**Spring Data JPA Auditing: Automatically Saving the Good Stuff**

Auditing provides valuable information, but it can be a nightmare to implement. Fortunately, through Spring Data JPA, you can persist the columns you need.

In any business application, auditing simply means tracking and logging every change we do in our persisted records, which simply means tracking every insert, update, and delete operation and storing it.

Auditing helps us in maintaining history records, which can later help us in tracking user activities. If implemented properly, auditing can also provide us similar functionality to version control systems.

I have seen projects storing these things manually. Doing so becomes very complex because you will need to write it completely on your own, which will definitely require lots of code — and lots of code means less maintainability and less focus on writing business logic.

But why should someone need to go to this path when both JPA and Hibernate provide automatic auditing, which we can be easily configured in your project?

Here in this article, I will discuss how we can configure JPA to automatically persist the CreatedBy, CreatedDate, LastModifiedBy, and LastModifiedDate columns for any entity.

[](https://2.bp.blogspot.com/--T30IEy6Z2k/WJ3yIskMf5I/AAAAAAAAKgc/f__nHPvpP8c1Q2CsLLA3YcA4kbiMGRVtACK4B/s1600/Spring-Data-JPA-Automatic_Auditing.png)

I will walk you through to the necessary steps and code that you will need to include in your project to automatically update these properties. We will use Spring Boot, Spring Data JPA, and MySQL to demonstrate this. We will need to add the following parent and dependencies to our POM file:

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.1.RELEASE</version>

<relativePath/>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

**Spring Data Annotations @CreatedBy, @CreatedDate, @LastModifiedBy, and @LastModifiedDate**

Let’s suppose we have a File entity, and a single record in the file table stores the name and content of the file. And say we also want to store who created and modified any file at any given time. So, the goal is to keep track of when the file was created, by whom, and when it was last modified, and by whom.

We will need to add the name, content, createdBy, createdDate, lastModifiedBy, and lastModifiedDate properties to our File entity and, to make it more appropriate, we can move the createdBy, createdDate, lastModifiedBy, lastModifiedDate properties to a base class, Auditable, and annotate this base class with**@MappedSuperClass**. Later, we can use the Auditable class in other audited entities.

Both classes will look like this:

@MappedSuperclass

@EntityListeners(AuditingEntityListener.class)

public abstract class Auditable<U> {

@CreatedBy

protected U createdBy;

@CreatedDate

@Temporal(TIMESTAMP)

protected Date creationDate;

@LastModifiedBy

protected U lastModifiedBy;

@LastModifiedDate

@Temporal(TIMESTAMP)

protected Date lastModifiedDate;

}

@Entity

public class File extends Auditable<String> {

@Id

@GeneratedValue

private int id;

private String name;

private String content;

}

As you can see above, I have used the @CreatedBy, @CreatedDate, @LastModifiedBy, and @LastModifiedDate annotation on their respective fields.

The Spring Data JPA approach abstracts working with JPA callbacks and provides us these fancy annotations to automatically save and update auditing entities.

**Using the AuditingEntityListener Class With @EntityListeners**

Spring Data JPA provides a JPA entity listener class,**AuditingEntityListener**, which contains the callback methods (annotated with the **@PrePersist**and **@PreUpdate**annotations), which will be used to persist and update these properties when we will persist or update our entity.

JPA provides the **@EntityListeners** annotation to specify callback listener classes, which we use to register our AuditingEntityListener class.

However, we can also define our own callback listener classes if we want to and specify them using the @EntityListeners annotation. In my next article, I will demonstrate how we can use @EntityListeners to store audit logs.

**Auditing Author Using AuditorAware and Spring Security**

JPA can analyze createdDate and lastModifiedDate using current system time, but what about the createdBy and lastModifiedBy fields? How will JPA recognize what to store in them?

To tell JPA about currently logged-in users, we will need to provide an implementation of**AuditorAware**and override the getCurrentAuditor() method. And inside getCurrentAuditor(), we will need to fetch a currently logged-in user.

As of now, I have provided a hard coded user, but if you are using Spring Security, then use it to find the currently logged-in user.

public class AuditorAwareImpl implements AuditorAware<String> {

@Override

public String getCurrentAuditor() {

return "Naresh";

}

}

**Enable JPA Auditing by Using @EnableJpaAuditing**

We will need to create a bean of type AuditorAware and will also need to enable JPA auditing by specifying **@EnableJpaAuditing**on one of our configuration classes. @EnableJpaAuditing accepts one argument, auditorAwareRef, where we need to pass the name of the AuditorAware bean.

@Configuration

@EnableJpaAuditing(auditorAwareRef = "auditorAware")

public class JpaConfig {

@Bean

public AuditorAware<String> auditorAware() {

return new AuditorAwareImpl();

}

}

And now, if we will try to persist or update a file object, the CreatedBy, CreatedDate, LastModifiedBy, LastModifiedDate properties will automatically get saved.

In the next article, JPA Auditing: Persisting Audit Logs Automatically using EntityListeners, I will discuss how we can use JPA EntityListeners to create audit logs and generate history records for every insert, update, and delete operation.