Hibernate

What is Hibernate framework?

Hibernate is a popular Object Relational Mapping (ORM) framework of Java. It helps in mapping the Object-Oriented Domain model to Relational Database tables. Hibernate is a free software distributed under GNU license.
Hibernate also provides implementation of Java Persistence API (JPA).
In simple words, it is a framework to retrieve and store data from database tables from Java.

What is an Object Relational Mapping (ORM)?

Object Relational Mapping (ORM) is a programming technique to map data from a relational database to Object oriented domain model. This is the core of Hibernate framework.

In case of Java, most of the software is based on OOPS design. But the data stored in Database is based on Relation Database Management System (RDBMS).

ORM helps in data retrieval in an Object-Oriented way from an RDBMS. It reduces the effort of developers in writing queries to access and insert data.

211. What is the purpose of Configuration Interface in Hibernate?

Configuration interface can be implemented in an application to specify the properties and mapping documents for creating a SessionFactory in Hibernate.

By default, a new instance of Configuration uses properties mentioned in hibernate.properties file.

Configuration is mainly an initialization time object that loads the properties in helps in creating SessionFactory with these properties.

In short, Configuration interface is used for configuring Hibernate framework in an application.

212. What is Object Relational Impedance Mismatch?

Object Relational Impedance Mismatch (ORIM) is also known as paradigm mismatch. It means that Object model and Relational model do not work well with each other.

Relational model or a RDBMS represents data in tabular format like a spreadsheet. Object model or OOPS represents the data as an inter-connected graph of objects.

Mixing these two models leads to various problems. The common name for these issues is Object Relational Impedance Mismatch.

213. What are the main problems of Object Relational Impedance Mismatch?

Object model and Relational models (RDBMS) have following problems that are part of Object Relational Impedance Mismatch:

Granularity: Object model is more granular than Relational model. There are more classes in object model than the corresponding tables in relational model.

Inheritance: Object model supports inheritance. But Relational model does not have any concept of inheritance.

Identity: Relational model has just one criteria for sameness of data. It is based on primary key. In object model like Java we can have equals as well as == for sameness of objects.

Associations: In Object model associations are uni-directional. In RDBMS, there is a concept of foreign key for association. Also multiplicity of a relationship is hard to judge by looking at object model.

Data navigation: In Object model, you can move from one object to another object for getting data. Egg. you can retrieve and Employee object, then go to its department object and then get the employees in the department object. In RDBMS, we try to minimize the SQL calls, so we get all the data by using joins.

214. What are the key characteristics of Hibernate?

Hibernate has following key characteristics:

Object/Relational Mapping (ORM): Hibernate provides ORM capabilities to developers. So then can write code in Object model for connecting with data in Relational model.

JPA Provider: Hibernate provides an excellent implementation of Java Persistence API (JPA) specification.

Idiomatic persistence: Hibernate provides persistence based on natural Object-oriented idioms with full support for inheritance, polymorphism, association,

composition, and the Java collections framework. It can work with any data for persistence.

High Performance: Hibernate provides high level of performance supporting features like- lazy initialization, multiple fetching strategies, optimistic locking etc. Hibernate does not need its own database tables or fields. It can generate SQL at system initialization to provide better performance at runtime.

Scalability: Hibernate works well in multi server clusters. It has built in scalability support. It can work well for small projects as well as for large business software.

Reliable: Hibernate very reliable and stable framework. This is the reason for its worldwide acceptance and popularity among developer community.

Extensible: Hibernate is quite generic in nature. It can be configured and extended as per the use case of application.

215. Can you tell us about the core interfaces of Hibernate framework?

The core interfaces of Hibernate framework are as follows:

Configuration: Configuration interface can be implemented in an application to specify the properties and mapping documents for creating a SessionFactory in Hibernate. Hibernate application bootstraps by using this interface.

SessionFactory: In Hibernate, SessionFactory is used to create and manage Sessions. Generally, there is one SessionFactory created for one database. It is a thread-safe interface that works well in multi-threaded applications.

Session: Session is a lightweight object that is used at runtime between a Java application and Hibernate. It contains methods to create, read and delete operations for entity classes. It is a basic class that abstracts the concept of persistence.

Transaction: This is an optional interface. It is a short-lived object that is used for encapsulating the overall work based on unit of work design pattern. A Session can have multiple Transactions.

Query: This interface encapsulates the behavior of an object-oriented query in Hibernate. It can accept parameters and execute the queries to fetch results. Same query can be executed multiple times.

Criteria: This is a simplified API to retrieve objects by creating Criterion objects. It is very easy to use for creating Search like features.

216. How will you map the columns of a DB table to the properties of a Java class in Hibernate?

We can map the class properties and table columns by using one of the two ways:

XML: We can map the column of a table to the property of a class in XML file. It is generally with extension hbm.xml

Annotation: We can also use annotations @Entity and @Table to map a column to the property of a class.

217. Does Hibernate make it mandatory for a mapping file to have .hbm.xml extension?

No. It is a convention to have.hbm.xml extension in the name of a mapping file. It is not a requirement enforced by Hibernate. We can use any other extension of our convenience for this.

218. What are the steps for creating a SessionFactory in Hibernate?

Steps to create a SessionFactory in Hibernate are:

Configuration: First create a Configuration object. This will refer to the path of configuration file.
Resource: Add config file resource to Configuration object.
Properties: Set properties in the Configuration object.
SessionFactory: Use Configuration object to build SessionFactory.

Egg.

```
Configuration config = new Configuration();
config.addResource("testInstance/configuration.hbm.xml");
config.setProperties( System.getProperties() ); SessionFactory sessions
= config.buildSessionFactory();
```

219. Why do we use POJO in Hibernate?

POJO stands for Plain Old Java Objects. A POJO is java bean with getter and setter methods for each property of the bean.

It is a simple class that encapsulates an object's properties and provides access through setters and getters.

Some of the reasons for using POJO in Hibernate are:

POJO emphasizes the fact that this class is a simple Java class, not a heavy class like EJB.

POJO is a well-constructed class, so it works well with Hibernate proxies.

POJO also comes with a default constructor that makes it easier to persist with a default constructor.

220. What is Hibernate Query Language (HQL)?

Hibernate Query Language is also known as HQL. It is an Object Oriented language. But it is similar to SQL.

HQL works well with persistent objects and their properties. HQL does not work on database tables.

HQL queries are translated into native SQL queries specific to a database.

HQL supports direct running of native SQL queries also. But it creates an issue in Database portability.

221. How will you call a stored procedure in Hibernate?

Hibernate supports executing not only simple queries but also stored procedure of database. There are three ways to call a stored procedure in Hibernate:

XML mapping file:
We can declare the store procedure inside XML Mapping file.

```
<!-- Employee.hbm.xml -->
...
<hibernate-mapping>
                        <class                    name="com.testHibernate.util.Employee"
table="employee" ...>
        <id name="employeeId" type="java.lang.Integer"> <column
            name="EMPLOYEE_ID" /> <generator class="identity"
            />
        </id>
        <property name="employeeId" type="string">
                        <column name="EMPLOYEE_ID" length="10" not-null="true"
unique="true" />
        </property>
</class>

            <sql-query name="callEmployeeStoreProcedure">
  <return alias="employee" class="com.testHibernate.util.Employee"/>
  <![CDATA[CALL GetEmployees(:employeeId)]]> </sql-
    query>

</hibernate-mapping>
```

We can call it with getNamedQuery().

```
Query                                    query                                    =
```

```
        session.getNamedQuery("callEmployeeStoreProcedure")
  .setParameter("employeeId", “1234”);
List result = query.list();
for(int i=0; i<result.size(); i++){
  Employee employee = (Employee)result.get(i);
  System.out.println(employee.getEmployeeCode());
}
```

Native SQL: We can use Native SQL to call a store procedure query directly. In this example GetEmployees() stored procedure is being called.

```
Query query = session.createSQLQuery( "CALL
  GetEmployees(:employeeId)")
  .addEntity(Employee.class)
  .setParameter("employeeId", “1234);

List result = query.list();
for(int i=0; i<result.size(); i++){
  Employee employee = (Employee) result.get(i);
  System.out.println(employee.getEmployeeCode());
}
```

Use annotation:

We can also mark out stored procedure with @NamedNativeQueries annotation.

```
//Employee.java

@NamedNativeQueries({
  @NamedNativeQuery(
  name = "callEmployeeStoreProcedure", query = "CALL
  GetEmployees(:employeeId)", resultClass =
  Employee.class )
})
@Entity
```

```
@Table(name = "employee")
public class Employee implements java.io.Serializable {
...
```

Call it with getNamedQuery().

```
Query query = session.getNamedQuery("callEmployeeStoreProcedure")
  .setParameter("employeeId", “1234”);
List result = query.list();
for(int i=0; i<result.size(); i++){
  Employee employee = (Employee)result.get(i);
  System.out.println(employee.getEmployeeCode());
}
```

222. What is Criteria API in Hibernate?

Criteria is a simplified API in Hibernate to get entities from database by creating Criterion objects.

It is a very intuitive and convenient approach for search features. Users can specify different criteria for searching entities and Criteria API can handle these.

Criterion instances are obtained through factory methods on Restrictions.

223. Why do we use HibernateTemplate?

This is a trap question. HibernateTemplate has been deprecated. There were earlier good reasons to use HibernateTemplate. But now the trend has changed towards not using it anymore.

224. How can you see SQL code generated by Hibernate on console?

To display the SQL generated by Hibernate, we have to turn on the show_sql flag.

This can be done in Hibernate configuration as follows:

```
<property name=”show_sql”>true</property>
```

225. What are the different types of collections supported by Hibernate?

Hibernate supports following two types of collections:

Indexed Collections: List and Maps
Sorted Collections: java.util.SortedMap and java.util.SortedSet

226. What is the difference between session.save() and session.saveOrUpdate() methods in Hibernate?

Save method first stores an object in the database. Then it persists the given transient instance by assigning a generated identifier. Finally, it returns the id of the entity that is just created.

SaveOrUpdate() method calls either save() or update() method. It selects one of these methods based on the existence of identifier.

If an identifier exists for the entity then update() method is called. If there is no identifier for the entity then save() method is called as mentioned earlier.

227. What are the advantages of Hibernate framework over JDBC?

Main advantages of Hibernate over JDBC are as follows:

Database Portability: Hibernate can be used with multiple types of database with easy portability. In JDBC, developer has to write database specific native queries. These native queries can reduce the database portability of the code.

Connection Pool: Hibernate handles connection pooling very well.
JDBC requires connection pooling to be defined by developer.

Complexity: Hibernate handles complex query scenarios very well with its internal API like Criteria. So developer need not gain expertise in writing complex SQL queries. In JDBC application developer writes most of the queries.

228. How can we get statistics of a SessionFactory in Hibernate?

In Hibernate we can get the statistics of a SessionFactory by using Statistics interface. We can get information like Close Statement count, Collection Fetch count, Collection Load count, Entity insert count etc.

229. What is the Transient state of an object in Hibernate?

When an object is just instantiated using the new operator but is not associated with a Hibernate Session, then the object is in Transient state.

In Transient state, object does not have a persistent representation in database. Also there is no identifier assigned to an object in Transient state.

An object in Transient state can be garbage collected if there is no reference pointing to it.

230. What is the Detached state of an object in Hibernate?

An object is in detached state if it was persistent earlier but its Session is closed now.

Any reference to this object is still valid. We can even update this object. Later on we can even attach an object in detached state to a new session and make it persistent.

Detached state is very useful in application transactions where a user takes some time to finish the work.

231. What is the use of Dirty Checking in Hibernate?

Dirty Checking is very useful feature of Hibernate for write to database operations. Hibernate monitors all the persistent objects for any changes. It can detect if an object has been modified or not.

By Dirty Checking, only those fields of an object are updated that require any change in them. It reduces the time-consuming database write operations.

232. What is the purpose of Callback interface in Hibernate?

Callback interface in Hibernate is mainly used for receiving notifications of different events from an object.

Egg. We can use Callback to get the notification when an object is loaded into or removed from database.

233. What are the different ORM levels in Hibernate?

There are following four different ORM levels in Hibernate:

Pure Relational ORM: At this level entire application is designed around the relational model. All the operations are SQL based at this level.

Light Object Mapping: At this level entity classes are mapped manually to relational tables. Business logic code is hidden from data access code. Applications with less number of entities use this level.

Medium Object Mapping: In this case, application is designed around an object model. Most of the SQL code is generated at compile time. Associations between objects are supported by the persistence mechanism. Object-oriented expression language is used to specify queries.

Full Object Mapping: This is one of the most sophisticated object modeling level. It supports composition, inheritance, polymorphism and persistence. The persistent classes do not inherit any special base class at this level. There are efficient fetching and caching strategies implemented transparently to the application.

234. What are the different ways to configure a Hibernate application?

There are mainly two ways to configure Hibernate application:
XML based: We can define the Hibernate configuration in an XML file like hibernate.cfg.xml file
Programming based: We can also use code logic to configure Hibernate in our application.

235. What is Query Cache in Hibernate?

Hibernate provides Query Cache to improve the performance of queries that run multiple times with same parameters.
At times Query Caching can reduce the performance of Transactional processing. By default Query Cache is disabled in Hibernate.
It has to be used based on the benefits gained by it in performance of the queries in an application.

236. What are the different types of Association mappings supported by Hibernate?

Hibernate supports following four types of Association mappings:
Unidirectional association: This kind of association works in only one direction.
Unidirectional association with join tables
Bidirectional association: This kind of association works in both directions.
Bidirectional association with join tables

237. What are the different types of Unidirectional Association mappings in Hibernate?

In Hibernate there can be following three types of Unidirectional Association mappings:
Many to one
One to one
One to many

238. What is Unit of Work design pattern?

Unit of Work is a design pattern to define business transactions.

A Unit of Work is a list of ordered operations that we want to run on a database together. Either all of these go together or none of these goes.

Most of the time, we use term business transaction in place of Unit of Work.

Egg. In case of money transfer from account A to B, the unit of work can be two operation Debit account A and Credit account B in a sequence. Both these operations should happen together and in right sequence.

239. In Hibernate, how can an object go in Detached state?

Once the session attached to an Object is closed, the object goes into Detached state. An Object in Detached state can be attached to another session at a later point of time.

This state is quite useful in concurrent applications that have long unit of work.

240. How will you order the results returned by a Criteria in Hibernate?

Hibernate provides an Order criterion that can be used to order the results. This can be order objects based on their property in ascending or descending order.

Class is org.hibernate.criterion.Order.

One example is as follows:

Egg.

```
List employees = session.createCriteria(Employee.class)
    .add( Restrictions.like("name", "F%")
    .addOrder( Order.asc("name") )
    .addOrder( Order.desc("age") )
    .setMaxResults(10)
    .list();
```

241. How does Example criterion work in Hibernate?

In Hibernate, we can create an object with desired properties. Then we can use this object to search for objects with similar object. For this we can use org.hibernate.criterion.Example criterion.

Egg. First we create a sample book object of author Richard and category mystery. Then we search for similar books.

```
Book book = new Book();
book.setAuthor('Richard');
book.setCategory(Category.MYSTERY);
List results = session.createCriteria(Book.class)
    .add( Example.create(book) )
    .list();
```

242. How does Transaction management work in Hibernate?

In Hibernate we use Session interface to get a new transaction. Once we get the transaction we can run business operations in that transaction. At the end of successful business operations, we commit the transaction. In case of failure, we rollback the transaction.

Sample code is a follows:

```
Session s = null;
Transaction trans = null;
try {
s = sessionFactory.openSession();
trans = s.beginTransaction();
doTheAction(s);
trans.commit();
} catch (RuntimeException exc) {
trans.rollback();
} finally {
s.close();
}
```

243. How can we mark an entity/collection as immutable in Hibernate?

In Hibernate, by default an entity or collection is mutable. We can add, delete or update an entity/collection.
To mark an entity/collection as immutable, we can use one of the following:
@Immutable: We can use the annotation @Immutable to mark an entity/collection immutable.
XML file: We can also set the property mutable=false in the XML file for an entity to make it immutable.

244. What are the different options to retrieve an object from database in Hibernate?

In Hibernate, we can use one of the following options to retrieve objects from database:
Identifier: We can use load() or get() method and pass the identifier like primary key to fetch an object from database.
HQL: We can create a HQL query and get the object after executing the query.
Criteria API: We can use Criteria API to create the search conditions for getting the objects from database.
Native SQL: We can write native SQL query for a database and just execute it to get the data we want and convert it into desired object.

245. How can we auto-generate primary key in Hibernate?

We can use the primary key generation strategy of type GenerationType.AUTO to auto-generate primary key while persisting an object in Hibernate.

Egg.
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
private int id;
We can leave it null/0 while persisting and Hibernate automatically generates a primary key for us.
Sometimes, AUTO strategy refers to a SEQUENCE instead of an IDENTITY.

246. How will you re-attach an object in Detached state in Hibernate?

We can call one of the methods Session.update(), Session.saveOrUpdate(), or Session.merge() to re-attach an object in detached state with another session in Hibernate.

247. What is the first level of cache in Hibernate?

A Hibernate Session is the first level of cache for persistent data in a transaction.
The second level of cache is at JVM or SessionFactory level.

248. What are the different second level caches available in Hibernate?

In Hibernate, we can use different cache providers for implementing second level cache at JVM/SessionFactory level.

Some of these are:

Hashtable
EHCache
OSCache
SwarmCache
JBoss Cache 1.x
JBoss Cache 2

249. Which is the default transaction factory in Hibernate?

In Hibernate, default transaction factory is JDBCTransactionFactory. But we can change it by setting the property hibernate.transaction.factory_class.

250. What are the options to disable second level cache in Hibernate?

This is a trick question. By default Second level cache is already disabled in Hibernate.
In case, your project is using a second level cache you can use one of the following options to disable second level cache in Hibernate:
We can set hibernate.cache.use_second_level_cache to false.
We can use CacheMode.IGNORE to stop interaction between the session and second-level cache. Session will interact with cache only to invalidate cache items when updates occur

251. What are the different fetching strategies in Hibernate?

Hibernate 3 onwards there are following fetching strategies to retrieve associated objects:

Join fetching: In Join strategy Hibernate uses OUTER join to retrieve the associated instance or collection in the same SELECT.

Select fetching: In Select strategy, Hibernate uses a second SELECT to retrieve the associated entity or collection. We can explicitly disable lazy fetching by specifying lazy="false". By default lazy fetching is true.

Subselect fetching: In Subselect strategy, Hibernate uses a second SELECT to retrieve the associated collections for all entities retrieved in a previous query or fetch.

Batch fetching: In Batch strategy, Hibernate uses a single SELECT to retrieve a batch of entity instances or collections by specifying a list of primary or foreign keys. This is a very good performance optimization strategy for select fetching.

252. What is the difference between Immediate fetching and Lazy collection fetching?

In Immediate fetching an association, collection or attribute is retrieved at the same time when the owner is loaded.
But in Lazy collection fetching, a collection is fetched only when an operation is invoked on that collection by client application.

This is the default fetching strategy for collections in Hibernate.

Lazy fetching is better from performance perspective.

253. What is 'Extra lazy fetching' in Hibernate?

In Extra lazy fetching, only individual elements of a collection are fetched from the database when they are required.

In this strategy, Hibernate does not fetch the whole collection into memory unless it is essential.

It is a good fetching strategy for large collections of objects.

254. How can we check is a collection is initialized or not under Lazy Initialization strategy?

Hibernate provides two convenient methods, Hibernate.initialize() and Hibernate.isInitialized() to check whether a collection is initialized or not.

By using Hibernate.initialize() we can force the initialization of a collection in Hibernate.

255. What are the different strategies for cache mapping in Hibernate?

Hibernate provides following strategies for cache mapping:

Read only: If an application requires caching only for read but not for write operations, then we can use this strategy. It is very simple to use and give very good performance benefit.

It is also safe to use in a cluster environment.

Read/Write: If an application also needs caching for write operations, then we use Read/Write strategy.

Read/write cache strategy should not be used if there is requirement for serializable transaction isolation level.

If we want to use it in a cluster environment, we need to implement locking mechanism.

Nonstrict Read/Write: If an application only occasionally updates the data, then we can use this strategy. It cannot be used in systems with serializable transaction isolation level requirement.

Transactional: This strategy supports full transactional cache providers like JBoss TreeCache.

256. What is the difference between a Set and a Bag in Hibernate?

A Bag in Hibernate is an unordered collection. It can have duplicate elements. When we persist an object in a bag, there is no guarantee that bag will maintain any order.

A Set in Hibernate can only store unique objects. If we add the same element to set second time, it just replaces the old one. By default a Set is unordered collection in Hibernate.

257. How can we monitor the performance of Hibernate in an application?

We can use following ways to monitor Hibernate performance:

Monitoring SessionFactory: Since there is one SessionFactory in an application, we can collect the statistics of a SessionFactory to
monitor the performance. Hibernate provides sessionFactory.getStatistics() method to get the statistics of SessionFactory.
Hibernate can also use JMX to publish metrics.
Metrics: In Hibernate we can also collect other metrics like-number of open sessions, retrieved JDBC connections, cache hit, miss etc.
These metrics give great insight into the performance of Hibernate. We can tune Hibernate settings and strategies based on these metrics.

258. How can we check if an Object is in Persistent, Detached or Transient state in Hibernate?

We can use following methods to check the state of an object in
Hibernate:
Persistent State: If call to EntityManager.contains(object) returns true, the object is in Persistent state.
Detached State: If the call to PersistenceUnitUtil.getIdentifier(object) returns identifier property then the object is in detached state.
Transient State: If call to PersistenceUnitUtil.getIdentifier(object) returns null then object is in Transient state.
We can get access to PersistenceUnitUtil from the EntityManagerFactory in Hibernate.

259. What is 'the inverse side of association' in a mapping?

Let us consider an example in which a customer can have multiple orders and for every order there has to be a customer.

In OO world, customer is the owner of order. In SQL world, an Order has reference to customer id.

It is a bi-directional one to many mapping from customer to order.

The inverse side in this mapping is the owner of object. In this case customer is the owner or order. Since an order cannot exist without a customer. But a customer can exist without an order.

Also customer has no column to save order data. But an Order table can store customer id, which is used for mapping.

260. What is ORM metadata?

ORM uses metadata for its internal work. ORM maintains metadata to generate code used for accessing columns and tables.

ORM maps classes to tables and stores this information in Metadata. It maps fields in classes to columns in tables. These kinds of mappings are also part of Metadata.

Application developers can also access Hibernate Metadata by using ClassMetadata and CollectionMetadata interfaces and Type hierarchy.

261. What is the difference between load() and get() method in Hibernate?

In Hibernate, load() and get() methods are quite similar in functionality.
The main difference is that load() method will throw an ObjectNotFoundException if row found corresponding to an object is not in the database.
On the other hand, get() method returns null value when an object is not found in the database.
It is recommended that we should use load() method only when we are sure that object exists in database.

262. When should we use get() method or load() method in Hibernate?

As a thumb rule we can follow these guidelines:

We should use get() method when we want to load an object.
We should use load() method when we need a reference to an object without running extra SQL queries.

263. What is a derived property in Hibernate?

In Hibernate, a derived property is not mapped to any column of a database table.

A derived property is computed at runtime by evaluation of an expression.

These are read only properties.

Egg. In this example profitMargin is derived from salePrice and buyPrice.

<property name="profitMargin" formula="(SELECT (i.salePrice – i.buyPrice) FROM item i WHERE i.Id = Id)"/>

264. How can we use Named Query in Hibernate?

A Named SQL query is the HQL query that is associated with a string name and can be referenced in the application by name.

It can be used in following ways:

XML Mapping File: We can define it in XML mapping file.

Egg. <query name="findBookByAuthor”> <![CDATA[from Book s
where s.author = :author]]>
</query>

Annotation: We can also mark Named SQL with annotation.

```
@NamedQueries({
 @NamedQuery(
 name = "findBookByAuthor”,
 query = "from Book s where s.author = :author”
 )
})
```

265. What are the two locking strategies in Hibernate?

There are two popular locking strategies that can be used in Hibernate:

Optimistic: In Optimistic locking we assume that multiple transactions can complete without affecting each other. So we let the transactions do their work without locking the resources initially.

Just before the commit, we check if any of the resource has changed by another transaction, then we throw exception and rollback the transaction.

Pessimistic: In Pessimistic locking we assume that concurrent transactions will conflict while working with same resources. So a transaction has to first obtain lock on the resources it wants to update.

The other transaction can proceed with same resource only after the lock has been released by previous transaction.

266. What is the use of version number in Hibernate?

Version number is used in optimistic locking in Hibernate. When a transaction modifies an object, it increments its version. Based on version number, second transaction can determine if the object it has read earlier has changed or not.

If the version number at the time of write is different than the version number at the time of read, then we should not commit the transaction.

267. What is the use of session.lock() method in Hibernate?

Session.lock() is a deprecated method in Hibernate. We should not use it.

Instead we should call buildLockRequest(LockMode).lock(entityName, object) method in Hibernate.

268. What inheritance mapping strategies are supported by Hibernate?

Hibernate supports following inheritance mapping strategies between classes and tables:

Table per class hierarchy: In case of multiple types of books, we can have one book class and one book table. We can store all child classes of book like- HardCoverBook, PaperBackBook etc in same table book. But we can identify the subclasses by a BookType column in Book table.

Table per subclass: In this case we can have separate table for each kind of book. HardCoverBook table for HardCoverBook book class. PaperBackBook table for PaperBackBook book class. And there will be a parent table, Book for Book class.

Table per concrete class: In this case also we have separate table for each kind of book. But in this case, we have even inherited properties defined inside each table. There is no parent table Book for Book class, since it is not a concrete class.