|  |
| --- |
| <https://howtodoinjava.com/hibernate/hibernate-one-to-many-mapping-using-annotations/>                **How second level cache works**  Lets write all the facts point by point:   1. Whenever hibernate session try to load 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 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 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.   For both options, **caching strategy** can be of following types:   * **none** : No caching will happen. * **read-only** : If your application needs to read, but not modify, instances of a persistent class, a read-only cache can be used. * **read-write** : If the application needs to update data, a read-write cache might be appropriate. * **nonstrict-read-write** : If the application only occasionally needs to update data (i.e. if it is extremely unlikely that two transactions would try to update the same item simultaneously), and strict transaction isolation is not required, a nonstrict-read-write cache might be appropriate. * **transactional** : The transactional cache strategy provides support for fully transactional cache providers such as JBoss TreeCache. Such a cache can only be used in a JTA environment and you must specify hibernate.transaction.manager\_lookup\_class.  4. HQL Named Parameters  10. HQL Named Queries  11. HQL – Native SQL  Persistence LifeCycle States    Removed Object |