Home » Blog » Programming » SQL » JPA CascadeType.REMOVE vs Hibernate @OnDelete

JPA CascadeType.REMOVE vs Hibernate @OnDelete

By Roger Keays, 13 June 2014

Somehow database models and ORM always end up being more difficult than you expect. Here is a common source of confusion between JPA cascade operations and database cascade operations. Basically they do the opposite thing. For example:

```
public class House {

    @OneToOne
    Object door;
}
```

If you use**CascadeType.REMOVE** then deleting the house will also delete the door (using an extraSQL statement).

```
    @OneToOne(cascade=CascadeType.REMOVE)
    Object door;
```

If you use **@OnDelete** then deleting the door will also delete the house (using an ON DELETE CASCADE database foreign key).

```
    @OneToOne
    @OnDelete(action = OnDeleteAction.CASCADE)
    Object door;
```

JPA has not standardized the ON DELETE and ON UPDATE foreign keyactions possibly because they are SQL-specific and JPA is supposed to be storage-agnostic. I think this is unfortunate -what I'm looking for is ON DELETE SET NULL which would mean that when I delete the door, House.door gets set to null automatically. It's a fairly common requirement and is implemented in OpenJPA like this:

```
    @OneToOne
    @ForeignKey(deleteAction=ForeignKeyAction.NULL)
    Object door;
```

For the moment it looks like I'll have to stick to OpenJPA. Not sure why this isn't an option in Hibernate.

About Roger Keays

Roger Keays is an artist, an engineer, and a student of life. He has no fixed addressand has leftfootprints on 40-something different countries around the world. Roger is addicted to surfing. His other interests are music, psychology, languages, the proper use of semicolons, and finding good food.