \* FROM TB\_AUTHOR

SELECT \* FROM TB\_AUTHOR;

ID	NAME
----	------

(no rows, 3 ms)

Edit



Precisa alterar o  
application.properties  
para forçar a execução  
do script.

```
# Garante que o data.sql rode após o Hibernate  
spring.jpa.defer-datasource-initialization=true
```

```
# Força execução do data.sql em qualquer ambiente  
spring.sql.init.mode=always
```

```
# Informa explicitamente a plataforma do banco  
spring.sql.init.platform=h2
```



application.properties ×

```
1  spring.application.name=livraria
2
3  # Configuração do banco de dados H2
4  spring.datasource.url=jdbc:h2:file:./DATA/livraria
5  spring.datasource.username=sa
6  spring.datasource.password=
7  spring.h2.console.enabled=true
8  spring.h2.console.path=/h2-console
9
10 spring.jpa.show-sql=true
11 spring.jpa.properties.hibernate.format_sql=true
12
13 # cria as tabelas automaticamente com base nas entidades
14 # ESTAVA: spring.jpa.hibernate.ddl-auto=create
15 spring.jpa.hibernate.ddl-auto=update
16
17 # Garante que o data.sql rode após o Hibernate
18 spring.jpa.defer-datasource-initialization=true
19
20 # Força execução do data.sql em qualquer ambiente
21 spring.sql.init.mode=always
22
23 # Informa explicitamente a plataforma do banco
24 spring.sql.init.platform=h2
```

# Executando novamente...



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos



Auto commit



Max rows: 1000



Auto complete

jdbc:h2:file:./DATA/livraria

+ TB\_AUTHOR

+ TB\_BOOK

+ TB\_BOOK\_AUTHOR

+ TB\_PUBLISHER

+ TB REVIEW

+ INFORMATION\_SCHEMA

+ Users

i H2 2.3.232 (2024-08-11)

Run

Run Selected

Auto complete

Clear

SQL statement:

```
SELECT * FROM TB_AUTHOR
```

```
SELECT * FROM TB_AUTHOR;
```

ID	NAME
e84ef63b-70df-4345-a137-82330f9e5ab0	Eric Evans
0b357215-9303-48b0-b57d-b952ab6e49c5	Paul Deitel
9eba3647-27b6-4795-a7bd-d48816707fde	Harvey Deitel

(3 rows, 5 ms)



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

jdbc:h2:file:./DATA/livraria | Auto commit | Max rows: 1000 | Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM TB_AUTHOR
```

TB\_AUTHOR  
TB\_BOOK  
TB\_BOOK\_AUTHOR  
TB\_PUBLISHER  
TB REVIEW  
INFORMATION\_SCHEMA  
Users  
H2 2.3.232 (2024-08-11)

SELECT	ID
	e84ef63
	0b3572
	9eba36
	(3 rows, 2 ms)

jdbc:h2:file:./DATA/livraria | Auto commit | Max rows: 1000 | Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM TB_PUBLISHER |
```

TB\_AUTHOR  
TB\_BOOK  
TB\_BOOK\_AUTHOR  
TB\_PUBLISHER  
TB REVIEW  
INFORMATION\_SCHEMA  
Users  
H2 2.3.232 (2024-08-11)

SELECT * FROM TB_PUBLISHER;	
ID	NAME
31bdfd3d-9667-4727-b54a-884433663e6d	Alta Books
ba6fa271-7c6b-49ce-b66c-0260a021a071	Alta Pearson
(2 rows, 2 ms)	



| Auto commit | Max rows: 1000 | Run | Run Selected | Auto complete | Clear | SQL statement:

jdbc:h2:file:/DATA/livraria

+ TB\_AUTHOR  
+ TB\_BOOK  
+ TB\_BOOK\_AUTHOR  
+ TB\_PUBLISHER  
+ TB REVIEW  
+ INFORMATION\_SCHEMA  
+ Users

H2 2.3.232 (2024-08-11)

SELECT \* FROM TB\_AUTHOR

SELECT

ID

e84ef63

0b3572

9eba364

(3 rows,



| Auto commit | Max rows: 1000 | Run | Run Selected | Auto complete | Clear | SQL statement:

jdbc:h2:file:/DATA/livraria

+ TB\_AUTHOR  
+ TB\_BOOK  
+ TB\_BOOK\_AUTHOR  
+ TB\_PUBLISHER  
+ TB REVIEW  
+ INFORMATION\_SCHEMA  
+ Users

H2 2.3.232 (2024-08-11)

SELECT \* FROM TB\_PUBLISHER |

SELECT \* FROM TB\_PUBLISHER;

ID	NAME
31bdfd3d-9667-4727-b54a-884433663e6d	Alta Books
ba6fa271-7c6b-49ce-b66c-0260a021a071	Alta Pearson

(2 rows, 2 ms)

Edit



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

Apague (ou comente)  
as últimas alterações que fizemos no  
**application.properties**  
senão toda vez que for rodar a aplicação  
vai tentar inserir os dados no banco,  
e vai dar erro por conta da  
restrição unique.



---

sql application.properties

```
spring.application.name=livraria

# Configuração do banco de dados H2
spring.datasource.url=jdbc:h2:file:./DATA/livraria
spring.datasource.username=sa
spring.datasource.password=
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

# cria as tabelas automaticamente com base nas entidades
# ESTAVA: spring.jpa.hibernate.ddl-auto=create
spring.jpa.hibernate.ddl-auto=update

# Garante que o data.sql rode após o Hibernate
# spring.jpa.defer-datasource-initialization=true

# Força execução do data.sql em qualquer ambiente
# spring.sql.init.mode=always

# Informa explicitamente a plataforma do banco
# spring.sql.init.platform=h2
```



# Criando um livro na base de dados.



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# Requisição no Postman para criar um livro:

HTTP PRW3 aula spring data / **criar livro**

POST  http://localhost:8080/bookstore/books

Params Authorization Headers (8) **Body**  Scripts Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **JSON**

```
1 {  
2     "title"      : "Domain Driven Design",  
3     "publisherId" : "...aqui vai o UUID do publisher... ",  
4     "authorIds"   : [ "...aqui vai uma lista de UUIDs dos autores..." ],  
5     "reviewComment": "Livro muito bom, bla bla bla bla bla."  
6 }  
7
```



# Requisição no Postman para criar um livro:

HTTP PRW3 aula spring data / **criar livro**

POST http://localhost:8080/bookstore/books

Params Authorization Headers (8) **Body** Scripts Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **JSON** ▾

```
1 {  
2     "title"      : "Domain Driven Design",  
3     "publisherId" : "...aqui vai o UUID do publisher... ",  
4     "authorIds"   : [ "...aqui vai uma lista de UUIDs dos autores..." ],  
5     "reviewComment": "Livro muito bom, bla bla bla bla bla."  
6 }  
7
```

Precisamos criar este DTO para receber estes dados



# BookRecordDto (na pasta dtos)

```
public record BookRecordDto(String title,  
                            UUID publisherId,  
                            Set<UUID> authorIds,  
                            String reviewComment) { }
```



# BookRecordDto (na pasta dtos)

```
public record BookRecordDto(String title,  
                            UUID publisherId,  
                            Set<UUID> authorIds,  
                            String reviewComment) { }
```

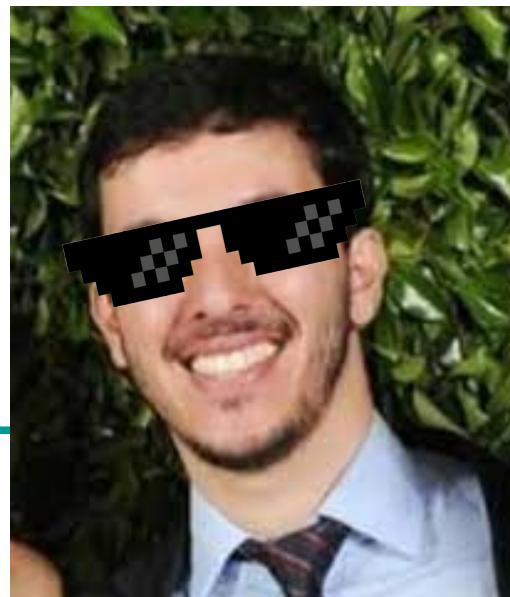
```
{  
    "title"      : "Domain Driven Design",  
    "publisherId": "... aqui vai o UUID do publisher... ",  
    "authorIds"  : [ "... aqui vai uma lista de UUIDs dos autores..." ],  
    "reviewComment": "Livro muito bom, bla bla bla bla bla."  
}
```



Vamos criar a classe de serviço,  
responsável por criar o livro  
no banco.

pasta services  
classe BookService

PS: obrigado Luke,  
pela aula de arquitetura, services, etc...



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

```
@Service
public class BookService {

    private final BookRepository bookRepository;
    private final AuthorRepository authorRepository;
    private final PublisherRepository publisherRepository;

    public BookService(BookRepository bookRepository,
                      AuthorRepository authorRepository,
                      PublisherRepository publisherRepository) {
        this.bookRepository = bookRepository;
        this.authorRepository = authorRepository;
        this.publisherRepository = publisherRepository;
    }

    @Transactional
    public BookModel saveBook(BookRecordDto bookRecordDto) {
        BookModel book = new BookModel();
        book.setTitle(bookRecordDto.title());
        book.setPublisher(publisherRepository.findById(bookRecordDto.publisherId()).get());
        book.setAuthors(
            authorRepository.findAllById(bookRecordDto.authorIds()).stream().collect(Collectors.toSet())
        );

        ReviewModel reviewModel = new ReviewModel();
        reviewModel.setComment(bookRecordDto.reviewComment());
        reviewModel.setBook(book);
        book.setReview(reviewModel);

        return bookRepository.save(book);
    }
}
```

```
@Service
public class BookService {

    private final BookRepository bookRepository;
    private final AuthorRepository authorRepository;
    private final PublisherRepository publisherRepository;

    public BookService(BookRepository bookRepository,
                      AuthorRepository authorRepository,
                      PublisherRepository publisherRepository) {
        this.bookRepository = bookRepository;
        this.authorRepository = authorRepository;
        this.publisherRepository = publisherRepository;
    }

    @Transactional
    public BookModel saveBook(BookRecordDto bookRecordDto) {
        BookModel book = new BookModel();
        book.setTitle(bookRecordDto.title());
        book.setPublisher(publisherRepository.findById(bookRecordDto.publisherId()).get());
        book.setAuthors(
            authorRepository.findAllById(bookRecordDto.authorIds()).stream().collect(Collectors.toSet())
        );
        ReviewModel reviewModel = new ReviewModel();
        reviewModel.setScore(bookRecordDto.score());
        reviewModel.setComment(bookRecordDto.comment());
        book.setReview(reviewModel);
        return bookRepository.save(book);
    }
}
```

## Propósito geral

A classe é um serviço Spring (`@Service`) responsável por orquestrar a criação e persistência de um `BookModel` completo (livro), juntando dados vindos de diferentes repositórios: autores, editora e o próprio repositório de livros. Em outras palavras: recebe um DTO com dados do livro, resolve as entidades relacionadas (publisher, authors), cria uma entidade `ReviewModel` associada ao livro e persiste tudo.

```
@Service  
public class BookService {
```

```
private final BookRepository bookRepository;
private final AuthorRepository authorRepository;
private final PublisherRepository publisherRepository;

public BookService(BookRepository bookRepository,
                  AuthorRepository authorRepository,
                  PublisherRepository publisherRepository) {

    this.bookRepository = bookRepository;
    this.authorRepository = authorRepository;
    this.publisherRepository = publisherRepository;
}
```

```
@Transactional
public BookModel saveBook(BookRecordDto bookRecordDto) {
    BookModel book = new BookModel();
    book.setTitle(bookRecordDto.title());
    book.setPublisher(publisherRepository.findById(bookRecordDto.publisherId()).get());
    book.setAuthors(
        authorRepository.findAllById(bookRecordDto.authorIds()).stream().collect(Collectors.toSet())
    );
    Review review;
    review.setBook(book);
    review.setAuthor(author);
    review.setPublisher(publisher);
    return review;
}
```

## Injeção de dependências e construtor

Os três repositórios necessários (book, author, publisher) são recebidos via injeção pelo construtor. Isso promove testabilidade (é fácil mockar/repor os repositórios em testes) e segue o padrão recomendado de injeção por construtor em Spring.

```
@Service  
public class BookService {
```

```
    private final BookRepository bookRepository;  
    private final AuthorRepository authorRepository;  
    private final PublisherRepository publisherRepository;
```

```
    public BookService(BookRepository bookRepository,  
                      AuthorRepository authorRepository,  
                      PublisherRepository publisherRepository) {
```

```
        this.bookRepository = bookRepository;  
        this.authorRepository = authorRepository;  
        this.publisherRepository = publisherRepository;
```

```
}
```

```
@Transactional
```

```
    public BookModel saveBook(BookRecordDto bookRecordDto) {
```

```
        BookModel book = new BookModel();  
        book.setTitle(bookRecordDto.title());  
        book.setPublisher(publisherRepository.findById(bookRecordDto.publisherId()).get());  
        book.setAuthors(  
            authorRepository.findAllById(bookRecordDto.authorIds()).stream().collect(Collectors.toSet())  
        );
```

```
        ReviewModel reviewModel = new ReviewModel();  
        reviewModel.setC(bookRecordDto.getReviewCount());  
        reviewModel.setB(bookRecordDto.getReviewScore());  
        book.setReview(reviewModel);  
  
        return bookRepository.save(book);
```

## Transacionalidade

O método que persiste o livro está anotado com `@Transactional`. Isso significa que:

- Tudo que acontece dentro do método é executado em uma única transação do JPA.
- Se ocorrer qualquer `RuntimeException` durante a execução, a transação será revertida (rollback) e nada será persistido.
- Objetos recuperados pelo repositório entram no contexto de persistência do Hibernate/JPA, o que facilita atualizações e flush automático ao final da transação.

```
@Service
public class BookService {

    private final BookModel bookModel;
    private final AuthorModel authorModel;
    private final PublisherModel publisherModel;

    public BookService(
        BookRepository bookRepository,
        AuthorRepository authorRepository,
        PublisherRepository publisherRepository) {
        this.bookRepository = bookRepository;
        this.authorRepository = authorRepository;
        this.publisherRepository = publisherRepository;
    }

    @Transactional
    public BookModel saveBook(BookRecordDto bookRecordDto) {
        BookModel book = new BookModel();
        book.setTitle(bookRecordDto.title());
        book.setPublisher(publisherRepository.findById(bookRecordDto.publisherId()).get());
        book.setAuthors(
            authorRepository.findAllById(bookRecordDto.authorIds())
                .stream()
                .collect(Collectors.toSet()));
        ReviewModel reviewModel = new ReviewModel();
        reviewModel.setComment(bookRecordDto.reviewComment());
        reviewModel.setBook(book);
        book.setReview(reviewModel);

        return bookRepository.save(book);
    }
}
```

## Fluxo de execução (o que o método faz, passo a passo)

1. Cria uma nova instância de `BookModel` e atribui o título recebido no DTO.
2. Busca a `PublisherModel` pelo `publisherId` contido no DTO, usando o repositório de publisher, e associa esse publisher ao livro.
3. Recupera os `AuthorModel` correspondentes ao conjunto de `authorIds` do DTO (usando um método que busca por vários ids) e atribui esse `Set` ao livro.
4. Cria uma nova instância de `ReviewModel`, coloca o comentário vindo do DTO, liga o `ReviewModel` ao `BookModel` (setando a referência do lado do review) e então associa essa review ao livro (seta o lado do livro também).
5. Persiste o `BookModel` chamando `bookRepository.save(book)`. Em razão do mapeamento JPA, as associações serão persistidas apropriadamente conforme cascade e propriedade de dono do relacionamento.

```
@Service
public class BookService {

    private final BookRepository bookRepository;
    private final AuthorRepository authorRepository;
    private final PublisherRepository publisherRepository;

    public BookService(BookRepository bookRepository,
                      AuthorRepository authorRepository,
                      PublisherRepository publisherRepository) {
        this.bookRepository = bookRepository;
        this.authorRepository = authorRepository;
        this.publisherRepository = publisherRepository;
    }
}
```

```
@Transactional
public BookModel saveBook(BookRecordDto bookRecordDto) {
    BookModel book = new BookModel();
    book.setTitle(bookRecordDto.title());
    book.setPublisher(publisherRepository.findById(bookRecordDto.publisherId()).get());
    book.setAuthors(
        authorRepository.findAllById(bookRecordDto.authorIds()).stream().collect(Collectors.toSet())
    );
    ReviewModel reviewModel = new ReviewModel();
    reviewModel.setComment(bookRecordDto.comment());
    reviewModel.setBook(book);
    book.setReview(reviewModel);
    return bookRepository.save(book);
}
```

A linha em questão pega os UUIDs de autores do DTO, busca os respectivos AuthorModel no banco e associa esse conjunto ao livro.

O método `findAllById` retorna um `Iterable` (ou seja, uma coleção genérica que pode ser percorrida com `for`), com os autores existentes; ao usar `stream().collect(Collectors.toSet())`, o resultado é transformado em um `Set<AuthorModel>`, removendo duplicatas.

Assim, o `book` passa a ter os autores vinculados corretamente, e ao salvar o `BookModel`, o JPA atualiza a tabela de junção `ManyToMany`.

```
@Service  
public class Books  
  
    private final  
    private final  
    private final  
  
    public BookSer  
  
        this.bookR  
        this.autho  
        this.publi  
    }  
  
}
```

## Comportamento de persistência e mapeamentos relevantes

- A relação OneToOne entre BookModel e ReviewModel tem cascade = CascadeType.ALL no lado do livro com mappedBy = "book". Isso garante que, ao salvar o BookModel, o ReviewModel recém-criado seja automaticamente persistido junto (insert) sem necessidade de salvá-lo separadamente.
- A relação ManyToMany entre livro e autores tem o BookModel como lado "dono" (definindo a @JoinTable) — portanto, ao salvar o livro com um conjunto de authors existentes, as entradas na tabela de junção serão criadas/atualizadas.

```
@Transactional  
public BookModel saveBook(BookRecordDto bookRecordDto) {  
  
    BookModel book = new BookModel();  
    book.setTitle(bookRecordDto.title());  
    book.setPublisher(publisherRepository.findById(bookRecordDto.publisherId()).get());  
    book.setAuthors(  
        authorRepository.findAllById(bookRecordDto.authorIds()).stream().collect(Collectors.toSet())  
    );  
  
    ReviewModel reviewModel = new ReviewModel();  
    reviewModel.setComment(bookRecordDto.reviewComment());  
    reviewModel.setBook(book);  
    book.setReview(reviewModel);  
  
    return bookRepository.save(book);  
}  
  
}
```

```
@Service
public class BookService {

    private final BookRepository bookRepository;
    private final AuthorRepository authorRepository;
    private final PublisherRepository publisherRepository;

    public BookService(BookRepository bookRepository,
                      AuthorRepository authorRepository,
                      PublisherRepository publisherRepository) {
        this.bookRepository = bookRepository;
        this.authorRepository = authorRepository;
        this.publisherRepository = publisherRepository;
    }
}
```

```
@Transactional
public BookModel saveBook(BookRecordDto bookRecordDto) {
    BookModel book = new BookModel();
    book.setTitle(bookRecordDto.title());
    book.setPublisher(publisherRepository.findById(bookRecordDto.publisherId()).get());
    book.setAuthors(
        authorRepository.findAllById(bookRecordDto.authorIds()).stream().collect(Collectors.toSet())
    );

    ReviewModel reviewModel = new ReviewModel();
    reviewModel.setComment(bookRecordDto.reviewComment());
    reviewModel.setBook(book);
    book.setReview(reviewModel);

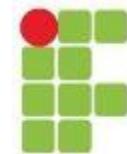
    return bookRepository.save(book);
}
```

Estão faltando códigos importantes para realizar checagens que evitam problemas e erros.

Isso foi feito propositalmente, devido ao caráter demonstrativo desta aula, cujo objetivo é mostrar os conceitos básicos e principais do tema apresentado.

Por exemplo, o método `findById` retorna um `Optional`. Não tratá-lo antes de chamar `.get()` pode gerar exceções se o valor não existir.

Finalmente,  
o controller que vai receber a requisição  
e chamar o método para  
persistir o livro na base.



# controllers/BookController

```
@RestController
@RequestMapping("/bookstore/books")
public class BookController {

    private final BookService bookService;

    public BookController(BookService bookService) {
        this.bookService = bookService;
    }

    @PostMapping
    public ResponseEntity<BookModel> saveBook(@RequestBody BookRecordDto bookRecordDto) {

        return ResponseEntity.status(HttpStatus.CREATED)
            .body(bookService.saveBook(bookRecordDto));
    }
}
```

# controllers/BookController

```
@RestController
@RequestMapping("/bookstore/books")
public class BookController {

    private final BookService bookService;

    public BookController(BookService bookService) {
        this.bookService = bookService;
    }

    @PostMapping
    public ResponseEntity<BookModel> saveBook(@RequestBody BookRecordDto bookRecordDto) {

        return ResponseEntity.status(HttpStatus.CREATED)
            .body(bookService.saveBook(bookRecordDto));
    }
}
```

## Visão geral

A classe `BookController` é um **controlador REST** do Spring.

Ela expõe **endpoints HTTP** relacionados à entidade `BookModel`, permitindo que clientes (como front-ends ou ferramentas de teste como Postman) enviem requisições para criar livros.

O papel do controller é **receber a requisição, validar os dados recebidos, e repassar a lógica de negócios para o serviço** (`BookService`).

# controllers/BookController

```
@RestController
@RequestMapping("/bookstore/books")
public class BookController {

    private final BookService bookService;

    public BookController(BookService bookService) {
        this.bookService = bookService;
    }

    @PostMapping
    public ResponseEntity<BookModel> saveBook(@RequestBody BookRecordDto bookRecordDto) {

        return ResponseEntity.status(HttpStatus.CREATED)
            .body(bookService.saveBook(bookRecordDto));
    }
}
```

## Anotações e mapeamentos

- `@RestController` :  
Indica que esta classe é um controlador REST — ou seja, seus métodos retornam dados diretamente no corpo da resposta (geralmente em JSON), e não páginas HTML.  
Internamente, é uma combinação de `@Controller` e `@ResponseBody`.
- `@RequestMapping("/bookstore/books")` :  
Define o prefixo de URL para todos os endpoints deste controlador.  
Assim, o caminho base será `/bookstore/books`.  
Por exemplo, um `POST` será enviado para `http://localhost:8080/bookstore/books`.

# controllers/BookController

```
@RestController
@RequestMapping("/bookstore/books")
public class BookController {

    private final BookService bookService;

    public BookController(BookService bookService) {
        this.bookService = bookService;
    }

    @PostMapping
    public ResponseEntity<BookModel> saveBook(@RequestBody BookRecordDto bookRecordDto) {

        return ResponseEntity.status(HttpStatus.CREATED)
            .body(bookService.saveBook(bookRecordDto));
    }
}
```

## Injeção de dependência

O controller recebe uma instância de `BookService` pelo **construtor**.

O Spring injeta automaticamente esse serviço, pois ele foi anotado com `@Service`.

Isso permite que o controlador use o serviço sem precisar criar o objeto manualmente.

# controllers/BookController

```
@RestController
@RequestMapping("/bookstore/books")
public class BookController {

    private final BookService bookService;

    public BookController(BookService bookService) { ... }

    @PostMapping
    public ResponseEntity<BookModel> saveBook(@RequestBody BookRecordDto bookRecordDto) {

        return ...
    }
}
```

**Método principal:** `saveBook`

- `@PostMapping`:

Mapeia requisições HTTP `POST` para o método.

Indica que o método será chamado quando o cliente fizer um `POST` para `/bookstore/books`.

- **Parâmetro** `@RequestBody BookRecordDto bookRecordDto`:

Indica que o corpo da requisição (em JSON) será convertido automaticamente em um objeto `BookRecordDto`.

Exemplo de corpo JSON esperado:

json

 Copiar código

```
{
    "title": "Clean Code",
    "publisherId": "c2f1e1b4-1c4d-4d7e-93a7-93bb7a4ed5f2",
    "authorIds": ["d4f2e8b1-5b4c-42a7-84f3-88b6dc7f07e1"],
    "reviewComment": "Excelente livro sobre boas práticas."
}
```

# controllers/BookController

```
@RestController
@RequestMapping("/bookstore/books")
public class BookController {

    private final BookService bookService;

    public BookController(BookService bookService) {
        this.bookService = bookService;
    }

    @PostMapping
    public ResponseEntity<BookModel> saveBook(@RequestBody BookRecordDto bookRecordDto) {

        return ResponseEntity.status(HttpStatus.CREATED)
            .body(bookService.saveBook(bookRecordDto));
    }
}
```

- **Chamada ao serviço:**

O método chama `bookService.saveBook(bookRecordDto)`, que cria e persiste um novo `BookModel` no banco.

Esse método retorna a entidade `BookModel` recém-criada.

- **`ResponseEntity.status(HttpStatus.CREATED)` :**

Indica que a resposta HTTP terá o status **201 Created**, que é o código correto para operações de criação bem-sucedidas.

- **`.body( ... )` :**

Define o corpo da resposta HTTP, que conterá o objeto `BookModel` retornado pelo serviço (automaticamente convertido em JSON).

## Resumo do fluxo da requisição

1. O cliente envia um **POST** para `/bookstore/books` com os dados do novo livro em JSON.
  2. O Spring converte o JSON em um `BookRecordDto`.
  3. O controlador repassa o DTO para o serviço (`bookService.saveBook`).
  4. O serviço cria as entidades (`BookModel`, `ReviewModel`, etc.) e as salva no banco.
  5. O `BookModel` salvo é retornado ao controller.
  6. O controller devolve uma resposta HTTP **201 (Created)** com o livro recém-criado em JSON no corpo.
- 

## Em resumo

O `BookController` funciona como **ponte entre o mundo externo (requisições HTTP) e a lógica de negócio (serviço)**.

Ele não contém regras de negócio, apenas **recebe dados, chama o serviço apropriado e retorna a resposta adequada**.



# Testando...



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

POST  http://localhost:8080/bookstore/books

Params Authorization Headers (8) Body  Scripts Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL  JSON

```

1 {
2     "title"          : "Domain Driven Design",
3     "publisherId"   : "31bdfd3d-9667-4727-b54a-884433663e6d",
4     "authorIds"     : [ "e84ef63b-70df-4345-a137-82330f9e5ab0" ],
5     "reviewComment" : "Livro muito bom, bla bla bla bla bla."
6 }
7

```

SELECT \* FROM TB\_PUBLISHER;

ID	NAME
31bdfd3d-9667-4727-b54a-884433663e6d	Alta Books
ba6fa271-7c6b-49ce-b66c-0260a021a071	Alta Pearson

(2 rows, 5 ms)

SELECT \* FROM TB\_AUTHOR;

ID	NAME
e84ef63b-70df-4345-a137-82330f9e5ab0	Eric Evans
0b357215-9303-48b0-b57d-b952ab6e49c5	Paul Deitel
9eba3647-27b6-4795-a7bd-d48816707fde	Harvey Deitel

(3 rows, 2 ms)

```
{
    "title"          : "Domain Driven Design",
    "publisherId"   : "31bdfd3d-9667-4727-b54a-884433663e6d",
    "authorIds"     : [ "e84ef63b-70df-4345-a137-82330f9e5ab0" ],
    "reviewComment" : "Livro muito bom, bla bla bla bla bla."
}
```



{ } JSON ▾

▷ Preview  Visualize ▾

```
1  {
2      "id": "03c07b05-c73b-4df5-a988-d9d6fbba5b24",
3      "title": "Domain Driven Design",
4      "publisher": {
5          "id": "31bdfd3d-9667-4727-b54a-884433663e6d",
6          "name": "Alta Books"
7      },
8      "authors": [
9          {
10             "id": "e84ef63b-70df-4345-a137-82330f9e5ab0",
11             "name": "Eric Evans"
12         }
13     ],
14     "review": {
15         "id": "24cf8d98-527a-47d2-9c26-480e967e2b1b",
16         "comment": "Livro muito bom, bla bla bla bla bla."
17     }
18 }
```



{ } JSON ▾ ▶ Preview Visualize ▾

```
1 {  
2     "id": "03c07b05-c73b-4df5-a988-d9d6fbba5b24",  
3     "title": "Domain Driven Design",  
4     "publisher": {  
5         "id": "31bdfd3d-9667-4727-b54a-884433663e6d",  
6         "name": "Alta Books"  
7     },  
8     "authors": [  
9         {  
10            "id": "e84ef63b-70df-4345-a137-82330f9e5ab0",  
11            "name": "Eric Evans"  
12        }  
13    ],  
14    "review": {  
15        "id": "24cf8d98-527a-47d2-9c26-480e967e2b1b",  
16        "comment": "Livro muito bom, bla bla bla bla bla."  
17    }  
18 }
```

SELECT \* FROM TB\_BOOK;

ID	TITLE	PUBLISHER_ID
03c07b05-c73b-4df5-a988-d9d6fbba5b24	Domain Driven Design	31bdfd3d-9667-4727-b54a-884433663e6d

(1 row, 1 ms)

SELECT \* FROM TB\_BOOK\_AUTHOR;

BOOK_ID	AUTHOR_ID
03c07b05-c73b-4df5-a988-d9d6fbba5b24	e84ef63b-70df-4345-a137-82330f9e5ab0

(1 row, 1 ms)

SELECT \* FROM TB\_REVIEW;

ID	COMMENT	TESTE	BOOK_ID
24cf8d98-527a-47d2-9c26-480e967e2b1b	Livro muito bom, bla bla bla bla bla.	null	03c07b05-c73b-4df5-a988-d9d6fbba5b24

(1 row, 2 ms)

{ } JSON ▾ ▶ Preview Visualize ▾

```
1 {  
2     "id": "03c07b05-c73b-4df5-a988-d9d6fbba5b24",  
3     "title": "Domain Driven Design",  
4     "publisher": {  
5         "id": "31bdfd3d-9667-4727-b54a-884433663e6d",  
6         "name": "Alta Books"  
7     },  
8     "authors": [  
9         {  
10            "id": "e84ef63b-70df-4345-a137-82330f9e5ab0",  
11            "name": "Eric Evans"  
12        }  
13    ],  
14    "review": {  
15        "id": "24cf8d98-527a-47d2-9c26-480e967e2b1b",  
16        "comment": "Livro muito bom, bla bla bla bla bla."  
17    }  
18 }
```

SELECT \* FROM TB\_BOOK;

ID	TITLE	PUBLISHER_ID
03c07b05-c73b-4df5-a988-d9d6fbba5b24	Domain Driven Design	31bdfd3d-9667-4727-b54a-884433663e6d

(1 row, 1 ms)

SELECT \* FROM TB\_BOOK\_AUTHOR;

BOOK_ID	AUTHOR_ID
03c07b05-c73b-4df5-a988-d9d6fbba5b24	e84ef63b-70df-4345-a137-82330f9e5ab0

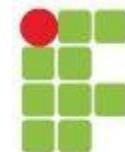
(1 row, 1 ms)



	TESTE	BOOK_ID
Livro muito bom, bla bla bla bla bla bla.	null	03c07b05-c73b-4df5-a988-d9d6fbba5b24

Service:  
método para listar os livros.

Controller:  
trata a requisição para  
retornar a lista de livros.



# Método inserido na classe BookService:

```
public List<BookModel> getAllBooks() {  
    return bookRepository.findAll();  
}
```



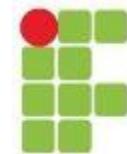
INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# Método inserido na classe BookController:

```
@GetMapping  
public ResponseEntity<List<BookModel>> getAllBooks() {  
    return ResponseEntity.status(HttpStatus.OK).body(bookService.getAllBooks());  
}
```



# Testando...



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

GET



http://localhost:8080/bookstore/books

Params Authorization Headers (6) Body Scripts Tests Settings

200 OK

- none
- form-data
- x-www-form-urlencoded
- raw
- binary
- GraphQL

This request

Body Cookies Headers (5) Test Results |

{ } JSON ▾ ▶ Preview Visualize ▾

```
1  [
2   {
3     "id": "03c07b05-c73b-4df5-a988-d9d6fbba5b24",
4     "title": "Domain Driven Design",
5     "publisher": {
6       "id": "31bdfd3d-9667-4727-b54a-884433663e6d",
7       "name": "Alta Books"
8     },
9     "authors": [
10    {
11      "id": "e84ef63b-70df-4345-a137-82330f9e5ab0",
12      "name": "Eric Evans"
13    }
14  ],
15  "review": {
16    "id": "24cf8d98-527a-47d2-9c26-480e967e2b1b",
17    "comment": "Livro muito bom, bla bla bla bla bla."
18  }
19 }
20 ]
```



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# Experimento

- Vamos configurar algumas relações, em BookModel, como “preguiçosas” ( FetchType.LAZY ).
- O que vai mudar no retorno da requisição ?



# Classe BookModel:

- ERA:

```
@ManyToOne  
@JoinColumn(name = "publisher_id")  
private PublisherModel publisher;  
  
@ManyToMany  
@JoinTable(  
    name = "tb_book_author",  
    joinColumns = @JoinColumn(name = "book_id"),  
    inverseJoinColumns = @JoinColumn(name = "author_id"))  
private Set<AuthorModel> authors = new HashSet<>();
```

- FICOU:

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)  
@ManyToOne(fetch = FetchType.LAZY)  
@JoinColumn(name = "publisher_id")  
private PublisherModel publisher;
```

```
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)  
@ManyToMany(fetch = FetchType.LAZY)  
@JoinTable(  
    name = "tb_book_author",  
    joinColumns = @JoinColumn(name = "book_id"),  
    inverseJoinColumns = @JoinColumn(name = "author_id"))  
private Set<AuthorModel> authors = new HashSet<>();
```



# Refazendo a requisição...



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

GET | http://localhost:8080/bookstore/books

Params Authorization Headers (6) Body Scripts Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL

This request

Body Cookies Headers (5) Test Results | ⏪

{ } JSON ▾ ▶ Preview 🖼 Visualize ▾

```
1 [  
2 {  
3     "id": "03c07b05-c73b-4df5-a988-d9d6fbba5b24",  
4     "title": "Domain Driven Design",  
5     "publisher": {  
6         "id": "31bdfd3d-9667-4727-b54a",  
7         "name": "Alta Books"  
8     },  
9     "authors": [  
10        {  
11            "id": "e84ef63b-70df-4345-",  
12            "name": "Eric Evans"  
13        },  
14    ],  
15    "review": {  
16        "id": "24cf8d98-527a-47d2-9c26",  
17        "comment": "Livro muito bom, b  
18    }  
19 }  
20 ]
```

GET

| http://localhost:8080/bookstore/books

Params Authorization Headers (6) Body Scripts Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL

Body Cookies Headers (5) Test Results | ⏪

{ } JSON ▾ ▶ Preview 🖼 Visualize ▾

```
1 [  
2 {  
3     "id": "03c07b05-c73b-4df5-a988-d9d6fbba5b24",  
4     "title": "Domain Driven Design",  
5     "review": {  
6         "id": "24cf8d98-527a-47d2-9c26-480e967e2b1b",  
7         "comment": "Livro muito bom, bla bla bla bla bla."  
8     }  
9 }  
10 ]
```

GET | http://localhost:8080/bookstore/books

Params Authorization Headers (6) Body Scripts Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL

This request

Body Cookies Headers (5) Test Results | ⏺

{ } JSON ▾ ▶ Preview 🖥 Visualize ▾

```
1 [  
2 {  
3     "id": "03c07b05-c73b-4df5-a988-d9d6fbba5b24",  
4     "title": "Domain Driven Design",  
5     "publisher": {  
6         "id": "31bd1d3d-9667-4727-b54a",  
7         "name": "Alta Books"  
8     },  
9     "authors": [  
10        {  
11            "id": "e84ef63b-70df-4345",  
12            "name": "Eric Evans"  
13        },  
14    ],  
15    "review": {  
16        "id": "24cf8d98-527a-47d2-9c26",  
17        "comment": "Livro muito bom, b  
18    }  
19 }]  
20 ]
```

GET

| http://localhost:8080/bookstore/books

Params Authorization Headers (6) Body Scripts Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL

Body Cookies Headers (5) Test Results | ⏺

{ } JSON ▾ ▶ Preview 🖥 Visualize ▾

```
1 [  
2 {  
3     "id": "03c07b05-c73b-4df5-a988-d9d6fbba5b24",  
4     "title": "Domain Driven Design",  
5     "review": {  
6         "id": "24cf8d98-527a-47d2-9c26-480e967e2b1b",  
7         "comment": "Livro muito bom, bla bla bla bla bla."  
8     }  
9 }]  
10 ]
```

## 1. O que você mudou

Você fez duas alterações principais:

1. Adicionou `@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)` nas propriedades `publisher` e `authors`.
2. Definiu `fetch = FetchType.LAZY` nessas mesmas relações (`@ManyToOne` e `@ManyToMany`).

Essas mudanças **não alteram a persistência no banco de dados**, mas mudam o comportamento da API e do carregamento dos dados em memória.



## 2. Efeito da anotação `@JsonProperty(access = WRITE_ONLY)`

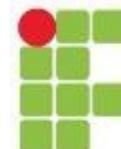
Essa anotação vem da biblioteca **Jackson**, que é responsável por converter objetos Java em JSON (e vice-versa).

- `WRITE_ONLY` significa que o campo será usado apenas na entrada (quando receber dados JSON), mas não será incluído na saída (quando o objeto for convertido em JSON para resposta).

### Na prática:

Quando você fizer um **POST** enviando um novo livro, o JSON pode conter `publisherId` e `authorIds` (via DTO), e tudo funcionará normalmente.

Mas quando fizer um **GET** para listar os livros (`findAll()` ou `/bookstore/books`), o JSON não mostrará os campos `publisher` nem `authors` dentro de cada livro.



## Exemplo prático:

Antes das mudanças, a resposta de um livro poderia ser algo assim:

json

```
{  
  "id": "1234",  
  "title": "Clean Code",  
  "publisher": {  
    "id": "987",  
    "name": "Pearson"  
  },  
  "authors": [  
    { "id": "555", "name": "Robert C. Martin" }  
  ],  
  "review": { "id": "333", "comment": "Excelente livro!" }  
}
```

 Copiar código

Depois das mudanças, o mesmo endpoint retornará algo assim:

json

```
{  
  "id": "1234",  
  "title": "Clean Code",  
  "review": { "id": "333", "comment": "Excelente livro!" }  
}
```

 Copiar código



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

## Exemplo prático:

Antes das mudanças, a resposta de um livro poderia ser algo assim:

json

Copiar código

```
{  
  "id": "1234",  
  "title": "Clean Code",  
  "publisher": {  
    "id": "987",  
    "name": "Pearson"  
  },  
  "authors": [  
    { "id": "555", "name": "Robert C. Martin" }  
  ],  
  "review": { "id": "333", "comment": "Excelente livro!" }  
}
```

Depois das mudanças, o mesmo endpoint retornará algo assim:

json

Copiar código

```
{  
  "id": "1234",  
  "title": "Clean Code",  
  "review": { "id": "333",  
 }
```

Ou seja:

- `publisher` e `authors` não aparecem mais no JSON de saída.
- Esses campos continuam existindo no banco e são carregados internamente, mas **não são enviados ao cliente**.

Esse comportamento evita **recursão infinita** (loop de serialização entre entidades relacionadas) e **respostas JSON muito grandes**, o que é uma prática comum em APIs REST.

### 3. Efeito do `fetch = FetchType.LAZY`

Por padrão:

- `@ManyToOne` é **EAGER** (carrega o objeto relacionado imediatamente).
- `@ManyToMany` é **LAZY** (só carrega quando acessado).

Você tornou **ambas as relações LAZY**, o que significa:

- O JPA **não carrega automaticamente** `publisher` e `authors` quando você busca um livro.
- Esse dados só serão buscados no banco **se forem acessados no código** (por exemplo, ao chamar `book.getAuthors()`).

👉 Isso melhora o desempenho em listagens (carrega apenas os dados do livro em si), mas, se você tentar acessar `book.getPublisher()` **fora de uma transação aberta**, pode ocorrer um `LazyInitializationException` — porque o Hibernate já fechou a sessão e não consegue mais buscar os dados atrasados.



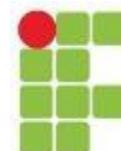
## ④ 4. O que muda na resposta do endpoint /bookstore/books

Quando você fizer uma requisição `GET` para listar livros:

- O resultado JSON conterá apenas os campos simples (`id`, `title`, `review`, etc.).
- As informações de autores e editora não virão no JSON (por causa do `WRITE_ONLY`).
- A consulta ao banco será mais leve, pois, devido ao `fetch = LAZY`, o Hibernate não fará joins automáticos com as tabelas `TB_AUTHOR` e `TB_PUBLISHER`.

Em resumo:

- **Antes:** JSON completo, com publisher e authors aninhados, e queries mais pesadas.
- **Depois:** JSON mais leve, sem publisher nem authors, e queries mais rápidas.



E pra fechar...

Apagar um livro.



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

Service:  
método para apagar um livro do banco.

Controller:  
trata a requisição para  
apagar um livro.



# Método inserido na classe BookService:

```
@Transactional  
public void deleteBook(UUID id){  
    bookRepository.deleteById(id);  
}
```



# Método inserido na classe BookController:

```
@DeleteMapping("/{id}")
public ResponseEntity<String> deleteBook(@PathVariable UUID id) {
    bookService.deleteBook(id);
    return ResponseEntity.status(HttpStatus.OK).body("Book deleted successfully.");
}
```



# Testando...



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

**DELETE**

▼

<http://localhost:8080/bookstore/books/03c07b05-c73b-4df5-a988-d9d6fbba5b24>[Params](#)[Authorization](#)[Headers \(8\)](#)[Body](#) •[Scripts](#)[Tests](#)[Settings](#)

### Query Params

	Key	Value
	Key	Value

[Body](#)[Cookies](#)[Headers \(5\)](#)[Test Results](#) [Raw](#) ▾ [Preview](#) [Visualize](#) ▾

1 Book deleted successfully.



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# Enviando novamente a requisição que devolve a lista de livros:

GET <http://localhost:8080/bookstore/books>

Params Authorization Headers (6) Body Scripts Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL

This request does not have a body

Body Cookies Headers (5) Test Results | 41 200 OK

{ } JSON ▾ ▶ Preview  Visualize ▾

1 [ ]



# Olhando as tabelas no banco...



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# Review (ligação direta com o livro)

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM TB REVIEW |
```

SELECT \* FROM TB REVIEW;

ID	COMMENT	TESTE	BOOK_ID

(no rows, 5 ms)

[Edit](#)



# Tabela auxiliar livro x autor

Run

Run Selected

Auto complete

Clear

SQL statement:

```
SELECT * FROM TB_BOOK_AUTHOR |
```

```
SELECT * FROM TB_BOOK_AUTHOR;
```

BOOK\_ID

AUTHOR\_ID

(no rows, 2 ms)

Edit



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# E, finalmente, a tabela de livros:

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM TB_BOOK
```

SELECT \* FROM TB\_BOOK;

ID	TITLE	PUBLISHER_ID
----	-------	--------------

(no rows, 1 ms)

Edit



# Último teste... Inserir um livro com 2 autores.

Retirei (comentei) aqueles últimos “lazy” que colocamos em BookModel, para poder ver todos os dados quando pedirmos a lista geral.

```
//@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@ManyToOne//(fetch = FetchType.LAZY)
@JoinColumn(name = "publisher_id")
private PublisherModel publisher;

//@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
@ManyToMany//(fetch = FetchType.LAZY)
@JoinTable(
    name = "tb_book_author",
    joinColumns = @JoinColumn(name = "book_id"),
    inverseJoinColumns = @JoinColumn(name = "author_id"))
private Set<AuthorModel> authors = new HashSet<>();
```



POST



<http://localhost:8080/bookstore/books>

Params   Authorization   Headers (8)   **Body** • Scripts   Tests   Settings

none    form-data    x-www-form-urlencoded    raw    binary    GraphQL   **JSON**

```
1 {  
2   "title"      : "Java: Como Programar",  
3   "publisherId" : "ba6fa271-7c6b-49ce-b66c-0260a021a071",  
4   "authorIds"   : [ "0b357215-9303-48b0-b57d-b952ab6e49c5" , "9eba3647-27b6-4795-a7bd-d48816707fde" ],  
5   "reviewComment": "Livro muito completo."  
6 }  
7
```



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

POST



http://localhost:8080/bookstore/books

Params Authorization Headers (8) Body Scripts Tests Settings

 none  form-data  x-www-form-urlencoded  raw  binary  GraphQL

JSON

```
1 {  
2   "title"      : "Java: Como Prog  
3   "publisherId": "ba6fa271-7c6b-4  
4   "authorIds"  : [ "0b357215-9303  
5   "reviewComment": "Livro muito com  
6 }  
7
```

Body Cookies Headers (5) Test Results

201 Created

{} JSON Preview Visualize

```
1 {  
2   "id": "a277d7ac-616f-4048-8fa8-1459626dca52",  
3   "title": "Java: Como Programar",  
4   "publisher": {  
5     "id": "ba6fa271-7c6b-49ce-b66c-0260a021a071",  
6     "name": "Alta Pearson"  
7   },  
8   "authors": [  
9     {  
10    "id": "0b357215-9303-48b0-b57d-b952ab6e49c5",  
11    "name": "Paul Deitel"  
12  },  
13  {  
14    "id": "9eba3647-27b6-4795-a7bd-d48816707fde",  
15    "name": "Harvey Deitel"  
16  }  
17 ],  
18   "review": {  
19     "id": "1543ed70-83db-4dc2-b342-fdf08fa21109",  
20     "comment": "Livro muito completo."  
21   }  
22 }
```

# Requisição que busca a lista completa:

Body Cookies Headers (5) Test Results ⏱ 200 OK • 2

{ } JSON ▾ ▶ Preview Visualize ▾

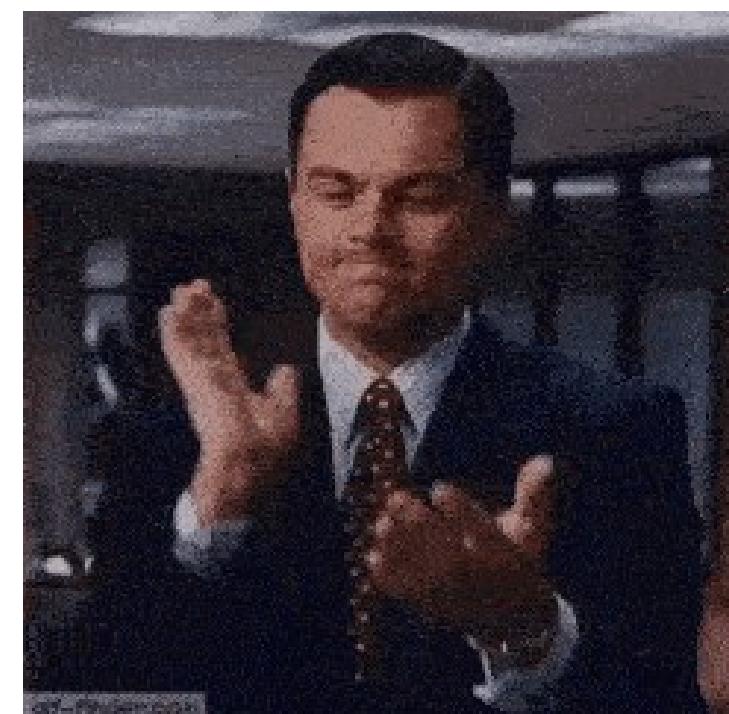
```
1 [  
2 {  
3     "id": "a277d7ac-616f-4048-8fa8-1459626dca52",  
4     "title": "Java: Como Programar",  
5     "publisher": {  
6         "id": "ba6fa271-7c6b-49ce-b66c-0260a021a071",  
7         "name": "Alta Pearson"  
8     },  
9     "authors": [  
10        {  
11            "id": "0b357215-9303-48b0-b57d-b952ab6e49c5",  
12            "name": "Paul Deitel"  
13        },  
14        {  
15            "id": "9eba3647-27b6-4795-a7bd-d48816707fde",  
16            "name": "Harvey Deitel"  
17        }  
18    ],  
19    "review": {  
20        "id": "1543ed70-83db-4dc2-b342-fdf08fa21109",  
21        "comment": "Livro muito completo."  
22    }  
23}  
24]
```

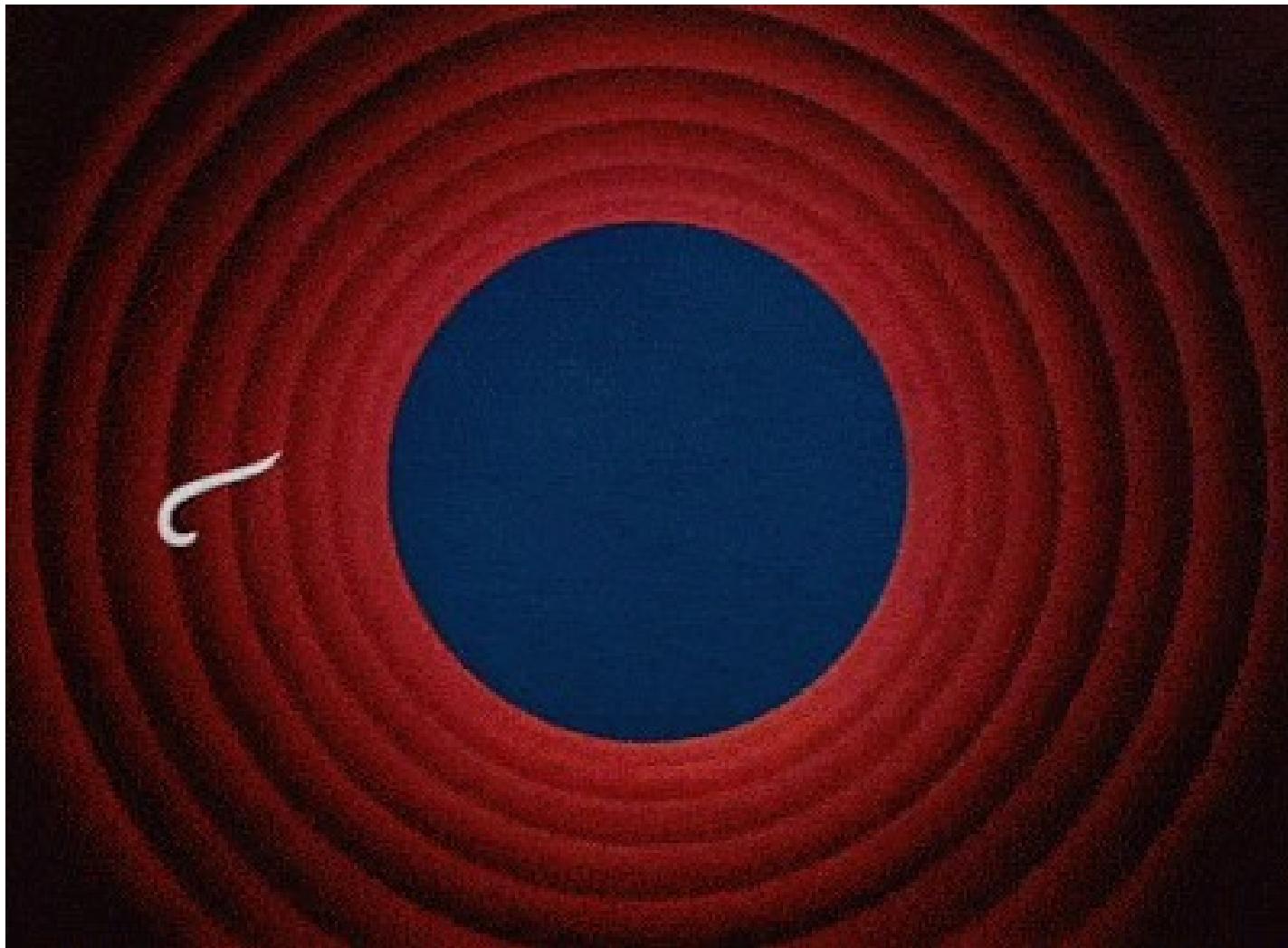
# Requisição que busca a lista completa:

Body Cookies Headers (5) Test Results ⏱ 200 OK • 2

{ } JSON ▾ ▶ Preview Visualize ▾

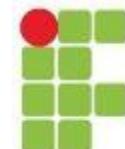
```
1 [  
2 {  
3     "id": "a277d7ac-616f-4048-8fa8-1459626dca52",  
4     "title": "Java: Como Programar",  
5     "publisher": {  
6         "id": "ba6fa271-7c6b-49ce-b66c-0260a021a071",  
7         "name": "Alta Pearson"  
8     },  
9     "authors": [  
10        {  
11            "id": "0b357215-9303-48b0-b57d-b952ab6e49c5",  
12            "name": "Paul Deitel"  
13        },  
14        {  
15            "id": "9eba3647-27b6-4795-a7bd-d48816707fde",  
16            "name": "Harvey Deitel"  
17        }  
18    ],  
19    "review": {  
20        "id": "1543ed70-83db-4dc2-b342-fdf08fa21109",  
21        "comment": "Livro muito completo."  
22    }  
23}  
24]
```





INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

Referência:  
Canal da @MichelliBrito  
no YouTube



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

Projeto finalizado disponível no github.

[https://github.com/carlaopereirasanca/prw3\\_springdata](https://github.com/carlaopereirasanca/prw3_springdata)



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos

# Exercícios



# Exercícios

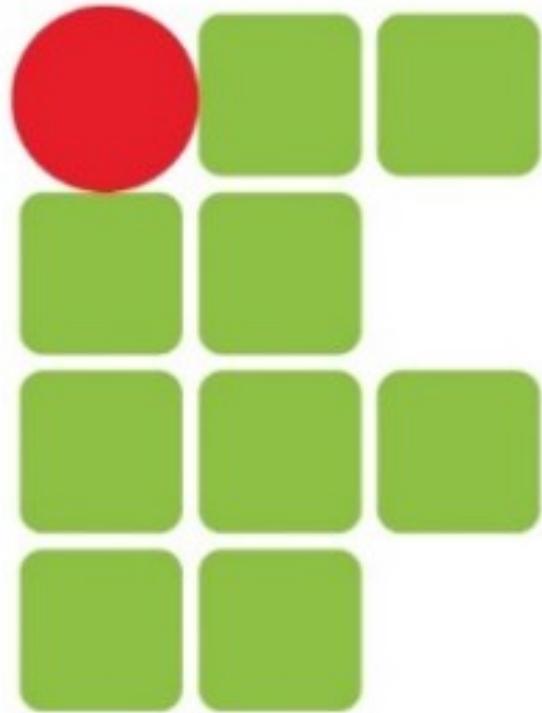
- Recrie o projeto e experimente com os códigos vistos na aula de hoje...



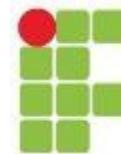
INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos



**INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos**



**INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Câmpus São Carlos**