## Design Document: Audit Trail Implementation with Spring Data Envers for Employee Table (SQL Server)

### Table of Contents

1. **Overview**
2. **Design Approach**
3. **Database Schema (DDL)**
4. **Spring Boot & Envers Integration**
5. **Entities & Repositories (Code Snippets)**
6. **Auditing Data Retrieval & Usage**
7. **Testing & Verification**
8. **Generic Implementation Steps**
9. **Appendix: DDL Scripts**

## 1. Overview

This document describes the design, code, and implementation steps for adding audit trail functionality for an Employee table using [Spring Data Envers](https://spring.io/projects/spring-data-envers) with SQL Server as the backend. Standard auto-DDL features are **not** used; DBA will create tables manually.

## 2. Design Approach

* **Objective:** Track changes, additions, deletions to the employee table for compliance and debugging.
* **Solution:** Use Spring Data Envers to automatically record entity revisions.
* **Database:** Explicit DDL scripts will be provided for the Employee, REVINFO, and Employee\_AUD tables.

## 3. Database Schema (DDL)

Below are DDL scripts for SQL Server. The DBA must execute these before deployment.

### 3.1 Base Table: Employee

CREATE TABLE [dbo].[Employee] (  
 [id] INT IDENTITY PRIMARY KEY,  
 [first\_name] NVARCHAR(64) NOT NULL,  
 [last\_name] NVARCHAR(64) NOT NULL,  
 [email] NVARCHAR(128) NOT NULL,  
 [department] NVARCHAR(64),  
 [hire\_date] DATETIME  
);

### 3.2 Audit Revision Table: REVINFO

CREATE TABLE [dbo].[REVINFO] (  
 [REV] INT IDENTITY PRIMARY KEY,  
 [REVTSTMP] BIGINT NOT NULL  
);

### 3.3 Audit Table: Employee\_AUD

CREATE TABLE [dbo].[Employee\_AUD] (  
 [id] INT NOT NULL,  
 [REV] INT NOT NULL,  
 [REVTYPE] TINYINT NOT NULL,  
 [first\_name] NVARCHAR(64) NULL,  
 [last\_name] NVARCHAR(64) NULL,  
 [email] NVARCHAR(128) NULL,  
 [department] NVARCHAR(64) NULL,  
 [hire\_date] DATETIME NULL,  
 CONSTRAINT PK\_Employee\_AUD PRIMARY KEY ([id], [REV]),  
 CONSTRAINT FK\_Employee\_AUD\_REV FOREIGN KEY ([REV]) REFERENCES [dbo].[REVINFO]([REV])  
);

*Further audit tables required for each audited entity; see “Generic Steps”.*

## 4. Spring Boot & Envers Integration

**Dependencies:**

Add these to your Maven pom.xml:

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>  
<dependency>  
 <groupId>org.hibernate</groupId>  
 <artifactId>hibernate-envers</artifactId>  
</dependency>  
<dependency>  
 <groupId>com.microsoft.sqlserver</groupId>  
 <artifactId>mssql-jdbc</artifactId>  
</dependency>

**application.properties:**

spring.datasource.url=jdbc:sqlserver://<server>:<port>;databaseName=<db>  
spring.datasource.username=<username>  
spring.datasource.password=<password>  
spring.jpa.hibernate.ddl-auto=none  
spring.jpa.properties.hibernate.hbm2ddl.auto=none  
spring.jpa.properties.org.hibernate.envers.audit\_table\_suffix=\_AUD  
spring.jpa.properties.org.hibernate.envers.revision\_field\_name=REV  
spring.jpa.properties.org.hibernate.envers.revision\_type\_field\_name=REVTYPE  
spring.jpa.properties.org.hibernate.envers.revision\_on\_collection\_change=true

## 5. Entities & Repositories (Code Snippets)

### 5.1 Employee Entity

import javax.persistence.\*;  
import org.hibernate.envers.Audited;  
  
@Entity  
@Audited  
@Table(name = "Employee")  
public class Employee {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Integer id;  
  
 @Column(length = 64, nullable = false)  
 private String firstName;  
  
 @Column(length = 64, nullable = false)  
 private String lastName;  
  
 @Column(length = 128, nullable = false)  
 private String email;  
  
 @Column(length = 64)  
 private String department;  
  
 private java.time.LocalDateTime hireDate;  
  
 // Getters, Setters, Constructors  
}

### 5.2 Employee Repository

import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {  
 // Standard CRUD  
}

### 5.3 Revision Entity (Optional)

If you need details about the revision itself, you can define:

import org.hibernate.envers.RevisionEntity;  
import org.hibernate.envers.RevisionNumber;  
import org.hibernate.envers.RevisionTimestamp;  
  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@RevisionEntity  
@Table(name = "REVINFO")  
public class CustomRevisionEntity {  
 @Id  
 @RevisionNumber  
 private int id;  
  
 @RevisionTimestamp  
 private long timestamp;  
  
 // Additional fields as needed  
}

## 6. Auditing Data Retrieval & Usage

To retrieve all historical revisions for an employee:

import org.hibernate.envers.AuditReaderFactory;  
import org.hibernate.envers.AuditReader;  
import javax.persistence.EntityManager;  
import org.springframework.beans.factory.annotation.Autowired;  
  
@Service  
public class EmployeeAuditService {  
 @Autowired  
 private EntityManager entityManager;  
  
 public List<Employee> getEmployeeHistory(Integer employeeId) {  
 AuditReader auditReader = AuditReaderFactory.get(entityManager);  
 List<Number> revisions = auditReader.getRevisions(Employee.class, employeeId);  
  
 List<Employee> auditHistory = new ArrayList<>();  
 for (Number rev : revisions) {  
 Employee emp = auditReader.find(Employee.class, employeeId, rev);  
 auditHistory.add(emp);  
 }  
 return auditHistory;  
 }  
}

## 7. Testing & Verification

* Insert, update, and delete employees via repository/service calls.
* Query the Employee\_AUD and REVINFO tables to verify audit data is written.
* Use the provided service method to view audit history.

## 8. Generic Implementation Steps

To apply Spring Data Envers auditing to any table/entity:

1. **Design Entity**: Standard JPA entity mapped to your business table.
2. **Annotate Entity**: Add @Audited at class level.
3. **DDL for Base Table**: DBA must create the normal table.
4. **DDL for Audit Table**: DBA must create <ENTITY>\_AUD table mirroring the entity fields plus [REV], [REVTYPE] fields, and PK on [id], [REV].
5. **DDL for Revision Table**: DBA must create REVINFO table as above.
6. **Spring Boot Config**: Add Envers + JPA dependencies, configure Hibernate properties as described.
7. **Repository**: Use standard JPA repository.
8. **Audit Reader**: Use Hibernate Envers API to access history.
9. **Deploy, Test**: Populate data, verify audit trail in <ENTITY>\_AUD and REVINFO.

## 9. Appendix: Sample DDL Scripts

(include for DBA; see section 3 above)

## Conclusion

This document ensures a compliant, maintainable, and repeatable audit trail solution with Spring Data Envers. The generic steps enable easy extension to any entity. SQL Server DDLs avoid auto-ddl pitfalls. Share this with developers and DBAs for consistent implementation.

**End of Document**