How to implement a soft delete with Hibernate

To implement a soft delete with Hibernate, you have to:

1. tell Hibernate to perform a SQL UPDATE instead of a DELETE operation and
2. exclude all “deleted” records from your query results.

Update the record instead of deleting it

To implement a soft delete, you need to override Hibernate’s default remove operation. You can do that with an *@SQLDelete* annotation. This annotation allows you to define a custom, native SQL query that Hibernate will execute when you delete the entity. You can see an example of it in the following code snippet.

```
@Entity
@SQLDelete(
        sql = “UPDATE account SET state = ‘DELETED’
WHERE id = ?”,
        check = ResultCheckStyle.COUNT)
public class Account { … }
```

That is all you need to do to create a basic soft delete implementation. But there are 2 other things you need to handle:

1. When you delete an Account entity, Hibernate doesn’t update the value of its state attribute in the current session.
2. You need to adapt all queries to exclude the deleted entities.

Update state property in current session

Hibernate doesn't parse the native query you provide to the *@SQLDelete* annotation. It just sets the values of the bind parameters and executes it. It, therefore, doesn't know that you provided an SQL UPDATE statement instead of a DELETE statement to the *@SQLDelete* annotation. It also doesn't know that the value of the *state* property is outdated after it performed the delete operation.

If your code might use the entity object after it got deleted, you need to update the *state* property yourself. The easiest way to do that is to use a lifecycle callback, as I do in the following code snippet. The *@PreRemove* annotation on the *deleteUser* method tells Hibernate to call this method before it performs the remove operation. I use it to set the value of the *state* property to *DELETED*.

```
@Entity
@SQLDelete(
        sql = "UPDATE account SET state = 'DELETED'
WHERE id = ?",
        check = ResultCheckStyle.COUNT)
public class Account {


        …


        @PreRemove
        public void deleteUser() {
                this.state = AccountState.DELETED;
        }


}
```

Implement a soft delete with Hibernate

Exclude "deleted" entities in queries

Hibernate's *@Where* annotation allows you to define an SQL snippet which Hibernate adds to the WHERE clause of all queries. The following code snippet shows a *@Where* annotation that excludes a record if its *state* is *DELETED*.

```
@Entity
@SQLDelete(
        sql = "UPDATE account SET state = 'DELETED'
WHERE id = ?",
        check = ResultCheckStyle.COUNT)
@Where(clause = "state <> 'DELETED'")
public class Account { … }
```

As you can see in the following code snippets, Hibernate adds the defined WHERE clause when you perform a JPQL query or call the *EntityManager.find* method.

```
Account a = em.find(Account.class, a.getId());
```

```
16:07:59,511 DEBUG SQL:92 – select account0_.id as
id1_0_0_, account0_.name as name2_0_0_, account0_.state
as state3_0_0_ from Account account0_ where
account0_.id=? and ( account0_.state <> 'DELETED')
```