# **Hibernate Interview Questions**

# **How to initialize lazy relationsships *?***

You will have to make an explicit call on the lazy collection in order to initialize it (common practice is to call .size() for this purpose). In Hibernate there is a dedicated method for this (Hibernate.initialize()), but JPA has no equivalent of that. Of course you will have to make sure that the invocation is done, when the session is still available, so annotate your controller method with @Transactional. An alternative is to create an intermediate Service layer between the Controller and the Repository that could expose methods which initialize lazy collections.

The static methods Hibernate.initialize() and Hibernate.isInitialized(), provide the application with a convenient way of working with lazily initialized collections or proxies. Hibernate.initialize(entity.getXXX()) will force the initialization of a proxy, entity.getXXX(), as long as its Session is still open. Hibernate.initialize( ) has a similar effect for the collection of entities as well.

e.g. If we have two entity types Student and Department, then we can easily initialize lazy load enabled collection of students like below:

|  |
| --- |
| Department dept = (Department) session.get(Department.class, deptId);    //\*\*\*\*\*Initialize student collection\*\*\*\*\*\*\*  **Hibernate.initialize**(dept.getStudents()); |

Above code will immediately, load the collection of students. Now you are safe to close the session and you will be able to get data as and when required. It’s much cleaner way of pre-initializing lazy load enabled proxy or collection objects.

[***http://www.thoughts-on-java.org/5-ways-to-initialize-lazy-relations-and-when-to-use-them/***](http://www.thoughts-on-java.org/5-ways-to-initialize-lazy-relations-and-when-to-use-them/)

[***http://howtodoinjava.com/hibernate/lazy-loading-in-hibernate/***](http://howtodoinjava.com/hibernate/lazy-loading-in-hibernate/)

[***https://www.laliluna.de/jpa-hibernate-guide/ch02s02.html***](https://www.laliluna.de/jpa-hibernate-guide/ch02s02.html)

[***http://howtodoinjava.com/hibernate/use-hibernate-initialize-to-initialize-proxycollection/***](http://howtodoinjava.com/hibernate/use-hibernate-initialize-to-initialize-proxycollection/)

***What is N+1 query problem in Hibernate? How to solve the problem?***

 results in two distinct queries to the database (one for the user, another one for its roles). If you want to achieve better performace add the following method to your Spring Data JPA repository interface:

public interface PersonRepository extends JpaRepository<Person, Long> {

@Query("SELECT p FROM Person p JOIN FETCH p.roles WHERE p.id = (:id)")

public Person findByIdAndFetchRolesEagerly(@Param("id") Long id);

}

This method will use JPQL's [fetch join](http://docs.oracle.com/html/E24396_01/ejb3_langref.html#ejb3_langref_fetch_joins) clause to eagerly load the roles association in a single round-trip to the database, and will therefore mitigate the performance penalty incurred by the two distinct queries in the above solution.

***How many Hibernate Sessions do you know?***

### Hibernate SessionFactory provides three methods through which we can get Session object – getCurrentSession(), openSession() and openStatelessSession().

### *What is openStatelessSession?*

Hibernate SessionFactory openStatelessSession() method returns instance of StatelessSession. There is another overloaded method where we can pass java.sql.Connection object to get a stateless session object from hibernate.

StatelessSession in Hibernate does not implement first-level cache and it doesn’t interact with any second-level cache. Since it’s stateless, it doesn’t implement transactional write-behind or automatic dirty checking or do cascading operations to associated entities.

Collections are also ignored by a stateless session. Operations performed via a stateless session bypass Hibernate’s event model and interceptors. **It’s more like a normal JDBC connection** and doesn’t provide any benefits that come from using hibernate framework.

However, stateless session can be a good fit in certain situations. For example where we are loading bulk data into database and we don’t want hibernate session to hold huge data in first-level cache memory.

### *What is openSession?*

### Hibernate SessionFactory openSession() method always opens a new session. We should close this session object once we are done with all the database operations.

### You need to explicitly flush and close these session objects.

### We should open a new session for each request in multi-threaded environment. For web application frameworks, we can choose to open a new session for each request or for each session based on the requirement.

### *What is getCurrentSession?*

### Hibernate SessionFactory getCurrentSession() method returns the session bound to the context. But for this to work, we need to configure it in hibernate configuration file like below.

<property name="hibernate.current\_session\_context\_class">thread</property>

### If its not configured to thread, then we will get below exception.

Exception in thread "main" org.hibernate.HibernateException: No CurrentSessionContext configured!

at org.hibernate.internal.SessionFactoryImpl.getCurrentSession(SessionFactoryImpl.java:1012)

### Since this session object belongs to the hibernate context, we don’t need to close it. Once the session factory is closed, this session object gets closed.

### Calling SessionFactory. getCurrentSession will provide you session object which is in hibernate context and managed by hibernate internally. It is bound to transaction scope.

### When you call SessionFactory. getCurrentSession , it creates a new Session if not exists , else use same session which is in current hibernate context. It automatically flush and close session when transaction ends, so you do not need to do externally. Hibernate Session objects are not thread safe, so we should not use it in multi-threaded environment. We can use it in single threaded environment because it’s relatively faster than opening a new session.

**The Current Session can be closed when**

//close session factory

sessionFactory.close();

**SessionFactory.getCurrentSession()**

* It creates a new Session if not exists , else use same session which is in current hibernate context.
* You do not need to flush and close session objects, it will be automatically taken care by Hibernate internally.
* In single threaded environment , It is faster than getOpenSession
* You need to configure additional property. "hibernate.current\_session\_context\_class" to call getCurrentSession method, otherwise it will throw exceptions.

### *Is Hibernate Session is thread safe object?*

### You should never use one session per web app - session is not a thread safe object - cannot be shared by multiple threads. You should always use "one session per request" or "one session per transaction"

***How to use .lock() method in order to reattach a detached object?***

The lock() method can be an important operation when looking at pessimistic operations, but it also has another less know use.  The lock() method, with LockOptions.NONE, can be used to associate a detached object to a session and put the object back into a persistence context.  Importantly, it does this without forcing an update of the database (as the update() or saveOrUpdate() method would).  With LockOptions.NONE, it doesn’t get a database lock and it doesn’t even do a version check against the database.  Importantly, changes that have been made to detached object before a call to the lock() method with LockOptions.NONE **are** **not** synchronized  to the database.  Therefore, only call lock() on detached objects that you know have not been modified outside of a persistent context.

As an example, look at these two code examples.  On top is Hibernate code to reattach a detached object using a typical update method call.  On the bottom is code to reattach a detached object using a .lock method call.

###### UPDATE CODE

1: SessionFactory sf = new Configuration().configure()

2: .buildSessionFactory();

3: Session sess = sf.openSession();

4: Transaction trx = sess.beginTransaction();

5: Vehicle v = (Vehicle) sess.get(Vehicle.class, 1L);

6: trx.commit();

7: sess.close();

8: System.out.println("vehicle now detached: " + v);

9:

10: //update the vehichle outside of session when detached.

11: v.setVin(123);

12:

13: //reattach and update

14: sess = sf.openSession();

15: trx = sess.beginTransaction();

16: sess.update(v);

17: System.out.println("completed the update call");

18: trx.commit();

19: sess.close();

20: System.out.println("vehicle synchronized again: " + v);

###### LOCKING CODE

1: SessionFactory sf = new Configuration().configure()

2: .buildSessionFactory();

3: Session sess = sf.openSession();

4: Transaction trx = sess.beginTransaction();

5: Vehicle v = (Vehicle) sess.get(Vehicle.class, 1L);

6: trx.commit();

7: sess.close();

8: System.out.println("vehicle now detached: " + v);

9:

10: //update the vehichle outside of session when detached.

11: v.setVin(678);

12:

13: //reattach using lock

14: sess = sf.openSession();

15: trx = sess.beginTransaction();

16: sess.**buildLockRequest(LockOptions.NONE).lock(v);**

17: System.out.println("completed the update call");

18: trx.commit();

19: sess.close();

20: System.out.println("vehicle synchronized again: " + v);

***What is Hibernate.isInitialized(object) used for?***

***What is Hibernate.* initialize*(object) used for?***

### If you are lazy-loading a collection normally, but for a particular use, you need to ensure the collection has been loaded before the session is closed, you can use Hibernate.initialize(Object obj) as you noted.

[**Hibernate.initialize(collection)**](http://docs.jboss.org/hibernate/core/3.5/api/org/hibernate/Hibernate.html#initialize%28java.lang.Object%29) This will initialize the collection if it is not yet.

(There is another method - Hibernate.isInitialized(collection), but since you want to actually load the collection, **initialize(..)** is the way to go - it makes the same check before proceeding to initialization)

**Use Hibernate.initialize() within Transactional to initialize lazy objects.**

start Transaction

Hibernate.initialize(entity.getAddresses());

Hibernate.initialize(entity.getPersons());

end Transaction

Now out side of the Transaction you are able to get lazy objects.

entity.getAddresses().size();

entity.getPersons().size();

# [***JPA JoinColumn vs mappedBy***](http://stackoverflow.com/questions/11938253/jpa-joincolumn-vs-mappedby)***?***

<http://stackoverflow.com/questions/11938253/jpa-joincolumn-vs-mappedby>

<http://stackoverflow.com/questions/12038380/how-to-define-unidirectional-onetomany-relationship-in-jpa/12041879#12041879>