HIBERNATE INTERVIEW QUESTION

Q15.What is Locking and explain optimistic and passimistic Locking?

Ans.JPA 2 supports both **optimistic locking** and **pessimistic locking**. Locking is essential to avoid update collisions resulting from simultaneous updates to the same data by two concurrent users. Locking in ObjectDB (and in JPA) is always at the database object level, i.e. each database object is locked separately

## Optimistic Locking

1. ObjectDB maintains a version number for every entity object. The initial version of a new entity object (when it is stored in the database for the first time) is 1. In every transaction in which an entity object is modified its version number is automatically increased by one. Version numbers are managed internally but can be exposed by defining a [version field](https://www.objectdb.com/java/jpa/entity/fields#Version_Field).

During [commit](https://www.objectdb.com/api/java/jpa/EntityTransaction/commit) (and [flush](https://www.objectdb.com/api/java/jpa/EntityManager/flush)), ObjectDB checks every database object that has to be updated or deleted, and compares the version number of that object in the database to the version number of the in-memory object being updated. The transaction fails and an [OptimisticLockException](https://www.objectdb.com/api/java/jpa/OptimisticLockException) is thrown if the version numbers do not match, indicating that the object has been modified by another user (using another [EntityManager](https://www.objectdb.com/api/java/jpa/EntityManager)) since it was retrieved by the current updater

2. Optimistic locking is completely automatic and [enabled by default](https://www.objectdb.com/java/jpa/setting/database#The_locking_element) in ObjectDB,

3.Hibernate provides two different mechanisms for storing versioning information, a dedicated version number or a timestamp.

### Dedicated version number

@Entity

**public** **class** Flight **implements** Serializable {

...

@Version

@Column(name="OPTLOCK")

**public** Integer getVersion() { ... }

}

### Timestamp

Timestamps are a less reliable way of optimistic locking than version numbers, but can be used by applications for other purposes as well. Timestamping is automatically used if you the @Version annotation on a Date or Calendar.

@Entity

**public** **class** Flight **implements** Serializable {

...

@Version

**public** Date getLastUpdate() { ... }

}

**Pessimistic**

Pessimistic locking assumes that concurrent transactions will conflict with each other, and requires resources to be locked after they are read and only unlocked after the application has finished using the data

1.Hibernate always uses the locking mechanism of the database, and never lock objects in memory.

|  |  |
| --- | --- |
| LockMode.WRITE | acquired automatically when Hibernate updates or inserts a row. |
| LockMode.UPGRADE | acquired upon explicit user request using SELECT ... FOR UPDATE on databases which support that syntax. |
| LockMode.UPGRADE\_NOWAIT | acquired upon explicit user request using a SELECT ... FOR UPDATE NOWAIT in Oracle. |
| LockMode.READ | acquired automatically when Hibernate reads data under Repeatable Read or Serializable isolation level. It can be re-acquired by explicit user request. |
| LockMode.NONE | The absence of a lock. All objects switch to this lock mode at the end of a Transaction. Objects associated with the session via a call to update() or saveOrUpdate() also start out in this lock mode.   * [PESSIMISTIC\_READ](https://www.objectdb.com/api/java/jpa/LockModeType/PESSIMISTIC_READ) - which represents a shared lock. * [PESSIMISTIC\_WRITE](https://www.objectdb.com/api/java/jpa/LockModeType/PESSIMISTIC_WRITE) - which represents an exclusive lock. |

**Q1.What is clear and evict and close method in hibernate and why we use them?**

Ans .clear()->**clear():**Completely clear the session and is used to dissociate/disconnect all the objects from the session

2. When this method get called inside transaction boundry then all objects which are currently associate with particular session will be disconnected / clean or no longer associate with that Session instance. Therefore, after calling this method nothing will be performed on persistance layer or DB

 Employee emp1 = (Employee)session.get(Employee.class, 1);

            Employee emp2 = (Employee)session.get(Employee.class, 2);

            //emp1 and emp2 are in persistent state.

            emp1.setEmpName("NEW SREE");

            emp2.setEmpName("NEW KALYAN");

            session.clear();

            //emp1 and emp2 are in detached state.

            session.getTransaction().commit();

            session.close();

* After calling session.clear(), emp1 and emp2 objects are disconnected from the session object. i.e emp1 and emp2 objects are moved from persistent state to detached state.
* The table is not updated even though transaction is committed becuase emp1 and emp2 are in non-transaction state.

2.evict () method->**evict():** Removes the object from the session. This method is used to dissociate/disconnect the specified object from the session.

public void evict(Object object) throws HibernateException

 Employee emp1 = (Employee)session.get(Employee.class, 1);

            Employee emp2 = (Employee)session.get(Employee.class, 2);

            //emp1 and emp2 are in persistent state.

            emp1.setEmpName("NEW KALYAN");

            emp2.setEmpName("NEW SREE");

            session.evict(emp1);

            //emp1 is in detached state and emp2 is in persistent state.

            session.getTransaction().commit();

            session.close();

emp1 is not updated becuase we have called session.evict(emp1) and emp2 is updated.

3.close()method->close the session by calling session.close() after transaction is completed.All the associated objects will be dissociated after calling session.close().It is not  strictly necessary to close the session but you must at least using disconnct it using session.disconnect()

public Connection close() throws HibernateException

Q2.What is hibernate state?

Ans. There are three types of Hibernate object states

## 1. Transient Object State

1. An object which is not associated with hibernate session and does not represent a row in the database is considered as transient. It will be garbage collected if no other object refers to it.

An object that is created for the first time using the new() operator is in transient state. When the object is in transient state then it will not contain any identifier (primary key value). You have to use save, persist or saveOrUpdate methods to persist the transient object.

Employee emp = new Employee();

emp.setName("Ravi Raj");

## 2. Persistent Object State:

An object that is associated with the hibernate session is called as Persistent object. When the object is in persistent state, then it represent one row of the database and consists of an identifier value.You can make a transient instance persistent by associating it with a Session.

Long id = (Long) session.save(emp);

// emp object is now in a persistent state

## 3. Detached Object State:

Object which is just removed from hibernate session is called as detached object.The state of the detached object is called as detached state.

Any changes made to the detached objects are not saved to the database. The detached object can be reattached to the new session and save to the database using update, saveOrUpdate and merge methods

session.close();

Q1.WHAT IS HIBERNATE ?

Ans.1. **Object-relational mapping** or ORM is the programming technique to map application domain model objects to the relational database tables. Hibernate is java based ORM tool that provides framework for mapping application domain objects to the relational database tables and vice versa.

2. Hibernate framework provide option to map plain old java objects to traditional database tables with the use of JPA annotations as well as XML based configuration

3. **What are the advantages of Hibernate over JDBC?**(detailed answer)  
Apart from Persistence i.e. saving and loading data from Database, Hibernate also provides following benefits  
1) Caching  
2) Lazy Loading  
3) Relationship management and provides code for mapping an object to the data  
4) The developer is free from writing code to load/store data into the database.  
  
  
Q2.Difference between get() vs load() method in Hibernate?  
Ans.1. get method of Hibernate Session class returns null if object is not found in cache as well as on database while load() method [throws](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html) ObjectNotFoundException if object is not found on cache as well as on database but never return null.

2. **2. Database hit**

Get method always hit [database](http://javarevisited.blogspot.sg/2011/10/how-to-use-truncate-and-delete-command.html) while load() method may not always hit the database, depending upon which method is called.

3. **3. Proxy**

Get method never returns a proxy, it either returns null or fully initialized Object, while load() method may return proxy, which is the object with ID but without initializing other properties, which is lazily initialized. If you are just using returned object for creating relationship and only need Id then load() is the way to go

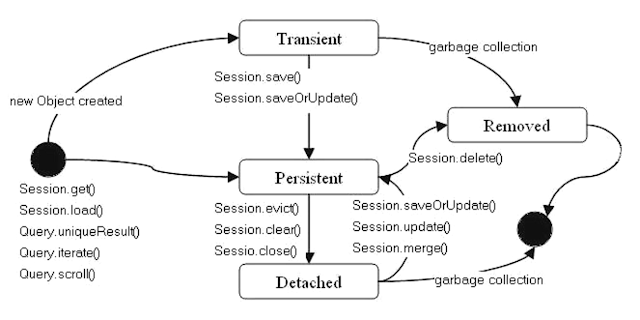
4.Load method return peroxy so its performance is more better than get method   
  
5. load method exists prior to get method which is added on user request.

Q4.When to use Session get() and load() in Hibernate  
Ans. 1. Use get method to determine if an [instance](http://javarevisited.blogspot.sg/2012/02/difference-between-instance-class-and.html) exists or not because it can return null if instance doesn’t exists in cache and database and use load method to retrieve instance only if you think that instance should exists and non availability is an error condition.

**Session** session = SessionFactory.getCurrentSession();  
**Employee** Employee = (Employee) session.get(Employee.**class**, EmployeeID);  
  
*//Example of calling load method of Hiberante Session*  
**Session** session = SessionFactory.getCurrentSession();  
**Employee** Employee = (Employee) session.load(Employee.**class**, EmployeeID);  
  
Q5.What is the difference between save() and persist() method in Hibernate?

Ans. Session interface in Hibernate provides a couple of methods to move an object from new or transient state to persistent state e.g. save(), saveOrUpdate() and persist() are used to store an object into the database,

2. The **Session.save()** method does an INSERT to store the object into the database and it also return the identifier generated by the database. On the other hand, **saveOrUpdate()** can be used to reattach a detached object in Hibernate Session i.e. it can do INSERT or UPDATE depending upon whether object exists in database or not. The third method, persist() is similar to Session.save() i.e. it is used to move a transient object to the persistent state by storing it into the database but it doesn't return the database identifier.

3. 1) The main difference between save() and saveOrUpdate() method is that save() method performs an INSERT operation to store the object into the database, but INSERT will fail if the [primary key](http://java67.blogspot.com/2015/12/difference-between-primary-and-foreign.html) is already persistent i.e. object already exists in the database. This is why, you should only call save() with an absolutely new object which doesn't have any database identifier. Calling save() with the detached object will fail. This is opposite of **saveOrUpdate()** method, which can do either INSERT or UPDATE SQL query depending upon whether an object exists in the database or not. The saveOrUpdate() method first executes a [SELECT query](http://javarevisited.blogspot.com/2011/10/selct-command-sql-query-example.html) to determine if it needs to do an INSERT or UPDATE operation  
  
4) Another key difference between save() and saveOrUpdate() method is that former is used to bring a **transient object** to **persistent state** but saveOurUpdate() can bring both transient (new) and detached (existing) object into persistent state. It is often used to re-attach a detached object into Session.  
  
  
4. 3) Coming to **persist()** method, the difference between save() and persist() method is that former returns the generated database identifier, a [Serializable object](http://java67.blogspot.com/2012/08/what-is-serialization-in-java.html) but persist() method doesn't return anything. It's return type is void.

Q6. Why JPA Entity or Hibernate Persistence Class Should Not be Final?  
  
Ans. **making a JPA Entity or Hibernate Persistence class final, limits the ability of Hibernate to use Proxies**, which in turn prevent Hibernate from applying some performance optimizations. Without proxies, your application loses *lazy loading*, and l*azy association fetching* will issue more SQL queries and make more database roundtrip, which will cost performance.  
  
The only way to make a Hibernate entity class final without losing lazy behavior is to use interface and define all public methods of persistence class there, this will still allow Hibernate to use a proxy in place of real class.

Q7. What are different types of caches available in Hibernate?

Ans. Hibernate provides the out-of-box caching solution but there are many

caches e.g.

1 first level cache, (Session level,)

2.second level cache and query cache. (SessionFactory level)

First level cache is maintained at Session level and cannot be disabled but the second level cache is required to be configured with external cache provider like EhCache.

1. The first level cache minimizes database access for the same object. For example, if you call the get() method to access Employee object with id = 1 from one session, it will go the database and load the object into memory, but it will also cache the object in the first level cache.  
     
   When you will call the get() method again for the same object from the same session, even after doing some updates on the object, it will return the object from the cache without accessing the database
2. if an object is modified several times within the same transaction, then Hibernate will only generate one SQL UPDATE statement at the end of the transaction, containing all the modification.
3. But, since this cache is associated with the Session object, which is a short-lived object in Hibernate, as soon as you close the session, all the information held in the cache is lost.
4. Hibernate provides another application level cache, known as [second level cache](http://javarevisited.blogspot.sg/2013/05/10-hibernate-interview-questions-answers-java-j2ee-senior.html), which can be shared among multiple sessions
5. The second level cache is maintained at the SessionFactory level, which is used to open sessions, hence every session is linked to SessionFactory. This cache is opposite to first level cache which is by default enabled in Hibernate, this one is by default disabled and you need to configure the second level cache in Hibernate configuration file to enable it.

Difference between First and Second Level Cache in Hibernate.

1. **Scope**  
First level cache is associated with Session Object, while the Second level cache is associated with the SessionFactory object. This means first level cache's scope is limited to session level while second level cache's scope is at the application level.  
  
2. **Configuration**  
First level cache is by default enabled in Hibernate, while the second level cache is optional. If you need it then you need to explicitly enable the second level cache on Hibernate configuration file i.e. the hibernate.cfg.xml file.  
  
Here is a sample configuration to configure Second level cache with EhCache:

<prop key="hibernate.cache.use\_second\_level\_cache">true</prop>

<prop key="hibernate.cache.provider\_class">org.hibernate.cache.EhCacheProvider</prop>

**3.Availability**

**1.** First level cache is available only until the session is open, once the session is closed, the first level cache is destroyed  
  
**Order**  
If an entity or object is loaded by calling the [get()](http://javarevisited.blogspot.sg/2012/07/hibernate-get-and-load-difference-interview-question.html) method then Hibernate first checked the first level cache, if it doesn't found the object then it goes to the second level cache if configured. If the object is not found then it finally goes to the database and returns the object, if there is no corresponding row in the table then it return [null](http://javarevisited.blogspot.sg/2014/12/9-things-about-null-in-java.html). When an object is loaded from the database is put on both second level and first level cache, so that other session who request the same object can now get it from the second level cache.  
  
Q7. **What are the three states of a Hibernate Persistence object can be?**(detailed answer)  
The Hibernate persistent or entity object can live in following three states:  
1) transient  
2) persistent  
3) detached  
  
**Q8.How do you log SQL queries issued by the Hibernate framework in Java application?**  
You can use the show\_sql property to log SQL queries issued by the Hibernate framework, Just add the following line in your Hibernate configuration file:

<property name=”show\_sql”> true </property>

**Q9.What is the difference between the transient, persistent and detached state in Hibernate?**  
New objects created in Java program but not associated with any hibernate Session are said to be in the transient state. On the other hand, an object which is associated with a Hibernate session is called Persistent object. While an object which was earlier associated with Hibernate session but currently it's not associate is known as a detached object. You can call save() or persist() method to store those object into the database and bring them into the Persistent state. Similarly, you can re-attach a detached object to hibernate sessions by calling either update() or saveOrUpdate() method.  
  
Q10.The difference between sorted and ordered collection in Hibernate?  
  
Ans. The main difference between sorted and ordered collection is that sorted collection sort the data in JVM's heap memory using Java's collection framework sorting methods while ordered collection is sorted using order by clause in the database itself. A sorted collection is more suited for small dataset but for a large dataset, it's better to use ordered collection to avoid [OutOfMemoryError in Java](http://java67.blogspot.com/2013/08/guide-of-javalangoutofmemoryerror-java-heap-space-tomcat-eclipse-minecraft-jboss.html) application.  
  
**Q11.When do you use merge() and update() in Hibernate?**  
Ans. You should use update() if you are sure that the Hibernate session does not contain an already persistent instance with the same id and use merge() if you want to merge your modifications at any time without considering the state of the session

**Update:**Suppose we are dealing with any employee object in the same session then we should use update() or saveOrUpdate() method.

**Update:**if you are sure that the session does not contains an already persistent instance with the same identifier,then use update to save the data in hibernate

**Merge:**Suppose we are creating a session and load an employee object. Now object in session cache. If we close the session at this point and we edit state of object and tried to save using update() it will throw exception. To make object persistent we need to open another session. Now we load same object again in current session. So if we want to update present object with previous object changes we have to use merge() method. Merge method will merge changes of both states of object and will save in database.

**Merge:**if you want to save your modifications at any time without knowing about the state of an session, then use merge() in hibernate.

Employee emp1 = new Employee();

emp1.setEmpId(100);

emp1.setEmpName("Dinesh");

//create session

Session session1 = createNewHibernateSession();

session1.saveOrUpdate(emp1);

session1.close();

//emp1 object in detached state now

emp1.setEmpName("Dinesh Rajput");//Updated Name

//Create session again

Session session2 = createNewHibernateSession();

Employee emp2 =(Employee)session2.get(Employee.class, 100);

//emp2 object in persistent state with id 100

//below we try to make on detached object with id 100 to persistent state by using update method of hibernate

session2.update(emp1);//It occurs the exception NonUniqueObjectException because emp2 object is having employee with same empid as 100. In order //to avoid this exception we are using merge like given below instead of session.update(emp1);

session2.merge(emp1); //it merge the object state with emp2

session2.update(emp1); //Now it will work with exception

Q12. **Does Hibernate Session interface is thread-safe in Java?**(detailed answer)  
No, Session object is not thread-safe in Hibernate and intended to be used with-in single thread in the application.  
  
**Q13.Does SessionFactory is thread-safe in Hibernate?**([detailed answer](http://javarevisited.blogspot.com/2013/05/10-hibernate-interview-questions-answers-java-j2ee-senior.html))  
Ans.SessionFactory is both Immutable and thread-safe and it has just one single instance in Hibernate application. It is used to create Session object and it also provide caching by storing SQL queries stored by multiple session. The second level cache is maintained at SessionFactory level  
**Q14.What is different between Session and Sessionfactory in Hibernate?**  
AnsThe main difference between Session and SessionFactory is that former is a single-threaded, short-lived object while later is Immutable and shared by all Session. It also lives until the Hibernate is running. Another difference between Session and SessionFactory is that former provides first level cache while SessionFactory provides the Second level cache.

Q15. **What is criterion query in hibernate?**(detailed answer)  
Criteria is a simplified API for retrieving entities by composing Criterion objects also known as Criterion query. This is a very convenient approach for functionality like "search" screens where you can filter data on multiple conditions as shown in the following example:

List books = session.createCriteria(Book.class)

.add(Restrictions.like("name", "java%") )

.add(Restrictions.like("published\_year", "2015"))

.addOrder(Order.asc("name") )

.list();

### Q17.What is difference between getCurrentSession() and openSession() in Hibernate? Ans. openSession

When you call SessionFactory.openSession, it always create new Session object afresh and give it to you. You need to explicitly flush and close these session objects. As session objects are not thread safe, you need to create one session object per request in multithreaded environment and one session per request in web applications too.

### getCurrentSession

When you call SessionFactory. getCurrentSession, it 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.

If you are using hibernate in single threaded environment , you can use getCurrentSession, as it is faster in performance as compare to creating  new session each time.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **openSession** | **getCurrentSession** |
| Session object | It always create new Session object | It creates a new Session if not exists , else use same session which is in current hibernate context |
| Flush and close | You need to explicitly flush and close session objects | You do not need to flush and close session objects, it will be automatically taken care by Hibernate internally |
| Performance | In single threaded environment , It is slower than getCurrentSession | In single threaded environment , It is faster than getOpenSession |
| Configuration | You do not need to configure any property to call this method | You need to configure additional property “hibernate.current\_session\_context\_class” to call getCurrentSession method, otherwise it will throw exceptions. |

# Q1. [Advantages and disadvantages of hibernate compared to jdbc](http://www.instanceofjava.com/2016/10/advantages-and-disadvantages-hibernate.html)

1. Hibernate is an ORM tool
2. Hibernate is an open source framework.
3. Better than JBDC.
4. Hibernate has an exception translator , which converts checked exceptions of JDBC in to unchecked exceptions of hibernate. So all exceptions in hibernate are unchecked exceptions and Because of this no need to handle exceptions explicitly.
5. Hibernate supports inheritance and polymorphism.
6. With hibernate we can manage the data stored across multiple tables, by applying relations(association)
7. Hibernate has its own query language called Hibernate Query Language. With this HQL hibernate became database independent.
8. Hibernate supports relationships like One-To-One, One-To-Many, Many-To-One ,Many-To-Many.
9. Hibernate has Caching mechanism. using this number of database hits will be reduced. so performance of an application will be increases.
10. Hibernate supports lot of databases.
11. [Hibernate supported databases List](http://www.instanceofjava.com/2016/10/hibernate-supported-databases-list.html).
12. Hibernate is a light weight framework because hibernate uses POJO classes for data transfer between application and database.
13. Hibernate has versioning and time stamp feature with this we can know how many number of times data is modified.
14. Hibernate also supports annotations along with XML.
15. Hibernate supports Lazy loading.
16. Hibernate is easy to learn it is developers friendly.
17. The architecture is layered to keep you isolated from having to know the underlying APIs.
18. Hibernate maintains database connection pool.
19. Hibernate  has Concurrency support.
20. Using Hibernate its Easy to maintain and it will increases productivity

------------------------Disadvantages of Hibernate----------------------------------

1.Sometime debugging and tuning become difficult.

2.Slower than JDBC BECAUSE it is generating lot of run time query.

3.it usel0ess for small project