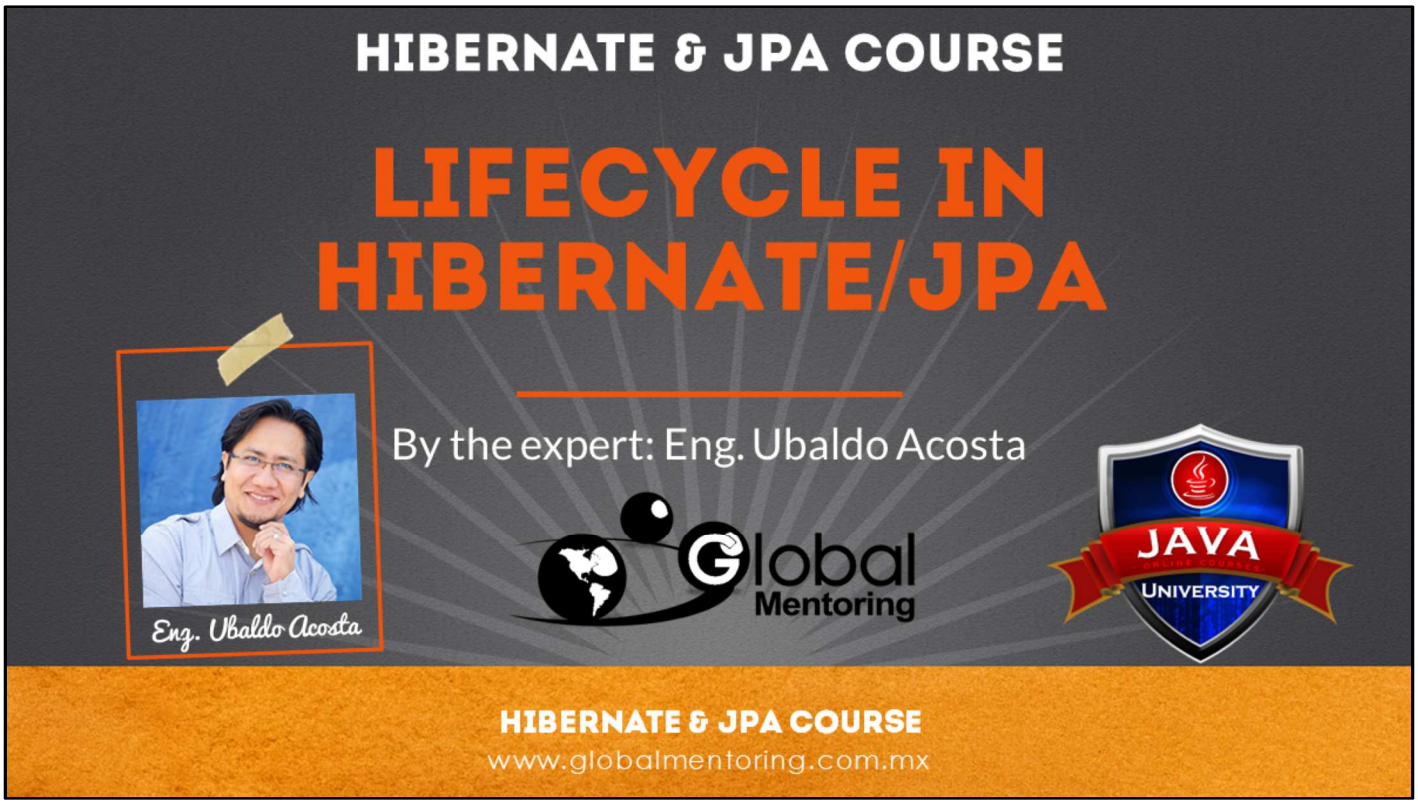
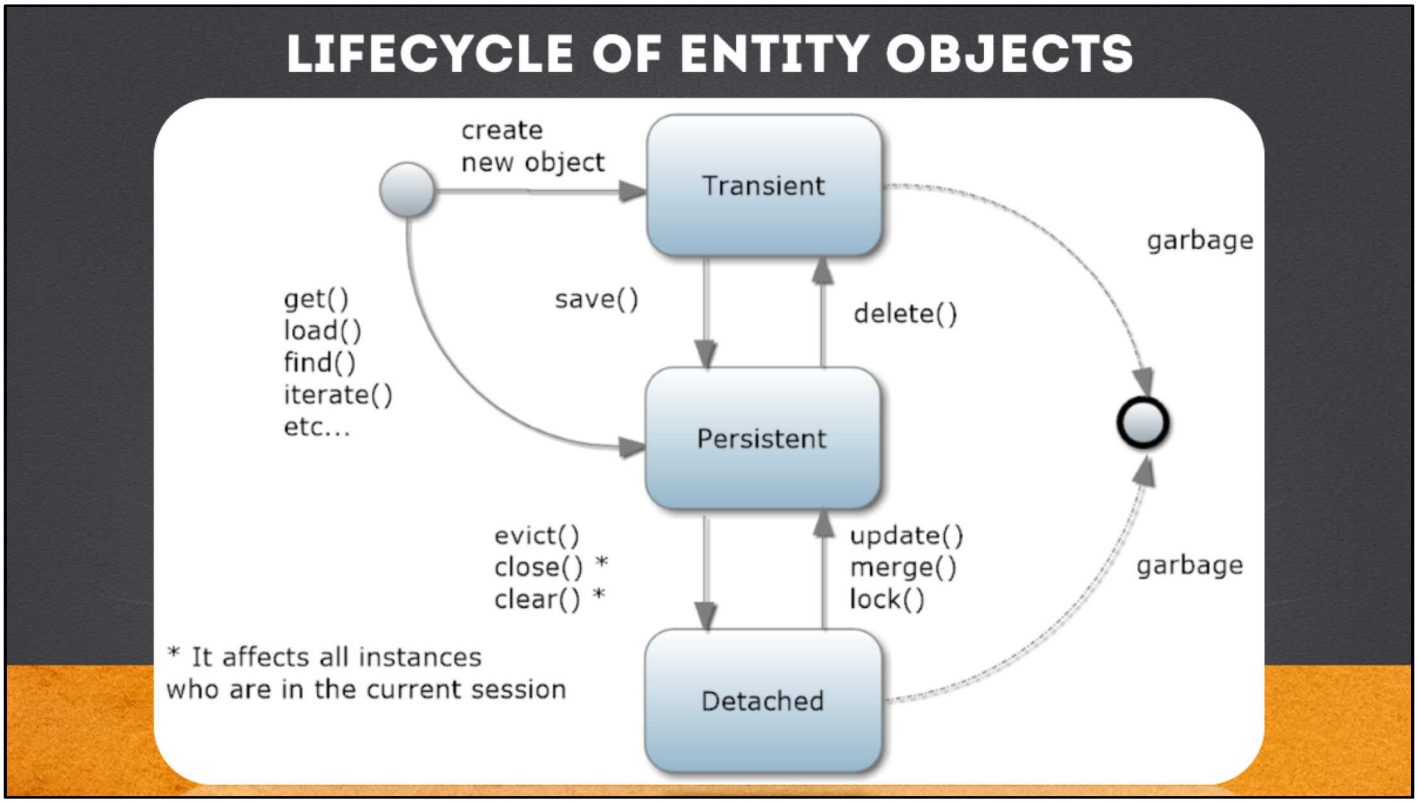


Hello, Ubaldo Acosta greets you again. I hope you're ready to start with this lesson ..

We are going to study the topic of Life Cycle in Hibernate and JPA.

Are you ready? Come on!



In this lesson we will review the life cycle topic in entity objects.

The entity objects, as we have commented, are Java classes which in the case of Hibernate or JPA are mapped with a database table. We will look at the states by which these entity objects can go through.

We observe first that we can create an object. When we create an entity object, for example an object of type Domicile, it is created by means of the operator new, but this does not mean that it is automatically persisted or saved in the database, but that it passes through different states.

The first state is known as a Transient state, since after we initialize a transaction we can execute the save () method or if this object already exists in the database.

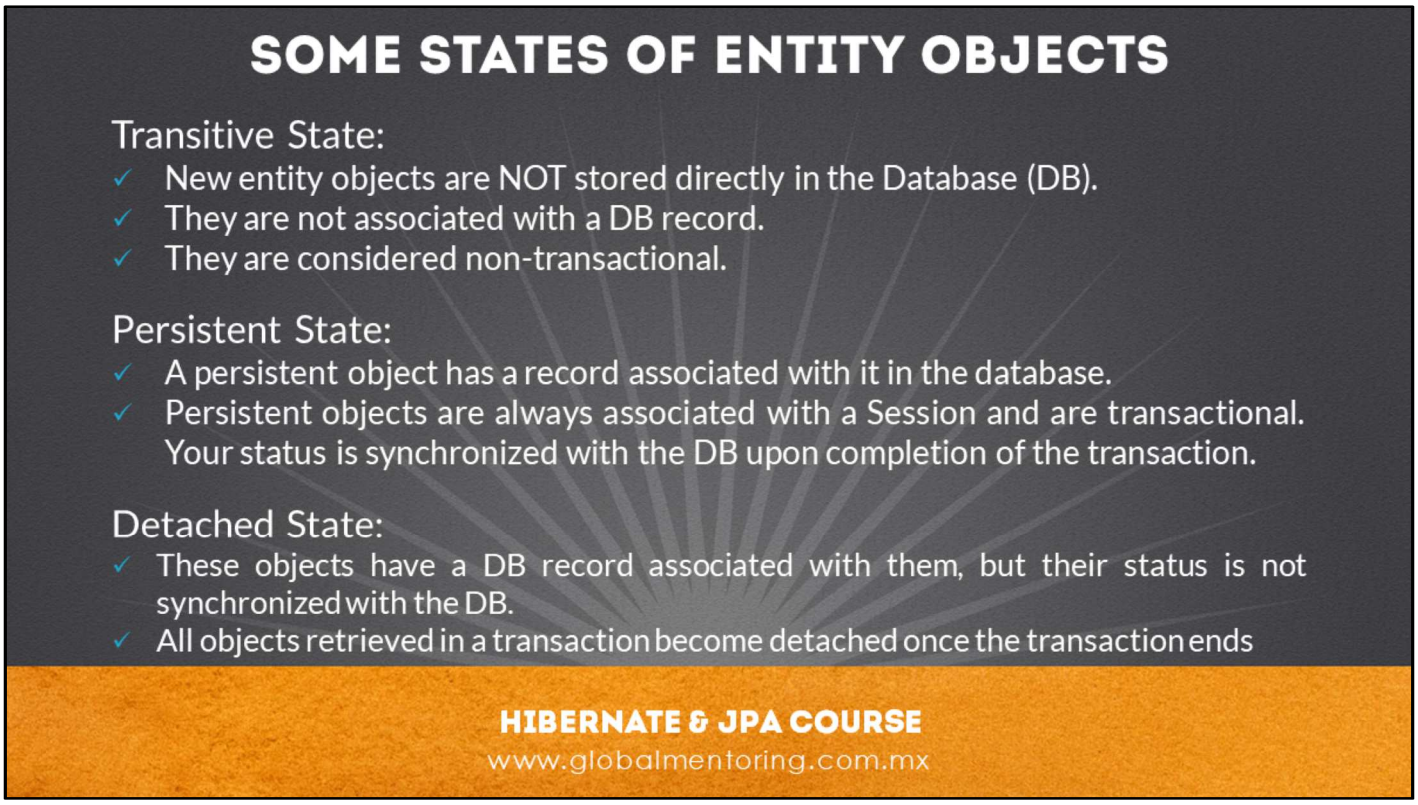
After we have already executed these methods, we move to a persistent state, in this state the transaction is still open, but once the transaction is closed when calling the close () method the entity object goes to a state known as Detached, that is, it is separated from the database and the synchronization with our database can no longer be guaranteed, therefore if we make any changes to our objects in this state we need to re-synchronize with the database. data either by one of the methods update, or merge (), the latter is particularly used to resynchronize an object in a detached state with the database.

In the diagram we observe the states through which Java objects pass, but we also observe what are the methods that allow us to change from one state to another. For example, if our object is not new, if we do not already have it stored in the database, we can recover it by means of the methods get, load, merge, among other methods and this will take us to the state of persistent, and a Once we close the session we go back to the detached or separate state.

We can see in the image that we have an asterisk in certain methods because these methods close () and clear () affect all the instances that are in the current session, therefore if the session is closed the objects go to the state of detached.

In case an object in a transitive state is not saved in the database, if it is not left in memory only, it will go to a garbage collection state, because it is never stored in the database and therefore this information will be lost and the garbage collector can delete the transitive object.

In the same way, a modified detached object, which was not synchronized with the database again, those changes will not be reflected in the base and the object will go directly to the garbage collector, which will be followed maintaining is the information that was synchronized at the time, but the changes that were made in the detached state is to say outside of the transaction, if they are not resynchronized, they will be lost because they were never synchronized with the base of data. These are the states that we have in Hibernate and they are very similar to JPA, and we are going to review more detail about these states below.

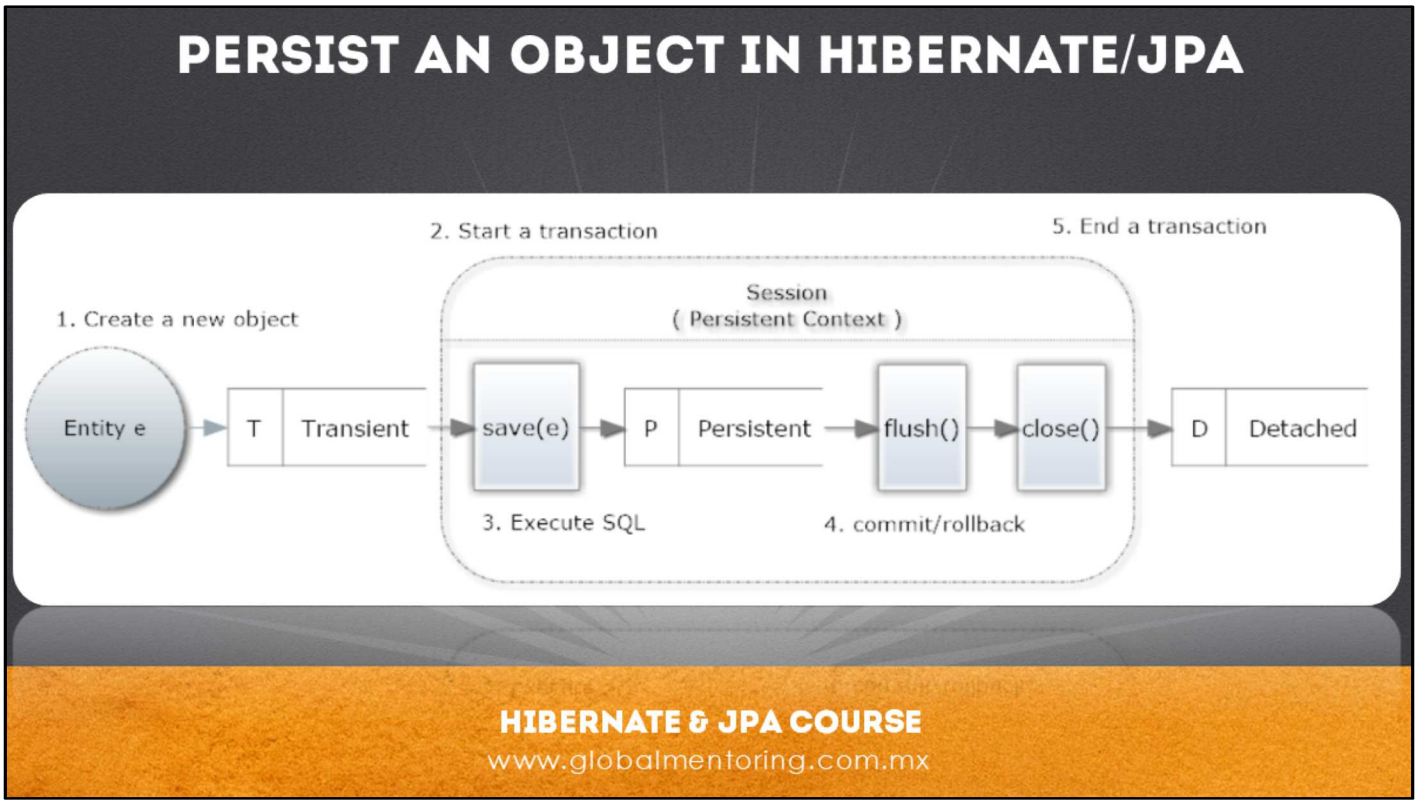


In the Transitive state, entity objects are stored directly in the database. These objects do not yet represent a record of the database because they are not yet synchronized with it, nor are they considered transactional because in this state there is still no transaction.

In our next state, a persistent object has a record associated with the database, so if we have an object which is already represented in a database table, we must already have the primary key associated with our object. Java. Persistent objects are also always associated with a session or transaction and their status is synchronized with the database upon completion of the transaction, so objects in the Persistent state will always be synchronized with the database and any modifications that are made to the database. Let's make the object will be reflected in the database at the end of the transaction.

When a transaction has already been completed, the objects move to a status of Detached, these objects have a record associated with the database, but their status is not synchronized with it and this implies that any modification to an object in the state of Detached is necessary to synchronize it again with the database and therefore reintegrate it into a transaction, if we do not integrate this object again into a transaction the values that we are modifying in this Java object will be lost once this process is finished.

The difference between the transitive state and the detached state is that the transitive state does not yet have an associated record in the database and the detached state already has a record associated with it, but any changes in the Java object are not synchronized so automatically and finally as a summary all the objects that are in a transaction are going to persist and synchronize automatically when the transaction ends.

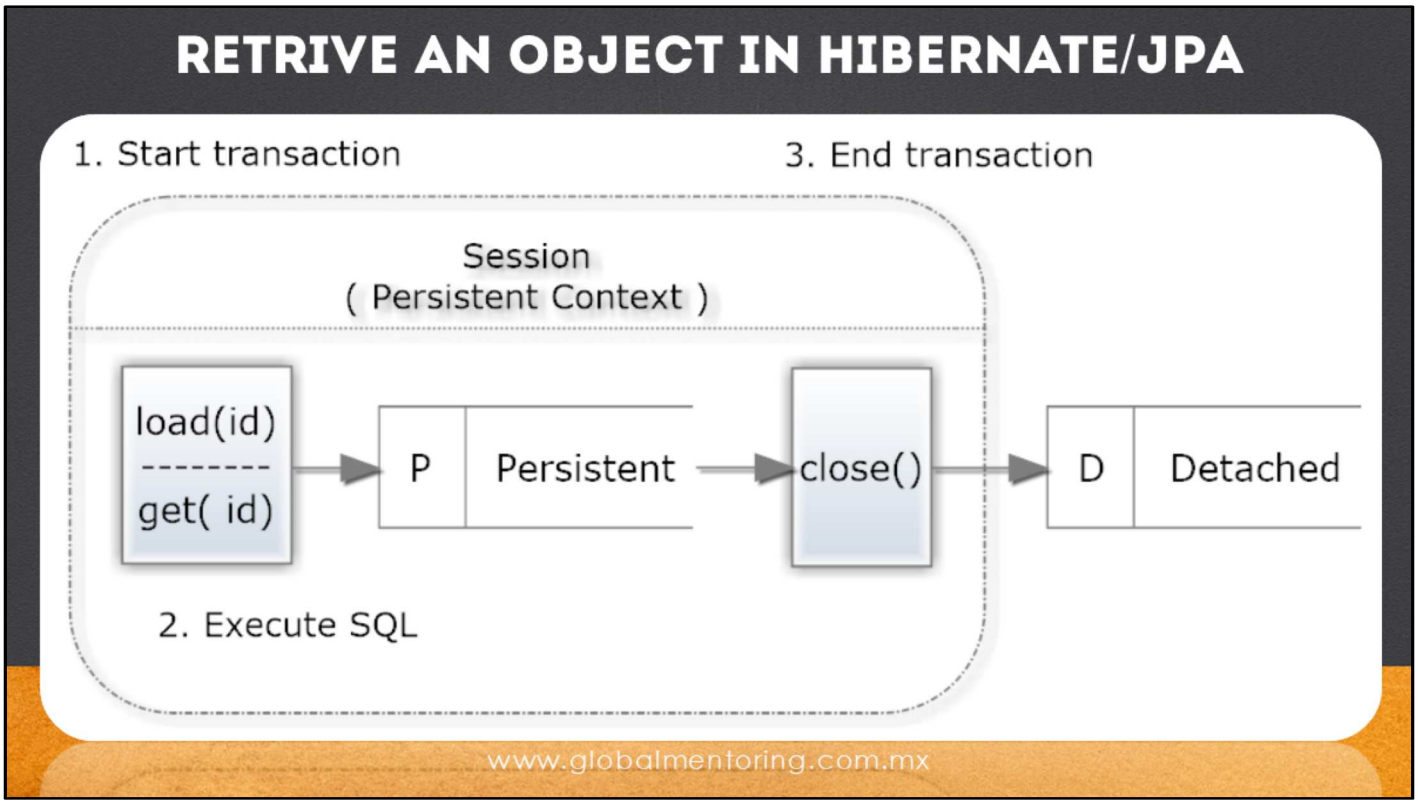


We will explain in the following images, the different states by which an object can pass in Hibernate or JPA. Here we can see how to persist an object in Hibernate or JPA. The first thing we have to do is create a new Entity object, for example, a new Domicile object, it is considered that this object because it is not yet saved in a database is in a transitive state.

Then we start a transaction, and everything that happens inside this block will be synchronized with the database. To create a new object and be able to store it in our database as we have, we have to send one of these methods to change the state of our object and go from transitive to a persistent state, so we send it to call the Save method () and this method what it is going to do is execute the insert statement so that this object is persisted in the database, later we save the changes by committing the transaction or calling the flush () method.

The commit method indirectly calls the flush method. The flush method means that any pending SQL in the current transaction will be finalized and the changes will be saved in the database and finally what we do is finish our transaction by means of the close () method, with this The transaction is completed and our object goes into a detached state.

This means that this object that we have created already has a representation in the database, but the modifications that we make in the future will not be synchronized more automatically with the database. That is the general process to persist an object in Hibernate or JPA.

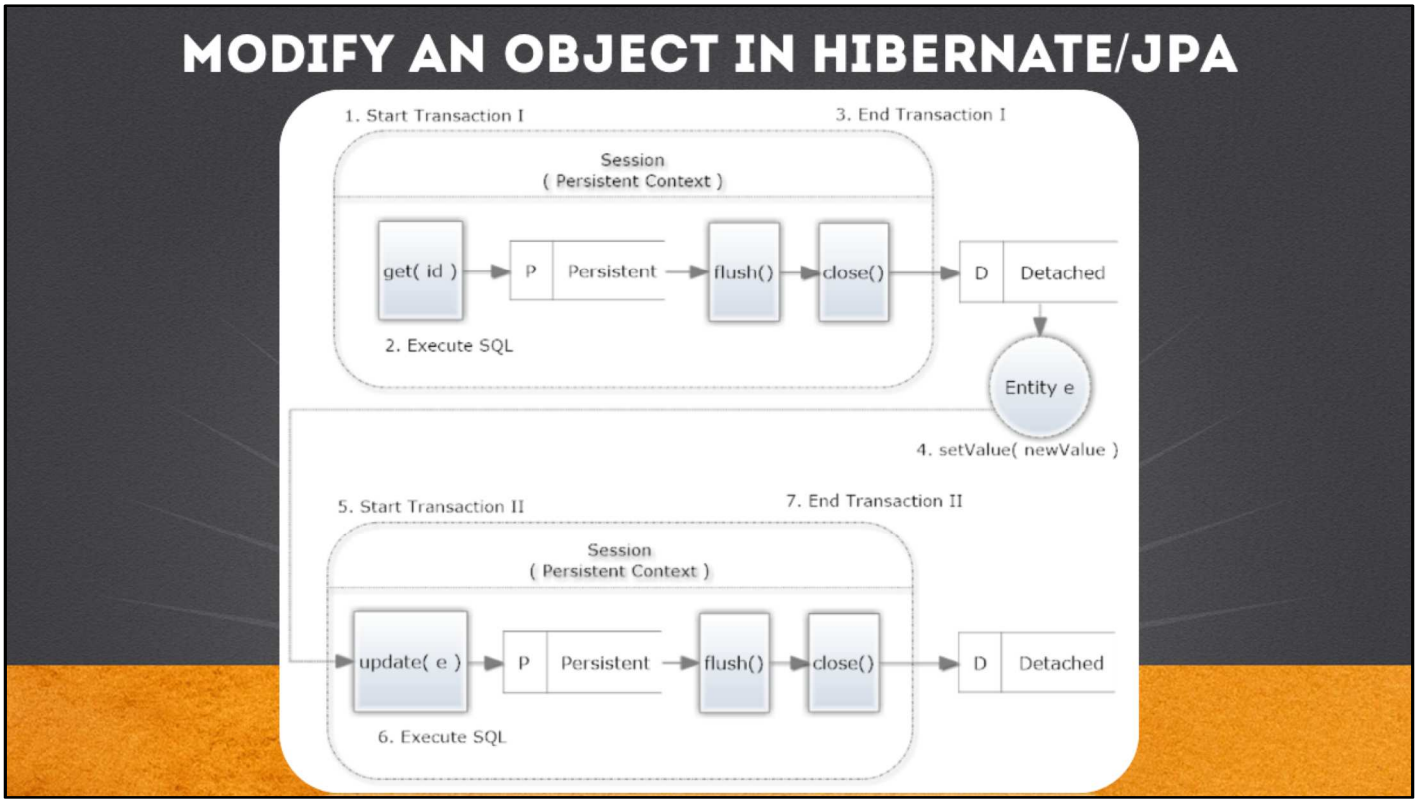


Let's review the flow below to recover an object already persisted in Hibernate or JPA.

The first thing we have to do is start a transaction and in that transaction we have several methods to recover an object that is already in the database. One method is called load, another method is called get, and a third is called merge. We must provide either the id or the object with the id or primary key to recover. The difference between these methods is that the load method if it does not find the primary key that we are looking for is going to throw an exception and the get method if it does not find the primary key that we are looking for simply leaves the object as null.

If we find the object in the database and because we are still in a transaction, the object that we have recovered goes to the persistent state and once we close our transaction the object goes into a detached state. In this case the detached objects can be sent, for example, to a form in the presentation layer and once we have finished modifying the object in the detached state, we have to integrate it back into a transaction in order to save the changes made by the user. Later we will see how to do this, but this is the general idea to recover an object that has a representation in our database.

There is also the merge method, which is used when we already have an open transaction, and then it is possible to pass an object to the state of transitive or detached to the persistent state using this merge method within an existing transaction. Later we will see in more detail how to use these methods.



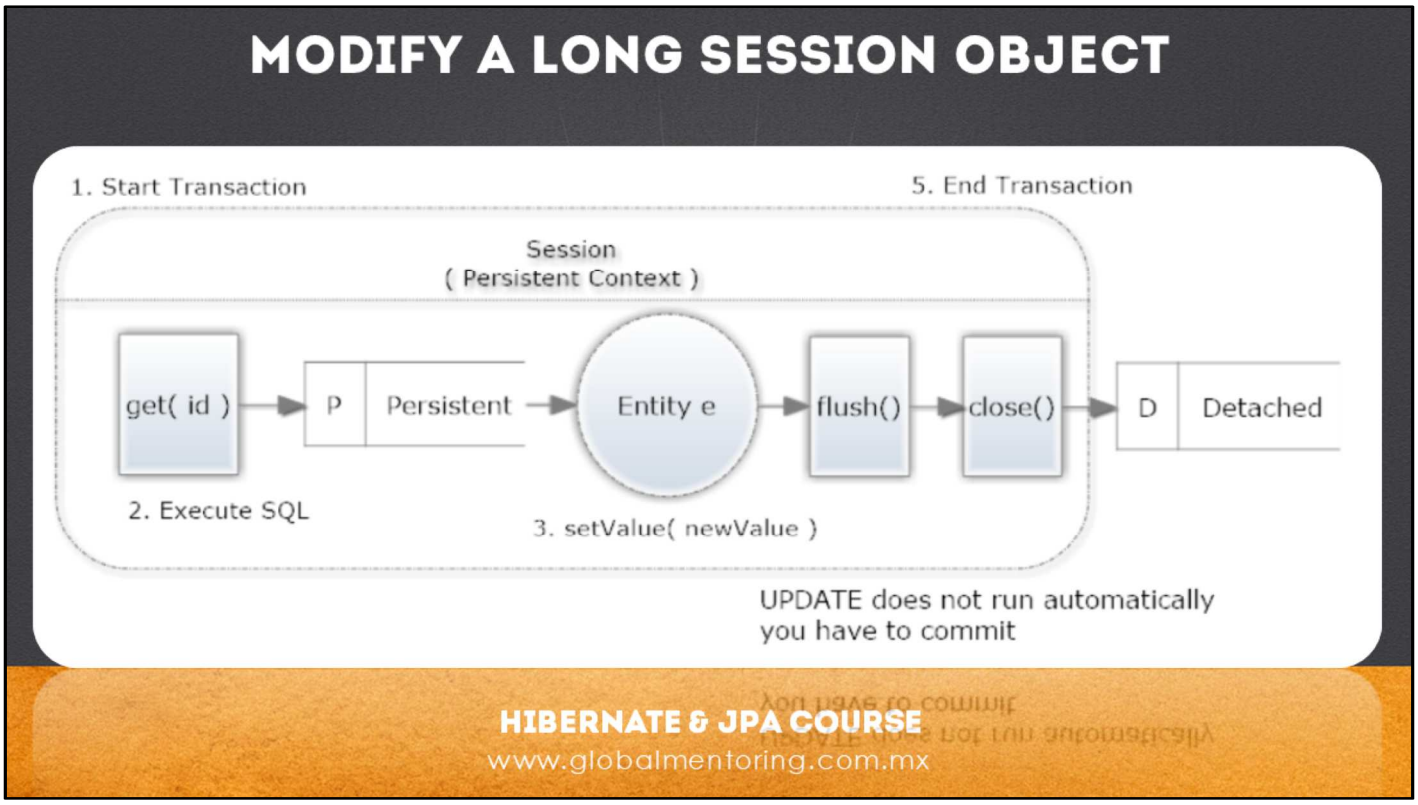
Let's now review how to modify a persistent object. What we have to do is start a transaction and, based on the previous flow, we must recover the object to be modified. We are going to provide a primary key to the object we are looking for and once we have finished recovering the object, we commit the transaction.

Once we have finished the transaction we move to the detached state, that is, we already have our Entity object, but in this case it is not synchronized automatically, so the values that we modify of our object by means of calls to the setValue methods will not automatically synchronize with the base because we have already completed the transaction.

Therefore, what we have to do is open a new transaction. The object that we have already obtained previously and that we have modified can use the update or merge method so that we change the state of the object to persistent. With this we have already made the respective modifications to our Entity object using a second transaction.

We commit or rollback the transaction. If we commit it is indirectly a flush, this means that the values that we have modified with the database have been synchronized again. Finally we send to call the close method and our object goes back to the detached state but the values that we modified previously we have already saved them in the database again.

This is a common example when we retrieve an object from the database, we modify it from the presentation layer and then we send it back to the data layer and in the data layer again we take care of saving the information synchronizing our object with the database in a new transaction.

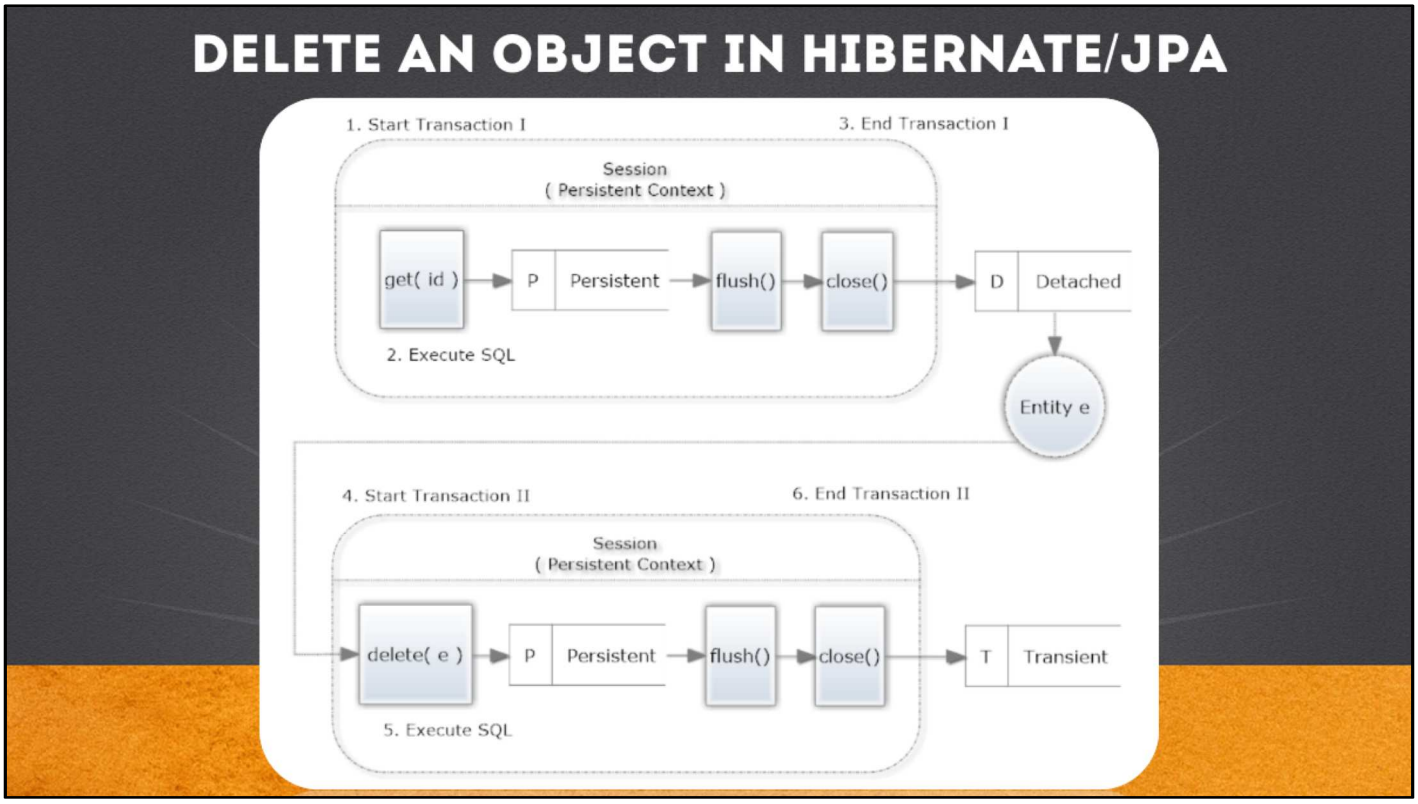


There is another way to modify our object already persisted in Hibernate or JPA.

If we have a long-term transaction, what we can do is start the transaction and keep it open as much as possible so that when we have recovered our object, through the get method it goes to a persisted state and within that same transaction we will modify the values of our object.

If we have not yet finished the transaction by committing indirectly, we will call the flush method, so these changes will be synchronized with the database record. Finally we close the transaction and our object goes to a detached state, so we can observe in this case that the values that we have modified to this entity through the respective setter methods because we are still inside the transaction are going to synchronize with the database.

A transaction is also known as the persistent context, therefore all SQL statements that we execute within our persistent context if we commit and rollback before finishing our transaction are going to synchronize with our database, therefore all this block is within a persistent context.



Finally we will review the status to remove a persistent object.

In this case we will do something very similar when we made the modification of our object, the first block is responsible for recovering the object we are looking for and therefore using the get method providing an identifier. We do the respective steps going through the persistent state, we make the flush and close our transaction.

This will return an object to us in Java and it will leave us in a detached state. To this object we no longer want to modify it, but to eliminate it from the database, therefore we initiate a new transaction and send the delete method and we pass it as parameter the object that we want to eliminate.

This changes the state of the object to persistent and when doing the flush or commit, the object is going to delete the record from the database. This does not imply that the Java object has been deleted, and therefore, since this object no longer has a representation in the database, it passes from the new account to the transitive state.

If we remember the state of transitive means that it is an object in memory, but that it has no representation in the database, and because we have removed it from the database, we have an object again in the transitive state. Here the difference of a state of transitive when it is new or when it is eliminated, is that when it is new the intention is to save it in a database, but when an object ends in a state of transitive because it was removed from the database it is no longer possible to save it in the database and this is one of the differences when we have objects in a transitive state but that come from a new creation of an entity object or when a transitive object was removed from the database. data.

Next we will create several exercises to put into practice the respective code of each of the states of our entity objects in Hibernate or JPA.