Auditoria automática em entidades JPA com Spring Boot e MySQL

A rastreabilidade de dados é uma exigência fundamental em aplicações corporativas modernas.

É indispensável registrar **quem criou**, **quem modificou**, **quando** e **como** determinada informação foi manipulada no sistema.

Além de atender exigências legais e de compliance, esse tipo de controle melhora a **transparência**, **segurança** e **confiabilidade** dos dados.

Neste artigo, demonstraremos como implementar **auditoria automática em entidades JPA** utilizando o ecossistema Spring Boot e persistência com **MySQL 8**.

O objetivo é registrar alterações em entidades sem a necessidade de lógica manual repetitiva, aproveitando os recursos do **Spring Data JPA**, **Aspect-Oriented Programming (AOP)** e **Flyway** para versionamento de schema.

A solução será implementada de forma genérica e reutilizável.

Requisitos da solução

Para garantir a compatibilidade com o Spring Boot 3 e os padrões atuais de desenvolvimento, recomenda-se o seguinte stack:

Tecnologia	Versão recomendada	Justificativa
JDK	17 (LTS)	Compatível com Spring Boot 3.x; traz melhorias de performance, segurança e novas construções de linguagem
Spring Boot	3.x	Versão moderna com suporte ao Jakarta EE 9+
Spring Data JPA	Incluso no Spring Boot	Camada de persistência reativa à mudança
MySQL	8.0 ou superior	Suporte aprimorado a TIMESTAMP(3), JSON e performance
Flyway	Última estável	Controle de versionamento de banco relacional
Maven	3.8+	Sistema de build e gerenciamento de dependências
IDE	IntelliJ / Eclipse / VS Code	Com suporte a Spring Boot e Java 17

1. Adicionando campos de auditoria à entidade JPA

A primeira etapa consiste em enriquecer a entidade Professor com os metadados necessários para rastreamento de ações. Abaixo os campos e anotações:

```
@Column(name = "created_by", nullable = false, updatable = false)
@CreatedBy
private String createdBy;

@Column(name = "created_date", nullable = false, updatable = false)
@CreatedDate
private Instant createdDate;

@Column(name = "modified_by")
@LastModifiedBy
private String modifiedBy;

@Column(name = "modified_date")
@LastModifiedDate
private Instant modifiedDate;
```

Detalhamento técnico:

- @CreatedBy / @LastModifiedBy: atribuem automaticamente o **usuário** responsável pela criação ou modificação da entidade.
- @CreatedDate / @LastModifiedDate: armazenam o **timestamp** do evento.
- updatable = false: impede que os campos sejam modificados por código após a persistência.
- Os tipos utilizados (String e Instant) são suportados pelo JPA e compatíveis com bancos como o MySQL 8.

Essas anotações são processadas pelo AuditingEntityListener, que será configurado nas próximas etapas.

2. Atualizando a estrutura do banco de dados com Flyway e MySQL

Crie um script de migração Flyway (ex: V2__adiciona_campos_auditoria_professor.sql) com a seguinte instrução SQL:

```
ALTER TABLE `professor`
ADD COLUMN `created_by` VARCHAR(50) NOT NULL DEFAULT 'Sistema' AFTER
`professor_curriculo`,
```

```
ADD COLUMN `created_date` TIMESTAMP(3) NOT NULL DEFAULT
CURRENT_TIMESTAMP(3) AFTER `created_by`,
ADD COLUMN `modified_by` VARCHAR(50) NULL AFTER `created_date`,
ADD COLUMN `modified_date` TIMESTAMP(3) NULL DEFAULT NULL ON UPDATE
CURRENT_TIMESTAMP(3) AFTER `modified_by`;
```

Por que essas definições?

- VARCHAR(50): tamanho adequado para nomes de usuários.
- TIMESTAMP(3): permite precisão em milissegundos.
- DEFAULT CURRENT_TIMESTAMP(3): insere automaticamente o horário de criação.
- ON UPDATE: o campo é atualizado automaticamente pelo MySQL ao modificar o registro.

Flyway executará esse script automaticamente no próximo startup da aplicação, aplicando a alteração ao schema versionado.

3. Ativando o listener de auditoria na entidade

Para que o JPA saiba que a entidade Professor deve ser monitorada quanto a eventos de persistência e atualização, usamos:

```
@EntityListeners(AuditingEntityListener.class)
```

Esse listener faz parte do Spring Data JPA e é responsável por invocar os métodos de auditoria definidos por @CreatedBy, @LastModifiedBy, etc.

4. Incluindo a dependência de aspectos do Spring (AOP)

No arquivo pom.xml, adicione a dependência abaixo para habilitar recursos de interceptação de chamadas e manipulação automatizada de eventos:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aspects</artifactId>
</dependency>
```

O **Spring AOP** é fundamental para que as anotações de auditoria funcionem corretamente, pois a lógica depende de interceptadores declarativos.

5. Criando a configuração de auditoria com @EnableJpaAuditing

Crie uma classe de configuração que habilite o módulo de auditoria no contexto da aplicação:

```
@Configuration
@EnableJpaAuditing(auditorAwareRef = "auditorProvider")
@EnableAspectJAutoProxy
public class AuditConfig {

    @Bean
    public AuditorAware<String> auditorProvider() {
        return new AuditorAwareImpl();
    }
}
```

Explicação das anotações:

- @EnableJpaAuditing: ativa a auditoria nas entidades JPA com base nas anotações.
- auditorAwareRef: define o bean que fornecerá o nome do auditor (usuário responsável).
- @EnableAspectJAutoProxy: ativa o suporte interno a AOP para funcionamento de spring-aspects.

6. Implementando a estratégia de obtenção do usuário (AuditorAware)

Cenário sem autenticação (fixo):

```
public class AuditorAwareImpl implements AuditorAware<String> {

    @Override
    @NonNull
    public Optional<String> getCurrentAuditor() {
        return Optional.of("Sistema"); // valor fixo
    }
}
```

Cenário com autenticação via Spring Security:

```
public class SpringSecurityAuditorAware implements AuditorAware<String> {

    @Override
    public Optional<String> getCurrentAuditor() {
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        if (auth == null || !auth.isAuthenticated()) {
            return Optional.empty();
```

```
        }
        return Optional.ofNullable(auth.getName());
    }
}
```

Essas classes implementam a interface AuditorAware<T>, responsável por fornecer dinamicamente o auditor atual (geralmente um usuário autenticado).

A implementação de **auditoria automática com Spring Boot, JPA e MySQL** é uma solução robusta, reutilizável e altamente recomendada para qualquer aplicação que demande governança de dados.

Com apenas algumas anotações e configurações, é possível garantir a rastreabilidade total das operações de escrita em qualquer entidade JPA.

Além disso, a escolha do **JDK 17** como base do projeto assegura compatibilidade com o ecossistema moderno do Spring, promovendo melhor performance e segurança.

O uso do **Flyway** garante versionamento controlado do schema, fundamental para ambientes produtivos.

Com essa abordagem, desenvolvedores podem atender requisitos técnicos e regulatórios sem duplicação de lógica, mantendo o código limpo e sustentável.

EducaCiência FastCode para a comunidade