07 Cascade

https://www.baeldung.com/jpa-cascade-types

소개

- **상태 전달**
  - 일대다 관계에서 "일" 쪽과 "다" 쪽을 모두 persist해야 하는가?
  - "일"쪽만 persist하면 "다"쪽도 자동 persist되면 좋겠다
  - 또한, "일" 쪽을 remove하면 "다" 쪽도 자동 remove되면 좋겠다
  - 이러한 동작은 Person과 Address처럼 person이 존재해야 address의 존재가 유의미한 관계에서 더욱 필요
  - Cascading → target 엔티티에 특정 action을 가하면 연관 엔티티에도 동일한 action을 가하라

[출처] https://www.baeldung.com/jpa-cascade-types

Cascade Type

- **JPA Cascade Type**
  - PERSIST
    - parent에서 child로 동작 전파
  - REMOVE
    - remove 동작 전파
  - MERGE
  - REFRESH
  - DETACH
  - ALL
    - 모든 동작을 전파

[출처] https://www.baeldung.com/jpa-cascade-types

Cascade Type

- **PERSIST**

```
@Entity
public class Person {
   @Id@GeneratedValue
   private int id;
   private String name;
   @OneToMany(mappedBy = "person", cascade = CascadeType.PERSIST)
   private List<Address> addresses;
}
```

```
@Entity
public class Address {
   @Id@GeneratedValue
   private int id;
   private String street;
   private String city;
   private int zipCode;
   @ManyToOne(fetch = FetchType.LAZY)
   private Person person;
}
```

[출처] https://www.baeldung.com/jpa-cascade-types

Cascade Type

- **PERSIST**

```
Address address1 = new Address();
address1.setCity("city1");
address1.setStreet("street1");
address1.setZipCode("zipCode1");

Address address2 = new Address();
address2.setCity("city1");
address2.setStreet("street1");
address2.setZipCode("zipCode1");

Person person = new Person();
person.setName("kim");

person.getAddresses().add(address1);
person.getAddresses().add(address2);

address1.setPerson(person);
address2.setPerson(person);

em.persist(person);
//em.persist(address1);
//em.persist(address2);
```

[출처] https://www.baeldung.com/jpa-cascade-types

Cascade Type

- **REMOVE**

```
@Entity
public class Person {
    @Id@GeneratedValue
    private int id;
    private String name;
    @OneToMany(mappedBy = "person", cascade = CascadeType.REMOVE)
    private List<Address> addresses;
}
```

테트스 코드

```
em.flush();
em.clear();

Person findPerson = em.find(Person.class, 1);
em.remove(findPerson);
```

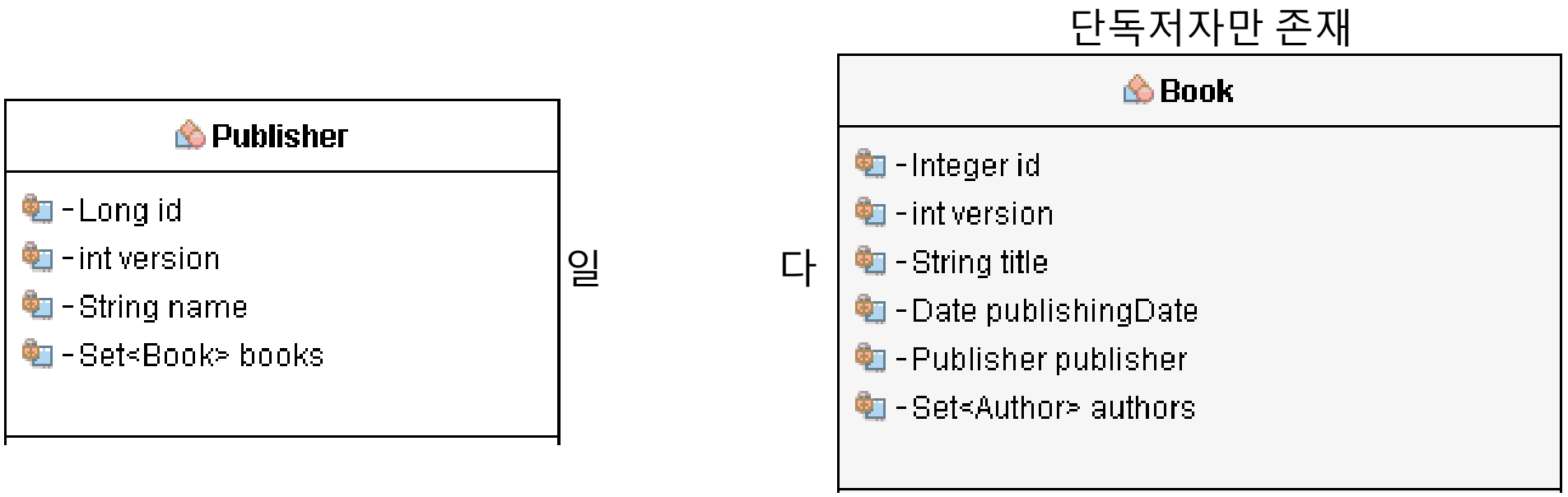
[출처] https://www.baeldung.com/jpa-cascade-types

Why you should avoid CascadeType.REMOVE for to-many associations and what to do instead

https://thorben-janssen.com/avoid-cascadetype-delete-many-assocations/

Cascade Type

- **To-Many연관관계에서 CascadeType.REMOVE의 문제**
  - 우선 의도하지 않고 자동으로 어떤 레코드가 삭제될 수 있다는 불안감이 있음
  - 더불어
    - 너무 많은 쿼리가 발생: 응용의 성능이 떨어지기는 하겠지만 데이터 유실이라는 큰 문제에 비해선 작은 문제
    - 의도한 것 이상으로 데이터가 삭제됨



출처: https://thorben-janssen.com/avoid-cascadetype-delete-many-assocations/

Cascade Type

- **One-To-Many연관관계에서 CascadeType.REMOVE의 문제**

```
@Entity
public class Publisher {

    @OneToMany(mappedBy = "publisher", cascade = CascadeType.REMOVE)
    private Set<Book> books = new HashSet<Book>();

    ...
}
```

```
Publisher p = em.find(Publisher.class, 1);
em.remove(p);
```

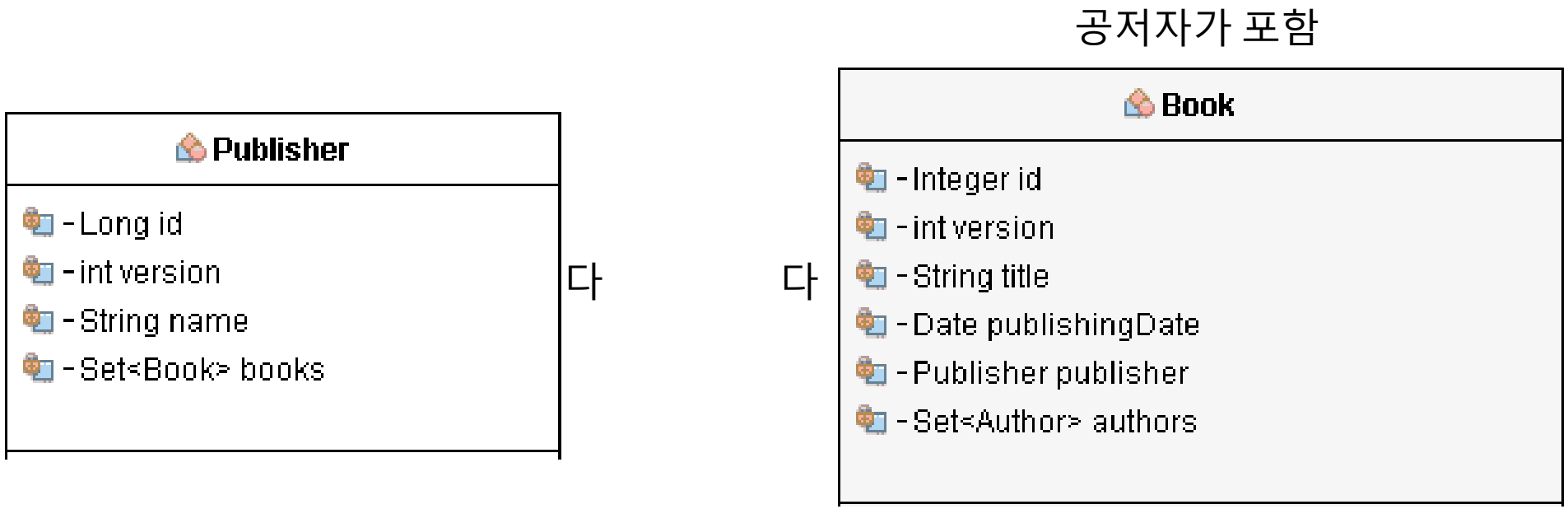
```
15:32:39,836 DEBUG [org.hibernate.SQL] - select publisher0_.id
15:32:39,885 DEBUG [org.hibernate.SQL] - select books0_.
15:32:39,933 DEBUG [org.hibernate.SQL] - delete
15:32:39,939 DEBUG [org.hibernate.SQL] - delete
15:32:39,940 DEBUG [org.hibernate.SQL] - delete
15:32:39,942 DEBUG [org.hibernate.SQL] - delete
```

remove 뿐만 아니라 select쿼리도 함께 나감
One by One으로 삭제. 응용이 커진다면?

출처: https://thorben-janssen.com/avoid-cascadetype-delete-many-assocations/

Cascade Type

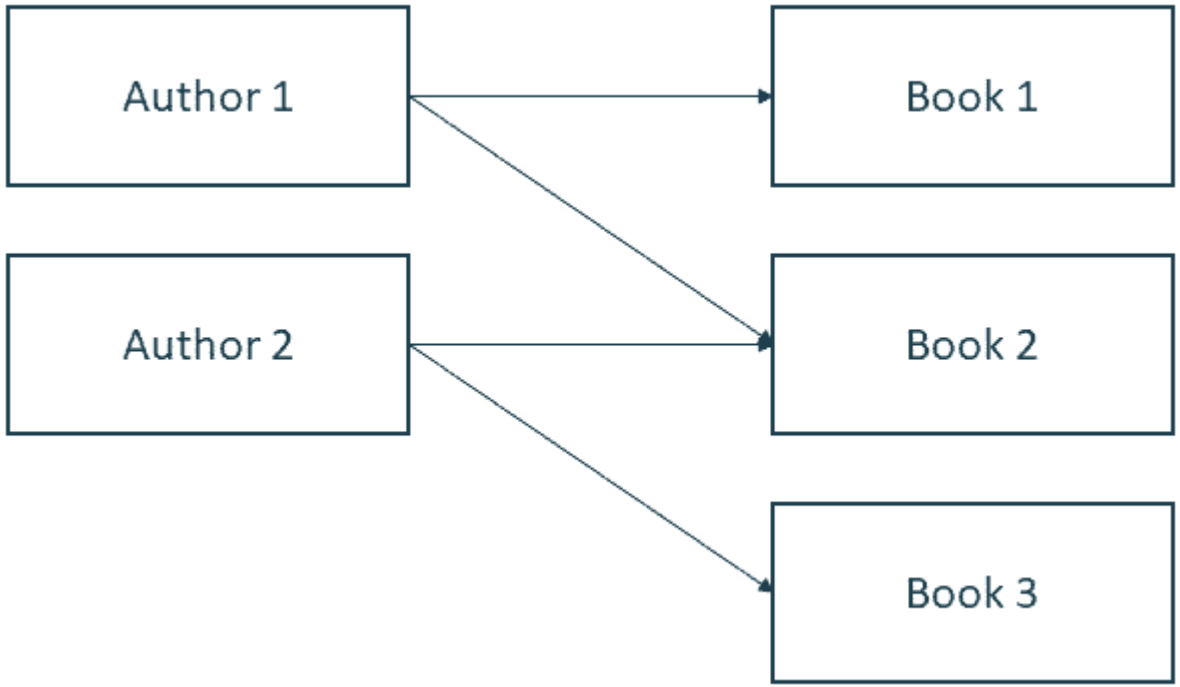
- **Remove More Than You Expected**
  - many-to-many 연관관계에서는 단순히 쿼리가 많이 발생하는 것보다 더 큰 문제가 야기될 수 있음



출처: https://thorben-janssen.com/avoid-cascadetype-delete-many-assocations/

Cascade Type

- **Remove More Than You Expected**
  - 아래와 같은 연관관계 설정



출처: https://thorben-janssen.com/avoid-cascadetype-delete-many-assocations/

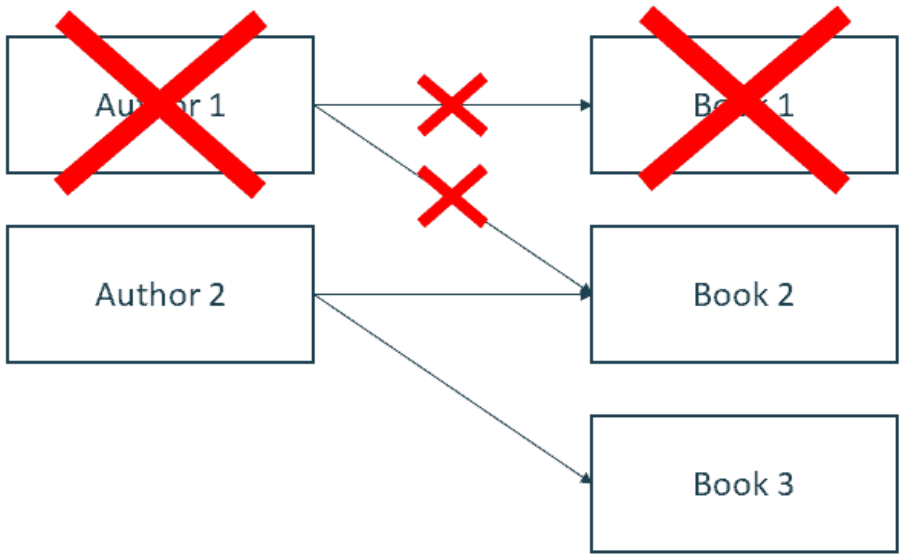
Cascade Type

- **Remove More Than You Expected**
  - ***Author 1을 삭제하면?***

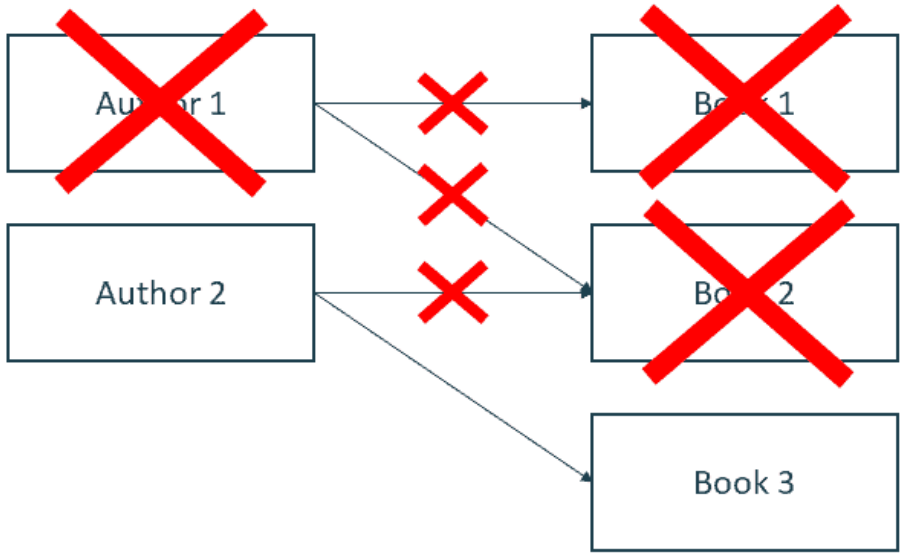
```
@Entity
public class Author {

    @ManyToMany(mappedBy = "authors", cascade = CascadeType.REMOVE)
    private Set<Book> books = new HashSet<Book>();

    ...
}
```



기대

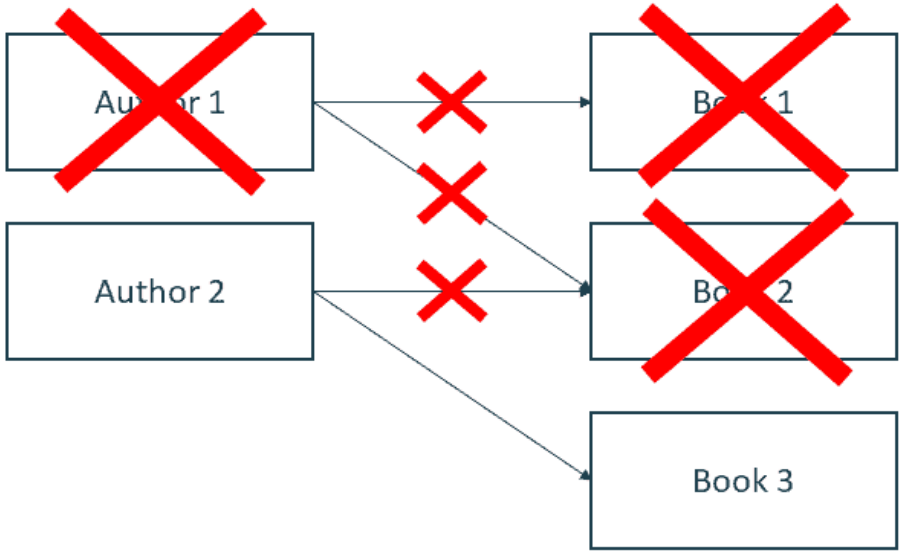


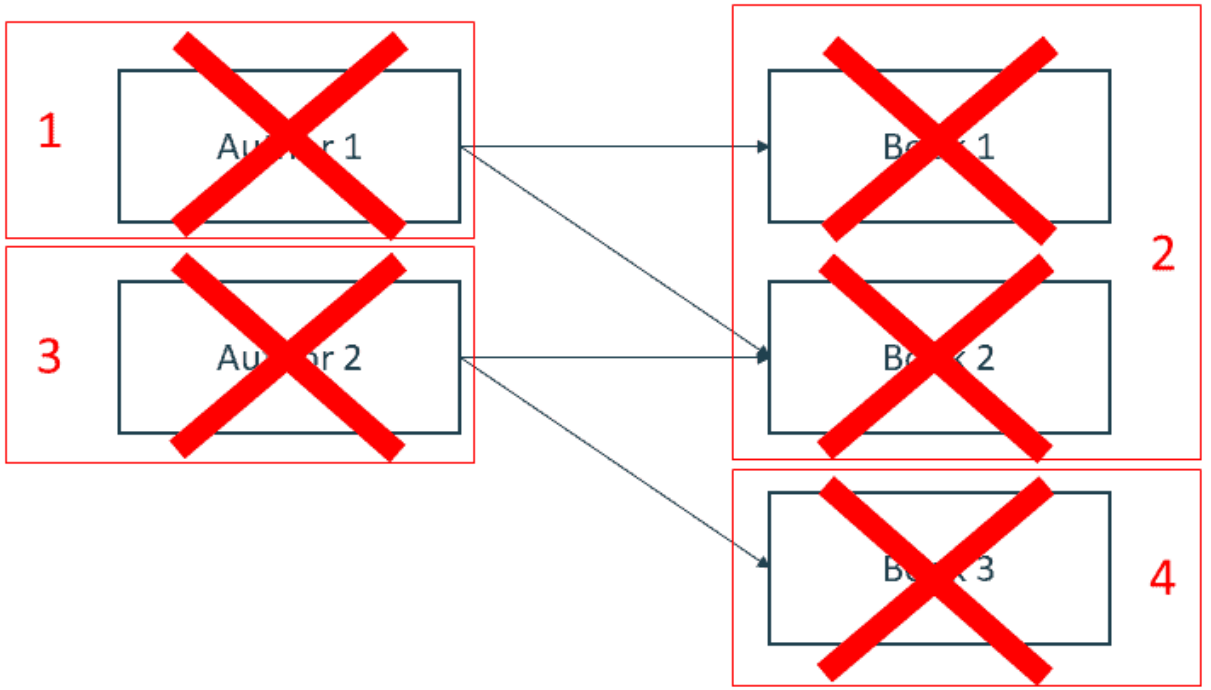
실제동작

출처: https://thorben-janssen.com/avoid-cascadetype-delete-many-assocations/

Cascade Type

- **Remove More Than You Expected**
  - 양방향으로 CascadeType.REMOVE를 적용했다면?





To-Many관계에서 persist에 대한 cascade는 괜찮지만
삭제와 관련된 remove(all)동작은 명시적으로 직접 제거

출처: https://thorben-janssen.com/avoid-cascadetype-delete-many-assocations/

Cascade Type

- **원인 분석과 해결방법**
  - 위와 같은 문제가 발생하는 근본 원인은 Author1과 Author2과 Book2를 공유하면 생기는 문제
  - Post와 Comment와 같이 하나의 Comment가 여러 Post에 공유되는 것이 아닌 하나의 Post에 종속적인 관계라면 문제가 발생하지 않음
  - 특정 자식이 하나의 부모에 속해 있을 경우에만 REMOVE 적용
  - 아니라면 직접 쿼리를 작성하여 명시적으로 연관관계 엔티티 제거

출처: https://thorben-janssen.com/avoid-cascadetype-delete-many-assocations/

Hibernate Tips: How to delete child entities from a many-to-one association

https://thorben-janssen.com/hibernate-tips-how-to-delete-child-entities/

orphanRemoval

- **Many-To-One관계에서 child는 부모가 존재하지 않을 경우 더 의상 의미가 없다고 할 때, 부모 엔티티를 삭제하면 자식 엔티티도 자동 삭제하려면**
  - orphanRemoval
- **예를 들어 Book과 Review의 관계**

```
@Entity
public class Book {

    @OneToMany(mappedBy = "book", orphanRemoval = true, cascade = CascadeType.PERSIST)
    private List<Review> reviews = new ArrayList<Review>();

    ...
}
```

- orphanRemoval = **true** 로 설정하면 persist, remove와 같이 엔티티 매니저의 메소드가 연관관계 엔티티에 적용되는 것이 아님. 위 예에서는 review를 null로 변경하면 변경감지에서 객체 관점에서 연관된 엔티티가 더 이상 없다고 생각하고 기존의 연관 review를 삭제
- 부모를 삭제는 경우: 부모가 삭제되어서 자식이 삭제되었다기보다는 부모가 삭제되면서 자식에 대한 참조를 잃기 때문에 자식도 삭제된다고 보는 것

출처: https://thorben-janssen.com/avoid-cascadetype-delete-many-assocations/