Caching in Hibernate

Hibernate caching improves the performance of the application by pooling the object in the cache. It is useful when we have to fetch the same data/objects multiple times.

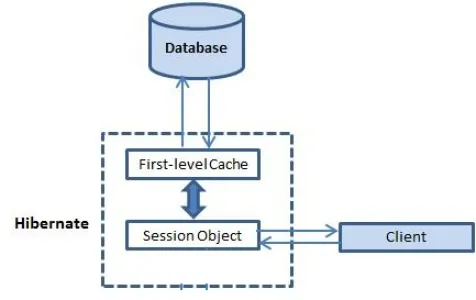
There are mainly two types of caching:

* First Level Cache, and
* Second Level Cache

**Fist level cache**

 It is enabled by default and you do not need to do anything to get this functionality working. In fact, you can not disable it even forcefully.

Its easy to understand the first level cache if we understand the fact that**it is associated with Session object**. As we know session object is created on demand from session factory and**it is lost, once the session is closed**. Similarly, first level cache associated with session object is available only till session object is live. It is available to session object only and is **not accessible to any other session object** in any other part of application.



## ****Important facts****

1. First level cache is associated with “session” object and other session objects in application can not see it.
2. The scope of cache objects is of session. Once session is closed, cached objects are gone forever.
3. First level cache is enabled by default and you can not disable it.
4. When we query an entity first time, it is retrieved from database and stored in first level cache associated with hibernate session.
5. If we query same object again with same session object, it will be loaded from cache and no sql query will be executed.
6. The loaded entity can be removed from session using evict() method. The next loading of this entity will again make a database call if it has been removed using evict() method.
7. The whole session cache can be removed using clear() method. It will remove all the entities stored in cache.

**Second level cache**:

This is apart from first level cache which is available to be used globally in session factory scope.

Above statement means, **second level cache is created in session factory scope** and is **available to be used in all sessions** which are created using that particular session factory.  
It also means that **once session factory is closed, all cache associated with it die** and cache manager also closed down.

## ****How second level cache works****

1. Whenever hibernate session try to load/get an entity, the very first place it look for cached copy of entity in first level cache (associated with particular hibernate session).
2. If cached copy of entity is present in first level cache, it is returned as result of load/get method.
3. If there is no cached entity in first level cache, then second level cache is looked up for cached entity.
4. If second level cache has cached entity, it is returned as result of load/get method. But, before returning the entity, it is stored in first level cache also so that next invocation to load method for entity will return the entity from first level cache itself, and there will not be need to go to second level cache again.
5. If entity is not found in first level cache and second level cache also, then database query is executed and entity is stored in both cache levels, before returning as response of load() method.
6. Second level cache validate itself for modified entities, if modification has been done through hibernate session APIs.
7. If some user or process make changes directly in database, the there is no way that second level cache update itself until “timeToLiveSeconds” duration has passed for that cache region. In this case, it is good idea to invalidate whole cache and let hibernate build its cache once again. You can use below code snippet to invalidate whole hibernate second level cache.

**Maven Dependency for Second level cache**

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-ehcache</artifactId>

<version>5.4.14.Final</version>

</dependency>

**Configuring Second level cache**

<property name="cache.use\_second\_level\_cache">false</property>

<property name=*"cache.region.factory\_class"*>

org.hibernate.cache.ehcache.EhCacheRegionFactory

</property>