Hibernate JPA Cascade Types

We learned about **mapping associated entities** in hibernate already in previous tutorials such as **one-to-one mapping** and **one-to-many mappings**. There we wanted to save the mapped entity whenever relationship owner entity got saved. To enable this we had use "**CascadeType**" attribute. In this **JPA Cascade Types** tutorial, we will learn about various type of available options for cascading via **CascadeType**.

How JPA Cascade Types Work?

Before moving forward, let's look at how this cascade type attribute is defined in your code. Let's have an example for more clear understanding. Take a scenario where an Employee can have multiple Accounts; but one account must be associated with only one employee. Let's create entities with minimum information for sake of clarity.

EmployeeEntity.java

```
@Entity
@Table(name = "Employee")
public class EmployeeEntity implements Serializable
{
    private static final long serialVersionUID = -1798070786993154676L;
    @Id
    @Column(name = "ID", unique = true, nullable = false)
    private Integer           employeeId;
    @Column(name = "FIRST_NAME", unique = false, nullable = false, length = 100)
    private String            firstName;
    @Column(name = "LAST_NAME", unique = false, nullable = false, length = 100)
    private String            lastName;
    @OneToMany(cascade=CascadeType.ALL, fetch = FetchType.LAZY)
    @JoinColumn(name="EMPLOYEE_ID")
    private Set<AccountEntity> accounts;

    //Getters and Setters Ommited
}
```

AccountEntity.java

```
@Entity
@Table(name = "Account")
public class AccountEntity implements Serializable
{
    private static final long serialVersionUID = 1L;
    @Id
    @Column(name = "ID", unique = true, nullable = false)
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Integer           accountId;
    @Column(name = "ACC_NO", unique = false, nullable = false, length = 100)
    private String            accountNumber;

    @OneToOne (mappedBy="accounts",  fetch = FetchType.LAZY)
    private EmployeeEntity employee;

}
```

Look at the bold line in above source code for `EmployeeEntity.java`. It defines " `cascade=CascadeType.ALL` " and it essentially means that any change happened on `EmployeeEntity` must cascade to `AccountEntity` as well. If you save an employee, then all associated accounts will also be saved into database. If you delete an Employee then all accounts associated with that Employee also be deleted. Simple enough.

But what if we only want to cascade only save operations but not delete operation. Then we need to clearly specify it using below code.

```
@OneToMany(cascade=CascadeType.PERSIST, fetch = FetchType.LAZY)
@JoinColumn(name="EMPLOYEE_ID")
private Set<AccountEntity> accounts;
```

Now only when save() or persist() methods are called using employee instance then only accounts will be persisted. If any other method is called on session, it's effect will not affect/cascade to accounts.

JPA Cascade Types

The cascade types supported by the Java Persistence Architecture are as below:

1. **CascadeType.PERSIST** : cascade type `presist` means that save() or persist() operations cascade to related entities.
2. **CascadeType.MERGE** : cascade type `merge` means that related entities are merged when the owning entity is merged.
3. **CascadeType.REFRESH** : cascade type `refresh` does the same thing for the refresh() operation.
4. **CascadeType.REMOVE** : cascade type `remove` removes all related entities association with this setting when the owning entity is deleted.
5. **CascadeType.DETACH** : cascade type `detach` detaches all related entities if a “manual detach” occurs.
6. **CascadeType.ALL** : cascade type `all` is shorthand for all of the above cascade operations.

There is no **default cascade type in JPA**. By default no operations are cascaded.

The cascade configuration option accepts an array of CascadeTypes; thus, to include only refreshes and merges in the cascade operation for a One-to-Many relationship as in our example, you might see the following:

```
@OneToMany(cascade={CascadeType.REFRESH, CascadeType.MERGE}, fetch = FetchType.LAZY)
@JoinColumn(name="EMPLOYEE_ID")
private Set<AccountEntity> accounts;
```

Above cascading will cause accounts collection to be only merged and refreshed.

Hibernate Cascade Types

Now lets understand what is cascade in hibernate in which scenario we use it.

Apart from JPA provided cascade types, there is one more cascading operation in hibernate which is not part of the normal set above discussed, called "**orphan removal**". This removes an owned object from the database when it's removed from its owning relationship.

Let's understand with an example. In our Employee and Account entity example, I have updated them as below and have mentioned "**orphanRemoval = true**" on accounts. It essentially means that whenever I will remove an 'account from accounts set' (which means I am removing the relationship between that account and Employee); the account entity which is not associated with any other Employee on database (i.e. orphan) should also be deleted.

EmployeeEntity.java

```
@Entity
@Table(name = "Employee")
public class EmployeeEntity implements Serializable
```

```java
{
    private static final long serialVersionUID = -1798070786993154676L;
    @Id
    @Column(name = "ID", unique = true, nullable = false)
    private Integer           employeeId;
    @Column(name = "FIRST_NAME", unique = false, nullable = false, length = 100)
    private String            firstName;
    @Column(name = "LAST_NAME", unique = false, nullable = false, length = 100)
    private String            lastName;

    @OneToMany(orphanRemoval = true, mappedBy = "employee")
    private Set<AccountEntity> accounts;

}
```

AccountEntity.java

```java
@Entity (name = "Account")
@Table(name = "Account")
public class AccountEntity implements Serializable
{
    private static final long serialVersionUID = 1L;
    @Id
    @Column(name = "ID", unique = true, nullable = false)
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Integer           accountId;
    @Column(name = "ACC_NO", unique = false, nullable = false, length = 100)
    private String            accountNumber;

    @ManyToOne
    private EmployeeEntity employee;
}
```

TestOrphanRemovalCascade.java

```java
public class TestOrphanRemovalCascade
{
   public static void main(String[] args)
   {
      setupTestData();

      Session sessionOne = HibernateUtil.getSessionFactory().openSession();
      org.hibernate.Transaction tx = sessionOne.beginTransaction();

      //Load the employee in another session
      EmployeeEntity employee = (EmployeeEntity) sessionOne.load(EmployeeEntity.class, 1);
      //Verify there are 3 accounts
      System.out.println("Step 1 : " + employee.getAccounts().size());

      //Remove an account from first position of collection
      employee.getAccounts().remove(employee.getAccounts().iterator().next());

      //Verify there are 2 accounts in collection
      System.out.println("Step 2 : " + employee.getAccounts().size());

      tx.commit();
      sessionOne.close();

      //In another session check the actual data in database
      Session sessionTwo = HibernateUtil.getSessionFactory().openSession();
      sessionTwo.beginTransaction();

      EmployeeEntity employee1 = (EmployeeEntity) sessionTwo.load(EmployeeEntity.class, 1);
      //Verify there are 2 accounts now associated with Employee
      System.out.println("Step 3 : " + employee1.getAccounts().size());

      //Verify there are 2 accounts in Account table
```

```
        Query query = sessionTwo.createQuery("from Account a");
        @SuppressWarnings("unchecked")
        List<AccountEntity> accounts = query.list();
        System.out.println("Step 4 : " + accounts.size());

        sessionTwo.close();

        HibernateUtil.shutdown();
    }

    private static void setupTestData(){
        Session session = HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();

        //Create Employee
        EmployeeEntity emp = new EmployeeEntity();
        emp.setEmployeeId(1);
        emp.setFirstName("Lokesh");
        emp.setLastName("Gupta");
        session.save(emp);

        //Create Account 1
        AccountEntity acc1 = new AccountEntity();
        acc1.setAccountId(1);
        acc1.setAccountNumber("11111111");
        acc1.setEmployee(emp);
        session.save(acc1);

        //Create Account 2
        AccountEntity acc2 = new AccountEntity();
        acc2.setAccountId(2);
        acc2.setAccountNumber("2222222");
        acc2.setEmployee(emp);
        session.save(acc2);
        //Create Account 3
        AccountEntity acc3 = new AccountEntity();
        acc3.setAccountId(3);
        acc3.setAccountNumber("33333333");
        acc3.setEmployee(emp);
        session.save(acc3);

        session.getTransaction().commit();
        session.close();
    }
}

Output:

Hibernate: insert into Employee (FIRST_NAME, LAST_NAME, ID) values (?, ?, ?)
Hibernate: insert into Account (ACC_NO, employee_ID, ID) values (?, ?, ?)
Hibernate: insert into Account (ACC_NO, employee_ID, ID) values (?, ?, ?)
Hibernate: insert into Account (ACC_NO, employee_ID, ID) values (?, ?, ?)
Hibernate: select employeeen0_.ID as ID1_1_0_, employeeen0_.FIRST_NAME as FIRST_NA2_1_0_, employeeen
LAST_NAM3_1_0_ from Employee employeeen0_ where employeeen0_.ID=?
Hibernate: select accounts0_.employee_ID as employee3_1_0_, accounts0_.ID as ID1_0_0_, accounts0_.ID
accounts0_.ACC_NO as ACC_NO2_0_1_, accounts0_.employee_ID as employee3_0_1_ from Account accounts0_
Step 1 : 3
Step 2 : 2
Hibernate: delete from Account where ID=?
Hibernate: select employeeen0_.ID as ID1_1_0_, employeeen0_.FIRST_NAME as FIRST_NA2_1_0_, employeeen
LAST_NAM3_1_0_ from Employee employeeen0_ where employeeen0_.ID=?
Hibernate: select accounts0_.employee_ID as employee3_1_0_, accounts0_.ID as ID1_0_0_, accounts0_.ID
accounts0_.ACC_NO as ACC_NO2_0_1_, accounts0_.employee_ID as employee3_0_1_ from Account accounts0_
Step 3 : 2
Hibernate: select accountent0_.ID as ID1_0_, accountent0_.ACC_NO as ACC_NO2_0_, accountent0_.employe
from Account accountent0_
Step 4 : 2
```

It’s a very good way of removing the matching/mismatching items from a collection (i.e. many-to-one or one-to-many relationships). You just remove the item from collection and hibernate take care of rest of the things for you. It will check whether entity is referenced from any place or not; If it is not then it will delete the entity from database itself.

Let me know of your thoughts and questions on **hibernate 5 cascade types** or **JPA cascade types**, if any.

Happy Learning !!