SpringBoot

# Auditing in SpringBoot (spring data envers)

3. Annotate entity classes with

@Audited  
@EntityListeners(AuditingEntityListener.**class**)

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
@Entity  
@Table(name = **"employee"**)  
@Accessors(chain = **true**)  
@Audited  
@EntityListeners(AuditingEntityListener.**class**)  
**public class** Employee **extends** BaseEntity{  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 **private** Long **id**;  
   
 @Column   
 **private** String **name**;  
  
 @Column   
 **private** Double **salary**;  
  
 @Column   
 **private** String **city**;  
  
 @OneToOne(fetch = FetchType.***LAZY***,cascade = CascadeType.***ALL***)   
 **private** Account **account**;  
  
 @OneToMany(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***, orphanRemoval = **true**)  
 @JoinColumn(name = **"employee\_id"**)  
 @NotAudited  
 **private** List<Document> **documentList**;  
  
}

Here

@Audited - When you annotate a JPA entity class with **@Audited**, Hibernate Envers will automatically track changes to instances of that entity, allowing you to maintain a history of modifications over time.

@EntityListeners - These listeners can be useful for tasks like auditing, validation, or custom behavior that needs to be triggered in response to entity state changes

**@NotAudited :** If you dont want the relationship to be audited mark it as *org.hibernate.envers.NotAudited*

4. Extend your Repository with RevisionRepository

**public interface** EmployeeRepository **extends** RevisionRepository<Employee, Long, Long>, JpaRepository<Employee, Long>, EmployeeRepositoryCustom,  
 QuerydslPredicateExecutor<Employee>,  
 QuerydslBinderCustomizer<QEmployee> {  
  
 @Override  
 **default void** customize(QuerydslBindings bindings, QEmployee root) {  
 bindings.bind(String.**class**).first((StringPath path, String value) -> path.containsIgnoreCase(value));  
 }  
}

*RevisionRepository has following methods to get Version history*

Certainly, here are the common methods you might find in a **RevisionRepository** for querying and retrieving historical data using Hibernate Envers, presented in a table format:

| **Method Signature** | **Description** |
| --- | --- |
| **findRevision(Number revision)** | Retrieves a specific revision. |
| **findRevisions(Class<T> entityClass, ID id)** | Retrieves all revisions for a specific entity with the given ID. |
| **findRevisions(Class<T> entityClass)** | Retrieves all revisions for a specific entity class. |
| **findRevisionsBetween(Class<T> entityClass, fromDate, toDate)** | Retrieves all revisions for a specific entity class that occurred between two given timestamps. |
| **findRevisionsByEntityIdAndRevisionNumber(Class<T> entityClass, ID id, Number revision)** | Retrieves a specific revision of an entity with the given ID. |

That’it . once you ran the application you can see both entity tables & audit tables get created in the DB.

Getting History API Example

//Controller

@ApiOperation(**"Get Employee Version History By Id"**)  
@GetMapping(**"/{id}/history"**)  
**public** List<EmployeeDto> getEmployeeHistoryById(@PathVariable(**"id"**) Long id, Pageable pageable) {  
 **return employeeService**.getEmployeeHistoryById(id, pageable);  
}

//service

@Override  
**public** List<EmployeeDto> getEmployeeHistoryById(Long id, Pageable pageRequest) {  
 **if** (!**employeeRepository**.findById(id).isPresent()) {  
 **throw new** EntityNotFoundException(Employee.**class**, id);  
 }  
 List<EmployeeDto> employeeHistoryDtoList = **null**;  
 **try** {  
 Pageable pageable = PageRequest.*of*(pageRequest.getPageNumber(), pageRequest.getPageSize(), RevisionSort.*desc*());  
 Page<Revision<Long, Employee>> employeeRevisions = **employeeRepository**.findRevisions(id, pageable);  
  
 employeeHistoryDtoList = employeeRevisions.stream().map((p) ->  
 **employeeMapper**.toDto(p.getEntity())  
 ).collect(Collectors.*toList*());  
  
 } **catch** (DataAccessException ex) {  
 ex.printStackTrace();  
 }  
 **return** employeeHistoryDtoList;  
}

//Repository

**public interface** EmployeeRepository **extends** RevisionRepository<Employee, Long, Long>, JpaRepository<Employee, Long>, EmployeeRepositoryCustom,  
 QuerydslPredicateExecutor<Employee>,  
 QuerydslBinderCustomizer<QEmployee> {  
  
   
 **default void** customize(QuerydslBindings bindings, QEmployee root) {  
 bindings.bind(String.**class**).first((StringPath path, String value) -> path.containsIgnoreCase(value));  
 }  
}

Errors

org.springframework.dao.InvalidDataAccessApiUsageException: Service is not yet initialized; nested exception is java.lang.IllegalStateException: Service is not yet initialized

* Make sure audit tables are created in DB or not
* Make sure **integration.envers.enabled**: true should be TRUE