JPA Auditing

💡 애플리케이션을 구현할 때 가장 일반적인 요구사항 중 하나는 **변경사항 추적**이다.

> 변경 사항 추적을 위해서는 데이터의 생성자와 생성 시간, 수정자와 수정 시간, 시점별 변경 이력을 기록해야 함

> **JPA를 선택했으나 MyBatis와의 차이점도 함께 검토해보았다 !!!**

✨MyBatis / Spring MVC 방식의 이력 관리

- AOP를 사용하여 데이터 변경 시점을 가로채서 이력을 저장
- Mapper에서 직접 히스토리 테이블 관련 쿼리를 작성
  - Insert, Update, Delete 쿼리를 실행할 때마다 히스토리 테이블에 Insert하는 쿼리를 같이 실행 하도록 함
- 각 테이블마다 별도의 이력 테이블 생성 및 관리

이로 인해 생기는 문제점

- 히스토리 테이블의 기본키(PK) 설정 오류로 인한 데이터 중복/누락 문제
- DBMS별로 다른 쿼리 문법을 처리해야 하는 번거로움
- UAT(사용자 수용 테스트) 단계에서 뒤늦게 발견되는 문제들
    - 히스토리 테이블에 데이터가 정상적으로 저장되는지 확인하는 과정에서 중복 데이터가 발견됨
    - 대량의 이력 데이터 조회 시 성능 저하 발생

✨JPA Auditing은 왜 사용할까?

장점

감시 추적(Audit Tracking)

- 각 엔티티의 생성과 수정 이벤트를 추적 이를 통해 언제, 누가, 어떤 데이터를 변경했는지 기록할 수 있다.

자동화된 날짜 및 사용자 정보 관리

- 생성일자, 수정일자, 생성자, 수정자 등의 필드를 자동으로 관리하며 각 엔티티에서 이러한 정보를 수동으로 관리할 필요가 없어 중복과 오류 가능성이 줄어든다.

일관성 유지

- BaseEntity 클래스에 감사 관련 로직을 집중화하면 모든 엔티티가 이를 공통으로 사용할 수 있고, 각 엔티티마다 감사 로직을 반복 작성할 필요 없이 일관성을 유지할 수 있다.

규정 준수 및 보안 강화

- 로그는 시스템 내 의도치 않은 행위를 감지하고 대응하는 데도 도움이 된다.

유지보수 및 확장성 향상

- BaseEntity의 감사 로직은 이를 상속받는 모든 엔티티가 공유하며 로직의 변경이나 확장 시 BaseEntity만 수정하면 되어 유지보수와 확장이 용이하다.

단점

복잡성 증가

- Auditing 구현을 위해 BaseEntity나 유사 클래스를 만들고 엔티티들이 이를 상속받아야 하며 이는 클래스 계층을 복잡하게 만들 수 있으며, 작은 프로젝트에서는 이런 구조가 불필요할 수 있다.

JPA와의 의존성

- BaseEntity 클래스가 JPA에 의존하게 되어 특정 JPA 구현체에 종속되며 향후 JPA 교체 시 추가 작업이 필요할 수 있다.

추가 필드의 오버헤드

- BaseEntity의 감사 필드들(생성일자, 수정일자, 생성자, 수정자 등)로 인해 데이터베이스 레코드 크기가 증가하여 저장 공간이 더 필요할 수 있다.

검색 및 쿼리 어려움

- 감사 필드가 늘어나면 특정 시점의 사용자별 변경 기록 검색이 복잡해질 수 있으며 특히 대용량 데이터베이스에서는 성능 저하가 발생할 수 있다.

생성자 및 수정자 관리 어려움

- 자동 관리되는 감사 필드에서 특정 로직에 따라 생성자나 수정자를 수동으로 변경해야 하는 경우가 존재한다.

✨JPA Auditing을 알아보자

순수 JPA

```
@MappedSuperclass
@Getter
public class JpaBaseEntity {

  @Column(updatable = false)
  LocalDateTime createdDate;
  LocalDateTime lastModifiedDate;

    //영속성컨텍스트 일어나기 전에 시행
  @PrePersist
  public void PrePersist(){
```

```
    LocalDateTime now = LocalDateTime.now();
    createdDate = now;
    lastModifiedDate = now;
  }

// 업데이트 일어나기 전에 실행
 @PreUpdate
  public void PreUpdate(){
    lastModifiedDate = LocalDateTime.now();
 }
}
```

Auditing 직접 구현해보기

- 생성일시, 생성자, 수정일시, 수정자는 결국 엔티티의 영속성이 변경될때 저장한다.
- 엔티티의 영속성이 변경되는 `생성 > 수정 > 삭제` 이 흐름을 엔티티 라이프 사이클 이벤트라고 한다.
- Auditing 도 이러한 엔티티의 라이프 사이클 이벤트를 통해 구현하고있다.
- 우린 엔티티 라이프 사이클을 직접 관리하여 구현할 수 있다.

💡 객체가 생성되면 자동으로 실행하도록 메소드에 붙이는 **PostConstruct 의 원리와 같음 !**

- @PostConstruct는 의존성 주입이 완료된 후 실행되는 초기화 메서드를 지정하는 어노테이션

1. **객체 생성**
2. **의존성 주입**
3. **초기화 단계**
    - 의존성 주입이 완료된 후 @PostConstruct 실행, 이 시점에서는 모든 의존성 주입이 완료

entity 저장 이벤트

`@PrePersist` : EntityManager가 엔티티를 영속성상태로 만들기 직전에 메소드 수행

`@PostPersist` : EntityManager가 엔티티를 영속성상태로 만든 직후에 메소드 수행

entity 수정 이벤트

`@PreUpdate` : EntityManager가 엔티티를 갱신상태로 만들기 직전에 메소드 수행

`@PostUpdate` : EntityManager가 엔티티를 갱신상태로 만든 직후에 메소드 수행

entity 삭제 이벤트

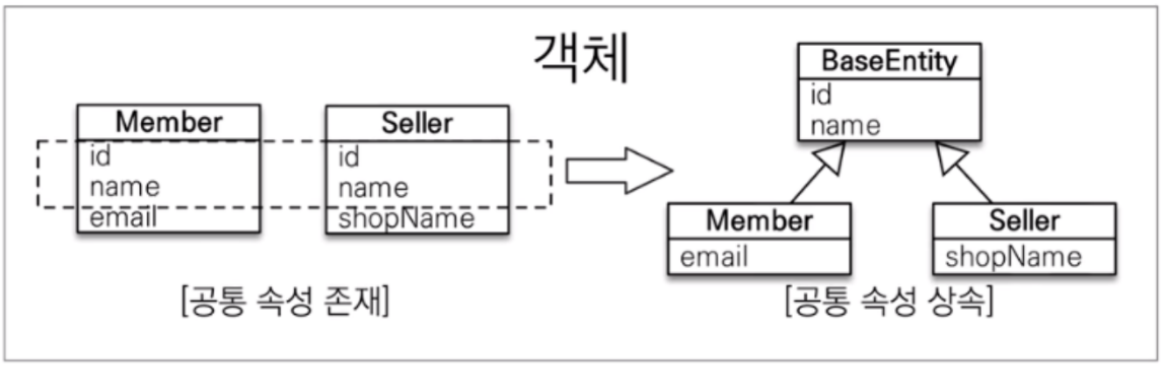
`@PerRemove` : EntityManager가 엔티티를 삭제상태로 만들기 직전에 메소드 수행

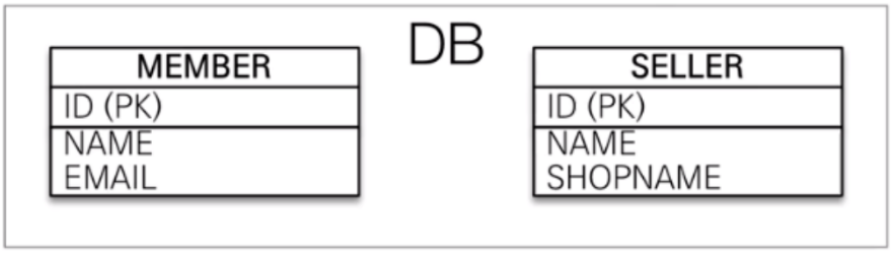
`@PostRemove` : : EntityManager가 엔티티를 삭제상태로 만든 직후에 메소드 수행

@MappedSuperclass

- 일반적으로, 상속 관계 매핑 전략에서 부모 클래스와 자식 클래스 모두 데이터베이스 테이블과 매핑
- 이와 다르게, 부모 클래스를 상속받는 자식 클래스에게 **매핑 정보 속성만 제공**하고 싶을 때 이 어노테이션을 사용

> 💡 엔티티는 엔티티만 상속받을 수 있음
>
> 엔티티가 아닌 클래스를 상속받기 위해서 **@MappedSuperclass 사용**





@AttributeOverride

- 만약 book 테이블의 생성일시만 createAt이 아닌 publishedAt으로 바꾸고 싶을 경우

```
@Entity
@AttributeOverride(
    name = "createdAt",     // 부모 엔티티의 필드명
    column = @Column(name = "publishedAt")  // 변경하고 싶은 컬럼
)
public class Book extends BaseEntity {
    // Book 엔티티의 다른 필드들...
}
```

여러 필드를 동시에 재정의 하고싶을 경우 **@AttributeOverrides 사용**

```
@Entity
@AttributeOverrides({
```

```
    @AttributeOverride(
        name = "createdAt",
        column = @Column(name = "publishedAt")
    ),
    @AttributeOverride(
        name = "modifiedAt",
        column = @Column(name = "lastUpdatedAt")
    )
})
public class Book extends BaseEntity {
    // Book 엔티티의 다른 필드들...
}
```

✨정리

- 상속관계 매핑이 아님
- @MappedSuperclass가 선언되어 있는 클래스는 엔티티가 아니다. 당연히 테이블과 매핑도 안된다.
- 단순히 부모 클래스를 상속 받는 자식 클래스에 매핑 정보만 제공
- 조회, 검색이 불가하다. 부모 타입으로 조회하는 것이 불가능하다는 이야기. (em.find(BaseEntity) 불가능)
- 직접 생성해서 사용할 일이 없으므로 추상 클래스로 만드는 것을 권장
- 테이블과 관계가 없고, 단순히 엔티티가 공통으로 사용하는 매핑 정보를 모으는 역할
- 주로 등록일, 수정일, 등록자, 수정자 같은 전체 엔티티에서 공통으로 적용하는 정보를 모을 때 사용
- JPA에서 @Entity 클래스는 @Entity나 @MappedSuperclass로 지정한 클래스만 상속할 수 있다.