**Hibernate Save() and saveOrUpdate() Methods**

By Lokesh Gupta | Filed Under: [Hibernate](https://howtodoinjava.com/hibernate/)

We have learned that [**hibernate**](https://howtodoinjava.com/hibernate-tutorials/) works only with[**persistent entities**](https://howtodoinjava.com/hibernate/hibernate-entity-persistence-lifecycle-states/) and persistent entities are classes which are attached to any hibernate session. Please not that creating an instance of a class, you mapped with a hibernate annotations, does not automatically persist the object to the database. It must be save explicitly after attaching it to a valid hibernate session.

**Using save() method**

In hibernate, we generally use one of below two versions of save() method:

public Serializable save(Object object) throws HibernateException

public Serializable save(String entityName,Object object) throws HibernateException

Both **save()** methods take a transient object reference (which must not be null) as an argument. Second method takes an extra parameter ‘***entityName***‘ which is useful in case you have mapped multiple entities to a Java class. Here you can specify which entity you are saving using **save()** method.

Let’s write down a simple example to see above theory in practical:

**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;

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof EmployeeEntity)) return false;

EmployeeEntity otherEmployee = (EmployeeEntity) o;

if (getEmployeeId() != null ?

!getEmployeeId().equals(otherEmployee.getEmployeeId()) : otherEmployee.getEmployeeId() != null)

return false;

if (getFirstName() != null ?

!getFirstName().equals(otherEmployee.getFirstName()) : otherEmployee.getFirstName() != null)

return false;

if (getLastName() != null ?

!getLastName().equals(otherEmployee.getLastName()) : otherEmployee.getLastName() != null)

return false;

return true;

}

@Override

public int hashCode() {

int result = getEmployeeId() != null ? getEmployeeId().hashCode() : 0;

result = 31 \* result + (getFirstName() != null ? getFirstName().hashCode() : 0);

result = 31 \* result + (getLastName() != null?getLastName().hashCode() : 0);

return result;

}

//Getters and Setters are hidden here

}

Now let’s save this entity.

**SimplestSaveEntityExample.java**

public class SimplestSaveEntityExample

{

public static void main(String[] args)

{

Session sessionOne = HibernateUtil.getSessionFactory().openSession();

sessionOne.beginTransaction();

//Create new Employee object

EmployeeEntity emp = new EmployeeEntity();

emp.setEmployeeId(1);

emp.setFirstName("Lokesh");

emp.setLastName("Gupta");

//Save employee

sessionOne.save(emp);

sessionOne.getTransaction().commit();

HibernateUtil.shutdown();

}

}

Output:

Hibernate: insert into Employee (FIRST\_NAME, LAST\_NAME, ID) values (?, ?, ?)

We got our Employee entity saved. So easy. But in reality, it is not so simple usecase. There you may need to update again employee entity and then save again in another session. Should you call save()method again? Let’s check out.

public class SaveEntityAgainInAnotherSession

{

public static void main(String[] args)

{

Session sessionOne = HibernateUtil.getSessionFactory().openSession();

sessionOne.beginTransaction();

//Create new Employee object

EmployeeEntity emp = new EmployeeEntity();

emp.setEmployeeId(1);

emp.setFirstName("Lokesh");

emp.setLastName("Gupta");

//Save employee

sessionOne.save(emp);

sessionOne.getTransaction().commit();

Session sessionTwo = HibernateUtil.getSessionFactory().openSession();

sessionTwo.beginTransaction();

emp.setLastName("temp");

//Save employee again second time

sessionTwo.save(emp);

sessionTwo.getTransaction().commit();

HibernateUtil.shutdown();

}

}

Hibernate: insert into Employee (FIRST\_NAME, LAST\_NAME, ID) values (?, ?, ?)

Hibernate: insert into Employee (FIRST\_NAME, LAST\_NAME, ID) values (?, ?, ?)

WARN SqlExceptionHelper:144 - SQL Error: -104, SQLState: 23000

ERROR SqlExceptionHelper:146 - Violation of unique constraint SYS\_PK\_49: duplicate value(s) for column(s) ID in statement [insert into Employee (FIRST\_NAME, LAST\_NAME, ID) values (?, ?, ?)]

INFO AbstractBatchImpl:208 - HHH000010: On release of batch it still contained JDBC statements

Here hibernate tried to insert the entity again. Though it was **failed due to primary key check**, but check may not be there for other entities and you may end up with duplicate rows.

***Note:*** While second save() method causes duplicate row in different sessions, BUT in same session they will work correct. Look at below example.

public class SaveEntityAgainInSameSession

{

public static void main(String[] args)

{

Session sessionOne = HibernateUtil.getSessionFactory().openSession();

sessionOne.beginTransaction();

//Create new Employee object

EmployeeEntity emp = new EmployeeEntity();

emp.setEmployeeId(1);

emp.setFirstName("Lokesh");

emp.setLastName("Gupta");

//Save employee

sessionOne.save(emp);

emp.setLastName("temp");

//Save employee again second time

sessionOne.save(emp);

sessionOne.getTransaction().commit();

HibernateUtil.shutdown();

}

}

Hibernate: insert into Employee (FIRST\_NAME, LAST\_NAME, ID) values (?, ?, ?)

Hibernate: update Employee set FIRST\_NAME=?, LAST\_NAME=? where ID=?

It’s confusing. Right? Let’s make it simple. And rule is below:

Remember that you should not call save() method on a persistent entity (entity associated with any hibernate session). Any changes done to persistent entity is automatically saved.

Let’s understand this concept in simple example:

public class NoSaveCallForPersistentEntity

{

public static void main(String[] args)

{

Session sessionOne = HibernateUtil.getSessionFactory().openSession();

sessionOne.beginTransaction();

//Create new Employee object

EmployeeEntity emp = new EmployeeEntity();

emp.setEmployeeId(1);

emp.setFirstName("Lokesh");

emp.setLastName("Gupta");

//Save employee

sessionOne.save(emp);

emp.setLastName("temp");

sessionOne.getTransaction().commit();

//Let's see what got updated in DB

Session sessionTwo = HibernateUtil.getSessionFactory().openSession();

sessionTwo.beginTransaction();

EmployeeEntity employee = ( EmployeeEntity ) sessionTwo.load(EmployeeEntity.class, 1);

System.out.println(employee.getLastName());

sessionTwo.getTransaction().commit();

HibernateUtil.shutdown();

}

}

Output:

Hibernate: insert into Employee (FIRST\_NAME, LAST\_NAME, ID) values (?, ?, ?)

Hibernate: update Employee set FIRST\_NAME=?, LAST\_NAME=? where ID=?

Hibernate: select employeeen0\_.ID as ID1\_1\_0\_, employeeen0\_.FIRST\_NAME as FIRST\_NA2\_1\_0\_,

employeeen0\_.LAST\_NAME as LAST\_NAM3\_1\_0\_ from Employee employeeen0\_ where employeeen0\_.ID=?

temp

In above example, we made the ‘emp‘ object persistent using first save() method. Afterward when we updated the last name to ‘***temp***‘, an update query was executed as expected. This we verified in returned data as well. This is the correct way to work with persistent entities.

**Using saveOrUpdate() method**

In discussion of save() method, we forgot about case where we had to save persistent entity in another session and that got resulted in duplicate key error. That is also a valid scenario.

To handle such cases, you must use saveOrUpdate() method. Strictly speaking, you should use saveOrUpdate() with even non-persistent entities. Personally, I do not see any harm in doing so. Though, It may make you a little bit careless. So be cautious.

Let’s see how it can be used along with entity persisted with save() method.

public class SaveOrUpdateMethodExample

{

public static void main(String[] args)

{

Session sessionOne = HibernateUtil.getSessionFactory().openSession();

sessionOne.beginTransaction();

//Create new Employee object

EmployeeEntity emp = new EmployeeEntity();

emp.setEmployeeId(1);

emp.setFirstName("Lokesh");

emp.setLastName("Gupta");

//Save employee

sessionOne.save(emp);

sessionOne.getTransaction().commit();

Session sessionTwo = HibernateUtil.getSessionFactory().openSession();

sessionTwo.beginTransaction();

emp.setLastName("temp");

//Save employee again second time

sessionTwo.saveOrUpdate(emp);

sessionTwo.getTransaction().commit();

HibernateUtil.shutdown();

}

}

Output:

Hibernate: insert into Employee (FIRST\_NAME, LAST\_NAME, ID) values (?, ?, ?)

Hibernate: select employeeen\_.ID, employeeen\_.FIRST\_NAME as FIRST\_NA2\_1\_, employeeen\_.LAST\_NAME as

LAST\_NAM3\_1\_ from Employee employeeen\_ where employeeen\_.ID=?

Hibernate: update Employee set FIRST\_NAME=?, LAST\_NAME=? where ID=?

Now we are able to save the entity as well as update the entity as well using saveOrUpdate() method.

Please remember that if you have used saveOrUpdate() method in place of save() method above, then also result would have been same. saveOrUpdate() can be used with persistent as well as non-persistent entities both. Persistent entities will get updated, and transient entities will be inserted into database.

**Suggestion For Production Code**

It wouldn’t be advisable to try to match this code construction in production code. Ideally, what you would do is pass VO object to DAO layer, load the entity from the session and update the entity with by copying VO data onto it. This means that the updates take place on a persistent object, and we don’t actually have to call Session.save() or Session.saveOrUpdate() at all.

Once an object is in a persistent state, Hibernate manages updates to the database itself as you change the fields and properties of the object. It’s big relief.

**Bullet Points**

1. Save() method stores an object into the database. It will Persist the given transient instance, first assigning a generated identifier. It returns the id of the entity created.
2. SaveOrUpdate() calls either save() or update() on the basis of identifier exists or not. e.g if identifier does not exist, save() will be called or else update() will be called.
3. Probably you will get very few chances to actually call save() or saveOrUpdate() methods, as hibernate manages all changes done in persistent objects.