mischok
better. software_

Datenbank-Auditierung mit Spring Data JPA und Envers

9. Juli 2026 – Java Forum Stuttgart

Julius Mischok

Als Engineering Manager löse ich mit Software-Teams komplexe Projektherausforderungen.

Diplom Mathematiker (univ.)

Spring Boot Experte

Speaker & Autor

Geschäftsführer Mischok GmbH

Was ist Auditierung und warum benötigen wir es?

Datenbank Auditierung sichert alle Versionen von Entities

Warum benötigen wir Datenbank Auditierung?

mischok
better. software_

Weil wir auditieren müssen

Zum Beispiel um Anforderungen der GoBD zu erfüllen.

Grundsätze zur ordnungsmäßigen Führung und Aufbewahrung von Büchern, Aufzeichnungen und Unterlagen in elektronischer Form sowie zum Datenzugriff

mischok
better. software_

Weil wir auditieren wollen

Es macht keinen Spaß, Änderungen in Logfiles nachzuvollziehen…

JPA Entity auditieren

Beispielprojekt: Taskliste

`https://gitlab.mischok.de/open/tasklist`

Einfacher REST Service, um Aufgaben zu verwalten. Aufgaben und Änderungen daran sollen versioniert werden.

Tech Stack: Plain Spring Boot 4

JPA Entity auditieren

Envers speichert alle (oder einen Teil) der Eigenschaften einer JPA Entity bei jeder Änderung in einer Audit-Tabelle.

Schritt 1: Envers einbinden

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.hibernate.orm</groupId>
    <artifactId>hibernate-envers</artifactId>
</dependency>
```

Schritt 2: Entity auditieren

```
@Entity
@Table(name = "task")
@Audited
public class Task {
    …
}
```

Schritt 3: Audit-Tabelle anlegen

Warum gibt es hier keinen Fremdschlüssel?

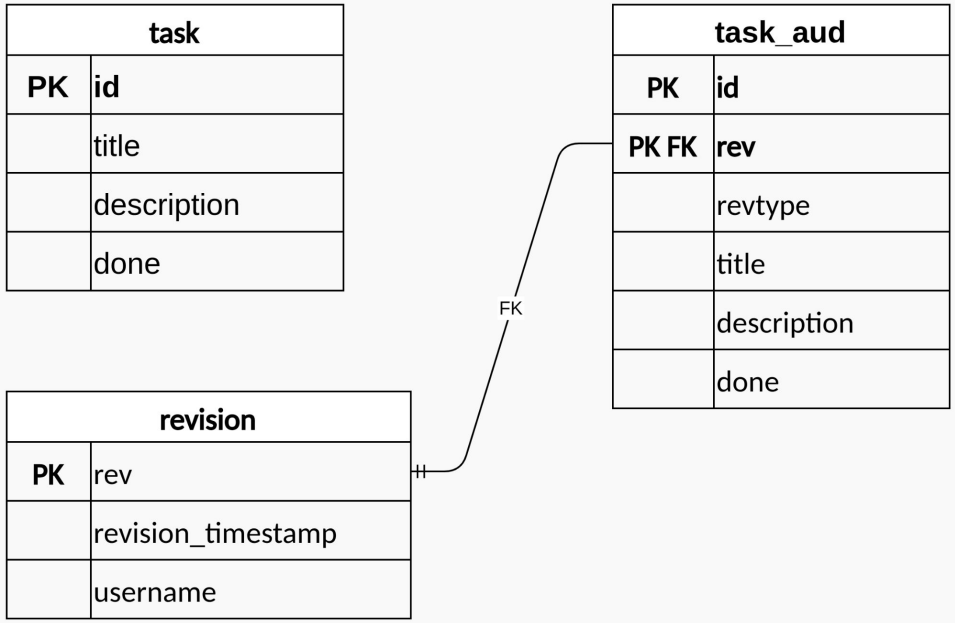
```
CREATE TABLE task_aud (
    id BIGINT NOT NULL,
    title VARCHAR(30),
    description VARCHAR(200),
    done BOOLEAN,
    rev BIGINT NOT NULL,
    revtype SMALLINT,
    PRIMARY KEY (id, rev),
    FOREIGN KEY (rev) REFERENCES revision(rev)
);
```

Revisionsinformationen anreichern

Revisionsinformationen anreichern

Neben den Eigenschaften der Entity legt Envers für jede Version einen Revisionseintrag an. Dieser enthält mindestens die Revisionsnummer und einen Zeitstempel, kann aber durch beliebige weitere Felder angereichert werden.

Datenbankschema



Schritt 1: Revisionstabelle anlegen/erweitern

```
CREATE TABLE revision (
    rev BIGSERIAL PRIMARY KEY,
    revision_timestamp BIGINT NOT NULL,
    username VARCHAR(255)
);
```

Schritt 2: Revision Entity anlegen

```
@Entity
@Table(name = "revision")
@RevisionEntity(CustomRevisionListener.class)
public class CustomRevisionEntity implements Serializable {

    ...

    private String username;
}
```

Schritt 3: Revision Listener anlegen

```
@Override
public void newRevision(Object revisionEntity) {
    CustomRevisionEntity customRevisionEntity =
            (CustomRevisionEntity) revisionEntity;

    // ... getting user from SecurityContext ...

    customRevisionEntity.setUsername(userName);
}
```

Revisionen abfragen

Revisionen abfragen

Der AuditReader von Envers bietet komfortablen Zugriff auf die Revisionsinformationen.

Schritt 1: Information zu bestimmter Revision abfragen

```
public Optional<Task> getTaskAtRevision(Long taskId,
                                        Number revision) {

    AuditReader auditReader = AuditReaderFactory
                                  .get(entityManager);

    Task task = auditReader.find(Task.class,                    !!!
                                  taskId, revision);

    return Optional.ofNullable(task);
}
```

Schritt 2: Audit Trail abfragen

```
List<Object[]> results = auditReader.createQuery()
                .forRevisionsOfEntity(Task.class, false, true)
                .add(AuditEntity.id().eq(taskId))
                .getResultList();
```

AuditReader API

Die API des AuditReaders unterstützt zahlreiche Möglichkeiten, um Revisionen nach unterschiedlichen Kriterien abzufragen.

Stolpersteine

Testing

Hibernate legt die Revisionsinformationen in der before-commit Phase an. Daher muss die Datenbank-Transaktion zwingend geschrieben werden, um die Revisionsinformationen zu persistieren.

Tests ohne @Transactional => Manueller Cleanup!

Default Verhalten exactMatch

Envers verwendet in AuditReader Queries per Default bei einer Suche nach Revisionsnummer das Maximum der bekannten Revisionen

Abhilfe:

```
spring.jpa.properties.org.hibernate.envers.find_by_revision_exact_match=true
```

AuditReader Connection Leak

Der AuditReader darf nicht in einer Spring Bean gespeichert werden, sonst droht ein Connection Leak.

DSGVO Löschung

Bei DSGVO-konformer Lösung die Audit-Tabellen beachten!

Ausblick

Änderungsüberwachung auf Feldebene

Annotation @Audited(withModifiedFlag = true) speichert in der Audit-Tabelle nicht nur jedes Feld, sondern auch noch einen Boolean, ob sich der Wert in der Revision geändert hat.

Änderungsüberwachung auf Feldebene

```
CREATE TABLE project_booking_aud (
    id INT NOT NULL,
    rev INT NOT NULL,
    ...
    booked_hours INT,
    note VARCHAR,
    ...
    booked_hours_mod BOOLEAN,
    note_mod BOOLEAN,
    PRIMARY KEY (id, rev),
    ...
);
```

Key Takeaways

Key Takeaways

- Envers bietet einen leichtgewichtigen Einstieg in die Historisierung mit Hibernate/JPA.
- Das Framework kümmert sich im Hintergrund um die Historisierung. Keine Vermischung mit der Geschäftslogik.
- Wesentlicher Trade-Off: Änderungen im Schema müssen synchron in den Audit-Tabellen nachgezogen werden.

Engineering Essentials

Zwei konzentrierte Stunden zu einer aktuellen Technologie.

Online, in kleinen Gruppen, mit direkter Interaktion, auf dein Level abgestimmt.

23.07.2026
Spring Data JPA & Envers

24.09.2026
JPA Performance Optimierung

Gutscheincode: JFS26

Vielen Dank!

Kontakt:

julius.mischok@mischok.de
https://www.linkedin.com/in/julius-mischok/

mischok
better. software_

Gaming icons created by Freepik - Flaticon:
https://www.flaticon.com/free-icons/gaming